제 20 권 제 1 호
Vol. 20 No. 1

2018

제 20 회 한국 소프트웨어공학 학술대회

논문집

Proceedings of the 20th Korea Conference on

Software Engineering (KCSE 2018)

- **일시: 2018년 1월 29일(월) ~ 1월 31일(수)**
- **장소: 강원도 평창 한화리조트(휘닉스파크점)**

주최: 한국정보과학회, 한국정보처리학회
주관: 한국정보과학회 소프트웨어공학 소사이어티
한국정보처리학회 소프트웨어공학연구회
후원: SOLUTIONLINK, redhat., ㈜비트컴퓨터,
T3Q㈜, ㈜다한테크, 슈어소프트테크㈜,
STA 테스팅컨설팅㈜, TTA 소프트웨어시험인증연구소,
지능형 블록체인 연구센터, 신뢰적 지능형 CPS 연구단,
SW 상시모니터링기술연구단,
소프트웨어 안전성 보증 연구센터

초대의 글

우리 나라 소프트웨어 공학인의 축제인 제 20 회 한국 소프트웨어 공학 학술대회 KCSE 2018 참가자 여러분을 환영합니다.

기업, 연구소 및 학계에서 활동하고 계신 소프트웨어공학 분야 전문가들의 모임인 한국정보과학회 소프트웨어공학 소사이어티와 한국정보처리학회 소프트웨어공학연구회는, 여러 산업분야에서 소프트웨어 공학 기술의 역할이 점점 더 커지고 국가산업발전에 있어서의 소프트웨어공학의 중요성에 대한 인식이 날로 높아짐에 따라, 소프트웨어 공학 기술의 발전 및 적용확산을 위하여 산, 학, 연과의 협력으로 한국 소프트웨어 공학 학술대회를 개최하게 되었습니다.

이번 학술대회에서는 특히 "지능정보사회를 위한 소프트웨어공학 기술"을 주제로 하여 소프트웨어공학 전분야에 걸쳐 각계에서 제출한 71 편의 논문이 발표됩니다. 또한 "AI 서비스 개발과 소프트웨어공학," "인공지능 기법을 이용한 결함예측 기술과 현황," "오픈 소스 블록체인 플랫폼 아키텍처 분석과 평가"를 포함하여 최근 기술 동향과 소프트웨어공학의 핵심기술을 학습할 수 있는 여섯 개의 튜토리얼이 준비되어 있습니다. 아울러 소프트웨어 공학의 중요한 이슈와 미래방향에 대하여 세 분의 저명인사께서 기조발표를 해주십니다.

본 학술대회가 소프트웨어 공학의 학문적 발전과 소프트웨어 산업기술 발전의 장이 되고, 활발한 학술 및 기술 교류를 위한 격의 없는 토론장이 될 수 있도록 여러분의 적극적인 참여를 부탁드립니다.

이번 행사를 위해 수고해 주신 조직위원회와 학술위원회 위원들과 후원해 주신 여러 기관께 깊이 감사 드립니다.

한국정보과학회 소프트웨어공학 소사이어티 회장 강성원

한국정보처리학회 소프트웨어공학연구회 운영위원장 박용범

학술대회 준비 위원회

공동대회장: 강성원 교수(KAIST), 박용범 교수(단국대)

조직위원장: 이병정 교수(서울시립대)

조직위원: 고인영 교수(KAIST), 백종문 교수(KAIST), 유준범 교수(건국대),
이관우 교수(한성대), 이선아 교수(경상대), 이정원 교수(아주대),
이찬근 교수(중앙대)

학술위원장: 최윤자 교수(경북대)

학술위원: 권영우 교수(경북대), 김문주 교수(KAIST), 김순태 교수(전북대),
김정아 교수(관동대), 남재창 교수(포항공대), 박수진 교수(서강대),
배경민 교수(포항공대), 서주영 교수(아주대), 염근혁 교수(부산대),
유신 교수(KAIST), 유철중 교수(전북대), 윤일철 교수(U. of Maryland, College Park),
이우진 교수(경북대), 이지현 교수(전북대), 정우성 교수(서울교대),
채흥석 교수(부산대), 홍신 교수(한동대), 홍장의 교수(충북대)

문의사항 연락처

학술대회 홈페이지 : http://www.sigsoft.or.kr/KCSE2018/
조 직 : 이병정 교수 (bjlee@uos.ac.kr 02-6490-6600)
학 술 : 최윤자 교수 (yuchoi76@knu.ac.kr 053-950-7549)

KCSE 2018 프로그램

<table>
<tr><th colspan="5">1 월 29 일 (월)</th></tr>
<tr><th>시간</th><th colspan="4">행 사 내 용</th></tr>
<tr><td>12:00–13:00</td><td colspan="4">KCSE 2018 등록</td></tr>
<tr><td></td><td>튜토리얼 T1:
좌장: 홍신(한동대) 장소: 세미나실 1</td><td>튜토리얼 T2:
좌장: 이찬근(중앙대) 장소: 세미나실 2</td><td>튜토리얼 T3:
좌장: 이우진(경북대) 장소: 세미나실 6</td><td rowspan="4">워크숍:
지능형
블록체인
연구센터

장소:
그랜드홀 2
(13:00-
16:00)</td></tr>
<tr><td>13:00-14:30
(90 분)</td><td>인공지능 기법을 이용한 결함예측 기술과 현황
남재창 연구교수 (포항공대)</td><td>오픈소스 블록체인 플랫폼 아키텍처 분석과 평가
금창섭 대표 (빅픽처랩)</td><td>정형검증 사례공유
박사천 수석연구원 (삼성전자 반도체연구소 SW 센터)</td></tr>
<tr><td>14:30-14:40</td><td colspan="3">휴식</td></tr>
<tr><td></td><td>튜토리얼 T4:
좌장: 이선아(경상대) 장소: 세미나실 1</td><td>튜토리얼 T5:
좌장: 이정원(아주대) 장소: 세미나실 2</td><td>튜토리얼 T6:
좌장: 이관우(한성대) 장소: 세미나실 6</td></tr>
<tr><td>14:40-16:10
(90 분)</td><td>대규모 SW 를 효과적이고 정교하게 테스트 하기 위한 SW 유닛 테스트 자동 생성
김윤호 박사 (KAIST)</td><td>AI 서비스 개발과 소프트웨어공학
윤경아 박사 (SK 텔레콤)</td><td>Personal Care Robot 의 Hazard 분석과 안전 요구사항 도출 프로세스: ISO 13482 를 중심으로
한혁수 교수 (상명대)</td></tr>
<tr><td>16:10-16:20</td><td colspan="4">휴식</td></tr>
<tr><td></td><td colspan="4">개회식 사회: 이병정 조직위원장
장소: 그랜드홀 2</td></tr>
<tr><td>16:20-16:40</td><td colspan="4">개회사: 강성원 회장 (한국정보과학회 소프트웨어공학소사이어티),
박용범 운영위원장 (한국정보처리학회 소프트웨어공학연구회)
축사: 홍봉희 교수 (정보과학회 회장)</td></tr>
<tr><td></td><td colspan="4">기조연설 I 사회: 이병정 조직위원장
장소: 그랜드홀 2</td></tr>
<tr><td>16:40-17:30</td><td colspan="4">전진옥 대표 (비트컴퓨터)</td></tr>
<tr><td></td><td colspan="2">신임교수 초청 세미나 1
좌장: 유준범(건국대) 장소: 세미나실 1</td><td>신임교수 초청 세미나 2
좌장: 백종문(KAIST) 장소: 세미나실 2</td><td>신임교수 초청 세미나 3
좌장: 고인영(KAIST) 장소: 세미나실 6</td></tr>
<tr><td>17:30-18:20</td><td colspan="2">배경민 교수 (포항공대)</td><td>홍신 교수 (한동대)</td><td>권영우 교수 (경북대)</td></tr>
<tr><td>18:20-19:10</td><td colspan="4">석식</td></tr>
</table>

1 월 30 일 (화)

시 간	행 사 내 용				
	논문 발표 A				
	A1: SW 품질 1 좌장: 남재창(포항공대) 장소: 그랜드홀 2	**A2: SW 테스팅 1** 좌장: 이병정(서울시립대) 장소: 세미나실 1	**A3: SW 프로세스** 좌장: 이관우(한성대) 장소: 세미나실 2	**A4: 보안 및 안전** 좌장: 배경민(포항공대) 장소: 세미나실 6	
09:00-10:40 (100 분)	개발자의 Bad Fix Rate 를 고려한 효과적인 체인지 레벨 결함 예측 기법 연구 [우수단편논문] 임형선, 백종문(KAIST) 요구사항 추적성 매트릭스에서 유연한 맵핑 변경을 가능하게 하는 스프레드시트 애플리케이션 [우수논문] 정세린, 이선아(경상대) 데이터 기반 소프트웨어 신뢰성 모델의 입력 도메인 선정 기술의 필요성 연구 [단편논문] 강상모, 백종문(KAIST) 무기체계 SW PCA 활동 및 수행 사례 [산업체논문] 최민관, 임종일, 국승학, 노성규, 이태호(국방과학연구소) Some Issues Survey about SQLite Performance [학부논문] 박성현(한양대)	C 언어 테스트 입력 생성에 대한 CAVM, Austin, CodeScroll 의 비교 및 분석 [일반논문] 유병현, 김준휘(KAIST), 권민혁(슈어소프트테크), Phil McMinn(University of Sheffield), 유신(KAIST) 유닛 테스트 케이스 반복 실행을 통한 자바 프로그램에서의 메모리 블롯 오류의 효과적 검출: Apache Commons VFS 대상 적용 사례 [단편논문] 이재훈, 홍신(한동대) TPC-DI 기반의 EAI 솔루션 주요기능 테스트 사례 [산업체논문] 강건희, 권하은, 한종혁, 김상기(한국정보통신기술협회) ISO 26262 Test Method 의 적용 방안과 효과 [산업체논문] 박찬혁, 전수남, 김태열(정보통신산업진흥원), Stuart Reid, 김선준(STA 테스팅컨설팅)	오픈소스 기반 소프트웨어 프로세스 구축 자동화 개선 방안 [단편논문] 이진협, 장우성, 박지훈(홍익대), 박용범(단국대), 김영철(홍익대) 안전 필수 시스템의 개념단계 해저드 분석 프로세스 [단편논문] 김은비, 전은진(상명대), 도성룡(소프트웨어 안전성 보증 연구센터), 한혁수(상명대) 지속적 통합의 빌드 및 테스트 성능 향상을 위한 동적 스케줄링 기법 [단편논문] 이장수, 권영우(경북대) SW 중소기업을 위한 경량 개발 방법론 개발 및 실무 적용에 대한 연구 [산업체논문] 송원민(정보통신산업진흥원) 유희진(STA 테스팅컨설팅)	안전한 소프트웨어 집약적 시스템을 위한 시스템 이론적 안전성 분석에서의 위험도 평가 [일반논문] 정대희, 권기현(경기대) Functional Safety(ISO 26262)의 Safety 향상을 위한 Security 를 고려한 통합적 분석방법에 대한 연구 [최우수논문] 정지현(만도), 민상윤(KAIST) An Observation of Use Case and Misuse Case in System Theoretic Process Analysis [일반논문] Anit Thapaliya, 권기현(경기대) Keystroke Dynamics 를 기반으로 한 OTP 게임계정 보안 시스템에 대한 이론적 강화방법 [학부논문] 안정수, 김상민, 신정우, 정재웅(경희고) A Statistical Approach for Mobile User Authentication based on Context Data [우수학부논문] 남상진, 강승현, 김순태(전북대), 고덕윤, 유용환(Finotech)	**워크숍: 신뢰적 지능형 CPS 연구단** 장소: 세미나실 4 (09:00-12:30)
10:40-10-50	**휴식**				

1 월 30 일 (화)

시 간	행 사 내 용				
	논문 발표 B				
	B1: 분석 및 평가 1 좌장: 이정원(아주대) 장소: 그랜드홀 2	**B2: 설계 & 모델링 1** 좌장: 이찬근(중앙대) 장소: 세미나실 1	**B3: 스마트 시스템** 좌장: 권영우(경북대) 장소: 세미나실 2	**B4: 검증** 좌장: 배경민(포항공대) 장소: 세미나실 6	
10:50-12:30 (100 분)	시스템 오브 시스템즈 메타모델의 시나리오 기반 분석 및 통계적 검증 활용 가능성 분석 [일반논문] 신용준, 박수민, 백영민, 배두환(KAIST) RGB 색 공간 분할 기법을 통한 AMOLED 디스플레이의 소모 전력 모델 생성 및 평가 [우수논문] 백두산, 최유림(아주대), 이병정(서울시립대), 이정원(아주대) 비전공자들을 위한 영어 텍스트 데이터 분석 프로그래밍 교육 방법론 (특정 사례를 중심으로) [단편논문] 오하영(아주대) 척도를 이용한 소프트웨어의 진화 단계 파악 [학부논문] 조희태, 이선아(경상대)	피처 간 상호작용 패턴 정의 [단편논문] 이영우, 이선열, 채흥석(부산대) A Systematic Review on Variability Modeling in Software Product Line [우수단편논문] Aman Jaffari, Jihyun Lee, Jun Hyuk Jo, Cheol Jung Yoo(전북대) DEVS 기반 에이전트 시뮬레이션 모델 생성기 [단편논문] 정준영, 배장원, 이천희, 백의현(한국전자통신연구원) AOP 를 적용한 Fault Manager 컴포넌트 설계 [산업체논문] 오성수, 김지성(효성중공업), 김수동(숭실대) ThingML 의 IoT 시스템 설계 적용 사례 연구 [학부논문] 김가영, 안지흔, 최은만(동국대)	Design and Implementation of Web Service End-To-End Performance Monitoring System Using Open Source Products [일반논문] 전향희, 나성엽(엔키아) Development farming platform between city and farm based on Internet of Things technology [단편논문] Gye Cheon Lee(강남대), Sangil Kim(아람인), Tae Jun Lee(홍콩과기대) 데이터 기반의 강화학습을 통한 선박 자율 운항에 관한 연구 [단편논문] 임지수, 주재걸(고려대) 독해력 증진 소프트웨어의 디자인을 위한 매체별 읽기 양상 차이 연구 [단편논문] 기경서, 정유진, 권가진(서울대) A Block-chain based System for Document Integrity Support by using Document Unit. [학부논문] 장지현, 노지우, 홍준기, 송성한, 김순태(전북대)	메모리 상태 모니터링 데몬 소프트웨어 개발 [단편논문] 손명규(서울대) 안남영, 이규선, 심응보(SK 하이닉스) 유성주(서울대) OSEK/VDX 기반 소프트웨어의 API 호출 검증 사례연구: Erika 어플리케이션 검증 [단편논문] 김동우, 최윤자(경북대) OSEK/VDX 기반 내장형 소프트웨어 검증을 위한 환경 설정 추출 및 소스코드 자동 변환 도구 [우수단편논문] 정유희, 최윤자(경북대) 무기체계 소프트웨어 IV&V 현황 및 사례 연구 [산업체논문] 김태현, 임종일, 최민관, 박다운, 백옥현, 박삼준(국방과학연구소) 인공지능 SW 오작동 유형 분류·분석 및 개발 시 고려사항 연구 [우수산업체논문] 김상기, 정종하, 강지성, 강건희, 권하은, 김정현, 김주리, 이상민, 공준익, 정성룡 (소프트웨어시험인증연구소)	**워크숍: 신뢰적 지능형 CPS 연구단** 장소: 세미나실4 (09:00-12:30)
12:30-13:30	**중식**				

<table>
<tr><td></td><td colspan="4">논문 발표 C</td></tr>
<tr><td></td><td>C1: 도구 및 개발환경
좌장: 권영우(경북대)
장소: 그랜드홀 2</td><td>C2: SW 테스팅 2
좌장: 이우진(경북대)
장소: 세미나실 1</td><td>C3: 패턴추출 & 빅데이터
좌장: 남재창 (포항공대)
장소: 세미나실 2</td><td>SW 중심대학 발표
좌장: 이병정(서울시립대)
장소: 세미나실 6</td></tr>
<tr><td>13:30-15:30
(120 분)</td><td>Concolic 테스팅 도구 CREST 의 사용자 친화성 향상 연구: Windows OS 로의 포팅과 개선된 CREST UI 을 통한 CREST 활용 및 분기 커버리지 분석 작업의 효율 증가 [우수논문]
김현우, 김윤호, 김문주(KAIST)

융복합 연구 증진과 산학협력 활성화를 위한 연구자 네트워크 시스템 개발 [우수논문]
황상원, 서강원, 류원철, 남영광(연세대)

SVM: 소프트웨어 제품 라인 개발을 위한 가변성 모델링 도구 [우수단편논문]
한영훈, 강성원, 박연하(KAIST), 박병훈(티쓰리큐)

V-Diagram 기반 SW Reliability Engineer 의 업무분석과 효과적인 도구 사용방안 [산업체논문]
서달미, 손근태, 서형오, 오정섭 (엔에스이 융합솔루션개발그룹)

소프트웨어 제품 라인 가변부 구성 자동 관리 시스템 개발 [산업체논문]
김태호, 강성원(KAIST), 이지현(전북대)</td><td>차량용 소프트웨어의 테스팅을 위한 자동화된 HiL 테스트 모의 환경 구축 [일반논문]
황규본, 최기용(아주대), 이병정(서울시립대), 이정원(아주대),

비행 테스트 프로파일을 이용한 드론 소프트웨어 테스트 자동화 기법 [일반논문]
김덕엽, 이성희, 이우진(경북대)

모바일 클라이언트와 서버간 통합테스트 애스팩트 구성 방법 [단편논문]
최윤석, 최은만(동국대)

코드 변경에 따른 테스트 케이스 갱신 분석 [단편논문]
강혜진, 이선아(경상대)

프로세스 간 공유 메모리 데이터 기록 및 재현 [단편논문]
윤기현, 유민수(한양대)</td><td>하이브리드 문서 표현을 이용한 효율적인 문서 분류 [우수논문]
최성옥, 백종문(KAIST)

다변량 대량 데이터 내의 소량의 이상 데이터 원인 규명 및 학습/추론 방법 [단편논문]
이제희, 이상형, 김동일(한국생산기술연구원)

최초의 민간 소프트웨어 품질 인증, ‘SW 인증’ 파헤치기! [후원업체발표]
김관호(STA 테스팅컨설팅)

비전문가들에 의한 소프트웨어 설계 [학부논문]
고대권, 김승섭(아주대), 김상훈, 성주혁(국민대), André van der Hoek, Ayushi Rastogi(캘리포니아대), 이석원(아주대)

SW 신뢰성 관리를 위한 테스팅 도구로부터 데이터 추출 방법 [산업체논문]
손현승, 박상건, 정인용, 류인수, 장주수 (모아소프트 신뢰성기술연구소) 김태현, 박삼준(국방과학연구소)</td><td>비전공자들을 위한 텍스트 데이트 분석 프로그래밍 교육 방법론 (특정 사례를 중심으로)
오하영 교수 (아주대)

민관협력 SW 중심대학사업 수행사례-부산대학교 SW 가치확산사업을 중심으로
정주훈 교수 (부산대)

한양대학교 SW 중심대학 사업수행 사례발표
유민수 교수 (한양대)</td></tr>
<tr><td>15:30-15:45</td><td colspan="4">휴식</td></tr>
<tr><td></td><td colspan="4">초청강연 사회: 최윤자 학술위원장 (경북대)
장소: 그랜드홀 2</td></tr>
<tr><td>15:45~16:45</td><td colspan="4">김성훈 교수(홍콩과기대/네이버)</td></tr>
<tr><td>16:45~17:00</td><td colspan="4">휴식</td></tr>
<tr><td></td><td colspan="4">기조연설 II 사회: 강성원 대회장 (KAIST)
장소: 그랜드홀 2</td></tr>
<tr><td>17:00-18:00</td><td colspan="4">김명준 소장 (소프트웨어 정책연구소)</td></tr>
<tr><td>18:00-21:00</td><td colspan="4">우수논문상, 공로상, 감사장 수여식 사회: 이병정 조직위원장 (서울시립대)
만찬 사회: 김순태 교수 (전북대)
장소: 그랜드홀 1</td></tr>
</table>

1 월 31 일 (수)			
시 간	**행 사 내 용**		
	논문 발표 D		
	D1: 지식기반 시스템 좌장: 김순태(전북대) 장소: 세미나실 1	**초청세미나** **(우수 국제학술대회 논문)** 좌장: 고인영(KAIST) 장소: 세미나실 2	**초청세미나** **(우수 국제학술대회 논문)** 좌장: 백종문(KAIST) 장소: 세미나실 6
9:00-10:30 (90 분)	시스템 오브 시스템즈(SoS) 사례 분석을 통한 온톨로지 기반의 SoS 메타모델 개발 [최우수논문] 백영민, 박수민, 신용준, 배두환(KAIST) 인공신경망을 이용한 코드 변경 추천 시스템 [단편논문] 이준범, 강성원, 안휘, 정필수(KAIST), 이선아(경상대) 지식베이스 기반 L-V-C 매핑 방법 연구 [단편논문] 이동재, 손윤희, 이규철(충남대) Ontology 를 이용한 지식 기반 지능적 강의 및 진로 추천 시스템 구축 [학부논문] 최준영, 이석원(아주대)	Testing Intermediate Representations for Binary Analysis (ASE'17) 김수민 (KAIST) All about Activity Injection: Threats, Semantics, and Detection (ASE'17) 이성호 (KAIST) Data-driven Context-Sensitivity for Points-to Analysis (OOPSLA'17) 전민석 (고려대)	Prioritizing Browser Environments for Web Application Test Execution 권정현 (KAIST) Applying Deep Learning Based Automatic Bug Triager to Industrial Projects (FSE'17) 김밀한 (LG 전자) FLUSSS : Using code and change metrics to improve fault localization (ISSTA'17) 손정주 (KAIST)
10:30-10:40	**휴식**		

1 월 31 일 (수)				
시 간	**행 사 내 용**			
	논문 발표 E			
	E1: 분석 및 평가 2 좌장: 홍신(한동대) 장소: 그랜드홀 2	**E2: SW 품질 2** 좌장: 고인영(KAIST) 장소: 세미나실 1	**E3: 유지보수** 좌장: 이선아(경상대) 장소: 세미나실 2	**E4: 설계 & 모델링 2** 좌장: 유준범(건국대) 장소: 세미나실 6
10:40-12:00 (80 분)	GitHub 에서의 이슈 및 풀 요청 관련 활동이 소스 코드 품질에 미치는 영향 분석 [단편논문] 조영준, 고인영(KAIST) 객체지향 재사용 메트릭스 [단편논문] 변은영, 박지훈, 김영철(홍익대) 역공학 기반 코드 가시화를 통한 기능점수의 비용 추정 검증 방안 [단편논문] 문소영, 김영철(홍익대) Code X-Ray: LG Electronics' code quality inspection platform [산업체논문] Milhan Kim, Jewhi Ryu, Heewon Choi (LG 전자)	가중치 기반 센서 데이터 획득 효율성 향상 기법 [단편논문] 송명호, 김수동(숭실대) MR3 엔진 ASM 단계 성능 비교 [우수단편논문] Hoang Le, Sungwoo Park(포항공대) 요구사항에 따른 이슈 분류 [단편논문] 심재경, 정세린, 이선아(경상대) 딥러닝 모델 기반의 버그 담당자 자동 배정 시스템 성능 측정 [학부논문] 박해성, 김수빈, 이찬근, 채병훈(중앙대)	유전 프로그래밍을 이용한 버그 자동 정정 기법에 유사 버그 정정 활용 [일반논문] 정영준, 민경식, 양근석(서울시립대), 이정원(아주대), 이병정(서울시립대) Toward Measuring Quality Factors for Software Product Operation [일반논문] Horm Daneth, 홍장의(충북대) 국방 소프트웨어 자원관리시스템의 소스 코드 품질을 위한 자바 파서 개발을 통한 적용 사례 [단편논문] 박보경, 문소영, 서채연, 김영철(홍익대) 김광남, 최영식, 신상훈(육군 정보체계관리단)	군집 비행을 위한 무인비행기 네비게이터 시스템 설계 [단편논문] 이성희, 김덕엽, 윤보람, 이우진(경북대) 동적 데이터 획득 시스템 시각화 프레임워크 [단편논문] 임성민, 김수동(숭실대) 클래스 변경 정보에 기반한 마이크로서비스 단위의 애플리케이션 구성 방법 [단편논문] 김대호, 김웅수, 윤동규(부산대), 박준석(물류혁신네트워킹연구소), 염근혁(부산대) 금속재료 피로시험의 최적설계 소프트웨어의 클린코드 적용 사례 [산업체논문] 정인용, 박상건, 손현승(모아소프트), 김광진(알에스피)
12:00-12:30 (30 분)	**폐회식** 장소: 그랜드홀 2			사회: 이병정 조직위원장(서울시립대)

* 위 일정은 사정에 따라 변경될 수가 있습니다.

KCSE 2018 튜토리얼

튜토리얼 T1: 인공지능과 결함예측

- 일시: 1월 29일(월) 13:00~14:30
- 장소: 세미나실 1
- 제목: 인공지능 기법을 이용한 결함예측 기술과 현황
- 연사: 남재창 박사(남재창, 포항공과대학교 컴퓨터공학사업단, 연구교수)
- 튜토리얼 초록:

  지능정보사회가 가속화 됨에 따라 다양한 소프트웨어들이 양산되고 있다. 하지만, 소프트웨어 결함 또한 증가되고 있어, 다양한 품질 관리 기법들이 요구된다. 본 튜토리얼의 주제는 다양한 품질관리 기법 중 하나인 인공지능 기반 소프트웨어 결함 예측 기술이다. 튜토리얼의 목적은 소프트웨어 공학 연구자 및 엔지니어들이 소프트웨어 결함 예측을 위해 알아야 할 연구의 동기 및 배경 지식, 연구와 실무에 필요한 다양한 지식 공유에 있다. 튜토리얼을 통해, 먼저 소프트웨어 결함 예측 기술의 다양한 종류와 연구 성과들을 살펴보고, 당면한 연구 현안들을 제시한다. 그리고, 후속 연구와 기술 개발을 위한 데이터와 도구, 소프트웨어 결함 예측 연구 설계를 위한 일반적인 가이드 라인을 소개한다.

- 약력:

  MSc, School of Computing, Blekinge Institute of Technology, Sweden.
  PhD, Dept. Computer Science and Engineering, 홍콩과기대,
  Postdoctoral Fellow: Dept. of Electrical and Computer Engineering, University of Waterloo, Canada
  현) 포항공과대학교 컴퓨터공학사업단 연구교수

튜토리얼 T2: 블록체인

- 일시: 1월 29일(월) 13:00~14:30
- 장소: 세미나실 2
- 제목: 오픈소스 블록체인 플랫폼 아키텍처 분석과 평가(Architecture Analysis and Evaluation of Open Source Blockchain Platform)
- 연사: 금창섭 대표(빅픽처랩)

- 튜토리얼 초록:

  최근 4 차 산업혁명의 핵심 플랫폼인 블록체인에 대한 관심은 최고조에 이르고 있으나 아키텍처에 대한 근본적인 이해 없는 단편적인 지식으로 인하여 지나친 낙관 혹은 비관이 난무하고 있다. 본 튜토리얼에서는 대표적인 오픈소스 블록체인 플랫폼인 이더리움과 하이퍼레저 아키텍처를 큰그림으로 복원하여 이해하고, 이를 토대로 ATAM 기법을 이용하여 분석하고 평가해본다. 아울러 향후 오픈소스 블록체인 플랫폼의 아키텍처 진화 방향을 모색해 본다.

- 약력:

2018 년~현재	빅픽처랩 대표이사, 체인투비 CTO
1994~2017 년	한국전자통신연구원 신뢰서비스플랫폼연구실장
2013 년	KAIST, 전산학 박사
2005 년	카네기멜론대학교, 소프트웨어공학 석사

  연구분야: 블록체인, 소프트웨어 아키텍처, 서비스플랫폼

튜토리얼 T3: 정형검증 사례 공유

- 일시: 1월 29일(월) 13:00~14:30
- 장소: 세미나실 6
- 제목: 정형검증 사례 공유
- 연사: 박사천 수석연구원 (삼성전자 반도체연구소 SW센터)

- 튜토리얼 초록:

  정형검증은 시스템의 정확성을 입증하는 매우 엄격한 검증 방법이다. 이 방법은 주로 safety-critical system 검증에 사용된다. 따라서 현업에 적용할 기회도 적을 뿐더러 효율적인 적용도 매우 어렵다. 본 튜토리얼은 최근 몇년간 현업에서 적용된 실사례를 공유하여 정형기법의 저변 확대에 기여하고자 한다.

- 약력:

  현재 삼성전자 반도체연구소 수석연구원

  ~ 2014/03 한국전자통신연구원 선임연구원

  ~ 2009/08 경기대학교 일반대학원 박사졸, 소프트웨어 공학 전공

튜토리얼 T4: SW 유닛 테스트

- 일시: 1월 29일(월) 14:40~16:10
- 장소: 세미나실 1
- 제목: 대규모 SW를 효과적이고 정교하게 테스트 하기 위한 SW 유닛 테스트 자동 생성
- 연사: 김윤호 박사 (KAIST)

- 튜토리얼 초록:

  본 튜토리얼에서는 정교한 SW 유닛 테스트 자동 생성 기법을 소개한다. 먼저 기반이 되는 SW 자동 테스트 생성 기법인 Concolic 테스팅 기법과 산업체 적용 사례 연구를 소개하고 실제 현장 적용에서 대규모 SW 에 적용할 때 발생한 확장성(scalability) 이슈를 공유한다. 확장성 이슈를 해결하고 대규모 SW 에 적용하기 위한 SW 유닛 테스트 자동 생성 기법을 설명하고 SW 유닛 테스트 자동 생성 기법에서 발생하는 거짓 경보 문제를 해결하기 위한 거짓 경보 제거 기법을 설명한다.

- 약력:

  ~ 현재: KAIST 정보전자연구소 연수연구원

  2017 년: KAIST 전산학부 박사

  2007 년: KAIST 전산학과 학사

튜토리얼 T5: AI 서비스 개발과 소프트웨어 공학

- 일시: 1월 29일(월) 14:40~16:10
- 장소: 세미나실 2
- 제목: AI 서비스 개발과 소프트웨어
- 연사: 윤경아 팀장 (SK텔레콤 AI분석솔루션개발팀)

- 튜토리얼 초록:

AI 산업의 도래와 함께 AI 기술을 탑재한 다양한 형태의 서비스 및 제품들이 시장에 쏟아져 나오고 있다. 현재의 상용 기술 수준을 고려할 때, AI 서비스는 기존에 우리가 개발해 오던 서비스(소프트웨어)와 크게 다른 것일까?

이 튜토리얼에서는 AI 서비스 개발을 위해 필요한 기술적 요소와 소프트웨어 공학 관점에서의 고려사항을 NUGU 서비스 개발 사례와 함께 설명하고자 한다.

- 약력:
  - 2010.3 ~ 현재 SK 텔레콤 (Software Engineering, AI Platform 개발(NUGU), AI Data Analysis)
  - 2003.3 ~ 2010.2 KAIST 전산학과 박사 (Software Engineering)
  - 2001.3 ~ 2003.2 KAIST 전산학과 석사 (Software Engineering)
  - 1996.3 ~ 2000.4 삼성 SDS

튜토리얼 T6: 소프트웨어 안전

- 일시: 1월 29일(월) 14:40~16:10
- 장소: 세미나실 6
- 제목: Personal Care Robot의 Hazard 분석과 안전 요구사항 도출 프로세스: ISO 13482를 중심으로
- 연사: 한혁수 교수 (상명대 컴퓨터과학과)

- 튜토리얼 초록:

본 강의의 목표는 안전성(Safety)에 대한 이해도를 높이고, Safety Critical 한 시스템 개발 시, 반드시 수행해야 할 Hazard 분석 활동과 안전 요구사항 도출 프로세스를 학습하는 것이다.

강의 전반부에서는 시스템의 안전성 개념과 이를 확보하기 위한 노력들을 살펴본다. 사고를 예방하기 위한 Hazard 분석의 필요성을 인식하고, 전통적인 분석 기법인 PHA, FMEA, FTA, HAZOP 등에 대해서 학습한다.

또한 시스템 이론에 기반한 STPA(System Theoretic Process Analysis)의 특징에 대해서도 알아본다.

강의 후반부에서는 ISO 13482 를 중심으로 Personal Care Robot 의 안전 요구사항을 파악한다.

Personal Care Robot 의 적용 범위에 대해 학습하고, 표준이 제공하고 있는 Hazard List 와 대응 방안을 살펴본다. 또한 파악된 Hazard 를 제거 및 완화하기 위한 절차들을 이해한다.

.

- 약력:
  - 상명대학교 컴퓨터과학과 교수
  - 소프트웨어 안전성 보증 연구센터(Software Safety Assurance Research Center) 센터장
  - 소프트웨어공학소사이어티 회장, 2012. 3 ~ 2014. 2

KCSE 2018 기조연설

기조연설 I

- 일시: 1월 29일(월) 16:40~17:30
- 장소: 그랜드홀 2
- 제목: 4차산업혁명, 그리고 우리 소프트웨어 인력
- 연사: 전진옥 대표(비트컴퓨터)

기조연설 II

- 일시: 1월 30일(화) 17:00~18:00
- 장소: 그랜드홀 2
- 제목: 제4차 산업혁명과 소프트웨어의 중요성
- 연사: 김명준 소장(소프트웨어정책연구소)

KCSE 2018 신임교수 세미나

신임교수 세미나 1: 논리 기반 소프트웨어 모델 검증

- 일시: 1월 29일(월) 17:30~18:20
- 장소: 세미나실 1
- 제목: 논리기반 소프트웨어 모델 검증
- 연사: 배경민 교수 (포항공대)

- 세미나 초록:

모델 검증은 소프트웨어 모델의 요구사항을 다양한 알고리즘을 이용하여 자동으로 검증하는 기술이다. 이 중 논리 기반 모델 검증 기법은 소프트웨어의 의미와 요구 조건을 논리식으로 표현하고 이에 관한 여러 자동 추론 알고리즘을 조합하여 모델 검증을 수행하는 방법이다. 논리 기반 모델 검증 기법을 이용하면 다양한 종류의 소프트웨어 모델에 대하여 모델 검증을 수행할 수 있다. 예를 들어, C 나 Java 와 같은 프로그래밍 언어, 미분방정식으로 표현되는 물리 모델, 산술적 제약조건으로 표현되는 인공신경망과 같은 다양한 종류의 모델에 대하여, 해당 자동 추론 알고리즘을 이용한 논리 기반 모델 검증의 적용이 가능하다. 본 세미나에서는 논리 기반 소프트웨어 모델 검증 기술의 개요와 현황, 미해결 문제 및 도전과제에 대하여 발표한다.

- 약력:

2016-현재: 포스텍 컴퓨터공학과 조교수

2015-2016: Postdoctoral fellow, Computer Science Laboratory, SRI International, USA

2014-2015: Postdoctoral fellow, School of Computer Science, Carnegie Mellon University, USA.

2007~2014: Ph.D. in Computer Science, University of Illinois at Urbana-Champaign, USA.

신임교수 세미나 2: Testing with Mutation Analyses

- 일시: 1월 29일(월) 17:30~18:20
- 장소: 세미나실 2
- 제목: Learning from Mutants: Effective Software Debugging and Testing with Mutation Analyses
- 연사: 홍신 교수(한동대학교)

- 세미나 초록:

Despite significant advances in software engineering during the last half century, debugging and testing remain as one of the most difficult, labor-intensive, tedious activity to date. Human developers in debugging and testing are asked to acquire deep understanding of the target program to reason about complicated cause-effect relations among an enormous number of code entities in real-world software. Researchers had developed automated techniques based on symbolic analyses and statistical reasoning with dynamic information on the target programs, however, the effectiveness of automated debugging and automated test case generation are limited. In this talk, I will present a new automated debugging approach called mutation-based fault localization and a new test case generation approach based on mutation analyses called invasive software testing. The key idea of these techniques is to acquire useful dynamic information by systematically applying mutations to the target programs. The case studies with real-world open-source projects will show how mutation-based fault localization successfully assists developers debugging complicated real-world faults with multilingual features. In addition, I will introduce an idea of utilizing mutation analyses to generate test inputs that effectively achieve high test coverage with preliminary experiment results.

- 약력:

Shin Hong is an assistant professor in the school of Computer Science and Electrical Engineering at Handong Global University. He received his PhD in computer science from KAIST in 2015, where he also earned his BS and MS. His PhD research provided systematic and practical testing methods for concurrent programs including multithreaded programs and event-driven programs. His research interest continues to develop automated software testing and techniques for complex software systems.

신임교수 세미나 3: Green Computing

- 일시: 1월 29일(월) 17:30~18:20
- 장소: 세미나실 6
- 제목: Improving the Energy Efficiency of Mobile Applications via Spur-of-the-Moment Distributed Execution
- 연사: 권영우 교수 (경북대학교)

- 세미나 초록:

지난 10 여년 간 모바일 컴퓨팅의 발전으로 인해 모바일 소프트웨어는 여러 소프트웨어 분야 중 가장 주목받는 분야가 되었다. 모바일 소프트웨어의 에너지 사용 문제는 가장 활발히 연구가 이루어진 분야임에도 불구하고 여전히 해결되어야 할 문제로 남아있다. 본 발표에서는 모바일소프트웨어의 에너지 사용량을 줄이기 위하여 지난 5 년간 수행했던 연구와 현재 진행 중인 연구를 소개한다. 먼저 급변하는 모바일 환경의 특징을 반영한 "오프로딩 기법을 이용한 에너지 효율적인 모바일 실행" 과 이를 확장한 "제한된 자원을 고려한 모바일 서비스 프레임워크"에 대해서 설명한 후, 현재 연구 중인 "다양한 모바일 환경을 고려한 지속적 통합"에 대해서 소개를 하며 본 발표를 마무리 한다.

- 약력:

2017. 08 - 현재: 경북대학교 컴퓨터학부 조교수

2014. 08 – 2017. 06: Utah State University, Dept. of Computer Science, Assistant Professor

2014. 05 – 2014. 12: Deutsche Telekom Research Lab, Consultant

2008. 08 – 2014. 07: Virginia Tech, Research Assistant

2007. 08 – 2008. 06: 네이버, 대리

2005. 02 – 2007. 08: 대우전자, 신호처리 연구소, 선임연구원

KCSE 2018 초청강연

- 일시: 1월 30일(화) 15:45~16:45
- 장소: 그랜드홀2
- 제목: 딥러닝이 여는 SOFTWARE 2.0 세상!
- 연사: 김성훈 교수 (VP/Naver Clova, Associate Prof/HKUST)

- 초록:

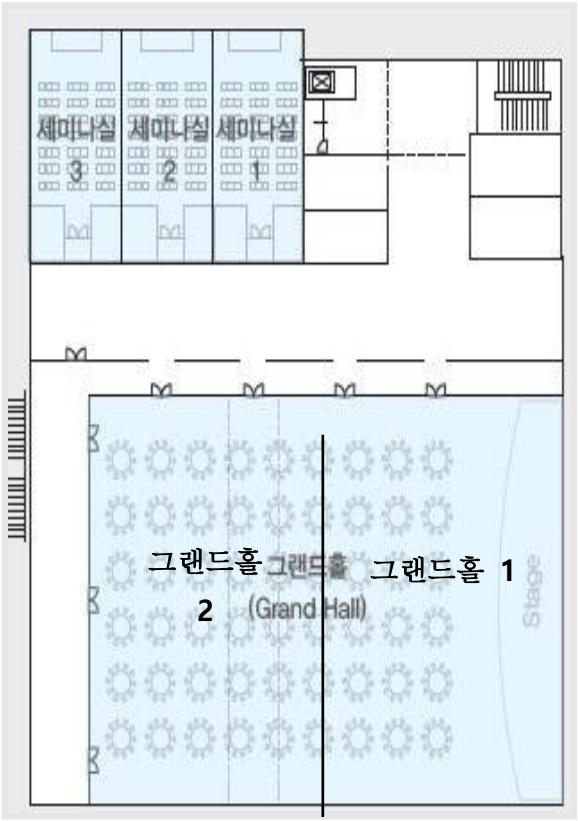
기존의 알고리즘 중심으로 사람에 의해 작성된 많은 소프트웨어들이, 이제는 SOFTWARE 2.0 이라 불리는 데이타 driven 으로 학습에 의해 만들어 지고 있습니다. 이러한 SOFTWARE 2.0 에 대해 소개해고, 소프트웨어 공학자 입장에서 고민해봐야할 새로운 challenge 에 대해서 논의합니다.

- 약력:

Sunghun Kim is a VP at Naver and an Associate Professor of Computer Science at the Hong Kong University of Science and Technology. He got his BS in Electrical Engineering at Daegu University, Korea in 1996. He completed his Ph.D. in the Computer Science Department at the University of California, Santa Cruz in 2006. He was a postdoctoral associate at Massachusetts Institute of Technology and a member of the Program Analysis Group. He was a Chief Technical Officer (CTO), and led a 25-person team at the Nara Vision Co. Ltd, a leading Internet software company in Korea for six years.

His core research area is Software Engineering, focusing on software evolution, machine learning, and program analysis and empirical studies. He publishes his work on top venues such as TSE, ICSE, FSE, AAAI, SOSP and ISSTA. He is a four-time winner of the ACM SIGSOFT Distinguished Paper Award (ICSE 2007, ASE 2012, ICSE 2013 and ISSTA 2014). Besides, he received various awards including 2010 and 2011 Microsoft Software Innovation Awards and 2011 Google Faculty Research Award. He served on a variety of program committees including FSE 2011, FSE 2013, FSE 2015, ASE 2013, ICSE 2012, ICSE 2013 and ICSE 2016. He was a program co-chair of MSR 2013 and 2014. Further information is available at http://www.cse.ust.hk/~hunkim.

행사장 및 식당 위치



1 F

B 1 F

논문 목차

일반논문

Functional Safety(ISO 26262)의 Safety향상을 위한 Security를 고려한 통합적 분석방법에 대한 연구 1

시스템 오브 시스템즈(SoS) 사례 분석을 통한 온톨로지 기반의 SoS 메타모델 개발 3

Concolic 테스팅 도구 CREST 의 사용자 친화성 향상 연구:
Windows OS 로의 포팅과 개선된 CREST UI 을 통한 CREST 활용 및 분기 커버리지 분석 작업의 효율 증가 8

하이브리드 문서 표현을 이용한 효율적인 문서 분류 10

요구사항 추적성 매트릭스에서 유연한 맵핑 변경을 가능하게 하는 스프레드시트 애플리케이션 12

C 언어 테스트 입력 생성에 대한 CAVM, Austin, CodeScroll의 비교 및 분석 14

비행 테스트 프로파일을 이용한 드론 소프트웨어 테스트 자동화 기법 23

시스템 오브 시스템즈 메타모델의 시나리오 기반 분석 및 통계적 검증 활용 가능성 분석 31

RGB 색 공간 분할 기법을 통한 AMOLED 디스플레이의 소모 전력 모델 생성 및 평가 42

차량용 소프트웨어의 테스팅을 위한 자동화된 HiL 테스트 모의 환경 구축 44

융복합 연구 증진과 산학협력 활성화를 위한 연구자 네트워크 시스템 개발 53

안전한 소프트웨어 집약적 시스템을 위한 시스템 이론적 안전성 분석에서의 위험도 평가 55

유전 프로그래밍을 이용한 버그 자동 정정 기법에 유사 버그 정정 활용 63

오픈 소스를 활용한 웹 서비스 구간 별 성능 모니터링 시스템 설계 및 구현 71

소프트웨어 제품 운영에 대한 품질 측정 79

An Observation of Use Case and Misuse Case in System Theoretic Process Analysis 84

단편논문

MR3 엔진 ASM 단계 성능 비교 92

OSEK/VDX 기반 내장형 소프트웨어 검증을 위한 환경 설정 추출 및 소스코드 자동 변환 도구 96

개발자의 Bad Fix Rate를 고려한 효과적인 체인지 레벨 결함 예측 기법 연구 100

SVM: 소프트웨어 제품 라인 개발을 위한 가변성 모델링 도구 102

독해력 증진 소프트웨어의 디자인을 위한 매체별 읽기 양상 차이 연구 106

OSEK/VDX 기반 소프트웨어의 API 호출 검증 사례연구: Erika 어플리케이션 검증 110

A Systematic Review on Variability Modeling in Software Product Line 114

데이터 기반 소프트웨어 신뢰성 모델의 입력 도메인 선정 기술의 필요성 연구 118

클래스 변경 정보에 기반한 마이크로서비스 단위의 애플리케이션 구성 방법 122

오픈소스 기반 소프트웨어 프로세스 구축 자동화 개선 방안 126

GitHub에서의 이슈 및 풀 요청 관련 활동이 소스 코드 품질에 미치는 영향 분석 128

모바일 클라이언트와 서버간 통합테스트 애스팩트 구성 방법 132

유닛 테스트 케이스 반복 실행을 통한 자바 프로그램에서의메모리 블롯 오류의 효과적 검출: Apache Commons VFS 대상 적용 사례 136

가중치 기반 센서 데이터 획득 효율성 향상 기법 140

An Observation of Use Case and Misuse Case in System Theoretic Process Analysis 144

안전 필수 시스템의 개념단계 해저드 분석 프로세스 152

객체지향 재사용 매트릭스 156

DEVS 기반 에이젼트 시뮬레이션 모델 생성기 159

인공신경망을 이용한 코드 변경 추천 시스템 163

지속적 통합의 빌드 및 테스트 성능 향상을 위한 동적 스케줄링 기법 167

피처 간 상호작용 패턴 정의 171

지식베이스 기반 L-V-C 매핑 방법 연구 175

프로세스 간 공유 메모리 데이터 기록 및 재현 179

역공학 기반 코드 가시화를 통한 기능점수의 비용 추정 검증 방안 182

요구사항에 따른 이슈 분류 184

코드 변경에 따른 테스트 케이스 갱신 분석 186

데이터 기반의 강화학습을 통한 선박 자율 운항에 관한 연구 192

메모리 상태 모니터링 데몬 소프트웨어 개발 196

국방 소프트웨어 자원관리시스템의 소스 코드 품질을 위한 자바 파서 개발을 통한 적용 사례 200

비전공자들을 위한 영어 텍스트 데이터 분석 프로그래밍 교육 방법론 (특정 사례를 중심으로) 202

동적 데이터 획득 시스템 가시화 프레임워크 205

다변량 대량 데이터 내의 소량의 이상 데이터 원인 규명 및 학습/추론 방법 209

군집 비행을 위한 무인비행기 네비게이터 시스템 설계 213

산업체논문

인공지능 SW 오작동 유형 분류·분석 및 개발 시 고려사항 연구 216

V-Diagram 기반 SW Reliability Engineer의 업무 분석과 효과적인 도구 사용 방안 219

Code X-Ray: LG Electronics' code quality inspection platform 247

ISO 26262 Test Method의 적용 방안과 효과 251

SW 신뢰성 관리를 위한 테스팅 도구로부터 데이터 추출 방법 265

SW중소기업을 위한 경량 개발 방법론 개발 및 실무 적용에 대한 연구 269

TPC-DI 기반의 EAI 솔루션 주요기능 테스트 사례 284

무기체계 소프트웨어 IV&V 현황 및 사례 연구 298

무기체계 SW PCA 활동 및 수행 사례 312

소프트웨어 제품 라인 가변부 구성 자동 관리 시스템 개발 330

금속재료 피로시험의 최적설계 소프트웨어의 클린코드 적용 사례 334

AOP를 적용한 Fault Manager 컴포넌트 설계 339

학부생논문

맥락 데이터 기반 모바일 사용자 인증을 위한 통계적 접근 방안 342

딥러닝 모델 기반의 버그 담당자 자동 배정 시스템 성능 측정 346

척도를 이용한 소프트웨어의 진화 단계 파악 352

블록체인 기반 문서 구성요소 단위 무결성 지원 시스템 356

ThingML의 IoT 시스템 설계 적용 사례 연구 360

Ontology를 이용한 지식 기반 지능적 강의 및 진로 추천 시스템 구축 365

Keystroke Dynamics를 기반으로 한 OTP 게임계정 보안 시스템에 대한 이론적 강화방법 369

비전문가들에 의한 소프트웨어 설계 374

후원업체발표

최초의 민간 소프트웨어 품질 인증, 'SW인증' 파헤치기! 379

Functional Safety(ISO 26262)의 Safety향상을 위한 Security를 고려한 통합적 분석방법에 대한 연구

정지현*, 민상윤**
*㈜ 만도, KAIST 소프트웨어대학원
**KAIST 전산학과
jihyun0235@kaist.ac.kr, symin@kaist.ac.kr

A Study on integrated analysis method adding security in Functional Safety(ISO 26262)

*Mando Corporation, Graduate School of Software, KAIST, Seoul, Korea
**Graduate School of Software, KAIST, Seoul, Korea

요 약

Automotive에서 2011년에 배포된 국제표준인 ISO 26262 도입으로 제품의 safety를 향상시켰다. 하지만 최근 프리미엄 자동차가 1G byte이상의 코드로 동작하고 70개 이상의 ECU로 구성되어있고 자율주행, connected car, ADAS 등장으로 complexity 증가하고 있다. 그리고 자동차에 internet, mobile application들과 V2x이 적용되면서 connectivity 증가 하고 있다. 최근 자동차의 complexity와 connectivity증가로 인해 security에 대한 risk가 증가하고 있고 이로 인해 safety 문제를 발생시키고 있다. 하지만 Security threat가 safety에 직접적으로 영향을 미치지만 Safety process(ISO 26262)와 Security process 따로 존재하고 있다. 그리고 ISO 26262 2nd Version에서 Security와 Safety를 상호 영향성에 대해 고려를 요구하고 있다. 본 논문에서 Security와 Safety를 통합적으로 분석하기 위해 Functional Safety(ISO 26262)에 Security를 고려한 통합적 분석 방법을 제시하여 Functional safety(ISO 26262)의 Safety를 향상 시키는데 기여하고자 한다.

Automotive industry에서 국제표준 ISO 26262 (Road vehicles)도입으로 safety의 중요성 개선 및 프로세스 및 방법론 적용을 통해 자동차의 electrical, electronic 시스템들의 safety를 향상시켰다. 최근 자동차의 complexity와 connectivity증가로 인해 security에 대한 risk가 증가하고 있고 이로 인해 safety문제를 발생시키고 있다.

차량의 시스템들이 다양한 interface를 통해 연결되어 있다. 이런 시스템들 간의 다양한 인터페이스들을 통해 다양한 attack vector들이 발생 가능할 수가 있다[1].

하지만 Security문제가 safety에 직접적으로 영향을 미치지만 Safety process(ISO 26262[2])와 Security process(J3061[3])는 따로 존재하고 있다. 그래서 Safety를 향상하기 위해 Functional Safety(ISO 26262)에 Security를 고려한 통합적으로 분석 방법을 제시한다. 그림 1에서 보듯이 통합 분석을 위해 ①ADAS and Chassis를 위한 Asset기반 Threat model 제안, ②Threat model을 통한 해당 제품의 Threat 식별, ③Safety related threat를 고려한 통합적 Risk Assessment, ④Safety와 Security 통합 분석을 위해 FTA를 확장 4가지 방법을 제시한다.

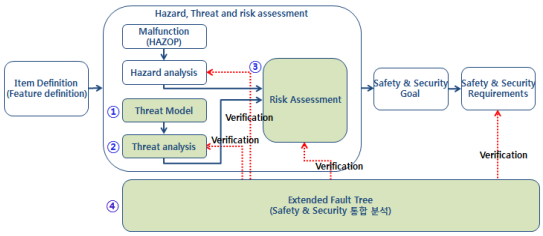


그림 1. Safety, Security 통합 Approach

ADAS와 Chassis를 위한 Asset기반 Threat model 제안하기 위해 security conference와 literature들의 분석을 하였다. 이를 통해 Asset별 Threat 잠재 케이스를 분석을 통해 Threat 모델을 제안한다.

그림 2는 ADAS와 Chassis를 위한 Asset 기반 threat model을 나타낸다.

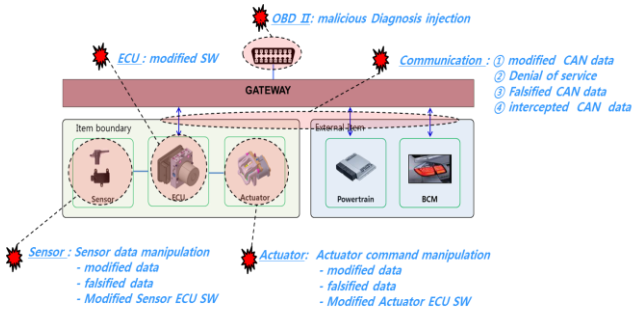


그림 2. ADAS와 Chassis를 위한 Asset기반 Threat model

그림 2에서 제안된 Threat model을 통해 ACC제품의 threat들을 식별할 수 있다. 식별된 Security threat중에 Safety hazard와 연관된 threat를 Risk assessment 기준 및 Combinded risk assessment calculator를 제안한다. Risk assessment 목적은 식별된 threat의 risk정도를 판단하고, Risk를 줄이는 수단이 필요 여부를 판단하는 것이다.

Risk는 probability of an accident 와 expected losses through that accident 인자들에 의해 값이 결정된다. Probability of an accident는 Attack potential으로도 정의가 가능하다. 그리고 probability of an accident는 damage potential으로도 정의가 가능하다.

Attack potential level을 계산하기 위해 산업에서 증명되고 표준화된 ISO 15408[4], ISO 18405[5]를 적용 하였다.

일반적으로 Damage potential은 safety, operation, privacy, finance 측면을 고려한다. 하지만 이런 4가지 측면 중에서 Chassis와 Advanced Driver Assistance System 제품은 ASIL B~D이므로 Safety가 가장 중요 인자이다. 본 논문에서는 safety 향상을 목표로 두고 있기 때문에 safety만 고려하였다.

ISO 26262에서 ASIL을 판단하기 위해 severity, exposure, controllability으로 판단한다. security threat로 문제 발생 시 위에서 언급한 3가지 인자 중에 damage와 관련한 인자는 부상의 정도를 나타내는 severity level과 운전자의 제어 가능 정도를 나타내는 controllability이다. 따라서 Risk는 아래 수식에서 보는 것처럼 Attack potential level, Severity level과 Controllability level의 값을 곱하여 Risk 값을 계산한다.

즉, Risk(Priority)

= (Attack Potential) x (damage potential)

= (Attack potential Level) x (Severity Level) x (Controllability Level)

식별된 Security threat중에 Safety hazard와 연관된 threat는 제안한 Combinded risk assessment calculator를 이용하여 safety와 security를 함께 고려한 Risk assessment를 수행하였다.

그림 3는 Combinded risk assessment calculator의 구성과 각 항목의 의미를 설명하고 있다.

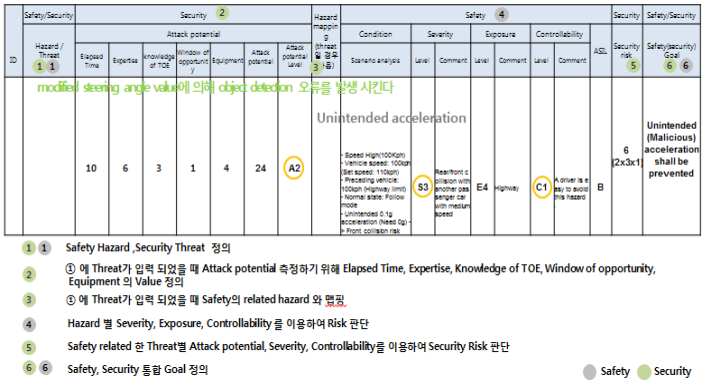


그림 3. Combinded risk assessment calculator for safety and security

ISO 26262에서는 FTA(Fault Tree Analysis)를 통해 Failure식별과 디자인의 강건함을 확인하는데 여기에 security를 고려하기 위해 Attack tree[6]추가 반영하여 FTA를 확장하였다. 이를 통해 safety와 security 동시에 분석 가능 할 뿐만 아니라 safety와 security 요구사항 간의 모순을 확인할 수 있다.

제안된 방법론을 Adaptive cruise control 시스템에 적용을 통해 Security threat 61개를 도출하였다. 그리고 61개의 Security threat 중에 분석을 통해 safety 문제를 발생시킬 수 있는 34개의 Threat를 확인하였다. 발견된 34개의 Threat를 safety와 security를 통합하여 risk assessment를 수행하였다. 확장된 FTA를 통해 식별된 Threat가 누락되어 있는지 검증하였고 식별된 Threat를 줄이거나 막기 위한 Security measure를 도출하였다.

그리고 Future work으로 risk assessment을 위한 operation, privacy, finance 측면을 추가하여 평가 기준을 수립이 필요하며 Risk Assessment 결과에 따른 prevention 또는 mitigation 관리 기준 수립할 예정이다.

참고문헌

[1] Stephen Checkoway, Damon Mccoy,. “Comprehensive Experimental Analyses of Automotive Attack Surfaces.” USENIX Security, 2011.

[2] International Organization for Standardization, “ ISO 26262 Road vehicles – Functional safety” 2011

[3] SAE International,“J3061 Cybersecurity Guidebook for Cyber-Physical Vehicle Systems“ January 2016

[4] International Organization for Standardization, “ ISO 15408 : information technology – security techniques – evaluation criteria for IT security ”, 2008

[5] International Organization for Standardization, “ ISO 18045 : common methodology for information technology security evaluation ”, 2008

[6] Schneier, B. , "Attack Trees: Modeling Security Threats," Dr. Dobb's Journal, December 1999

시스템 오브 시스템즈(SoS) 사례 분석을 통한 온톨로지 기반의 SoS 메타모델 개발

백영민○, 박수민, 신용준, 배두환
한국과학기술원
{ ymbaek, smpark, yjshin, bae}@se.kaist.ac.kr

Development of Ontology-based System-of-Systems Meta-model Based on the Analysis of SoS Case Scenario

Young-Min Baek○, Sumin Park, Yong-Jun Shin, Doo-Hwan Bae
School of Computing, Korea Advanced Institute of Science and Technology (KAIST)

요 약

현대의 소프트웨어 산업은 고차원의 기능과 서비스를 제공하기 위해 다양한 컴포넌트 시스템들의 협업을 필요로 한다. 이러한 요구에 대응하기 위해 여러 독립적인 구성 시스템들을 통합하여 다양한 분야에서 시스템 오브 시스템즈(System-of-Systems, SoS)가 연구 및 개발되고 있다. 하지만 SoS는 크기가 매우 큰 대규모 시스템이며 시스템 구성이 매우 복잡하기 때문에 SoS 전반에 걸친 체계적인 설계와 개발을 위해서는 모델 기반의 분석 기법이 요구된다. 본 연구는 SoS와 가장 유사한 시스템인 다중 에이전트 시스템(Multi-agent System)의 메타모델 기법을 분석하고 SoS 사례 시나리오를 조사함으로써 온톨로지 기반의 SoS 메타모델인 M2SoS(Meta-model for SoS)를 개발하고자 한다.

1. 서 론

시스템 오브 시스템즈(System-of-Systems, 이하 SoS)는 상위 수준의 목표를 달성하기 위해 여러 독립적인 시스템들을 통합하여 개발되는 대규모 복잡 시스템을 가리킨다. SoS는 개별 시스템이 달성할 수 없는 상위 수준의 공통 목표를 달성하기 위해, 여러 구성 시스템(Constituent System, 이하 CS)들을 통합하고 그로부터 상위 수준의 기능 및 서비스 수행 역량을 기대하는 시스템이다 [1, 2]. 이와 같이 복잡한 SoS의 개발과 검증을 위해, 다양한 모델 기반 소프트웨어/시스템 공학 기법들이 적용되고 있다. MBSE를 활용하여 SoS의 특성을 효과적으로 표현하기 위한 모델링 언어를 개발하거나 [2], SoS를 아키텍쳐 관점에서 분석하고 테스팅 및 검증을 수행한 연구 [3] 등이 진행되어 왔다. 그러나 이들 연구는 SoS를 각각 다른 기반 모델로 표현하여 SoS의 부분적 특성을 표현하기 위한 기법을 제안하는 데에 그치고 있으며, SoS의 분석 및 개발에 앞서 대상 SoS의 핵심적인 특성을 포괄할 수 있는 모델 기반 기법 및 방법론 연구는 부족한 실정이다.

SoS와 가장 유사한 시스템으로 다중 에이전트 시스템(Multi-agent System, 이하 MAS)이 있다. MAS 역시 다수의 컴포넌트 에이전트가 협력하여 상위 수준의 서비스를 제공하는 특징을 갖기 때문에 SoS와 많은 특성을 공유한다. MAS를 연구한 다양한 분야에서는 에이전트 및 MAS의 핵심적인 특성을 포괄하는 모델 기반 기법 개발을 위해 메타모델(meta-model)을 적극 활용해 오고 있다 [5, 6]. 이들의 메타모델 기반 기법은 시스템의 분석 및 설계 단계에서 모델링의 대상이 되는 요소들을 철저하게 정의할 수 있도록 가이드라인을 제공해 줄 수 있을 뿐만 아니라, 대상 모델들 간의 관계 정의를 통해 체계적인 시스템 설계의 절차를 마련해 준다. 특히, 시스템 전반의 특징을 포괄적으로 표현할 수 있는 메타모델을 통해 시스템의 다양한 이해관계자가 공통된 이해를 바탕으로 지식을 구축하고, 분석, 설계 및 개발에 활용할 수 있다.

따라서 본 연구는 이러한 메타모델 기반 기법의 이점을 모델 기반 SoS 공학을 위해 활용하고자, 개발 대상 SoS를 효과적으로 표현할 수 있는 온톨로지 기반의 메타모델을 제안한다. 이를 위해 유사 시스템인 MAS의 대표적인 메타모델 기법을 분석하고 SoS 사례 시나리오를 분석함으로써 SoS 메타 모델을 개발하고 활용 가능성을 분석한다.

*이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원(No. R0126-18-1101, (SW 스타랩) 모델 기반의 초대형 복잡 시스템 분석 및 검증 SW 개발)과 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임 (2017M3C4A7066212)

2. 배 경

2.1 시스템 오브 시스템즈(System-of-Systems)

기존의 다양한 SoS 공학 연구 문헌들에서 시스템 오브 시스템즈(이하 SoS)는 '하위 시스템 수준에서는 개별적으로 달성하기 어려운 상위 수준의 목표를 달성하기 위해 독립적으로 운영 및 관리되는 시스템들로 구성/통합된 대규모 복잡 시스템'으로 정의된다 [1, 2, 7]. 또한 SoS를 기존의 다른 시스템들과 구분 짓기 위해 주요 특징들을 통해 분석되었는데, CS들이 운영 및 관리 독립성을 갖는 특징이 있으며, 자율적인 CS들이 통합되면서 소속성/연대성, 연결성, 다양성, 창발성 등을 주요 특성으로 갖는다. 또 다른 연구에서는 모델 기반 SoS 공학을 위한 분석 기준으로 8가지 차원(dimension)을 자율성, 독립성, 분산, 진화, 동적 재구성, 창발적 행위, 상호의존성, 상호운용성으로 정의했다 [7].

2.2 다중 에이전트 시스템(MAS)과 메타모델 기반 기법

SoS와 많은 공통 특성을 공유하고 있는 다중 에이전트 시스템(Multi-agent System, 이하 MAS)은 공통 목표를 달성하기 위해 자율적인 행동과 의사 결정을 수행하는 에이전트들로 구성된 상위 수준 시스템이다. MAS를 대상으로 한 메타모델 기반 방법론으로는 Gaia 방법론과 O-MaSE 방법론이 대표적이며 [5, 6], 이들은 MAS를 이루는 컴포넌트들와 그들 사이의 관계를 정의하고 있다. Gaia 방법론은 객체 지향 기법에 기반하여 MAS를 하나의 조직으로 간주하여, 이를 설계하기 위한 조직 구조 모델, 역할 모델, 상호작용 모델, 에이전트 모델, 서비스 모델 등을 명세할 수 있도록 가이드라인을 제공해 준다. 또 다른 대표적인 MAS 메타모델인 O-MaSE 역시 에이전트의 조직 및 상호작용 관점에서 MAS에 대한 표현력을 높이고자 제안된 기법이다. O-MaSE는 각 에이전트가 특정한 MAS 수준의 목표(Goal)에 따라 역량(Capability)과 역할(Role)을 명세할 수 있도록 체계적인 가이드라인과 모델링 기법을 제공한다.

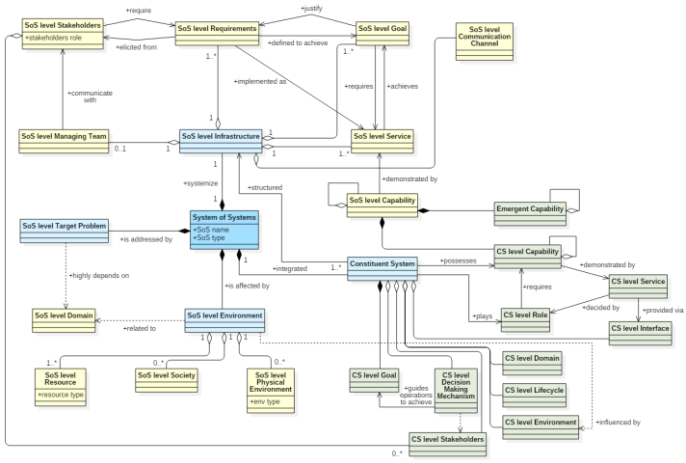


그림 1. SoS 온톨로지 표현을 위한 M2SoS (간략화된 M2SoS)

3. SoS 사례 분석을 통한 온톨로지 기반의 SoS 메타모델 개발

3.1 다중손상사고(MCI) 대응 시스템 사례 분석

다중손상사고(Mass Casualty Incident, 이하 MCI)는 매우 높은 수의 사상자와 피해자를 발생시키며 기존 의료 및 구조 자원으로 해결이 어려운 대형 사고나 자연 재난을 의미한다. 이러한 MCI 상황은 단시간 내에 통합되지 않은 개별 시스템의 역량과 자원으로 해결하기 어렵기 때문에, 적절한 시스템들의 협업에 기반해 지속적인 의료/구조 자원을 제공할 수 있도록 SoS 수준의 시스템 구축이 요구된다. 본 연구에서는 이와 같이 MCI에 대응하기 위한 시스템을 MCI 대응 SoS (MCI Response SoS, 이하 MCIRSoS)로 정의하며, 실제 SoS 사례가 포함하는 요소를 철저히 도출해 내기 위해 2000년 이후에 발행된 약 30개의 MCI 관련 문헌(부록 참조)을 문헌 조사를 수행했다.

문헌 조사를 통해 실제 MCI 대응 시스템을 구성하는 요소(entity)를 분석하고 실제 SoS 사례가 갖는 특징을 분석할 수 있었다. 첫째, MCIRSoS는 명시적으로 SoS 수준 목표, 요구사항, 서비스, 행동, 통신 요소를 포함하고 있으며, SoS 수준 목표를 달성하기 위해 CS들의 역량에 기반한 구체적인 역할 할당을 통해 임무를 수행한다. 둘째, 이를 위해 SoS 수준 요소와 CS 수준 요소를 명확하게 구분하여 정의하고 있으며 SoS 수준 공학자/관리자와 CS 수준 공학자/관리자의 역할이 뚜렷이 구분된다. 셋째, MCI는 높은 불확정성을 갖는 환경 요소를 포함하고 있기 때문에, 많은 문헌들에서 철저한 환경 요소 분석을 통한 대응 프로시져를 정의하고 있다. 넷째, MCIRSoS를 이루고 있는 요소들은 계층적으로 구성되어 있으며, 제일 상위 수준 요소는 SoS 수준 대상 문제, SoS 수준 인프라(infrastructure), SoS 수준 환경, 그리고 다수의 CS로 이루어진 조직으로 구분할 수 있었다.

3.2 SoS 메타모델 개발 요구사항 정의 및 단계별 요소 정의

앞서 수행한 조사 결과를 바탕으로 우리는 다음과 같이 메타모델 설계의 네 가지 요구사항을 정의할 수 있었다:

i. SoS 수준 요소와 CS 수준 요소의 구분

ii. SoS 수준 목표, 요구사항, 서비스, 행위와 CS 수준 역할, 역량간 명시적 관계 정의

iii. SoS 운영 시 분석이 필요한 환경 요소 분류 및 정의

iv. 온톨로지 표현이 가능한 SoS 메타모델 개발

위의 요구사항을 바탕으로 개발한 SoS 메타모델인 M2SoS (Meta-model for System-of-Systems)는 그림 1에 간략하게 표현하였다(전체 버전은 부록 참조). 그림과 같이, M2SoS는 MCIRSoS 사례 분석을 통해 도출한 SoS의 상위 수준 구성 요소를 포함하고, 명시적으로 SoS 수준 요소와 CS 수준 요소를 구분하고 있다. 또한 SoS 수준 환경 요소를 SoS 수준 최상위 요소로 정의함과 동시에, SoS 수준 자원(resource), 사회(society), 물리적 환경(physical environment)으로 분류하여 명세하였다. 그리고, M2SoS는 개발 대상 SoS의 온톨로지를 표현할 수 있도록 SoS를 이루는 구성 요소와 그 관계를 정의함으로써 도메인-일반적인(domain-general) 메타모델의 역할을 수행할 수 있다. 다시 말해, 본 메타모델을 활용하여 SoS의 다양한 이해관계자들이 공통된 SoS에 대한 이해를 수행하기 위한 온톨로지를 구축할 수 있다.

4. M2SoS 분석 및 확장 방안

MAS의 분석과 설계를 위한 다양한 메타모델들(예. Gaia, O-MaSE)은 에이전트 기반의 시스템과 상호작용을 명세하는 데에는 효과적이었지만, 구성 시스템의 운영/관리 독립성을 가지며 복잡도가 높은 SoS를 표현하는 데에는 한계가 존재했다. 우선, MAS 대상 기법은 컴포넌트 시스템(Agent 혹은 CS) 수준 요소와 상위 시스템(MAS 혹은 SoS) 수준 요소를 명시적으로 구분하지 못한다. 둘째, MAS 대상 기법은 컴포넌트 시스템의 운영 및 관리 독립성을 고려하지 않아도 되므로 SoS의 CS가 독립적인 목표 하에 운영 및 관리되기 위한 요소(CS 수준 목표, 생명주기 등)를 표현하지 못한다. 셋째, MAS 대상 기법은 전반적으로 개발 대상 SoS의 온톨로지를 정의하기 위한 요소들을 모두 포함하고 있지 않은 반면, M2SoS는 SoS의 핵심 특성을 충분히 반영할 수 있도록 설계되었다. 표현력 비교 분석 결과는 부록에 첨부하였다.

5. 결론 및 향후 연구

대규모 복잡 시스템인 시스템 오브 시스템즈(System-of-Systems, SoS)의 개발을 위해서 개발 대상 SoS를 체계적으로 분석하기 위한 기법이 요구되는데, 본 연구는 이러한 문제를 메타모델 기반 접근 방식으로 다루었다. 본 연구에서 제안한 SoS 메타모델인 M2SoS (Meta-model for SoS)는 다중 에이전트 시스템의 메타모델 기반 기법에 기반하여 기본 뼈대를 구성하고, SoS 사례 시나리오인 다중손상사고 대응 SoS 문헌 분석을 통해 요소를 정의하여 개발되었다. 본 M2SoS는 SoS의 다양한 이해관계자가 개발 대상 SoS에 대한 공통된 지식을 구축하는 데에 온톨로지로 활용이 가능하며, SoS를 설계하는 단계에서 모델링의 대상이 되는 요소과 요소 사이의 관계를 파악하는 데에 활용될 수 있다. M2SoS는 향후 SoS의 모델 기반 시뮬레이션 및 통계적 검증에도 활용될 예정이며, M2SoS에 정의된 요소들에 대한 구체적인 모델링 기법이 개발될 예정이다.

참 고 문 헌

[1] J. Boardman, and B. Sauser, "System of Systems – the Meaning of Of," IEEE/SMC International Conference on System of Systems Engineering (SoSE), 2006.

[2] W. C. Baldwin, and B. Sauser, "Modeling the Characteristics of System of Systems," IEEE International Conference on System of Systems Engineering 2009 (SoSE 2009).

[3] M. Hause, "The Unified Profile for DoDAF/MODAF (UPDM) Enabling Systems of Systems on Many Levels," Systems Conference, San Diego, CA, USA, 2010.

[4] C. Bernon, M. Cossentino, and J. Pavon, "Agent-Oriented Software Engineering," 2005 Cambridge University Press, vol. 20, issue 2, pp. 99-116, 2005.

[5] S. A. DeLoach, J. C. Garcia-Ojeda, "O-MaSE: A Customizable Approach to Designing and Building Complex, Adaptive Multi-agent Systems," International Journal of Agent-Oriented Software Engineering, vol. 4, issue 3, 2010.

[6] F. Zambonelli, N. R. Jennings, M. Wooldridge, "Developing Multiagent Systems: The Gaia Methodology," Journal of ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 12, issue 3, 2003.

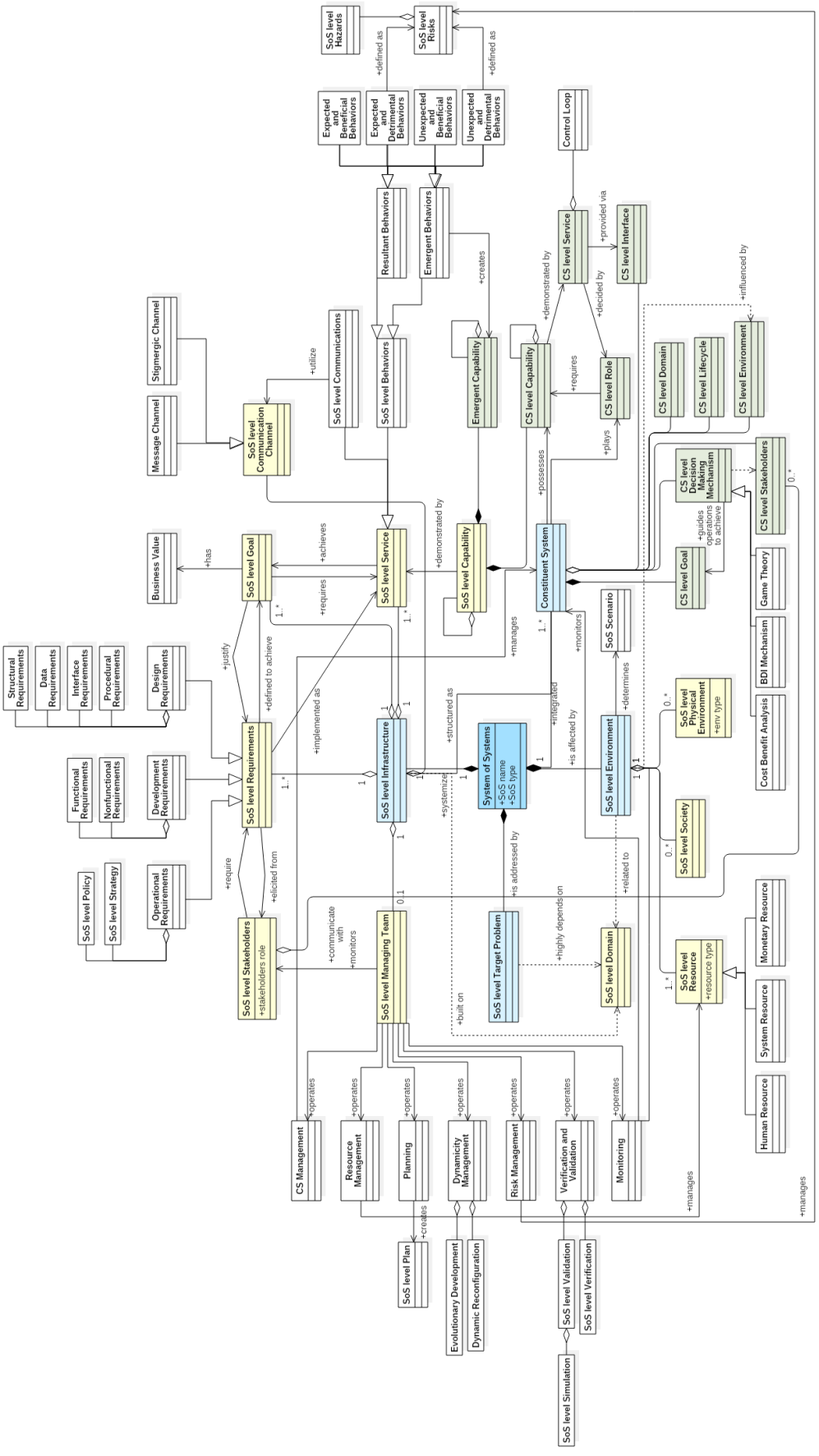
[7] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions," Journal of ACM Computing Surveys (CSUR), vol. 48, issue 2, 2015.

부 록

부록 A. 다중손상사고 조사 문헌

번호	문헌 이름	문헌 작성 기관	연도
1	Rogers Fire Department Standard Operating Procedures – Policy Title: Mass Casualty Incidents	Rogers Arkansas	2013
2	Emergency Medical Services Policies and Procedures – Policy Title: Multi Casualty Incident Response	County of Ventura Health Care Agency	2014
3	Mass Casualty Incident	Shao Foong Chong	2013
4	Planning & Triage	Peds (Pediatrics in Disasters)	
5	Florida Field Operations Guide – Mass Casualty	Florida State	2012
6	Dane County Regional Mass Casualty Incident Response Plan	Metropolitan Medical Response System (MMRS)	2006
7	MCI PLAN – Franklin County Multiple Casualty Incident Response Plan	Franklin County EMS, MCI Planning Committee	2009
8	The Massachusetts Emergency Medical Services (EMS) Mass Casualty Incident (MCI) Plan	The Massachusetts Department of Public Health	2016
9	Integrated Explosive Event and Massive Casualty Event	Greater New York Hospital Association	–
10	Emergency Management Plan for Mass Casualty Incidents	–	–
11	Mass Casualty Incident Plan – Adopted by the Marion County Fire Defense Board	Marion County Fire Defense Board	2001
12	Mass Casualty Response: NHS Tactical Command Framework	NHS England Cumbria Northumberland Tyne and Wear Area Team	2014
13	Appendix N – Detailed Sample Scenarios	–	–
14	Multi-Casualty Incident (MCI) Response Plan	Santa Barbara County – Emergency Medical Services Agency	2013
15	Mass Casualty Management Systems – Strategies and Guidelines for Building Health Sector Capacity	World Health Organization (WHO)	2007
16	National Incident Management System	Homeland Security	2008
17	MCI Triage Drill	Alabama Fire College Workplace Safety Training Program	–
18	Utah Department of Health Bureau of EMS and Preparedness Emergency Operations Plan	Utah Department of Health	2011
19	Recommendations of the Standing Committee on Multiple Casualty Incident Planning and Evaluation	The Commonwealth of Massachusetts	2004
20	Emergency Medical Services – Multi-Casualty Incident Plan	Contra Costa County	2012
21	Multiple Casualty Incident (MCI) Response Plan	Monterey County EMS	2014
22	MCI Operational Policy – Clear Creek County Emergency Services	Clear Creek Fire Authority/ Clear Creek EMS	2011
23	Multi-Casualty Incident Policy	San Francisco Emergency Medical Services Agency	2005
24	Mass Casualties Incident Plan for NHS Scotland	NHS Scotland Resilience	2015
25	Multi-Casualty Incident Policy	NPS EMS Field Manual	2016
26	ODEMSA Mass Casualty Incident (MCI) Plan	Heidi M. Hooker, ODEMSA	2014
27	SUFFOLK Mass Casualty Plan	Suffolk NHRP on behalf of Suffolk Resilience Forum	2015
28	Operational Templates and Guidelines for EMS Mass Incident Deployment	FEMA	2012
29	Hospital Medical Surge Planning for Mass Casualty Incidents	Florida Department of Health	–
30	Mass Casualty Incident (MCI): An Overview	Jim Thomas, Captain	–
31	2017 소방전술 I: 신임교육과정(화재3)	중앙소방학교(NFSA)	2017

부록 B. M2SoS: 온톨로지 기반의 SoS 메타모델



부록 C. MAS 메타모델과 M2SoS의 표현력 비교 분석

구성 요소	MAS 메타모델		M2SoS
	Gaia	O-MaSE	
컴포넌트 시스템(Agent or CS) 수준 요소			
컴포넌트 시스템	Agent (Organizational Agent)		Constituent System (CS)
컴포넌트 역할	Role AgentType	Role	CS-level Role
컴포넌트 목표	×	×	CS-level Goal
컴포넌트 서비스	Service Activity Action Responsibility Permission	Action	CS-level Service
컴포넌트 역량	×	Capability	CS-level Capability
컴포넌트 인터페이스	×	×	CS-level Interface
컴포넌트 의사결정 매커니즘	Permission	Plan, Action	CS-level Decision Making Mechanism
컴포넌트 이해관계자	×	×	CS-level Stakeholder
컴포넌트 도메인	×	Domain Model	CS-level Domain
컴포넌트 생명주기	×	×	CS-level Lifecycle
컴포넌트 환경	Environment	Environment Object, Property	CS-level Environment
상위 시스템(MAS or SoS) 수준 요소			
상위 수준 시스템	MAS		SoS
상위 시스템 조직	Organizational Rule, Structure	Organization	×
상위 시스템 인프라	×	×	SoS-level Infrastructure
상위 시스템 행위	×	×	SoS-level Service
상위 시스템 통신	Communication, Protocol	Protocol	SoS-level Communication
상위 시스템 관리자	×	(Actor)	SoS-level Managing Team
상위 시스템 목표	×	Goal	SoS-level Goal, Target Problem, Requirements
상위 시스템 정책	×	Policy	SoS-level Policy
상위 시스템 도메인	×	×	SoS-level Domain
상위 시스템 환경	×	×	SoS-level Environment
상위 시스템 관리	×	×	Management of CS, Resource, Dynamics, Risk,
시스템 위험	×	×	SoS-level Risk, Hazards

Concolic 테스팅 도구 CREST 의 사용자 친화성 향상 연구:

Windows OS 로의 포팅과 개선된 CREST UI 을 통한
CREST 활용 및 분기 커버리지 분석 작업의 효율 증가

김현우 ○ 김윤호 김문주

한국과학기술원

hyunoo@kaist.ac.kr yunho.kim03@gmail.com moonzoo@cs.kaist.ac.kr

Improving Applicability and User Interface of CREST :

Porting CREST to Windows Platform and Improving Work Efficiency
for Coverage Analysis

Hyunwoo Kim○ Yunho Kim Moonzoo Kim

Korea Advanced Institute of Science and Technology

요 약

본 연구에서는 Concolic 테스팅 기법 도구인 CREST 를 포팅하여 기존에는 불가능했던 Windows OS 에서도 동작이 가능하도록 확장했다. 추가로, 기존에는 CREST 가 달성한 분기 정보들을 사용자가 확인하려면 추가로 해석하는 작업이 필요했으나, CREST 의 기존 출력을 변경하고 분기 정보 확인/분석에 유용한 정보들을 추가로 제공하도록 개선하였다. 이로 인해 CREST 를 사용한 테스팅 과정에서 사용자의 작업 부담을 줄이고, 작업 효율을 늘리고자 하였다.

1. 서 론

소프트웨어의 늘어나는 필요성과 요구사항으로 인해 그 복잡성이 높아짐에 따라 자동화 테스팅에 대한 중요성 역시 높아지고 있다. 테스팅의 목적은 소프트웨어 내부에 존재하는 에러를 검출하는 것으로, 이를 달성하기 위해서 소프트웨어의 가능한 많은 path 탐색을 필요로 한다.

테스팅 기법 중 하나인 Concolic 기법[1]은 소프트웨어를 실행한 뒤, 실행한 path 을 기반으로 새로운 path 을 탐색하는 테스트를 생성하고, 이 테스트로 타겟 소프트웨어를 다시 실행하는 일련의 과정을 반복한다. 이론적으로 Concolic 기법을 사용하면 타겟 소프트웨어의 모든 path 을 탐색하는 것이 가능하다.

CREST[2]는 C 프로그램용 Concolic 기법 도구이다. CREST 는 현재 Linux 환경만 지원하고 있기 때문에 Linux 환경에서 동작하지 않는 소스 코드를 대상으로 CREST 를 실행하기 위해서는 Linux 환경에서 동작하는 소스 코드로 변환하는 추가 작업이 필요한 상태이다. 또한 CREST 실행 결과의 분석에 필요한 정보들은 사용자가 곧바로 알아볼 수 없는 형태이기 때문에 추가로 해석하는 작업이 필요하다.

본 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원(NRF-2016R1A2B4008113), 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대 정보 컴퓨팅기술개발사업의 지원(NRF-2017M3C4A7068177), 정부(교육부)의 재원으로 한국연구재단의 지원(NRF-2017R1D1A1B03035851)을 받아 수행한 연구임.

본 논문에서는 CREST 을 사용하고, 결과를 분석하는데 필요한 노력과 시간을 줄여서 효율적인 CREST 사용이 가능하고 생산성을 높이고자 하였다. 본 논문에서 수행한 작업은 구체적으로 다음과 같다. 첫 번째로, 가장 많이 사용되는 개발 환경 OS 중 하나인 Windows OS 에서 CREST 사용이 가능하도록 CREST 포팅을 수행하였다. 두 번째로, CREST 결과 분석에 필요한 정보들을 사용자가 쉽게 이해할 수 있는 형태로 추가 제공하여 결과 분석에 필요한 추가 비용을 크게 줄이고자 했다.

2. CREST 소개

CREST 는 Concolic 기법(Concrete + Symbolic execution)을 사용하는 C 프로그램 용 테스팅 도구이다. CREST 는 먼저 Concrete execution 을 통해 Symbolic execution 을 유도한다. Concrete execution 은 입력 변수에 값을 넣고, 타겟 프로그램의 한 path 를 실행시킨다. Symbolic execution 은 입력 변수를 심볼로 가정하고, Concrete execution 에서 실행한 path 에 도달하기 위해 입력 변수가 만족해야하는 심볼릭 경로 수식(symbolic path formula)을 수집한다. 수집한 심볼릭 경로 수식을 수정하여 아직 실행하지 못한 path 에 도달하기 위한 새로운 심볼릭 경로 수식(next formula)을 만들어낸다. 심볼릭 경로 수식을 수정하는 방법은 여러가지가 존재하며 기본적인 “DFS” 방법의 경우 마지막 수식에 negation 을 취해서 새로운 심볼릭 경로 수식을 획득한다. Z3 solver[3]에 새롭게 구한

심볼릭 경로 수식을 전달하면 심볼(입력 변수)의 해(값)를 구할 수 있다. 새롭게 구한 입력 변수 값으로 다시 타겟 프로그램을 실행시키고, 또 다른 입력 변수의 값을 추출하는 일련의 과정을 반복하면서 프로그램의 path 들을 탐색하게 된다.

CREST 는 Instrumentation 작업이 선행 되야 하는데 이 작업은 심볼릭 경로 수식 기록에 필요한 probe 들을 코드에 삽입해주는 작업이다. 특히, 분기마다 __CrestBranch 함수 호출 코드가 삽입되며, 인자 중 하나로 분기 식별이 가능한 분기 id 를 가진다. CREST 를 실행하면 달성한 분기 id 가 "coverage"파일에 기록되는데, __CrestBranch 의 인자인 분기 id 와 "coverage"파일에 기록 된 분기 id 를 매칭시켜서 분기 달성 여부, 위치 등의 파악이 가능하다.

3. 사용자 친화성 향상 기법

3.1 Windows 포팅

CREST 가 개발 환경에서 널리 사용되는 Windows OS 를 지원하지 못하여 초래되는 문제와 비용 비효율성을 없애고자 Windows OS 에서도 CREST 의 사용이 가능하도록 포팅하는 작업을 수행했다. 아래 표 1 은 포팅 과정에서 변경 된 주요 구문들이다.

표 1 포팅 시 변경 된 주요 구문

구성 요소	주요 변경 구문
Front-end	1 CIL 에 의해 파싱이 불가능한 구문을 제거하는 코드 추가. 2 Visual studio 컴파일러가 지원하지 않는 GCC 확장 구문 생성 방지.
Middle-end	1 Visual studio 컴파일러가 지원하지 않는 CREST 코드 안의 GCC 확장 구문 제거. 2 Visual Studio 가 지원하지 않는 POSIX 라이브러리 대체.
Back-end	1 Visual Studio 가 지원하지 않는 POSIX 라이브러리 대체. 2 Visual Studio 가 지원하지 않는 출력 형식 대체.

3.2 CREST 달성/미달성 분기정보 UI 향상

CREST 도구에 이미 탑재 되어있는 print_execution 명령어의 결과를 사용자 친화적으로 변경하였고, 커버리지 결과 분석에 필요한 추가 정보가 담긴 파일 "new_coverage"를 생성하는 새로운 명령어 gen_new_cov 를 추가하였다.

3.2.1 print_execution

print_execution 은 CREST 에 내재된 명령어로, CREST 로 프로그램을 실행시킨 후 사용 가능한 명령어이다. 본 명령어 사용 시 CREST 가 타겟 프로그램을 실행했을 때 사용 된 심볼릭 변수가 갖는 값, 심볼릭 경로 수식, 실행 된 분기 id 가 출력된다. 이러한 정보들을 바탕으로, 심볼릭 변수가 갖는 값과 그에 대응되는 심볼릭 경로 수식을 확인할 수 있다. 개선 된 사항은 다음 1 - 4 항목과 같다.

1. 가명 심볼릭 변수명(x0, x1, x2, …)이 아닌 실제 변수명 출력.
2. 심볼릭 경로 수식 출력을 기존 전위(pre-order)방식에서 사용자에게 친숙한 중위(in-order) 방식으로 변경.
3. 형 변환 표기 시, 실제 형 변환 된 자료형으로 출력.
4. 각 정보 출력 시, 위치 정보(line, file)를 함께 표기하는 방식으로 변경.

3.2.2 gen_new_cov

gen_new_cov 는 새로 추가 된 스크립트로서, 실행 시 CREST 가 달성한 상세 분기 정보가 담긴 파일 "new_coverage"을 생성한다.

분기 id 를 소스 코드와 매칭시켜볼 필요 없이 new_coverage 파일에는 분기에 대한 상세 정보(분기 id, 분기식, 달성 여부, 분기 위치 정보)와 분기의 동향(파일 별, 함수 별 분기 커버리지)을 파악할 수 있는 분기 요약 정보를 추가했다. 따라서, 기존 coverage 파일의 분기에 대한 정보 부족으로 인한 사용자의 추가 수행 작업을 줄일 수 있었다.

4. 결론 및 향후 연구

본 논문에서는 Linux 환경에서만 동작이 가능했던 CREST 를 Windows OS 에서도 사용이 가능하도록 확장하는 작업을 수행했다. 지금까지는 Windows OS 에서만 실행 가능한 C 소스 코드에 대해 CREST 적용을 포기하거나, Linux 환경에서 동작가능하도록 변환하는 작업을 거치고 난 뒤 OS 간의 동작 차이를 감안하고 CREST 를 적용해야 했다. 하지만 본 연구에서 수행한 포팅 작업으로 인해 Windows OS 에서 동작하는 소스 코드에 대해서도 쉽게 CREST 적용이 가능해졌다.

두번째로 기존 CREST 에서 제공했던 print_execution 의 출력 메시지를 개선하고, 커버리지 분석에 필요한 추가 정보가 담긴 new_coverage 를 사용자에게 제공하여 테스팅에 필요한 시간을 크게 줄이고자 하였다.

향후 연구로써 현재 포팅 된 CREST 는 Windows OS 의 이점인 GUI 특성을 살리도록, 현재 프롬프트 창에서 명령어로 동작하는 CREST 에 GUI 인터페이스를 추가할 계획이다.

참 고 문 헌

[1] K. Sen, D. Marinov, G. Agha, "CUTE: a concolic unit testing engine for C," in *ACM SIGSOFT Software Engineering Notes*, 2005.

[2] J. Burnim and K. Sen, "Heuristics for scalable dynamic test generation," in *ASE*, 2008.

[3] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *TACAS*, 2008.

하이브리드 문서 표현을 이용한 효율적인 문서 분류

최성옥 ○, 백종문
한국과학기술원
sungok2@kaist.ac.kr, jbaik@kaist.ac.kr

An Effective Document Classification Using a Hybrid Document Representation

Sungok Choi○, Jongmoon Baik
Korea Advanced Institute of Science and Technology

요 약

일반적으로 문서 분류는 단어들을 기반으로 특징을 추출하고 문서 벡터를 만들어 기계학습 알고리즘을 통해 분류하게 된다. 본 논문에서는 한국어 문서 분류를 위한 효과적인 모델을 제안한다. *doc2vec*에 코사인 유사도 대신 자체 고안한 *C-ISM*(Cosine - Inversed Space and Magnitude)을 적용하고 *TF-IDF*를 결합하여 벡터를 생성한 뒤 기계 학습 알고리즘을 이용하여 한국어 문서를 자동으로 분류한다. 검증을 위해서 한국 정부가 공개한 전자문서를 입력 데이터로 사용하였다. 제안하는 모델에 의해 분류를 시도한 결과 기존 벡터화 방법보다 4.8% 높은 분류 성능을 보인다

1. 서 론

행정 업무 관리 시스템의 발전은 기관 간 문서 유통을 크게 증가 시켰다. 이에 따라 문서를 분류하여 적절한 담당자에게 일을 지정해주는 것은 매우 중요한 일이 되었다. 일반적으로 수신된 문서는 분배 담당자가 먼저 내용을 분석하고 각 직원들의 업무 관련성을 파악하여 문서를 할당하는 일련의 과정을 거치게 된다. 이렇게 문서를 할당하는 과정이 수작업으로 진행되면 업무추진이 늦어지고 부정확한 분류로 인해 직원들의 불만을 야기하는 등 여러가지 문제를 발생시킨다. 따라서 신속 정확한 업무 진행을 위해서는 정확한 문서 분류가 선행되어야 한다.

본 연구에서는 벡터 유사도 측정법의 오류를 제거하고 문서 표현 방법의 조합을 통해 기존 방식보다 한국어 문서 분류 자동화에 좀더 효과적인 모델을 개발하는 것이 목적이다.

2 장은 유사도 측정의 분류 체계와 개념 및 단점에 대해 간략하게 언급한다. 3 장에서는 *C-ISM* 이라 불리는 새로운 유사도 측정법, *TF-IDF*[1]와 *doc2vec*[2]의 조합을 사용하여 제안 된 분류 모델을 설명, 4 장에서는 실험 설정 및 결과를 살펴본다. 마지막으로 5 장에서 결론과 향후 연구에 대해 설명한다.

2. 관련 연구

텍스트를 수학적으로 계산할 수 있는 다른 형식으로 변환하는 프로세스를 문서 표현(Document Representation)이라고 한다. 이 문서 표현은 주로 카운트 기반 방법인 *TF-IDF* 와 예측 방법인 *doc2vec* 이 있다. 그리고 벡터공간 모델에서 유사도 측정 방법에는 일반적으로 코사인 유사도와 유클리드 거리 측정이 사용된다. 하지만 이러한 유사도 측정 방법은 각도 또는 거리만을 고려하기 때문에 좋은 유사도 측정법이라고 판단하기 어렵다[3].

3. 제안 모델

3.1 개요

한국어 문서를 분류함에 있어서 먼저 문서마다 누구의 문서인지 라벨링을 하고 한글 텍스트 데이터에서 형태소 분석과 품사 태깅 처리를 한다. 그런 다음 새로운 *C-ISM* 유사도 측정법을 접목한 *doc2vec* 을 통해 첫 번째 벡터화 과정을 수행하고 그 후 *TF-IDF* 를 통해 가중치를 부여한다. 그리고 잘 알려진 기계학습 알고리즘을 통해 분류를 시도하여 예측 레이블과 원본 레이블이 일치하는지 확인한다.

3.2 단어 임베딩(Word Embedding)과 *C-ISM* 유사도

문서 분류의 입력 데이터는 원문 내용이지만 이러한 텍스트의 배열을 기계 학습 알고리즘에 직접적으로 적용 할 수는 없다. 텍스트를 수학적으로 계산할 수 있는 벡터 형식으로 변환하는 과정을 거쳐야 한다. 최근 인공신경망을 이용한 *doc2vec* 이 각광 받고 있다. *doc2vec* 은 예측 값과 목표 값을 비교하여 차이를 최소화하도록 가중치 벡터 값을 반복적으로 조정한다. 여기서 두 벡터를 비교할 때 코사인 유사도가 사용되는데 코사인 유사도는 벡터 크기를 고려할 수 없다는 고유의 문제를 가지고 있다. 이러한 단점을 보완하고자 새로운 유사도 측정법으로 *C-ISM* (Cosine-Inversed Space and Magnitude)을 고안했다.

C-ISM 유사도는 두 벡터 간의 각도와 크기 차이를 모두 고려하여 기존 유사도 측정 도구에서 식별되는 오류를 최대한 보완하고자 한다. *C-ISM* 은 두 벡터가 이루는 각도가 0 에 가까워짐에 따라 유사도가 증가하고 벡터 크기 차이가 변함에 따라 유사도가 선형적으로 변하기 때문에 유사도 측정에 있어 유용한 도구라고 판단된다. *C-ISM* 은 식 (1)을 사용하여 구할 수 있다.

$$\text{C-ISM}(\vec{A}, \vec{B}) = \frac{\cos\theta}{\frac{1}{2}ab\sin\theta + |a-b| + 1} \qquad \text{식 (1)}$$

3.3 *doc2vec* 과 *TF-IDF* 의 조합

doc2vec 과 *TF-IDF* 의 조합은 상호 한계점을 보완하는데 있어 유용하다고 판단된다. 먼저, 텍스트 데이터를 *doc2vec* 을 통해 벡터로 변환하여 각 단어에 대한 벡터를 구한다. 또한, 텍스트 데이터를 *TF-IDF* 의 입력 데이터로 사용하여 텍스트에 대한 새로운 벡터값을 생성한다. 생성된 두 벡터의 벡터곱을 함으로써 doc2vec 의 특징과 TF-IDF 의 특징이 조합이 될 수 있다.

4. 평 가

4.1 데이터 세트

한국 정부에서 공개한 전자문서[1] 자료 중 임의의 한 기관의 자료를 입력 자료로 선정 하였다.

4.2 실험 및 결과

과적합(overfitting)의 위험을 방지하기 위해 10-fold 교차 검증 기법을 사용한다. *doc2vec* 과 Count vector, *doc2vec*, *TF-IDF* 를 조합하는 방법으로 벡터화 한 문서를 *ExtraTrees*(Extremely randomized trees)로 분류를 시도했다.

제안된 방법에 의해 벡터화 했을 경우 *ExtraTrees* 로 84.7%의 분류 정확도를 보인다. *doc2vec* 만으로 벡터화 시켰을 때보다 4.8%의 분류 정확도가 향상된 것이다.

5. 결 론

본 논문에서는 *C-ISM* 유사도 측정법을 새롭게 고안하고 *doc2vec* 과 *TF-IDF* 와의 조합의 방법을 채택하여 한국어 문서 분류의 자동화에 적용 가능성을 연구하였다. 한국어 텍스트 데이터에 대해 형태소 분석한 뒤 이를 제안한 모델에 적용하여 벡터화 시켰고 *ExtraTrees* 분류기를 이용하여 최고 84.7%의 분류 정확률을 얻었다. 이는 한국어 문서 분류에 있어서는 본 연구에서 제안하는 벡터화 방법이 기존 *doc2vec* 에 비해서 전반적인 성능 면에서 더 우수함을 보인다.

향후 한국어 데이터의 효과적인 전처리 방법과 새롭게 고안된 C-ISM 의 다른 영역으로의 확대 적용 가능성에 대한 연구가 지속 필요하다.

6. 참고 문헌

[1] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information processing & management*, vol. 24, pp. 513-523, 1988.

[2] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 1188-1196.

[3] A. Heidarian and M. J. Dinneen, "A Hybrid Geometric Approach for Measuring Similarity Level Among Documents and Document Clustering," in *Big Data Computing Service and Applications (BigDataService), 2016 IEEE Second International Conference on*, 2016, pp. 142-151.

[1] https://www.open.go.kr/

요구사항 추적성 매트릭스에서 유연한 맵핑 변경을 가능하게 하는 스프레드시트 애플리케이션

정세린°[1], 이선아[1]
[1]정보과학과, 경상대학교
{rin0608, saleese}@gnu.ac.kr

An Spreadsheet Application that Enables to Flexibly Change Mappings in Requirement Traceability Matrix

Serin Jeong °[1], Seonah Lee[1]
[1]Department of Informatics, Gyeongsang National University

요 약

요구사항 추적성은 개발 과정 동안 지속적으로 관리해야 한다. 그러나, 실제에서는 추적성을 갱신하는 노력에 비해 추적성을 통해 얻는 혜택이 적을 있기 수 있기 때문에 품질 보증을 점검하는 단계에서 갱신한다. 이러한 관점에서 우리는 가장 흔히 사용하는 스프레드시트 형태의 요구사항 추적성 매트릭스를 개선해 보기로 하였다. 이를 위해 각 시트에 두 산출물 간의 관계만을 기입하여 변경이 발생할 때, 이를 즉시 반영할 수 있도록 하였다. 다음, 자동으로 모든 산출물의 관계를 계산하여 추적성을 보여준다. 또한 누락된 맵핑 관계를 표시하여 추적성 매트릭스의 완전성을 파악하도록 돕는다. 제안한 매트릭스의 효과를 파악하기 위하여 사례 연구를 수행하였다. 사례 연구는 제안한 요구사항 추적성 매트릭스가 적용가능하며 변경된 맵핑 관계를 쉽게 수용함을 보여준다.

1. 서 론

요구사항 추적성은 요구사항을 구현하는 산출물들 간에 논리적 맵핑을 기술한 것으로 [1], 개발하는 시스템에 요구사항이 얼마나 잘 반영되었는지 파악하기 위해 필요하다. 개발자들은 요구사항 추적성을 수립하기 위해 간편한 스프레드시트 형태의 요구사항 추적성 매트릭스(Requirement Traceability Matrix, 이하 RTM)를 사용한다. 전형적인 RTM은 각 산출물의 열을 만들고 산출물 항목 간의 맵핑을 행으로 배열한다. 그러나 이러한 매트릭스에는 임의의 산출물에 변경이 발생하면, 전체 맵핑 관계를 점검해야하기 때문에, 담당자는 지속적인 추적성 관리에 어려움을 겪게 된다 [1].

Hassain과 동료들은 요구사항을 관리하기 위한 스프레드시트의 활용을 조사했지만, 이를 활용하기 위한 방법은 제시하지 않았다 [2]. 또한 김과 동료들은 추적성에 변경을 반영하기 위해 새로운 ID를 부여하는 제안을 하였다. 그러나 이는 기존 추적성 방법에 변경에 따라 어떻게 요구사항 ID를 추가할 것인가의 연구이다 [3]. 이를 개선하기 위해 우리는 스프레드시트 형태의 새로운 RTM을 제안한다.

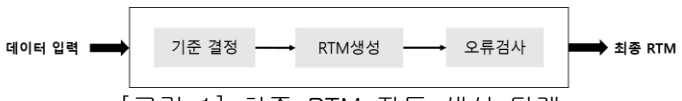
본 논문에서 제안하는 아이디어는 다음과 같다. 개별 스프레드시트에 한 쌍의 맵핑 관계를 기입하여, 변경사항을 즉시 반영하도록 한다. 이후, 이러한 맵핑 시트의 집합을 바탕으로 전형적인 추적성 매트릭스 형태를 자동 생성한다. 또한 일관되지 않은 추적성 링크를 색상으로 표시하여 사용자가 이슈를 쉽게 발견할 수 있도록 한다.

본 논문에서는 제안하는 RTM이 변경 사항에 대해 유연하게 대처할 수 있는지 확인하기 위해 사례연구를 수행하였다. 그 결과 우리가 제안한 RTM이 변경 사항을 유연하게 대처할 수 있음을 확인하였다.

본 논문의 구성은 다음과 같다. 다음 2에서는 제안 방법을 설명한다. 3절에서는 제안한 방법을 평가하기 위해 수행한 사례 연구에 대해 논의한다. 4절에서는 결론과 향후 연구 방향을 논의한다.

2. 제안 요구사항 추적성 매트릭스

우리는 각 시트에 한 쌍의 맵핑 관계를 표현하고, 이러한 맵핑 집합을 바탕으로 전체 추적성 매트릭스를 자동으로 생성하는 방법을 제안한다.

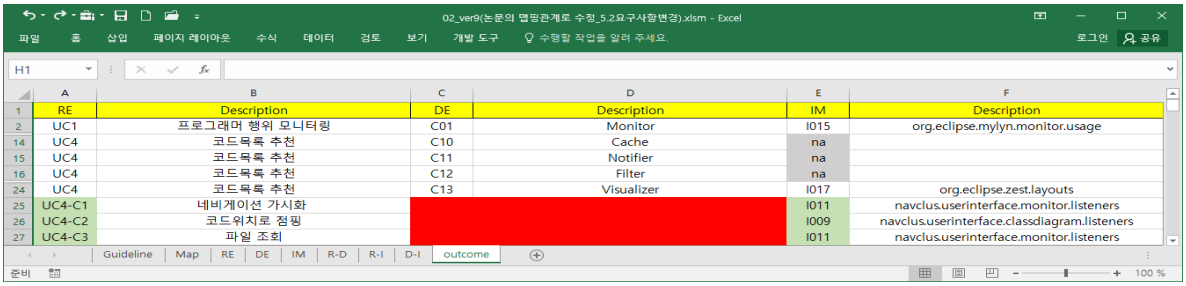


[그림 1] 최종 RTM 자동 생성 단계

2.1. 제안 단계

그림 1은 본 논문에서 제안하는 RTM을 생성하기 위한 단계이다.

1) **데이터 입력**: RTM을 생성하기 위해 다음 3가지 정보를 입력한다.



	A	B	C	D	E	F
1	RE	Description	DE	Description	IM	Description
2	UC1	프로그래머 행위 모니터링	C01	Monitor	I015	org.eclipse.mylyn.monitor.usage
14	UC4	코드목록 추천	C10	Cache	na	
15	UC4	코드목록 추천	C11	Notifier	na	
16	UC4	코드목록 추천	C12	Filter	na	
24	UC4	코드목록 추천	C13	Visualizer	I017	org.eclipse.zest.layouts
25	UC4-C1	네비게이션 가시화			I011	navclus.userinterface.monitor.listeners
26	UC4-C2	코드위치로 점핑			I009	navclus.userinterface.classdiagram.listeners
27	UC4-C3	파일 조회			I011	navclus.userinterface.monitor.listeners

[그림 2] 네브마인 추적성 매트릭스 예시

- MapSheet: 그림 2의 탭 중 탭 명이 Map인 시트로 전체 산출물 관계를 기입한다.
- ArtifactSheet: 그림 2의 탭 중 산출물의 이름을 탭 명(예: RE, DE)으로 하는 시트로 각 산출물 항목에 대한 정보를 기입한다.
- MappingSheet: 그림 2의 탭 중 맵핑 관계의 이름을 탭 명(예: R-D, D-I)으로 하는 시트로 두 개의 산출물 항목 간의 맵핑 정보를 기입한다.

2) **기준 결정**: 어떤 산출물부터 맵핑을 수행할지 기준을 결정한다.
3) **RTM생성**: 기준 산출물에 따라 각 산출물 간의 맵핑 관계를 정렬하여 전형적인 RTM을 생성한다.
4) **오류 검사**: 생성된 RTM에 대한 오류 검사를 통해 추적성 링크가 누락된 셀을 표시하여 최종 RTM의 점검한다.
5) **출력**: 개별 시트에 분할되어 있던 맵핑 관계가 정렬되어 전형적인 형태의 RTM이 생성된다.

3. 사례 연구

제안한 RTM이 기존의 전형적인 RTM보다 변경에 대해 유연하게 대처할 수 있는지를 확인하기 위하여 사례 연구를 진행하였다. 그 대상으로 코드 탐색을 돕는 오픈 소스 프로젝트인 네브마인을 선정하였다[1]. 네브마인은 개발자 활동을 바탕으로 탐색할 코드를 클래스 다이어그램으로 가시화하는 도구로, 개발 도중 가시화 기능에 대한 요구사항의 세분화가 발생하였다. 이러한 변경에 대해 새로운 RTM을 생성 후, 변경이 반영된 새로운 RTM을 생성하여 기존의 맵핑 관계의 유지 및 변경 사항의 반영이 제대로 이루어졌는지 확인한다.

변경이 발생한 요구사항은 UC4로, 이는 현재 개발자가 수행하는 작업과 관련된 태스크 코드 목록의 추천을 가시화한다. UC4는 구현단계에서 다음과 같이 세분화되었다.

- UC4-C1: 사용자가 네비게이션한 활동에 따라 방문한 코드 클래스 다이어그램을 가시화한다.
- UC4-C2: 사용자가 다이어그램에서 클래스를 더블클릭 할 때, 그 소스파일을 에디터에 보여준다.
- UC4-C3: java 파일 뿐만 아니라 xml, html, 파일 등 다른 종류의 파일도 클래스 다이어그램에 보여준다.

그림 2는 변경사항을 반영하여 생성된 최종 RTM이다. 먼저, 그림 2에서 RTM의 마지막 행에 초록색으로 표시된 셀은 요구사항이 세분화되어 새로 추가된 맵핑 항목을 나타낸다. 빨간색으로 표시된 셀은 구현과정에서 세분화가 발생하여 해당 추적성 링크에 맵핑되는 설계항목이 누락된 것을 나타낸다. 회색으로 표시된 셀은 요구사항을 구현하지 않았음을 나타낸다.

4. 결 론

우리는 두 개의 산출물 사이의 맵핑 관계에 변경이 발생 할 경우, 개발자가 다른 맵핑 관계를 고려할 필요없이 변경을 즉시 반영할 수 있는 새로운 스프레드시트 형태의 RTM을 제안한다. 제안한 RTM는 두 개의 산출물 간의 맵핑을 포함하는 다수의 시트를 가지고 있으며, 버튼을 클릭하여 전형적인 형태의 RTM을 자동 생성할 수 있다. 그리고 변경이 발생한 셀은 녹색으로 표시하고 링크가 누락된 셀은 붉은색으로 표현하여 개발자의 검증을 용이하게 하였다. 또한, 사례연구를 통해 우리의 RTM이 변경에 유연하게 대처할 수 있다는 것을 확인하였다. 그러나 하나의 엑셀 파일에 다수의 매크로를 적용하는 과정에서 도구의 로딩속도가 저하되는 문제점이 발생하였다. 이를 해결하기 위해 현재 알고리즘을 개선하여 도구를 최적화시키고 공개할 예정이다. 그리고 사용자의 피드백을 반영하여 도구를 실용화할 것이다.

5. 참고 문헌

[1] Garcia, Jorge Esparteiro, and Ana CR Paiva. "A Requirements-to-Implementation Mapping Tool for Requirements Traceability." J. of Software. 193-200. 2016.

[2] Hussain, Waqar, and Tony Clear. "Spreadsheets as collaborative technologies in global requirements change management." Global Software Engineering (ICGSE), 2014.

[3] 김주영, 류성열, 황만수. "추적테이블을 이용한 요구사항 변경관리 및 추적 효과 연구." 정보처리학회논문지. D 17.4, 271-282. 2010.

[1] https://marketplace.eclipse.org/content/navmine

C 언어 테스트 입력 생성에 대한 CAVM, Austin, CodeScroll 의 비교 및 분석

유병현 O,1, 김준휘 1, 권민혁 2, Phil McMinn3, 유신 1
한국과학기술원 1, 슈어소프트테크(주)2, University of Sheffield3
{byou, junhwi.kim23, shin.yoo}@kaist.ac.kr1, minhyuk@suresofttech.com2,
p.mcminn@sheffield.ac.uk3

Evaluation of CAVM, Austin, and CodeScroll for Test Data Generation for C

Byeonghyeon You O,1, Junhwi Kim1, Minhyuk Kwon2, Phil McMinn3, Shin Yoo1
Korea Advanced Institute of Science and Technology1, Suresoft Technologies Inc. 2,
University of Sheffield3

요 약

본 연구는 새로운 검색 기반 C 언어 테스트 데이터 생성 도구인 CAVM 을 제시한다. CAVM 은 상용화된 테스트 데이터 도구인 CodeScroll 을 향상하기 위해서 개발되었다. CodeScroll 은 정적 분석과 입력 파티셔닝 기법을 사용하여 테스트 데이터를 생성한다. 현재의 최첨단 C 언어용 자동 테스트 데이터 생성 도구인 Austin 과 다르게 CAVM 은 순수하게 검색 기반 기법 만을 사용하여 동적 자료 구조 데이터를 생성한다. 본 연구는 CAVM 을 49 개의 실험 대상 함수를 이용해 Austin 및 CodeScroll 과 비교하였다. 대상 함수는 소규모의 안티 패턴 케이스 및 실제 프로덕트 레벨의 오픈 소스 코드와 상용 코드로부터 수집하였다. 비교 결과 CAVM 은 CodeScroll 과 Austin 이 커버하지 못하는 분기들을 커버할 수 있었으며, 20 개의 대상 함수에서 가장 높은 분기 커버리지를 달성하였다.

1. 서 론

검색 기반 자동 테스트 데이터 생성은 십 여년 이상 연구자들의 관심을 받았으며 [4,12,16,17,19,21,25] 프로그램 자동 수정 [2,7,26]이나 자동 기능 이식 및 향상 [2, 10], 그리고 결함 위치 추정 [14,27] 등의 자동 분석 기술과 밀접하게 연결되어 연구되고 있다. 하지만 연구의 결과가 본래의 목적을 달성하기 위해 산업계에서 널리 쓰이고 있다고 보기는 힘들다. 또한 C 언어가 여전히 두 번째로 많이 사용되는 언어임에도 불구하고 [1], 기호 분석에 기반한 도구들 [3,24]과 달리 검색 기반 자동 테스트 데이터 생성 기법의 최근 연구는 널리 사용되는 다른 언어들을 제외한 채 Java [4-6]로 작성된 객체 지향 코드의 테스트 데이터 생성에 집중되었다.

본 연구는 C 프로그램의 검색 기반 테스트 데이터 생성을 위해 해결해야 할 기술적 과제, 특히 동적 데이터 구조로 구성된 입력의 생성에 촛점을 맞추었다. 포인터 입력으로 C 함수를 테스트하려면 올바른 “형태(shape)”의 데이터 구조를 찾아야한다. 일반적으로 Java 프로그램의 경우 메소드를 테스트하는 데 필요한 객체의 구조는 테스트

중인 코드베이스에 존재하는 생성자와 메소드 호출을 사용하여 생성 할 수 있기 때문에 이러한 문제가 존재하지 않는다. 반면 C 언어의 경우 포인터 조작을 통해 생성한 다양한 동적 메모리 형태를 테스트 데이터로 사용할 수 있으므로, C 언어를 위한 상용화 단계의 자동 테스트 데이터 생성을 위해서는 해당 문제를 해결하는 것이 매우 중요하다.

현존하는 C 언어로 작성된 함수에 대한 테스트 입력 생성도구로는 상용 툴인 CodeScroll 과 검색 기반 테스트 데이터 생성 도구인 Austin[17] 등이 있다. CodeScroll 은 동적 데이터 구조로 구성된 입력을 생성하는 간단하면서도 효과적인 휴리스틱을 사용한다. 이와 대조적으로 Austin 은 가벼운 기호 분석을 수행하여 동적 데이터 구조의 형태를 생성한 다음, AVM (Alternating Variable Method) [20]을 사용하여 구체적인 데이터를 채운다. 현재 C 언어를 대상으로 하는 최신 검색 기반 테스트 데이터 생성 도구는 Austin 이지만 더 이상 활발히 유지 관리되지 않고있다 [18].

본 연구에서는 C 에 대한 새로운 검색 기반 테스트 데이터 생성 도구 인 CAVM (“ka-boom”이라고 발음)을 제시한다. CAVM 또한 AVM 을 기반으로 하지만 Austin 과는 다르게 동적 데이터 구조의 형태를 순수히 검색 기반 방법으로 생성한다. CAVM 은 동적 데이터 구조의 적절한 형태를 키워가면서 동시에 데이터를 채우는 검색을 함께 진행한다. 또한 코드 재작성을 통해 `strcmp` 라이브러리 함수를 위한 문자열 테스트 데이터, 즉 `char` 배열 생성을 지원한다.

본 연구에서는 동적 데이터 구조를 포함하는 C 코드의 테스트 데이터 생성에 대하여 CAVM 과 CodeScroll 및 Austin 의 유효성을 비교한다. 실증적 비교를 위해 기존에 CodeScroll 이 해결하지 못하는 안티 패턴 사례 뿐만 아니라 오픈 소스 및 상용 코드를 대상으로 실험을 진행하였다. 실험결과를 통해 CAVM 에 구현된 새로운 알고리즘이 CodeScroll 이나 Austin 이 해결하지 못하는 분기를 만족할 수 있음을 확인 할 수 있다.

본 연구의 기술적인 기여 내용은 다음과 같다.

1. 동적 데이터 구조로 구성된 테스트 입력 생성을 위한 새로운 검색 기반 알고리즘 및 구현, CAVM (3 장)

2. 작은 안티 패턴에서부터 상업용 C++ 프론트엔드 모듈까지 49 가지 C 함수를 대상으로 한 실험을 통해 CAVM, CodeScroll 및 Austin 의 동적 데이터 구조를 포함한 테스트 데이터 생성 능력을 비교 (4 장)

3. 기존의 최첨단 도구인 Austin 이나 상용 도구 CodeScroll 이 해결하지 못하지만 CAVM 을 통해 해결할 수 있는 분기 유형에 대한 분석 (5 장).

먼저 CodeScroll 및 Austin 에 대한 설명을 통해 본 연구의 분석을 시작하고자 한다.

2. 배 경

이 장에서는 C 함수에 대한 단위 테스트 데이터를 생성하는 두 가지 도구를 소개한다. 첫째로, CodeScroll 은 정적 분석을 통해 입력을 도출하는 상용 도구이다. 두 번째 도구인 Austin [17] 은 숫자 입력 값을 찾기 위한 동적 검색과 동적 데이터 구조의 “형태” 정보를 찾기 위한 간략화된 기호 실행(Symbolic Execution)을 결합하여 사용한다. Austin 은 현존하는 C 코드에 대한 가장 최첨단의 검색 기반 테스트 데이터 생성 기술이다.

```c
void foo() { return 10; }

void testMe(int a, int b, int c) {
  if(a < 42)
    if(b == c)
      if(c == foo())
        // target
}
```

(a) 숫자 자료형 입력 예제

```c
typedef struct _data {
  int* internal;
  int a, b;
  struct _data* next;
} Data;

void testMe(Data *d) {
  if (d != NULL) {
    Data *d_in;
    d_in = d->next;
    if (d_in != NULL)
      if (d_in->a == 2)
        // target
  }
}
```

(b) 포인터 입력 예제

그림 1 테스트 데이터 자동 생성을 설명하는 코드 예제

2.1 CodeScroll

CodeScroll 은 Suresoft Technology Inc.에서 개발한 C 및 C ++ 용 상용 테스트 데이터 생성 도구이다. Safety critical 소프트웨어 시스템을 테스트하고 구조적 커버리지를 달성하기 위해 자동차, 방위 및 우주 항공 분야의 다양한 업계에서 CodeScroll 을 사용한다[1]. CodeScroll 은 주기적인 값 생성, 랜덤 생성 및 쌍단위 입력 파티셔닝(pairwise input partitioning)을 비롯한 여러 가지 기술을 사용한다. CodeScroll 엔지니어의 경험에 따르면 쌍단위 입력 파티셔닝이 가장 효과적인 방법이었다.

그림 1a 의 예제는 CodeScroll 의 쌍단위 입력 파티셔닝의 동작 과정과 약점을 보여준다. CodeScroll 은 경량 정적 분석을 사용하여 경계 값과 입력 파티션을 식별한다. 그림 1a 의 4 행과 같이 고정된 명확한 경계 값이 있는 분기의 경우 CodeScroll 은 변수 `a` 의 값을 효율적으로 생성 할 수 있다. 예를 들어, `a < 42` 의 조건에 대한 `a` 의 값으로 경계 값과 그 이웃 값인 42, 41, 43 과, 일정 간격 떨어진 값(42 보다 충분히 높거나 낮은 값), 자료형 파티션 (type partitions) (주어진 자료형의 최솟값, 최댓값, 중간값) 들을 시도한다.

결과적으로 CodeScroll 은 조합 상호 작용 테스트 (Combination Interaction Testing, CIT) [23]와 유사한

[1] http://www.suresofttech.com/ko/engineering/dev_vehicle 참조

방식으로 각 변수의 가능한 값들 사이의 쌍단위 상호 작용 커버리지를 달성하고자 한다. 이 프로세스는 대상 함수의 실행을 요구하지 않으며, 따라서 CodeScroll 은 실행 시간의 대부분을 정적 분석에 사용한다. 이 같은 CodeScroll 의 쌍단위 접근은 특정 분기에는 효과적이지만 약점이 있다. 예를 들어 매개 변수 (그림 1a 의 라인 5) 또는 함수 호출 (그림 1a 의 라인 6)에 의해 설정되는 경계 값은 정적 분석에 의해 확정될 수 없으며,낮은 분기 커버리지를 야기한다.

CodeScroll 은 동적 데이터 구조의 테스트 데이터를 생성하는 간단하면서도 효과적인 휴리스틱을 사용한다. 첫째, 포인터는 단일 항목을 포함하는 배열로 간주된다. 항목이 복합 자료형 (예: 구조체) 인 경우 대상 함수에서 사용되는 멤버의 값이 쌍단위 방법을 사용하여 채워진다. 둘째, 복합 자료형에 재귀 포인터 (자신의 자료형을 가리키는 포인터)가 포함된 경우 포인터는 1 의 깊이로 인스턴스화되지만 멤버는 인스턴스화하지 않는다. 예를 들어 그림 1b 의 코드에서 CodeScroll 은 매개 변수 `d` 와 `d-> next` 를 인스턴스화하지만 `d-> next-> a` 변수에는 값을 할당하지 않는다. 이는 쌍단위 조합으로 인해서 테스트 케이스의 가짓수가 지수적으로 늘어나는 것을 피하기 위한 것이다.

2.2 Austin

Austin[17] 은 C 에 대한 검색 기반 테스트 데이터 생성 도구로, 동적 기호 실행과 제약조건 풀이(Constraint Solving)를 사용한다. 수치 입력 값은 AVM 을 사용하여 찾고 포인터 입력은 포인터 등가성 그래프를 CUTE [24]와 유사한 메커니즘으로 "풀이"하여 할당된다.

Austin 은 전술한 접근 방식을 다음과 같이 사용한다. 우선 Austin 은 테스트 대상 C 함수에서 커버되지 않은 분기를 "타겟"으로 선택한다. 임의로 생성 된 초기 입력이 주어지면 Austin 의 계측기는 타겟 분기가 커버 되었는지 아니면 실행 경로가 타겟에서 벗어났는지 여부를 감지한다. 실행 경로가 타겟에서 벗어난 경우 Austin 은 대상 분기가 커버 되지 못하도록 유도한 "중요" 결정 구문(decision statement)의 조건을 평가한다. 이 중요 결정 구문은 함수의 제어 흐름 그래프(Control Flow Graph) 상에 있는 노드 중, 타겟 분기를 제어 종속 (control dependent)하는 노드이다. 구문 a 의 결과가 구문 b 의 실행 여부를 좌우할 때, 구문 b 가 구문 a 에 제어 종속된다고 한다. 그 예시로는 분기가 중첩된 "`if`"문이 있다. 그림 1b 에서 타겟 구문은 8 행, 11 행의 `False` 분기 및 12 행의 `True` 분기에 제어 종속한다.

그 다음 Austin 은 동적 기호 실행 엔진을 호출하여 경로 조건(Path Condition)을 도출한다. 경로 조건은 함수의 입력에 대한 제약 조건으로써 함수가 어느 실행 경로를 따를지를 기술한다. 경로 조건은 현재 입력 값이 따르는 중요 결정 노드까지의 실행 경로로부터 구해진다. 현재 입력값으로 함수를 실행 시켜 실행 경로를 구한 후, 중요 결정 노드에서의 조건을 반전시켜 타겟 분기에 도달하기 위한 올바른 실행 경로를 나타내는 경로 조건을 만든다.

Austin 은 경로 조건을 분석하여 함수의 숫자 자료형 입력 값을 찾기 위해서 AVM 을 호출할지, 포인터 입력을 풀기 위해 기호 실행을 할지 결정한다. 후자의 경우 Austin 은 앞서 언급한 포인터 등가성 그래프를 작성한다. 경로 조건은 포인터에 해당하는 제약 조건을 서술 하기 위해서 $x = y$ 및 $x \neq y$ 의 형태로 단순화 된다. 여기서 x, y 는 `NULL` 이나 포인터 입력에 해당하는 기호 변수(symbolic variable)다. Austin 은 경로 조건으로부터 서로 동등한 포인터를 그룹화 하여 그래프의 노드로 하고 노드 간의 비등가성을 나타내는 엣지를 추가하여 포인터 등가성 그래프를 만든다. 이 그래프는 다음의 과정을 통해 포인터 입력을 생성하기 위해 "풀이"된다. 각 노드 n 에서 `NULL` 포인터를 나타내는 노드로 향하는 엣지가 없으면 Austin 은 노드 n 의 기호 변수가 나타내는 모든 포인터를 `NULL` 로 설정한다. 만약 n 이 다른 기호 변수 s 의 주소를 나타내는 경우 Austin 은 s 가 가리키는 곳으로 구체적인 포인터 입력을 할당한다. 그렇지 않으면, Austin 은 새로운 메모리 위치를 생성하여 n 이 가리키는 구체적인 포인터 입력을 그 위치로 할당한다.

예시인 그림 1b 에서 타겟 분기는 12 행의 `True` 분기이다. 해당 분기의 조건문이 숫자로 된 제약 조건이지만 동적 자료 구조가 그 전에 올바른 모양으로 초기화 되어야한다. Austin 은 세 개의 노드로 구성된 포인터 등가성 그래프를 그리며, 그 노드는 각각 `d` (n1), `d-> next` (n2) 및 `NULL` (n3)이다. 8 행에서 `d` 가 `NULL` 이 아니라는 조건문의, `True` 분기가 실행되어야 하기 때문에 n3 과 n1 사이에는 비등가성을 나타내는 엣지가 형성된다. 기호 분석으로 10 행에서 `d-> next` 가 `d_in` 로 할당 된다는 것을 알 수 있다. `d_in` 은 `d-> next` 와 동등한 포인터이므로 n2 노드로 그룹이 된다. 기호 분석은 이어서 11 행을 `True` 로 실행한다는 것을 알아내고 이는 11 행의 `NULL` 과 `d-> next` 이 같지 않아야 하므로 각 기호 변수에 해당하는 n3 과 n2 노드 간에 엣지가 형성된다. 이 포인터 등가성 그래프를 풀면 `d` 가 새로운 메모리 위치로 초기화되고 `d-> next` 도 별도의 위치로 초기화된다. 그 이후 AVM 검색이 `d_in-> a`(즉, `d-> next-> a`)가 2 로 설정되어야 한다는 라인 12 의 조건을 해결한다.

검색 기반 테스트 데이터 생성 도구 중 C 를 대상으로 하는 가장 최신의 도구이기는 하지만, Austin 은 더 이상 활발하게 관리되지 않으며 오픈 소스 저장소의 마지막 수정은 2011 년에 이루어졌다 [18]. 또한, 제 5 장에서 기술하듯이 Austin 은 커버하지 못하는 특정 분기문 유형들이 존재한다. 이를 동기로 본 연구는 새로운 도구를 제안 및 소개한다.

3. CAVM

CAVM 은 검색 기반 소프트웨어 테스팅 기술을 활용하여 상용 수준의 도구인 CodeScroll 을 보완함으로써 어려운 분기를 보다 효과적으로 처리 하려는 목적의 산학 협력 연구의 오픈 소스 부산물이다.

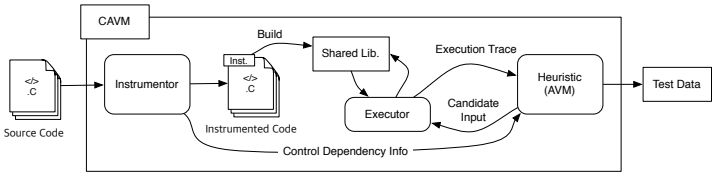


그림 2 CAVM 의 아키텍처

3.1 아키텍처

그림 2 는 CAVM 의 전체 아키텍처를 묘사한다. CAVM 은 먼저 검색에 사용될 분기 커버리지 달성을 위한 적합도(fitness)를 동적으로 측정하기 위해 주어진 소스 코드를 계량화(Instrumentation) 한다. 계량화는 Clang 을 사용하여 C 소스 코드를 직접 수정하는 방식으로 이루어진다. 원본 소스 코드를 계량화된 버전으로 바꾼 후 CAVM 은 대상 프로그램을 공유 라이브러리로 빌드한다. 공유 라이브러리를 통해 CAVM 은 엔트리 포인트를 추가하지 않고도 대상 함수를 직접 가져 오고 호출 할 수 있다. 이후 CAVM 은 타겟 분기를 커버하는 테스트 입력을 찾기 위하여 확장된 버전의 AVM 을 적용한다.

3.2 동적 자료구조에 대한 AVM 의 확장

알고리즘 1 CAVM 의 국소 탐색 알고리즘

LocalSearch($\vec{x}$, *current*)
Input: An input vector, $\vec{x}$, and the current search target, *current*
Output: The minimum fitness value found, *fitness*, and the input vector that corresponds to the value, $\vec{x}$

```
(1)   while budget > 0 and fitness > 0
(2)     e = NEXT(x, current)
(3)     if ISPRIMITIVE(e)
(4)       fitness ← ITERATIVEPATTERNSEARCH(x, e)
(5)     else if ISSTRUCT(e)
(6)       fitness ← LOCALSEARCH(x, e.members)
(7)     else if ISPOINTERTOPRIMITIVES(e)
(8)       foreach i in e
(9)         fitness ← ITERATIVEPATTERNSEARCH(x, i)
(10)      e' ← GROW(e)
(11)      x ← x(e ← e')
(12)      fitness ← EVALUATE(x)
(13)    else if ISPOINTERTOSTRUCT(e)
(14)      if e is NULL
(15)        e' ← INSTANTIATE(e)
(16)        x ← x(e ← e')
(17)        fitness ← EVALUATE(x)
(18)      else
(19)        fitness ← LOCALSEARCH(x, *e)
(20)  return x, fitness
```

알고리즘 1 은 CAVM 에서 사용하는 검색 알고리즘에 대한 의사 코드(pseudo code)이다. 간결함을 위해 의사 코드에는 적합도가 전역 최적 값에 도달 할 때마다 알고리즘이 종료하는 것을 생략하여 표기하였다. 또한 입력 벡터 x 의 모든 요소에 대하여 탐색을 마친 뒤 적합도가 향상되었는 지를 확인하는 부분도 생략하였다. 기본 원시 자료형의 경우 4 행은 반복 패턴 검색 (IPS: Iterative Pattern Search) [15,20]을 호출한다. 나머지 경우는 동적 데이터 구조를 다루는 알고리즘이다. CAVM 은 입력 벡터 x 의 모든 포인터를 `NULL` 로 초기화한다.

5 ~ 6 행은 구조체 입력의 속성 값들에 대해 재귀적으로 검색을 수행한다. 7 ~ 19 행은 포인터 입력을 처리한다. CAVM 은 기본 자료형을 가르키는 포인터를 배열로 간주한다. 7 행부터 12 행에서는 현재 배열의 각 요소에 IPS 를 적용하고 검색이 완료되지 않으면 배열의 크기를 1 만큼 증가시킨다.

마지막으로 13 ~ 19 행은 구조체에 대한 포인터를 처리한다. 현재 값이 `NULL` 인 경우 CAVM 은 메모리를 할당하는 것으로 현재 타겟 분기를 커버 될 수 있는지 여부를 확인한다 (15 행, 구체화 된 구조체의 기본 속성값들이 랜덤으로 초기화 됨). 메모리가 할당된 이후에도 검색이 성공하지 못하면 CAVM 은 구체화된 새로운 구조체의 값, 즉 현재 대상 구조체의 속성값들에 대하여 차례대로 검색을 수행한다 (18 ~ 19 행).

그림 1 의 코드를 예시로 살펴보자. CAVM 은 입력 `d` 를 `NULL` 로 초기화하여 시작한다. 이 입력은 목표 분기에 도달하지 않으므로 CAVM 은 `d` 에 메모리를 할당하게 되고 이는 8 행의 `True` 조건을 만족시킨다. 그런 다음 CAVM 은 `d-> a` 및 `d-> b` 에 IPS 를 적용하나 적합도는 향상되지 않는다. 마지막으로 CAVM 은 `d-> internal` 과 `d-> next` 에 메모리를 할당한다. 다시 IPS 를 속성값들에 적용하는 과정에서 CAVM 은 `d-> next-> a` 의 값을 2 로 만들어 목표 분기에 도달한다.

3.3 문자열 비교

C 언어에서 문자열은 `char` 의 배열로 표현되기 때문에 CAVM 은 `char` 배열을 검색하는 것으로 문자열 입력을 처리한다. CAVM 은 원시 자료형의 배열을 검색 할 수 있지만, 분기에서 `strcmp` 함수가 사용되면 그 리턴 값으로는 검색해야할 방향을 올바르게 구하기 어렵다는 문제가 추가로 발생한다. `strcmp` 는 두 개의 문자열이 동일한지, 아니면 어느 한 문자열이 다른 것보다 알파벳 순으로 더 우선하는지 여부를 나타내는 단일 정수를 리턴한다. 이 리턴 값에는 두 문자열 간의 거리 정보가 없기 때문에 CAVM 이 검색 방향을 결정하기가 어렵다.

문자열 간의 거리 정보를 제공하기 위해 CAVM 은 `strcmp` 의 래퍼 (wrapper)인 `strcmp2` 를 정의하고, 계량화 프로세스 도중에 소스 코드의 조건문이 사용하는 모든 `strcmp` 를 `strcmp2` 로 교체한다. 이것은 Fraser 와 Arcuri [4]가 JAVA 에서 `String.equals()` 메소드를 교체한 것과 동일한 접근법이다. 그러나 C 를 대상으로 하는 기존의 검색 기반 테스트 데이터 생성 도구들 중 이 아이디어를 구현한 것은 없었다.

`strcmp2` 는 부호있는 정수 대신 부호있는 문자열 거리를 반환한다. 문자열 간의 거리는 *lengthDiff* + *norm (charDiff)*로 정의된다. *lengthDiff* 는 두 입력 문자열 사이의 길이의 차이다. *charDiff* 는 두 입력 문자열 내의 문자 간의 요소 단위 거리를 누적한 것이다. 두 문자열의 길이가 다를 때는 더 짧은 문자열의 길이까지 누적한다. 정규화를 위해서는 $norm(x) = 1 - 1.001^{-x}$ 를 사용한다 [11].

4. 실험 설계

4.1 연구 질문

본 연구에서는 아래의 질문들에 대해 살펴보고자 한다.

RQ1. 유효성: CodeScroll, Austin 및 CAVM 중에서 어느 도구가 가장 높은 분기 커버리지를 달성하는가?

표 1 연구 대상 함수

Subjects	Description	Br.	*	Rec. *	`struct`	`strcmp`
AllZeros	Examples from AVMf	6	✓	-	-	-
Calendar		46	-	-	-	-
Line		14	-	-	✓	-
Triangle		16	-	-	-	-
CodeScroll Antipatterns	Set of branches that CodeScroll cannot cover	16	✓	✓	✓	✓
LinkedList	5 utility function for singley linked list	26	✓	✓	✓	-
BinaryTree	7 tree-related functions from a textbook by Horowitz et al.	30	✓	✓	✓	-
busybox-ls	5 functions from ls in Busybox 1.2.0	32	✓	-	-	-
decode.c	22 functions from decode.c	296	✓	-	✓	-
Total	49 C functions	482				

RQ2. 효율성: CodeScroll, Austin 및 CAVM 중에서 분기 커버리지를 가장 효율적으로 달성하는 도구는 무엇인가?

본 연구에서는 각각의 도구를 C 함수에 적용하여 RQ1 과 RQ2 에 대해 살펴본다. Austin 과 CAVM 은 확률적 접근 방식을 채택하기 때문에 RQ1 의 경우 커버리지와 대상 함수 실행 횟수 그리고 RQ2 의 경우 실행 시간을 총 20 번 반복한 실험의 평균으로 측정하였다. CodeScroll 은 확률적이지 않으므로 실행을 반복하지 않는다.

4.2 대상 함수

표 1 은 본 연구에서 대상으로 한 함수의 목록이다. 간단한 함수에서부터 상용 C++ 프론트엔드에 이르기까지 다양한 종류의 함수를 대상으로 실험을 진행하였다. CodeScroll Antipatterns 는 CodeScroll 이 커버하지 못하는 분기 패턴의 유형들로 구성된 예제 함수 컬렉션이다. Line, Calendar, Triangle 및 AllZeros 예제는 McMinn 과 Kapfhammer [20]의 기본 예제를 C 로 구현하였다. LinkedList 는 웹에서 발췌한 C 의 단일 연결 리스트에 대한 유틸리티 함수 구현 모음[2] 이다. BinaryTree 의 7 가지 함수들은 Horowitz 의 책 [13]에서 발췌하였다. busybox-ls 의 5 가지 함수는 busybox 패키지[3] 의 ls 유틸리티 오픈 소스 구현체 에서 선정하였다. 마지막으로, decode.c[4]에는 Edison Design Group 에서 개발 한 C++ 프론트엔드 용 demangler 모듈에서 선택된 24 개의 함수가 포함되어 있다. 총 49 개의 함수, 482 개의 분기를 대상으로 하였다.

"*"로 표시된 열은 포인터 입력이 포함 된 함수인지 여부를 나타내며 "Rec. *"는 재귀적 데이터 구조 (예 : 자신의 자료형을 다시 가리키는 포인터를 포함한 구조체)가 포함되어 있는지 여부를 표시한다. 비슷하게, `struct` 열은 입력 변수가 C 구조체(`struct`) 타입인지 여부를 보여 주며 `strcmp` 열은 입력 변수가 조건문 안에서 `strcmp` 를 사용하여 문자열과 비교되는지 여부를 나타낸다.

이 실험 대상 함수들은 임의의 C 함수를 편향되지 않게 샘플하기보다 오히려 본 연구에서 중점적으로 다루는 주제인 포인터, 배열 및 동적 데이터 구조를 입력으로 하는 C 함수들을 중심으로 선정 되었다. 실험에 사용된 모든 함수들의 소스 코드는 http://coinse.kaist.ac.kr/projects/cavm/ 에서 확인할 수 있다.

4.3 설정

CAVM 에는 대상 함수의 각 입력 변수에 대한 검색 범위를 설정하는 기능이 있지만 Austin 에는 이러한 제어 기능이 없다. 따라서 입력 범위를 좁히지 않고 각 기본 자료형의 전체 범위를 사용하여 두 도구가 같은 공간에서 검색을 수행하였다. Austin 과 CAVM 모두에 대해 최대 실행시간 5 분, 대상 함수 최대 실행 횟수 1,000 번의 제한 조건을 설정하였다. 두 도구 모두 "부수적인" 테스트 입력 [8] (즉, 현재 목표하는 것 이외의 분기를 우연히 커버하는

[2] http://milvus.tistory.com/17 의 예제에서 발췌

[3] Busybox는 작은 단일 실행파일에 유닉스 유틸리티를 모은 컬렉션이다: https://busybox.net .

[4] https://www.edg.com/c

입력[5])을 커버리지 달성으로 인정하고 결과로 수집한다. 실험에서 5 분 이내에 달성된 분기 커버리지는 최종 결과로 포함되나 도구가 5 분 제한 시간 내에 종료되지 않으면 0% 분기 커버리지를 기록하였다.

CodeScroll 은 여러가지 테스트 입력 생성 기술을 지원하지만, 본 실험에서는 2.1 장 에서 논의 된 것처럼 가장 효과적이라고 알려진 쌍단위 입력 파티셔닝 방식을 사용하였다. Austin, CAVM 과 달리 CodeScroll 은 테스트 입력을 생성하기 위해 대상 함수를 실제로 실행하지 않아도 된다 (2.1 장 참조). 또한 CodeScroll 은 GUI 를 통해서만 조작할 수 있기 때문에 다른 도구와 직접적으로 실행 시간을 비교하기는 어렵다. 따라서 CodeScroll 에 대해서는 대표적인 경우에 한하여 대략적인 실행 시간을 측정하였다.

Austin 은 C Intermediate Language (CIL) [22]에 의존하기 때문에 Austin 에서 세는 분기의 개수는 CodeScroll 및 CAVM 에서의 분기의 개수와 다를 수 있다. 예를 들어 Austin 은 복합 논리 연산자를 중첩된 `if` 문으로 나눈다. 따라서 이러한 영향을 받은 Austin 의 결과를 수동으로 조사하고 CodeScroll 및 CAVM 과 같은 분기의 개수를 가지도록 커버리지를 다시 계산하였다.

4.4 실험 환경

CAVM 은 Python 과 C/C++로 작성되었다. 대상 함수에 대한 계량화 탐침 구현은 C/C++로 작성되었으며 Clang 버전 3.9.0 및 GNU GCC 버전 4.9 이상을 요구한다. AVM 검색은 Python3 로 구현되었으며 Python 런타임 버전 3.5 이상을 요구한다. 또한 CFFI[6] 패키지를 필요로 한다.

실험 환경으로는 CAVM 은 Intel 코어 i7-6700K 4.0GHz 및 32GB RAM 이 장착된 컴퓨터에서 실행되며 운영체제는 Ubuntu 14.04 LTS 를 사용하였다. Austin 은 특정 요구사항으로 인해 동일한 사양의 Ubuntu 12.04.5 LTS 컴퓨터 상에서 실험하였다. CodeScroll 은 Intel Core i5-6600 3.9GHz 및 16GB RAM 이 있는 컴퓨터에서 Windows 7 상에서 실행하였다. 4.3 장에서 언급한대로 CodeScroll 실행 시간을 다른 도구와 직접 비교하지 않기 때문에 다른 하드웨어 환경이 문제가 되지 않는다.

5. 결 과

5.1 유효성

표 2 CodeScroll, Austin, CAVM 의 평균 커버리지 및 표준 편차. 가장 높은 커버리지는 굵게 표시, Br. 은 분기 개수, C.S.는 CodeScroll 을 의미함

Function	Br.	C.S.	Austin		CAVM	
			μ	σ	μ	σ
AVMf						
allzeros□	6	0.00	0.00	0.00	**83.33**	0.00
calendar*	46	**100.00**	0.00	0.00	0.00	0.00
line†	14	**100.00**	0.00	0.00	28.57	0.00
triangle‡	16	**93.75**	0.00	0.00	89.06	5.32
Antipatterns						
case1	4	0.00	100.00	0.00	**100.00**	0.00
case2	4	75.00	100.00	0.00	**100.00**	0.00
case3	2	50.00	100.00	0.00	**100.00**	0.00
case4§	2	0.00	0.00	0.00	**100.00**	0.00
case5	2	50.00	**100.00**	0.00	**100.00**	0.00
case6	2	50.00	**100.00**	0.00	**100.00**	0.00
LinkedList						
delete	6	**100.00**	**100.00**	0.00	60.83	23.12
insert	8	87.50	**100.00**	0.00	70.62	14.21
modify	4	75.00	**100.00**	0.00	66.25	20.32
print_list	2	**100.00**	**100.00**	0.00	**100.00**	0.00
search	6	**100.00**	0.00	0.00	**100.00**	0.00
busybox-ls						
bold	2	50.00	**100.00**	0.00	**100.00**	0.00
dnalloc	2	**100.00**	**100.00**	0.00	**100.00**	0.00
fgcolor	2	**100.00**	**100.00**	0.00	**100.00**	0.00
my_stat	10	0.00	0.00	0.00	0.00	0.00
scan_one_dir	16	**6.25**	0.00	0.00	0.00	0.00
BinaryTree						
inorder	2	**100.00**	**100.00**	0.00	**100.00**	0.00
iter_inorder	4	0.00	0.00	0.00	**100.00**	0.00
iter_search	6	**100.00**	0.00	0.00	**100.00**	0.00
level_order	8	62.50	0.00	0.00	**100.00**	0.00
postorder	2	50.00	**100.00**	0.00	**100.00**	0.00
preorder	2	50.00	**100.00**	0.00	**100.00**	0.00
search	6	**100.00**	0.00	0.00	**100.00**	0.00
decode.c						
func1	2	**100.00**	0.00	0.00	**100.00**	0.00
func2	2	**100.00**	0.00	0.00	**100.00**	0.00
func3	48	10.42	0.00	0.00	**29.90**	5.63
func4	14	21.43	0.00	0.00	**71.07**	6.34
func5	14	**21.43**	0.00	0.00	0.00	0.00
func6	16	18.75	0.00	0.00	**27.14**	9.44
func7	30	6.67	0.00	0.00	**11.56**	1.79
func8	6	50.00	0.00	0.00	**75.83**	12.65
func9	44	4.55	0.00	0.00	**69.66**	7.31
func10	28	7.14	0.00	0.00	**62.32**	10.20
func11	2	**100.00**	0.00	0.00	**100.00**	0.00
func12	4	25.00	0.00	0.00	**27.50**	7.69
func13	4	50.00	0.00	0.00	**73.75**	5.59
func14	2	50.00	0.00	0.00	**52.50**	11.18
func15	2	50.00	0.00	0.00	**97.50**	11.18
func16	12	8.33	0.00	0.00	**22.50**	18.56
func17	4	25.00	0.00	0.00	**27.50**	11.18
func18	4	50.00	0.00	0.00	**64.17**	6.11
func19	28	3.57	0.00	0.00	**8.75**	3.57
func20	8	87.50	0.00	0.00	**100.00**	0.00
func21	4	**100.00**	0.00	0.00	**100.00**	0.00
func22	18	**100.00**	0.00	0.00	**100.00**	0.00

5.1 절에서 다음의 이슈를 다룸
□: 간접적인 제어 종속성, *: 큰 검색 공간
†: 낮은 성공률, ‡: 불가능한 분기, §: `strcmp` 의 사용

표 2 는 Austin 과 CAVM 을 20 번 반복 실행하고 CodeScroll 을 단일 실행한 커버리지 결과를 나타낸다. 표에서 decode.c 의 대상 함수는 표의 공간을 절약하기 위해

[5] 부수적인 커버리지는 원래 타겟 분기 이외에도 최종 생성된 테스트 케이스가 커버하는 분기를 의미한다.

[6] C Foreign Function Interface: http://cffi.readthedocs.io

함수 이름을 간략화했다. 대상 함수의 원래 함수 이름과, 소스 코드와 커버리지의 박스 플롯은 CAVM 웹 페이지[7]에서 확인할 수 있다. Austin 과 CAVM 은 반복 실행에서 얻어진 커버리지의 평균(μ)과 표준 편차 (σ)를 표기하였으며, 각 대상 함수에 대해 가장 높은 커버리지를 굵게 표시했다. 총 49 개 함수 중 5 가지의 함수에서 CodeScroll 이 가장 높은 분기 커버리지를 달성하였으며, Austin 은 2 가지 함수에서 가장 높은 분기 커버리지를 보였다. CAVM 은 20 개의 함수에서 가장 높은 분기 커버리지를 유일하게 달성했다. 한편 Austin 은 decode.c 의 어떤 대상 함수도 5 분 제한 시간 내에 커버하지 못했다.

실험에서 커버되기 어려웠던 분기들을 수동으로 분석하여 다음과 같은 공통적인 이슈를 확인했다 (각 이슈의 기호는 표 2 에서 상호 참조 할 수 있음).

간접적인 제어 종속성 (□) : allzeros 함수의 분기 중 하나는 입력 배열 내의 값이 0 인 요소의 개수가 입력의 크기와 같아야 커버된다. 즉 입력 배열 내의 모든 요소의 값이 0 이어야 한다. CAVM 은 이 분기를 효과적으로 처리하지 못한다. 플래그 문제 [9]와 마찬가지로, 대상 분기와 제어 종속 관계가 없는 다른 분기에서 0 의 개수 카운터가 변경되기 때문에 적합도 함수가 탐색에 필요한 정보를 주지 못하며 그 결과 랜덤 재시작을 반복한다.

너무 큰 검색 공간 (*) : Calendar 함수의 `for` 루프는 큰 범위 내에서 초기화 된 인풋을 처리할 때 많은 시간을 소비한다. 반복문은 두 개의 정수 입력 값 사이를 차례로 반복하므로 반복 횟수는 최대 C 의 정수 범위까지 될 수 있다. 큰 범위를 반복하는 함수를 빈번하게 호출하는 것은 제한시간 초과 및 결과적으로 0%의 커버리지를 야기한다. 논문에 실린 결과와 별도로 입력 변수 범위가 [-100, 100]으로 설정되면 CAVM 이 일관되게 100 %의 분기 커버리지를 달성하는 것을 추가로 확인하였다.

낮은 성공률 (†) : Line 함수의 분기는 지정된 제한 시간 및 실행 횟수 제한내에서 커버되기 어려운 조건을 가진다. CAVM 이 모든 분기를 커버하는데 성공하는 경우도 있지만 평균 커버리지는 어려운 분기를 커버하지 못한 실행으로 인해 낮아졌다.

불가능한 분기 (‡) : Triangle 함수는 논리적으로 불가능한 분기를 포함한다. Triangle 함수에는 다음 함수 스니펫과 같은 형태의 구문이 있다.

```
if (a == b) {…} else {if (a == b) {…}}
```

두 번째 `if` 문의 `True` 분기는 첫 번째 `if` 문 때문에 논리적으로 실행 불가능하다. 이를 제외하면 CAVM 과 CodeScroll 은 Triangle 함수의 모든 분기를 커버한다.

`strcmp` 의 사용 (§) : Antipatterns 의 case4 는 `strcmp` 를 호출한다. CodeScroll 이나 Austin 을 `strcmp` 의 호출 결과를 포함하는 조건문을 지원하지 않는다.

마지막으로 Austin 은 다음과 같은 제약사항을 보인다. Austin 이 포인터를 인스턴스화하기 위해서는 대상 함수의 소스 코드 내에 포인터 `NULL` 체크 조건이 명시적으로 쓰여있어야 한다. 소스 코드 내에서 주어진 포인터와 `NULL` 을 명시적으로 비교한 코드가 없으면 기호 실행 단계에서 포인터가 인스턴스화되지 않는다. 이러한 동작이 정상적인 것임을 Austin 의 주 개발자에게 확인한 후 작은 벤치마크(Antipatterns, AVMf, LinkedList 및 BinaryTree)에는 포인터 `NULL` 검사를 명시적으로 삽입한 후 실험하였으나 실제 오픈 소스 및 상용 대상 함수, 즉 ls 및 decode.c 는 수정하지 않았다. 결과적으로 decode.c 의 함수들은 모두 포인터 파라미터를 요구하기 때문에 decode.c 의 모든 대상 함수에서 Austin 은 0% 커버리지를 달성하였다.

표 2 의 결과에 따라 RQ1 에는 다음과 같은 답을 할 수 있다: CAVM 은 CodeScroll 이나 Austin 이 커버하지 못하는 분기를 커버할 수 있으며, 특히 Austin 은 포인터 입력을 인스턴스화 하는 데에 있어 중요한 한계를 가지고 있다.

5.2 효율성

표 3 Austin, CAVM 을 20 번 동작한 평균 실행 시간 및 표준 편차(단위: 초): 가장 짧은 시간은 굵게 표시, 제한 시간 초과는 - 로 표시

Function	Austin		CAVM	
	μ	σ	μ	σ
AVMf				
allzeros□	-	-	**3.62**	0.34
calendar*	-	-	-	6.66
line†	-	-	**43.75**	1.16
triangle‡	-	-	**12.04**	2.28
Antipatterns				
case1	2.01	1.20	**1.30**	0.58
case2	1.85	0.20	**0.43**	0.02
case3	0.76	0.11	**0.74**	0.18
case4	-	-	**0.75**	0.11
case5	**0.72**	0.19	0.79	0.19
case6	**0.99**	0.17	1.18	0.02
LinkedList				
delete	**2.44**	0.79	15.73	15.35
insert	**2.55**	0.87	15.09	13.67
modify	**1.81**	0.26	9.21	0.00
print_list	**0.15**	0.01	0.30	0.14
search	-	-	**150.72**	150.22
busybox-ls				
bold	0.88	0.89	**0.31**	0.15
dnalloc	**32.31**	17.54	62.82	37.86
fgcolor	1.0.1	0.75	**0.36**	0.22
my_stat	-	-	-	-
scan_one_dir	-	-	-	-
BinaryTree				
inorder	**0.14**	0.02	0.19	0.01
iter_inorder	-	-	**71.08**	0.34
iter_search	-	-	**0.86**	0.22
level_order	-	-	**143.82**	0.49
postorder	**0.13**	0.01	0.19	0.00
preorder	**0.14**	0.01	0.19	0.00
search	-	-	**0.83**	0.22
decode.c				
func1	-	-	**0.98**	0.02
func2	-	-	**0.59**	0.01
func3	-	-	**196.48**	17.38
func4	-	-	**16.61**	3.01
func5	-	-	-	-
func6	-	-	**41.46**	5.69

[7] http://coinse.kaist.ac.kr/projects/cavm/

func7	-	-	**150.08**	4.83
func8	-	-	**8.18**	1.31
func9	-	-	**77.69**	15.59
func10	-	-	**54.38**	10.44
func11	-	-	**0.67**	0.06
func12	-	-	**11.21**	1.38
func13	-	-	**5.68**	0.97
func14	-	-	**3.82**	0.81
func15	-	-	**2.05**	0.90
func16	-	-	**37.74**	9.25
func17	-	-	**10.98**	1.66
func18	-	-	**9.99**	1.21
func19	-	-	**123.74**	6.86
func20	-	-	**1.51**	0.06
func21	-	-	**0.73**	0.01
func22	-	-	**12.07**	0.71

5.2 절에서 다음의 이슈를 다룸
□: 간접적인 제어 종속성, *: 큰 검색 공간
†: 낮은 성공률, ‡: 불가능한 분기,

표 3 은 대상 함수에 대한 테스트 데이터를 생성하기 위해 Austin 및 CAVM 이 소모하는 실행 시간을 나타낸다. 대시 (-) 기호는 20 회의 실행이 모두 제한 시간인 300 초 내에 종료되지 않은 완전한 시간 초과를 나타낸다. 두 도구가 모두 제한 시간을 초과하는 경우를 제외하고, 실행 시간이 더 짧은 도구의 평균 실행 시간 값을 굵게 표시하였다. 5.1 절의 분기 커버리지에 영향을 주는 대부분의 요인이 실행 시간에도 영향을 미친다.

간접적인 제어 종속성 (□), 낮은 성공률 (†), 불가능한 분기 (‡) : 이 요인들에 의하여 발생한 반복된 랜덤 재실행 및 제한 시간 초과는 분기 커버리지에는 기여하는 바 없이 실행 시간만을 소모한다.

큰 검색 공간 (*) : 유효성 연구 (5.1 절)와 마찬가지로 입력 변수의 범위가 축소되면 CAVM 의 평균 실행 시간은 약 7 초로 떨어지고 분기 커버리지는 100 %로 유지된다.

비교 결과, decode.c 의 22 개 함수 모두에 대해 CodeScroll 을 실행하면 일반적으로 정적 분석에 5 초, 쌍단위 테스트 데이터 생성에 3 초가 소요된다. 비슷하게 LinkedList 와 BinaryTree 의 함수에서는 각 단계에서 5 초와 2 초가 소요된다.

따라서 표 3 의 결과를 바탕으로 RQ2 에 다음과 같이 답할 수 있다: Austin 과 CAVM 모두 CodeScroll 보다 오래 걸리며 최악의 경우 자릿수 단위의 실행 시간 차이가 있을 수 있다. CodeScroll 은 속도가 빠른 대신 높은 커버리지를 달성하지 못한다. CAVM 은 실생 시간이 오래 걸리지만 더 많은 분기를 커버한다. 마지막으로 Austin 은 CAVM 과 동일한 AVM 을 사용함에도 불구하고 decode.c 파일 내의 함수를 커버할 수 없었으며, 실행 시간 역시 CodeScroll 보다 더 오래 걸릴 수 있다.

6. 고 찰

본 연구를 통하여 향후 테스트 데이터 생성 기법의 개발 방향에 대해 다음과 같은 고찰을 할 수 있었다.

하이브리드화를 통한 이득: CodeScroll 이 사용하는 휴리스틱은 매우 간단하지만 특정 패턴의 분기에는 놀라울 정도로 효과적으로 동작한다. 따라서 이 간단한 휴리스틱을 AVM 같은 보다 자원이 많이 드는 기술과 하이브리드화할 경우 다양한 가능성이 있다고 예상한다. CAVM 의 향후 CodeScroll 통합은 CAVM 이 CodeScroll 이 커버하지 못한 분기에만 동작하는 계단식 모델을 따를 예정이다.

소스 코드를 직접 계량화의 적합성: CAVM 은 의도적으로 개량화 도구로 Clang 을 사용했다. 이는 Clang 이 널리 사용되는 컴파일러의 프론트엔드가 될 만큼 강력하기 때문이다. CAVM 과 비교하였을 때 Austin 이 요구하는 CIL 변환은 종종 빌드 관련 문제를 야기하여 사용성을 저하시킨다.

동적 자료 구조의 순수 검색 기반 처리: CAVM 은 검색 기반으로 동적 자료구조를 인스턴스화 할 수 있다. CAVM 이 현재 채택하고있는 성장 및 검색 접근법이 완전하지는 않지만 (예: 이중 링크드 리스트의 역방향 포인터는 키워서는 안됨), 본 논문은 순수한 검색 기반 기술이 가능함을 보인다.

7. 연구 타당성에 대한 분석

내적 타당도는 사용된 도구의 정확성을 비롯하여 연구의 결과가 주장을 얼마나 지지하는지의 정도를 나타낸다. 소스 코드 계량화에 사용한 Clang 은 널리 사용되는 LLVM 기반의 C 컴파일러 프론트엔드이며 광범위하고 공개적인 정밀 조사가 이루어지고 있다. CodeScroll 은 상용 도구이며 Austin 은 동료 검토 결과를 기반으로 하는 공개 소스 도구이다. CAVM 자체는 추가 검사를 위해 공개 소스로 공개될 예정이다.

외적 타당성 저해 요인으로는 본 연구의 결론을 일반화하는 데에 제약을 미칠 수 있는 요소가 포함된다. 본 논문에서 도출한 결론은 실험을 위해 선택한 대상 프로그램 및 함수의 특성에 귀속되며, 더 넓은 일반화는 광범위한 실증 연구를 통해서만 가능하다. 하지만 현재의 연구는 포인터와 동적 데이터 구조와 같이 C 언어에 특화된 특징을 서로 다른 도구가 어떻게 다루는지를 비교하는 데에 있어 충분한 증거를 제공한다.

8. 결 론

본 연구는 순수한 검색 기반 접근 방식을 사용하여 동적 자료 구조를 처리하는 AVM 기반 테스트 데이터 생성 도구 CAVM 을 제시한다. 기호 분석을 사용하여 필요한 자료 구조의 모양을 결정하는 현재의 최첨단 도구인 Austin 과는 달리, CAVM 은 포인터를 연쇄적으로 인스턴스화하며 동적 데이터 구조를 키워나간다. 본 연구는 CAVM 을 Austin 과 상용 테스트 데이터 생성 도구인 CodeScroll 에 비교하는 실험을 수행하였으며, 실증적 비교의 결과 CAVM 이 다른 도구들이 다루지 못하는 여러 분기를 커버 할 수 있음을 보였다. 본 연구의 향후 작업으로는 CAVM 내부의 제어 의존성 분석 모듈의 개선 및 AVM 알고리즘의 성능 개선 등이 계획되어 있다.

Acknowledgement. 이 논문은 2017 년도 정부 (과학기술정보통신부)의 재원으로 정보통신기술 진흥센터의 지원을 받아 수행된 연구임 (No. R7117-16-0005:

고신뢰 고위험(의료,항공,차량) 소프트웨어 시험 검증을 위한 연동형 클라우드 플랫폼)

참고문헌

[1] TIOBE Software: Tiobe Index. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[2] Barr, E.T., Harman, M., Jia, Y., Marginean, A., Petke, J.: Automated software transplantation. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis. pp. 257–269. ISSTA 2015, ACM, New York, NY, USA (2015)

[3] Cadar, C., Dunbar, D., Engler, D.: KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation. pp. 209–224. OSDI'08, USENIX Association, Berkeley, CA, USA (2008)

[4] Fraser, G., Arcuri, A.: Whole test suite generation. IEEE Trans. Softw. Eng. 39(2), 276–291 (Feb 2013)

[5] Fraser, G., Arcuri, A., McMinn, P.: A memetic algorithm for whole test suite generation. Journal of Systems and Software 103, 311 – 327 (2015)

[6] Fraser, G., Staats, M., McMinn, P., Arcuri, A., Padberg, F.: Does automated white-box test generation really help software testers? In: International Symposium on Software Testing and Analysis, ISSTA '13, Lugano, Switzerland, July 15-20, 2013. pp. 291–301 (2013)

[7] Goues, C.L., Dewey-Vogt, M., Forrest, S., Weimer, W.: A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In: Proceedings of the 34th International Conference on Software Engineering. pp. 3–13 (2012)

[8] Harman, M., Kim, S.G., Lakhotia, K., McMinn, P., Yoo, S.: Optimizing for the number of tests generated in search based test data generation with an application to the oracle cost problem. In: Proceedings of the 3rd International Workshop on Search-Based Software Testing (SBST 2010). pp. 182 –191 (apr 2010)

[9] Harman, M., Hu, L., Hierons, R., Wegener, J., Sthamer, H., Baresel, A., Roper, M.: Testability transformation. IEEE Transactions on Software Engineering 30(1), 3–16 (Jan 2004)

[10] Harman, M., Langdon, W.B., Jia, Y., White, D.R., Arcuri, A., Clark, J.A.: The gismoe challenge: Constructing the pareto program surface using genetic programming to find better programs (keynote paper). In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. pp. 1–14. ASE 2012 (2012)

[11] Harman, M., McMinn, P.: A theoretical and empirical study of search based testing: Local, global and hybrid search. IEEE Transactions on Software Engineering 36(2), 226–247 (2010)

[12] Harman, M., Wegener, J.: Evolutionary testing. In: Genetic and Evolutionary Computation (GECCO). Chicago (Jul 2003)

[13] Horowitz, E., Sahni, S., Anderson-Freed, S.: Fundamentals of Data Structures in C. W. H. Freeman & Co., New York, NY, USA (1992)

[14] Jin, W., Orso, A.: F3: Fault localization for field failures. In: Proceedings of the 2013 International Symposium on Software Testing and Analysis. pp. 213–223. ISSTA 2013, ACM, New York, NY, USA (2013)

[15] Kempka, J., McMinn, P., Sudholt, D.: Design and analysis of different alternating variable searches for search-based software testing. Theoretical Computer Science 605, 1–20 (2015)

[16] Korel, B.: Automated software test data generation. IEEE Transactions on Software Engineering 16(8), 870–879 (1990)

[17] Lakhotia, K., Harman, M., Gross, H.: AUSTIN: A tool for search based software testing for the c language and its evaluation on deployed automotive systems. In: 2nd International Symposium on Search Based Software Engineering. pp. 101–110 (Sept 2010)

[18] Lakhotia, K.: Open source code of Austin, https://github.com/kiranlak/austin-sbst, (Accessed 12/4/2017)

[19] McMinn, P.: IGUANA: Input generation using automated novel algo- rithms. A plug and play research tool. Tech. Rep. CS-07-14, Department of Computer Science, University of Sheffield (2007)

[20] McMinn, P., Kapfhammer, G.M.: AVMf: An open-source framework and implementation of the alternating variable method. In: International Symposium on Search-Based Software Engineering (SSBSE 2016). Lecture Notes in Computer Science, vol. 9962, pp. 259–266. Springer (2016), code and examples available at: http://avmframework.org

[21] Miller, W., Spooner, D.L.: Automatic generation of floating-point test data. IEEE Transactions on Software Engineering 2(3), 223–226 (1976)

[22] Necula, G.C., McPeak, S., Rahul, S.P., Weimer, W.: CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs, pp. 213–228. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)

[23] Petke, J., Cohen, M.B., Harman, M., Yoo, S.: Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection. IEEE Transactions on Software Engineering 41(9), 901–924 (September 2015)

[24] Sen, K., Marinov, D., Agha, G.: CUTE: A concolic unit testing engine for C. In: Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 263–272. ESEC/FSE-13, ACM (2005)

[25] Tonella, P.: Evolutionary testing of classes. In: Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 119–128. ISSTA '04, ACM, New York, NY, USA (2004)

[26] Weimer, W., Nguyen, T., Goues, C.L., Forrest, S.: Automatically finding patches using genetic programming. In: Proceedings of the 31st IEEE International Conference on Software Engineering (ICSE '09). pp. 364–374 (May 2009)

[27] Yoo, S.: Evolving human competitive spectra-based fault localisation techniques. In: Fraser, G., Teixeira de Souza, J. (eds.) Search Based Software Engineering, Lecture Notes in Computer Science, vol. 7515, pp. 244–258. Springer Berlin Heidelberg (2012)

비행 테스트 프로파일을 이용한 드론 소프트웨어 테스트 자동화 기법

김덕엽 ○, 이성희, 이우진

경북대학교 컴퓨터학부, 소프트웨어기술연구소

ejrduq77@naver.com, lee3229910@gmail.com, woojin@knu.ac.kr

Test Automation Technique of Drone Software using Flight Test Profile

Deok Yeop Kim○, Sunghee Lee, Woo Jin Lee

School of CSE & SWRC, Kyungpook National University

요 약

최근 드론의 활용이 많아지면서 관련 사고들도 증가하고 있다. 이렇게 발생하는 사고들 중 대부분이 드론의 기술적 결함으로 발생한다. 드론 사고는 사람의 안전과 직결되는 경우가 많기 때문에 드론의 안전성 검증을 위해 충분한 테스트가 이루어져야 한다. 기존 테스트 방법들은 테스트 비용이 크거나 제한적인 경우가 많고 자동화하기가 쉽지 않다. 따라서 본 논문에서는 기존 테스트 방법들의 문제를 해결하고 드론 소프트웨어의 결함도 확인할 수 있는 비행 테스트 프로파일을 이용한 방법을 제안한다. 드론 테스트 일련의 과정을 순차적인 프로토콜 명령 모음인 비행 테스트 프로파일로 관리하고 이를 이용하여 테스트 과정을 자동화한다. 또한 드론 소프트웨어의 설정들을 기준으로 하여 소프트웨어의 결함을 발견하거나 결함의 원인을 파악할 수 있다.

1. 서 론

드론이 최근 소형화되고 다목적으로 활용되면서 임무수행에 필요한 기술 연구도 활발히 이루어지고 있다[1-2]. 예를 들어 자동이착륙이나 위성항법장치를 이용한 경로비행이나 이륙지점으로의 복귀, 카메라를 이용한 촬영 및 감시 등의 기술들이다. 그러나 기술의 개발에만 몰두하여 관련법이 미비하고 드론에 대한 기술적, 제도적 안전장치가 제대로 마련되어 있지 않다[3-4]. 이런 상황에서 드론 활용이 많아지면서 관련 사고도 늘어나고 있다[5].

드론 관련 사고들이 이슈가 되면서 최근 드론 안전성에 대한 관심이 높아지고 있으며 이미 드론 비행 사고들에 대한 원인도 연구되고 있다[6-7]. 사고 원인에 대한 연구결과들은 실제로 드론 사고가 빈번하게 발생하고 있으며 드론의 기술적 완성도가 아직 그렇게 높지 않음을 보여준다. 드론의 기술적 완성도가 아직 높지 않기 때문에 드론의 안전성 검증을 위해서는 충분한 테스트를 통해 남아있는 결함을 찾아서 해결해야 한다. 그러나 현재 드론은 테스트를 위한 방법이나 시스템이 제대로 구축되어 있지 않으며 테스트도 소극적으로 이루어진다[8]. 드론 테스트를 위한 방법들이 까다롭고 비효율적이거나 제한적이기 때문이다[9-10]. 따라서 테스트의 자동화나 재사용성을 높여 테스트를 충분히 수행할 수 있는 방법이 필요하다.

본 논문에서는 기존 테스트 방법의 문제를 해결하고 결함도 찾을 수 있는 비행 테스트 프로파일을 이용한 테스트 방법을 제안한다. 제안하는 방법은 드론 테스트 일련의 과정을 순차적인 프로토콜 명령 모음인 비행 테스트 프로파일로 관리하여 테스트를 자동화 할 수 있다. 그리고 프로토콜 명령을 사용하기 때문에 다른 테스트나 소프트웨어에 재사용 가능하다. 또 드론 소프트웨어에서 사용하는 설정들을 기준으로 테스트 시나리오와 프로파일을 생성하여 테스트 중 발견되는 결함의 원인을 파악하는데 도움이 된다. 제안하는 방법의 유용성은 테스트 데모 및 기존 테스트 방법들과의 비교를 통해 보인다.

2. 관련연구

2.1 전용 설정 어플리케이션 기반 테스트

일반적으로 드론 소프트웨어들은 내부적으로 수많은 설정들을 가지고 있다. 그래서 사용자가 원하는 드론의 기능들을 테스트 하기 위해서는 해당하는 설정들에 맞게 변경해야 한다. 이러한 설정들의 변경을 위해서 소프트웨어에서는 프로토콜 명령을 이용하며, 드론과 연결하여 프로토콜 명령으로 여러 소프트웨어 설정들을 변경해주는 어플리케이션을 소프트웨어 전용 설정 어플리케이션이라고 한다[11]. 설정 변경뿐만 아니라 비행 정보를 받아와 표시해주기도 하지만 비행 제어는 지원해주지 않기 때문에 드론의 제어는 조종기나 조이스틱 등 다른 방법으로 수행해야 한다. 그래서 사용자가 어플리케이션을 이용하여 테스트를 하는 경우

설정이나 모드를 변경하려고 할 때마다 드론과 어플리케이션을 연결해야 한다. 또 설정 변경이 완료된 후 조종기와 드론을 연결하여 테스트 시나리오에 맞게 비행 제어와 기능 활성화를 모두 사용자가 직접 수행해야 한다. 즉 테스트를 위한 절차가 까다롭고 사용자가 테스트 시나리오에 맞게 동일한 제어를 반복적으로 직접 수행하므로 테스트 비용이 크다.

2.2 드론 로깅 정보 분석 기반 테스트

드론 비행 중 기록하고자 하는 정보들을 저장장치에 지속적으로 로깅하여 비행이 끝난 후 로깅된 정보를 분석해주는 도구를 이용하여 테스트하는 방법이다[12]. 드론 소프트웨어에 있는 로깅 기능을 이용하며 로깅 주기가 매우 짧아 사용자가 원하는 정보들의 변화를 파악할 수 있다. 실제 비행 테스트에서 특정 정보들의 짧은 시간 동안의 변화를 알 수 있기 때문에 데이터나 상태의 변화를 파악하여 결함을 찾는데 유용하다. 그러나 일반적으로 로깅 및 분석 가능한 정보들이 드론 소프트웨어 별로 사전에 정의되어 있으며 사용자가 테스트를 위해 원하는 정보를 로깅하려면 소스코드뿐만 아니라 분석도구의 수정도 필요하다. 또 로깅하는 정보가 많을수록 로깅에 소요되는 시간이 늘어나서 로깅 주기를 늘리거나 로깅하는 정보를 줄여야 하는 한계가 존재한다. 이 방법도 테스트는 처음부터 끝까지 사용자가 직접 수행해야 하며 비행 후 분석을 위한 추가 비용이 들기 때문에 테스트 비용은 더 커진다.

2.3 드론 소프트웨어의 단위 테스트

드론 소프트웨어의 단위 테스트는 드론의 전체 시스템에 대해서 테스트를 하는 것이 아닌 소프트웨어 내부의 일부 기능에 대해서만 테스트를 수행하는 것이다. 일반적으로 새로운 소스코드의 추가나 기존 소스코드의 변경 후 기능이 정상적으로 동작하는지 확인하기 위한 기능 테스트에 사용한다. 테스트가 실제 비행 플랫폼이 아닌 테스트 플랫폼에서도 가능해서 개발 중에 많이 사용하는 테스트 방법이다[13]. 단위 테스트는 모든 드론 소프트웨어에서 지원해주는 것이 아니며 일부 소프트웨어만 지원해주거나 사용자가 직접 테스트 환경을 갖추어야 한다. 단위 테스트는 주로 기능 수행 후 예상되는 값과 실제 결과 값을 비교하거나 상태를 확인하는 방식으로 이루어진다. 일반적으로 구글 테스트 프레임워크를 사용하며 사전 설정과 특정 상황을 만족시킨 후 원하는 정보의 값이나 상태를 확인한다[14]. 테스트를 수행하기 위해서는 테스트에 맞는 소스코드를 작성해야 하며 한번 만들어둔 테스트 코드로 테스트를 자동화 할 수 있다. 그러나 단위 테스트는 개발 단계에서 단일 기능에 대한 검증이 주 목적이기 때문에 다른 기능과의 간섭이나 영향이 고려되지 않는다. 또 실제 비행 플랫폼이 아닌 환경에서 테스트가 가능하기 때문에 실제 비행 상황에서는 테스트 할 때와는 다른 결과가 발생할 수 있다.

2.4 드론 비행 시뮬레이터 기반 테스트

드론 비행 시뮬레이터는 드론에서 제공하는 각종 기능들의 사용법이나 성능 테스트를 위해 사용하는 시뮬레이터를 의미한다[15]. 조종기나 조이스틱과 연결하여 실제 드론을 조종하는 것처럼 비행 시뮬레이션을 하는 것이 가능하다. 시뮬레이터에서 3D 시뮬레이션이나 위성 2D 지도 등을 지원하여 드론의 움직임 방식이나 이해하기 힘든 비행 모드 및 기능들에 대해서 사전 테스트를 해볼 수 있다. 그리고 테스트를 통해 기능들의 기술적 결함도 파악할 수 있다. 그러나 시뮬레이터는 소프트웨어 단위 테스트와 마찬가지로 어떤 입력이나 사전 상황이 만족됐을 때의 이론적 결과 값들을 바탕으로 구현되었기 때문에 실제 비행 테스트에서는 다른 결과가 나올 수 있다. 또 시뮬레이터에서 지원하는 기능들만 테스트할 수 있기 때문에 테스트가 제한적인 문제가 있다.

2.5 드론 테스트 자동화와 재사용의 필요성

설정 어플리케이션과 로깅 분석도구를 이용한 방법은 실제 비행 플랫폼에서의 테스트로 기능 테스트 및 결함 확인에 적합하다. 그러나 테스트를 위한 사전 설정과 테스트 전체 과정을 사용자가 직접 수행해야 하기 때문에 테스트 비용이 크다. 따라서 테스트가 충분히 이루어지기 어렵다. 단위 테스트 방법은 한번 작성한 테스트 코드로 테스트의 자동화가 가능하지만 테스트가 주로 개발 중에 개발 플랫폼이나 테스트 플랫폼에서 이루어진다. 드론 시뮬레이터도 단위 테스트와 마찬가지로 외부요인들을 배제하고 이론적 데이터로만 테스트가 수행되어 드론 테스트에 적합하지 않다. 표 1은 기존 드론 테스트 방법들의 테스트 플랫폼과 수행 방식 및 재사용성을 나타낸 것이다.

표 1. 기존 드론 테스트 방법들 비교

기법	플랫폼	테스트 자동화	재사용성
어플리케이션	Real	Manual	Low
로깅 정보 분석	Real	Manual	Low
단위 테스트	Test	Manual/Auto	Med
시뮬레이터	Test	Manual	Low

기존 테스트 방법들의 문제를 해결하여 실제 비행 플랫폼에서 테스트를 효율적으로 수행할 수 있는 방안이 필요하며 이것은 테스트 자동화를 통해 해결이 가능하다. 또 다른 테스트나 소프트웨어에도 쉽게 적용할 수 있도록 재사용성을 높여야 한다. 본 논문에서는 테스트 자동화와 재사용을 높이는 방법으로 비행 테스트 프로파일을 이용한 방법을 제안한다.

3. 비행 테스트 프로파일을 이용한 드론 테스트

3.1 비행 테스트 프로파일

현재 대부분의 드론 소프트웨어들은 필요한 정보들을 주고받기 위해 전용 프로토콜을 사용한다. 드론의 제어뿐만 아니라 드론으로부터 비행 정보를 받아오거나 드론의 설정을 변경하기 위해서는 프로토콜 명령을 이용해야 한다. 실제로 드론 소프트웨어 별로 지원해주는 설정 어플리케이션들도 프로토콜 명령을 통해 각종 설정들을 변경하고 드론으로부터 비행 정보들을 받아와 표시해준다. 표 2는 드론 소프트웨어 별로 사용하는 프로토콜을 나타낸다.

표 2. 드론 소프트웨어 프로토콜

드론 소프트웨어	프로토콜
Baseflight	MSP
Cleanflight	MSP / Mavlink
Betaflight	MSP / Mavlink
Autopilot	Mavlink / Syslink
Ardupilot	Mavlink / Syslink

드론의 테스트는 시나리오에 맞게 연속된 프로토콜 명령들을 보내 특정 기능들을 수행시키는 것이라고 할 수 있다. 따라서 본 논문에서는 드론에 보내는 순차적인 프로토콜 명령들의 모음을 비행 테스트 프로파일로 정의한다. 테스트를 위한 사전 설정이나 드론의 제어에는 명령들간의 순서가 중요하기 때문에 테스트 시나리오를 구성하는 순차적인 명령들을 비행 테스트 프로파일로 관리하는 것이다. 테스트 일련의 과정을 테스트 프로파일로 관리하면 한번 생성한 비행 테스트 프로파일을 재사용하여 테스트의 자동화가 가능하다. 또 프로토콜 명령을 이용하기 때문에 동일한 프로토콜을 사용하는 다른 드론 소프트웨어나 기체에 간단하게 적용하여 테스트할 수 있다. 그림 1은 비행 테스트 프로파일을 이용한 시나리오 테스트를 나타낸다. 테스트 시나리오의 여러 테스트 케이스들의 테스트 단계를 구성하는 순차적 명령들을 하나 또는 여러 비행 테스트 프로파일로 관리한다.

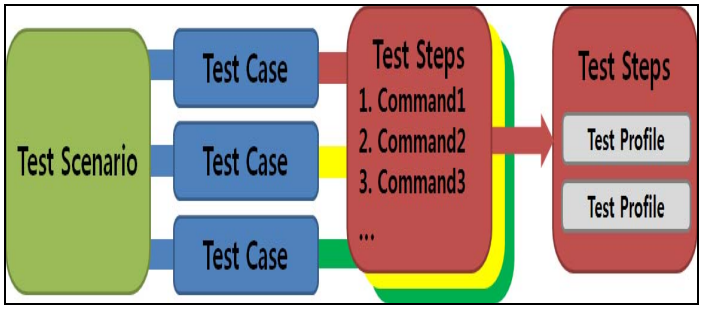


그림 1. 테스트 프로파일을 이용한 시나리오 테스트

3.2 테스트 시나리오 생성 기준

드론의 테스트는 시나리오 테스트 방법으로 수행하기 때문에 테스트 시나리오의 생성 기준이 중요하다. 드론 테스트의 목적이 기능의 정상 동작을 확인하기 위한 것도 있지만 소프트웨어 내의 결함을 찾아내는 것도 중요하다. 따라서 테스트 시나리오를 생성하기 위해서는 소프트웨어 내부에서 사용하는 각종 설정들과 기능들을 고려할 필요가 있다. 그러나 드론 소프트웨어에서 사용하는 설정과 기능들은 보통 수십 개에서 수백 개씩 존재한다. 이 모든 설정들과 기능들을 고려하여 시나리오를 생성하면 경우의 수가 너무 많아지고 의미 없는 테스트들을 수행할 가능성도 높아지므로 수많은 설정들 중 다른 기능이나 드론의 동작에 영향을 끼칠 수 있는 것들을 선별해야 한다. 표 3은 드론 오픈소스 소프트웨어인 베이스 플라이트에서 사용하는 설정들 중 다른 기능이나 드론의 동작에 영향을 끼칠만한 것들을 선별한 것이다.

표 3. 테스트 시나리오 생성 기준

설 정	설 명
Battery monitor	Battery voltage monitoring
Inflight calibration	Calibration during flight
Don't spin motors	Don't spin motors when armed
Failsafe setting	Failsafe on signal loss
Oneshot	Synchronize PWM with main loop
기 능	**설 명**
Acc calibration	Acceleration sensor calibration
Arm / Disarm	Start / Stop the drone flight
Horzion mode	Drone flight mode
Altitude hold	Holding the height of the drone
Head free	Fixed the direction at the begin

선별한 설정과 기능들은 드론의 기본적인 비행 제어를 위해 필요한 것들이나 상황에 따라 다른 기능이나 드론 동작에 간섭하거나 영향을 줄 가능성이 있다고 의심할 수 있는 것들이다. 이러한 생성 기준을 가지고 테스트 시나리오를 생성하여 테스트를 수행하면 테스트 중 문제가 발생할 경우 시나리오와 테스트 케이스의 설정과 기능 정보를 바탕으로 소프트웨어의 특정 설정이나 기능의 잔존 결함을 파악할 수 있다.

3.3 비행 테스트 프로파일 생성

드론 소프트웨어에서 사용하는 프로토콜은 헤더 부분과 데이터 부분으로 나뉘며 헤더 부분의 프로토콜 명령 아이디를 통해 다양한 프로토콜 명령들이 구분된다. 그림 2는 드론 소프트웨어들에서 사용하는 프로토콜 중 하나인 MSP의 구조를 나타낸다. MSP는

프로토콜 인식을 위한 Preamble, 프로토콜 명령 전달 방향을 의미하는 Direction, 데이터의 크기를 나타내는 Size, 프로토콜 명령 구분을 위한 Command, 마지막으로 프로토콜 명령의 데이터가 정상적인지 확인하기 위한 Checksum으로 구성된다. 모두 고정된 크기를 가지고 있으며 데이터 크기도 프로토콜 명령에 따라 고정되어 있다.

Preamble	Direction	Size	Command	Data	Checksum
Preamble	36($), 77(M)				2byte
Direction	60(<) or 62(>) [60 : -> FC], [62 : FC ->]				1byte
Size	Number of data bytes				1byte
Command	Protocol message id				1byte
Data	Data value				nbyte
Checksum	Checksum (XOR)				1byte

그림 2. 프로토콜 구조 (MSP)

드론 테스트는 프로토콜 명령들 간의 순서가 중요하며 테스트 시나리오에 맞게 설정과 기능들을 사전에 변경해야 한다. 즉 사전 설정을 위한 과정이 필요하며 사전 설정과 관련된 프로토콜 명령들이 비행 테스트 프로파일의 가장 앞쪽에 위치한다. 테스트를 위한 사전 설정이 끝나면 드론의 제어와 기능을 동작시켜 테스트를 진행해야 한다. 따라서 테스트 프로파일의 사전 설정 부분 뒤에는 비행 제어 및 기능 동작 프로토콜 명령들로 구성된다. 비행 제어 명령은 좌우로 움직이는 롤, 앞뒤로 움직이는 피치, 회전시키는 요, 상승 및 하강시키는 스로틀과 각종 기능의 활성화 정보들이 있다. 그림 3은 MSP 프로토콜을 예로 생성한 비행 테스트 프로파일의 구조를 보여준다.

1.Config Setting	$M<	25	67	Selected Configuration Data Value	Checksum
2.Function Setting		18	203	Selected Function Data Value	
3.Setting Save		0	250	Nothing	
4.Board Reboot		0	68	Nothing	
5.Flight Control		16	200	Roll, Pitch, Yaw, Throttle, Function Activation Data	
6.Flight Control		16	200	Roll, Pitch, Yaw, Throttle, Function Activation Data	

그림 3. 생성된 비행 테스트 프로파일 구조

비행 테스트 프로파일의 앞쪽에서는 설정과 기능 변경 및 변경된 정보 저장과 보드를 재부팅하는 사전 설정과 관련된 프로토콜 명령들이 위치한다. 그 뒤에는 롤, 피치, 요, 스로틀, 기능 활성화 정보가 있는 비행 제어 프로토콜 명령들로 구성된다. 즉 비행 테스트 프로파일은 크게 사전 설정을 위한 설정 부분과 실제 드론 제어 정보가 있는 제어 부분으로 나뉜다. 각 부분은 독립적이기 때문에 쉽게 재사용할 수 있다.

비행 테스트 프로파일의 생성은 사용자가 테스트 시나리오에 맞게 선택한 설정과 기능 정보를 바탕으로 이루어진다. 그리고 사전 설정 관련 프로토콜 명령들과 비행 제어 프로토콜 명령들이 순차적으로 구성되어야 하기 때문에 테스트 프로파일을 생성할 때 반드시 주의해야 할 부분이다. 그림 4는 이런 점들을 고려하여 비행 테스트 프로파일을 생성하기 위한 비행 테스트 프로파일 생성 알고리즘을 나타낸다.

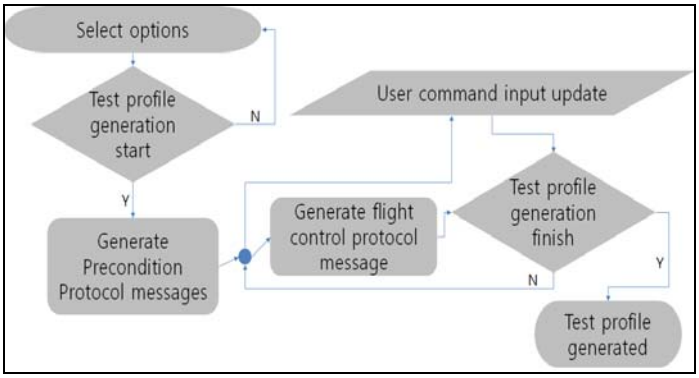


그림 4. 비행 테스트 프로파일 생성 알고리즘

비행 테스트 프로파일의 생성을 위해서는 먼저 사용자가 테스트 시나리오에 맞는 설정과 기능을 선택해야 한다. 그리고 테스트 프로파일 생성을 시작하면 사용자가 선택한 정보들을 바탕으로 적합한 사전 설정 프로토콜 명령들을 생성한다. 사전 설정 프로토콜 명령들의 생성이 끝나면 사용자의 비행 제어 및 각종 기능들의 활성화 정보에 대한 프로토콜 명령들이 지속적으로 생성되어야 한다. 따라서 사용자의 조종기 입력 정보를 계속 모니터링 하여 롤, 피치, 요, 스로틀, 기능 활성화 정보를 업데이트 해주고 이 정보들로 비행 제어 프로토콜 명령을 지속적으로 생성한다. 이 작업은 사용자가 비행 테스트 프로파일 생성을 마칠 때까지 반복해서 이루어져야 한다. 마지막으로 사용자가 비행 테스트 프로파일 생성을 마치면 그때까지 생성된 프로토콜 명령들로 비행 테스트 프로파일이 생성된다.

3.4 비행 테스트 프로파일 생성기

비행 테스트 프로파일의 구조와 생성 과정을 앞에서 살펴보았으며 이제 비행 테스트 프로파일을 생성하기 위한 비행 테스트 프로파일 생성기를 설명한다. 비행 테스트 프로파일 생성기는 여러 드론 소프트웨어에 대한 비행 테스트 프로파일 생성이 가능해야 하고 각각의 드론 소프트웨어에서 사용하는 설정과 기능들을 선택할 수 있도록 제공해야 한다. 또 사용자의 비행 제어 및 기능 활성화 같은 제어 정보들을 지속적으로 모니터링 하여 업데이트해야 한다. 마지막으로 생성된 비행 테스트 프로파일을 저장할 수 있어야 한다. 그림 5는 이러한 조건들을 만족하고 테스트에 사용할 비행 테스트 프로파일을 생성하는 생성기의 구조를 보여준다.

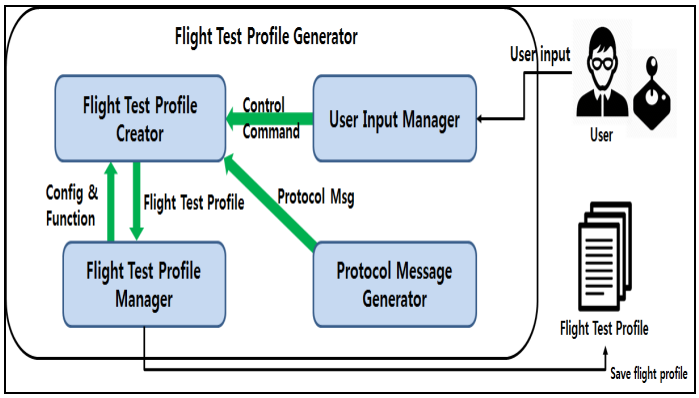


그림 5. 비행 테스트 프로파일 생성기 구조

비행 테스트 프로파일 생성기는 비행 테스트 프로파일 매니저, 비행 테스트 프로파일 크리에이터, 프로토콜 명령 생성기, 사용자 입력 매니저로 구성된다. 비행 테스트 프로파일 매니저는 드론 소프트웨어들에서 사용하는 각종 설정들과 기능들을 관리하며 비행 테스트 프로파일 크리에이터로부터 생성된 프로파일을 저장한다. 그리고 비행 테스트 프로파일 크리에이터는 사용자가 프로파일 매니저를 통해 선택한 소프트웨어, 프로토콜, 설정, 기능 정보들과 사용자의 드론 제어 정보들로 필요한 프로토콜 명령들을 판단하여 프로토콜 명령 생성기에 프로토콜 명령 생성을 요청한다. 프로토콜 명령 생성기는 크리에이터로부터 받은 요청에 맞는 프로토콜 명령을 생성한다. 마지막으로 사용자 입력 매니저는 사용자의 조이스틱이나 키보드 입력 정보를 지속적으로 모니터링 하여 이 정보들을 비행 제어명령 정보인 롤, 피치, 요, 스로틀 수치와 기능 활성화 정보로 변경하여 업데이트한다.

그림 6은 앞서 설명한 비행 테스트 프로파일 생성 알고리즘에 따라 생성기에서 비행 테스트 프로파일 생성을 하는 과정을 나타낸다.

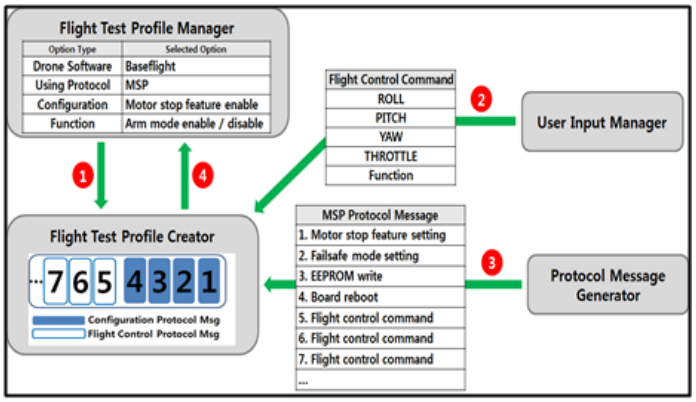


그림 6. 비행 테스트 프로파일 생성 과정

비행 테스트 프로파일 생성은 비행 테스트 프로파일 매니저로부터 시작된다. 먼저 사용자가 테스트 할 드론 소프트웨어와 설정 및 기능을 선택하고 비행 테스트 프로파일 생성을 시작한다. 테스트 프로파일 생성이 시작되면 테스트 프로파일 크리에이터가 사용자가 선택한 정보를 바탕으로 프로토콜 명령 생성기에 사전 설정 프로토콜 명령 생성을 요청한다. 사용자 입력 매니저는 비행 테스트 프로파일 생성이 시작되면 지속적으로 사용자 입력을 비행 제어 명령과 기능 활성화 정보로 변경하여 업데이트 한다. 그리고 비행 테스트 프로파일 생성이 끝날 때까지 비행 제어 프로토콜 명령을 리스트에 추가한다. 비행 테스트 프로파일 생성이 끝나면 리스트에 있는 프로토콜 명령들로 비행 테스트 프로파일을 생성하고, 생성된 테스트 프로파일을 프로파일 매니저로 전달한다. 마지막으로 테스트 프로파일 매니저가 생성된 테스트 프로파일을 파일로 저장한다.

3.5 비행 테스트 프로파일을 이용한 드론 테스트 환경

생성된 비행 테스트 프로파일로 드론 테스트를 하기 위해서는 드론과 통신하여 테스트 프로파일을 구성하는 프로토콜 명령들을 드론으로 전달해 줄 지상 제어 시스템이 필요하다. 일반적인 조종기는 프로토콜 명령이 아닌 펄스 신호로 드론의 비행 제어를 수행하기 때문에 프로토콜 명령으로 드론의 제어를 할 수 있는 시스템의 역할을 한다. 지상 제어 시스템은 저장된 테스트 프로파일을 읽어 프로파일을 구성하는 프로토콜 명령들을 주기적으로 드론에 전달하고 드론으로부터 비행 정보를 받아 표시해준다.

비행 테스트 프로파일을 이용한 드론의 테스트는 크게 두 단계로 나뉜다. 첫 번째 단계는 사용자가 드론을 테스트하는 것을 모니터링 하여 비행 테스트 프로파일을 생성하는 것이며, 두 번째 단계는 생성된 비행 테스트 프로파일로 드론의 테스트를 자동으로 수행하는 것이다. 그림 7은 비행 테스트 프로파일 생성기를 이용한 테스트 프로파일의 생성 단계를 보여준다.

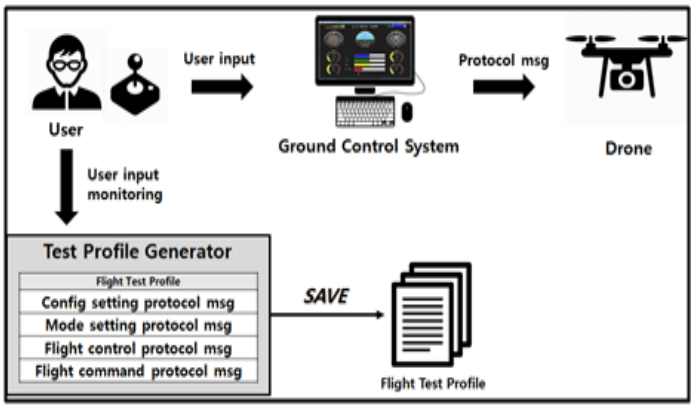


그림 7. 비행 테스트 프로파일 생성 단계

비행 테스트 프로파일 생성 단계는 드론과 지상 제어 시스템, 비행 테스트 프로파일 생성기로 수행된다.

드론과 지상 제어 시스템은 프로토콜 명령을 이용하여 서로 데이터를 주고받는다. 비행 테스트 프로파일 생성기에서는 사용자가 드론을 테스트하는 동안 사용자의 제어 정보를 지속적으로 모니터링 하여 테스트 프로파일을 생성한다. 이렇게 생성된 비행 테스트 프로파일로 드론 테스트를 자동으로 수행하는 단계에서는 지상 제어 시스템의 테스트 프로파일 실행기에서 저장된 테스트 프로파일을 읽어와 프로파일을 구성하는 프로토콜 명령 정보들을 순차적으로 드론에 전달한다. 그림 8은 생성기로부터 생성되어 저장된 테스트 프로파일을 이용하여 드론을 테스트하는 단계를 나타낸다.

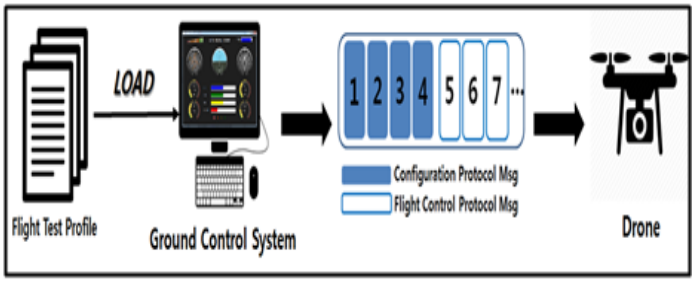


그림 8. 테스트 프로파일을 이용한 드론 테스트 단계

4. 드론 테스트 환경의 시범 적용

제안한 테스트 방법으로 실제 드론의 테스트 자동화와 재사용성이 높아짐을 보여주기 위해 테스트 데모를 진행하였다. 그리고 테스트 데모 결과를 통해 드론 소프트웨어의 결함도 확인 가능함을 보인다. 드론 소프트웨어는 오픈소스이며 MSP 프로토콜을 사용하는 베이스 플라이트와 클린 플라이트를 대상으로 하였다. 그림 9는 테스트 데모를 위해 직접 개발한 두 드론 소프트웨어와 MSP 프로토콜로 비행 테스트 프로파일을 생성할 수 있는 테스트 프로파일 생성기이다.

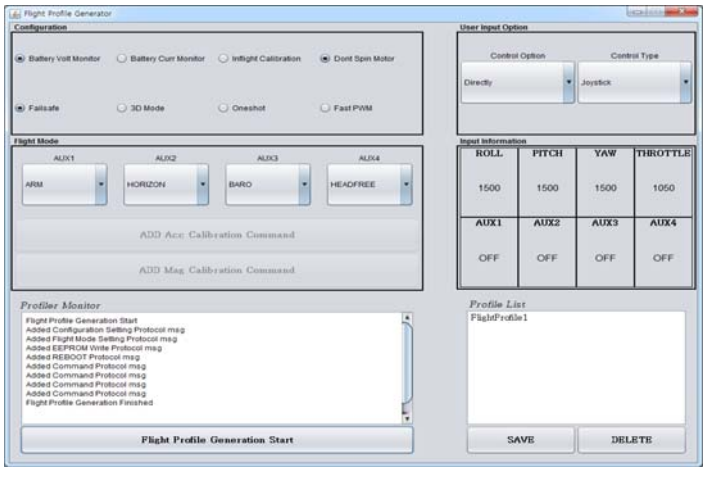


그림 9. 비행 테스트 프로파일 생성기

테스트 진행을 위해 테스트 시나리오를 설정하였으며 설정한 시나리오는 모터 정지 설정 상태에서 드론 시동 시 페일세이프 기능의 처리 상태 확인이다. 테스트 케이스는 모터 정지 설정을 기준으로 페일세이프 모드를 설정한 경우와 설정하지 않은 두 경우로 나누었다. 두 경우 모두 테스트 과정은 테스트 케이스에 맞도록 사전 설정 변경을 수행한 후 드론 시동을 켠 상태에서 일정 시간 뒤 드론과의 연결을 종료하는 것이다. 테스트를 위해 각 테스트 케이스마다 사전 설정 부분과 제어 부분을 위한 프로토콜 명령들이 필요하기 때문에 개발한 테스트 프로파일 생성기를 통해 두 소프트웨어와 두 테스트 케이스에 맞춰 총 4개의 비행 테스트 프로파일을 생성하였다. 그림 10은 테스트 정보와 생성한 테스트 프로파일을 나타낸다.

Software	Protocol	Configuration	Function	Control
Base / Clean	MSP	Don't spin motor	Arm / Failsafe	Joystick

Order	Protocol Message ID (MSP)
1	MSP_SET_CONFIG
2	MSP_SET_BOX
3	MSP_EEPROM_WRITE
4	MSP_SET_REBOOT
5	MSP_SET_RAW_RC
6	MSP_SET_RAW_RC
7	MSP_SET_RAW_RC
8	MSP_SET_RAW_RC
9	MSP_SET_RAW_RC
10	MSP_SET_RAW_RC

(a) Baseflight test profile

Different

Same

Order	Protocol Message ID (MSP)
1	MSP_SET_FEATURE
2	MSP_SET_ARMING_CONFIG
3	MSP_SET_MODE_RANGES
4	MSP_EEPROM_WRITE
5	MSP_SET_REBOOT
6	MSP_SET_RAW_RC
7	MSP_SET_RAW_RC
8	MSP_SET_RAW_RC
9	MSP_SET_RAW_RC
10	MSP_SET_RAW_RC

(b) Cleanflight test profile

그림 10. 생성된 비행 테스트 프로파일

그림 10 (a)는 베이스 플라이트의 프로파일이며 (b)는 클린 플라이트의 프로파일이다. MSP 프로토콜을 사용하는 두 드론 소프트웨어간에는 설정 변경을 위해 사용하는 프로토콜에 차이가 존재한다. 그러나 드론 제어를 위한 프로토콜은 동일한 프로토콜을 사용한다. 따라서 4개의 비행 테스트 프로파일을 생성할 때 제어 부분은 처음 생성한 비행 테스트 프로파일의 제어 부분을 재사용 할 수 있었다. 생성한 4개의 비행 테스트 프로파일로 각각 테스트를 수행하였다. 드론 F.C 보드의 경우 두 소프트웨어를 모두 사용할 수 있는 Bluebee 보드를 사용했으며 기체는 Bluebee 보드를 사용한 쿼드콥터를 2대 사용하였다[16]. 테스트는 먼저 두 기체 모두 베이스 플라이트 소프트웨어로 먼저 테스트하였으며 2가지 테스트 케이스를 각각 5번씩 총 20번 진행하였다. 그 다음 클린 플라이트 소프트웨어로 변경하여 클린 플라이트 테스트 프로파일을 이용해 각각 5번씩 총 20번 수행하였다. 그림 11은 테스트를 진행하는 과정에서 지상 제어 시스템의 화면이다. 지상 제어 시스템에서는 프로파일을 리스트에 추가하여 프로파일을 구성하는 프로토콜 명령들을 드론으로 보낸다.

그림 11. 지상 제어 시스템에서 테스트 프로파일 수행

모든 테스트는 같은 방식으로 이루어졌으며 총 40번의 테스트를 진행하면서 드론의 동작을 관찰하였다. 모든 테스트들은 지상 제어 시스템에 프로파일을 추가하는 과정을 제외하면 자동으로 수행되었다. 그리고 같은 드론 소프트웨어가 올라가 있는 서로 다른 기체에 동일한 테스트 프로파일을 재사용하여 테스트하였다. 그림 12는 드론 테스트 데모 결과로서 테스트 후 테스트 기체, 테스트 한 드론 소프트웨어, 테스트 케이스, 테스트 결과를 보여주고 있다.

Test Drone	Drone Software	Testcase	Test Result
Bluebee A	Baseflight	Testcase 1	Props spin up without warning
Bluebee B			
Bluebee A		Testcase 2	Keep the props stopped
Bluebee B			
Bluebee A	Cleanflight	Testcase 1	Props spin up without warning
Bluebee B			
Bluebee A		Testcase 2	Keep the props stopped
Bluebee B			

그림 12. 드론 테스트 시범 적용 결과

테스트 데모 결과 두 소프트웨어 모두 테스트 케이스에 따라 드론이 다르게 동작하는 것을 확인 할 수 있었다. 페일세이프 모드를 설정하지 않은 경우에는 테스트 결과 드론의 시동 후 드론과 지상 제어 시스템 간의 연결이 끊어져도 모터가 정지된 상태가 유지되어 프로펠러가 회전하지 않았다. 그러나 페일세이프 모드를 설정한 경우는 테스트 결과 드론 시동 후 드론과 지상 제어 시스템 간의 연결이 끊어지고 일정 시간이 지난 뒤에 아무런 경고 없이 모터가 갑자기 동작하여 프로펠러가 회전하였다. 테스트 결과와 테스트 케이스 정보로 원인을 파악한 결과 모터가 갑자기 동작한 이유는 드론 시동 후 모터 정지 설정으로 인해 모터가 동작하지 않고 있다가 지상 제어 시스템과 연결이 끊어지고 일정시간이 초과하여 페일세이프 기능이 수행되었기 때문이다. 드론 소프트웨어에서 사용하는 설정과 기능으로 테스트 시나리오를 생성하여 테스트를 통해 소프트웨어 결함을 발견했을 때 원인을 쉽게 파악 할 수 있었다.

5. 기존 테스트 방법과 테스트 소요시간 비교 결과

테스트 데모를 통해 제안한 방법으로 테스트 자동화와 재사용이 가능함을 보여 테스트 비용을 크게 줄였음을 보였으며 추가적으로 기존 테스트 방법과 테스트 소요시간도 비교해보았다. 비교는 실제 비행 플랫폼에서 드론 전체 시스템에 대한 테스트가 가능한 설정 어플리케이션, 로깅 정보 분석도구, 비행 테스트 프로파일 총 세 가지의 테스트 방법을 대상으로 했다. 테스트를 진행한 드론 소프트웨어는 베이스 플라이트, 테스트 시나리오는 5개의 설정과 4개의 기능을 기준으로 총 6개를 생성하였다. 테스트 환경은 표 4와 같다.

표 4. 비교 실험을 위한 테스트 환경

테스트 환경	테스트 환경 내용
드론 소프트웨어	Baseflight
프로토콜	MSP
드론 기체	Bluebee Quad X
설정	Inflight calibration (C1) Don't spin motor (C2) Failsafe setting (C3) Oneshot(C4)
기능	Acc calibration (F1) Arm / Disarm (F2) Horizon flight mode (F3) Altitude hold (F4)

테스트는 각 시나리오 별로 각각의 테스트 케이스를 5번, 6개의 테스트 케이스에 대해서 총 30번씩 수행하고 테스트에 소요된 시간을 비교하였다. 사전 설정을 위한 시간과 비행 테스트 프로파일 생성 시간은 테스트 시간에 포함시켰으며 배터리 교체 시간은 제외하였다. 설정 어플리케이션과 로깅 정보 분석도구를 이용한 테스트의 경우 사전 설정부터 드론의 제어까지 모두 직접 수행하였으며 비행 테스트 프로파일을 이용한 테스트의 경우는 비행 테스트 프로파일을 처음 생성하는 작업만 직접 수행하고 실제 테스트는 자동으로 수행하였다. 설정 어플리케이션을 이용한 테스트 중에는 사전 설정 오류나 조작 실수 등으로 인한 테스트 재시도 문제가 있었다. 그리고 로깅 정보 분석도구를 이용한 방법의 경우도 테스트는 설정 어플리케이션과 동일하게 진행했으나 테스트 후 로깅 된 정보를 분석도구로 분석하기 위해 추가 시간이 소요되었다. 비행 테스트 프로파일을 이용한 테스트는 처음 비행 테스트 프로파일을 생성하기 위한 시간

소요가 컸다. 그림 13은 세 방법의 테스트 소요시간 비교 결과를 나타낸 것이다.

Test Scenario	Time (Application)	Time (Logging)	Time (Test Profile)
Calibration during flight (C1, F1, F2)	08m 09s	10m 07s	06m 21s
Motor stop at drone start (C2, F2)	05m 53s	08m 02s	05m 40s
Motor stop at drone start with failsafe (C2, C3, F2)	06m 18s	07m 33s	04m 42s
Flight test with oneshot configuration (C4, F2, F3)	06m 23s	07m 52s	06m 14s
Flight test with horizon (C4, F2, F3, F4)	06m 51s	08m 11s	05m 35s
Altitude hold with oneshot (C4, F2, F3, F4)	07m 48s	08m 57s	05m 17s

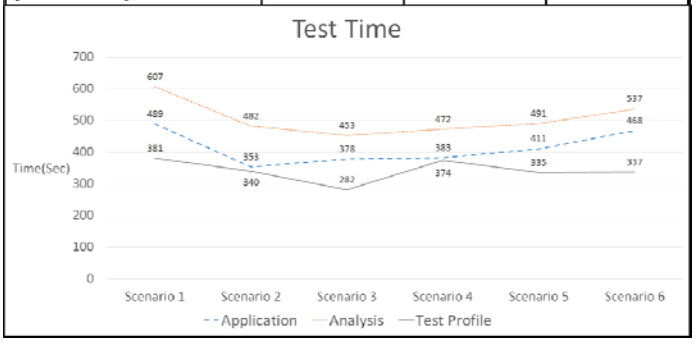


그림 13. 테스트 소요시간 비교 결과

세 가지 테스트 방법의 테스트 소요시간과 누적시간을 비교했을 때 테스트 시나리오에 따라 차이는 있었지만 본 논문에서 제안한 비행 테스트 프로파일을 이용한 테스트 방법이 다른 두 방법보다 빠른 시간에 테스트를 완료하였다. 그리고 각 테스트 방법들에서의 테스트 비용을 비교했을 때 기존의 두 방법은 테스트의 모든 과정을 직접 수행하였지만 제안한 방법의 경우 테스트를 자동으로 수행하였기 때문에 가장 효율적이었다.

6. 결 론

본 논문에서는 비행 프로토콜 명령들로 구성된 비행 테스트 프로파일을 이용한 시나리오 기반의 테스트 방법을 제안하였다. 제안한 방법은 테스트를 위한 사전 설정 부분과 실제 드론을 제어하며 테스트하는 부분을 자동화하여 한번 생성한 테스트 프로파일로 반복적인 테스트를 수행할 수 있다. 따라서 드론 테스트에 필요한 테스트 비용을 크게 줄였으며 드론 소프트웨어에서 사용하는 프로토콜 명령들을 이용하기 때문에 동일한 프로토콜을 사용하는 드론 기체나 소프트웨어에 쉽게 적용하여 테스트 할 수 있어 재사용성이 높다. 또 테스트를 위한 테스트 프로파일 생성기 구조를 제안하고 실제로 개발하여 테스트의 유용성을 보였다.

※ 본 논문은 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행하였으며(No. NRF-2017R1D1A3B04035880) 또한 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보·컴퓨팅 기술개발사업의 지원을 받아 수행된 연구임(No. 2017M3C4A7066010).

7. 참고문헌

[1] 이윤석, 김은, 김영섭, “항공 드론 제어용 소프트웨어 및 알고리즘 개발 동향”, 한국멀티미디어학회지, 제20권, 제1・2호, pp. 6-15, 2016.06
[2] 선경문, 장상진, 권다솔, 안종석, “자동 회기 드론”, 한국통신학회 학술대회 논문집, pp. 206-208, 2015.11
[3] 최병록, “무인항공기(드론) 사고의 법적책임 연구”, 한국기술혁신학회 학술대회, pp. 71-79, 2017.05
[4] 임정균, 조경록, “드론 안전실태 조사”, 한국소비자원, 안전보고서, pp.1-31, 2017.08
[5] 박승근, “추락을 잘 시키는 드론 파일럿?”, Retrieved from http://it.chosun.com/news/article.html?no=2823911&sec_no=394&pos=main_1
[6] Drone Crash Database, Retrieved from https://dronewars.net/drone-crash-database/
[7] G. Wild, J. Murray, G. Baxter, “Exploring civil drone accidents and incidents to help prevent potential air disasters,” Aerospace, vol. 3, No. 3, 22, 2016
[8] Dawei Deng, Haiwen Yuan, “UAV flight safety ground test and evaluation,” IEEE AUTOTESTCON, pp. 422-427, 2015
[9] Mission Planner, Retrieved from http://ardupilot.org/planner/
[10] QGroundControl, Retrieved from http://qgroundcontrol.com/
[11] Baseflight Configurator, Retrieved from https://github.com/multiwii/baseflight-configurator
[12] 소명호, “드론 비행사고의 시각적 분석을 위한 도구”, 경상대학교 대학원 정보과학과 네트워크항공, 석사학위논문, 2017.02
[13] Cleanflight Unit testing, Retrieved from https://github.com/cleanflight/cleanflight/blob/master/docs/development/Development.md
[14] Google Test, Retrieved from https://github.com/google/googletest
[15] jMAVSim, Retrieved from https://pixhawk.org/dev/hil/jmavsim
[16] Bluebee, Retrieved from rgblab.kr

시스템 오브 시스템즈 메타모델의 시나리오 기반 분석 및 통계적 검증 활용 가능성 분석

신용준○, 박수민, 백영민, 배두환
한국과학기술원
{yjshin, smpark, ymbaek, bae}@se.kaist.ac.kr

Scenario-based Analysis of System-of-Systems Meta-model and Applicability Analysis for Statistical Verification

Yong-Jun Shin○, Sumin Park, Young-Min Baek, Doo-Hwan Bae
School of Computing, Korea Advanced Institute of Science and Technology (KAIST)

요 약

단일 시스템으로 해결할 수 없는 복잡한 문제를 해결하기 위해 독립성과 자율성을 가지는 다수의 구성 시스템(Constituent System, CS)이 통합된 시스템인 시스템 오브 시스템즈(System-of-Systems, SoS) 개념이 대두되었다. SoS 공학에서 핵심적인 문제 중 하나는 SoS 수준의 목표 달성 여부를 검증하는 것이며, 이를 효율적으로 수행하기 위해 시뮬레이션 기반의 통계적 모델 체킹을 활용할 수 있다. 통계적 모델 체킹을 하기 위해서 시뮬레이션 가능한 SoS 모델이 필요한데, 기존 많은 연구에서는 시뮬레이션과 통계적 검증을 위한 SoS 모델링에 대한 지침을 제공해주지 못하고 있다. 이전 연구에서 SoS 모델을 생성하기 위한 온톨로지 기반의 메타모델을 정의하였는데, 본 연구에서는 M2SoS의 SoS 사례 시나리오 표현력과 검증 및 시뮬레이션에 대한 활용성을 분석하고자 한다.

1. 서 론

정보 통신 및 지능형 기술이 발달함에 따라 사용자에게 제공할 수 있는 서비스의 다양성과 질이 높아지고 있으며 이에 따라 시스템의 복잡도가 증가하고 있다. 또한 사회의 복잡도가 증가함에 따라 해결해야 할 문제의 크기가 증가하여 단일 시스템으로 해결할 수 없는 문제들이 발생한다. 이를 해결하기 위해 다수의 독립 시스템들을 하나의 거대한 시스템으로 묶는 시스템 오브 시스템즈(System-of-Systems, SoS) 개념이 대두되었다 [1]. SoS를 구성하는 구성 시스템(Constituent System, CS)은 각각 독립성과 자율성을 가지며 서로 간의 상호작용과 협력을 통해 공통의 목표를 달성하고자 한다. 예를 들어 대형 재난 대응 SoS는 인명 구조 최대화라는 SoS 수준의 목표를 위해 소방 시스템, 이송 시스템, 의료 시스템들이 협력하는 SoS라고 할 수 있다.

CS들이 협력하는 목적인 SoS 수준의 목표의 달성 여부를 검증하는 것은 SoS 공학에서 핵심적인 문제 중 하나다. 그러나 SoS의 거대한 규모와 복잡도 때문에 전통적인 정형 검증을 이용하는 것은 어렵고 높은 비용이 요구된다 [2]. SoS를 구성하는 CS들의 상호작용에 따라 목표 달성 여부가 달라질 수 있고, 독립성과 자율성을 갖는 CS들의 행동을 정확하게 예측하기 어렵다. 또한 CS는 독립적인 시스템이기 때문에 SoS의 관점에서 CS의 행동과 의사 결정 과정을 정확하게 알 수 없을 수 있으며, 불확정성을 나타낼 수도 있다.

전통적인 방법을 이용한 SoS 수준의 목표 달성 여부 검증은 SoS의 복잡성과 규모로 인한 상태 폭발 문제 등으로 인해 어렵다. 이를 해결하기 위해 통계적 모델 체킹(Statistical Model Checking, SMC)을 사용하여 정량적 SoS 목표 달성을 검증할 수 있다 [3]. SMC를 이용해 SoS를 검증하기 위해서는 시뮬레이션 가능한 SoS 모델을 만드는 작업이 필요하다. 그러나 독립적인 CS들로 구성된 SoS의 모델을 만드는 것은 큰 비용을 초래한다. 또한 SoS 모델링을 위한 전형적인 SoS 메타모델이 제안되지 않아 SoS의 목표 달성 검증을 위한 일반적인 모델 기반 프레임워크가 제안되기 어렵다. COMPASS 그룹에서 SoS 모델링 언어를 만들었지만, 구체적인 SoS 모델을 만들기 위한 기준인 SoS 메타모델은 제안된 바가 없다 [5]. 따라서 SoS 모델링을 위한 SoS 메타모델의 정의가 필요하다. SoS 메타모델은 SoS 엔지니어가 SoS를 모델링 할 때 따라야 할 지침이 될 수 있다. 또한 SoS 메타모델은 통계적 검증이 가능해야 하며 이를 위해 시뮬레이션 가능한 SoS 모델을 만들 수 있어야 한다.

따라서 본 논문은 이전 연구에서 정의한 SoS 메타모델(Meta-model for SoS, M2SoS)을 소개한다. 이후 SoS 사례 시나리오를 기반으로 M2SoS의 표현력과 SoS의 통계적 검증 활용 가능성을 확인한다. 그리고 SoS의 통계적 검증을 위한 요구사항을 도출한

* 이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원(No. R0126-18-1101, (SW 스타랩) 모델 기반의 초대형 복잡 시스템 분석 및 검증 SW 개발)과 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보・컴퓨팅기술개발사업의 지원을 받아 수행된 연구임 (2017M3C4A7066212)

다. SoS 사례 시나리오는 대형 재난 대응 시나리오와 연구팀 SoS 시나리오를 사용한다. 또한 도출된 요구사항을 반영해 메타모델 기반 시뮬레이션 및 통계적 검증 도구의 아키텍쳐를 보인다.

본 논문의 구성은 다음과 같다. 2장에서 본 연구의 배경 지식과 관련 연구를 살펴본다. 3장에서는 정의한 SoS 메타모델인 M2SoS에 대해 소개한다. 4장에서는 시나리오 기반 분석에 사용할 SoS 사례 시나리오를 보인다. 5장에서는 시나리오 기반으로 M2SoS의 SoS 표현력과 통계적 검증 활용 가능성을 분석하고 SoS 검증을 위한 요구사항 도출과 이에 따른 통계적 검증 도구의 아키텍쳐를 보인다. 6장에서는 본 연구의 한계점과 향후 연구 방향을 논의하고 7장에서 결론을 보인다.

2. 배경 지식 및 관련 연구

2.1 시스템 오브 시스템즈

단일 시스템으로 해결할 수 없는 복잡한 문제를 다수의 구성 시스템으로 해결하는 시스템을 시스템 오브 시스템즈(System-of-Systems, SoS)라고 부른다. SoS는 단일 시스템으로 해결하기 어려운 서비스, 변화 등에 의해 발생하는 문제들을 더 효과적으로 해결할 수 있는 시스템이라 할 수 있다.

SoS는 Boardman과 Sauser의 정의에 의해 5가지의 특징을 가지고 있다 [11]: 자율성(autonomy), 소속성(belonging), 연결성(connectivity), 다형성(diversity), 창발성(emergence). 자율성은 각 구성 시스템은 운영 목적에 따라 자유롭고 독립적임을 의미한다. 소속성은 더 높은 목적은 위해 시스템들이 협력함을 의미한다. 연결성은 구성 시스템들이 동적 분산 네트워크에 의한 시너지 효과를 낸다는 것을 의미한다. 다형성은 구성 시스템은 이질적인 자급자족 시스템이며, 자율적으로 발전함을 의미한다. 마지막으로 창발성은 시스템 간의 상호작용으로 SoS에 기여하는 새로운 행동이 생겨남을 의미한다.

Nielsen은 여러 문헌으로부터 SoS의 8가지 차원(dimension)에 대하여 정의했다 [1]: 자율성(autonomy), 독립성(independence), 분산(distribution), 진화(evolution), 동적 재구성(dynamic reconfiguration), 행동의 창발성(emergence of behavior), 상호의존성(interdependence), 상호운용성(interoperability). 8가지 차원에 대한 설명은 표 1에 나타난다. 본 논문은 앞서 소개한 여러 연구 중 Nielsen이 정의한 모델 기반 시스템 오브 시스템즈 공학(System of Systems Engineering, SoSE)의 8가지 차원에 따라 분석을 수행한다.

표 1. Nielsen의 연구[1]에서 정의한 SoS의 8가지 차원(dimension)

차원(Dimension)	내용
자율성 (Autonomy)	CS의 행동은 SoS 수준의 규칙이 아닌 CS 자신의 규칙에 의해 통제됨
독립성 (Independence)	CS가 SoS로부터 분리되어도 CS는 운영될 수 있는 능력을 의미함
분산 (Distribution)	직접적, 간접적 연결을 통해 통신 또는 정보 공유를 가능하게 함
진화 (Evolution)	SoS는 CS의 구조 및 구성에 관계 없이 오래 지속되며 변경 가능함
동적 재구성 (Dynamic Reconfiguration)	계획되지 않은 개입에 의해 SoS의 구조 및 구성이 변화할 수 있음
행동의 창발성 (Emergence of Behavior)	CS들의 협력을 통해 생겨난 시너지 효과로 인해 발생하는 행동을 의미함
상호의존성 (Interdependence)	SoS 공동 목표를 달성하기 위해 CS가 서로 의존해야 함
상호운용성 (Interoperability)	SoS가 다양한 이질성의 CS들을 통합할 수 있는 능력을 의미함

2.2 모델 기반 SoS 공학 기법

2.2.1 SoS 모델링 및 아키텍쳐

SoS는 규모가 비교적 크고 시스템 구성이 매우 복잡하기 때문에 SoS의 전반적인 시스템을 이해하고 분석하기 위해 다양한 모델링 및 아키텍쳐 기법이 연구되어 왔다. 특히, SoS가 갖는 주요 특성을 모델링 언어에 반영하고자 하는 연구가 많이 이뤄졌다. 2009년 Baldwin과 Sauser의 연구에서는 SoS의 다섯 가지 특성을 이론적인 방정식 형태의 모델로 표현하였으며 이를 간단한 시뮬레이션을 통한 실험을 진행했다 [10]. SoS의 적응성 및 진화에 대한 설계를 연구한 DANSE (Designing for Adaptability and Evolution in System of Systems Engineering)의 문헌에서 역시 SoS를 모델링하기 위한 정형적 시맨틱(formal semantic)을 정의하고 계약 기반의(contract-based) 의 접근 방법을 활용한다 [17]. 그 외에도, 국방 SoS를 위한 DoDAF (DoD Architecture Framework)는 SoS의 관점 기반(viewpoint-based) 분석과 설계를 위한 아키텍쳐를 정의하였고 [20], AMADEOS 그룹 역시 관점 기반의 분석을 수행하고 시뮬레이션이 가능한 형태의 모델링을 위한 SysML 기반의 아키텍쳐 및 메타모델을 제안했다 [21].

COMPASS 그룹에서 소개한 모델 기반 SoS 연구의 로드맵에 따르면 총 11가지의 모델 기반 기법 요구사항을 정리하였다. 그 중 가장 중요한 요구사항은 SoS의 모델링과 아키텍쳐가 이종의(heterogeneous) 시스템들이 상호 연결되어 통합될 수 있도록 모델링 기법 및 아키텍쳐를 지원해야 한다는 것이다 [22]. 또한 앞서 소개한 Nielsen의 연구에서는 다양한 모델링 기법과 아키텍쳐에 대한 문헌 분석을 수행하였으며, SoS의 효과적인 모델링과 아키텍쳐를 위해서는 8가지 차원에 대해 충분하고 철저히 반영될 수 있어야 하고 SoS의 특성을 반영한 유연한 아키텍쳐 제공이 필요하다고 설명한다 [1].

2.2.2 모델 기반 SoS 통계적 검증 및 시뮬레이션

SoS는 분석과 모델링뿐만 아니라, 효과적으로 CS들을 통합하여 목표를 달성할 수 있는지에 대해 검증할 필요성이 있다. 특히 SoS 전반의 모든 구조와 행동을 검증하거나 테스팅하기 어렵기 때문에 많은 연구에서 모델로 표현된 SoS를 기반으로 모델 기반의 테스팅, 시뮬레이션, 검증 등을 수행한다. 그러나 SoS는 몇 가

지 특성으로 인해 기존에 수행하던 일반적인 모델 기반 검증 기법을 적용하는 데에 한계가 따른다. 첫번째 이유는 SoS의 상태 집합(set of states)이 매우 크기 때문에 전통적인 모델 체킹을 적용한 검증 수행이 어렵다는 점이 있고, 둘째로 SoS를 이루는 여러 CS들의 자율성과 독립성으로 인해 매우 비결정적이고 불확실한 행위를 보일 수 있기 때문이다.

위와 같은 문제를 해결하기 위해 SoS와 같은 대규모 복잡 시스템에는 통계적 모델 체킹(statistical model checking) 검증 기법이 자주 활용된다. 이는 가설 검증에 기반한 기법으로써, 시스템을 확률 모델(probabilistic model)이나 비결정적 행동을 포함한 모델로 표현하고 반복적인 시뮬레이션 수행을 통해 검증 속성이 만족되는 지에 대한 정량적 결과를 도출한다 [4]. 통계적 검증을 사용한 저자의 이전 연구는 PRISM을 활용해 네 가지 타입의 SoS를 확률 모델로 표현하고 목표 달성률을 검증한 연구가 있다 [23]. 또한 CS들의 자율적인 의사 결정 매커니즘을 행동-이익-비용(action-benefit-cost) 모델로 정의하고, 자율 로봇 시나리오를 적용한 통계적 모델 체킹 연구도 진행된 바 있다 [3]. 또한 이를 SoS 수준 목표 달성 검증 도구로 구현한 연구도 진행되었다 [12].

2.3 사례 시나리오 기반 분석

시나리오 기반 분석(scenario-based analysis)이란 특정 시나리오를 구체적인 사례로 이용하여 풀고자 하는 문제에 대한 통찰이나 구체적인 정보 및 해답을 얻는 방법이다. 시나리오 기반 분석은 문제에 대한 완벽한 해답을 내기 어렵거나 구체적인 정보를 필요로 할 때, 문제를 대표하거나 그 특성을 잘 보이는 시나리오를 이용해 문제를 분석한다. 시나리오 기반 분석은 소프트웨어 아키텍처, 모델링, 요구사항 공학 등에서 사용되었다.

소프트웨어 아키텍처 연구에서는 시나리오 기반으로 아키텍처의 요구 사항 만족 여부를 분석하거나 평가하는 연구가 진행되었다 [6]. 또한 시나리오 기반 모델링 방법과 도구에 대한 연구가 진행되었다 [8]. 요구사항 공학에서는 시나리오를 이용한 구사항 명세 및 비 기능 요구사항 분석에 대한 연구가 진행되었다 [7]. 또한 레거시 시스템에서의 관심 특징 추출 등에도 시나리오 기반 분석 방법이 적용되었다 [18]. 본 논문은 구체적인 SoS 사례 시나리오 기반 분석을 이용해 SoS 메타모델의 통계적 검증 활용 가능성을 분석한다.

3. M2SoS: SoS 메타모델

본 연구에 앞서 개발된 SoS 메타모델은 M2SoS (Meta-model for System-of-Systems)이며, 온톨로지 형식으로 개발 대상 SoS를 포괄적으로 표현하고 모델링할 수 있도록 지원한다. M2SoS를 개발하기 위해 본 연구는 SoS 모델 기반 공학 기법이 갖춰야 할 네 가지 요구사항을 만족시킬 수 있도록 설계하였다. 첫째, 운영 및 관리 독립성을 갖는 CS를 통합하여 구성된 SoS를 모델링하기 위해서는 SoS 수준 요소(SoS-level entities)와 CS 수준 요소(CS-level entities)가 명시적으로 구분되어야 한다. 둘째, SoS의 효과적인 모델링 및 검증 수행이 가능하도록 SoS 수준 목표, 요구사항, 서비스, 행동을 명시적으로 정의하고 상호 간의 관계를 표현할 수 있도록 메타모델이 지원할 수 있어야 한다. 셋째, SoS을 철저하게 분석하기 위해서는 많은 불확정성(uncertainty)을 포함하는 SoS 수준 환경 요소를 메타모델에서 명시적으로 정의하고 분류할 수 있어야 한다. 넷째, SoS 수준과 CS 수준의 시스템/소프트웨어 공학자의 충분한 이해에 기반한 의사소통 및 의사 결정을 할 수 있도록 메타모델은 도메인 지식을 표현할 수 있는 온톨로지 형식을 갖추고 있어야 한다.

앞서 정의한 네 가지 요구사항을 바탕으로 아래에서는 수준별 요소(entity)를 단계별로 정의하고자 한다. M2SoS는 기본적으로 UML의 클래스 다이어그램을 이용하여 메타모델링 되었으며, 온톨로지 형식으로 각 요소와 관계를 정의하였다. M2SoS는 SoS 수준 요소와 CS 수준 요소를 명시적으로 구분하여 정의하고, SoS 수준 목표, 요구사항, 서비스, 역할 등을 명시적으로 표현함으로써 각 요소들을 명세하는 데에 활용할 수 있다. 또한 SoS 수준 환경 요소를 정의하고 세 가지 유형으로 분류한다. 마지막으로, M2SoS는 그림 1과 같이 온톨로지 형식의 메타모델로 구성되며, M2SoS를 기반으로 특정 시나리오 모델링하기 위한 개체와 관계를 충분히 정의할 수 있는 가이드라인으로 활용할 수 있다. (전체적인 M2SoS 그림은 부록에 첨부).

[1단계] SoS 주요 구성 요소 정의

M2SoS에서는 SoS를 이루는 네 가지 주요 구성 요소를 아래와 같이 정의하며, 각 주요 구성 요소는 세부적으로 다양한 하위 구성 요소를 포함한다.

- SoS-level Target Problem: 개발될(된) SoS가 CS들의 역량을 바탕으로 해결하고자 하는 상위 수준의 문제를 가리키며, SoS 개발 및 운영의 목표(Goal)로 연결되는 요소
- SoS-level Infrastructure: SoS가 SoS-level Target Problem을 해결하고 상위 수준의 목표를 달성하기 위해 개발된 기반 시설이며, 다수의 CS를 통합(integrate)하고 조정 및 지휘(orchestrate)하는 역할을 수행하는 요소. SoS의 경계(boundary)를 결정하는 역할을 하며, CS들의 상호작용을 위한 네트워크 등의 시설을 제공하는 요소
- Constituent System (CS Organization): SoS를 이루는 컴포넌트 시스템의 기본 단위이며, SoS 수준 서비스를 제공하기 위한 부분적인 역할을 수행하는 요소. CS는 새로 개발될 시스템과 이미 존재하는 레거시 시스템(legacy system)을 모두 포함하며, 관리 독립성(managerial independence)과 운영 독립성(operational independence)을 가지는 요소
- SoS-level Environment: SoS-level Infrastructure 외부에서 SoS와 상호작용하는 물리적이거나 가상으로 존재하는 요소

[2단계] SoS 수준 요소(SoS-level Entities) 정의

M2SoS에서는 SoS를 구성하는 상위 수준 요소를 SoS 수준 요소(SoS-level Entity)라고 정의하며, SoS 수준 요소는 다수의 CS

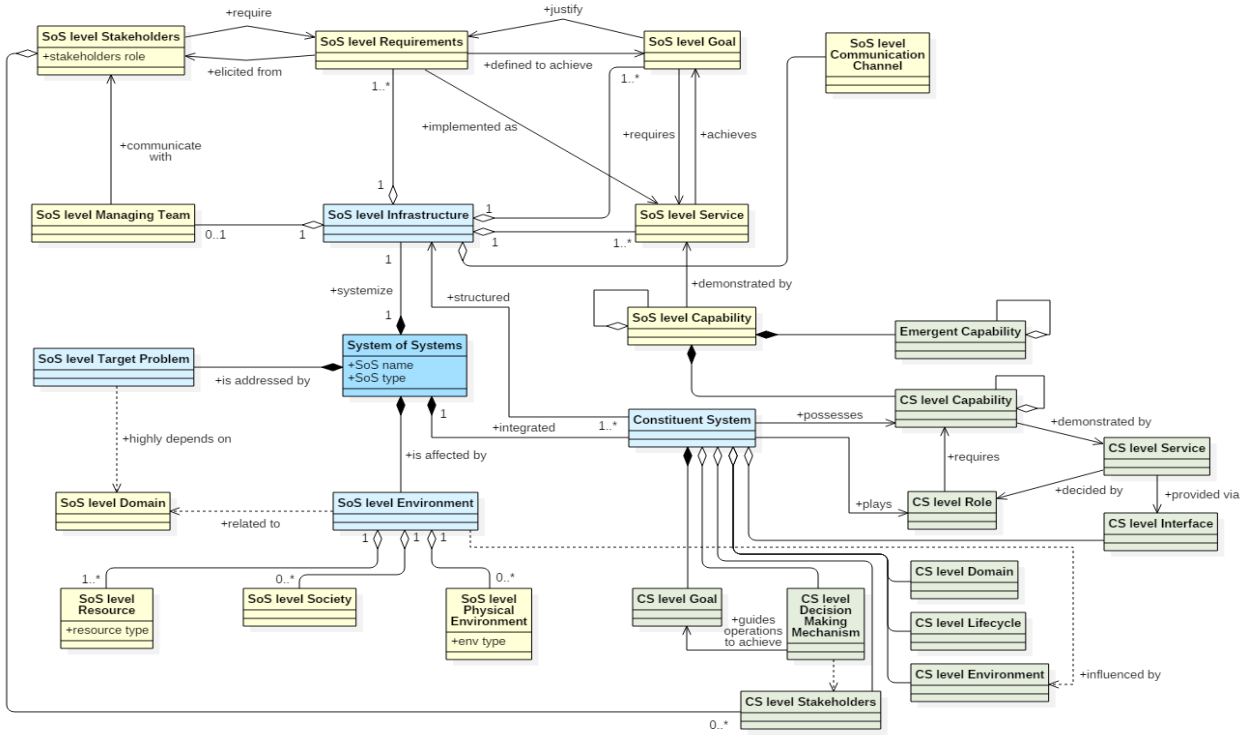


그림 1. SoS 수준 요소(SoS-level Entities)와 CS 수준 요소(CS-level Entities)를 포함하여 간략하게 표현된 M2SoS

들을 통합함으로써 상위 수준에서 정의하거나 분석해야 하는 요소들을 포함한다.

- SoS-level Goal: 다수의 CS들의 협업을 통해 달성하고자 하는 SoS 수준의 공통 목표이며, 일반적으로 독립적인 CS가 달성하기 어려운 수준의 목표로 설정
- SoS-level Requirements: SoS 수준 목표(SoS-level Goal)를 달성하기 위해 정의되는 요구사항이며, 운영 요구사항(operational requirements), 개발 요구사항(requirements), 설계 요구사항(design requirements)으로 구분하여 정의 가능
- SoS-level Service: CS들의 협업을 통해 제공되는 상위 수준의 서비스를 의미하며, 순서적(sequential), 결과적(resultant), 창발적(emergent) 행동이 서비스로 정의 가능
- SoS-level Capability: CS들의 협력을 통해 수행 가능한 SoS-level Service로부터 획득할 수 있는 SoS 수준 역량
- SoS-level Communication Channel: 여러 CS들 간, CS와 SoS 관리자 간 통신을 수행하는 데에 사용되는 통신 채널
- SoS-level Managing Team: 전체적인 SoS를 관리하고, SoS 운영에 있어 공학적 이슈를 해결하기 위한 관리팀 및 공학팀
- SoS-level Domain: 개발될 SoS가 속해있는 대상 분야(도메인)이며, 도메인 모델 등을 활용한 도메인 지식 표현을 위한 요소
- SoS-level Stakeholders: SoS 수준 서비스를 통한 목표 달성에 영향을 받는 모든 개인, 그룹 및 조직

[3단계] CS 수준 요소(CS-level Entities) 정의

M2SoS에서 정의한 CS 수준 요소(CS-level Entity)는 CS가 갖는 관리 독립성(managerial independence)과 운영 독립성(operational independence)을 표현하기 위해 SoS 수준 요소와 구분하여 정의가 필요한 요소이다. 여기서 CS 수준 요소는 CS 수준 공학자에 의해 분석 및 설계되는 요소일 수 있으며, SoS 관리자 및 공학자의 직접적인 접근 및 조작이 불가할 수 있다.

- CS-level Goal: CS가 독립적으로 갖는 기존의 시스템 목표를 의미하며, CS 수준 목표는 SoS 수준 목표와 다르거나 일부 상충할 수 있음
- CS-level Role: CS가 SoS 조직에 통합되면서 SoS 수준 목표 달성을 위해 부여된 역할
- CS-level Capability: CS가 수행 가능한 기능과 서비스로부터 도출 가능한 역량
- CS-level Service: CS의 기능적 요구사항 개발을 통해 수행 가능한 서비스로, CS 수준 목표를 달성하기 위해 구현된 요소
- CS-level Interface: CS가 다른 시스템 및 외부 환경과 통신하기 위해 필요한 인터페이스로 데이터 등을 전달하기 위해 사용되는 요소
- CS-level Decision Making Mechanism: CS가 SoS 운영 중 상위 수준의 서비스를 제공하는 동안 독립적으로 내리는 의사 결정에 필요한 매커니즘. 대표적으로 BDI (Belief-Desire-Intention) 모델이나 CBA(Cost Benefit Analysis) 등을 활용 가능
- CS-level Stakeholders: 기존의 CS 운영에 있어 CS 수준 목표를 달성하기 위한 요구사항을 정의하고, CS의 운영에 영향을 받는 모든 개인, 그룹 및 조직
- CS-level Domain: CS가 속해있는 도메인을 의미하며, 이는

SoS 수준 도메인과 다를 수 있음

- CS-level Lifecycle: CS 수준의 독립적인 운영에서 갖는 생명주기로, 항시 동일한 서비스 제공 상태를 갖지 않을 수 있는 부분을 표현하기 위한 요소
- CS-level Environment: CS가 각자의 서비스를 제공하는 데에 있어 영향을 받는 모든 외부 요소

[4단계] SoS 수준 환경(SoS-level Environment) 정의

다수의 CS가 통합되면서 구성된 SoS는 상위 수준에서 독립적인 환경 요소를 가지므로, M2SoS에서는 이를 SoS 수준 환경(SoS-level Environment)이라고 명시적으로 정의한다. 이는 SoS 수준 인프라와 상호작용 할 수 있는 모든 SoS 외부의 요소를 가리키며, 자원, 사회, 물리적 환경 요소 등을 정의할 수 있다.

- SoS-level Resource: SoS 수준 환경 요소 중 SoS의 운영에 활용될 수 있는 다양한 자원 요소로써, 인적 자원(Human Resource), 시스템 자원(System Resource), 금전적 자원(Monetary Resource) 등을 정의하기 위해 명세하는 요소
  - Resource State: 사용 가능 자원(Available Resource)과 잠재적 자원(Potential Resource)으로 구분하여 표현
- SoS-level Society: SoS 수준 환경 중 사람 혹은 외부 조직 및 시스템을 정의하기 위해 명세하는 요소
- SoS-level Physical Environment: SoS 수준 환경 요소 중 물리적으로 SoS에 영향을 주거나 SoS가 정보를 획득해야 하는 측정 가능한(measurable) 요소.

4. 메타모델의 검증 가능성 분석을 위한 시나리오

4.1 다중 손상 사고 대응 SoS (MCI Response SoS)

4.1.1 사례 시나리오 개요 및 구성

다중 손상 사고(Mass Casualty Incident, 이하 MCI)는 일반적으로 많은 수의 사상자와 피해자가 발생하여, 의료 및 구조 지원이 부족한 사고를 의미한다 [13]. MCI가 발생하는 대표적인 원인으로는 대형 사고, 자연 재해 등이 있다. MCI 대응 시나리오는 대형 사고 또는 자연 재해가 발생한 경우 상위 수준 시스템을 구성하여 부족한 의료 및 구조 자원을 지원하여 문제를 해결하는 시나리오다.

다중 손상 사고 대응 SoS(MCI Response SoS, MCIRSoS)를 SoS 레벨로, 여러 대응 시스템들을 CS로 생각한다. CS들은 독립성과 자율성을 지니고, 서로 다른 목표, 능력 그리고 서비스 등을 제공한다. MCIRSoS는 CS들과 함께 인지하는 공통의 목표가 존재하며, 목표를 달성하기 위한 능력과 역할을 가진다. 또한 환경, 인프라 그리고 자원을 가지고 있다.

MCIRSoS 시나리오에서 MCIRSoS는 구조 파견 시스템, 소방 시스템, 응급 의료 시스템 및 이송 시스템, 의료 시스템으로 구성된다. 구조 파견 시스템의 목표는 사고 정보를 통해 환자 관리를 지시하고, 소방 시스템은 환자 구조 및 화재를 진압한다. 응급 의료 시스템 및 운송 시스템은 환자 응급 처치 및 이송을 담당하고, 의료 시스템은 환자 치료를 통해 환자의 생명 유지를 목표로 한다.

MCIRSoS는 환자 95% 이상 구조, 80% 이상의 생존율을 목표로 한다. 또한 금전적 자원, 인적 자원, 도로 환경, 날씨 환경 에 영향을 받는다. MCIRSoS에는 대응을 지휘하는 관리팀이 존재하며, 정보 공유를 위한 커뮤니케이션 채널 등을 가지고 있다. MCIRSoS 사례 시나리오의 적합함을 보이기 위해 SoS의 8가지 차원으로 분석하여 표 2에 정리하였다.

4.1.2 다중 손상 사고 대응 SoS 시나리오 예시

A시에서 고층 건물에 화재가 발생하였다. 다중 손상 사고 대응 SoS(MCI Response SoS, MCIRSoS)의 목표는 환자 95% 이상을 구조하고, 80% 이상 생존할 수 있는 구조 및 치료를 제공하는 것이다. MCIRSoS의 지휘 센터는 구성 시스템에 업무를 분배한다. 현재 MCIRSoS는 1개의 구조 파견 시스템, 5개의 소방 시스 템, 2개의 응급 의료 시스템 및 이송 시스템, 1개의 의료 시스템으로 구성된다.

구조 파견 시스템은 전화 제보를 통해 화재의 위치, 크기 그리고 심각성 등을 파악한다. 또한 주변의 환경 및 지원 가능한 자원들을 파악 후 소방, 응급 의료 및 운송 시스템을 파견하고, 의료 시스템의 수용 능력을 조사한다.

소방 시스템의 소방 대원은 환자를 안전한 위치로 구조를 하는 동시에 소방차를 이용해 화재를 진압한다. 5개의 소방 시스템이 존재하므로 3명의 소방 대원은 환자를 구조하고 2명의 소방 대원은 소방차를 이용해 화재를 진압할 수 있다.

응급 의료 시스템 및 이송 시스템의 의사는 응급 처치를 제공하고, 구급차와 헬리콥터는 의료 시스템으로 환자의 이송을 담당한다. 2개의 응급 의료 시스템 및 이송 시스템이 존재하므로 각각 1대씩의 소방차와 헬리콥터를 파견한다. 소방 시스템으로부터 구조된 환자들은 구급차를 통하여 응급 처치를 진행하며 이송한다. 지상에서 구조가 어려운 경우, 헬리콥터를 이용하여 환자를 구조하여 응급 처치를 진행하며 이송한다. 이송 시스템은 의료 시스템과 통신하여 의료 시스템의 수용 능력을 판단 후 환자를 이송한다. 가까운 거리의 의료 시스템의 수용 능력이 부족할 경우 수용이 가능한 의료 시스템으로 이송한다. 또한 지휘 센터에서 이송 시스템이 부족하다고 판단하면 이송 시스템이 추가될 수 있다.

의료 시스템은 수용 능력 내에서 환자를 수용하며, 수용한 환자에 대하여 환자의 상태를 분석하여 적절한 치료를 제공한다. 현재 1개의 의료 시스템이 존재하므로, 해당 의료 시스템의 수용 능력을 초과할 경우 환자를 수용할 수 없다. 이럴 경우 지휘 센터에서는 의료 시스템의 공간과 인적 자원을 고려하여 더 많은 의료 시스템을 공급하여야 한다.

4.2 연구팀 SoS (Research Team SoS)

4.2.1 사례 시나리오 개요 및 구성

연구팀 시나리오(Research Team SoS, RTSoS)는 논문 출판을 위해 연구원들이 협업하는 과정을 나타낸 시나리오이다. RTSoS

표 2. Nielsen이 정의한 8가지 차원(dimension)에 따른 SoS 사례 시나리오 분석

차원(Dimension)	MCI 대응 SoS (MCIRSoS)	연구팀 SoS (RTSoS)
자율성 (Autonomy)	각각의 CS들은 고유한 목표를 가지고 해당 목표를 이루기 위해 각자의 규칙에 따라 행동이 통제되므로 자율성을 가짐	연구원들은 자신이 연구에 어느정도 참여할지 스스로 결정하며 결정 메커니즘을 소유하여 자율성을 가짐
독립성 (Independence)	각각의 CS들은 목표, 능력, 규칙 등에 의해 결정되어 운영하므로 독립성을 가짐	연구원은 연구팀에 소속되지 않아도 스스로 연구하고 논문을 쓸 수 있는 능력 소유하여 독립성을 가진다고 할 수 있음
분산 (Distribution)	지리적으로 다른 위치에 존재하는 CS들은 SoS 수준의 인프라와 통신을 하여 상호작용을 이루므로 분산되어 있음	연구원들은 지리적으로 분산되어 있을 수 있고, 분산된 상황에도 협업이 가능함
진화 (Evolution)	MCI의 상황이 변함에 따라 CS들의 목표, 역량, 행동 등은 변화하므로 진화함	새로운 기능, 도메인의 연구원 등 연구팀 구성 변화와 환경의 변화로 인해 진화함
동적 재구성 (Dynamic Reconfiguration)	환경의 변화로 인해 관리 팀이 재구성 될 수 있으며, 상황에 따라서는 MCIRSoS의 지휘관이 바뀔 수 있으므로 동적 재구성을 가짐	목표 논문지가 달라지거나 마감 기한이 달라지는 등 변화하는 환경과 상황에 따라 연구 팀 내부 구성을 동적으로 재구성함
창발성 (Emergence)	서로 다른 CS들 간에 협업을 하고 협업을 통해 시너지 효과를 창출하여 더 나은 목표를 달성 가능하므로 창발성을 가짐	새로운 능력이나 도메인을 가진 연구원 고용으로 연구팀의 행동 변화, 새로운 기능이 발현되어 창발성을 가짐
상호 의존성 (Interdependence)	SoS 수준의 목표 달성을 위해 각각의 CS들은 분업을 하여 행동을 하는 것으로 보아 서로 상호 의존성을 가짐	연구원들의 공통의 목표를 달성하기 위한 상호작용, 협력 및 장점 극대화함으로 상호 의존성을 가짐
상호운용성 (Interoperability)	SoS 수준의 관리 팀은 여러 개 존재할 수 있으며 관리 팀들은 서로 협력을 통해 SoS 수준의 목표를 달성할 수 있는 것으로 보아 상호운용성을 가짐	연구팀은 능력과 목표 등이 서로 다른 연구원들을 연구팀에 포함하여 상호운용성을 가짐

는 기존의 연구에서 SoS의 사례 시나리오로 사용된 바 있으며 본 연구는 이를 구체화하였다 [9]. 연구팀은 SoS 레벨로, 여러 개인 연구원들을 CS로 생각한다. CS인 개별 연구자는 독립성과 자율성을 가지는 독립 시스템으로 볼 수 있다. 따라서 연구자들은 개인의 각자 다른 능력, 목표 등을 가지며 다른 연구자들과 상호작용과 협력한다. 이 때 RTSoS는 연구자들이 함께 인지하는 공통의 목표가 있으며, 개별 연구자들이 공통의 목표를 달성하기 위해 가지는 능력, 역할을 가진다. 또한 이들 주위의 환경과 자원을 가진다.

RTSoS 시나리오의 구성은 다음과 같다. 연구팀은 개별 연구자들로 구성된다. 연구원은 교수, 박사과정, 석사과정, 등으로 나뉠 수 있다. 연구팀의 목표는 논문 및 특허 출원 실적을 달성하는 것이고, 특정 연구 분야를 가지고 있다. 연구팀은 운영 자금, 물적 자원, 연구 환경을 갖추고 있으며, 공통 연구 분야를 가진 연구 커뮤니티에 속해있다. 또한 연구팀의 기반 시설로 연구팀 관리팀, 커뮤니케이션 채널 등을 가지고 있다. SoS 사례 시나리오로 사용하기 적합함을 보이기 위해 RTSoS를 SoS의 8가지 차원으로 분석하여 표 2에 보였다.

4.2.2 연구팀 SoS 시나리오 예시

연구팀 A의 공통의 목표는 연간 국외 논문 5편 이상을 포함한 논문 10편 이상 작성이라는 실적을 달성하는 것이다. 또한 특허 출원이라는 추가 실적이 존재한다. 연구팀에는 지도 교수 1명, 박사과정 학생 3명, 석사과정 학생 4명이 있다. 지도교수는 학생의 연구를 지도하고, 박사과정은 연구 및 프로젝트 팀을 이끌고, 석사과정은 연구 및 연구 보조 등의 역할을 가진다. 이렇게 각각의 개인들은 개별의 능력과 역할이 정해진다. 연구팀은 정기적인 모임, 세미나, 면담이나 이메일을 통해 소통과 상호작용한다. 그리고 연구팀은 컴퓨터와 같은 물적 자원과 자금을 가지고 있고, 학교나 연구실 건물의 환경에 영향을 받는다. 또한 소프트웨어 공학 연구 커뮤니티와 학교의 학과에 속해있다.

지도교수 또는 지도교수와 박사과정으로 이뤄진 연구팀 관리팀은 목적 달성을 위해 연구팀 내부 소그룹을 운영한다. 연구팀 A는 현재 소그룹 3개를 운영 중이며 각각 박사과정 한명이 한 개의 소그룹을 이끈다. 박사과정 한 명이 졸업으로 연구팀을 떠나게 되었을 때, 목적 달성에 차질이 없도록 다른 소그룹의 박사과정과 석사과정 2명을 해당 소그룹에 투입하여 구조를 변경한다. 또한 연구팀 관리팀은 추가 실적 달성을 위해 변리사를 투입함으로 특허 출원 실적을 달성하도록 할 수 있다. 따라서 새로운 도메인의 전문가를 통해 연구팀은 새로운 기능을 갖게 된다.

5. M2SoS의 검증 활용성 분석

5.1 M2SoS의 사례 시나리오 표현력

M2SoS는 여러 도메인의 SoS의 요소를 모두 표현할 수 있는 능력이 있어야 하는데, 이를 위해 M2SoS가 사례 시나리오의 주요 구성 요소를 표현 가능한지 분석하였다. 분석 결과에 따라 두 가지 SoS 사례 시나리오의 구성 요소들을 M2SoS에서 정의한 요소들과 대응시킬 수 있었으며 이를 표 3에 정리하였다. 표를 통해 M2SoS에서 SoS의 주요 구성 요소를 포괄적으로 모델링하기 위해 정의했음을 알 수 있었고, SoS 메타모델 개발을 위해 필요한 요구사항도 만족됨을 확인할 수 있었다. 특히, 기존 메타모델 연구에서 중점적으로 다루지 않았던 상위 수준(SoS 수준)과 하위 수준(CS 수준) 요소 간 구분이 명확하게 이루어졌고, SoS 수준 인프라를 통해 CS를 조직하고 연결하기 위한 기반 요소들의 명세가 가능하다고 분석할 수 있다.

표 3. M2SoS의 SoS 시나리오 표현력 분석

		MCI 대응 SoS (MCIRSoS)	연구팀 SoS (RTSoS)
SoS-level Target Problem		다중 손상 사고 상황의 환자 발생	학과와 프로젝트에서 요구하는 연간 실적
SoS-level Domain		다중 손상 사고 재난 대응, 대형 사고 대응 계획	공학 연구, 소프트웨어 공학, 비즈니스 프로세스 관리
SoS-level Infrasturcture	SoS Level Requirements	● 소방 시스템과 의료 시스템 은 행동 지침에 따라야 함 ● 이송 시스템은 의료시스템과 통신을 통하여 환자를 최대한 빠르게 이송할 수 있어야 함 ● 환자 구조를 위한 구조 행동의 순서를 따라야 함	● 연구원은 연구팀이 공식 일정에 참석해야 함 ● 연구실 운영에 필요한 최소한의 연구원이 필요하고, 학사 졸업 이상의 능력이 요구됨 ● 연구원은 한국어나 영어로 의사소통이 가능해야 함
	SoS Level Goal	MCI 상황에서 발생한 환자를 95% 이상 구조하고 80%이상 생존할 수 있는 구조 및 치료	연간 논문 10편 출판, 국외 논문을 5편 이상 작성 연간 2개 이상의 프로젝트를 진행
	SoS Level Service	구조 파견 시스템과 소방 시스템의 협력을 통한 인력 분배 및 구조 시간 단축 소방 시스템과 이송 시스템의 협력으로 환자 구조 및 이송	국외 및 국내 논문 작성 프로젝트 연간 보고서 및 발표 자료 작성 연구실 내부 워크샵 진행
	SoS Level Communication Channel	문자, 재난 문자, 전화, 무전과 같은 데이터 전송을 위한 directed channel	정기적, 비정기적인 모임과 면담, 이메일 등을 통한 상시적인 소통, 정기적인 세미나
	SoS Level Stakeholders	정부, MCI 지역의 주민, 후원금을 기부한 사람 등 MCI상황에 관련된 모든 이해 관계자들	교수, 학생 등 연구실 구성 인원, 소속 학교, 연구사업 프로젝트 관계자(국가 또는 기업)
	SoS Level Managing Team	다중 손상 사고의 사건 지휘관 및 지휘 센터 (Incident Commander (IC), Command Center 등)	지도 교수 또는 지도 교수와 박사과정 학생으로 구성된 연구실 관리팀
SoS-level Environment	SoS Level Resource	버스나 택시와 같은 잠재적 이동 수단 정부 지원금, 후원금 등을 통해 사용할 수 있는 금전적 자원	연구실 운영을 위한 자금 컴퓨터, 프린터, 사무 용품 등의 연구실의 물적 자원
	SoS Level Society	다중 손상 사고 재난 대책 본부, 재난 대응 팀, 재난 구조 팀 등 다중 손상 사고에 대응할 수 있는 다양한 그룹	소프트웨어 공학 연구 커뮤니티(타 연구 그룹 등), 소속 학교의 컴퓨터 공학 학과
	SoS Level Physical Environment	지리적 환경으로 도로 상태, 신호등 등이 있으며, 기후와 날씨 환경 및 관련되는 모든 물리적인 환경 요소	소속 학교 내 연구실 건물과 연구실, 교내 학과 건물
Constituent Systems (CS)		**구조 파견 시스템:** 사고 정보를 통해 적절한 CS에 환자 관리 지시 ● 보고된 정보를 통해 사고 분석 수행 역량 필요 ● 다중 손상 사고의 규모 및 심각도를 파악하여 CS들에 업무 분담	**교수:** 연구실 운영 및 학생 지도 ● 본인 연구와 연구실의 지도 교수로서 학생의 연구 방향을 지도 ● 논문 작성 방법 지도를 통해 학생의 논문의 질을 향상시키고, 연구실이 운영될 수 있도록 프로젝트 수주
		소방 시스템: 환자 구조 및 화재 진압 ● 다중 손상 사고의 현장 상황에 적절한 장비를 사용하여 환자 구조 및 화재 진압 능력 필요 ● 환자의 위치 정보를 통해 환자에게 접근하여 환자 구조 및 안전 장소로 이송, 그리고 화재 진압	**박사 과정:** 연구 및 해외 저널 논문 출판, 석사 과정 지도 ● 연구 팀의 팀장과 프로젝트의 리더를 수행함 ● 실적 달성을 위해 연구 및 논문을 작성하고, 석사 과정 학생 면담과 회의 및 프로젝트를 진행
		응급 의료 시스템 및 이송 시스템: 최단 시간에 환자 응급 처치 및 의료 시스템으로 이송 ● 응급 처지 또는 수술을 제공 능력 및 빠른 시간 안에 의료 시스템으로 이송 가능한 능력 필요 ● 환자의 위치 정보를 통해 환자에게 접근하여 응급처치 제공 및 의료 시스템으로 이송	**석사 과정:** 연구 및 국내 저널 논문 출판, 연구 보조 ● 연구 팀의 팀원으로서 연구실에서 진행 중인 프로젝트에 참여하며 행정을 보조함 ● 실적 달성을 위해 연구 및 논문을 작성하고 프로젝트 보고서 작성 보조
		의료 시스템: 환자 치료를 통한 환자의 생명 유지 ● 병원은 환자를 수용할 수 있는 능력 필요 ● 전달 받은 환자의 상태를 분석 후 적절한 치료 제공	**변리사:** 프로젝트 실적을 위한 특허 출원 진행을 보좌 ● 연구실 실적에 필요한 특허 출원과 관련된 정보를 자문 ● 특허 출원에 문제가 발생하지 않도록 특허권 검색

다시 말해, 표 3에 정리된 내용은 M2SoS를 기반으로 개발하고자 하는 SoS를 이루고 있는 물리적이거나 가상의 구성 요소를 분석한 것이라고 할 수 있다. 이러한 분석을 통해 개발하고자 하는 SoS의 요소를 정의하고 SoS의 도메인을 표현했다면, 이를 다양한 모델링 기법을 통해 모델링 할 수 있다. M2SoS는 사용되는 모델링 기법에 중립적인 기술-중립적인(technique-neutral) 메타모델이기 때문에 각 요소에 대한 구체적인 모델링 기법의 개발은 본 논문에서 다루지 않는다.

5.2 모델 기반의 SoS 통계적 검증

5.2.1 검증을 위한 M2SoS의 개선 요구사항

M2SoS는 MCIRSoS와 RTSoS 시나리오에 대한 표현력을 갖추었으므로, M2SoS를 따라 SoS를 모델링하면 검증 대상을 충분히 표현할 수 있는 모델을 만들 수 있다. 그러나 SoS의 표현력을 갖춘 모델만으로 통계적 검증을 수행할 수는 없다. SMC를 이용한 통계적 검증을 위해서는 시나리오를 시뮬레이션 및 검증 가능해야 하며 이를 위해 필요한 요소가 정의되어야 한다. 검증을 위한 M2SoS 개선 요구사항을 도출하기 위해 AMADEOS에서 정의한 SoS 분석의 7가지 관점 중 통계적 검증과 연관된 역동성, 진화, 시간, 창발성의 관점으로 분석하였다 [14]. 다음은 이를 통해 도출된 M2SoS의 개선 요구사항이다.

시뮬레이션 실행 가능한 모델 정의: M2SoS는 시나리오를 정적으로 표현하는 능력만 가지고 있어 SoS의 역동성, 진화, 시간, 창발성의 관점은 보이지 못한다. 현재의 M2SoS를 따르는 SoS 모델이라 하더라도 통계적 검증을 위한 시뮬레이션이 불가능할 수 있다. 따라서 실행 가능한 SoS 모델로의 정의가 필요하다.

시뮬레이션 시간 정의: 시뮬레이션에서 시간 요소는 필수적이나 M2SoS는 SoS의 시간을 표현하지 못하고 있다. 시뮬레이션에서 시간을 표현하는 방식은 크게 이산 시간과 연속 시간으로 나눌 수 있다. 또한 시뮬레이션 시작 및 종료 시간에 대해서도 명시해야한다.

시나리오의 이벤트 정의: M2SoS는 검증하고자하는 SoS를 표현 가능하나 SoS의 행동과 구조의 변화에 대한 표현이 어렵다. 즉 SoS의 역동성, 진화, 창발성 관점을 표현하지 못한다. 따라서 검증 대상 모델에서 발생하는 일련의 이벤트들이 명시되어야 한다. SoS 시뮬레이션과 검증을 위해서는 SoS의 행동, 구조 변화 등을 표현할 수 있는 시나리오의 이벤트 정의가 필요하다.

검증 속성 및 모델 체커: SoS 시나리오를 검증하기 위해서는 검증 속성과 모델 체커가 필요하다. 모델 체커는 주어진 모델의 검증 속성 만족 여부를 확인한다. 그러나 M2SoS는 검증 속성과의 관계를 보이지 않아 통계적 검증에 직접 이용하기 어렵다. 따라서 검증 속성과 이를 검증 할 체커를 명시 해야한다.

5.2.2 모델 기반 통계적 검증을 위한 아키텍쳐 설계

위 절에서 도출한 SoS 검증을 위한 요구사항 요소를 반영해 M2SoS를 따르는 SoS 모델을 통계적 검증하기 위한 아키텍쳐를 설계했다. 그림 2는 SoS 메타모델 기반의 시뮬레이션 및 통계적 검증 도구의 아키텍쳐다.

시뮬레이션 실행 가능한 모델(Simulation Executable SoS Model)은 M2SoS를 따르는 모델(SoS Model)을 Java와 같은 프로그래밍 언어로 변환하여 만들어진다. 따라서 M2SoS의 표현력과 현실성은 검증의 대상이 되는 실행 가능한 모델에 직접적으로 반영되고 영향을 준다. 시뮬레이션 시간(Time)은 실행 가능한 SoS 모델의 CS들의 생명 주기, 서비스 수행 시간 등 동적인 요소에 표현된다. 또한 시나리오의 각 이벤트에 발생 시간, 종료 시간 등이 함께 명시된다. 이런 시뮬레이션 시간 요소는 통계적 검증 수행 시간에 영향을 준다.

시나리오(SoS Scenario)는 검증 대상인 SoS에서 나타날 수 있는 이벤트들로 구성되고 SoS 시뮬레이션 엔진(Simulation Engine)에 의해 실행된다. 또한 SoS 시나리오는 불확정적인 요소나 동적인 요소를 가질 수 있다. 검증 속성(Verification Property)은 SoS 목표를 정형적으로 표현한 것이다. 모델 체커는 정형적으로 표현된 검증 속성으로 시뮬레이션의 결과를 검증한다.

M2SoS에 기반한 SoS 모델과 검증을 위해 요구사항으로부터 정의된 요소들은 시뮬레이션 기반 SoS 검증(Simulation-based Verification for SoS)에 사용된다. 이는 크게 SoS 시뮬레이터(SoS Simulator)와 통계적 검증기(Statistical Verifier)로 구성된다. 실행 가능한 모델과 시나리오는 각각 시뮬레이션의 모델과 시나리오 데이터로 시뮬레이션 엔진의 입력이 된다. 시뮬레이션 엔진은 입력들을 실행하고 결과물인 트레이스를 통계적 검증기의 입력으로 전달한다. 검증 체커는 시뮬레이션 트레이스와 SoS 목표를 표현한 검증 속성을 입력으로 받아 검증 결과를 반환하고 이를 분석한다. 본 아키텍쳐는 SoS의 8가지 차원을 표현할 수 있는 M2SoS에 기반하여, M2SoS가 가지는 표현력을 보존하며 통계적 검증을 수행할 수 있다. 또한 이를 기반으로 SoS 모델링과 검증을 통합하는 연구로 이어질 수 있을 것이다.

6. 연구 한계점 분석 및 향후 연구 방향

시나리오 기반 분석에 사용한 SoS 시나리오: 본 연구의 시나리오는 메타모델 분석을 위한 용도로 저자들에 의해 작성되었다. 즉, 분석 대상인 M2SoS에 기반하여 기술된 시나리오라고 보일 수 있다. 그러나 MCIRSoS의 경우 다중 손상 사고 대응에 대한 전문 문헌을 30건 이상 참고하여 현실을 나타낸 시나리오이고, RTSoS는 선행 연구에서 모델링과 시뮬레이션이 수행되었던 시나리오에 구체적인 스토리텔링을 추가하여 기술한 것이다 [10]. 그리고 기존의 SoS 연구들에서는 본 논문의 시나리오와 같이 구체적으로 기술된 공통된 시나리오가 없었다는 문제가 있다. 따라서 본 연구를 기반으로 SoS 연구에 공통적으로 사용될 수 있는 구체적인 시나리오를 확립해나가는 연구가 필요할 것이다.

전통적인 검증 기법 고려의 부재: 본 연구는 전통적인 검증 기법을 고려하지 않으며 통계적 모델 체킹에만 집중하고 있다. 실제로 전통적인 검증 방법을 사용해 SoS를 검증하는 연구가 존재

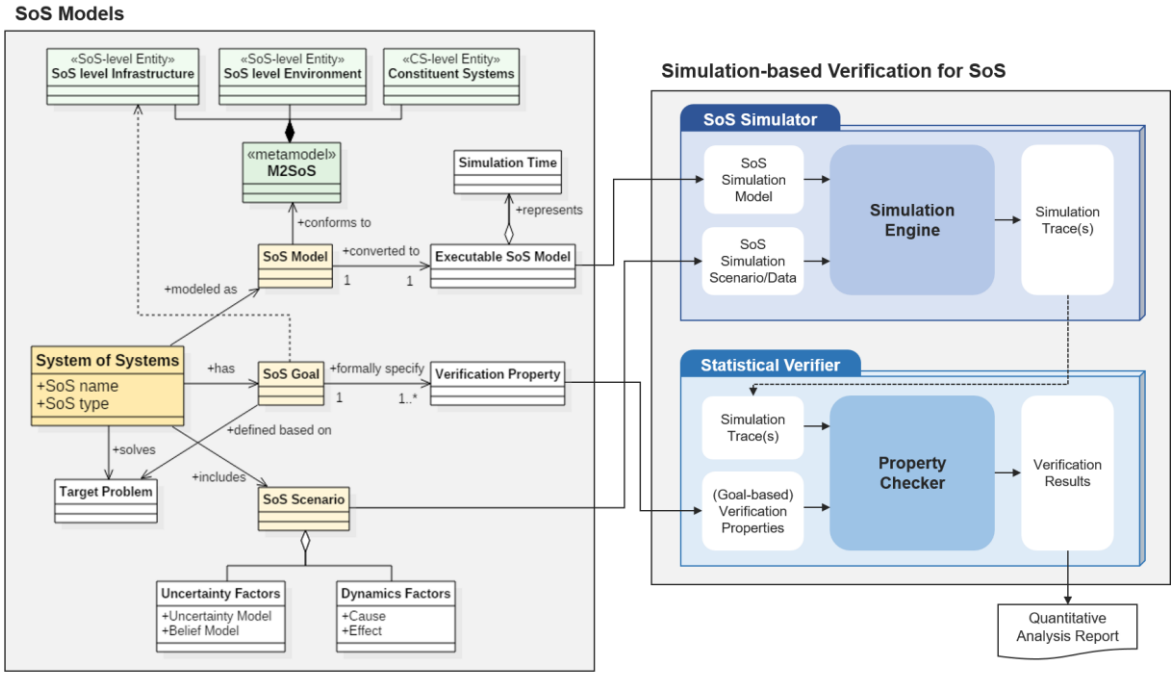


그림 2. SoS 메타모델 기반 시뮬레이션 및 통계적 검증 도구의 아키텍쳐

한다 [15]. 그러나 전통적인 검증 기법을 이용한 SoS 연구의 경우, 제한된 행동을 하는 디바이스간의 프로토콜 수준의 사례에만 적용하여 그 크기와 복잡도가 한정된다. 하지만 MCIRSoS와 같은 초대형 SoS의 경우 SoS가 가지는 상태의 수가 무한에 가까워 모델 체킹과 같은 전통적인 검증 기법에 의해 다뤄지기 어렵고, SMC를 이용한 검증이 효과적임을 저자의 이전 연구를 통해 확인했다 [23]. 또한 SoS 통계적 검증 도구를 개발 중에 있어, 통계적 검증에 기반한 SoS 연구를 지속적으로 이어나갈 수 있을 것이다.

메타모델에 기반한 모델링 결과의 부재: 본 연구는 정의한 M2SoS를 시나리오 기반으로 요소를 도출하여 두 가지 SoS 도메인에서의 표현력을 분석한 후, 최종적으로 통계적 검증에의 활용 가능성을 분석하였다. 그러나 본 논문에서는 M2SoS를 기반으로 요소를 도출하여 비교 분석을 수행했지만, 두 가지 사례 SoS의 실제 모델링 결과를 통한 표현력 분석은 이루어지지 않았다. 그 이유는 현재의 M2SoS는 개발하고자 하는 여러 도메인의 SoS를 표현할 수 있도록 도메인 일반적인(domain-generic) 표현을 지원하는 데에 초점을 맞추고 있기 때문이다. 또한, 본 연구는 다중 에이전트 시스템을 대상으로 한 Gaia 방법론과 같이 기술 중립적(technique-neutral)인 방법론의 기반 메타모델로 활용하는 데에 목적이 있다 [16].

7. 결 론

본 연구는 이전 연구에서 개발한 SoS 메타모델인 M2SoS를 소개하고, SoS 표현력과 통계적 검증 가능성을 시나리오 기반으로 분석했다. 두 가지 도메인의 SoS 사례 시나리오를 8가지 차원에 기반해 분석한 결과, M2SoS가 SoS의 모델링 및 도메인 지식을 표현하기 위한 가이드라인으로 활용될 수 있음을 보였다. 하지만 현재 개발된 메타모델을 시뮬레이션 기반의 통계적 검증에 활용하기 위해서 M2SoS의 개선을 위한 요구사항 정의와 통계적 검증 기법 및 도구의 아키텍쳐 설계가 필요했다. 이를 위해 본 연구에서는 M2SoS가 통계적 검증에 활용되기 위해 만족해야 할 네 가지 개선 요구사항(시뮬레이션 모델, 시간 요소, 시나리오와 이벤트, 검증 속성 및 모델 체커)을 정의하였으며, M2SoS 기반 통계적 검증 도구의 아키텍처를 제안했다. 향후 연구로는, 본 연구에서 제안한 SoS 통계적 검증 도구의 아키텍처를 활용해 SoS의 시뮬레이션 기반 통계적 검증 도구(Simulation-based Statistical Verification Framework for SoS)를 개발하고자 한다. 또한 본 논문에서 소개한 두 가지 SoS 사례 시나리오를 보다 구체화하여 시뮬레이션 및 검증에 실제 적용할 예정이다.

참 고 문 헌

[1] Nielsen, Claus Ballegaard, et al. "Systems of systems engineering: basic concepts, model-based techniques, and research directions." ACM Computing Surveys (CSUR) 48.2: 18, 2015.

[2] Kim, Youngjoo, et al. "Statistical model checking for safety critical hybrid systems: An empirical evaluation." Haifa Verification Conference. Springer Berlin Heidelberg, 2012.

[3] 김준호, 신동환, 배두환 (2017). 시스템 오브 시스템즈 수준의 목표 달성 검증을 위한 행동-이익-비용 모델과 통계적 모델 체킹 적용 연구. 정보과학회 컴퓨팅의 실제 논문지, 23(4), 256-261.

[4] Legay, A., Delahaye, B., & Bensalem, S. (2010). Statistical Model Checking: An Overview. RV, 10, 122-135.

[5] Woodcock, J., Cavalcanti, A., Fitzgerald, J., Larsen, P., Miyazawa, A., & Perry, S., "Features of CML: A formal modelling language for systems of systems." 2012 7th International Conference on IEEE System of Systems Engineering (SoSE), pp. 1-6, Jul. 2012. [5] Kazman, R., Abowd, G., Bass, L., & Clements, P. (1996). Scenario-based analysis of software architecture. IEEE software, 13(6), 47-55.

[6] Kazman, R., Abowd, G., Bass, L., & Clements, P. (1996). Scenario-based analysis of software architecture. IEEE software, 13(6), 47-55.

[7] Sutcliffe, A. (1998). Scenario-based requirements analysis. Requirements engineering, 3(1), 48-65.

[8] Greenyer, J., Gritzner, D., Gutjahr, T., König, F., Glade, N., Marron, A., & Katz, G. (2017). ScenarioTools-A tool suite for the scenario-based modeling and analysis of reactive systems. Science of Computer Programming, 149, 15-27.

[9] Baldwin, W. C., Sauser, B., & Cloutier, R. (2015). Simulation approaches for system of systems: Events-based versus agent based modeling. Procedia Computer Science, 44, 363-372.

[10] Baldwin, W. C., & Sauser, B. (2009, May). Modeling the characteristics of system of systems. In System of Systems Engineering, 2009. SoSE 2009. IEEE International Conference on(pp. 1-6). IEEE.

[11] Boardman, John and Brian Sauser. 2006. System of systems - the meaning of Of. In the Proceedings of the 2006 IEEE/SMC International Conference on System ©2009 IEEE of Systems Engineering, 118-123, Los Angeles, CA. April 24-26.

[12] 진민규, 신동환, 김준호, 배두환. (2017). 시스템 오브 시스템즈 수준의 목표 달성 검증 도구. 한국정보과학회 학술발표논문집, 552-554.

[13] CHONG, Shao Foong. Mass Casualty Incident. In: Disaster Medicine. Springer London, 2013. p. 165-177.

[14] Mori, M., Ceccarelli, A., Lollini, P., Fromel, B., Brancati, F., Bondavalli, A., Andrea, "Systems-of-systems modeling using a comprehensive viewpoint-based SysML profile," (2017), Journal of Software: Evolution and Process, Article in Press

[15] Faldik, O., Payne, R., Fitzgerald, J., & Buhnova, B. (2017). Modelling System of Systems Interface Contract Behaviour. arXiv preprint arXiv:1703.07037.

[16] L, Cernuzzi, T. Juan, L. Sterling, and F. Zambonelli, "The Gaia Methodology: Basic Concepts and Extensions," Methodologies and Software Engineering for Agent Systems, pp. 69-88.

[17] A. Arnold, B. Boyer, and A. Legay, "Contracts and Behavioral Patterns for SoS: The EU IP DANSE Approach," In Proceedings AiSoS 2013, arXiv:1311.3195 (https://arxiv.org/abs/1311.3631)

[18] Kim, J. A., Lee, H., Jung, R., & Kim, S. (2014). Goal and Scenario-based Feature Identification Techniques from Legacy System. International Journal of Software Engineering and Its Applications, 8(2), 145-150.

[19] Song, Jiyoung & Baek, Young-Min & Jin, Mingyu & Jee, Eunkyoung & Bae, Doo-Hwan. (2017). SoS GaP slicer: SoS Goal and PRISM Models for Change-Responsive Verification of SoS.

[20] M. Hause, "The Unified Profile for DoDAF/MODAF (UPDM) Enabling Systems of Systems on Many Levels," 4th Annual IEEE Systems Conference, 2010.

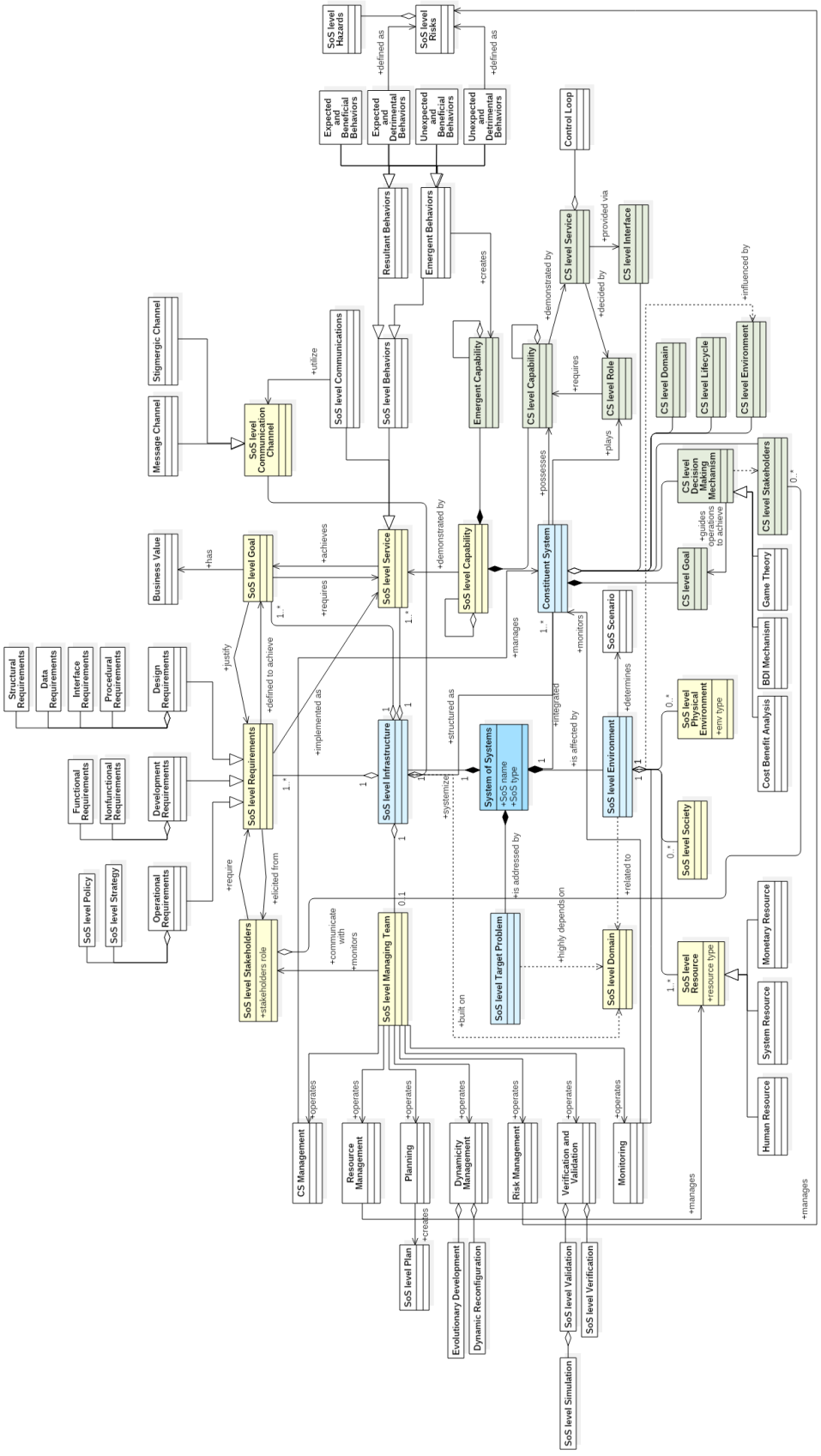
[21] M. Mori, et al., "Systems-of-Systems Modeling Using a Comprehensive Viewpoint-Based SysML Profile," Journal of Software: Evolution and Process, Special Issue - HASE 2016, 2017.

[22] COMPASS Group, "Roadmap for Research in Model-Based SoS Engineering," Comprehensive Modelling for Advanced Systems of Systems (COMPASS) - Public Document (Grant Agreement: 2872829), 2014.

[23] D. Seo, D. Shin, Y. M. Baek, J. Song, W. Yun, J. Kim, E. Jee, and D. H. Bae, "Modeling and Verification for Different Types of System of Systems using PRISM," Workshop of Software Engineering for System of Systems (SESoS), 2016.

부 록

부록 A. M2SoS: Meta-model for System-of-Systems



RGB 색 공간 분할 기법을 통한 AMOLED 디스플레이의 소모 전력 모델 생성 및 평가

백두산 [1], 최유림 [1], 이병정 [2], 이정원 [1]
[1]아주대학교 전자공학과
whitedusan@gmail.com, {cyl5780, jungwony}@ajou.ac.kr
[2]서울시립대학교 컴퓨터과학과
bjlee@uos.ac.kr

Generation and Evaluation of Power Model for Mobile AMOLED Display using RGB Color Space Partitioning Method

Dusan Baek[1], Yoo-Rim Choi[1], Byungjeong Lee[2], Jung-Won Lee[1]
[1]Department of Electrical and Computer Engineering, Ajou University.
[2]Department of Computer Science and Engineering, The University of Seoul.

요 약

모바일 AMOLED 디스플레이의 소모 전력을 소프트웨어 수준에서 참조하기 위해서는 소모 전력에 대한 모델이 필요하다. 하지만, 전력 모델을 생성하거나 평가하기 위한 기존 연구들은 전력 측정 활동을 위한 환경 및 장비 구축이 필수적으로 요구되었기 때문에 이를 구축하지 못할 경우에는 연구를 진행 할 수 없었다. 또한, 전력 모델을 생성하거나 평가하기 위해 사용되는 RGB 값의 조합이 무분별하고 적어 RGB 값들 간의 상호 영향을 반영하기 어려웠다. 본 연구는 이러한 문제점을 해결하기 위해 RGB 색 공간 분할 기법 및 <RGB 값, 소모 전력>으로 구성된 맵핑 테이블 제공 방법을 제안한다. 제안된 분할 기법을 통해 색 공간에서 균등한 색 차이를 갖는 RGB 조합들을 샘플링하고, 샘플링된 RGB 조합들로 구성된 맵핑 테이블을 생성한다. 실험을 통해 분할의 깊이에 따른 샘플의 특징을 분석하였고, 이를 바탕으로 AMOLED 디스플레이에 대한 맵핑 테이블을 생성하였다. 이러한 맵핑 테이블은 연구자들에게 제공되어 전력 측정 활동 없이도 전력 모델을 생성하거나 평가하는데 활용 될 수 있다.

1. 서 론

모바일 기기를 구성하고 있는 다양한 모듈들의 소모 전력을 소프트웨어 수준에서 관리하기 위한 다양한 연구들이 존재하고 있다[1]. 이러한 연구들은 모바일 기기가 가지고 있는 제한된 전력이라는 한계점을 극복하기 위해 소모 전력을 프로파일링 한 뒤, 비효율적으로 소모되고 있는 전력 버그를 식별하고, 수정한다. 특히, 디스플레이는 모바일 시스템의 소모 전력 중 많은 비중을 차지하고 있기 때문에 디스플레이의 소모 전력 모델의 필요성은 크다고 할 수 있다.

이러한 흐름에 맞추어, AMOLED 디스플레이의 전력 모델에 대한 연구들이 진행되고 있다. [2]은 R, G, B 값과 소모 전력 간의 비선형적 특성을 반영한 모델을 제안하였다. [3]은 R, G, B 채널 간의 의존관계를 밝히고, 이를 모델에 반영하여 [2]보다 높은 정확도를 갖는 모델을 생성하였다. 이와 같은 연구들이 제안하는 모델들은 회귀 분석에 기반한 모델로서 모델 내 계수는 AMOLED 디스플레이의 종류에 따라 달라지며, 샘플 이미지를 활용해서 획득한다.

하지만 기존 연구들이 사용한 샘플 이미지는 실제 RGB 값의 조합(각 8bits 시, 약 1670만개)에 비해 매우 적은 조합을 사용하기 때문에 RGB 값들 간의 상호 영향을 모델에 반영하기 어려우며, RGB 값의 조합들을 무분별하게 샘플링하기 때문에 효과적인 테스트를 기대하기 어렵다. 또한, 모델 생성 과정에서 AMOLED 디스플레이의 소모 전력 측정 활동을 진행해야 하기 때문에 필수적으로 소모 전력 측정을 위한 실험 환경 및 장비를 구축해야만 한다.

본 연구는 이러한 문제점을 해결하기 위해 RGB 색 공간 분할 기법을 제안한다. RGB 색 공간 분할 기법은 모든 RGB 값의 조합을 테스트할 수 없는 한계를 개선하고자, 주어진 테스트 제약 사항(테스트 시간

"이 논문은 2018 년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행되었으며(NRF-2016R1A2B1014376), 2018 년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No. NRF-2014M3C4A7030504)."

등)을 고려하며, 샘플을 추출하는 방법이다. 제안된 기법을 이용할 경우, 인접 샘플과의 색 차이가 동일한 샘플을 추출할 수 있을 것이다. 더불어, 본 연구는 <RGB 값, 소모 전력>으로 구성된 맵핑 테이블 제공 방법에 대해 제안한다. RGB 색 공간 분할 기법을 통해 얻어진 RGB 값의 조합을 대상으로 소모 전력을 측정하였으며, 이를 맵핑 테이블로 생성하였다. 맵핑 테이블은 관련 연구자들에게 제공되어 전력 측정 활동 없이도 전력 모델을 생성하거나 평가할 수 있도록 재사용될 수 있다.

2. RGB 색 공간 분할 기법

R, G, B 채널로 구성되어 있는 RGB 색 공간에서 RGB 값의 조합을 샘플링 하기 위해서는 3차원 공간을 다루어야 한다. 그림 1은 R, G, B 채널로 구성된 색 공간을 3차원의 직교 좌표계에 표현한 그림으로서 본 논문에서 제안하는 RGB 색 공간 분할 기법을 설명하기 위한 그림이다.

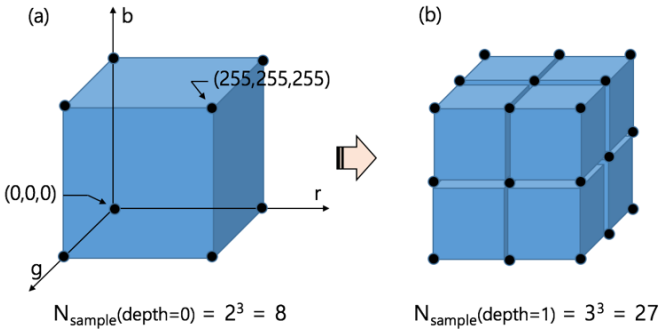


그림 1 RGB 색 공간 분할

그림 1의 정육면체(a)는 원점 (0,0,0)과 최대값 (255,255,255)을 양 끝 꼭지점으로 하는 정육면체를 나타내며, 공간 분할을 진행하는 첫 번째 대상이 된다. 정육면체를 구성하고 있는 각 꼭지점(8개)은 샘플링의 대상이 되는 RGB 값의 조합이며, 중복을 제거하기 위해 RGB 값의 조합은 집합의 원소로 저장된다. 그 뒤, 보다 많은 RGB 값의 조합을 샘플링하기 위해서는 그림 1에서 보여지는 바와 같이 정육면체(a)를 8개의 작은 정육면체(b)로 분할하고, 분할된 각각의 정육면체를 구성하고 있는 꼭지점을 샘플링한다. 이와 같은 과정을 주어진 시간에서 수행할 수 있는 샘플의 개수를 고려하며 반복한다. 순차적인 분할 과정 중 분할 이후 샘플 간의 최소 간격이 분할 이전의 최소 간격의 1/2이하가 될 때, 깊이가 증가한다고 표현한다.

3. 분할 깊이에 따른 샘플의 특징 및 맵핑 테이블

RGB 색 공간 분할은 깊이가 증가함에 따라 샘플의 수가 기하 급수적으로 증가된다. 따라서, 테스터는 주어진 테스트 제약 사항에 근거하여 샘플의 개수를 선택해야한다. 테스터가 샘플의 개수를 선택함에 있어 참조 가능한 정보를 제공하기 위해 분할 깊이에 따른 샘플의 특징을 분석하였다. 실험은 갤럭시 S8을 대상으로 진행하였으며, 인접 샘플 간 전력 차이의 최대값은 [2]에서 제안한 모델을 활용하여 분석하였다.

표 1 RGB 색 공간 분할 깊이에 따른 샘플 정보

깊이	0	1	2	3	4	5
색차이 (RGB 색 공간)	256	128	64	32	16	8
샘플 개수	8	27	125	729	4913	35937
인접 샘플 간 전력 차이의 최대값 (mW)	83.41	60.98	37.92	21.14	11.16	5.73

위 실험의 결과에서 보이는 바와 같이 색 차원의 분할 깊이에 따라 인접 샘플 간 전력 차이의 최대값이 감소하나, 샘플들의 개수는 기하급수적으로 증가하는 것을 확인할 수 있다.

표1에서 분석된 정보에 기반하여 125개 샘플로 구성된 맵핑 테이블을 생성하였다. 그림 2는 맵핑 테이블의 예이다.

R	G	B	전력
239	255	255	0.160221136
239	223	239	0.140740477
175	191	127	0.084311117
175	191	159	0.125030

그림 2 맵핑 테이블의 예

이와 같은 맵핑 테이블은 전력 모델을 생성하거나 평가하는데 재사용 될 수 있다.

4. 결론 및 향후 계획

본 논문은 모바일 AMOLED 디스플레이의 전력 모델을 생성하고, 평가하는 과정에서 전력 측정 활동을 제거하기 위해 RGB 색 공간 분할 기법 및 <RGB 값, 소모 전력>으로 구성된 맵핑 테이블 제공 방법을 제안한다. 실험을 통해 분할 깊이에 따른 샘플 특성을 확인하였고, 갤럭시 S8에 탑재된 AMOLED 디스플레이의 RGB 값과 소모 전력 간 맵핑 테이블을 생성하였다. 향후 연구로서, 더욱 다양한 모바일 AMOLED 디스플레이를 대상으로 맵핑 테이블을 생성하고, 공개할 계획이다.

참고문헌

[1] Hoque, Mohammad Ashraful, et al. "Modeling, profiling, and debugging the energy consumption of mobile devices." ACM Computing Surveys (CSUR) 48.3 (2016): 39.

[2] Dong, Mian, Yung-Seok Kevin Choi, and Lin Zhong. "Power modeling of graphical user interfaces on OLED displays." Proceedings of the 46th Annual Design Automation Conference. ACM, 2009.

[3] Hong, Seongwoo, Suk-Won Kim, and Young-Jin Kim. "3 channel dependency-based power model for mobile AMOLED displays." Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE. IEEE, 2017.

차량용 소프트웨어의 테스팅을 위한 자동화된 HiL 테스트 모의 환경 구축

황규본 [1O], 최기용 [1], 이병정 [2], 이정원 [1]
아주대학교 전자공학과 [1], 서울시립대학교 컴퓨터과학과 [2]
bon1229@ajou.ac.kr, ki815kaisian@gmail.com, bjlee@uos.ac.kr, jungwony@ajou.ac.kr

Automated HiL test simulation environment for automotive software testing

Gyubon Hwang[1O], Ki-Yong Choi[1], Byeongjeong Lee[2], Jung-Won Lee[1]
Department of Electrical and Computer Engineering, Ajou University[1],
Department of Computer Science and Engineering, The University of Seoul[2]

요 약

차량용 소프트웨어의 개발 과정에서 HiL(Hardware in the Loop) 테스트는 소프트웨어와 하드웨어의 통합을 검증하기 위해 수행된다. 그러나 차량용 소프트웨어의 검증을 위한 HiL 테스트 환경을 구축하는 것은 도구와 설비, 자동화 수준, 디버깅 정보 제공 능력 등의 제약 사항을 고려하여야 한다. 따라서 본 논문에서는 차량용 소프트웨어의 테스팅을 위한 자동화된 HiL 테스트 모의 환경 구축 방법을 제안한다. 이를 위해 자동화된 HiL 테스트 모의 환경의 기본 특성을 설명하고 SUT의 기능 요구사항 정의, SUT 기능 모델링, 테스트 실행기 구현의 세 단계로 구축 방법을 제안한다. 제안하는 방법은 메모리 업데이트를 통한 차량용 소프트웨어의 결함 지역화 연구에 적용해 평가하였다. 실험 결과, 기본 특성을 모두 만족하여 자동화된 HiL 테스트가 가능함을 확인하였고, 디버깅 정보 제공 방법이 반영되어 있었다. 이를 통해 차량용 소프트웨어의 테스팅을 위한 자동화된 HiL 테스트 모의 환경을 구축할 수 있음을 확인하였다.

1. 서 론

최근 자동차의 다양한 기능들은 전자 장치와 소프트웨어로 구현됨에 따라 차량용 소프트웨어의 오동작은 대량의 자동차 리콜이나 치명적인 사고의 원인이 된다. 이처럼 차량용 소프트웨어의 품질 보장이 중요함에 따라, 차량용 소프트웨어의 개발 전주기에 대한 안정성 보증 및 품질 관리를 위한 표준이 있다. 자동차 기능 안전 국제 규격 (ISO 26262)에서는 통합 시험을 포함한 여러 단계로 품질 보증 과정을 구분하고 있다[1]. 차량용 소프트웨어의 개발 단계에 따른 검증 방법에서 HiL(Hardware-in-the-loop) 테스트는 소프트웨어와 하드웨어의 통합 단계의 검증 방법이다.

하지만 실제 HiL 테스트 환경을 구축하기 위해서는 고려해야 할 제약사항이 있다. 첫째, HiL 테스트 환경을 구축하기 위한 도구나 설비의 규모를 고려해야 한다. 일반적으로 자동차, 철도, 선박, 항공, 국방 등 고신뢰가 요구되는 임베디드 시스템의 개발에서 HiL 테스트가 수행된다[2, 3, 4]. 그러므로 신뢰수준을 만족시킬 수 있는 도구와 설비가 필요하다[5]. 둘째, 테스트 수행의 자동화 정도를 고려하여야 한다. 자동화된 테스트는 수행 시간의 단축, 반복 테스트 등 여러 장점을 얻을 수 있다. 자동화에는 테스트 케이스 생성의 자동화, 테스트 케이스 입력의 자동화, 테스트 수행의 자동화, 오라클의 자동화, 출력 평가의 자동화 등이 있다. 자동화 정도에 따라 HiL 테스트 환경을 구축하는 방법은 다양하다[6]. 셋째, HiL 테스트 환경의 테스트 정보 제공 능력을 고려하여야 한다. 기본적으로 HiL 테스트는 입력에 대한 출력을 평가하는 검증 방법이다[2]. 따라서 테스트 수행 중 발견된 결함에 대해 얻을 수 있는 정보는 제한적이다.

이와 같은 고려사항들에 의해 일반적인 HiL 테스트 환경을 구축하는 것은 쉽지 않다. 따라서 본 논문에서는 차량용 소프트웨어의 테스팅을 위한 자동화된 HiL 테스트 모의 환경 구축 방법을 제안한다. 제안하는

"이 논문은 2018년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No. NRF-2014M3C4A7030504)."

방법은 먼저, 구축할 테스트 환경이 만족해야 할 기본적인 특성을 분석한다. 다음, 출력 평가의 근거가 되는 오라클을 구현하는 것을 중심으로 테스트하려는 대상 시스템(SUT, System Under Test)의 기능 요구사항 정의, 기능 모델링, 테스트 실행기 구현의 세 단계로 구축한다.

제안하는 자동화된 HiL 테스트 모의 환경이 만족해야 할 기본적인 특성은 세 가지이다. 첫째, 자동화된 테스트가 가능해야 한다. 테스트 수행, 출력 평가, 반복 수행 등이 자동화 되어야 한다. 둘째, 정확하게 테스트를 수행할 수 있어야 한다. 즉, 원하는 시간에 입력을 줄 수 있어야 하며, 출력에 대하여 테스트 결과를 정확하게 평가하고, 횟수에 상관없이 동일해야 한다. 셋째, 차량용 인터페이스에 의해 동작해야 한다. 자동차 내부의 ECU(Electronic Control Unit)에서 보편적으로 사용하는 통신 네트워크로는 FlexRay, CAN(Controller Area Network), LIN(Local Interconnect Network) 통신이 있다. 실제 차량에서 사용되는 통신 네트워크를 사용하여 인터페이스를 구성해야 한다.

이러한 특성을 만족하는 자동화된 HiL 테스트 모의 환경은 SUT의 기능 요구사항 명세, 기능 모델, 테스트 실행기로 구성된다. SUT 기능 요구사항 명세는 HiL 테스트를 통해 검사할 기능을 정의한 것으로, 입력과 출력을 정의한 기능 명세를 통해 기능 모델링을 가능케 한다. 기능 모델은 입력에 대하여 SUT가 어떤 출력을 내는지를 모델링한 것이다. 이 기능 모델은 실행 가능한 모델로서 테스트 실행기에서 오라클로 동작한다. 오라클은 입력에 대한 예상 출력을 만들어 테스트 통과 여부를 판별하는 근거가 된다. 테스트 실행기는 출력을 평가하는 근거인 오라클과 SUT와의 입출력 인터페이스를 포함한다.

본 논문에서 제안하는 방법은 메모리 업데이트를 통한 차량용 소프트웨어의 결함 지역화 연구[7]에 적용하여 평가한다. 제안하는 방법에 따라 테스트 실행기를 구현하고 메모리 데이터를 전송할 수 있는 에이전트를 포함하는 SUT와 메모리 데이터 수집기를 포함하는 모델 기반 HiL 테스트 모의 환경을 구축한다. 실험을 통해 테스트 통과 여부를 오류 없이 판단하는 것을 확인하여 차량용 소프트웨어의 테스팅을 위한 자동화된 HiL 테스트 모의 환경을 구축할 수 있음을 확인한다. 또한, 제안하는 방법이 HiL 테스트 환경의 세 가지 제약사항을 고려하였는지 알아본다.

논문의 구성은 다음과 같다. 2장 관련연구에서는 일반적인 HiL 테스트 환경 구축 현황을 살펴보고 3장에서는 차량용 소프트웨어 테스팅을 위한 자동화된 HiL 테스트 모의 환경 구축 방법을 제안하고, 이에 따라 구축한다. 제안한 방법은 4장에서 실험을 통해 평가한다. 마지막으로 5장에서 결론을 맺는다

2. 관련연구

차량용 소프트웨어와 하드웨어의 통합은 HiL 테스트를 통해 검증할 수 있다[2]. 2장에서는 HiL 환경과 관련된 연구들을 통해 HiL 테스트 모의 환경 구축 방법의 필요성을 알아보고 HiL 테스트의 특징과 한계를 통해 자동화된 HiL 테스트 모의 환경이 만족해야 할 기본적인 특성을 살펴본다.

2.1 HiL 환경

HiL 환경은 일반적으로 HiLS (HiL simulation) 환경을 의미하며 주로 실제 시스템을 동작 시키기 어려운 항공기 같은 분야에서 외부 환경을 모사한 가상의 환경에서 시스템의 성능을 평가하는 방법으로 연구되었다[8, 9]. 또한, 소프트웨어와 하드웨어의 통합 검증을 위한 목적으로 주로 자동차 업계에서 연구되고있다[10]. 하지만 기존의 연구들은 특정 기능의 검증을 위한 환경 구성에 초점이 맞춰져 있어, 기능에 의존적이다. 일반적인 기능의 검증을 위한 체계적인 HiL 테스트 환경 구축 방법에 대한 연구는 미비하다.

또한, 기존 HiL 테스트 환경은 상용 제품들을 통해 구축되었다[10]. 상용 HiL 테스트 시스템들은 실차를 모사하여 ECU의 기능 및 성능을 테스트할 수 있는 시스템으로서 개별 ECU 테스트에서부터 전체 네트워크를 통한 통합 테스트까지 수행할 수 있도록 지원한다. 예를 들어, dSPACE SCALEXIO 제품 라인은 자동차, 우주 항공, 의료, 운송 등 다양한 산업 분야의 HiL 테스트 및 RCP(Rapid Control Prototyping) 프로젝트를 위한 모듈식 시스템이다[11]. 동사 제품의 SYNECT와 같은 소프트웨어와 함께 사용 시 테스트케이스의 DB화, 결과 분석, 실행 계획, 시스템 모델, 요구사항 등 개발 및 테스트 과정의 전반적인 데이터 관리가 가능하다[12]. ETAS의 LABCAR 또한 전체 개발 사이클(V 사이클)을 커버하는 표준화된 툴로서 ECU를 위한 소형의 실시간 테스트 시스템이며 개방형 및 모듈 방식의 구조이다[13]. ADAS iit의 SET도 위의 시스템과 유사한 모듈 구조의 HiL 테스트 시스템으로 높은 확장성을 가진다[14]. 이러한 제품들은 대규모 시스템의 전체 계통을 테스트하기 위한 HiL 테스트 환경을 쉽게 구축할 수 있게 한다. 하지만 대부분 고가로 이를 이용하여 HiL 테스트 환경을 구축하는 데에는 많은 비용이 요구된다.

그러나 차량용 소프트웨어의 일부 기능 및 편의 기능을 대상으로 한 테스트에 이런 상용 설비가 반드시 필요한 것은 아니다. 유사한 모의 환경에서도 구축할 수 있다면 충분히 테스트가 가능하다. 따라서, 차량용 소프트웨어의 테스팅을 위한 HiL 테스트 모의 환경을 구축할 수 있는 체계적인 방법이 필요하다.

2.2 HiL 테스트의 특징 및 한계

일반적으로 HiL 테스트는 주로 시간 제약 조건을 만족하지 못하는 경우 시스템에 치명적인 영향을 줄 수 있는 실시간 시스템에서 수행된다. 이러한 시스템에서의 결함은 대량의 리콜이나 치명적인 사고를 일으킬 수 있다. 따라서 높은 신뢰 수준을 보장하기 위해 대규모 산업체에서는 HiL 테스트를 수행하고 있으며 이를 위해 고가의 도구나 대규모 설비를 사용하고 있다.

또한, 과거에 비해 차량 내부에서 차량용 ECU가 차지하는 비중과 복잡성이 매우 증가함에 따라 효율적인 테스트의 필요성이 증가하였다. 이에 따라 테스트의 자동화에 대한 중요성도 증가하였다. 테스트 자동화는 테스트 케이스 생성의 자동화, 테스트 케이스 입력의 자동화, 테스트 수행의 자동화, 오라클의 자동화, 출력 평가의 자동화 등 다양한 수준에서 이루어질 수 있다[6].

한편, HiL 테스트는 시뮬레이터를 통해 검사할 대상 시스템에 입력을 인가하고, 입력에 대한 출력을 관찰하는 블랙박스 테스트이다. 따라서, 결함이 발생하면 관련 정보를 알 수 없는 한계가 있다. 우선, HiL 테스트는 개발이 완료된 시스템을 대상으로 수행되므로 일반적으로 JTAG(Joint Test Action Group)나 BDM(Background Debug Mode)과 같은 디버깅 인터페이스가 노출되어 있지 않다. 또한, HiL 시뮬레이터와 ECU는 독립적인 개별 시스템으로 이루어져 있어, 설계 단계에서 고려되어 있지 않다면 디버깅을 위해 임의의 시간에 중단시킬 수 없다[15]. 그러므로 결함 발생 시 소프트웨어의 어느 부분이 실행되고 있는지, 또는 내부 변수의 값이 무엇인지 등 디버깅을 위한 정보를 얻는 데 한계가 있다[2].

이러한 HiL 테스트의 특성에 따라 HiL 테스트 환경을 구축하기 위해서는 HiL 테스트 도구 및 설비의 규모, 테스트 자동화 수준, 디버깅 방법을 고려하여야 한다.

3. 자동화된 HiL 테스트 모의 환경 구축

이 장에서는 차량용 소프트웨어를 테스트하기 위한 자동화된 HiL 테스트 모의 환경이 가져야 할 기본 특성을 설명하고 HiL 테스트 모의 환경 구축 방법을 제안한다. 제안한 방법은 차량용 소프트웨어의 결함 위치 추정 연구에 적용한다.

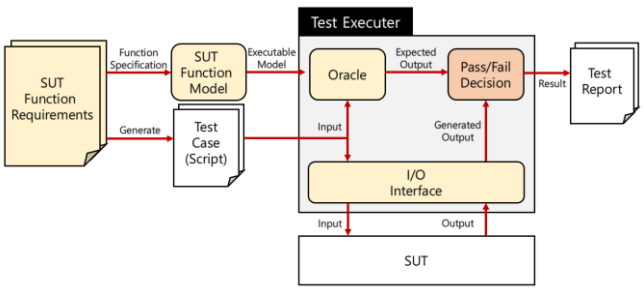


그림 1 모델 기반 HiL 테스트 모의 환경 구조도

3.1 자동화된 HiL 테스트 모의 환경 구축 방법

그림 1은 제안하는 자동화된 HiL 테스트 모의 환경의 구조도로 크게 SUT의 기능 요구사항, SUT의 기능 모델, 테스트 실행기로 구성된다. 먼저 테스트 실행기는 출력을 평가하는 근거인 오라클과 평가할 대상인 SUT와의 입출력 인터페이스를 포함한다. 기능 모델은 입력에 따라 SUT가 어떤 동작을 할지 모델링한 것이다. 마지막으로 기능 요구사항은 HiL 테스트를 통해 검사할 기능의 입력에 대한 출력을 정의한 것이다.

차량용 소프트웨어의 동작을 검증하기 위해 자동화된 HiL 테스트 모의 환경이 가져야할 최소한의 특성을 정리하면 표 1과 같다.

표 1 자동화된 HiL 테스트 모의 환경 기본 특성

No.	특성	설명	
1	자동화	차량용 소프트웨어의 반복 테스트 수행을 위해 테스트 케이스 입력, 오라클, 수행, 평가의 자동화가 이루어져야 한다.	
2	정확성	시간	원하는 시간에 입력을 인가할 수 있어야 한다.
		평가	테스트 케이스의 통과 여부 판별에 오류가 없어야 한다.
		반복	반복 수행에도 동일한 결과를 보장해야 한다.
3	적합성	테스트 실행기는 차량용 인터페이스에 의해 동작해야 한다.	

첫째, 오라클을 통한 자동화가 되어야 한다. 자동화에는 여러 단계가 있다. 반복 테스트를 통해 차량용 소프트웨어를 검증하기 위해서는 최소한 테스트 케이스 입력, 오라클, 테스트 수행, 출력 평가의 자동화가 이루어져야 한다. 이러한 자동화된 테스트 수행을 위해서는 그림 1과 같이 테스트 실행기에 오라클이 포함되어야 한다. 오라클은 SUT의 기능을 시뮬레이션하여 결과를 예측할 수 있어, 테스트 입력에 대한 출력을 판단하는 근거가 된다. 따라서, 오라클을 포함한 테스트 실행기는 테스트 스크립트에 따라 자동화된 테스트 수행이 가능하다.

둘째, 정확성을 만족해야 한다. 차량용 소프트웨어가 대상이므로 테스트 환경의 정확성은 매우 중요하다. 이때, 정확성이란 기본적으로 시간, 평가, 반복의 정확성을 말한다. 시간의 정확성이란 원하는 시간에 SUT 및 오라클에 입력을 인가할 수 있음을 의미한다. 자동차는 고 신뢰가 요구되는 임베디드 시스템으로서 시스템의 적시성이 무엇보다 중요하다. 입력에 대해

정해진 시간 내에 동작했는지를 파악하기 위해서는 기준이 되는 입력 시간을 정확히 하는 것이 필요하다. 평가의 정확성이란 테스트의 케이스의 통과 여부 판별에 오류가 없음을 의미한다. 사소한 결함에 의해서도 치명적인 사고를 일으킬 수 있는 시스템인만큼 테스트 결과의 신뢰도 또한 높아야 하기 때문이다. 마지막으로 차량용 소프트웨어는 동일한 조건에서 동일하게 동작하는 결정적 특징을 가져야 한다. 반복의 정확성이란 반복 수행에도 동일한 결과를 보장함을 의미한다.

셋째, 적합한 차량용 인터페이스에 의해 동작해야 한다. 차량용 ECU 는 디지털 I/O 및 아날로그 I/O를 통해 외부의 환경값이나 제어값을 입력으로 받거나 다른 ECU와 FlexRay, CAN, LIN과 같은 통신 네트워크를 통해 연결되어 동작한다. 이렇게 실제 자동차에서 사용하는 인터페이스를 사용함으로써 테스트가 완료된 ECU를 그대로 실제 차량에 적용할 수 있게 된다.

이러한 기본 특성을 만족하는 자동화된 HiL 테스트 모의 환경은 그림 1의 구조로 구축하기 위해 SUT 기능 요구사항 정의, SUT 기능 모델링, 테스트 실행기 구현의 세 과정을 거쳐야 한다. 제안하는 방법은 실제 연구에 적용하여 구축 순서에 따라 자동화된 HiL 테스트 모의 환경을 구축한다.

3.2 차량용 소프트웨어 결함 지역화를 위한 HiL 테스트 환경 구축

이 장에서는 앞서 제안한 자동화된 HiL 테스트 모의 환경 구축 방법을 차량용 소프트웨어 결함 지역화 연구에 적용하였다.

실험에 사용한 SUT는 4 개의 ECU가 탑재된 실제 동작 가능한 모형 자동차 시스템이다. 노드(ECU) 1, 2, 3은 Freescale S12XF 프로세서와 OSEK OS 표준을 준수하는 RTOS를 탑재하였다. 노드 4는 Atmel ATMega128 프로세서를 탑재하였다. 각 노드는 개별 기능을 수행하며, 주요 기능은 표 2와 같다. 노드들은 CAN을 사용하여 네트워크를 구성하였다.

3.2.1 SUT 기능 요구사항 정의

자동화된 HiL 테스트 환경의 테스트 실행기는 앞서 설명과 같이 오라클이 있어야 한다. 이를 위해서는 SUT의 기능 요구사항 정의가 선행되어야 한다. 각각의 요구사항은 HiL 테스트를 통해 검사할 기능에 대한 입력과 출력으로 정의한다. 정의한 요구사항은 소프트웨어 요구사항 명세 표준(IEEE 830)에 따라 작성하며, 작성한 명세는 기능 명세 및 인터페이스 명세를 포함한다[16]. 그림 2는 표 2의 SUT 주요 기능 중 노드 1의 램프 제어 기능에 대해 작성한

표 2 SUT의 주요 기능

노드	MCU	주요 기능
1	Freescale S12XF	- 차체 조향: 조향 서보 모터 제어 - 램프 제어: 전조등, 지시등, 안개등 제어 - 주변 환경 센싱: 온도, 조도 센싱
2		- 컨트롤러와 통신: 제어값 수신 및 차량 정보 송신 - 차량 추진: DC 모터 제어를 통한 차량 추진 및 제동 기능
3		- 램프 제어: 후미등 및 방향 지시등 제어 - 주변 환경 센싱: 3축 가속도 - 후방 경고장치: 후진 시 후방 경고 장치
4	Atmel ATMega128	- 주변 환경 센싱: 4개의 초음파 센서를 이용한 장애물 거리 측정

3. 기능 요구사항

3.1. Node1

3.1.1. 전조등 및 안개등 기능

3.1.1.1. 기능 개요

설 명	전조등 및 안개등, 좌측 및 우측 방향 지시등을 제어한다.
입 력	CAN ID 가 CAN_LAMPMODE_SET 인 CAN Message 의 1 번째 Byte(LAMPMODE) 3.1.2 의 정의된 주변 조도 값인 ambientLightValue
출 력	전조등 및 안개등, 방향 지시등 출력

3.1.1.2. 상세 요구사항

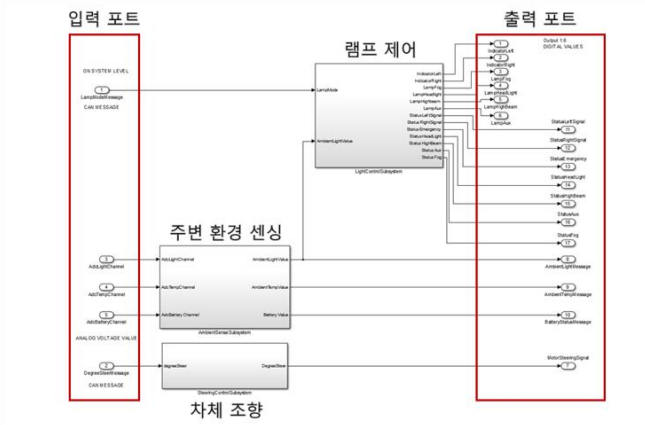
요구사항 ID	기능명	요구사항	Pseudo code
SRS-NODE1-HLAMP-F001	AUTO LAMP	LAMPMODE 가 LAMP_HEAD_AUTO 이면, 전조등 모드를 자동으로 설정한다. 1) 0 <= ambientLightValue < LIGHT_LOW 인 경우 상향등을 점등한다. 2) LIGHT_LOW <= ambientLightValue < LIGHT_MID 인 경우 전조등을 점등한다. 3) LIGHT_MID <= ambientLightValue < LIGHT_HIGH 인 경우 보조등을 점등한다.	If [LAMPMODE == LAMP_HEAD_AUTO] Then 전조등 모드 = 자동 1) If [0 <= ambientLightValue < LIGHT_LOW] Then 상향등 점등(Highbeam) 2) If [LIGHT_LOW <= ambientLightValue < LIGHT_MID] Then 전조등 점등(Headlight) 3) If [LIGHT_MID <= ambientLightValue < LIGHT_HIGH] Then 보조등 점등(Auxlamp)
SRS-NODE1-HLAMP-F002	전조등	LAMPMODE 의 전조등 BIT 가 전조등 신호(LAMP_HEAD_LIGHT_ON)와 일치하면(LAMPMODE&*LAMP_HEAD_LIGHT_ON* == *LAMP_HEAD_LIGHT_ON*), 전조등을 점등한다. 그렇지 않은 경우 소등한다.	If [LAMPMODE&LAMP_HEAD_LIGHT_ON == LAMP_HEAD_LIGHT_ON] Then 전조등 점등 Else 소등
SRS-NODE1-HLAMP-F003	상향등	LAMPMODE 의 상향등 BIT 가 상향등 신호(LAMP_HEAD_LIGHT_UP)와 일치하면(LAMPMODE&LAMP_HEAD_LIGHT_UP == LAMP_HEAD_LIGHT_UP), 상향등을 점등한다. 그렇지 않은 경우 소등한다.	If [*LAMPMODE&LAMP_HEAD_LIGHT_UP* == *LAMP_HEAD_LIGHT_UP*] Then 상향등 점등 Else 소등
SRS-NODE1-HLAMP-F004	안개등	LAMPMODE 의 안개등 BIT 가 안개등 신호(LAMP_FOG_LIGHT)와 일치하면(LAMPMODE&LAMP_FOG_LIGHT == LAMP_FOG_LIGHT), 안개등을 점등한다. 그렇지 않은 경우 소등한다.	If [*LAMPMODE&LAMP_FOG_LIGHT* == *LAMP_FOG_LIGHT*] Then 안개등 점등 Else 소등

그림 2 SUT 기능 요구사항 예(노드 1 램프 제어)

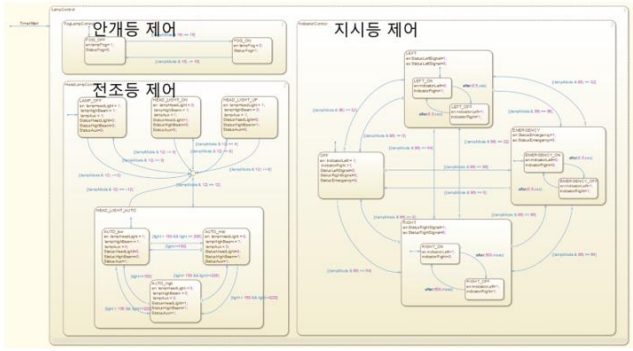
SRS(Software Requirements Specifications)이다. 이렇게 작성한 요구사항은 모델링에 앞서 그림 2와 같이 pseudo code나 use case diagram 등으로 나타낼 수 있다.

3.2.2 SUT 기능 모델링

자동화된 오라클을 구현하기 위해 필요한 모델은



(a) 노드 1



(b) 노드 1 램프 제어

그림 3 SUT 기능 모델 예(노드 1)

앞서 작성한 기능 요구사항 명세를 기반으로 실행 가능한 형태로 구현되어야 한다. 그림 3(a)는 노드 1의 기능 모델이며 그림 3(b)는 노드 1의 램프 제어 기능에 대한 상태 모델이다. SUT의 기능 모델은 Mathwork의 Matlab&Simulink Stateflow를 이용해 구현하였다[17].

노드에 대한 모델은 노드별 주요기능을 기준으로 블록을 나누고 각각의 입출력 포트를 나타낸 것이다. 주요 기능 모델은 각 기능을 상태 차트 다이어그램으로 나타냈다. 이를 통해 지시등 같은 계속 변화하는 출력을 하나의 상태로 표현하여 테스트 결과를 판단하기 용이하다.

3.2.3 테스트 실행기 구현

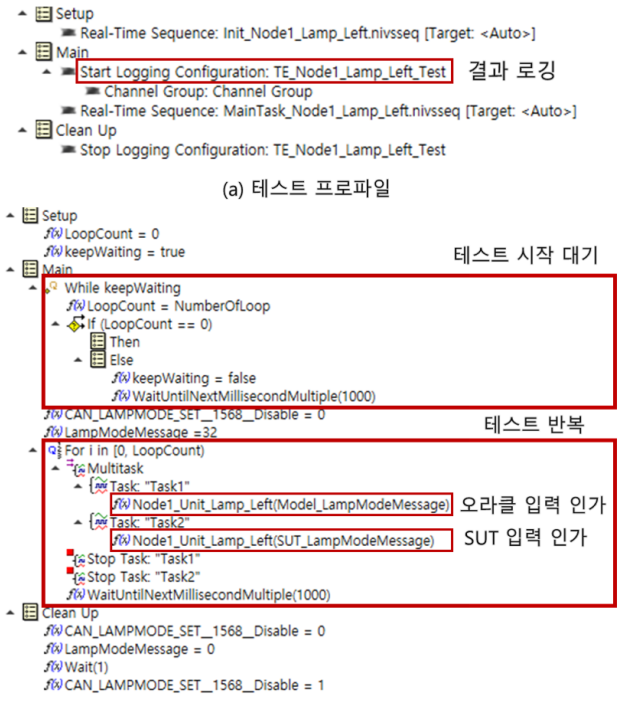
테스트 실행기는 그림 1과 같이 차량용 인터페이스와 자동화된 오라클 및 테스트 통과 여부 판별 모듈을 포함해야 한다. 또한 세 가지 정확성(시간의 정확성, 평가의 정확성, 반복의 정확성)을 만족하도록 구현되어야 한다. 테스트 실행기는 National Instruments의 VeriStand[18]를 사용하여 구현하였다. 테스트 실행기는 테스트 케이스 입력을 오라클에 인가하여 시뮬레이션한 결과(Expected Output)와 SUT에 동일한 입력을 인가하여 얻은 출력(Generated Output)을 비교하여 Pass/Fail 여부를 판별한다.

테스트 실행기의 장비 구성은 표 3과 같다. 테스트 실행기의 입출력 인터페이스의 경우 VeriStand가 설치된 PC에 National Instruments의 DAQ 장비를 설치하여 SUT와 연결한 후 각각의 입출력에 대하여 채널을 설정하였다. 특히 CAN 채널 설정 시 CAN 데이터베이스에 메시지 규격을 정의해둔다. 오라클의 경우 Matlab Stateflow로 구현한 모델을 빌드하여 시뮬레이션 모델로 추가함으로써 구현하였다.

표 3 테스트 실행기의 장비 구성

구분	장비 구성
SUT 인터페이스	NI cDAQ-9174 USB 4 Slot-Chassis - NI 9403 32 Digital I/O DAQ - NI 9862 High-Speed CAN Interface - NI PCI-6733 16-bit, 8 Analog Output DAQ
테스트 오라클	Matlab 2014b 상에서 구성한 Node 1, 2, 3, 4의 Simulink Model
테스트 수행 프레임워크	NI VeriStand 2017
PC 환경	Xeon E3-1275v5 3.7Ghz, 64GB RAM, Windows 10 Pro

구현한 테스트 실행기를 동작시키기 위해서는 미리 정의한 테스트 케이스를 정해진 주기에 맞춰 오라클과 SUT에 인가하고 결과를 기록하는 일련의 흐름을 구현해야 한다. VeriStand의 자극 프로파일을 통해 테스트의 반복 수행과 입력 주기 설정을 구현하고 실행할 수 있다. 그림 4는 노드 1의 램프 제어기능을 테스트하기 위한 테스트 실행기를 구현한 것이다. 그림



(a) 테스트 프로파일

(a) 테스트 시퀀스

그림 4 테스트 실행기 구현 예(노드 1 램프 제어)

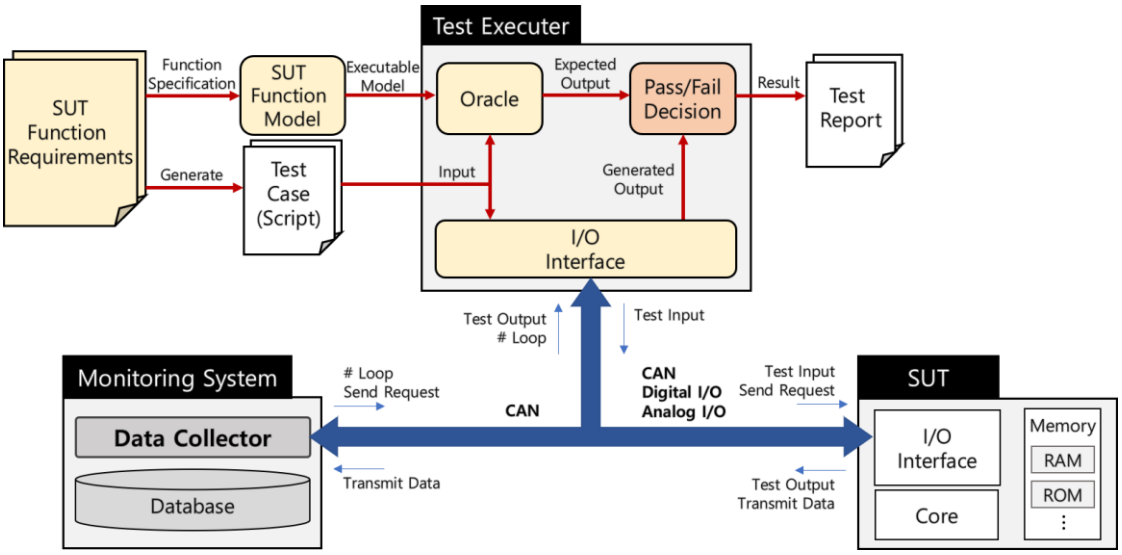


그림 5 차량용 소프트웨어 결함 지역화 연구를 위한 자동화된 HiL 테스트 모의 환경 구조

4(a)는 테스트를 실행하고 결과를 로깅하기 위한 프로파일이며, 그림 4(b)는 실제 입력을 인가하고 반복 수행하는 흐름(테스트 시퀀스)을 구현한 것이다.

4. 통합 테스트 환경 구축 및 실험

이번 장에서는 앞서 구축한 차량용 소프트웨어 테스팅을 위한 자동화된 HiL 테스트 모의 환경을 확장하여 차량용 소프트웨어 결함 지역화 연구를 위한 통합 테스트 환경을 구축하고 실험을 통해 평가한다.

4.1 통합 테스트 환경 구축

적용하고자 하는 차량용 소프트웨어 결함 지역화 연구는 블랙박스 환경에서 이용 가능한 정보인 메모리 갱신 정보를 이용하여 결함 위치 정보를 제공하기 위한 것이다[7, 19]. 이를 위해 동일한 테스트 스크립트를 반복하여 각 시뮬레이션마다 서로 다른 메모리 영역을 관찰하여 전체 데이터를 수집하는 기법을 사용한다[15]. 차량용 소프트웨어의 결함 지역화 연구에 적용하기 위해 앞서 구현한 테스트 실행기를 SUT, 모니터링 시스템과 연결하여 통합 테스트 환경을 구축하였다. 그림 5는 이러한 통합 테스트 환경의 구조이다. 그림 6은 실제 구축한 통합 테스팅 환경이다. 모니터링 시스템은 메모리 데이터를 수집하는 시스템이다. 이는 메모리 데이터를 수집하기 위한 데이터 수집기와 데이터 저장을 위한 DB로 구성된다. 모니터링 시스템은 테스트 실행기와 동일한 PC 환경에서 동작하며 장비 구성은 표 4와 같다.

4.2 실험 및 평가

구축한 자동화된 HiL 테스트 환경의 평가를 위해

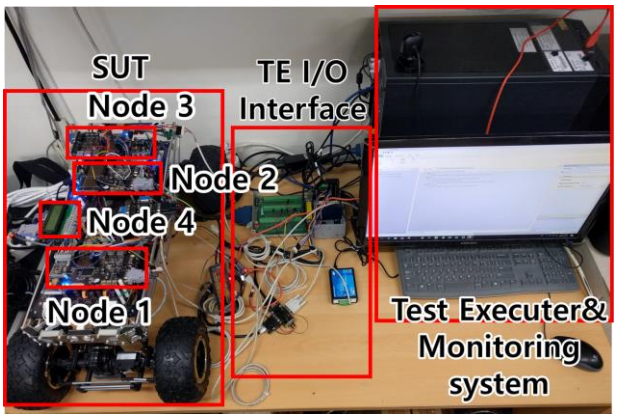


그림 6 통합 테스팅 환경

표 4 모니터링 시스템의 장비 구성

구분	장비 구성
PC 환경	Xeon E3-1275v5 3.7Ghz, 64GB RAM, Windows 10 Pro
DB	MariaDB 10.0.14
CAN 인터페이스	Maeol USB-CAN Analyzer

실제 테스트를 수행하였다. 표 5는 테스트하고자 하는 SUT의 기능과 구축한 테스트 환경을 통한 테스트 수행 가능 여부를 나타낸 것이다. 실험을 위해 표 2에서 정리한 SUT 중요 기능으로부터 총 22개의 세부 기능을 도출하였다. 20개의 단위 테스트, 2개의 통합 테스트를 수행한 결과, 입력에 대한 출력을 확인할 필요가 없는 4개의 단순 센싱 기능을 제외한 모든 기능의 입력에 대한 출력을 테스트할 수 있었다. 표 5에서 노드 4의 경우, 노드 3의 후방 경고 기능에 사용되는 거리 센싱을 위한 ECU이므로 평가에서는 제외되었다. 테스트 가능 여부는 다음과 같은 세 가지 적합성 확인을 통해 판단하였다.

표 5 테스트 대상 기능

노드	주요 기능	기능	테스트 여부	비고
1	램프 제어	상향등	O	
		하향등	O	
		자동 전조등	O	조도와 연계
		좌측 지시등	O	
		우측 지시등	O	
		안개등	O	
		비상등	O	
	조향	조향 제어	O	
	센싱	조도	△	단순 센싱
		온도	△	단순 센싱
2	통신	정보 송수신	O	
	차량 추진	전진모드	O	
		후진모드	O	
		주차모드	O	
3	램프 제어	브레이크등	O	
		긴급제동신호	O	
		좌측 지시등	O	
		우측 지시등	O	
		비상등	O	
	센싱	3축 가속도	△	단순 센싱
	경고	후방 경고	O	거리와 연계
4	센싱	거리	△	단순 센싱

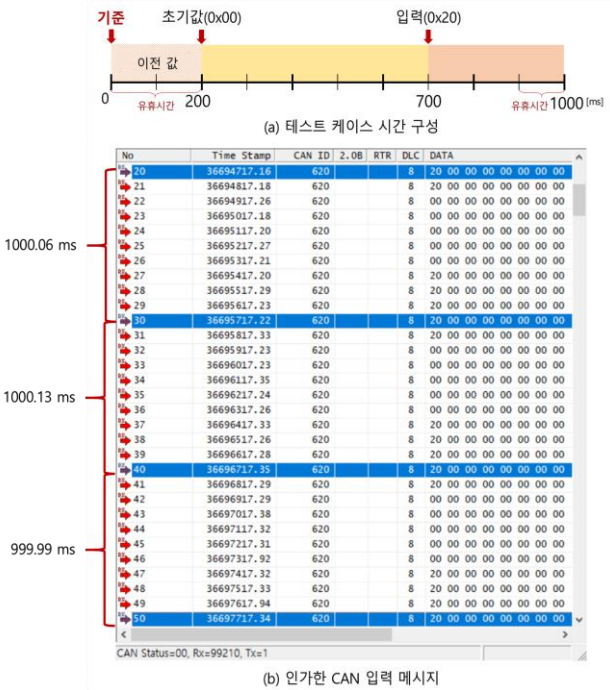


그림 7 테스트 케이스 입력 확인 (노드 1 램프 제어-좌측 지시등)

구축한 테스트 환경이 시간의 정확성을 만족하는지 확인하기 위하여 그림 7(a)와 같은 테스트 케이스를 생성하여 동일한 시간 간격으로 반복 수행하였다. 하나의 테스트 케이스는 초기화-입력으로 구성하였다. 그림 7(b)는 실제 SUT에 인가한 CAN 입력 메시지를 관찰한 것으로 (a)에 표시된 시점을 기준으로 다음 입력까지의 시간 간격을 측정한 것이다. 실험 결과, 1초의 테스트 시간에서 오차가 평균 0.06ms가 나왔다. 이것은 0.006%의 오차로 테스트 환경은 시간 정확성을 만족함을 확인하였다.

평가의 정확성을 확인하기 위하여 동일한 테스트 케이스에 대하여 정상 동작하는 SUT와 결함이 주입된 SUT의 테스트 수행 결과를 비교하였다. 테스트 통과(pass)는 오라클의 예상 결과와 실제 출력이 같은 것이며, 다를 경우 실패(fail)이다. 실험 결과, 표 6과 같이 정상 SUT에 대해서는 모든 케이스에 대해서 통과하였고, 각각의 케이스에 실패하도록 결함이 주입된 SUT는 모든 케이스에서 실패하였다.

표 6 평가 정확성의 만족 조건

	Pass (Predicted)	Fail (Predicted)
Pass (Actual)	100%	0%
Fail (Actual)	0%	100%

그림 8은 노드 1의 램프 제어 기능 중 지시등 및 비상등에 대한 테스트 수행 결과 예시이다. 그림 8(a)는 테스트 보고서로 기능과 관련된 반복 횟수, 입력, 예상 결과와 SUT 출력, 테스트 결과가 포함된다. 예를 들어 그림 8(a)는 42회 반복에 입력은 96(0x60)이다. 여기서 0x60은 비상등을 켜는 명령이다. 오라클의 기대 출력은 비상등이 켜져 있고 실제 SUT 또한 비상등이 켜져 있는 것을 확인할 수 있다. 따라서 결과는 1(pass)로 출력된다. 그림 8(b, c)는 정상 상태와 결함 주입 후 테스트 보고서이다. 그림에서 확인할 수 있는 것은 정상일 때, 오라클의 기대 출력이 바뀐 후 일정 시간이 지나서 SUT의 출력도 변한 것을 확인할 수 있다. 따라서 모든 경우에서 통과하였다. 그러나 결함 주입한 경우 기대 출력의 변경에도 불구하고 좌측 지시등이 켜지지 않은 것을 확인할 수 있다. 그 결과, 실패 판정을 받았다. 이를 통해 평가의 정확성을 만족함을 확인하였다. 연속성을 가진 값의 평가의 경우, 오라클의 기대 출력은 목표값이 되고 SUT의 출력이 목표값을 달성했는지 여부로 판단할 수 있다.

반복의 정확성은 앞서 기능이 정상적으로 동작함을 확인한 상태에서 지정한 반복 횟수만큼 입력이 인가되었는지 확인하여 평가하였다. 그림 9는 노드 1 램프 제어 기능 중 좌측 지시등을 대상으로 수행한 결과이다. 1초의 테스트 시간에서 반복 횟수를 42회로 지정했을

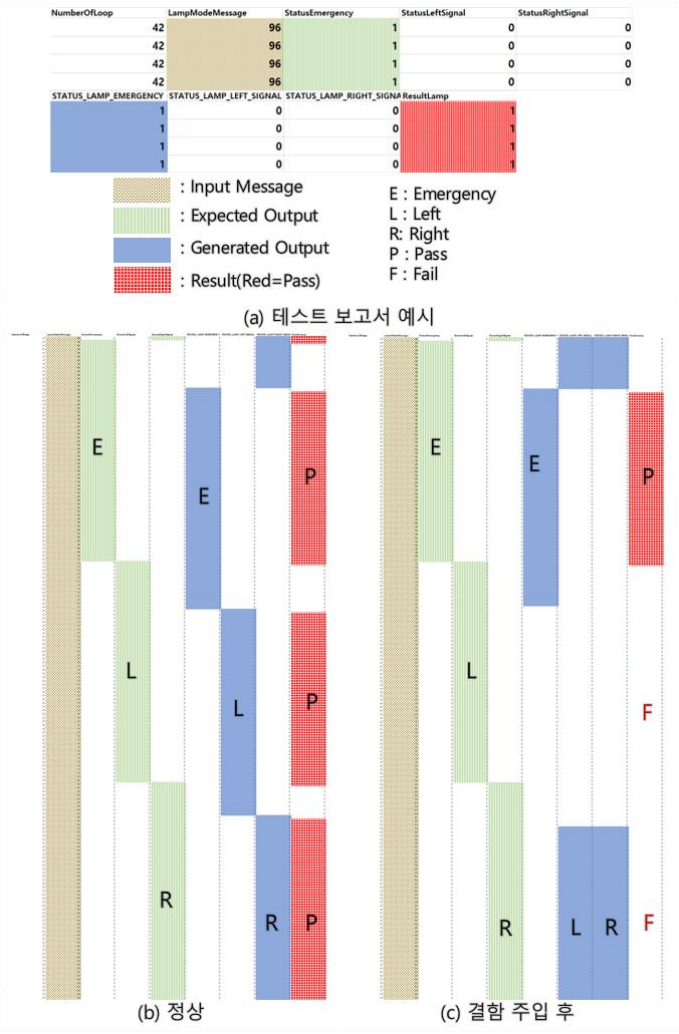


그림 8 테스트 수행 결과
(노드 1 램프 제어-지시등 및 비상등)

때, 각 수행은 시간 정확성을 만족하였다. 전체 수행 시간은 테스트 시작부터 종료까지 총 42,001.55ms로 1.55ms의 차이가 발생하였다. 이는 0.155%의 오차로 테스트 환경은 반복의 정확성을 만족하는 것을 확인하였다.

차량용 인터페이스를 사용하여 구축한 HiL 테스트 모의 환경에서 모델을 이용한 오라클을 통해 자동으로 테스트를 수행하였다. 실험을 통해 단순 센싱 기능을 제외한 모든 경우에서 세 가지 정확성(시간, 평가, 반복)을 만족하는 것을 확인하였다. 따라서 제안하는 자동화된 HiL 테스트 모의 환경의 세 가지 기본 특성을 모두 만족하며 이를 통해 제안한 자동화된 HiL 테스트 모의 환경으로 차량용 소프트웨어를 테스트 할 수 있음을 확인하였다.

구축한 통합 테스팅 환경은 기존 상용 HiL 테스트 시스템을 사용하지 않고 적정 규모의 테스트 도구와 설비를 사용하였다. 실험을 통해 자동화가 이루어 졌음을 확인하였다. 그림 10은 통합 테스트 환경의 모니터링 시스템이 수집한 데이터 중에서 8600번지의

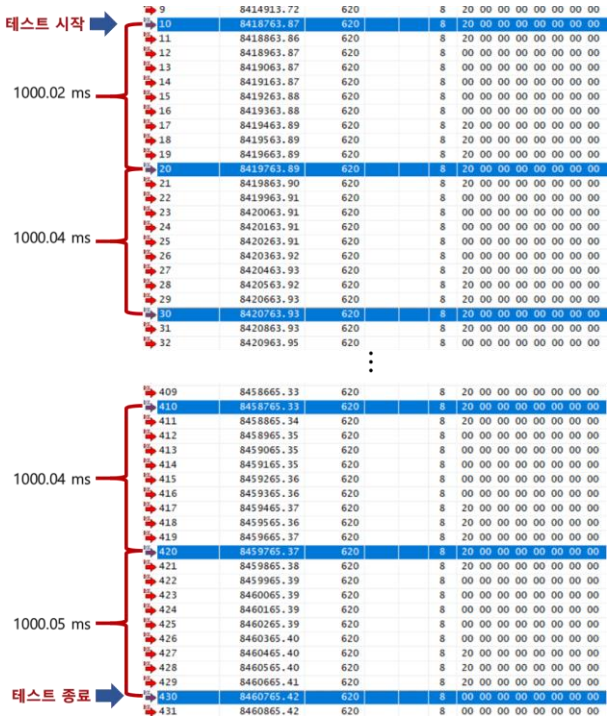


그림 9 테스트 반복 수행 확인
(노드 1 램프 제어-좌측 지시등)

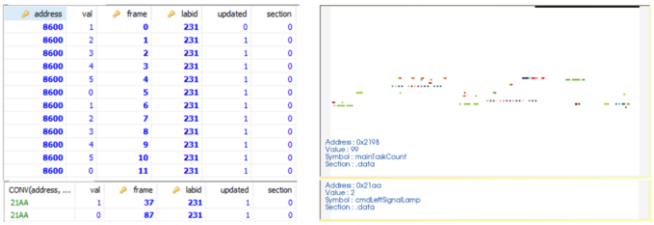


그림 10 모니터링 시스템이 수집한 데이터 예시

메모리 값의 예시와 그것을 시각화한 예시이다. 이를 통해 디버깅 정보 제공이 가능해짐을 알 수 있다. 이로써 HiL 테스트 환경을 구축하기 위한 고려 사항을 반영한 자동화된 HiL 테스트 모의 환경을 구축 하였다.

5. 결론

본 논문에서는 차량용 소프트웨어의 테스팅을 위한 자동화된 HiL 테스트 모의 환경 구축 방법을 제안하였다. 제안하는 방법에서는 먼저 자동화된 HiL 테스트 모의 환경에 대한 기본 특성을 설명하였다. 다음, 차량용 소프트웨어의 결함 지역화 연구에 적용하여 SUT의 기능 요구사항 정의, SUT의 기능 모델링, 테스트 실행기 구현의 세 단계에 따라 자동화된 HiL 테스트 모의 환경을 구축하였다. 실제 테스팅을 위하여 SUT와 모니터링 시스템을 포함한 통합 테스트 환경을 구축하여 실험한 결과, 제안한 테스트 환경은 자동화된 HiL 테스트 모의 환경의 기본 특성을 모두 만족하는 것을 확인하였다. 다만, 본 논문은 테스팅 환경 구축에

집중하여 테스트 케이스 생성 방법이나 오라클 자체의 검증 부분은 고려하지 않은 한계를 가진다.

제안한 방법을 통해 자동화된 HiL 테스트 모의 환경을 구축함으로써 비교적 저렴한 비용으로 자동화된 HiL 테스트를 수행할 수 있었다. 또한, 결함 지역화를 위한 모니터링 시스템을 추가함으로써 세 가지 고려 사항(도구와 설비, 자동화 수준, 디버깅 정보 제공 능력의 제약 사항)이 모두 고려된 환경을 구축하였다.

향후에는 HiL 테스트 모의 환경을 기반으로 차량용 소프트웨어의 결함 위치 추정 연구를 진행하고자 한다.

6. 참고문헌

[1] ISO 26262-6:2011, Road vehicles - Functional safety - Part 6: Product development at the software level, 2011

[2] Martin Gomez, Hardware-in-the-loop simulation, Embedded Systems Programming Vol. 14, No. 13, pp. 38-49, 2001

[3] R Isermann, J Schaffnit, S Sinsel, Hardware-in-the-loop simulation for the design and testing of engine-control systems, Control Engineering Practice, Vol. 7, Issue. 5, pp. 643-653, 1999

[4] Yun CHEN, et al, dSPACE Based HIL Simulation Platform for Missile Control System, Information Technology and Intelligent Transportation Systems: Proceedings of the 2nd International Conference on Information Technology and Intelligent Transportation Systems (ITITS 2017), pp. 283-289, 2017

[5] Susanne Köhl, and Jegminat Dirk, How to do hardware-in-the-loop simulation right, SAE Technical paper 2005-01-1657, 2005

[6] R. Boot, J. Richert, H. Schutte and A. Rukgauer, Automated test of ECUs in a hardware-in-the-loop simulation environment, Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design (Cat. No.99TH8404), pp. 587-594, 1999

[7] Choi Ki-Yong, et al, HiL Test Based Fault Localization Method Using Memory Update Frequency, Advances in Computer Science and Ubiquitous Computing, Springer Singapore, pp. 765-772, 2015

[8] 고영진 외, 상용차용 HILS 기반 능동형 공기현가 시스템의 가상 Components 개발에 관한 연구, 한국자동차공학회논문집, Vol. 21, No. 2 pp. 26-36, 2013

[9] 변진구 외, 무인항공기 비행제어 HILS 시험환경 연구, 한국항공우주학회지 Vol. 44, No. 4, pp.316-323, 2016

[10] 변재훈, 다단 자동변속기 소프트웨어 검증을 위한 Hardware-In-the-Loop System 환경의 구성, 한국자동차공학회 학술대회 및 전시회, pp.1399-1401, 2014

[11] SCALEXIO, https://www.dspace.com/en/inc/home/products/hw/simulator_hardware/scalexio.cfm

[12] dSPACE TargetLink, ttp://www.hancommds.com/solutions/?no=320

[13] LABCAR 컴퍼넌트 개요, HiL 시스템, https://www.etas.com/ko/products/solutions_labcar_component_overview.php

[14] Standardized & Modular hiL Test System Family, https://www.adas-iit.com/hardware-in-the-loop/hil-test-systems/standardized_test-system_family/#measurementtechnology

[15] Jeong-Woo Lee, Ki-Yong Choi, and Jung-Won Lee, Collecting Big Data from Automotive ECUs beyond the CAN Bandwidth for Fault Visualization, Mobile Information Systems, 2017

[16] IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications, 1998

[17] Stateflow - MATLAB & Simulink, https://kr.mathworks.com/products/stateflow.html

[18] NI VeriStand - http://www.ni.com/veristand/ko/

[19] Kim, Kwanhyo, Ki-Yong Choi, and Jung-Won Lee, Fault Localization Method by Partitioning Memory Using Memory Map and the Stack for Automotive ECU Software Testing, Applied Sciences, 6(9), 266, 2016

융복합 연구 증진과 산학협력 활성화를 위한 연구자 네트워크 시스템 개발

황상원, 서강원, 류원철, 남영광
연세대학교 컴퓨터정보통신공학부
arsenal@yonsei.ac.kr, seogs12@naver.com, hkjkkl@naver.com, yknam@yonsei.ac.kr

Development of a System to Promote Convergence Research and Activate University-industry Collaboration by Mapping Diverse Research Area

Sangwon Hwang, Kangwon Seo, Wonchul Ryu, Youngkwang Nam
Dept. of Computer & Telecommunications Eng., Yonsei University

요 약

본 연구에서는 융복합 기술의 발전에 영향을 줄 수 있는 대학의 역할에 주목하여, 대학 내의 연구자와 융복합 산학협력을 원활히 수행할 수 있도록 정보를 제공하는 연구자 네트워크 시스템을 구축하였다. 연구자 간의 관계를 분석하기 위해서 본 시스템에서는 다양한 학문 분야의 논문 약 23,000개와 과제 약 65,000개의 제목, 초록, 키워드를 바탕으로 형태소 분석과 토픽 모델링을 수행하였다. 추출된 모델링 결과는 사용자 키워드와 매칭시켜 연구자 키워드 셋을 생성하고, 모든 연구자와의 유사도를 측정하여 연관 관계 네트워크를 생성한다. 실험 결과, 본 연구에서 제안된 기법이 실제 응용에서 보다 유용하게 사용됨을 확인할 수 있었다.

1. 서론

과학 기술적인 측면에서 융복합 기술은 현대 과학기술 혁신의 보편적인 현상으로 자리 잡았고, 최근에 개발 및 상업화에 성공하고 있는 대부분의 제품과 서비스는 융복합 기술의 산물로 인식되고 있다.[1] 기업과 대학은 융복합 기술 혁신의 핵심 주체로서 주목을 받고 있으며, 이에 대한 많은 정책적인 지원을 제공받고 있다. 본 연구에서는 이와 같은 융복합 기술의 발전에 영향을 줄 수 있는 대학의 역할에 주목하여, 기업으로 하여금 대학과의 융복합 산학협력을 원활히 수행할 수 있도록 정보를 제공해 주는 연구자 네트워크를 구축하고자 하였다.

본 연구에서는 전공이 서로 다른 연구자의 논문 및 과제의 제목과 초록, 그리고 사용자 정의 키워드를 비교하여 연구자 간의 유사도를 분석 하였으며, 이를 기반으로 연구자 간의 융합 연구 가능성을 분석하고, 잠재적 융복합 연구자에 대한 가시화 맵을 구축하였다.

본 논문에서 제안한 방법을 평가하기 위해 시스템 구축에 사용된 다양한 알고리즘들을 적절히 조합한 다수의 시스템을 생성하여, 실험에 참가한 지원자들의 만족도를 조사하였다. 평가 결과, 본 시스템을 구축하기 위해 다양한 알고리즘을 조합하였을 경우에 가장 만족도가 높은 것으로 나타났다.

2. Researcher Network System

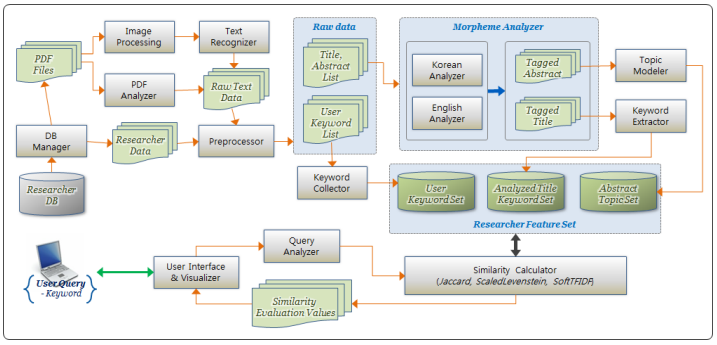


그림 1. 연구자 네트워크 시스템 구조도

연구자 맵 시스템의 구조는 그림 1과 같다. 데이터 추출 부분에서는 연구자 실적 DB를 활용하여 연구자가 작성한 논문 및 과제에서 유용한 정보를 추출한다. 연구자 실적 DB에는 연구자, 논문, 과제 정보가 저장되어 있고, 논문 원본 파일이 pdf 형태로 저장되어 있다. DB Manager는 DB로부터 연구자의 기본 정보를 추출하고, PDF 파일은 PDFBox를 이용하여 그림 혹은 텍스트 형태의 PDF 파일로부터 DB에 누락된

키워드/초록 정보를 추출한다.

Morpheme Analyzer에서는 논문 및 과제의 제목/초록에 대한 형태소 분석을 수행하는데, 문장에 대한 품사 태깅을 수행하고, 태그 정보가 NN*, JJ인 의미 있는 단어들을 추출한다.

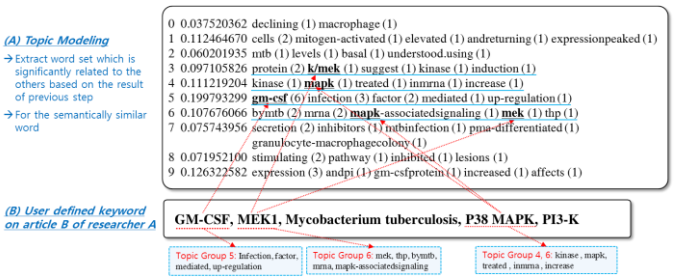


그림 2. 초록에 대한 토픽 모델링 결과(A)와 연구자가 직접 설정한 키워드와의 매칭 그룹 생성(B) 예제

추출된 결과는 LDA 토픽 모델링[2]을 수행하기 위해 활용되며, 그림 2-(A)와 같은 결과가 출력된다. 토픽 모델링을 수행한 결과를 확인하면, 그림 2-(B)와 같이 연구자가 직접 설정한 해당 논문의 중요 키워드인 “gm-cfs”, “mek", “mapk”가 나타나는 것을 확인할 수 있고, “mapk”와 집합을 이루고 있는 “kinase” 또한 의미 있는 단어임을 알 수 있다. 이와 같은 키워드 토픽 집합은 사용자 키워드 및 형태소 분석된 키워드 집합과 함께 researcher feature set으로 저장된다. 이와 같은 방법으로 전 연구자의 논문/과제에 대해서 분석을 수행하고, 각 연구자에 대한 feature set을 추출하여 저장소를 구축한다. 이 정보를 활용하여 연구자 간의 유사도를 계산한다.

표 1. 유사도 계산 결과 예

Auth.	KW1	KW2	KW3	Jacc.	S.Leven.	S. TFIDF
Auth1	HSP70/CHIP	P42 Ebp1	P85 subunit	0.2632	0.1887	0.4167
Auth2	Hepatocellular carcinoma	Perifosine	PI3K/Akt pathway	0.2	0.2048	0.3333
Auth3	Akt	Bim	CHOP	0.1875	0.3019	0.3273
Auth4	c-Abl	p38 MAPK	Prostate cancer	0.1667	0.1887	0.2887

3. 구현 및 결과

본 시스템은 Java 1.8 기반으로 구현하였으며, 토픽 모델링은 MALLET 툴킷을 사용하였다. PDF 분석은 PDFBox를 활용하였고, GUI는 JQuery와 JSP로 구성하였다. 서버는 Intel Core i7, 32GB RAM, 256GB SSD으로 구성되었다. 토픽 모델링을 위한 토픽 집합의 수는 10, 토픽별 단어 수는 5, 반복 샘플링 횟수는 1,000회, 유사도 임계값은 25% 로 설정하였다. 실험은 논문 23,086개, 과제 65,567개를 대상으로 수행하였다. 구현 화면은 그림 3과 같다.

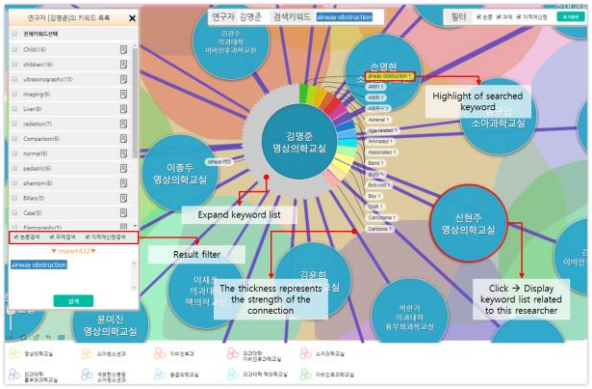


그림 4. 연구자 네트워크 화면 예제

실험을 위해 각 알고리즘을 조합한 시스템을 구축하였고, 실험 참가자가 직접 검색 시스템을 활용해서 만족도를 평가하였다. 평가 결과, Jaccard 등의 단일 알고리즘 기반 시스템보다 본 연구에서 제시한 Jaccard + Scaled Levenshtein + Soft TF/IDF + LDA 기반의 시스템이 만족도가 높은 것으로 나타났다.

4. 결론

본 연구에서는 산학협력 융복합 연구를 위해 전공이 다른 연구자 사이의 연구 유사성을 측정하는 시스템을 개발하였다. 시스템 구축을 위해 연구자가 가지고 있는 논문 및 과제의 제목, 키워드, 초록 정보를 활용하였고, 의미적 유사성 검증을 위해 Jaccard, Scaled Levenshtein, Soft TF/IDF 알고리즘과 LDA 토픽 모델링을 결합하여 결과를 추출하였다. 수행 결과는 GUI 화면으로 제시하였다. 향후 연구로, 실시간으로 입력되는 실적 정보들에 대한 비교를 수행하고 단순 단어의 분석이 아닌 의미 있는 단어 쌍과 개체명 식별을 통해 키워드를 추출하여 비교한다면 전반적인 시스템의 완성도가 높아질 것으로 보인다.

Acknowledgement

이 논문은 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연 구임 (No.NRF-2014M3C4A7030505)

참고문헌

[1] 이공래, 성태경, “융합기술 혁신을 위한 산학협력 활성화 방안”, 정책연구, 2009.

[2] Blei, D., Ng, A., Jordan, M., “Latent Dirichlet Allocation”, Journal of Machine Leaning Research 3, pp. 993-1022, MIT Press, 2003.

안전한 소프트웨어 집약적 시스템을 위한 시스템 이론적 안전성 분석에서의 위험도 평가

정대희 ○, 권기현
경기대학교 컴퓨터과학과
Jdhpen100@kgu.ac.kr, khkwon@kgu.ac.kr

Risk Assessment at System Theoretic Safety Analysis for Safe Software-intensive System

Daehui Jeong ○, Gihwon Kwon
Department of Computer Science, Kyonggi University

요 약

소프트웨어의 규모가 커지고 복잡할수록, 소프트웨어로 인한 위험원을 관리하는 안전성 분석이 더욱 중요하다. STPA(System Theoretic Process Analysis)는 시스템 이론에 기반한 안전성 분석 기법으로 소프트웨어의 안전성 분석에 널리 사용되고 있다. 비록 STPA가 기존의 신뢰성 이론 기반의 안전성 분석 방법에 비해서 소프트웨어적인 측면을 잘 다루고 있다고는 하지만, STPA는 위험도 평가를 다루지는 않는다. 본 논문에서는 STPA에 위험도 평가를 추가하여 소프트웨어의 안전성을 분석 및 관리하고자 한다. 제안하는 방법은 사례로 선정한 모형 철도 시스템에 적용해 그 효과성을 확인한다.

1. 서 론

소프트웨어의 규모가 커지고 복잡해지면서 소프트웨어로 인한 시스템의 위험원을 파악하기가 어려워지고 있다[1]. 만약 소프트웨어로 인한 위험원이 파악되지 못하면 시스템이 위험한 상황에 직면하거나, 사고로 이어질 수 있다. 특히 심각한 인명, 환경, 그리고 재산 피해를 발생시키는 안전 필수 시스템에서는 더욱 중요시된다.

안전성 분석은 시스템 또는 소프트웨어가 가질 수 있는 위험원을 식별하여 분석하고 위험도를 평가해 체계적으로 위험원을 관리하는 활동이다[2]. 안전성 분석을 통해 식별되고 분석된 위험원은 어느 정도의 위험도를 가졌는지에 따라 우선 순위가 정해지고 제어 조치가 시행된다. 만약 제어 조치가 위험도를 허용 수준까지 낮추지 못할 경우에는 위험도가 허용 수준에 도달할 때까지 제어 조치를 추가한다.

시스템 이론적 안전성 분석 기법인 STPA(System Theoretic Process Analysis)는 소프트웨어의 안전성을 분석한다[3]. STPA 는 시스템에서 차지하는 소프트웨어의 역할에 대한 포괄적인 이해를 돕고 소프트웨어로 인해 발생되는 시스템의 위험원을 식별 및 분석한다. 하지만 STPA 에서는 위험도 평가는 다루어지지 않고 있기 때문에 위험원이 어떻게 관리되고 있는지 파악하기 어렵다.

본 논문에서는 시스템 이론적 안전성 분석에서의 위험도 평가를 위해 RSTPA(Risk-driven System Theoretic Process Analysis)를 제안하여 소프트웨어의 안전성을 분석 및 관리할 수 있도록 돕고자 한다. 이를 위해 시스템 명세로부터 시스템 기능을 파악해 제어 구조 다이어그램을 작성하고 UCA(Unsafe Control Action)를 식별한다[4]. 이후 UCA 마다 FTA(Fault Tree Analysis)와 ETA(Event Tree Analysis)를 수행하고 발생 빈도와 심각도를 추정해 위험도를 평가한다[5,6,7]. 그리고 분석의 결과로 소프트웨어 안전 요구사항을 도출한다.

본 논문에서는 제안하는 방법을 모형 철도 시스템에 적용해 효과성을 확인하였다. 철도 시스템은 안전 필수 시스템의 하나로, 사고 발생 시 심각한 피해가 발생하기 때문에 안전성의 확보가 요구된다. 분석을 통해 도출된 소프트웨어 안전 요구사항은 시스템에 반영해 소프트웨어 오류가 발생하여도 사고를 방지 또는 회피하는 것을 확인하였다.

본 논문에서는 제안하는 방법을 통해 두 가지의 효과를 기대한다[8,9]. 먼저 제어 조치의 우선 순위를 결정할 수 있다. 평가된 위험도에 따라 우선 순위를 결정하는 것이 가능하기 때문이다. 또다른 효과로는 잔류 위험원의 정보를 명확히 제공할 수 있다. 위험원의 위험도를 제로(Zero)로 만드는 것은 불가능하기 때문에, 위험도를 통해 얼마만큼의 위험원이 잔류 되었는지를 알 수 있다.

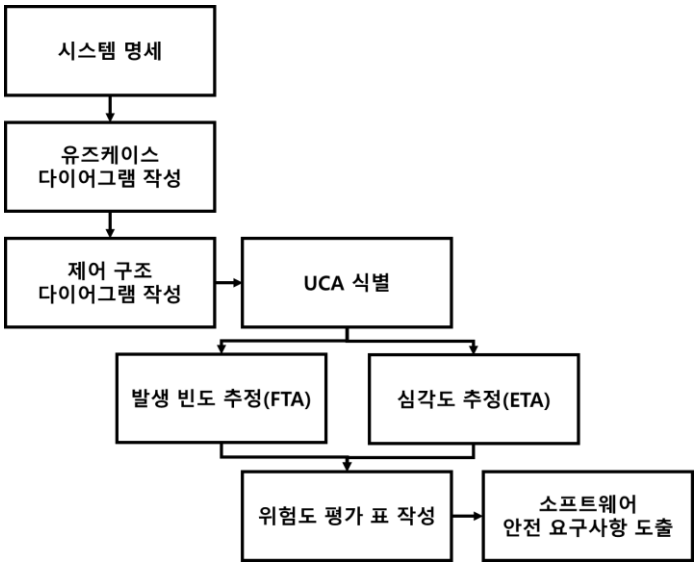
"본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성지원사업의 연구결과로 수행되었음" (IITP-2017-2015-0-00445)

2. STPA

STPA 는 기존의 안전성 분석 기법들이 신뢰성 이론에 기반한 것과는 다르게 시스템 이론에 기반한 안전성 분석 기법으로 소프트웨어의 안전성을 분석한다. STPA 는 시스템이 가지는 제어 구조를 바탕으로 소프트웨어가 시스템에서 차지하는 역할에 대한 포괄적인 이해를 돕는다. 또한 시스템의 구성 요소 간의 상호 작용에 중점을 두고 소프트웨어로 인해 발생되는 시스템의 위험원을 식별 및 분석한다.

[3]에서 소개되고 있는 STPA의 절차는 크게 기초 자료 수집, UCA 식별, 사고 시나리오 식별의 세 단계로 이루어진다. 먼저 기초 자료 수집에서는 분석을 수행하기 위해 기초가 되는 자료를 수집한다. 시스템 수준의 설계 문서를 바탕으로 분석의 범위를 설정하고 시스템에서 발생 가능한 사고와 위험원을 식별한다. 그리고 제어 구조 다이어그램을 작성하고 제어 명령을 식별한다. UCA 식별에서는 제어 명령 별로 네 가지의 키워드를 적용하여 UCA를 식별하고 초기 안전 요구사항을 식별한다. 이 때 적용되는 키워드는 'Wrongly provided', 'Not provided', 'Provided wrong timing or order', 'Stopped too soon or applied too long'의 네 가지 이다. 사고 시나리오 식별에서는 제어 변수와 제어 값을 포함하는 제어 모형을 식별해 제어 구조 다이어그램을 재작성 하고 문맥 표를 작성한다. 문맥 표가 작성되면 이를 이용해 안전 요구사항을 재작성하거나 추가한다.

하지만 STPA에서는 식별되고 분석된 위험원에 대한 위험도 평가는 다루어지지 않고 있다. 이는 위험원이 어떻게 관리되고 있는지 파악하기 어렵게 만든다. 본 논문에서는 시스템 이론적 안전성 분석에서 위험도를 평가하기 위해 RSTPA를 제안한다.



<그림 1> RSTPA 절차

3. RSTPA

RSTPA 를 수행하기 위해서는 <그림 1>과 같은 절차의 안전성 분석이 수행된다. 안전성 분석은 크게 위험원 식별, 위험원 분석, 위험도 평가, 소프트웨어 안전 요구사항 도출의 네 가지로 이루어진다. 위험원 식별에서는 시스템 명세로부터 시스템 기능을 파악하여 제어 구조 다이어그램을 작성하고 UCA 를 식별한다. 위험원 분석에서는 UCA 마다 FTA 를 수행해 발생시키는 원인을 분석하고, ETA 를 수 행해 발생되는 영향을 분석한다. 위험도 평가에서는 FTA 의 수행 결과로 도출되는 발생 빈도와 ETA 의 수행 결과로 도출되는 심각도를 이용하여 각 UCA 에 대한 위험도를 평가한다. 마지막으로 안전 요구사항 식별에서는 UCA 와 발생시키는 원인 별로 방지 또는 회피하기 위한 안전 요구사항을 식별한다.

3.1 위험원 식별

위험원 식별은 안전성 분석을 수행하기 위한 첫 과정이다. 위험원 식별을 통해 시스템이 가질 수 있는 위험원의 목록이 도출되고, 이를 이용해 위험원 분석과 위험도 평가가 수행된다. 즉 위험원 식별에서 파악되지 못한 위험원이 있을 경우, 사고로 이어질 가능성이 있기 때문에 위험원 식별은 명확하고 철저하게 수행되어야만 한다.

위험원 식별은 시스템 명세로부터 시스템 기능을 파악해 제어 구조 다이어그램을 작성하는 것으로부터 시작된다. 시스템 명세가 주어지면 유즈케이스 다이어그램으로 시스템의 기능과 사용자를 분석하고, [4]에서 제안된 바와 같이 클래스 다이어그램을 이용하여 제어 구조 다이어그램을 작성한다. 제어 구조 다이어그램은 기본적으로 제어기(Controller), 실행기(Actuator), 센서(Sensor), 제어되는 프로세스(Controlled Process)의 클래스로 구성되고, 각 클래스의 변수와 함수는 유즈케이스 다이어그램에서 분석한 내용을 바탕으로 설정한다. 여기서 제어기와 실행기의 함수는 시스템의 제어 명령을 나타낸다.

제어 구조 다이어그램이 작성되면 시스템이 가질 수 있는 UCA 를 식별한다. UCA 는 [3]에서와 같이 각 시스템의 제어 명령 별로 네 가지의 키워드를 적용하여 식별한다.

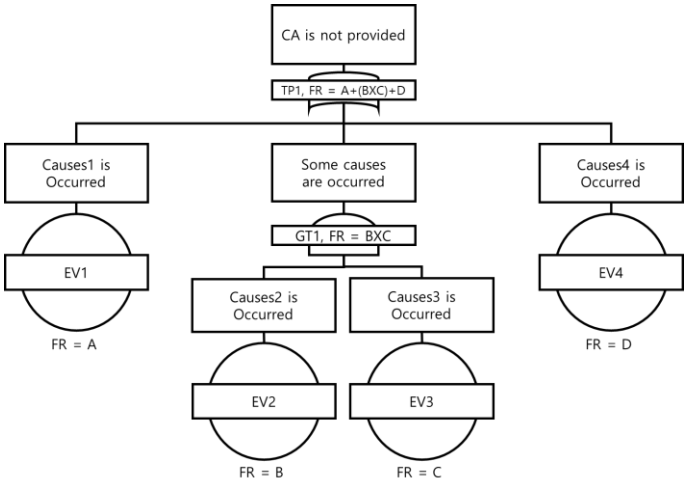
3.2 위험원 분석

위험원 분석은 FTA 와 ETA 를 이용하여 수행된다. 위험원 분석을 통해 각 위험원이 어떠한 원인으로 인하여 발생하였고 인명, 환경, 그리고 재산에 얼마만큼의 영향을 미치는지가 분석된다.

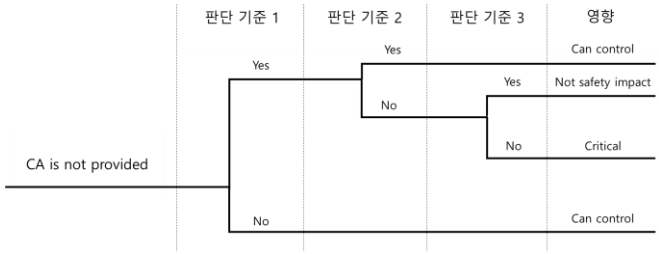
FTA 에서는 UCA 를 발생시키는 원인을 분석한다 [5,6]. 각 UCA 를 정상 사상으로 하여 결함 트리를 작

성해, 이를 발생시키는 원인들을 파악 및 분석하고 고장률을 계산한다. 예를 들어 <그림 2>는 “CA is not provided”에 대한 FTA 수행 결과로, 이 위험원을 발생시키는 원인은 4 개가 있다. 원인들을 분석하면, 위험원은 ‘Cause1’ 또는 ‘Cause4’에 의해 발생되거나 ‘Cause2’와 ‘Cause3’이 동시에 일어날 경우 발생될 수 있다. 또한 위험원의 고장률을 계산하면 A+(BXC)+D 가 된다.

ETA 에서는 UCA 로 인해 발생되는 영향을 분석한다[5,7]. 각 UCA 마다 판단 기준을 적용해 어떠한 피해들이 발생하는지를 파악한다. 예를 들어 <그림 3>은 “CA is not provided” 대한 ETA 수행 결과로, 3 개의 판단 기준을 적용하여 네 가지의 발생되는 영향을 파악하였다. 영향들을 분석하면, 두 가지는 제어가 가능하고 한 가지는 안전과 관련이 없지만 나머지 한 가지는 안전에 ‘Critical’한 영향을 미친다.



<그림 2> FTA를 이용한 발생 원인 분석



<그림 3> ETA를 이용한 발생 영향 분석

3.3 위험도 평가

위험원이 식별되고 분석되면 위험도가 평가된다. 위험도 평가에서는 위험원 분석의 내용을 기반으로 각 위험원이 가지는 발생 빈도와 심각도를 추정하여 위험도를 평가한다.

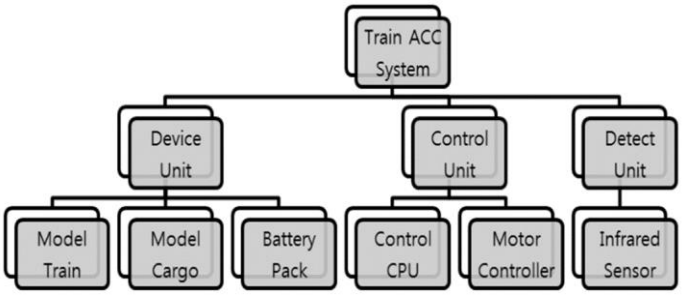
FTA 수행 결과에서는 발생 빈도가 추정되고, ETA 수행 결과에서는 심각도가 추정된다. 그리고 발생 빈도와 심각도의 곱으로 위험도 우선 순위(Risk Priority Number)를 정하고 그에 따라 위험도를 평가한다. 위험도의 평가에는 ‘Intolerable’, ‘Undesirable’, ‘Tolerable’, ‘Negligible’의 네 가지 기준이 사용된다[10].

3.4 소프트웨어 안전 요구사항 도출

RSTPA 의 마지막은 소프트웨어 안전 요구사항의 도출이다. 소프트웨어 안전 요구사항은 [3]에서와 마찬가지로 UCA 와 발생 원인의 조합으로 이루어지는 사고 시나리오를 제어하기 위해 식별된다. 예를 들어 <그림 2>에서 “CA is not provided”와 발생 원인 중 하나인 ‘Cause1’이 조합되면 “When cause1 is occurred, CA is not provided”라는 사고 시나리오가 된다. 그리고 이를 제어하는 “When cause1 is occurred, CA must be provided”가 소프트웨어 안전 요구사항이 된다.

4. 적용 사례

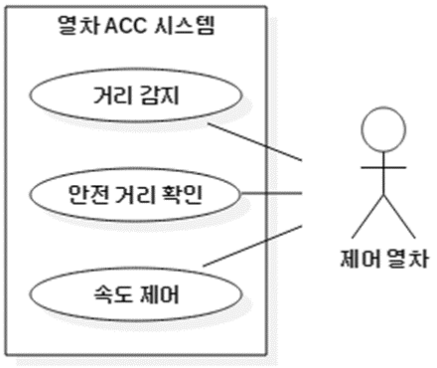
본 연구에서는 제안한 안전성 분석을 사례 연구로 선정한 열차 ACC(Adaptive Cruise Control) 시스템에 적용하였다. 열차 ACC 시스템은 앞 차와의 거리를 감지해 자동으로 열차의 속도를 조절하여 항상 안전 거리를 유지하는 시스템으로, 사고 발생 시 앞 차와의 충돌이 일어나기 때문에 철저한 안전성 분석의 수행이 요구되는 시스템이다. <그림 4>는 열차 ACC 시스템의 구조도를 나타낸다.



<그림 4> 열차 ACC 시스템

4.1 유즈케이스 다이어그램

열차 ACC 시스템은 적외선 센서를 이용하여 앞 차와의 거리를 감지해 자동으로 열차의 속도를 조절함으로써 항상 안전 거리를 유지하는 시스템이다. <그림 5>는 열차 ACC 시스템을 유즈케이스 다이어그램으로 나타낸 것으로 사용자로는 제어 열차가 있고, 앞 열차로부터의 거리가 감지하는 ‘거리 감지’와 거리가 감지되면 안전 거리를 확인하는 ‘안전 거리 확인’, 그리고 제어 열차의 속도를 제어하는 ‘속도 제어’의 세 함수를 가진다.



<그림 5> 열차 ACC 시스템 유즈케이스 다이어그램

4.2 제어 구조 다이어그램

열차 ACC 시스템의 제어 구조도 다이어그램은 열차 ACC 시스템 제어기, 모터 제어기, 제어 열차, 적외선 센서의 4 개 클래스를 가진다. 열차 ACC 시스템 제어기는 제어기 클래스에 해당하는 것으로 2 개 변수와 5 개 함수로 이루어진다. 모터 제어기는 실행기 클래스에 해당하는 것으로 1 개 변수와 3 개 함수로 이루어진다. 또한 제어 열차는 제어되는 프로세스 클래스에 해당하는 것으로 1 개 변수로 이루어지고, 적외선 센서는 센서 클래스에 해당하는 것으로 1 개 변수와 1 개 함수로 이루어진다. 여기서 열차 ACC 시스템 제어기 클래스와 모터 제어기 클래스의 accelerate(), decelerate(), fullyStpo()의 3 개 함수가 열차 ACC 시스템의 제어 명령이다. <그림 6>은 열차 ACC 시스템의 제어 구조 다이어그램이다.

<그림 6> 열차 ACC 시스템 제어 구조 다이어그램

4.3 UCA

열차 ACC 시스템은 accelerate(), decelerate(), fullyStpo()의 세 개 제어 명령을 가진다. 각 제어 명령마다 [3]과 같이 'Wrongly provided', 'Not provided', 'Provided wrong timing or order', 'Stopped too soon or applied too long'의 네 가지 키워드를 적용하여 총 12 개의 UCA 를 식별하였다. <표 1>은 열차 ACC 시스템의 UCA 목록을 나타낸다.

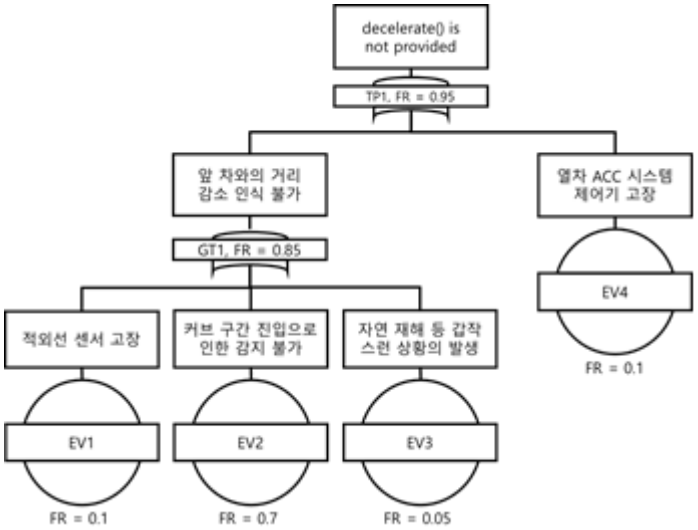
<표 1> 열차 ACC 시스템 UCA 목록

UCA	accelerate()	decelerate()	fullyStop()
Wrongly provided	accelerate() is wrongly provided (H1)	decelerate() is wrongly provided (H5)	fullyStop() is wrongly provided (H9)
Not provided	accelerate() is not provided (H2)	decelerate() is not provided (H6)	fullyStop() is not provided (H10)
Provided wrong timing or order	accelerate() is provided wrong timing or order (H3)	decelerate() is provided wrong timing or order (H7)	fullyStop() is provided wrong timing or order (H11)
Stopped too soon or applied too long	accelerate() is stopped too soon or applied too long (H4)	decelerate() is stopped too soon or applied too long (H8)	fullyStop() is stopped too soon or applied too long (H12)

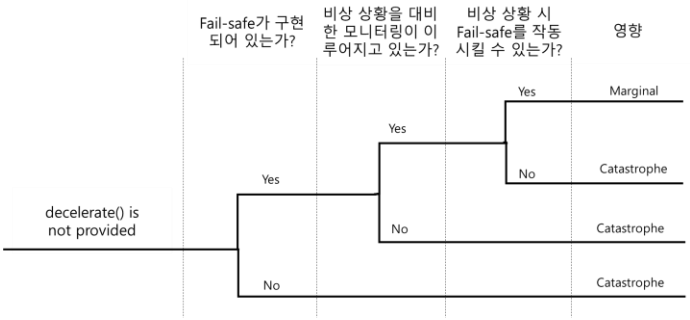
4.4 FTA 와 ETA

열차 ACC 시스템이 가지는 12 개의 UCA 각각 마다 FTA 와 ETA 를 수행해 발생시키는 원인과 발생되는 영향을 분석하였다. <그림 7>은 12 개의 UCA 중 하나인 "decelerate() is not provided"에 대한 FTA 수행 결과로, 'H6'은 적외선 센서 고장, 커브 구간 진입으로 인한 감지 불가, 자연 재해 등 갑작스런 상황의 발생으로 인한 앞 차와의 거리 감소 인식 불가나 열차 ACC 시스템 제어기 고장으로 인하여 발생될 수 있다. <그림 8>은 'H6'에 대한 ETA 수행 결과로, 현재 열차 ACC 시스템은 'Fail-safe' 기능은 구현이 되어 있지만 비상 상황을 대비한 모니터링이 이루어지지 않고 있기 때문에 매우 심각한 영향을 초래할 수 있다는 것을 알 수 있다.

FTA 와 ETA 의 수행 결과는 위험도 평가에서도 사용된다. [10]에서 나온 기준에 따라 발생 빈도와 심각도를 추정하고 위험도를 평가하였다. <표 2>와 <표 3>은 열차 ACC 시스템의 위험도 평가 표이다.



<그림 7> H6의 FTA 결과



<그림 8> H6의 ETA 결과

<표 2> 열차 ACC 시스템 위험도 평가 표

UCA	발생 원인	발생 빈도	심각도	위험도
accelerate() is wrongly provided	적외선 센서 고장	Remote	Catastrophic	Undesirable
	커브 구간 진입으로 인한 감지 불가	Possible	Catastrophic	Intolerable
	자연 재해 등 갑작스런 상황 발생	Improbable	Catastrophic	Tolerable
	열차 ACC 시스템 제어기 고장	Remote	Catastrophic	Undesirable
accelerate() is not provided	적외선 센서 고장	Remote	Insignificant	Negligible
	커브 구간 진입으로 인한 감지 불가	Possible	Insignificant	Negligible
	자연 재해 등 갑작스런 상황 발생	Improbable	Insignificant	Negligible
	열차 ACC 시스템 제어기 고장	Remote	Insignificant	Negligible
accelerate() is provided wrong timing or order	모터 제어기 고장	Remote	Insignificant	Negligible
	잘못 작성된 소프트웨어 제어 알고리즘	Remote	Insignificant	Negligible
	열차 ACC 시스템 제어기 고장	Remote	Insignificant	Negligible
accelerate() is stopped too soon or applied too long	적외선 센서 고장	Remote	Insignificant	Negligible
	잘못 작성된 소프트웨어 제어 알고리즘	Remote	Insignificant	Negligible
	열차 ACC 시스템 제어기 고장	Remote	Insignificant	Negligible
decelerate() is wrongly provided	적외선 센서 고장	Remote	Insignificant	Negligible
	커브 구간 진입으로 인한 감지 불가	Possible	Insignificant	Negligible
	자연 재해 등 갑작스런 상황 발생	Improbable	Insignificant	Negligible
	열차 ACC 시스템 제어기 고장	Remote	Insignificant	Negligible

<표 3> 열차 ACC 시스템 위험도 평가 표 (계속)

UCA	발생 원인	발생 빈도	심각도	위험도
decelerate() is not provided	적외선 센서 고장	Remote	Catastrophic	Undesirable
	커브 구간 진입으로 인한 감지 불가	Possible	Catastrophic	Intolerable
	자연 재해 등 갑작스런 상황 발생	Improbable	Catastrophic	Tolerable
	열차 ACC 시스템 제어기 고장	Remote	Catastrophic	Undesirable
decelerate() is provided wrong timing or order	모터 제어기 고장	Remote	Catastrophic	Undesirable
	잘못 작성된 소프트웨어 제어 알고리즘	Remote	Catastrophic	Undesirable
	열차 ACC 시스템 제어기 고장	Remote	Catastrophic	Undesirable
decelerate() is stopped too soon or applied too long	적외선 센서 고장	Remote	Marginal	Tolerable
	잘못 작성된 소프트웨어 제어 알고리즘	Remote	Marginal	Tolerable
	열차 ACC 시스템 제어기 고장	Remote	Marginal	Tolerable
fullyStop() is wrongly provided	적외선 센서 고장	Remote	Insignificant	Negligible
	커브 구간 진입으로 인한 감지 불가	Possible	Insignificant	Negligible
	자연 재해 등 갑작스런 상황 발생	Improbable	Insignificant	Negligible
	열차 ACC 시스템 제어기 고장	Remote	Insignificant	Negligible
fullyStop() is not provided	적외선 센서 고장	Remote	Catastrophic	Undesirable
	커브 구간 진입으로 인한 감지 불가	Possible	Catastrophic	Intolerable
	자연 재해 등 갑작스런 상황 발생	Improbable	Catastrophic	Tolerable
	열차 ACC 시스템 제어기 고장	Remote	Catastrophic	Undesirable
fullyStop() is provided wrong timing or order	모터 제어기 고장	Remote	Catastrophic	Undesirable
	잘못 작성된 소프트웨어 제어 알고리즘	Remote	Catastrophic	Undesirable
	열차 ACC 시스템 제어기 고장	Remote	Catastrophic	Undesirable
fullyStop() is stopped too soon or applied too long	적외선 센서 고장	Remote	Marginal	Tolerable
	잘못 작성된 소프트웨어 제어 알고리즘	Remote	Marginal	Tolerable
	열차 ACC 시스템 제어기 고장	Remote	Marginal	Tolerable

4.5 소프트웨어 안전 요구사항

열차 ACC 시스템의 위험원을 식별 및 분석하고 위험도를 평가해 UCA 와 발생 원인들을 제어하기 위한 소프트웨어 안전 요구사항을 식별하였다. 예를 들어 소프트웨어 안전 요구사항 중 하나인 “커브 구간 진입으로 인한 감지 불가로 인하여 앞 차와의 거리 감소 인식이 불가 하여도, 제어 명령 decelerate()가 반드시 제공되어야 한다”는 사고 시나리오 “커브 구간 진입으로 인한 감지 불가로 앞 차와의 거리 감소 인식이 불가하여, 제어 명령 decelerate()가 제공되지 않는다”를 제어한다. <표 4>는 위험도가 허용 수준을 벗어난 위험원을 제어하기 위한 열차 ACC 시스템의 소프트웨어 안전 요구사항 목록이다.

<표 4> 열차 ACC 시스템 소프트웨어 안전 요구사항 목록

안전 요구사항	제어 위험도
앞 차와의 거리가 감소 했을 때 적외선 센서 고장으로 인하여 앞 차와의 거리 감소 인식이 불가 하여도, 제어 명령 accelerate()는 반드시 제공되면 안된다.	Undesirable
앞 차와의 거리가 감소 했을 때 커브 구간 진입으로 인한 감지 불가로 인하여 앞 차와의 거리 감소 인식이 불가 하여도, 제어 명령 accelerate()는 반드시 제공되면 안된다.	Intolerable
앞 차와의 거리가 감소 했을 때 열차 ACC 시스템 제어기 고장이 발생하여도, 제어 명령 accelerate()는 반드시 제공되면 안된다.	Undesirable
앞 차와의 거리가 감소 했을 때 적외선 센서 고장으로 인하여 앞 차와의 거리 감소 인식이 불가 하여도, 제어 명령 decelerate()가 반드시 제공되어야 한다	Undesirable
앞 차와의 거리가 감소 했을 때 커브 구간 진입으로 인한 감지 불가로 인하여 앞 차와의 거리 감소 인식이 불가 하여도, 제어 명령 decelerate()가 반드시 제공되어야 한다	Intolerable
앞 차와의 거리가 감소 했을 때 앞 차와의 거리가 감소 했을 때 열차 ACC 시스템 제어기 고장이 발생하여도, 제어 명령 decelerate()가 반드시 제공되어야 한다	Undesirable
앞 차와의 거리가 감소 했을 때 모터 제어기 고장이 발생 하여도, 제어 명령 decelerate()는 반드시 잘못된 시간이나 순서에 제공되면 안된다.	Undesirable
앞 차와의 거리가 감소 했을 때 잘못 작성된 소프트웨어 제어 알고리즘이 있어도, 제어 명령 decelerate()는 반드시 잘못된 시간이나 순서에 제공되면 안된다.	Undesirable
앞 차와의 거리가 감소 했을 때 열차 ACC 시스템 제어기 고장이 발생하여도, 제어 명령 decelerate()는 반드시 잘못된 시간이나 순서에 제공되면 안된다.	Undesirable
앞 차와의 거리가 안전 거리 미만일 때 적외선 센서 고장으로 인하여 앞 차와의 거리가 안전 거리 미만임을 확인 불가 하여도, 제어 명령 fullyStop()가 반드시 제공되어야 한다.	Undesirable
앞 차와의 거리가 안전 거리 미만일 때 커브 구간 진입으로 인한 감지 불가로 인하여 앞 차와의 거리가 안전 거리 미만임을 확인 불가 하여도, 제어 명령 fullyStop()가 반드시 제공되어야 한다.	Intolerable
앞 차와의 거리가 안전 거리 미만일 때 열차 ACC 시스템 제어기 고장으로 인하여 앞 차와의 거리가 안전 거리 미만임을 확인 불가 하여도, 제어 명령 fullyStop()가 반드시 제공되어야 한다.	Undesirable
앞 차와의 거리가 안전 거리 미만일 때 모터 제어기 고장이 발생 하여도, 제어 명령 fullyStop()는 반드시 잘못된 시간이나 순서에 제공되면 안된다.	Undesirable
앞 차와의 거리가 안전 거리 미만일 때 잘못 작성된 소프트웨어 제어 알고리즘이 있어도, 제어 명령 fullyStop()는 반드시 잘못된 시간이나 순서에 제공되면 안된다.	Undesirable
앞 차와의 거리가 안전 거리 미만일 때 열차 ACC 시스템 제어기 고장이 발생하여도, 제어 명령 fullyStop()는 반드시 잘못된 시간이나 순서에 제공되면 안된다.	Undesirable

5. 결과 분석

본 연구에서는 식별한 소프트웨어 안전 요구사항들의 타당성을 확인하기 위해 열차 ACC 시스템에 반영하였다. 이를 위해 5 개의 안전 기능이 열차 ACC 시스템에 구현되었다. 먼저, 일정 간격마다 설치된 RFID 태그 값을 RFID 리더기로 읽어 서버로 보내 각 열차의 위치를 서버에 저장하고, 서버는 열차의 위치를 전송 받을 때마다 앞 차와의 거리 위치 차이 값을 안전한 열차 ACC 시스템으로 보낸다. 위치 차이 값이 보내지면 안전한 열차 ACC 시스템은 적외선으로 감지한 앞 차와의 거리와 위치 차이 값의 두 매개변수를 이용하여 다음 속도를 제어한다. 또한, 서버는 일정 시간마다 열차의 위치가 바뀌지 않을 시 비상 상태로 판단하여 열차의 'Fail-

safe' 기능을 수행하도록 한다. <표 5>는 구현된 안전 기능과 이를 위한 기능 구현 요구사항을 나타낸다.

안전성 분석의 결과로 식별된 소프트웨어 안전 요구사항을 열차 ACC 시스템에 반영한 결과, 커브 구간에서 적외선 센서 감지 각도로 인하여 앞 차와의 거리를 감지하지 못하는 것을 방지할 수 있었다. 또한, 열차들의 위치를 모니터링 하고 수시로 각 열차에게 알리고 있기 때문에 비상 상태 발생 시 대처할 수 있었다.

<표 5> 열차 ACC 시스템에 구현된 안전 기능

안전 기능	구분	기능 구현 요구사항	
		요구사항	구분
열차가 현재 위치 정보 값을 저장하는 기능	소프트웨어 기능	RFID 태그	하드웨어 추가
		RFID 리더기	하드웨어 추가
서버가 열차들의 현재 위치 정보 값을 저장하는 기능	소프트웨어 기능	아두이노 Wifi	하드웨어 추가
		서버	소프트웨어 추가
서버가 각 열차에게 앞 차와의 위치 차이 값을 전달하는 기능	소프트웨어 기능	아두이노 Wifi	하드웨어 추가
		서버	소프트웨어 추가
적외선 센서가 감지한 앞 차와의 거리 값과 서버가 보낸 앞 차와의 위치 차이 값을 이용한 다음 속도 결정 기능	소프트웨어 기능	아두이노 CPU	소프트웨어 수정
		서버	소프트웨어 추가
통신 장애를 대비한 Heartbeat 와 비상 정지 기능	소프트웨어 기능	아두이노 CPU	소프트웨어 수정
		서버	소프트웨어 추가

6. 결론 및 향후 연구

본 연구에서는 안전한 소프트웨어 집약적 시스템을 위한 RSTPA 를 제안하고 열차 ACC 시스템에 적용하였다. 또한 분석 결과로 도출된 소프트웨어 안전 요구사항을 시스템에 반영하였다. 그 결과, 소프트웨어의 오류가 발생하여도 사고를 방지 또는 회피 할 수 있었다.

본 연구가 기여하는 바는 두 가지로 본다. 첫째, 위험도에 따라 제어 조치의 우선 순위를 결정할 수 있다. 이는 시스템 이론적 안전성 분석에서 위험원의 체계적인 관리를 돕고, 더 나아가 소프트웨어의 안전성 분석을 돕는다. 둘째, 잔여 위험원의 정보를 명확히 제공할 수 있다. 위험원의 위험도는 제로로 만드는 것은 불가능하기 때문에, 위험도를 통해 얼마만큼의 위험원이 잔류 되었는지를 알 수 있다.

향후 연구 방향은 본 연구의 경험을 바탕으로 소프트웨어 안전성 분석 결과를 반영해 안전한 시스템과 소프트웨어를 모형 기반으로 설계 및 구현하는 방법을 연구하고자 한다.

참고문헌

[1] 김의섭, 윤상현, 유준범, "안전 필수 시스템을 위한 안전성 분석 기법", 중소기업정보기술융합학회 논문지, 제 2 권 제 1 호, pp. 11-18, 2012

[2] Clifton A. Ericson, II, "Hazard Analysis Techniques for System Safety", Wiley-Interscience, 2005

[3] A. Abdulkhaleq, "A System-Theoretic Safety Engineering Approach for Software-Intensive Systems", 2017

[4] Z. Deming, W. Nan, W. Qianpeng, S. Rui, "A Multi-view Extended Software Control Structure Modeling and Safety Analysis Method", Prognostics and System Health Management, 2015

[5] European Railway Agency, "Functional Safety Analysis of ETCS DMI", 2009

[6] IEC 61025:2006, Fault tree analysis (FTA)

[7] IEC 62502:2010, Analysis techniques for dependability – Event tree analysis (ETA)

[8] 정진우, "위험성평가의 기본", 산업보건, 제 299 권, 제 7 호, pp. 30-35, 2013

[9] 정진우, "위험성평가의 필요성과 효과", 산업보건, 제 300 권, 제 7 호, pp. 35-39, 2013

[10] IEC 62278:2002, Railway Applications, Specification and Demonstration of Reliabliity, Availablility, Maintainability and Safety (RAMS)

유전 프로그래밍을 이용한 버그 자동 정정 기법에 유사 버그 정정 활용

정영준 O1, 민경식 1, 양근석 1, 이정원 2, 이병정 1
서울시립대학교 컴퓨터과학과 1 아주대학교 전자공학과 2
{realjuni, ksmin1710, ypats87, bjlee}@uos.ac.kr1, jongwony@ajou.ac.kr2

Towards Using Similar Bug Fixing Information for Automatic Program Repair with Genetic Programming

Youngjun Jeong O1, Kyeongsic Min1, Geunseok Yang1, Jung-Won Lee2, Byungjeong Lee1
Dept. of Computer Science, University of Seoul1
Dept. of Electrical and Computer Engineering, Ajou University2

요 약

최근 소프트웨어 유지보수의 비용과 노력을 줄이기 위해 소프트웨어 결함 자동 정정 연구가 활발히 진행되고 있다. 일반적으로 정정작업 이전에 미리 정의된 템플릿이나 제약조건을 이용해 결함을 수정하는 방법과 휴리스틱 기법의 일종인 유전 프로그래밍을 이용하는 두 가지 방법으로 정리할 수 있다. 하지만 제시된 조건 이외의 결함에 대해서는 효과를 보기가 어려우며, 무작위한 패치를 생성함으로써 이전의 정정 정보와 같은 유용한 정보를 재사용할 수 없다는 단점이 있다. 이러한 문제점을 해결하기 위해 본 연구는 유지보수 비용을 줄이기 위해 유전 프로그래밍 기법과 함께 과거 정정된 유사 버그의 정보를 활용하는 기법을 제안한다. 또한 이 과정에 대한 사례 연구와 함께 해당 예제 코드에 대한 간단한 결과를 제시하여 본 논문에서 제안하는 기법이 효과적임을 보인다.

1. 서 론

소프트웨어는 규모가 커지고 복잡해짐에 따라 인한 버그 발생이 필연적이다. 이에 따라 개발자가 소화 가능한 역량을 넘어서는 많은 수의 버그가 발생하는데, 일례로 널리 사용되는 오픈 소스 소프트웨어 프로젝트인 Eclipse와 Mozilla는 하루에 약 350여건의 버그 리포트가 보고될 정도이다 [1].

이러한 버그들을 자동으로 정정해주기 위한 연구가 최근 진행되고 있는데, 그 중 유전 프로그래밍을 활용해 자동으로 패치를 생성해가며 성공 여부를 판단하는 기법 [2] 과 사전 조사를 통해 버그의 패턴과 그 정정 동작에 관한 템플릿을 작성해 올바른 패치를 생성하는 기법 [3] 이 대표적이다. 그러나 유전 프로그래밍만 활용할 경우 버그를 포함한 프로그램 코드만으로부터 유전 프로그래밍을 위한 정보를 얻으므로 해당 코드에 정정 코드에 대한 정보가 전혀 없다면 성공적 패치를 생성할 수 없고, 템플릿에 기반한 버그 정정은 정확한 정정 과정에 대해 명시함으로써 식별된 패턴을 가진 결함에 대해서 효율적이나 사전에 많은 노력을 들여 결함과 그 정정 패턴에 대한 템플릿을 작성해야 하는 단점이 있다.

본 논문에서는 유사한 버그에 대한 정정 정보를 활용, 유전 프로그래밍을 기반으로 한 버그 자동 정정 기법을 이용해 소프트웨어 버그 자동 정정의 효율을 높이는 방안을 제시하고자 한다.

본 논문의 연구 동기는 다음과 같다.

- 나날이 늘어가는 버그를 자동으로 수정할 수 있다면 개발자의 업무량을 감소시킴으로써 전체 소프트웨어 유지보수 비용을 줄일 수 있다.
- 유사한 버그가 여러 개 존재하는 것을 식별할 수 있고 그 중 정정된 버그가 있다면 비슷한 디버깅 작업 과정을 통해 해당 버그들이 고쳐질 수 있으므로 전체 작업량이 감소할 것이다.
- 버그를 해결해야 하는 담당자가 해당 버그에 대한 많은 정보가 없을 경우 결함을 완전히 수정하지 못하거나 예기치 않은 동작을 일으키는 정정 작업을 진행할 수 있으나 과거 정정 정보를 이용하면 어느 정도 결과가 보장될 수 있다.

본 논문의 기여는 다음과 같다.

- 본 연구는 자동화된 버그 정정으로 소프트웨어 유지보수 작업의 노력과 시간을 줄이는 데에 기여한다.
- 본 연구는 신뢰성 있는 과거 유사 버그 정정 정보를

활용하여 버그 정정의 품질을 향상시키는 데에 기여한다.
- 본 연구는 유전 프로그래밍과 유사 버그 정정 정보를 활용하여 버그 자동 정정 기법의 향상에 기여한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구를 위한 배경 지식을 설명한다. 3장에서는 본 연구와 관련된 연구를 소개하고 4장에서 본 연구의 모델을 제시한다. 5장은 제시된 모델의 실제 사례 연구를 소개하고 6장에서는 본 연구에 대한 평가와 토의를 수행한 뒤 7장에서 결론을 맺는다.

2. 배경 지식

2.1 유전 프로그래밍 (GP: Genetic Programming)

유전 프로그래밍은 생명체의 유전자가 다음 세대로 전해지는 과정에서 유전자 풀을 다양하게 함으로써 환경 변화에 적응할 수 있는 능력을 확보하고 진화해가는 방법을 모사한 유전 알고리즘 [4]의 원리를 프로그래밍에 적용한 것으로서, 부모로부터 유전자를 물려받을 때에 유전자가 교차 및 변이되는 것을 연산화하고 향후 자연선택 등의 과정을 거치며 환경에 적응하는 유전자만 남아 진화해가는 것을 적합도 계산을 통한 선택 연산으로 표현한 것이다.

소스 코드에 유전 알고리즘을 적용하기 위해 먼저 대상이 되는 프로그램 소스 코드를 입력으로 받아 추상 구문 트리(AST: Abstract Syntax Tree)로 변환, 유전 알고리즘에서 비롯된 연산을 적용하고 이를 다시 소스 코드로 변환한다. 연산이 적용된 후의 제안된 패치 코드는 적절한 평가 기법을 통해 적절한 패치인지 판단한다. 제안된 패치가 버그를 고쳤다고 판단되면 성공으로 간주하고, 설정한 세대 수가 될 때까지 성공하지 못하면 실패한 것으로 간주한다.

2.2 프로그램 분석

자동으로 생성된 패치가 어느 정도 보장된 품질을 갖도록 버그의 형태와 해당 버그를 정정하는 패턴을 조사해 작성한 템플릿에 의해 프로그램을 정정하거나 [3] 문제 없는 정정 과정의 정보를 이용하여 대상 값을 식별하고 조건을 결합하는 과정을 통해 버그를 정정하는 기법 [5] 과 달리 유전 프로그래밍을 이용할 경우 대상 프로그램을 변형하면서 코드 구문의 배열에 무작위성을 갖는 프로그램이 생성된다. 이러한 무작위성에 의한 품질 저하를 예방하기 위해 생성된 프로그램의 적합성을 평가하고 더 나은 품질을 갖는 프로그램 소스 코드를 판별해 다음 생성 과정에서 이용하는 것이 필수적이다. 이를 위해 본 연구에서는 테스트 케이스를 이용한 동적 분석 방식을 이용한다. 이 과정은 유전 프로그래밍 과정과 독립적으로 수행되므로 적용되는 기법을 발전시켜 후보 패치 코드의 품질을 향상시킬 경우 손쉽게 전체 과정의 성공률 향상을 가져올 수 있다. 본 연구는 Le Goues et al. [2] 이 이용한 방법에 기반을 둔 프로그램 분석 방식을 이용한다.

3. 관련 연구

3.1 버그 자동 정정

Le Goues et al. [2] 의 연구는 유전 프로그래밍을 버그 자동 정정에 성공적으로 적용한 최초 연구 사례로 볼 수 있다.

이를 확장한 GenProg [6] 은 버그가 발생한 C언어 프로그램 소스 코드 하나를 초기 입력값으로 받아 추상 구문 트리로 변환한 뒤 변이(Mutation)연산을 적용해가며 초기 모집단을 형성하고, 변이 연산 외의 유전 프로그래밍 연산인 선택(Selection)과 교차(Crossover)를 거쳐 원래의 추상 구문 트리와 다른 트리를 만들어낸다. 이렇게 만들어낸 추상 구문 트리를 프로그램 소스 코드로 바꾸어 테스트 케이스를 통해 버그가 정정되었는지 판단하는 과정을 반복해 패치된 프로그램을 생성해낸다. 해당 연구는 여타 정정 정보를 활용하지 않고, 버그가 있는 프로그램만을 입력으로 받아들여 그 안에 포함된 정보만을 이용해 버그를 정정한다.

Kim et al. [3] 은 사람이 작성한 디버깅 템플릿을 기반으로 버그를 정정한 PAR 기법을 제시한다. 자주 발생하는 버그와 그에 따른 정정 패턴을 템플릿으로 정의한 뒤, 대상이 되는 프로그램의 결함(fault) 위치를 기반으로 템플릿에 부합되는 코드가 있는지 식별한다. 이후 해당 코드를 테스트 케이스를 통해 적합도를 계산한 뒤 tournament selection을 이용해 선택한다. 마지막으로 생성된 패치들이 개발자가 직접 수정한 패치와 비교하여 받아들일 수 있는지까지 조사하여 비교한다.

Long et al. [5] 은 프로그램을 파라미터화 하고 값을 추출한 뒤 조건을 결합/수정하는 방법을 통해 버그 자동 정정의 효과를 높였다. 다만 이러한 방법은 버그를 정정하기 위해 프로그램에 포함되지 않은 새로운 조건이 필요할 경우 버그를 제대로 수정할 수 없다.

Qi et al. [7] 은 SBSE(Search-Based Software Engineering) 기법에 기반하여 GenProg [6] 에서 버그가 정정된 대상 프로그램의 검색 공간(Search Space) 내의 패치 대상을 무작위하게 설정(예: 변이 연산)함으로써 얻을 수 있는 이점을 제시한다. 또한 유전 프로그래밍 연산을 적용한 뒤 테스트 케이스를 기능 레벨, 시스템 모델 레벨, 블럭 레벨, 메서드 레벨로 세분하여 적용함으로써 패치 적합도를 평가하는 더 나은 방법을 제안한다.

F. Long and M. Rinard [8] 는 수정된 코드뿐 아니라

변경된 코드와 주변의 코드가 어떻게 상호작용하는지에 대한 정보를 추출하고 올바른 코드에 대해 기계 학습을 적용해 정정에 필요한 정보를 추출, 학습함으로써 자동으로 패치를 생성하는 기법을 제안했다.

Nguyen et al. [9] 는 프로그램 결함을 고치기 위해 소스 코드 자체만을 활용하는 것이 아니라 코드를 통해 파악할 수 있는 시멘틱 정보까지 활용한 SemFix 기법을 제안했다.

4. 버그 자동 정정 기법

본 연구에서 제시하는 버그 자동 정정 기법은 패치 생성 과정에서 유전 프로그래밍 기법을 이용한다. 그림 1은 본 연구에서 제시하는 모델의 전체 과정을 나타낸다.

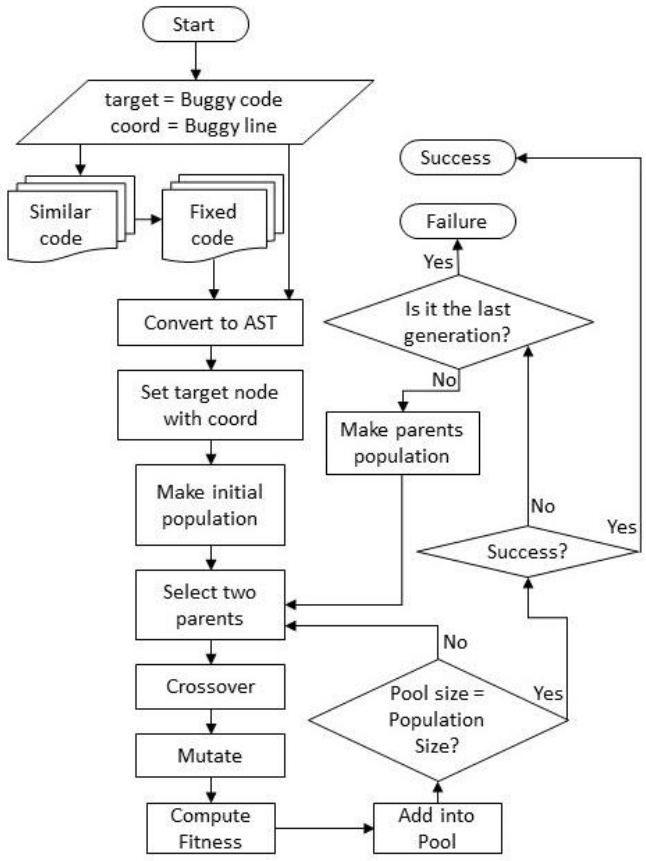


그림 1. 버그 자동 정정 기법 개요

4.1 결함 위치 추적

기본적으로 소스 코드를 대상으로 하여 버그를 정정하기 위해서는 소스 코드의 어느 위치에서 버그가 발생했는지 알아야 한다. 이는 버그 자동 정정 기법이 적용될 직접적인 대상 위치를 특정하는 것으로 규모가 큰 소프트웨어를 정정 대상으로 할수록 중요한 역할을 한다. 완전한 소프트웨어 버그 자동 정정을 위해 해당 단계가 정확한 결과를 내놓는 것은 필수적이다. 이를 위해 IR 모델을 이용하거나 [10], 잠재 디리클레 할당(LDA: Latent Dirichlet Allocation)을 이용 [11], 소스 코드 내의 정보뿐 아니라 코드 변경 기록 및 버그 리포트를 활용한 기법 [12] 등이 제안되었다. 다만 결함 위치 추적 기술은 본 연구의 주요 논점이 아니므로 각 소스 코드에서 결함 위치 정보를 이미 알고 있다고 가정한다.

4.2 추상 구문 트리 변환

유전 프로그래밍을 적용하기 위해서는 소스 코드를 트리 형태로 변환해야 한다. 이를 위해 각 프로그램의 프로그래밍 언어에 맞는 분석 과정이 필요하다. 그림 2의 라인 2는 주어진 프로그램 소스 코드를 추상 구문 트리로 변환해 반환한다. 이 과정을 거쳐 소스 코드가 트리와 각 노드의 정보로 표현되므로 노드 연결 관계 또는 노드 내의 정보를 조작함으로써 실제 소스 코드를 바꿀 수 있다.

4.3 유전 프로그래밍 적용

p: buggy program
p_{sim}: patched program of similar bug
T: abstract syntax tree
l: buggy Line
N: set of nodes
n: node of AST
$InitPop$: initial population
$fitness$: measured fitness value
$threshold$: set threshold to judge sucess
$parentPoolSize$: the number of parent population
$genNum$: maximum generation number

```
1   l ← FaultLocal(p)
2   T(p) ← ConvertToAST(p)
3   N ← TargetNode(T(p), l)
4   InitPop ← {T(p)}
5   fitness ← 0
6   genCount ← 0
7   for Size(InitPop) < parentPoolSize
8       n ← GetRandom(N)
9       Crossover(T(p), T(p_sim))
10      InitPop ← InitPop ⋃ Mutate(T(p), n)
11  Pop ← InitPop
12  for genCount < genNum
13      for Size(Pop) < popSize and fitness ≥ threshold
14          T1, T2 ← SelectParents(Pop)
15          T1 ← Crossover(T1, T2, N1)
16          T1 ← Mutate(T1)
17          Pop ← Pop ⋃ T1
18          fitness ← Max(ComputeFitness(T1), fitness)
19      genCount ← genCount+1
20      Pop ← Select(Sort(Pop), parentPoolSize)
```

그림 2. 유전 프로그래밍 과정 의사 코드

유전 프로그래밍의 수행은 크게 파라미터 초기화, 유전 프로그래밍 연산 적용, 적합도 평가 및 다음 세대 생성을

위한 개체 선택 및 보존으로 나눌 수 있다. 그림 2는 유전 프로그래밍 연산을 포함한 전체 알고리즘의 의사 코드이다.

4.3.1 초기화

그림 2의 라인 3-6은 초기 값을 설정한다. 라인 3은 식별한 버그의 위치 정보를 이용해 수정이 필요한 노드 집합을 설정한다. 결함 위치 추적 기법에 따라 대상 노드는 여러 개가 될 수 있다. 라인 4는 초기 모집단을 형성하기 위해 입력받은 프로그램 소스 코드의 추상 구문 트리를 초기 모집단에 추가한다. 라인 5는 각 패치에서 얻은 최대 적합도, 라인 6은 반복되는 세대 수를 나타낸다. 라인 7-10은 입력받은 프로그램의 추상 구문 트리를 이용해 초기 모집단을 형성하는 과정이다. 문제가 발생한 것으로 의심되는 노드 중 하나를 대상으로 유사 버그의 정정된 프로그램의 트리와 교차 연산을 거쳐 정정된 코드 정보를 가져온 뒤 변이 연산을 수행해 그 결과를 모집단에 추가해가며 초기 모집단의 수를 채울 때까지 반복한다. 이를 통해 초기 모집단 형성시 유사 버그의 정정된 코드 정보가 포함될 수 있도록 한다.

4.3.2 유전 프로그래밍 연산

라인 12-17은 각 세대를 생성한다. 라인 14에서 유전 프로그래밍 연산을 적용할 두 대상 트리를 정하고 라인 15-16에서 교차 연산과 변이 연산을 수행한다. 교차 연산시, 라인 14에서 얻은 두 트리 중 하나(T_1)는 대상 노드를 버그가 있는 노드로 한정하고, 다른 트리(T_2)의 연산 대상은 모든 노드를 대상으로 한다. 교차 연산을 통해 얻은 두 트리 중 버그가 있는 노드를 대상으로 연산을 수행한 트리만 현재 세대에 추가한다 (라인 17). 이를 통해 버그가 있는 노드가 버그가 없는 노드로 대체되는 것을 기대할 수 있다. 유전 프로그래밍을 소스 코드에 적용하기 위해 본 논문의 구현은 다음과 같은 세부 연산 과정을 따른다.

(1) 선택(Selection): 부모 집단으로부터 연산의 대상이 될 두 대상(추상 구문 트리)을 선택한다. 이 때, 각 개체(트리)는 적합도 값에 따른 룰렛 휠 방식을 적용한다. 이를 위해 먼저 이전 세대의 트리들을 적합도를 기준으로 내림차순으로 정렬한 뒤 설정된 부모 집단 수에 맞춰 집단 수를 조정한 뒤 남은 트리들에 한해 적합도에 따라 확률적으로 선택되도록 한다.

(2) 교차(Crossover): 두 트리에서 대상이 되는 노드로부터 서브 트리를 뒤바꾼다. 이 때, 두 노드가 서로 교환 가능한 노드 타입인지를 부모 노드로부터 추적한다. 교차 연산시 부모가 되는 두 트리 중 하나는 버그가 있는 것으로 판단되는 노드만을 연산 대상으로 하고, 다른 트리는 트리 내에 존재하는 모든 노드에 대해 전자와 교환 가능한 노드 타입인지만 확인하여 교환한다. 교차 연산은 높은 확률로 적용되며, 특히 초기 모집단 형성시에는 하나의 트리를 제외한 모든 트리에 적용한다.

(3) 변이(Mutation): 대상 노드를 삭제하거나, 대상 노드에 새로운 자식 노드를 추가하거나, 대상 트리에 있는 두 노드의 정보를 부모-자식 노드 정보를 제외하고 노드 내부의 정보만을 교환한다. 이외에도 조건문에 사용되는 비교 연산자가 대상 노드로 선택될 경우, 해당 조건문이 가졌을지도 모르는 결함을 수정하기 위해 비교 연산자를 수정한다. 단, 변이 연산은 의도치 않은 결과값을 반환할 수 있으므로 경우가 많으므로 그 확률을 적절히 조절해야 한다.

4.3.3 적합도 평가

유전 프로그래밍은 무작위한 결과를 반환하므로 얻어낸 결과가 목적에 부합하는 코드인지 판별하는 것이 중요하다. GenProg [6] 은 테스트 케이스 중심 방식을 사용해 적합도 값을 측정했고, PAR [3] 은 그 외에도 관련자들을 대상으로 한 설문을 통해 해당 연구가 적합한 결과를 내놓았는지를 평가했다.

라인 18은 새로 얻은 트리의 적합도를 계산해 가장 큰 적합도로 갱신한다. 새로 생성된 트리는 하나의 제안된 패치 프로그램이라고 할 수 있다. 실제 구현에서는 트리를 통해 바로 적합도를 계산할 수 없으므로 이 과정에서 구문 분석기가 트리를 소스 코드로 변환하는 작업이 추가된다. 얻어낸 소스 코드는 컴파일한 후 데이터셋에 포함된 테스트 케이스를 입력하여 출력값을 비교해 평가된다. 본 논문에서는 각 테스트 케이스를 통과하는 수가 높을수록 적합도가 높은 것으로 간주하며 주어진 모든 테스트 케이스(데이터셋에 포함된 블랙 박스 테스트 케이스 및 화이트 박스 테스트 케이스)를 통과한 경우 적합한 패치를 작성한 것으로 간주한다. 그림 3은 적합도를 평가하기 위한 식을 나타낸다. P는 평가 대상이 되는 각 프로그램 소스 코드, $|Passed\ WB|$ 와 $|Passed\ BB|$는 각각 테스팅에 성공한 테스트 케이스 수, $|WB|$와 $|BB|$는 각각의 테스팅을 위해 주어진 전체 테스트 케이스 수를 나타낸다. 결과값이 클수록 적합도가 더 높은 패치이며, 1.0 이면 경우 패치 생성에 성공했음을 나타낸다.

$$\text{Fitness}(P) = \frac{|Passed\ WB| + |Passed\ BB|}{|WB| + |BB|}$$

그림 3. 적합도 평가 수식

5. 사례 연구

본 논문에서 제안하는 모델과 그 진행 과정을 자세히 설명하기 위해 Le Goues et al. [13] 의 벤치마크된 데이터셋 중 간단한 프로그램과 그에 따른 테스트 케이스 및 각 프로그램의 목적에 맞는 모범 답안이 포함된 IntroClass의 코드를 사용한다. 해당 데이터셋은 학생들이 간단한 목적을 가진 프로그램을 C언어를 이용해 작성한 것으로, GenProg [6], AE [15], TrpAutoRepair [16] 기법을 이용해 정정에 성공한 벤치마크도 포함한다. 그림 4는 IntroClass 에 포함돼 있는 프로그램 소스 코드 중 하나이다. 해당 프로그램은 일련의 문자열을 입력으로 받아 그에 해당하는 체크섬 값을 출력하는 프로그램이다. 이용하는 데이터셋에는 소스 코드와 테스트 케이스가 모두 포함된다. 해당 코드에서 결함의 원인이 되는 부분은 7번 라인과 14번 라인이다. 다만 7번 라인은 초기화되지 않음으로 인해 런타임 오류가 발생할 수 있는 라인으로, 실제 테스크 케이스의 출력에 영향을 주는 것은 14번 라인으로 잘못된 수를 더함으로 인해 테스트를 통과하지 못한다.

```
1  /**/
2
3  #include<stdio.h>
4
5  int main (){
6
7      char letter;
8      int sum = 0;
9      printf("Enter an abitrarily long string
10     while(letter != '\n'){
11         scanf("%c",&letter);
12         sum = sum + (int)letter;
13     }
14     sum =sum%64+22;
15     printf("Check sum is %c\n",(char)sum);
16 return 0;
17 }
```

그림 4. 실제 소스 코드와 버그 위치

5.1 유사 코드 판별

먼저 유사 코드를 식별하기 위해 코드 클론 디텍터를 활용한다. 이 단계에서는 T. Kamiya et al. 이 제안한 CCFinder [14] 를 이용한다. 해당 도구를 이용하면 소스 코드를 토큰화해 분석, 그림 5와 같이 도식화된 결과를 얻을 수 있다. 해당 그림의 행과 열은 비교 대상이 되는 각각의 소스 코드를 나타낸다. 코드가 유사할수록 각 행렬 요소에 해당하는 선이 길게 나타나 쉽게 유사한 코드를 찾아낼 수 있다. 정정이 완료된 유사 버그는 초기 모집단 형성에 이용함으로써 제안되는 패치들이 정정된 코드의 정보를 포함할 수 있도록 한다. 많은 양의 버그 정정 정보를 포함할수록 더 다양한 정정 정보를 가질 수 있지만 그만큼 초기에 유사한 버그를 찾아내기 위한 오버헤드가 증가할 수 있다. 본 예제에서는 IntroClass [13] 데이터셋에 포함된 소스 코드 중 그림 4의 코드를 대상으로 하여 가장 유사한 코드 하나만을 식별했다. 이 과정에서 찾아낸 가장 유사한 코드는 그림 6과 같으며 해당 코드가 정정된 코드는 그림 7과 같다. 그림 4와 그림 6을 비교해보면 while문을 사용한 점, while문 내에 scanf() 함수와 수식이 들어간 점, 수식을 구하는 과정에서 모듈로 연산과 가산 연산을 사용한 점 등이 유사함을 알 수 있다.

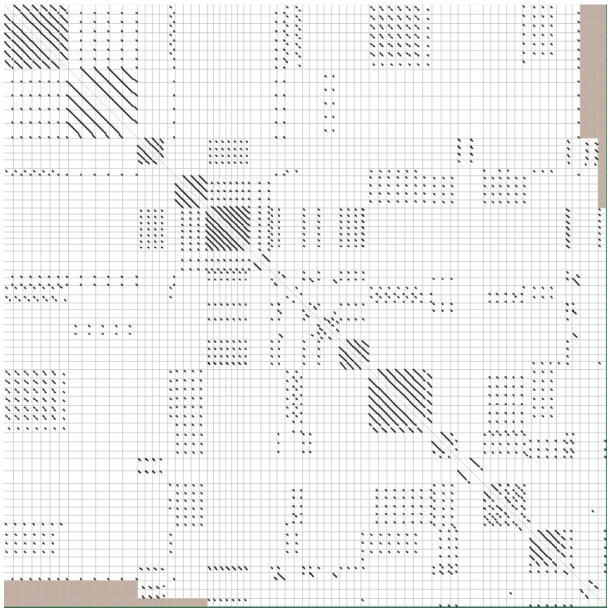


그림 5. 대상 소스 코드에 CCFinder를 적용한 결과

```
1
2  #include <stdio.h>
3  #include <math.h>
4
5  int
6  main (void)
7
8  {
9    char character;
10   char remainder;
11   int sum;
12   printf("Enter an abitrarily long string,
13   sum = 0;
14 while(character != '\n')
15 {
16     scanf("%c", &character);
17     sum = sum + (int)character;
18
19 }
20  remainder = (char)((sum % 64) + 22);
21  printf("Check sum is %c\n", remainder);
22  return (0);
23 }
```

그림 6. 데이터셋 내에서 원본 버그 코드와 가장 유사한 버그 코드

5.2 추상 구문 트리 변환

먼저 각 소스 코드를 추상 구문 트리로 변환하는 작업을 거친다. 그림 8 및 그림 9는 원본 소스 코드와 유사 버그의 정정 코드를 추상 구문 트리로 변환한 결과 중

버그 발생과 관련된 부분을 나타낸다.

```
1
2  #include <stdio.h>
3  #include <math.h>
4
5  int
6  main (void)
7
8  {
9    char character;
10   char remainder;
11   int sum;
12   printf("Enter an abitrarily long string, ending with carr
13   sum = 0;
14   do
15 {
16     scanf("%c", &character);
17     sum = sum + (int)character;
18
19 }
20   while(character != '\n');
21   remainder = (char)(((sum - (int)'\n') % 64) + (int)' ');
22  printf("Check sum is %c\n", remainder);
23  return (0);
24 }
```

그림 7. 유사 버그 코드의 정정 코드

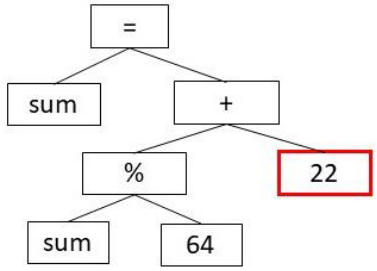


그림 8. 원본 소스코드의 버그가 포함된 부분를 추상 구문 트리로 변환한 결과

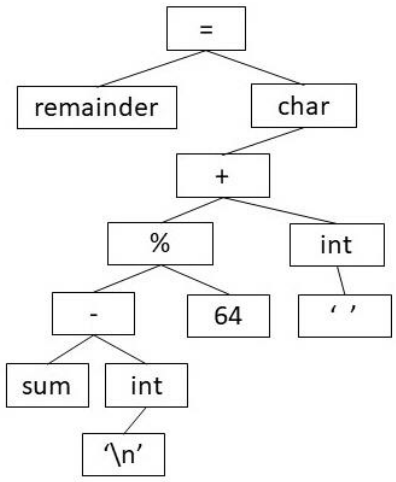


그림 9. 유사 버그 코드의 정정된 부분에 대한 AST

그림 8의 추상 구문 트리는 그림 4 코드의 14라인 코드를 나타낸다. 트리에서 빨간색으로 표시된 노드가 버그를 발생시키는 노드이다. 소스 코드를 추상 구문 트리로 변환할 때에 각 노드가 원래의 소스 코드에서 몇 라인에 위치하는지를 포함하여 추출한 트리에서 버그가 있는 라인 정보를 이용, 유전 프로그래밍 연산이 적용될 대상 노드를 식별한다. 그림 9는 유사 버그 코드에서 버그가 정정된 코드 부분을 트리로 변환한 것이다. 이는 그림 7의 21라인에 해당한다.

5.3 유전 프로그래밍 적용

5.2절에서 얻은 트리에 유전 프로그래밍을 적용하면 그림 10과 같은 그래프를 얻을 수 있다. 각 연산은 수행 이전에 정해진 확률에 따라 적용되며, 여러 번의 반복을 거친다. 해당 트리는 원본 트리(그림 8)에서 빨간색으로 표시된 노드가 변경되었음을 알 수 있다.

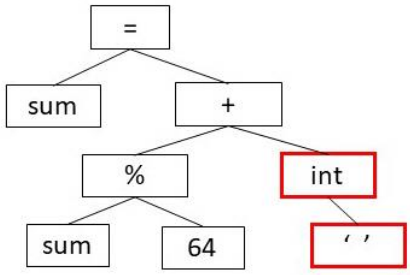


그림 10. 유전 프로그래밍이 적용된 후 얻은 AST

원본 버그 에는 공백 문자에 해당하는 정수형 상수값이 없다. 따라서 유사 버그의 정정 코드를 이용하지 않는다면 해당 버그는 정정될 수 없다. 단, 여기서 이용한 예제 코드들의 교차 연산에서는 이러한 과정을 확인할 수 없지만 유사 코드로부터 트리 노드의 교환 또는 삽입이 일어날 때에는 유사 코드의 변수명들이 같은 타입의 원본 코드 내 변수명으로 바뀌어야 한다. 그림 10의 트리를 소스 코드로 변환하면 그림 11과 같이 패치가 적용된 소스 코드를 얻을 수 있다.

```
1  #include<stdio.h>
2
3  int main (){
4
5      char letter;
6      int sum = 0;
7      printf("Enter an abitrarily long string, e
8      while(letter != '\n')
9      {
10         scanf("%c", &letter);
11         sum = sum + (int)letter;
12     }
13     sum = sum % 64 + (int)' ';
14     printf("Check sum is %c\n", (char)sum);
15 return 0;
16 }
17
```

그림 11. 패치가 적용된 소스 코드

5.4 적합도 평가

유전 프로그래밍을 적용해 프로그램 코드를 생성할 경우, 각 패치는 무작위한 조합 및 추가/삭제/수정 단계를

거치기 때문에 생성된 각 결과를 평가하는 것이 중요하다. 우리가 제시하는 모델은 테스트 케이스 중심 방식을 이용한다. 추출한 소스 코드를 컴파일한 후, 실제로 실행시킨 뒤 주어진 테스트 케이스를 입력값으로 넣어가며 기대하는 결과가 나오는지 확인한다. 총 테스트 케이스 중 몇 개의 테스트 케이스를 통과하는지 확인해 그림 3의 식을 이용, 적합도를 계산한 후 결과값이 1.0이면 패치가 성공한 것으로 간주한다. 한 세대 내의 패치 적합도 최대값이 1.0이 아닌 경우(성공적인 패치를 생성해내지 못했을 경우) 설정해둔 부모 집단 수에 맞춰 낮은 적합도를 가진 패치를 제거한 뒤, 남은 코드에 재차 유전 프로그래밍 연산을 적용해 새로운 패치 집단을 생성한 뒤 적합도 평가 과정을 거친다. 이러한 과정을 반복할수록 전체 집단의 적합도가 높아지는 효과를 기대할 수 있다. 설정한 세대 수에 도달하거나 패치에 걸리는 시간이 일정 시간 이상이면 패치에 실패한 것으로 간주하고 해당 버그에 대한 패치는 더 이상 진행하지 않는다.

6. 평가 및 토의

본 연구에서 제안한 기법과 관련 연구와의 가장 큰 특징은 큰 비용의 사전조사 없이도 자동적으로 관련된 버그를 식별해 그 정정 정보를 이용, 무작위성을 갖는 유전 프로그래밍과 결합해 새로운 패치를 생성해내는 것이다. 이러한 기법을 이용해 이전의 비슷한 버그의 디버깅 코드를 활용하면서 동시에 새로운 패치를 생성해내는 것이 가능하다. 기존 연구와의 비교는 아래와 같다.

6.1 기존 연구와의 비교

본 연구에서는 버그가 포함된 소스 코드와 유사한 코드를 판별하여 해당 코드가 수정된 정보를 유전 프로그래밍 과정에 포함하였다. 이를 통해 얻을 수 있는 이점과 타 연구와의 차이점은 크게 다음과 같다.

(1) 패치 생성에서 유전 프로그래밍 방식만을 이용해 패치를 생성해내는 방식 [2], [6] 은 버그가 포함된 프로그램만을 대상으로 유전 프로그래밍을 적용하므로 대상 프로그램이 올바르게 정정된 코드 정보를 포함하지 않을 경우 버그 정정이 불가하다. 하지만 본 연구는 정정 대상이 되는 코드와 유사한 버그의 정정 코드를 이용함으로써 대상 코드에 디버깅을 위한 정보가 전혀 포함되지 않았을 경우에도 올바른 패치를 생성하는 것이 가능하다.

(2) 버그 패턴과 그에 따른 디버깅 작업 과정을 일련의 템플릿화하거나 프로그램 구성 요소를 재조합하는 방식을 사전에 정의하는 기법 [3], [5] 과 달리 복잡한 사전 정의 작업이 필요하지 않고, 주어진 소스 코드와 유사한 버그를 판별해냄으로써 여러 유형의 버그에 대응할 수 있다.

본 연구가 갖는 한계점은 다음과 같다.

6.2 한계점

(1) 결함 위치 식별: 본 연구는 결함 위치 추적기술을 이용하지 않고 정정 대상 코드의 어느 위치에 버그가 있는지 수동적으로 판단했다. 따라서 각 패치 생성시 결함 가능성이 있는 위치를 일일이 판별하기 힘드므로 이후의 패치 생성시 초기 결함이 있는 것으로 생각되는 코드 라인 및 추가/수정되는 라인만을 대상으로 패치를 생성했다. 이러한 방법은 패치가 진행됨에 따라 프로그램이 많이 변형될수록 부적절한 패치를 생성할 수 있다. 따라서 완전한 자동 디버깅 과정을 위해서는 자동화된 결함 위치 추적 기술을 적용할 필요가 있다. 결함 위치 추적 기술을 매 세대 생성마다 적용할 경우 매 후보 패치 생성시 문제가 있는 코드 라인만을 변경 대상으로 정할 수 있으므로 패치 생성의 정확도를 높일 수 있다.

(2) 유사 코드 판별의 신뢰성: 디버깅 대상이 되는 코드 외에 패치 생성 과정에서 사용하는 유사 코드가 정정 대상 버그와 유사하지 않거나 유사한 버그를 찾아냈더라도 해당 정정 정보가 적절한 수정과정을 거치지 않은 코드라면 본 기법의 효율이 오히려 낮아질 수 있다. 이 문제는 유사한 코드를 판별하기 위한 기법을 향상시키고 더 신뢰성 있는 정정 정보를 가려내는 과정을 추가 연구함으로써 효율을 높일 필요가 있다.

(3) 데이터셋의 범위: 본 논문의 사례 연구에 사용한 데이터셋은 수업 시간에 학생들이 작성한 코드이다. 따라서 각 프로그램 규모가 크지 않고, 실제 많이 사용되는 구조 또는 키워드를 포함하지 않을 수 있다. 본 논문에서 제안하는 기법은 결함이 있는 원본 코드가 비교 대상 코드가 비슷하고 비교 대상 코드의 정정된 코드가 확보되어야 적용이 가능하다. 따라서 비슷한 버그 코드를 찾아내고 그 정정 정보를 이용할 수 있는 데이터셋이 확보가 되어야만 정량적 비교가 가능하다. 실제 널리 사용되는 오픈 소스 소프트웨어 프로젝트를 대상으로 할 경우 공개된 버그 및 정정 데이터가 많은 만큼 좋은 성능을 보일 것으로 기대한다.

(4) 변수명 일치화: 본 연구는 정정 대상 코드뿐 아니라 유사 버그의 정정 코드를 함께 이용하므로 유전

프로그래밍 연산시 정정 대상 프로그램에 포함되지 않은 변수가 삽입될 수 있다. 이러한 문제를 해결하기 위해 사례 연구에서 정정 대상 프로그램의 변수 목록을 작성하여 유사 코드 삽입시 해당 목록에서 무작위하게 변수명을 선택하여 삽입하는 방식을 제안했다. 실제 규모가 큰 소프트웨어 프로젝트를 대상으로 할 경우 이러한 방법의 효율성이 낮을 것으로 예상되므로 변수명을 변경하는 방법이 체계적으로 정립되어야 한다.

7. 결 론

본 논문에서는 짧은 분량의 소스 코드에 포함된 버그를 수정하기 위해 유전 프로그래밍을 기반으로 하여 유사 버그를 판별하고 그 정정된 코드를 활용하는 기법을 제안했다. 이전의 연구는 완전히 무작위한 패치를 생성함으로 인해 높은 효율을 기대하기 어려웠거나 [2], [5], [6], [7] 결함의 패턴과 정정 방식을 조사하기 위해 많은 노력이 드는 사전 조사가 필요했다 [3], [5]. 우리는 이러한 문제점을 개선하기 위해 버그가 식별된 위치를 기준으로 소스 코드만을 이용해 이전에 발생한 유사한 버그를 찾아낸 뒤, 해당 버그의 정정 코드를 유전 프로그래밍의 초기 모집단 형성에 이용함으로써 버그 정정 효율이 높아질 것을 기대했다. 향후 발전된 유사 코드 식별 기법을 적용하고 더 많은 버그 정정 데이터를 포함하는 데이터셋을 이용해 그 정보를 이용할 수 있으면 본 논문에서 제안하는 기법의 효율이 증대될 것으로 기대된다.

8. 참고 문헌

[1] G. Yang, S. Baek, J. Lee and B. Lee, “Analyzing Emotion Words to Predict Severity of Software Bugs: A Case Study of Open Source Projects,” ACM Symposium on Applied Computing, pp. 1280-1287, 2017.

[2] C. Le Goues, M. Dewey-Vogt, S. Forrest and W. Weimer, “A systematic Study of Automated Program Repair: Fixing 55 out of 105 Bugs for $8 Each,” IEEE International Conference on Software Engineering, pp. 3-13, 2012.

[3] D. Kim, J. Nam, J. Song and S. Kim, “Automatic Patch Generation Learned from Human-Written Patches,” IEEE International Conference on Software Engineering, pp. 802-811, 2013.

[4] S. Forrest, “Genetic Algorithms: Principles of Natural Selection Applied to Computation,” Science, vol. 261, pp. 872-878, 1993

[5] F. Long and M. Rinard, “Staged Program Repair with Condition Synthesis,” European Software Engineering Conference and the ACM Sigsoft Symposium on the Foundations of Softare Engineering, pp. 166-178, 2015.

[6] C. Le Goues, T. Nuguyen, S Forrest and W. Weimer, “GenProg: A Genetic Method for Automatic Software Repair,” IEEE Transaction on Software Engineering, pp. 54-72, 2012.

[7] Y. Qi, X. Mao, Y. Lei, Z. Dai and C. Wang, “The Strength of Random Search on Automated Program Repair,” IEEE International Conference on Software Engineering, pp. 254-265, 2014.

[8] F. Long and M. Rinard, “Automatic Patch Generation by Learning Correct Code,” ACM symposium on Principles of Programming Languages, pp.298-312, 2014.

[9] H. D. T. Nguyen, D. Qi, A. Roychoudhury and S. Chandra, “SemFix: Program Repair via Semantic Analysis,” IEEE International Conference on Software Engineering, pp.772-781, 2013.

[10] R. K. Saha, M. Lease, S. Khurshid and D. E. Perry, “Improving Bug Localization using Structured Information Retrieval,” IEEE Automated Software Engineering, pp. 345-355, 2013.

[11] S. K. Lukins, N. A. Kraft, L. H. Etzkorn, “Bug localization using latent Dirichlet allocation,” Information and Software Technology, Vol 52, No. 9, pp. 972-990, 2010.

[12] K. C. Youm, J. Ahn and E. Lee, “Improved bug localization based on code change histories and bug reports,” Information and Software Technology, Vol. 82, No. 1, pp. 177-192, 2017.

[13] C. Le Goues, N. Holtschulte, E. K. Smith, Y. Brun, P. Devanbu, S. Forrest, and W. Wemier, “The ManyBugs and IntroClass Benchmarks for Automated Repair of C Programs,” IEEE Transactions on Software Engineering, Vol. 41, No. 12, pp. 1236-1256, 2015.

[14] T. Kamiya, S. Kusumoto and K. Inoue, “CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code,” IEEE Transaction on Software Engineering, pp. 654-670, 2002.

[15] W. Weimer, Z. P. Fry, and S. Forrest, “Leveraging program equivalence for adaptive program repair: Models and first results,” in Proc. IEEE/ACM 28th International Conference Automated Software Engineering, pp. 356-366, 2013.

[16] Y. Qi, X. Mao, and Y. Lei, “Efficient Automated Program Repair through Fault-recorded Testing Prioritization,” in Proc. International Conference on Software Maintenance, pp. 180-189, 2013.

오픈 소스를 활용한 웹 서비스 구간 별 성능 모니터링 시스템 설계 및 구현

전향희 ○ 나성엽
㈜ 엔키아
gaiajeon@gmail.com, darkerkorean@gmail.com

Design and Implementation of Web Service End-To-End Performance Monitoring System Using Open Source Products

HyangHee Jeon○ Sungyub NA
NKIA Corporation

요 약

웹 서비스의 성능은 온라인 비즈니스 성과와 밀접한 관계가 있다. 웹 사이트 접속 반응 속도가 느리다면 사용자는 해당 사이트를 더 이상 방문하지 않는다. 이러한 문제를 개선하기 위해서는 지속적인 모니터링을 통해 웹 서비스의 성능을 주기적으로 측정하고 느려지는 원인이 되는 구간을 분석하여야 한다. 지연 원인 분석을 위해서는 브라우저에서의 웹 반응 속도뿐만 아니라 네트워크, 웹 어플리케이션 서버, 데이터베이스 트랜잭션 처리 속도를 측정하여 느려지는 구간을 분석, 개선하여 웹 서비스를 원활하게 제공하여야 한다. 본 논문에서는 지연 구간 모니터링을 위해 오픈 소스를 활용하여 구간별 소요 시간을 수집하고 지속적으로 모니터링 할 수 있는 방법을 제시한다.

1. 서 론

아마존, 구글, 카카오와 같은 온라인 서비스 기업뿐만 아니라 기존의 제조, 물류, 의료 등 기존 기업들에게도 온라인 서비스의 제공은 비즈니스의 핵심 요소가 되었다. 많은 사용자들이 기업이 제공하는 온라인 서비스를 통해 기업의 제품을 구입하거나 서비스를 제공 받는다. 이러한 사용자들은 다양한 모바일 장치와 데스크 탑 클라이언트를 이용해 위치에 관계없이 온라인 컨텐츠에 빠르고 안정적으로 접근하길 원한다. 이에 따라 웹 사이트의 성능은 온라인 비즈니스 기업의 핵심 요소가 되었다.

인터넷과 모뎀에 등장하기 시작한 1990년대에는 20~30초 사이의 응답 시간이 주를 이루었으며 1996년 Nielsen,Jacob은 최대 응답시간이 10초가 되기 전에 사용자는 흥미를 잃는다고 하였다[1]. 광대역망과 무선망 기술이 지속적으로 발달됨에 따라 현대의 온라인 서비스 사용자들은 초고속 인터넷망 환경에서 2~3초 정도의 빠른 응답시간을 요구하고 있다. 이에 따라 온라인 서비스 기업들은 사용자들의 만족도 향상을 위해 웹 성능 모니터링에 많은 자원을 투자하고 있다.

웹 성능 측정을 위해 기업에서는 성능 측정 툴을 구매하거나 오픈 소스를 활용하여 모니터링 환경을 구현하기도 한다. 일부 상용 제품의 경우 웹 서버, 데이터베이스 서버 등 다양한 리소스에 대한 응답시간 정보를 연계하여 제공해주기도 하나 오픈 소스의 경우 웹 브라우저 혹은 웹 어플리케이션 서버 등 특정 구간을 모니터링 하는 제품이 주를 이루고 있다. 또한 최근에는 각종 모바일 장비의 발전에 따라 기존 데스크탑 클라이언트보다 모바일 클라이언트를 통한 웹페이지 요청 비율이 급격히 늘어나고 있다[2]. 이에 따라 모바일 웹 클라이언트 반응 속도 측정도 주요 측정 요소가 되었다.

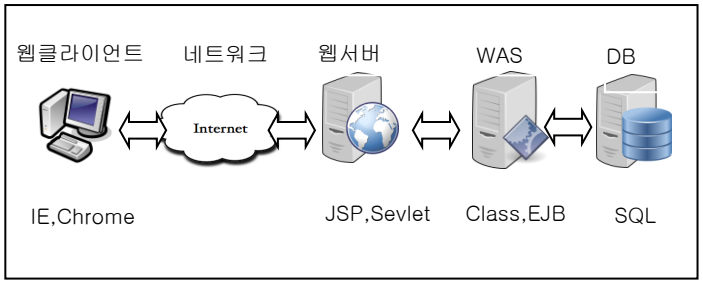
웹 서비스 성능 측정에 있어서 웹 서버 성능 뿐만 아니라 URL내의 호스트명을 IP로 전환 하기 위한 dnslookup 절차를 위한 도메인 네임 서버(DNS:Domain Name Server)연결, Http세션 연결과 같은 네트워크 응답시간, 웹 어플리케이션 서버, 데이터 베이스 응답시간, 사용자의 데스크탑, 모바일 디바이스에서의 반응 속도 등도 매우 중요한 성능 측정 요소이다. 본 논문에서는 이러한 구간 성능 모니터링 필요성을 인지하여 응답 시간에 따른 웹 서비스 성능 구간에 대해 알아보고 웹 서비스 호출에 대한 구간별 성능 측정을 위한 오픈 소스 툴과 이를 활용하여 웹 서비스 호출에 대한 구간별 응답시간 모니터링 시스템을 설계, 구현하였다.

2. 관련 연구

2.1 웹 서비스 성능 구간과 응답 시간

웹 서비스의 성능은 응답시간이라는 정량적인 수치로 주로 표현된다[3]. 응답시간은 사용자 측면에서 웹

서비스의 성능을 평가하는 KPI(Key Performance Indicator)로 사용된다. 응답시간을 구간 별로 구분하면 [그림 1]과 같이 클라이언트 시간, 네트워크 시간, 웹 서버 시간, 웹 어플리케이션 서버(WAS: Web Application Server)시간, 데이터베이스 시간 등으로 구분할 수 있다[3].



[그림 1] 웹 서비스 성능 구간 구성도

웹 서비스의 응답 시간 측정은 소프트웨어 공학의 한 분야인 소프트웨어 품질 관리에서 언급되는 소프트웨어 품질 관리 특성 중 시간 효율성에 부합한 항목이다. 소프트웨어 품질 특성과 메트릭에 관한 국제 표준인 ISO/IEC9126과 소프트웨어 품질 요구사항 및 평가에 관한 국제 표준인 ISO/IEC 25051에서는 소프웨어 품질과 관련하여 기능성, 신뢰성, 사용성, 효율성 등 소프트웨어 품질과 관련한 주 특성을 제안하였다[4]. 표준에 명시되어 있는 항목 중 시간 효율성은 주어진 조건에서 기능을 수행할 때 응답시간, 반환시간, 처리율을 제공하는 능력을 나타낸다[4]. 지속적인 구간 응답시간 모니터링을 통해 성능 지연을 야기하는 구간을 빠르게 인지하여 이를 분석,수정,개선함으로써 소프트웨어 품질을 보다 향상 시킬 수 있다.

2.2 구간 별 성능 수집 방법

2.2.1 웹 클라이언트 구간 성능 수집

웹 클라이언트 구간 성능 측정을 위해서는 다양한 장비와 클라이언트 프로그램의 측정이 필요하다. 구글 안드로이드 OS가 설치된 모바일 기기와 IE(Internet Explorer), Chrome, Firefox, Safari 등이 설치되어 있는 데스크탑에서 웹 반응 속도 측정이 필요하다.

웹 클라이언트 반응 속도를 측정해주는 소프트웨어로는 Fiddler와 같은 프리 웨어 소프트웨어가 있으며 현재 IE, Chrome 등의 웹 브라우저 개발자 도구에서도 클라이언트의 웹 반응 속도를 측정할 수 있는 기능을 제공한다.

WebPageTest와 같이 특정 장비에서 모바일 장비와 데스크 탑 웹 브라우저의 반응 속도 및 컴포넌트 로딩을 위한 네트워크 응답시간 등을 측정해주는 오픈 소스도 있다[5].

2.2.2 네트워크 구간

네트워크 구간 성능은 웹 클라이언트 즉 웹 브라우저에서 웹 컨텐츠에 접근 하기 위해 도메인 네임 서버에 접속하거나 웹 컨텐츠를 로딩하기 위한 다운로드 시간에 영향을 받는다. 본 논문의 2.2.1 “클라이언트 구간 성능 수집 방법”에서 언급한 브라우저 개발자 툴 중 Chrome DevTools의 Network 메뉴는 네트워크 모니터링 툴로 웹 브라우저와 서버 사이의 흐름을 분석해주는 기능을 제공한다. 데이터의 송수신 내역을 쉽게 확인할 수 있으며, 웹 페이지가 필요로 하는 데이터를 서버에게 받아오는 흐름을 볼 수 있다[2].

2.2.3 웹 어플리케이션 서버 구간 성능 수집 방법

금융권이나 대규모 웹 서비스의 경우 높은 비중의 시스템이 자바로 개발된 웹로직, 웹스피어, 제우스와 같은 WAS를 사용하고 있으며 2009년 Node.js가 발표된 이후 Node를 이용한 자바 스크립트 웹 서비스 어플리케이션 개발도 증가하는 추세이다.

과거 웹 어플리케이션 서버 모니터링 툴이 개발 되기 전에는 웹 어플리케이션 코드에 로깅 코드를 삽입하여 어플리케이션 처리 시간을 로그 파일에 추가하여 성능을 모니터링 하는 로깅 방식을 주로 사용하였다. 이후 JDK(Java Development Kit)의 발전에 따라 성능 모니터링을 위한 BCI(Byte Code Injection)를 이용한 프로파일링, javaagent 등 다양한 기술 들도 함께 지원되기 시작했다. 이러한 기술을 바탕으로 사용자의 요청을 WAS에서 실시간으로 모니터링 할 수 있는 APM(Application Performance Monitoring)이라고 불리는 에이전트가 개발되어 지기 시작했다.

현재의 웹 어플리케이션 서버 모니터링은 과거의 로깅 방법과 달리 JMX를 통한 모니터링 방식과 상용 APM들이 사용하고 있는 BCI방식이 주를 이룬다. JMX는 Java Management Extenstion의 약자로 JVM(Java Virtual Machine)이 제공하는 가상 머신 모니터링 및 관리용 MBean API를 호출 하여 JVM의 상태,리소스 사용 정보 등을 조회할 수 있으나 웹 트랜잭션의 성능을 개별적으로 모니터링하는 기능을 구현하기에는 그 기능이 미흡하다. 이에 반해 BCI 기술은 자바의 클래스가 실행되기 위해 자바 가상 머신에 로딩될 때 자바 클래스의 바이트 코드에 모니터링 등을 위한 바이트 코드를 삽입하여 기존 클래스 파일을 조작할 수 있어 트랜잭션 추적 코드 삽입에 용이하다. 이런 BCI 기술을 지원해주는 오픈 소스 라이브러리는 Apache와 같이 널리 알려진 몇몇 오픈 소스 프로젝트 그룹에서 개발되고 있다. 대표적인 BCI라이브러리 목록은 다음과 같으며 이를 활용한 오픈 소스 프로젝트 그룹들도 증가하고 있다.

- BCEL : Apache 프로젝트 제공
- ASM : objectweb 프로젝트 제공
- SERP : SourceForge 프로젝트 제공
- Javassit : JBoss 프로젝트에서 제공

Appdynamics, NewRelic, 제니퍼 등의 상용 APM 제품들과 Stagemonitor, Pinpoint, Scouter와 같은 오픈소스 제품 들도 BCI방법을 사용하여 구현되어 있다.

Node로 구현된 시스템의 경우 자바스크립트로 개발된 스크립트 파일에 성능 수집을 위한 에이전트 코드를 삽입하는 방식의 성능 수집이 가장 대표적인 방법이다. AppDynamic, NewRelic, 제니퍼등의 상용 Node.js 에이전트와 Appmetrics와 같은 오픈소스 등이 코드 삽입 방식을 사용하고 있다.

2.2.4 데이터베이스 구간 성능 수집 방법

데이터베이스는 일반적으로 데이터베이스 제품에서 제공하는 유틸리티를 사용하여 모니터링 할 수 있다. Alert기능, 스냅샵 기능을 이용한 성능 모니터, 통계 보고서 등을 통해 데이터베이스 엔진의 상태 정보나 Late SQL 정보 등을 사용자에게 제공 하고 있다.

상용 데이터베이스 모니터링 툴의 경우 DBA 권한을 가진 로그인 정보로 데이터베이스에 접근한 후 데이터베이스 엔진이 성능 분석을 위해 저장한 테이블에 접근하여 SQL을 이용해 성능 정보를 수집한 후 시각화하여 사용자에게 제공한다. 일부 Oracle 데이터베이스 모니터링 제품의 경우 DBA 권한 없이 데이터베이스 엔진이 사용하는 SGA(System Global Area) 메모리를 직접 액세스 하여 성능 정보를 수집하기도 한다.

웹 어플리케이션 구간별 성능 수집에서 데이터베이스 구간 성능 수집은 Oracle, MySQL과 같은 데이터베이스 엔진의 스냅샷, Late SQL등의 성능 수집보다는 사용자의 웹 요청에 의해 실행된 SQL의 처리 속도 측정에 주 관점을 둔다. 다수의 웹 어플리케이션 서버 모니터링 툴이 웹 요청에 대한 SQL 실행 내역 및 SQL 응답시간, 실행 후 리턴 되는 Results 수 등 다양한 데이터베이스 구간 정보를 수집, 제공하고 있다.

3. 오픈 소스를 활용한 구간별 성능 모니터링 시스템 설계

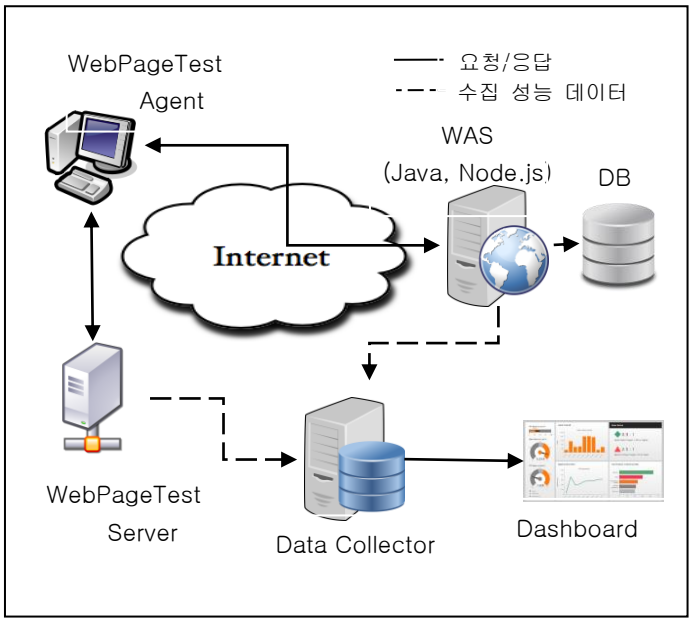
웹 요청을 구간별 측정하기 위해서는 각 구간별 응답시간을 측정할 수 있는 모니터링 에이전트가 필요하다. 현재 github과 같은 코드 공유 시스템에 구간별 모니터링을 위한 다양한 에이전트 오픈소스들이 코드와 함께 공유되고 있으며 이를 활용하여 사용자의 요청에 대한 구간별 소요시간을 수집, 시각화하여 보여줄 수 있다.

3.1 구간별 성능 모니터링 시스템 구조

본 논문의 성능 모니터링 시스템 구조는 [그림 2]로 웹 클라이언트 측정을 위한 웹 페이지 테스트 에이전트와 웹 클라이언트 측정을 명령하고 측정 정보를 수집하는 웹 페이지 테스트 서버로 구성되어 있다.

웹 어플리케이션 서버 구간은 자바로 구현된 웹 어플리케이션 서버(Tomcat등)와 Node.js로 구현된 웹 서비스 시스템을 모니터링 대상으로 한다.

데이터 수집 시스템은 주기적으로 웹 페이지 테스트 서버에 모니터링 하고자 하는 웹 어플리케이션 서버의 URL 정보 등을 전달하고 수집된 성능 데이터를 수신 받아 성능 데이터베이스에 저장한다. 저장된 성능 데이터는 다양한 시각화 시스템을 이용해 대시보드에 정보를 제공하고자 한다.



[그림 2] 성능 모니터링 시스템 구조

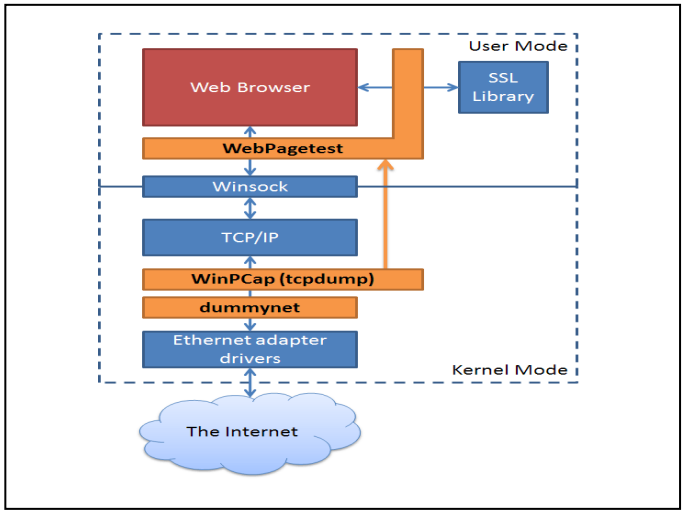
3.2 오픈 소스를 활용한 모니터링 시스템 설계

3.2.1 오픈 소스 웹 클라이언트 에이전트

웹 클라이언트 시스템은 웹 페이지 성능 테스트 오픈소스 중 가장 많이 알려진 WebPageTest 에이전트와 서버를 사용하여 구축하였다. WebPageTest는 세계 각지에 운영 중인 클라우드 시스템에 에이전트를 설치하여 지역별 웹 클라이언트의 반응 속도를 측정할 수 있어 글로벌 서비스를 제공하는 국내외 웹 서비스 기업의 성능 모니터링 시스템에 활용되고 있으며 사용 사례, 사용 기술에 대해 많은 참고 자료를 찾을 수 있다.

WebPageTest 에이전트는 오픈소스 라이브러리의 윈도우 함수 콜을 인터셉션하는 기능을 사용하여 웹 요청이나 응답이 서버나 브라우저로 패싱될 때 이를 인터셉션하여 정보를 수집한다.

[그림 3]은 WebPageTest 에이전트의의 웹 브라우저 성능 수집 아키텍처를 설명한다[6]. WebPageTest에서 사용한 NcodeHook 라이브러리는 먼저 메모리상에서 대상이 될 함수를 찾아 낸 후 메모리의 내의 함수 엔트리에서 성능 수집을 위해 새로운 함수로의 jmp 명령을 삽입하고 기존 함수가 호출될 때 라이브러리에 의해 대치된 새로운 함수가 호출 되어지며 삽입된 함수는 기존 함수로부터 성능 수집에 필요한 정보를 매개변수 전달 등의 방식을 통해 성능 수집을 수행한다[7].



[그림 3] WebPageTest 웹 브라우저 아키텍처

본 논문에서는 WebPageTest가 제공하는 클라인언트 로딩 소요 시간 정보와 다양한 네트워크 성능 지표를 사용하여 클라이언트 구간과 네트워크 구간 응답 시간을 수집하였다.

3.2.2 오픈 소스 웹 어플리케이션 서버 에이전트

웹 어플리케이션 서버 모니터링은 다양한 오픈 소스가 존재하지만 최근 국내에서 가장 성장세가 보이는 Scouter를 사용하여 구축하였다. 2015년 오픈 소스로 공개된 Scouter는 BCI 라이브러리 중 하나인 ASM 라이브러리를 사용하여 웹 요청에 의한 트랜잭션 성능 정보와 Oracle 등의 상용 데이터베이스와 MySQL, Maria 데이터베이스 등을 연동 시 SQL 응답 시간 등의 데이터베이스 구간 성능 정보를 제공하며 자바 가상 머신이 사용하고 리소스에 대한 성능 정보 또한 제공하고 있다.

웹 트랜잭션 성능 수집은 HttpServlet 계열의 클래스가 자바 가상 머신으로 로딩될 때 웹 요청 처리 시작 시 호출되는 메소드에 웹 성능 수집 코드를 호출하는 코드를 삽입(Injection)하여 수행된다. 삽입된 코드에 의해 웹 트랜잭션 처리 시간을 계산하고 웹 요청 시 수신 된 HttpRequest 객체를 읽어 요청자의 IPAddress, Cookie 정보 등도 수집할 수 있다. JDK에서 표준으로 제공되는 HttpServlet 클래스의 경우 service 메소드에 성능 수집 코드가 삽입되며 ServletFilter 클래스의 경우 doFilter 메소드에 성능 수집 코드가 삽입된다. 또한 JDBC 관련 클래스에 대한 성능 수집 코드 인젝션을 통하여 PreparedStatement등의 SQL 수행 시간 등을 수집할 수 있어 본 논문의 데이터베이스 응답 시간 구간 정보는 Scouter APM을 통해 수집하였다.

Node 어플리케이션 성능 수집은 IBM에서 제공한 RuntimeTools 중 하나인 Appmetrics를 사용하여 구축하였다. Appmetrics 에이전트는 본 논문에서 언급한 Jennifer, Appdynamics 등의 사용 Node 어플리케이션 모니터링 툴과 동일한 방법인 코드 삽입 방식을 사용한다. 모니터링하고자 하는 Node 어플리케이션 코드에 Appmetrics를 require 메소드를 사용해 모듈 사용을 요청하고 Appmetrics 객체를 생성한 후 monitor 메소드를 호출하는 간단한 작업으로 Node 어플리케이션을 모니터링하는 에이전트 설치 작업을 완료할 수 있다.

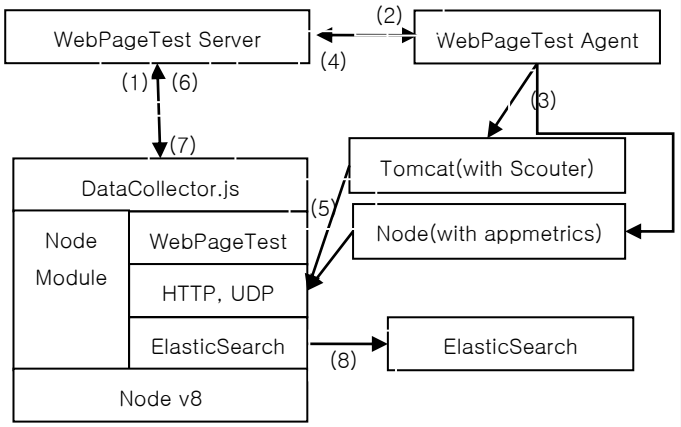
3.2.3 데이터 수집 시스템 설계

데이터 수집 시스템은 WebPageTest 서버로부터 에이전트가 수행한 웹 클라이언트에서의 웹 페이지 성능 정보와 웹 어플리케이션 서버에서 수행한 트랜잭션 정보를 하나의 트랜잭션으로 연결하기 위해 두 구간에서 수집된 정보를 수신하여 저장소에 성능 데이터를 저장한다. [그림 4]에서 데이터 수집 시스템의 주요 동작 절차를 설명하였다.

데이터 수집 기동부인 DataCollect.js 영역은 Node.js로 구현된 시스템으로 성능 테스트 기동을 위한 WebPageTest, 에이전트로부터 성능 데이터를 전송 받을 HTTP/UDP 수신 모듈, 수신한 성능 데이터를 저장할 수 있는 ElasticSearch 모듈로 구성되어 있다.

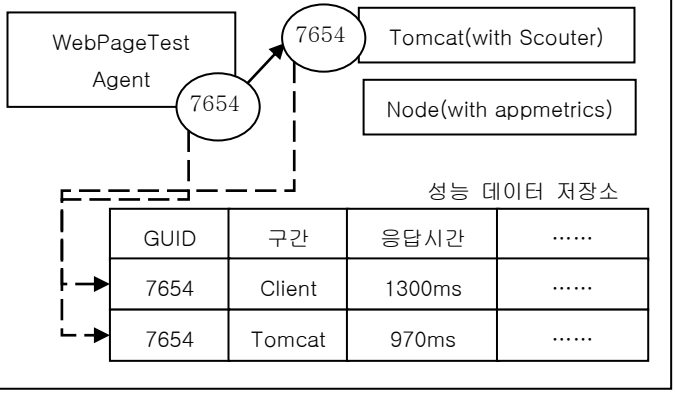
성능 데이터 수집을 담당하는 에이전트 영역은 웹 클라이언트 성능을 호출하고 저장하는 WebPageTest 서버/에이전트, Scouter와 연동되어 있는 웹어플리케이션 서버 모니터링 에이전트, Node 어플리케이션 모니터링을 담당하는 Appmetrics 에이전트로 구성되어 있다.

성능 데이터 저장부는 Apache Lucene를 실무에서 적용하기 위해 개발된 ElasticSearch로 구성하여 병렬 처리와 실시간 검색을 지원하고 확장성이 뛰어나 다수의 에이전트를 이용한 대규모 모니터링 시스템을 구축하고자 할 경우 유용하도록 하였다.



(1) DataCollector 시스템 (DataCollector.js)이 WebPageTest 서버에 테스트 요청. 테스트 URL을 전송하면 테스트 결과 URL이 회신됨.
(2) WebPageTest 서버는 WebPageTest 에이전트에 테스트URL에 대한 웹 브라우저(IE등)테스트 요청
(3) 테스트 URL이 Scouter APM이 설치된 WAS이거나 Appmetrics가 설치된 Node어플리케이션에 대한 URL일 경우 웹 요청 전송
(4) WebPageTestAgent에서 수집된 테스트 URL에 대한 웹 브라우져 성능이 서버로 전송
(5) WebPageTest 에이전트에 의해 호출된 웹 요청을 처리한 WAS(예시:Tomcat)과 Node 어플리케이션은 웹 요청 처리 성능을 DataCollector에 전송
(6) WebPageTest 서버로 웹 클라이언트 처리 성능을 요청(테스트 결과 URL사용)
(7) (6)번의 요청에 의해 Json 타입으로 웹 클라이언트 처리 성능 수신
(8) 수신한 웹 클라이언트 처리 성능과 (5)번의 Scouter와 appmetrics에서 수신된 성능 정보를 ElasticSearch에 저장

[그림 4] 데이터 수집 시스템 동작 절차



[그림 5] GUID를 이용한 구간 데이터 연동

데이터 수집 시스템의 DataCollector모듈에 의해 요청된 웹 호출은 WebPageTest 서버와 에이전트를 거쳐 Scouter 또는 Appmetrics가 설치되어 있는 Node 어플리케이션으로 웹 요청을 전송한다. WebPageTest 에이전트에 의해 호출된 웹 요청을 모든 구간에서 동일 요청으로 인지하기 위해서는 한 번의 웹 요청에 대해 전 구간을 이동할 수 있는 유일한 값을 가진 키(Key)가 필요하다.

WebPageTest 에이전트에서 생성한 유일한 값의 키를 GUID(Global Unique Identifier)라고 하며 성능 데이터 저장소에서 동일 요청에 대한 웹 클라이언트 성능, WAS 성능을 연동할 수 있는 조인 키 역할을 한다. 각 구간 성능데이터는 [그림 5]와 같이 GUID를 전송 받아 수집 구간 성능 정보와 GUID를 함께 데이터 저장소에 저장된다. Http 요청을 통하여 각 구간으로 GUID를 전송하기 위해서는 WebPageTest 에이전트에서 전 구간에서 공통으로 사용할 수 있는 키를 생성하고 Http 헤더 혹은 Cookie를 사용해 키 값을 전달하도록 한다.

3.2.5 오픈 소스를 활용한 시각화

데이터 수집 시스템에 의해 ElasticSearch에 저장한 성능 데이터는 시각화 도구를 사용하여 모니터링 담당자 혹은 웹 비즈니스 담당자에게 유용한 정보를 제공할 수 있다.

Kibana, Grafana 등의 다양한 오픈 소스 시각화 도구가 본 논문에서 사용한 데이터 저장소인 ElasticSearch와의 연동 기능을 제공한다. 특히 Kibana의 경우 ElasticSearch와 함께 제공되는 어플리케이션으로 최소한의 설정으로 ElasticSearch와 연동할 수 있다. 또한 실시간으로 저장소의 데이터를 확인하면서 Line Chart, HeatMap Chart ,Table Chart등을 이용한 대시보드를 빠른 시간 내에 작성할 수 있다[8].

4. 구간별 성능 모니터링 시스템 구현

4.1 데이터 수집 시스템 구현

“3.2.3 데이터 수집 시스템 설계”에서 설명한 데이터 수집 시스템은 구간별 모니터링 시스템의 핵심 요소이다. 데이터 수집 시스템은 WebPagetTest를 통해 웹 클라이언트 성능 수집을 요청하고 수집된 성능 데이터를 저장한다. 오픈 소스인 WebPagetTest 서버 연동, 성능 수집 데이터 송수신을 위한 Http/UDP통신, ElasticSearch 저장 기능 등을 모듈로 제공하는 Node 어플리케이션을 사용하여 구현하였다.

Node.js는 구글 크롬의 자바 스크립트 엔진(V8 Engine)에 기반해 만들어진 서버 사이드 플랫폼이다.

Node.js는 이벤트 기반, 논블록 I/O 모델을 사용해 가볍고 효율적이며 npm이라는 다양한 오픈 소스 패키지 라이브러리를 제공한다[9]. 자바 스크립트 언어를 이용하여 npm 모듈을 활용하면 단 몇 줄의 코딩만으로도 간단한 성능 수집 정보 저장을 위한 Http서버, UDP서버를 구현할 수 있다.

4.1.1 웹 브라우저 성능 수집 기능 구현

데이터 수집 시스템이 WebPageTest 서버 및 에이전트 시스템과 연동하기 위해서는 WebPageTest npm 설치가 선행되어야 한다. Node 어플리케이션에서 WebPageTest를 실행하기 위해서 테스트 URL 혹은 WebPageTest에서 제공하는 다양한 스크립트 명령어를 사용하여 작성한 스크립트를 매개변수로 전달한다.

[그림 6]의 소스 코드는 본 논문에서 사용한 Node 어플리케이션을 WebPagetTest 에이전트를 통해 해당 URL에 대한 IE 브라우저의 웹 페이지 성능 측정을 요청하는 코드이다.

```
const WebPageTest = require('webpagetest');
var script = wpt.scriptToString([
  {navigate: 'http://35.200.63.209:18088/web/board-
list.do?boardId=notice'},])
wpt.runTest(script, {
        connectivity: 'Cable',
         runs: 1
},function (err,result){…}
```

[그림 6] WebPageTest npm 활용 코드

WebPageTest는 웹 브라우저 테스트 후 페이지의 기본 URL 정보 및 웹 응답 헤더, 웹 브라우저 반응 속도 등 20 여개 이상의 다양한 성능 정보를 제공하고 있다. <표 1>은 WebPageTest가 제공하는 웹페이지 요청에 대해 제공하는 주요 성능 지표를 나타낸다.

<표 1> WebPageTest 웹 페이지 주요 성능 지표

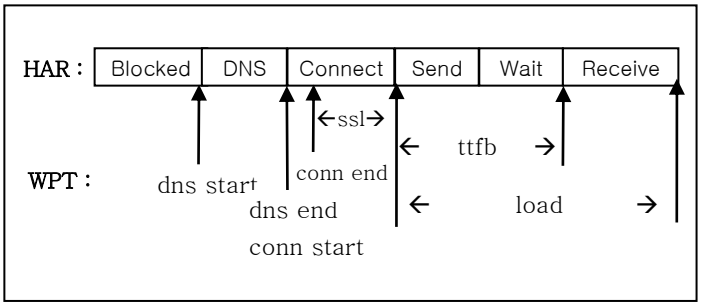
지표명	설명
Fully Loaded	페이지가 로딩되는 데 소요된 총 시간
FullyLoadedCPUms	페이지가 로딩되는 소요된 CPU 시간
DocTime	도큐멘트 객체가 로딩 되는 데 소요된 시간

<표 2>는 웹 페이지 내의 이미지, 스크립트, 미디어 등의 웹 컴포넌트들을 로딩하기 위한 웹 호출에 의해 수집 되는 주요 성능 지표들이다. 본 논문에서 성능 데이터 수집 시스템은 웹 요청에 대한 구간별 응답시간을 모니터링 하기 위해 <표 1>, <표 2>에서 설명한 주요 응답시간 관련 지표들을 수신하여 성능 데이터 저장소에 저장하도록 개발되었다.

<표 2> WebPageTest 웹 컴포넌트 주요 성능 지표

지표명	설명
Time to Load	요청으로부터 다운로드 완료까지 소요된 시간(ms)
TTFB(Time to First Buffer)	요청으로부터 첫 번째 byte 응답까지 걸린 시간(ms)
DNS TIme	DNS 요청 응답 시간(ms)
Connect Time	소켓 접속하는데 소요된 시간(ms)
SSL Time	SSL 연결에 소요된 시간(ms)
Download Time	웹 컴포넌트 다운로드에 소요된 시간(ms)

웹 페이지의 클라이언트 응답시간은 <표 1>에 설명되어 있는 FullyLoaded 값으로 표시되며 네트워크 응답시간은 웹 페이지 컴포넌트들의 DNS Time, SSL Time, Connection Time 등을 합산하여 계산한다. [그림 7]은 HTTP 로딩 정보에 대한 파일 표준인 HAR 포맷과 WebPageTest가 제공하는 성능 지표 간의 비교 항목을 설명한 그림이다. HAR는 2009년 구글 YSSlow의 개발자인 Stwve Soulder와 FireBug의 Net Pandel 개발자인 Jan Odvarko와 HTTP 정보를 저장하기 위해 제안한 표준 포맷이며 HttpArchive 약자를 따 HAR라고 명명하였다. IE, 크롬 개발자 도구, Fiddler, Charlse, dynaTrace등 많은 어플리케이션에서 이를 지원하고 있다. HAR는 페이지에 대한 정보, 요청과 응답 헤더, 리소스의 타이밍 정보 등을 가지고 있다[10].



[그림 7] HAR와 WebPageTest 성능 지표 비교

4.2 GUID 생성 기능 구현

각 구간에서 수집된 성능 데이터를 연동하기 위한 키인 GUID를 WebPageTest 에이전트에서 생성한다.

WebPageTest의 공식 코드 저장소인 github을 통해 WebPageTest 에이전트 소스를 다운 받을 수 있다[11].

WebPageTest 소스는 Agent/웹서버(www)등 웹 페이지 테스트에 필요한 다양한 모듈로 구성되어 있다. WebPageTest 에이전트와 어플리케이션 서버

에이전트와 연동을 위한 GUID 생성 기능을 위해 테스트 요청을 수신한 후 웹 브라우저를 실행하여 웹 요청을 처리해 주는 wptdriver 프로젝트의 HttpRequest 전송 코드를 변경해 주어야 한다. MS 개발 라이브러리가 제공하는 CoCreateGuid API를 사용해 GUID를 생성 후 HttpRequest 헤더를 통해 이를 전송하도록 기존 코드를 수정하여 구현하였다.

4.3 에이전트 수집 데이터 전송 기능 구현

웹 어플리케이션 서버 혹은 Node 어플리케이션에 설치된 Scouter와 Appmertrics의 웹 트랜잭션 성능 수집이 완료되는 시점에 HTTP 헤더를 통해 전송된 GUID와 수집된 성능 데이터를 데이터 수집 시스템으로 전송하는 기능을 구현하였다. 수집 데이터를 전송하기 위한 방법으로는 UDP 통신 방법을 채택하였다.

많은 웹 사용자의 요청을 처리하는 WAS에서 APM에 의해 수집된 성능 데이터를 전송하기 위해 TCP가 제공하는 세션 연결, 유지, 데이터 재전송과 같은 네트워크 작업에 리소스를 소모하는 것은 비효율적이다. UDP는 TCP와 달리 연결 수립을 위한 HandShaking 절차가 필요 없는 비 연결 형 방식으로 대용량 데이터를 전송하기 위해 많이 활용되고 있다[11].

4.4 구간별 응답 시간 시각화

페이지 호출에 대한 웹 브라우저 응답 시간의 주요 성능 지표인 FullyLoaded 성능 지표의 값은 페이지 구성 방법에 따라 웹 페이지에 속해 있는 웹 컴포넌트 로딩을 위해 소요한 네트워크 소요시간의 합보다 작을 수 있다. 이는 웹 브라우저가 웹 페이지내의 모든 컴포넌트를 다운로드 후 렌더링을 하는 싱글 쓰레드 방식이 아닌 웹 컴포넌트를 병렬적으로 로딩할 수 있는 멀티쓰레딩 방식을 사용하여 구현 되어 있기 때문이다.

[그림 8] 멀티쓰레드 방식의 웹 컴포넌트 로딩

[그림 7]은 WebPageTest에서 제공하는 WaterFall Chart로 호출 웹 페이지 내의 컴포넌트들이 백그라운드 쓰레드를 이용하여 병렬적으로 로딩되고 있음을 보여주고 있다. 따라서 본 논문에서 표시되는 웹 클라이언트 응답 시간은 웹페이지를 로딩하기 위해 소요된 전체 네트워크 응답 시간보다 작을 수 있다. 따라서 구간별 응답시간 시각화 시 클라이언트 응답시간 대비 네트워크 응답 시간을 비교하는 것은 적절하지 않다고 판단하였다.

5. 구현 결과

5.1 구현 및 테스트 환경 구축

본 논문에서 제시한 웹 어플리케이션 구간 성능 수집 시스템을 구현 및 테스트하기 위해 구글 클라우드와 MS AZURE를 이용해 WebPageTest 서버와 에이전트를 설치할 플랫폼을 생성 하여 구현 및 테스트를 진행하였다. 테스트 대상이 되는 WAS는 Tomcat8.5로 설치하였으며 Node 어플리케이션은 mongodb를 사용하여 작성된 웹 보드 프로그램을 설치하였다.

데이터 수집 시스템은 1분 주기로 WebPageTest 서버를 통해 테스트 환경인 Tomcat과 Node 어플리케이션으로 브라우저를 통한 웹 페이지 테스트를 요청하였다.

5.2 구현 결과 및 테스트 결과

본 논문은 Java로 구현된 웹 어플리케이션과 Node 어플리케이션의 구간별 응답시간을 모니터링하는 시스템 구현을 주 목적으로 한다. Node.js로 구현된 데이터 수집 시스템의 호출에 따라 WebPageTest와 Scouter를 통해 수집된 성능 정보들이 저장소에 저장된 후 시각화 툴인 Kibana를 이용해 작성한 대시보드를 통해 성능을 확인 할 수 있다.

[그림 9] J2EE 웹 어플리케이션 구간별 성능 모니터링

[그림 9]는 Kibana의 다양한 차트를 활용하여 Tomcat으로 호출된 J2EE 어플리케이션에 대한 클라이언트, 네트워크, WAS, 데이터베이스 구간에 대한 응답 시간 정보를 보여준다. 테스트 어플리케이션의 경우 페이지 내에 컴포넌트가 많지 않아 클라이언트 응답시간 차트와 네트워크 응답 시간 차트가 거의 유사한 형태로 표시된다. Pie chart는 동일 GUID를 가진 웹 트랜잭션에 대한 구간별 응답시간을 분석한 차트이다. [그림 9]에 표시된 J2EE 웹 어플리케이션의

경우 98%(3,846ms)의 시간이 웹 페이지내의 컴포넌트 로딩을 위한 네트워킹 시간으로 소모되었으면 WAS 응답시간은 1.27%(45ms), 데이터베이스 응답시간은 0.59%(21ms)가 소요 되었다. 해당 트랜잭션의 클라이언트 소요시간 1.437로 본 논문 4.4 구간별 응답시간 시각화에서 언급한대로 웹 클라이언트 응답시간이 네트워크 응답시간의 합보다 작은 것으로 나타났다.

[그림 10]은 Node 시스템에 구현된 웹 어플리케이션에 대한 구간별 성능 모니터링 결과를 보여준다. Node 어플리케이션의 경우 J2EE와 모니터링 결과와 유사하게 클라이언트 타임과 네트워크 응답시간 차트가 유사한 패턴으로 표시된다. 클라이언트 응답시간이 1399ms인 웹 요청에 대해 네트워크 요청은 1,782ms로 약 85.92%를 차지하였으며 이에 비해 노드 수행 시간은 292ms로 14.08%를 차지하는 것으로 나타났다. Node 어플리케이션 역시 J2EE 어플리케이션과 동일하게 클라이언트 수행 시간이 웹 컴포넌트 로딩을 위한 네트워크 시간의 합보다 작은 것이 확인되었다.

[그림 10] Node 웹 어플리케이션 구간별 성능 모니터링

Node 구간별 어플리케이션 모니터링의 경우 데이터베이스 구간 응답 시간 모니터링에 대한 응답시간은 제공하지 않는다. 본 논문에서 사용한 Appmetrics 에이전트는 mongodb와 같은 데이터베이스에 대한 성능 모니터링 기능을 제공하지만 데이터베이스 성능 정보가 어떤 웹 요청에 의해서 수행된 정보인지를 구별할 수 있는 방법을 본 논문 연구 기간 내에 찾을 수 없었다.

고가의 상용 모니터링 툴을 사용하지 않고 오픈 소스를 활용하여 웹 서비스 모니터링을 하고자 하는 기업 혹은 개인 웹 서비스 운영자들은 본 논문에서 제시한 방법을 이용하여 웹 요청에 대한 구간별 성능 모니터링 시스템을 구축할 수 있을 것이다.

6. 결론 및 향후 과제

본 논문에서는 웹 어플리케이션 성능을 구간별로 주기적으로 측정하는 시스템을 오픈 소스를 활용하여 설계, 구현하였다. 기존의 웹 서비스 성능 모니터링 방식은 웹 브라우저와 같은 Front-End에서는 Fiddler 혹은 브라우저에서 제공하는 개발자 툴을, WAS나 Node 어플리케이션과 같은 Back-End 시스템에는 APM 에이전트나 로그 모니터링을 통해 개별적으로 성능을 수집하여 모니터링 하였다. 사용자로부터 발생한 요청은 네트워크, 어플리케이션 서버, 데이터베이스 등 다양한 구간을 거쳐 응답을 회신하게 된다. 웹 요청에 대해 각 구간 별 응답시간을 주기적으로 모니터링 할 수 있다면 웹 서비스 지연 시 원인이 되는 지연 구간을 개별 모니터링보다 빠르게 인지 후 분석하여 안정적인 웹 서비스를 제공할 수 있을 것이다.

향후 Node 어플리케이션 호출에 의한 SQL 실행 구간에 대한 응답시간을 수집할 수 있는 방안을 연구하고 구간별 에이전트들이 제공하는 다양한 성능 데이터를 활용할 수 있는 방법을 연구하고자 한다.

본 연구는 과학기술정보통신부 '범부처 Giga KOREA 사업[GK17P0100,Giga Media 기반 Tele-Experience 서비스 SW플랫폼 기술 개발]'의 지원을 받아 수행하였다.

참 고 문 헌

[1] Galdo, Nielsen,Jacob, “International User Interface”, 1996

[2] ParkJaeYang. "전자상거래 웹사이트의 성능 측정 및 향상 방법”, 디지털융복합연구, 제15권 제9호, pp.223-230, 2017

[3] 박성훈, 류성곤, 이원영, “프로젝트 성공을 결정짓는 성능시험 방법론과 실무.” 2017

[4] 컨설팅테스팅 NIPA-STA, “소프트웨어 테스트 실무 가이드”, 2012.

[5] “WEBPAGETEST”. http://www.webpagetest.org/

[6] Stoyan StefanovMeenan, Nicholas ZakasPatrick. “Web Performance Daybook.” ,2012

[7] “NCodeHook”. http://newgre.net/ncodehook

[8] “Elastic Kibana”. https://www.elastic.co/products/kibana

[9] “Node js”. https://nodejs.org/en/

[10] “HTTP Archive (HAR) format”, https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/HAR/Overview.html

[11] “WPO-Foundation/webpagetest” https://github.com/WPOFoundation/webpagetest

[12] 박종선, 김승해, 황건준, 조기환. UDT 기반의 대용량 데이터 전송도구 설계 및 구현. 전자 공학회 논문지-TC, 49(2), pp.23-31, 2012

소프트웨어 제품 운영에 대한 품질 측정

혼 데니스, 홍장의
충북대학교, 컴퓨터과학과
horndaneth@selab.cbnu.ac.kr, jehong@chungbuk.ac.kr

Toward Measuring Quality Factors for Software Product Operation

Horn Daneth, Jang-Eui Hong
Department of Computer Science, Chungbuk National University

Abstract

ISO 9001 is an international quality management system standard applicable to organizations within all type of businesses. ISO 9001 defined software quality is capability of a software product to conform to requirements. Conformance to requirements is tough on reaching the acceptable compromises between stakeholders for the sake of different point of view. However, there can be exist a technique to deal with those complexity issues by defining product operation software quality factors and software metrics to operational factors. Our proposed product operation quality factors show how to provide correctness, performance, reliability, and usability during software product operation phase.

1. Introduction

Software development now a day is suffering from producing the product in different with the stakeholder's perspective due to the failures of defining and measuring criteria for product quality and metrics [1]. Generally the quality requirements of software system were defined during requirement analysis, but it is difficult to preserve the quality requirements because the quality is less visible than functional requirements, and also stakeholders frequently change their minds to more nice system (i.e., higher quality) than initial specification. Therefore, software quality factors must be defined in cleared and detailed manner to achieve the desirable product quality with considering product operation.

Software metric deals with the measurement of software product and software product development process, and it can be used to guide and evaluate software development. Software metric is quantitative measure of the extent to which a system, component, or process. Software factors are going importance and acceptance in corporate sectors as organizations grow nature and strive to improve enterprise quality. The metrics are the quantitative measures of the degree to which software possess a given attribute that affects its quality [2]. In this measuring of quality factors of product operation and metrics documentation, we describe several significant quality factors that could avert unwanted product from stakeholder's viewpoint, and also propose how to measure the quality factors in quantitative manner with smartphone example.

This paper suggests product operation quality factors and their metrics. This quality classification is very meaningful because software product has a life cycle in the development environment like DevOps; in the fast application development approach like Agile and DevOps, the product operation quality can be considered to meet users' satisfaction

The organization of this paper as follows: Section 2 describes current issues of software quality in product operation by citing the existing references. Section 3 explains the quality goals and their metrics by operational quality factors that are correctness, performance, reliability, and usability. The conclusion and some discussion will be described in Section 4.

2. Quality Control Issues in Industry

All the software projects satisfactorily fulfilled the basic requirements for correctness but some projects suffered from poor performance, a few areas were failed to execute caused by unexpected erroneous and spending a loads time to restart the services. There are some examples regarding with these problems:

- "Our new sales information application seems okay, the invoice reports are correct, the inventory records are correct, the discounts granted to our clients exactly

follow our very complicated discount policy, but our new sales information application frequently fails, usually at least twice a day, each time for twenty minutes or more. Yesterday it took an hour and half before we could get back to work". Imagine how embarrassing it is to store managers… soft best, the software house that developed this computerized sales system" [3].

- "The new digital leasing application really have amazing functions by allowing users to apply leasing product even mobile device is in offline state (without internet connect). The application has already processed 5000 user requests. There were no errors. But we did face a severe unexpected problem- when users download application from Google Play or App Store and install into their smart phones take about 1 hour in the initialize or startup application at first using. This is a real problem in user suffering from waiting too long to use the application after downloaded into their smart phone. The development team says that as the application could be able to apply for leasing products in offline mode, so in the initialization stage we need to download all data from sever and store in smart phone storage, an additional in order to solve these problems we decided to develop new digital leasing application".

The cause for these developed software projects in these areas was the lack of predefined for thoroughly of performance, correctness, reliability and usability quality factors that's why this document would be substantially beneficial to solve this issues.

3. Product Operation Quality Factors

We select major four quality factors which can be directly related with product operation as shown in Table 1.

Table 1. Measurement significant quality factors

Quality factors	Definition
Correctness	The ability of software products to perform their exact tasks, as defined by their specification [2].
Performance	Equates performance with efficiency (space and time) [1].
Reliability	The probability of a component, or system, functioning correctly over a given period of time under a given set of operating conditions [1].
Usability	The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use [4].

3.1. Correctness

Correctness is the ability of products to perform their exact tasks, as defined by their specification, without a specification against which to compare behavior, the notion of correctness is meaningless [2].

In terms of product operation, the correctness can be explained with a different aspect. For example, suppose that a requirement for the screen brightness of a smartphone is as follows.

> "Smartphone users should be able to recognize the contents by providing adequate screen brightness, whether daytime or nighttime, indoors or outdoors."

For such quality requirements, it is difficult to clearly define the screen brightness as a predefined Lux value. Therefore, the quality requirements for correctness can be met even when operational environment factors are considered.

Table 2 summarizes the functional requirements and the quality rationale for SMART Self Care Application from this point of view, and Table 3 defines the quality metrics for the application.

Table 2. Example of functional requirements and their quality goals for SMART Self Care Application

Functional Requirements	Quality Goals
As anonymous user, shall select preferred language at first starting app.	Selectivity: In order to use language in app. as user wanted and understand app.
As registered user, should be able to manage his/her phone number plans.	Diversity: In order to accelerate and virtualize his/her phone plans.
As registered user, should be able to give a feedback about applications	Updatability: In order to improve application to adopt with user experiences.
As registered user, should be able to review on callings history by month.	Conformity: In order to view history of calling by month.
The graph to represent data should be identifiable for the difference.	Distinguishable: In order to recognize the difference between data values, intuitively.
The sound should be heard clearly, regardless of the surrounding noise env.	Obstacle-freeness: In order to serve correctly if any difficult condition or situation.

In Table 2, we defined the operational requirements for the SMART Self Care Application and the associated quality goals. From this table, we can know that other different factors should be considered for product operation, compared with the existing definition of the quality, Correctness. The metrics for these quality goals can be defined as shown in Table 3.

Table 3. Correctness metrics for SMART Self Care Application

Correctness metrics	Meanings
Selectivity = A/B	A: Selected item (either 0 or 1) B: Total number of items to be selected
Diversity= A/B	A:Number of selectable items B: Total number of items to be selected
Updatability=A/B	A: Number of feedback by application B: Total number of applications
Conformity= A/B	A: Number of logging by application B: Total number of applications
Distinguishable = A/B	A: Number of visible difference values B: Total number of graph values
Obstacle-freeness = A/B	A: Impedance power of a function B: Impedance power of obstacle

3.2. Performance

Performance is the degree of the application functionalities perform with efficiency time and space. It is a simple fact that if performance is not a stated criterion of the application requirements then the system designers will generally not consider performance issues. While loose or incorrectly defined performance specifications can lead to disputes between stakeholder and development team. In many cases performance requirements are never ridged as system that does not fully meet its defined performance requirements may still be released as other consideration such as time to market [5].

For smartphone applications, users can download and install a variety of applications from Appstore or Playstore, which allows multiple processes (or tasks) to run concurrently or as background processes. In these situations, operational performance can be very different from what was initially predicted or performance estimated during its development. Table 4 summarizes the quality factors and goals for a smartphone application.

Table 4. Sub factors and goals of Performance [5]

Sub Factors	Quality Goals
Response Time	To define period (of seconds) of acceptable responses time in each pages of application functionalities.
Workload	To define of acceptable concurrent transactions in application.
Scalability	To be specified as the extension of the application's workload that the application should be able to process.
Platform	To be defined as the underlying hardware and software (operating system and software utilities) which will house the application.

The metric of "Response Time" can be defined with the difference between the time of service request and the time of reply. The metric of Workload = A/B where, A is the average number of transactions to be executed concurrently, and B is the number of maximum number of concurrent transaction can be processed. The metric of Scalability = A/B, where A is the maximum number of additional entities, and B is total possible number of entities onto platform capability. The metric of platform can be defined with the division of the number of supporting OSs by all number of existing OSs.

3.3. Reliability

Reliability requirements deal with failures to provide right services. They determine the maximum allowed software application failure rate, and can refer to the entire application or to one or more of its separate functions [1]. The examples of reliability requirements are as followings;

1) The failure frequency of an application that will operate is required to be less than one in 1 year.
2) Activate/deactivate or subscribes any plans or services, balance enquiry, and report functions is required to have a failure rate of less than one per million cases.
3) Application System, which operates 10,000, 000 users, is that it will not fail, on average, more than 10 minutes per month during this 8 AM to 20 PM.
4) The probability that the off-time (the time needed for repair and recovery of all the service) be more than 30 minutes.

One of often used measure of reliability in software system is mean time between failures (MTBF). The sum of mean time to failure (MTTF) and mean time to repair (MTTR) gives the measure [6].

$$MTBF = MTTF + MTTR$$

Recently, almost software systems were operated in connection with the Internet. Therefore, the ability to cope with defects is also fast and excellent. In the case of smartphone, background process for automatic update is always running. Therefore, if developer found any potential defect of smartphone application, the updated version of the application is pushed and automatically recovered without user intervention.

Therefore, although general metrics of reliability has considered the repair time to fix the defect, it is now possible to consider only nearly zero update time instead of developer's repairing time.

Table 5 defines the quality factors for Reliability of a smartphone application. And also explains why those factors should be required in the product operation.

Table 5. Sub factors and their goals of the quality Reliability

Sub Factors	Quality Goals
Latent fault density	To define the frequency of failure occurrence that leads to unavailability of a service during given period.
Updatability	To define automatically update a software service that has latent fault or failure occurrence under user's unconscious.
Resilience	To define the provisioning of the continuity for a service even any hidden (invisible) failure like network breakdown occurs.
Restart ability	To define how often the system (or service) can restart within predefined time to provide serviceability.

The sub factors for the Reliability defined in Table 5 can be measured using the metrics as shown in Table 6.

Table 6. Quality metrics for Reliability

Reliability metrics	Meanings
Latent fault density = (A1-A2)/B	A1: Total number of latent faults in a software(or service) A2: Total number of actual faults B: Software product size
Updatability= A/B	A: Total number of updated automatically B: Total number of applications
Resilience=A/B	A: Number of visible service halts B: Total number of breakdowns
Restart ability= A/B	A: Number of restart within required time during product operation. B: Total number of restarts during product operation.

3.4. Usability

Usability requirements deal with the user satisfaction in a specified context of using application. It is important to consider the following three aspects of usability for all types of software [5]:

1) More efficient to use: takes less time to navigate a particular function.
2) Easier to learn: operations can be learned by observing the object.
3) Graphic user interface: provide convenient interface to use functions easily.

Hence, there are several user characteristics that influence the use of application [7]:

1) User experience with particular type of device and specific device, user ages, gender, user level, the nature of education, occupation
2) Device/Equipment: A mobile device differs from another with several aspects of form and design as the keyboard type, screen sizes, and storage capacity. Also a mobile application can be launched on a smartphone, or on a tablet under different OS.

Table 7 summarizes the sub factors and quality goals for usability requirements of the SMART Self Care Application. This sub factors were defined with consideration of product operation in user perspective. There can be existed other sub factors for usability of software system, but we just consider practical issues in smartphone application.

Table 7. Example of SMART Self Care usability requirements

Sub Factors	Quality Goals
Users experience	Application must design following by Apple IOS human interface guidelines [8] for IOS platform and Google User Interface Guidelines [9] for android platform.
Device interoperability	Application must support IOS/Android tablet and smart phone devices with storage capacity 2G up. Application keyboard must use Swype keyboard.

The two sub factors shown in Table 7 can be defined as following metrics;

$$\text{User experience} = \frac{\text{Number of rule compliance by application}}{\text{Total number of rules in the guideline}}$$

$$\text{Device interoperability} = \frac{\text{Number of supported platforms}}{\text{Total number of existing platforms}}$$

Where, the term platform in the above metrics includes middleware, operating systems and hardware.

4. Discussion and Conclusion

To achieve the acceptable compromises between stockholders is a complex engineering activity. It involves interactions between development teams, customers and related person to develop a completely product. Consequently, tension between development teams and customer would probably arise in coincidentally. In this paper, we have identified the need for defining product operation quality factors and metrics would be a prominent role to sort out tension between development teams and customer and producing undesired products.

In this paper, we selected four quality factors; correctness,

performance, reliability, and usability as the operational quality characteristics for software products, especially smartphone applications. And also, we explained how these quality factors should be considered in the operation of the product. Especially the sub-factors of each quality factor are defined and their quality metrics are suggested.

When considering various software products, more quality factors for product operation may be considered. In particular, quality factors such as safety, variability, and security have been studied extensively. These quality factors can be important as the operational qualities in areas of safety-critical or mission-critical systems.

Another issue to be discussed in this paper is on the metric of each quality factor. These metrics may be somewhat variable in the different operating environment or in the context of software product. Even in the case of performance, there is no restriction on the response time in a normal operation, but in an emergency situation, a faster response time may be required.

This paper suggests product operation quality factors and their metrics. This quality classification is very meaningful because software product has a life cycle in the development environment like DevOps; in the fast application development approach like Agile and DevOps, the product operation quality can be considered to meet users' satisfaction. Therefore we consider operational quality is more critical rather than process quality or product quality.

Acknowledgement

This research was supported by Next-Generation Information Computing Development Program through the NRF funded by the Ministry of Science and ICT (NRF-2014 M3C4A7030505)

Reference

[1] Ian Sommerville, Software Engineering (10th Edition), Pearson, 2015
[2] Daniel GALIN, Software Quality Assurance from theory to Implementation, Addison Wesley, 2004
[3] Software Correctness Property: Correctness is a relative notion: https://accu.org/index.php/journals/1585, at November 2017
[4] Write Performance Requirements, http://www.1202performance.com/atricles/how-to-write-performance-requirements-with-example/, at November 2017
[5] Fatih Nayebi, Jean-Marc Desharnais, The State of the art of mobile application usability evaluation, 25th IEEE Canadian Conference on Electrical and Computer Engineering, 2012, 4 pages.
[6] Karima Moumane, Ali Idri, and Alain Abran, Usability evaluation of mobile applications using ISO 9241 and ISO 25062 standards, SpringerPlus, 5(548), 2016, 15 pages
[7] Apple iOS Human Interface Guidelines: http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html, at December 2017
[8] Google User Interface Guidelines: https://developer.android.com/guide/practices/ui_guidelines/index.html, at December 2017
[9] Ming-Chang Lee, Software Quality Factors and Software Quality Metrics to Enhance Software Quality Assurance, British Journal of Applied Science & Technology, 4(21), 2014, pp.3069-3095

An Observation of Use Case and Misuse Case in System Theoretic Process Analysis

Anit Thapaliya, Gihwon Kwon

Department of Computer Science, Kyonggi University

{anitdgret, khkwon}@kgu.ac.kr

Abstract

Use case diagram is a representation of user's interaction which can recognize the impact on the use cases and involved actors. In this paper, use case and misuse cases are used to elaborate system theoretic process analysis (STPA). We generate a set of basic software safety requirements from misuse cases based STPA. This provides a pathway where system requirements can be identified as misuse cases and analyze their connection in terms of failure. The results of misuse cases analysis describe the basic inputs for traditional STPA. Unlike to traditional STPA inputs where hazards and accident are taken from real world scenarios, brainstorming and domain knowledge, but this paper present a framework to generate hazard and accident from misuse case diagrams. A case study of railway level crossing system is also provided to verify the applicability of proposed method.

1. Introduction

Software safety doesn't have a clear and concise definition in nature, the portion of software in safety critical system is deep-rooted. Safety related problems are both manually and automatically handled in safety engineering process. This inherits the difficulties in verifying the safety aspect of the software system. In other word, both manual and automatic safety analysis are more based on the analyst domain knowledge and experiences. This is the reason why there is no guarantee of fully safe software system.

In the modern computerized era of science and technology, the safety aspect of critical systems is normally handled by using reliability and system theory-based safety analysis. These analysis techniques are based on the informal specification of the system [1].

In addition, the design and development of safety critical system requires a joint effort of system engineering, software engineering as well as safety engineering. To be more specific in software and safety engineering, safety analysis techniques requires system descriptions, general assumptions and requirements from the stakeholders. This information provides the basic inputs for safety analysis.

This research was supported by the MSIP (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2017-2015-0-00445) supervised by the IITP (Institute for Information & communications Technology Promotion)

In traditional techniques such as fault tree analysis (FTA), failure mode and effect analysis (FMEA) the input parameters are taken from the system description along with requirements, but in modern techniques such as system theoretic process analysis (STPA) uses hazards and accident as input parameter [2]. These hazards and accident are derived manually from the system description using the domain knowledge and expertise of analyst.

In this paper, we present misuse case-based safety analysis through system theoretic process analysis. This process includes misuse case diagram (MUC) generated from the system description, requirements and assumptions made before the safety analysis. Use case diagrams can describe the reliability of software system through a sequence of actions that required in examining functional requirements [3].

Misuse case diagrams are an extension of use care diagrams which traditionally used to generate security requirements in software system [4]. This is because UML use case diagrams are not suitable for eliciting non-functional requirements such as security. This paper presents an experimental use of misuse case diagrams in safety analysis. Similarly, the outputs of misuse case diagrams are considered as an input to STPA based analysis.

In other word, this paper aims to generate inputs for STPA analysis from misuse case diagrams. Misuse case requirements modeling includes the all the function that the system should not allow, all the

miss-actor who is responsible for the misuse cases etc. In other word, traditional STPA uses system requirements directly as an input, but this paper focuses on requirements modeling using misuse case diagram before the actual STPA analysis. We investigate how system requirements can be modeled into STPA inputs to generate software safety requirements.

The rest of the paper is structured as follows: Section 2 reviews background of proposed concepts and related work. Section 3 presents the proposed method for the experiment in details. Section 4 presents the experimental case study, and section 5 presents the discussion about the proposed method from the experiment. Section 7 concludes the paper and indicates some directions for future work.

2. Background

Unified modeling language (UML) is a standardized language for modeling object-oriented software. UML can describe two aspects of software system such as static and dynamic structure [5]. Use case diagram, class diagram is usually used for describing the static structure of the system including the characteristic of the software system. Activity diagrams and interaction diagram is usually used for describing the dynamic structure of the system.

2.1 Misuse case

Being an extension of UML use case diagram, misuse case diagrams are widely used for eliciting security requirements in software system. A misuse case is triggered by the actor hostile to the system. It describes the threats posed by a hostile actor to the system including the user errors and faults [4, 6].

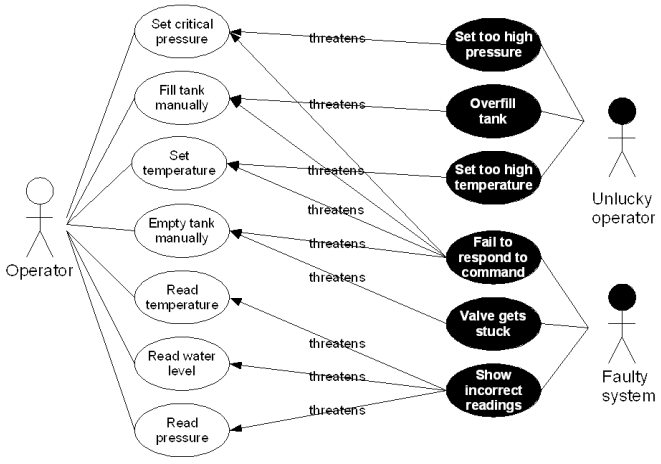


Fig. 1 Example of Misuse case

The above Fig. 1 represents the overview of misuse case diagram [4]. In the figure, the normal use cases are represented in oval shape without color, whereas the misuse cases are represented by color oval with the corresponding actors who are responsible for the misuse cases. The normal use cases are very effective in modeling the user requirements in software development. The misuse cases represent the threat or damage imposed by the faulty actors in the system.

2.2 System theoretic process analysis

System theoretic process analysis (STPA) is a modern safety analysis technique originally proposed by Leveson to identify the causal factors leading to possible accident in system theoretic accident model (STAMP). It involves identification of accidental scenarios based on the design error, requirements fault and other human related factors.

The major goals of STPA is to address the limitation imposed by traditional hazard analysis techniques in the field of modern complex systems. STPA is a systematic approach which focuses on the view of safety control issues [7]. In other word, STPA identify all the potential inadequate control of system which could lead to the possible accident in the future. This is because a hazardous state occurs because of poor system level thinking such as poorly performed control actions for safety, unsafe control actions provided [8]. Fig. 2 describes the foundation of STPA analysis based on control theory with process model [9].

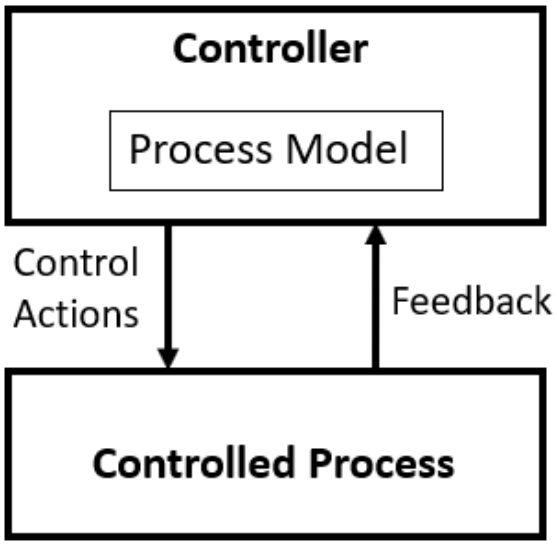


Fig. 2 STPA Overview

3. Proposed Methodology

Generating software safety requirements always starts with hazard analysis followed by risk assessment. In safety software development life cycle, various methods are proposed for hazard identification and risk assessment. Yet, the basics of these methods are based on historical data, brainstorming and expert domain knowledge. This paper presents a methodology to generate software safety requirements from system specification using misuse case diagrams and system theoretic process analysis.

Traditionally, safety analysis technique such as FMEA, FTA starts with system requirements in addition with several assumptions, experiences and domain knowledge. Similarly, STPA also uses hazards and accidents from domain knowledge. But, this paper aims to introduce misuse case diagrams before the actual safety analysis. From the misuse case, we generate a list of hazards and accident as an input for STPA analysis.

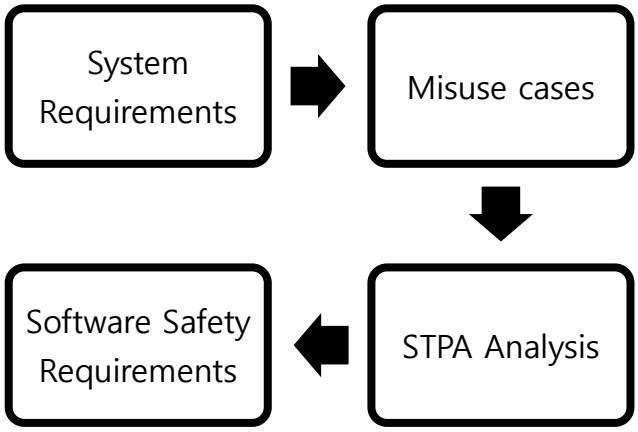


Fig. 3 Proposed Methodology

The above Fig. 3 describes the proposed methodology of safety analysis. We suggest four different steps to generate software safety requirements starting from system requirements. In step 1, we describe the system requirements of proposed system which includes user's requirement, safety plans, general guidelines and standards etc. In step 2, we generate misuse case diagram from system specification. This diagram reflects two different aspect of requirements such as normal use cases for describing functional characteristic of system and misuse cases for describing safety related characteristic of system. This safety related characteristic includes those factors responsible for the misuse cases. For example, those behaviors that the system is not intended to do triggered by some errors or faulty system or external users. We evaluate misuse case diagram to generate hazards and accident. MUC is more useful to analyze failure mode related to user's interaction in the system [10].

In addition, MUC enforce the possible accident in the future if specific control mechanism doesn't exist in the system. Hence, the outputs from MUC concentrates on system safety and high-level hazards or risks. In step 3, the actual STPA process starts with hazardous behaviors by identifying unsafe control actions, followed by the hierarchical safety control structure. This process includes modeling the original control structure diagrams with process model. To be more specific, we select a set of unsafe control actions applicable in railway system to identify hazardous control actions root causes. After analyzing all the hazardous control action and their corresponding accidental scenario's software safety requirements are generated in step 4.

4. Case Study

In this paper, we demonstrate our approach on the case study of railway level crossing system. Level crossing is an intersection point between a railway line and a road or a path. In other words, level crossing is a term used for describing a section where railway track crosses the roadway.

This paper case study is more focused on the system requirements of railway level crossing system. Among the different types of level crossing accidents in the world, like train – vehicles collision, train – pedestrian's collision, this case study only focuses on the accidental scenarios of train and vehicles collision. This case study is concerned with a sensor-based crossing controlling system used in the intersection area between the track and the road as illustrated in Fig. 4.

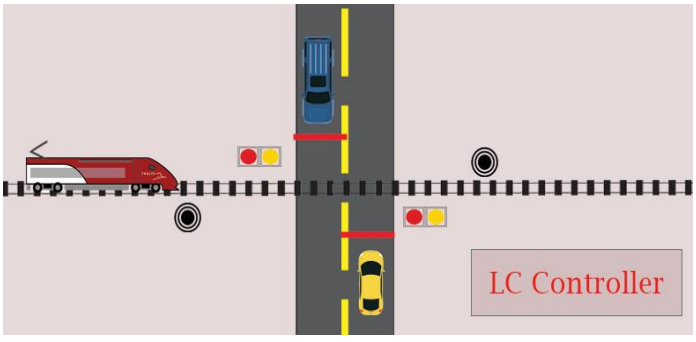


Fig. 4 Level Crossing System Overview

This case study consists of two sensors, two gates and one controller which control the overall working mechanism of the level crossing system. Sensor is used to detect the train in the level crossing area and the gates are used as the barriers for the roadway to block the entry of vehicles in the track. Similarly, the lights, red and yellow, are for the roadway traffic indicating track busy (red) and track free (yellow). Additionally, there are some basic considerations for the case study in terms of requirements such as the reliable configuration of level crossing gate and lights, reducing the accidental scenarios through the reliable safety control mechanism etc. With these generation system specification, we draw a misuse case diagram for level crossing system as mentioned in Fig. 5. This diagram consists of two different aspects of system, one as a normal requirement modeling based on the functionalities and second as misuse case-based requirement modeling with possible failure in the system.

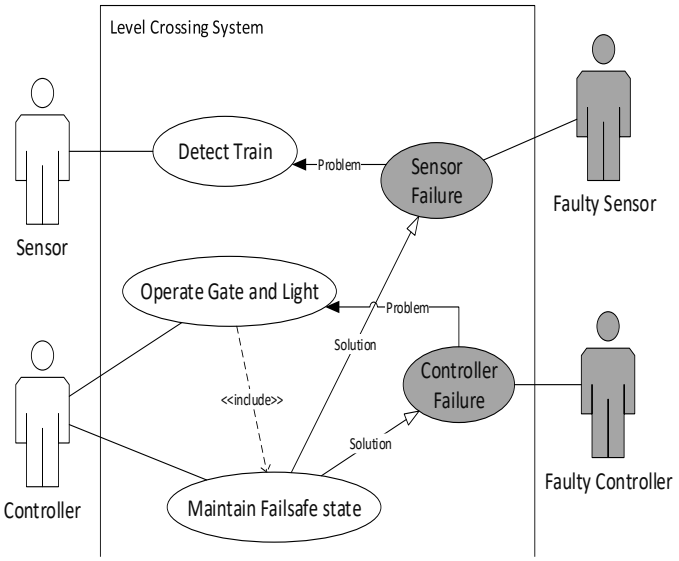


Fig. 5 Misuse case Diagrams

In the left-hand side of the figure, we can see major actors responsible for the normal operation i.e. uncolored use case in Fig. 5. On the other hand, there are faulty actors responsible for possible failure in the system. In general, our consideration focuses on the statement that a major actor in the use case diagrams will eventually be a faulty actor due to internal or external factors in the system. Similarly, the light black coloured misuse case imposed a threat to normal use cases which could eventually lead to system failure. During misuse case-based requirement modeling, we consider a failsafe use case as a mitigation for misuse case. Moreover, a failsafe use case diagram acts as a safety constraint for level crossing system.

S.N.	Use case Name	Description
1	Detect train	Whenever the train approach, sensor is activated and forward the train status to controller
	Precondition: sensor ready and train approaching	
	Post condition: forward train status to controller	
2	Operate gate and light	Whenever the controller receives train status, the controller operate gate accordingly
	Precondition: receive train status	
	Post condition: open/close the gate and turn light red/yellow	

Tab. 1 Use case Specification

In Tab. 1 we explain the first aspect of misuse case diagram of Fig. 5 using use case specification. This table describes the normal scenarios of level crossing system. Similarly, Tab. 2 describes the second aspect of failure propagation and its effects in Fig. 5. From the table, we investigate misuse cases such as sensor failure and controller failure using use case specification. In general, use case and misuse case specification contain actors for responsible for the corresponding use cases, pre and post condition of related use cases with simplified description.

S.N.	Misuse case Name	Description
1	Sensor failure	In case of sensor problem, this misuse case provides a threat to detect train
	Precondition: faulty sensor or sensor not ready	
	Post condition: unable to detect the train [Hazard 1]	
2	Controller failure	In case of controller failure, this misuse case provides threat to operate gate and light safely
	Precondition: faulty controller or controller not ready	
	Post condition: unable to operate gate and light safely [Hazard 2]	

Tab. 2 Misuse case Specification

From Tab. 2 we identify hazards from post condition of each misuse cases. In fact, these hazards are the outcomes of each faulty actors of the system. The identified hazards are mentioned in Tab. 3 below.

S.N.	Hazards
H1	Unable to detect train
H2	Unable to operate gate and light

Tab. 3 List of Identified Hazards

This identified hazards list works as an input for next STPA analysis. After identifying hazards, we evaluate failsafe use case diagrams to generate initial safety constraints for those hazards identified in misuse case specification. The precondition at this point those misuse cases based on faulty actors. And the post condition imposed the required safety actions for those misuse cases.

S.N.	Use case Name	Description
3	Maintain Failsafe state	As a solution for faulty sensor's and controller's threats, this use case prevents the system from level crossing accident i.e. Train and Vehicle collision [Accident 1]
	Precondition: faulty sensor or faulty controller related misuse cases	
	Postcondition: If unable to detect the train or the case with controller failure, close gate and turn light red [Safety Constraint]. Else, open gate and turn light yellow accordingly when train is not approaching.	

Tab. 4 Use case Specification for Failsafe

From this failsafe use case specification in Fig. 4, we identify accident from the description and safety constraints from the post condition. This safety constraints further explain in terms of control action in the next steps of proposed methodology. In fact, safety constraint aims to prevent the accidental scenarios defined in the case study.

Considering the viewpoint of proposed methodology, the outputs of misuse case diagram is an input for STPA analysis. In general, system theoretic analysis uses hazards list and accidental scenarios because of their past experiences and domain expertise. Here, we integrate use case and misuse specification in order to generate a list of hazards their consequences in terms of accident and preventive safety constraints.

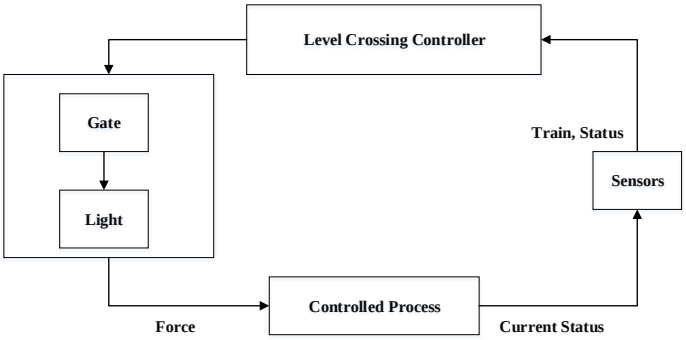


Fig. 6 Control Structure Diagram

Fig. 6 presents the control structure for level crossing system. The inputs such as hazards and accidental scenarios are taken from earlier misuse case analysis from the case study. Again, we model the functional control of the level crossing system using the control structure diagram having controller, sensors, actuators and control actions such as "open/close gate" and "turn red/yellow lights" are generated using failsafe use case specification.

In fact, these control actions are general for normal railway level crossing systems but in this paper, we are more focused on safety related control actions to prevent the accident. Similarly, after control structure diagrams we further analyze the unsafe control actions using the lead words from system thinking. The STPA provides 4 different lead worlds to analyze the unsafe control actions such as 'wrongly provided', 'not provided', 'wrong time' and 'stop too soon'. The analysis results are mentioned in Tab. 5 with the lead words from STPA.

Control Actions	Wrongly Provided	Not Provided	Wrong Time	Stop too Soon
Open Gate	Hazard	Safe	Hazard	N/A
Close Gate	N/A	Hazard	N/A	N/A
Turn Red Light	N/A	Hazard	N/A	N/A
Turn Yellow Light	Hazard	Safe	Hazard	N/A

Tab. 5 Unsafe Control Actions

According to the table, the safety of level crossing system depends of inadequate control of 4 different control actions identified using misuse case diagram.

The identified control actions can be critical for the system in following ways:

1. Wrongly provided: The level crossing system open the gate when the train is approaching or crossing. And, the system turns yellow light on when the train is approaching or crossing. Similarly, whenever the system closes the gate and turn red light on, then there will no hazard as the vehicles cannot are not permitted to pass the crossing. In fact, in this paper, we consider N/A inadequate control for this scenario but in general if the vehicles are struck in between the gates then it will be the critical condition.
2. Not Provided: The level crossing system does not close the gate whenever the train is approaching or crossing. This is another unsafe control for level crossing system which is the same to 'turn red light" control action in the case study.
3. Wrong time: The level crossing system open the gate before the train is leaving. Similarly, whenever the system 'turn yellow light' before the train is leaving is another unsafe control.
4. Stop too soon: The unsafe control generated by this lead word is quite deviated from our case study and it's not applicable. But still if the system closes the gate too soon or for the short time and then again open the gate then it will be critical for level crossing system.

The results of inadequate control are transferred in the following Tab. 6, which describe the applicable unsafe controls of the level crossing system. In the table, we extract only applicable unsafe controls that could lead to the potential accident.

In other word, unsafe control related to control actions 'close gate' and 'turn red light' are the same in meaning as control actions 'open gate' and 'turn yellow light'. For example, the unsafe control related to control action 'open gate' i.e. the system opens the gate when the train is crossing is directly equivalent to the unsafe control related control action 'close gate' i.e. system does not close the gate when the train is crossing.

S.N.	Control Actions	Unsafe Control Action (UCA)
1	Open gate	UCA1. The system opens the gate when the train is approaching or crossing [Hazard 1,2] UCA2. The system opens the gate before the train is leaving [Hazard 1, 2]
2	Turn yellow light	UCA3. The system turns yellow light on when the train is approaching or crossing [Hazard 1, 2] UCA4. The system turns yellow light before the train is leaving [Hazard 1, 2]

Tab. 6 Unsafe Control Related to Case Study

Likewise, unsafe control actions and their corresponding safety constraints are generated as mentioned in Tab. 7, which describes the unsafe controls based on the external factors like train status.

Unsafe Control Actions	Safety Constraints
UCA1	The gate must not be opened while the train is approaching or crossing
UCA2	The gate must not be opened before the train leaving
UCA3	The yellow light must not turn on while the train is approaching or crossing
UCA4	The yellow light must until the train is leaving

Tab. 7 UCAs and Safety Constraints

After identifying unsafe controls and safety constraints, the control structure diagram presented in figure 6, can be transferred into safety control structure diagram with process model variables. These process model variables and basic feedback from the safety constraints can model a safety control structure diagram [8]. Fig. 7 describes the safety control structure for the proposed case study. It contains 5 different process model variables generated from unsafe controls and safety constraints such as Train containing 3 different variables approaching, crossing and leaving the level crossing area, gate containing 2 variables up and down, light with containing 2 variables red and yellow, sensor with activated and deactivated to identify the train status and lastly, process model status with safe and failsafe. Safe denotes the normal operation of level crossing system

whereas failsafe denotes the emergency operation of the system. On the other hand, safety control structure diagrams also contain the control actions generated from misuse case diagrams.

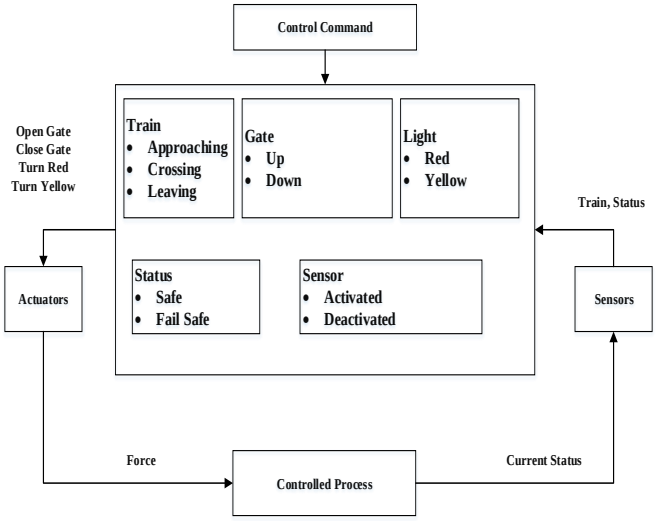


Fig. 7 Safety Control Structure Diagram

Similarly, after generating safety control structure for level crossing system, UCAs their corresponding casual factors are generated along with relevant controls as mentioned in Tab. 7

UCA	Casual Factors	Controls
1-2-3-4	a) Incorrect sensor inputs b) Sensor malfunction c) Faulty control algorithms d) Controller malfunction	a) Provide additional safety inspection to ensure the reliability of sensors and controller b) Provide failsafe mechanism to prevent and protect from the possible accident in the future

Tab. 8 Casual Factors and Controls

At the end of this safety analysis, detailed software safety requirements are generated in the system level of the level crossing system. These software's safety requirements are the results of different aspect of STPA analysis including the analysis of unsafe control actions, safety control structure, safety constraints, casual factors and their controls. The generated software safety requirements are mentioned as follows.

S.N.	Software Safety Requirements (SSR)
SSR1	Open gate command must not be provided while the train is approaching or crossing.
SSR2	Open gate command must not be provided before the train is leaving.
SSR2	Close gate command must be provided while the system status is failsafe.
SSR3	Turn light yellow must not be provided while the train is approaching or crossing.
SSR4	Turn light yellow must not be provided before the train is leaving.
SSR5	Turn light red must be provided while the system is in failsafe mode.

Tab. 9 Generated Software Safety Requirements

5. Discussion

By applying this misuse case-based framework for safety analysis, it is feasible to extract hazards from the faulty actors in the use case diagrams. On the other hand, it is possible to reduce the theatrical gap in STPA analysis input parameters. In fact, identifying hazards, accident and initial safety constraints before is very useful to quantify the result of STPA. From the experiment, we present one accident scenarios of level crossing accident using the misuse case-based system requirements modeling and generate 5 different software safety requirements.

Similarly, the proposed method supports the early safety consideration during requirement engineering phase in safety software development life cycle. The transition of traceability in proposed method make it more comprehensive while interpreting the failure in system level. Likewise, the early focus on the safety maintain the balance in the traditional approach where safety considerations are usually taken at the design phase and are more focused in the failure of physical components rather than the user related interaction and software requirements faults. Using STPA to describe human related and software requirements related faults in misuse case diagram is another strong direction of this research. The usual STPA process are followed only after the misuse case analysis which usually covers both physical component failure and the involved actors. Hence, the usual STPA approach is completely different than the proposed framework. This is because, the traditional misuse case diagrams are used to reflect the human interaction related failure and threats in refining the security requirements.

But, we present our proposed technique in the

laboratory case study where some scope of safety analysis is limited. This results the practical deviation of this method in the real time framework.

6. Conclusion and Future Work

From the experiment, we present a misuse case based STPA analysis for safety critical level crossing system. The proposed method starts with general system requirements specification followed by use case and misuse case-based requirements modeling. The use of UML use case diagram and misuse case is to generate basic list of hazards and their possible accident in the safety critical system. Normally, STPA analysis uses hazards and accident as the input for safety analysis but there is not a clear direction on how to generate a list of hazards and their related accident. This paper aims to provide a use case and misuse case-based hazards generate framework for STPA analysis. As a result, this paper concludes that it is possible to use an extension of UML case diagram to generate the basic parameter for STPA analysis. This is because use case diagrams can model the requirement of the system as well as misuse case can represent the actual failure flow in the system due to the faulty actors.

Similarly, from this research it can also be concluded that the basic UML use case diagrams and its extension can be used for safety analysis in critical system. The pictorial representation of use case diagrams is easy to understand and is very useful to represent the threat caused by the faulty actors in the normal use case representing the functional behavioural of the system.

Lastly, in addition to this the future research dimension might be the use of other UML diagrams in STPA analysis. On the other hand, another direction is to study the analysis and comparison of misuse case in reliability-based safety analysis techniques such as fault tree analysis (FTA), failure and effects analysis (FMEA) and investigating the generated software safety requirements.

References

[1] E. Troubitsyna. Integrating Safety Analysis into Formal Specification of Dependable Systems. Proceedings International Parallel and Distributed Processing Symposium, pp.8, 2003

[2] Nancy Levenson, Engineering a Safer World. MIT Press, 2011

[3] Neledath Damodarn, Secure Software Development Using Use Cases and Misuse Cases. Issues in Iformation Systems, pp.1-2, Vol. VII, 2006

[4] Guttorm Sindre and Andreas L. Opdahl. Capturing Security Requirements Through Misuse Cases. Norsk Informatikkonferanse, 2011

[5] Jacobson. Object Oriented Software Engineering A Use Case Driven Approach, Boston, 1992

[6] Ian Alexander. Misuse Cases: Use Cases with Hostile Intent. IEEE Software, 2003

[7] Fierab Mason Blakley and Ryan Habibi. Prospective Hazard Analysis for Information Systems. IEEE International Conference on Healthcare Informatics, 2014

[8] Nancy Levenson, A New Accident Model for Engineering Safer Systems. MIT Press, 2004

[9] John Thomas. Extending and Automating a Systems-Theoretic Hazard Analysis for Requirements Generation and Analysis, Ph.D. Dissertation, MIT, Boston, 2013

[10] Tor Stalhane and Guttorm Sindre. A Comparison of Two Approaches to Safety Analysis Based On Use Cases. International Conference on Conceptual Modeling, 2007

MR3 엔진 ASM 단계 성능 비교

Performance Evaluation of the ASM Layer of MR3

Hoang Le, Sungwoo Park
Pohang University of Science and Technology (POSTECH)
77 Cheongam-Ro. Nam-Gu. Pohang. Gyeongbuk. Korea 37673
{lemin1991,gla}@postech.ac.kr

Abstract

This paper presents a performance evaluation of the ASM layer of the MR3, which is a next generation computing engine on Hadoop. Firstly, we briefly explain MR3's core layer, how it organizes and manages service components. Secondly, we propose a design of ASM layer, which is mainly responsible for processing and computing data. More specifically, we present a design for text grep application that can filter text data with a given pattern as input arguments. Finally, we demonstrate through extensive experimentation that with the map-only text grep program, our design performs better than MapReduce2 and Apache Tez.

Keywords: parallel, concurrent, distributed system, Hadoop.

1. Introduction

"Big data" describes data sets that are large, complex and grow quickly in volume. Big data is important, and it is transforming how people manage their businesses.

To retrieve information from big data, conventional applications are inadequate. That is the reason why Apache Hadoop was introduced to meet other companies' needs in processing and storing big data. Apache Hadoop's MapReduce and distributed file system (HDFS) [1] have become popular in industrial and academic fields as processing engines and a data storage for big data. Gradually, the term Hadoop has come to mean not only Apache Hadoop's modules but the whole ecosystem including a plethora of software packages that can be run on top of Hadoop, e.g., Apache Hive [2], Apache Tez [3], and Apache Spark [4].

In this paper, we present MR3, which is a new computing engine for Hadoop, and the ASM layer of MR3, which is responsible for processing data. MR3 has two main layers: CORE and ASM. The CORE layer's primary functions are to manage service components and to schedule tasks. The CORE layer communicates directly with Hadoop's services, such as YARN and HDFS. The ASM layer is the computing layer, which means that we can define processing logic based on data type. We focus mainly on processing text data in this paper. We also define our memory structure in ASM layer for better utilizing direct memory and Java Heap Memory (JVM). After that, we compare our engine with others such as MapReduce2, and Apache Tez in the map-only job that search plain-text data sets for lines containing a given pattern.

2. Preliminaries – Hadoop Environment

Hadoop is an open-source framework that enables us to analyze and transform very large data sets using the MapReduce paradigm [5] with the Hadoop distributed file system (HDFS). HDFS is highly fault-tolerant so that even a low-cost cluster can be managed to run HDFS. If application data are large and applications need high throughput access to the data, HDFS is a good choice.

With Hadoop HDFS, we are capable to store very large data and perform parallel computations across computer cluster including a single master and many nodes. From Hadoop 2, YARN [6] is introduced as a resource negotiator and job scheduling to separate these concerns, which eases the workload burden upon MapReduce. With the invention of YARN, we can execute numerous types of applications in the Hadoop cluster. Our engine runs on top of YARN.

3. Method

There are two main layers in our system: the CORE layer, and the ASM layer (Fig. 1). The CORE layer is

responsible for scheduling and managing the job in the whole program. The ASM layer is the main data-computing layer, in which data is processed byte by byte. We implement *TextGrep* program, which is on top of ASM layer and can search plain-text data for lines containing a given pattern.

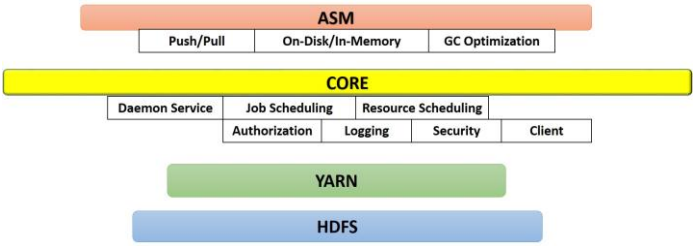


Fig. 1: System overview

3.1 CORE Layer

Our program at the client's side takes several arguments, which depends on specific kinds of application. Then, the program builds a Directed Acyclic Graph (*DAG*) from specified configurations and given arguments. This *DAG* is submitted to Hadoop YARN via its client's interface. YARN launches an Application Master (*DAGAppMaster*) process within one container in the cluster. The *DAGAppMaster* analyzes DAG to get the information about *Vertices* and *Edges*. Next, the *DAGAppMaster* asks YARN to allocate resources to run the tasks. These resources are containers in the cluster, and they each run *ContainerWorker* programs. We can consider *DAGAppMaster* to be a master node that manages all information of the job, and many *ContainerWorkers* to be slave nodes that receive orders and perform tasks.

3.2 ASM Layer

3.2.1 Buffer Hierarchy

We do not use the original Buffer structure from Java, but we implement our Buffer hierarchy (Fig. 2) so that it can store and process data efficiently and effectively. The hierarchy includes *BufferSeqManager*, *BufferSeq*, *DataBuffer*, and *DataView*. *BufferSeq* is a list of *DataBuffer*, which can represent both Direct Memory and Heap Memory in JVM (*DirectDataBuffer* and *HeapDataBuffer*). If a client wants to read or write the data from or to this *DataBuffer*, he needs to use the *DataView*'s interface, which is a wrapper of ByteBuffer in Java and can contain a maximum of 2 GB of data. To manage many *BufferSeqs* and when they should create or release Data Buffer, we have a corresponding *BufferSeqManager*. A client can interact with elements in the hierarchy from our interface, and it begins from the top *BufferSeqMananger* to the bottom *DataView*. For example, a writer client can request *BufferSeq* from *BufferSeqManager*, request to write to *DataBuffer* from *BufferSeq*, request to write to *DataView* from *DataBuffer*, and get *ByteBuffer* from *DataView*.

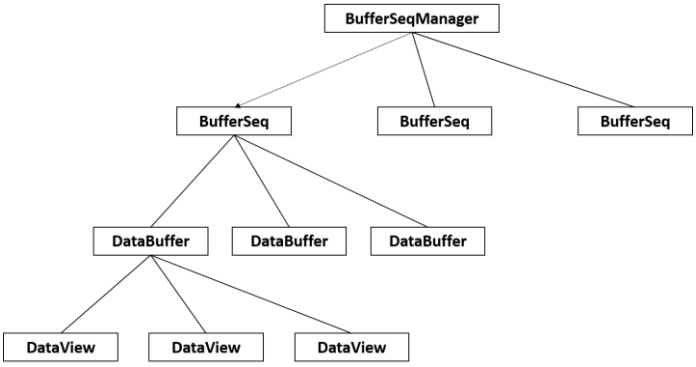


Fig. 2: Buffer hierarchy for storing and processing data.

We design our components in ASM layer so that all of them can process data concurrently by exchanging *BufferSeq*. For example, a component A notifies a component B about an empty *BufferSeq*. While the component A is writing to *DataBuffer* in the *BufferSeq* in one position, the component B can read *DataBuffer* in the *BufferSeq* in another position.

3.2.2 TextGrep Data Flow in ASM

The data flow (Fig. 3) starts with *TextReadWriteInputInitializer*. This class takes input paths as an argument and divides input into many smaller splits. These splits' information is sent to *TextFileLogicalInput*. *TextFileLogicalInput* uses this information to request *TextInputReader* to produce data. At the same time, *TextFileLogicalInput* notifies new BufferSeq will come to *TextGrepProcessor*. *TextInputReader* load data from the cluster storage and produce *DataBuffer* to *BufferSeq* that *TextGrepProcessor* receives. While processing data in *BufferSeq*, *TextGrepProcessor* also notifies new *BufferSeq to TextLogicalOutput*, which in turn asks *TextOutputWriter* to consume *DataBuffer* by spilling data to storage. *TextLogicalOutput* also sends output information to *OutputCommitter*, which is responsible for copying the temporary output to final output directory if nothing goes wrong and stops the whole program.

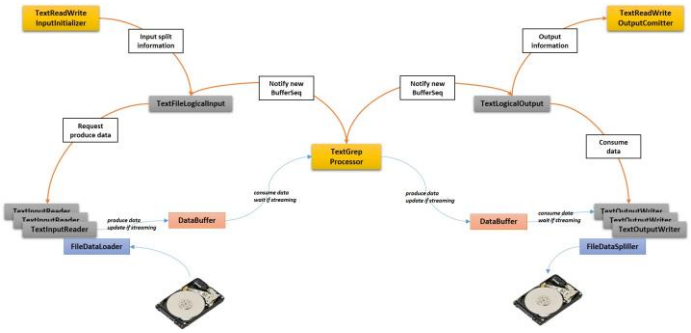


Fig. 3: Data flow in TextGrep program in ASM.

4. Experiments

4.1 Cluster Setup

We use a single big cluster to run our experiments. This cluster has 40 Dell PowerEdge R610 machines. Each machine has two 6-cores CPUs, 94.6 GB RAM, and a total of 3 TB hard drive. All the machines run CentOS 6.6. We install HDP-2.4.2.0 [7] on the cluster. This version of HDP supports Apache Hadoop 2.7 (MapReduce2 and YARN).

4.2 Experiment Setup

We run text grep program with three engines (MapReduce2, Apache Tez, and MR3). Each engine needs to run the program with a variety of input sizes, ranging from 30 GB to 300 GB. Then, we compare their performance based on following criteria: execution time, and task running time. For measuring execution time, we run each engine on input data three times on the cluster, and then take an average of three running times. We then calculate the overall running time of the three engines on all input data. For evaluating task running time, we calculate the mean and standard variance of the running time of each tasks that performed by each engine with varied input sizes. These tasks' information is derived from programs' log files.

We need to make sure three engines get the similar resources from clusters by setting up their configurations as follow.

The first part of this experiment is running the text grep task with Hadoop MapReduce2 engine. Obviously, we need to write a text grep program with MapReduce API. Then, we configure the MapReduce2 engine with *mapred-site.xml* in which we set *mapreduce.framework*.name to yarn, *yarn.app.mapreduce.am.resource.mb* to *5120* (allocate around 5 GB in memory to YARN *AppMaster* Container), and *mapreduce.map.memory.mb* to *5120* (allocate around 5 GB in memory to each YARN Container that runs map jobs). For the rest of configurations, we leave them as default.

The second part of this experiment is running the text grep task with Apache Tez engine. We do not need to write a text grep program with Apache Tez API. We re-use the program from Hadoop MapReduce2 experiments by changing the configuration *mapreduce.framework.name* to *yarn-tez*. Other configurations are specified in the file *tez-site.xml*. We make sure some of necessary configurations to be the same with Hadoop MapReduce2 configurations. For example, we set *tez.am.resource.memory.mb* to *5120* (allocate around 5 GB in memory to YARN *AppMaster* Container), *tez.task.resource.memory.mb* to *5120* (allocate around 5 GB in memory to YARN Container that runs tasks), and *tez.am.container.reuse.enabled* to *false* (we do not set re-use feature at this time). For the rest of configuration of Apache Tez, we leave them as default.

The final part of the measuring-performance experiment is running the text grep task with our engine MR3 and its ASM layer. MR3 also has a configuration file *mr3-site.xml*. We set up this site file so that MR3 has resources from YARN similar to the other two engines. For instance, we set *mr3.am.resource.memory.mb* to *5120* (allocate around 5 GB in memory to YARN *AppMaster* Container), *mr3.map.containergroup.memory.mb* to *5120* (allocate around 5 GB in memory to YARN Container), *mr3.map.task.memory.mb* to *2048* (allocate around 2 GB of *mr3.map.containergroup.memory.mb* to Java Heap memory for running map tasks), and *mr3.map.containergroup.direct.memory.mb* to *2048* (allocate around 2 GB of *mr3.map.containergroup.memory.mb* to direct memory for running jobs in *ContainerWorker*).

5. Results

5.1 Execution Time

We measured the time required to filter text data of three engines on the cluster (Fig. 4). MR3 engine processed data much faster than the other two. For the amount of input data ranging from 30GB to 150GB, MR3 engine ran at least 10 seconds faster than MapReduce2 and Apache Tez. MR3 engine's performance increased significantly when processed the input data ranging from 240GB to 300GB, as MR3 engine's data processing time was less than the other two engines at least 20 seconds.

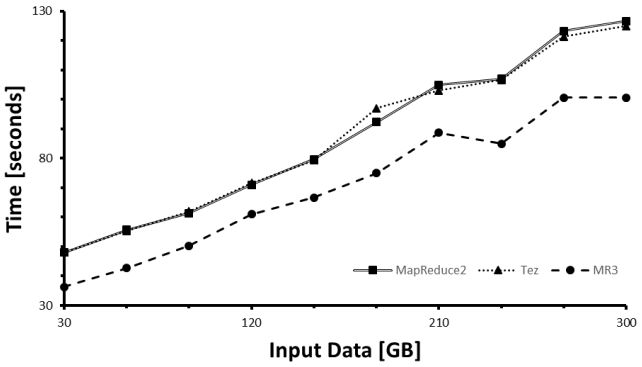


Fig. 4: Time required to process a given input data of three engines (MapReduce2: rectangles; Tez: triangles; MR3: circles) on the cluster.

5.2 Task Running Time

We calculated the mean and the standard variance of the running time of tasks performed by three engines (Table 1).

Table 1. Mean and s.d. of time [seconds] required to process given input data of three engines.

			Engine			
	MapReduce2		Tez		MR3	
Input Size [GB]	mean	s.d.	mean	s.d.	mean	s.d.
30	22.61	6.61	16.19	6.71	12.19	3.96
60	27.53	6.64	20.26	5.95	15.16	4.33
90	27.17	8.03	20.61	5.98	15.07	4.56
120	24.43	8.32	19.06	6.73	14.06	5.12
150	24.67	8.29	19.01	6.57	14.27	4.54
180	25.75	9.49	21.31	8.15	15.06	6.16
210	24.67	8.95	20.76	8.66	14.62	6.41
240	23.08	8.09	19.37	7.21	12.94	4.81
270	24.11	9.68	20.95	8.31	14.22	6.41
300	22.67	8.81	19.64	7.36	13.12	5.14
Overall	24.67	8.29	19.72	7.16	14.07	5.14

The mean of running time of tasks in MR3 was less than that of Apache Tez by around 5 seconds, and MapReduce2 by around 10 seconds.

6. Conclusion and Discussion

Overall, MR3 runs the whole program faster than the other two engines in a map-only job (text grep). The ASM layer enables MR3 to process data faster than MapReduce2 and Tez, as tasks' running time in MR3 is lower than in MapReduce2 and Tez. This is due to the implementation of *BufferSeq* design, which can utilize both direct memory and JVM. An ordinary Java program uses only JVM to process data, but this heap memory is much smaller compared with direct memory, so they usually store some parts of data in hard drive, and load them into JVM to process. In ASM layer, we exploit both direct memory and JVM, so our processor does not need to store data in hard drive but leave it in direct memory, and load them into JVM to process. We assume that the memory of the cluster's machines is enough to such kind of job. In the future, we will extend our performance evaluation for more general kind of jobs such as map-and-reduce jobs.

References

1. K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System," IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1-10, 2010.

2. A. Thusoo et al., "Hive - a petabyte scale data warehouse using Hadoop," IEEE 26th International Conference on Data Engineering (ICDE 2010), pp. 996-1005, 2010.

3. Bikas Saha, Hitesh Shah, Siddharth Seth, Gopal Vijayaraghavan, Arun Murthy, and Carlo Curino. "Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications". In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 1357-1369, 2015.

4. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing". In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI'12), pp. 2-2, 2012.

5. J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters". In Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6, pp. 10-10, 2004.

6. V. Vavilapalli, A. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. "Apache Hadoop YARN: yet another resource negotiator". In Proceedings of the 4th Annual Symposium on Cloud Computing, pp. 5:1-5:16, 2013.

7. Hortonworks Data Platform, https://hortonworks.com/products/data-center/hdp/

OSEK/VDX 기반 내장형 소프트웨어 검증을 위한 환경 설정 추출 및 소스코드 자동 변환 도구

정유희 ○ 최윤자
경북대학교
lauryn94@knu.ac.kr, yuchoi76@knu.ac.kr

Automatic Configuration File Generator and Code Annotation for the Verification of OSEK/VDX-based Embedded Software

Yoohee Chung○ Yunja Choi
Kyungpook National University

요 약

내장형 소프트웨어를 정확하게 검증하기 위해서는 운영체제의 행위를 고려하는 것이 중요하며 이를 효율적으로 반영하기 위해 모델 검증 기법이 사용되어왔다. [2]에서는 기존 모델 검증 기법의 한계인 모델과 응용 프로그램 사이의 언어적 불일치성을 제거하기 위하여 OSEK/VDX[1] 운영체제 모델을 C 언어로 작성하는 방법을 제안하였다. 그러나 특정 운영체제를 고려하여 작성된 응용 프로그램을 운영체제 C 모델과 결합해 검증하기 위해서는 추가적인 환경 설정 코드 및 소스 코드의 수정, 검증 제약조건의 삽입이 필요하다. 본 논문은 이러한 작업을 자동화한 도구를 소개하고, 이를 사용해 자동으로 생성 및 변환된 코드를 사용해 [2]의 C 모델과 응용 프로그램을 함께 검증한 결과를 제시하였다.

1. 서 론

차량 전장용 운영체제 및 이를 기반으로 하는 제어 소프트웨어는 대표적인 안전 중요 시스템이며 차량 제어 소프트웨어의 정확하고도 효율적인 검증을 위해서는 운영 체제의 동작을 고려한 모델 기반 검증 기법을 사용하는 것이 바람직하다.

[2]는 기존 모델 검증 기법의 한계였던 운영체제 모델과 응용 프로그램 소스 코드 간의 언어적 불일치성을 제거하기 위하여 OSEK/VDX[1] 운영체제의 C 모델을 작성하는 방법을 제시하였다. 그 결과 둘 간의 언어적 불일치성 및 이로부터 파생되는 문제점을 제거할 수 있었다.

그러나 특정 운영체제에 종속적인 응용 프로그램을 OSEK/VDX의 추상 모델과 결합하기 위해서는 환경 설정을 반영해줄 수 있는 소스 코드의 추가와 기존 응용 프로그램 코드의 수정이 불가피하다. 또한 일부 제약 속성의 경우 다수의 모니터링 코드와 assert 문을 사용해 검증하게 되는데 이를 수동으로 코드에 삽입하는 것은 많은 노력을 요구하며 이 부분이 자동화될 경우 생산성이 증가한다. 본 논문에서는 환경 설정 추출, 응용 프로그램 코드 수정 및 검증 속성 코드의 추가를 자동화한 도구 Switchy에 대해 소개한다.

2. 연구 배경

2.1. OSEK/VDX 운영체제의 C 언어 모델

OSEK/VDX 응용 프로그램을 검증하기 위해 다양한 모델 기반 검증 기법이 사용되었다. 그러나 형식 언어를 사용하는 운영체제 모델은 일반적으로 C언어로 작성되는 응용 프로그램과 언어적 불일치성을 가진다. 기존의 모델 기반 검증 기법에서는 응용 프로그램을 운영체제의 모델링에 사용된 언어로 모델링 하여 검증하였다. 그러나 이 방법은 모델링 과정에서 응용 프로그램의 원본 소스 코드에 존재하던 포인터의 사용, 자료 구조의 응용, 외부 라이브러리 참조와 같은 코드 단계의 중요한 특징들이 소실된다는 문제를 가진다.

소스 코드의 특징을 최대한 유지하며 모델 기반 검증 기법의 장점을 확보하기 위하여 [2]에서는 응용 프로그램의 구현에 사용되는 C 언어로 운영체제를 모델링하는 방법을 제시하였다. [2]에서 정의된 C 운영체제 모델은 구현 코드보다 높은 추상 레벨을 가지며, 모델과 운영체제를 결합해 시뮬레이션한 결과 OSEK/VDX 기반 실시간 운영체제의 일종인 Erika Enterprise[6] 와 비교하였을 때 같은 행위를 보이는 것을 확인할 수 있었다.

2.2. 응용 프로그램의 수정 및 운영체제 모델과의 결합

[2]에서 제시된 운영체제 C 모델을 사용할 경우 응용 프로그램의 추가적인 추상화가 필요하지는 않지만, 특정 운영체제에 종속된 소스코드를 모델과 결합하기 위해서는 기존 코드에 대한 수정이 불가피하다.

OSEK/VDX 는 멀티 태스킹을 지원하는 운영체제이며 선점형/비선점형/혼합형 스케줄링을 지원한다. 따라서 태스크 문맥 변환이 일어날 특정 조건이 만족되는 경우 기존의 태스크는 실행을 중지하고 스케줄러가 결정한

이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보・컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No. 2017M3C4A7068175).

다음 태스크가 실행되어야 한다. OSEK/VDX 시스템에서 태스크 문맥 변환이 일어나기 위한 조건은 다음과 같다.

1) 시스템을 유휴 상태로 전이시킬 가능성이 있는 TerminateTask, ChainTask, WaitEvent의 호출
2) 선점 가능한 태스크가 현재 실행중인 태스크보다 높은 우선순위를 가졌으면서 태스크 대기열에 추가된 경우
3) 비선점형/혼합형 스케줄링에서 비선점형 태스크가 Schedule API 함수를 호출한 경우

이러한 조건들은 OSEK/VDX의 API 함수가 호출될 때 만족되는 것들이므로 API 함수가 호출될 때 마다 스케줄러가 이러한 조건을 확인하여 문맥 변환 여부를 결정한다. [2]에서는 특정 라이브러리나 아키텍처에 종속적이지 않은 방법을 사용해 문맥 변환을 모델링하기 위하여 C 언어의 goto문과 while 반복문을 사용하였다. 코드 1은 [2]에서 이 둘을 사용해 문맥 변환을 어떻게 표현했는가를 보여주는 예시 코드이다. 왼쪽의 C 코드는 원래 API 함수 호출만 있던 응용 프로그램 코드에 goto 문과 while 문을 사용한 코드를 추가한 것이다. 오른쪽의 코드는 두 태스크의 환경 설정을 보여준다. 앞서 언급된 문맥 변환 조건이 만족되는 경우 API 함수는 1을, 그렇지 않을 경우 0을 반환하며 이 값에 따라 태스크는 실행을 계속하거나 중지한다. 이 때 태스크의 프로그램 카운터를 의미하는 보조 변수의 값이 증가하며 이 값은 나중에 태스크의 실행이 재개될 경우 태스크를 다시 시작할 위치로 즉각적으로 점프 연산을 수행하는 데 사용된다.

```
1: /* jump(x) is a macro which means
goto Label_x. */
2: void task_t1 (){
3:   int ret ;
4:   jump(pc[1]);
5:   Label1 :
6:   ret = ActivateTask(2);
7:   pc[1]++;
8:   if( ret ) return ;
9:   Label2 :
10:  ret = ChainTask(1);
11:  pc [1] = 1;
12   if( ret ) return;
13: }
14: void task_t2 (){
15:   int ret ;
16:   jump (pc[2]);
17:   Label1 :
18:   ret = TerminateTask();
19:   pc [2] = 1;
20:   if( ret ) return;
21: }
22: void app (){
23:   while ( run_tid ){
24:     if( run_tid == 1)
25:       task_t1 ();
26:     else if( run_tid == 2)
27:       task_t2();
28:   }
29: }
```

```
TASK t1 {
    PRIORITY = 2;
    AUTOSTART = TRUE;
    SCHEDULE = FULL;
};

TASK t2 {
    PRIORITY = 1;
    AUTOSTART = FALSE;
    SCHEDULE = FULL;
};
```

코드 1. 태스크 문맥 변환의 명시적 표현

3. 도구의 설계 및 구현

2장에서 설명하였듯, 특정 운영체제에 종속적인 응용 프로그램을 [2]에서 제시된 운영체제 모델과 결합하기 위해서는 코드의 수정이 요구된다. 본 장에서는 이러한절차를 자동화하기 위한 도구의 설계와 구현에 대해 소개한다.

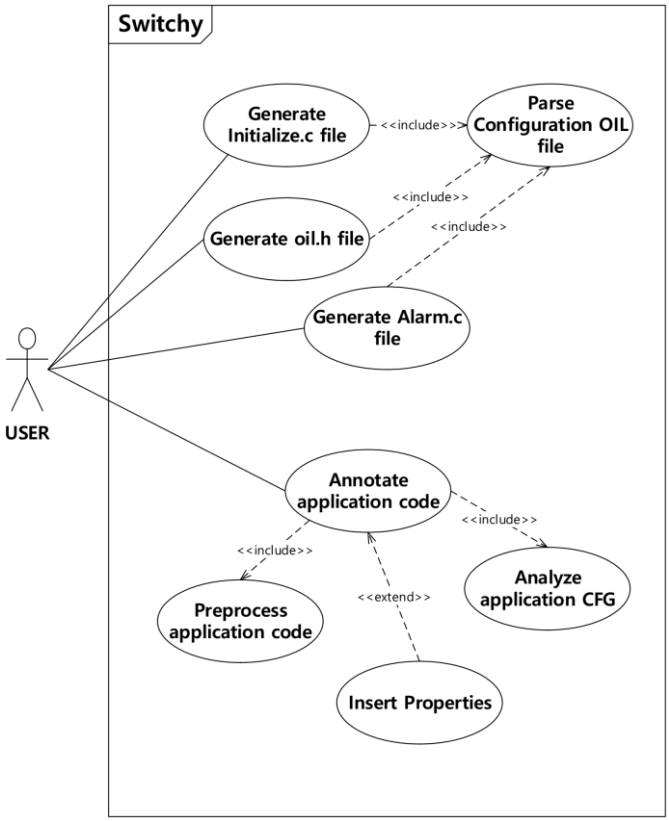


그림 1. 도구의 기능적 요구사항

3.1. 기능적 요구사항

3.1.1. 시스템 초기화를 위한 환경설정 파일 생성

OSEK/VDX 기반 운영체제와 응용 프로그램을 빌드할 때는 환경 설정 파일이 함께 입력 파일로 주어진다. 마찬가지로 [2]에서 작성된 OSEK/VDX 운영체제 C 모델과 응용 프로그램을 함께 모델 체킹 하거나 혹은 시뮬레이션을 위하여 컴파일 하는 경우 환경 설정 파일에 담긴 OSEK/VDX 객체들의 정적인 속성들을 참조할 수 있게끔 하는 작업이 필요하다. 응용 프로그램의 설정을 담고 있는 OIL (OSEK Implementation Language)[3] 파일의 파싱 정보로부터 태스크, 리소스, 이벤트와 같은 시스템 내 객체들의 정보를 받아올 수 있으며 이 정보를 사용해 컴파일 시 사용할 헤더 파일(oil.h)과 객체 속성 값 초기화를 위한 코드(initialize.c) 파일을 생성하는 것이 가능하다. 일부 OSEK/VDX 프로그램의 경우 반복되는 행위를 처리하기 위하여 알람을 사용한다. 시스템에서 사용 가능한 알람의 정보 역시 OIL 환경 설정 파일 내에 정의되어 있으므로 파싱 된 정보를 통하여 알람 모델(alarm.c)을

자동으로 생성할 수 있다.

3.1.2. CFG 분석을 통한 응용 프로그램 코드 수정

원본 응용 프로그램 소스 코드에 태스크의 문맥 변환을 명시적으로 모델링 하기 위해서는 먼저 전체 프로그램 소스 코드 내에서 문맥 변환의 기준점이 되는 특정 구문을 찾아낼 필요가 있다.

이를 위해 필요한 것이 제어 흐름 그래프(Control Flow Graph, CFG) 분석이다. [4]에서는 소프트웨어 검증 효율 향상을 위하여 코드 자르기 기법을 제안하였으며 이 과정에서 함수 단위로 프로그램을 나누어 각 함수의 제어 흐름 그래프를 생성하였다. 본 연구에서는 [4]에서 사용된 함수 별로 CFG 를 생성해 이를 탐색하여 OSEK/VDX API 함수를 호출하는 구문을 찾아내었다. 태스크 내에서 API 함수가 호출되면 그 직후 태스크의 문맥 변환이 일어날 수 있다고 판단되므로 프로그램 카운터의 값이 갱신된다. 문맥 변환이 실제로 일어나는 경우 스케줄러에 의해 API 함수는 1을 반환하게 되므로 해당 반환 값에 따라 현재 태스크 함수는 종료를 하게 된다.

3.1.3. API 호출 제약조건 검증을 위한 모니터링 코드와 assert 문의 삽입

OSEK/VDX API는 부적절하게 호출될 경우 시스템 실패를 야기할 수 있다. 예를 들어, 한 태스크가 WaitEvent API 함수를 호출하여 *waiting* 상태로 빠졌으나 다른 태스크가 SetEvent API 함수를 호출하지 않는다면 해당 태스크는 무한히 *waiting* 상태에 머무를 수 밖에 없다. 이러한 경우를 방지하기 위하여 [5]에서는 OSEK/VDX에서의 API 함수 호출에 관한 제약 조건을 명시하고 있으며 이러한 제약조건을 표현하는 OSEK_CSL(OSEK Constraint Specification Language) 를 제안하였다.

본 도구는 OSEK_CSL을 입력으로 받아 해당 제약조건을 검사하는 데 사용될 모니터링 코드 및 assert 문을 자동적으로 코드에 추가해준다. 모니터링 코드는 API 함수가 호출되면 같이 실행되면서 현재 시스템의 상태를 파악하고 이를 [2]에서 제공하는 상태 검사 오토마타로 넘겨준다. API 호출 제약조건이 만족된다면 마지막 모니터링 코드가 호출된 이후 오토마타의 상태는 최종적으로 안전 상태에 머무르고 있을 것이다. 그렇지 않은 경우 API 호출 제약조건이 위배된 것으로 간주한다.

3.2. 도구의 구조적 디자인

3.2.1. 환경 설정 파일 생성 모듈

그림 2는 OIL 파일로부터 환경 설정을 위한 initialize.c, oil.h, alarm.c 파일을 자동적으로 생성하는 모듈의 클래스 다이어그램이다. OilInitCodeGenerator 클래스와 OilHeaderFileGenerator 클래스, AlarmFileCodeGenerator 클래스는 모두 OIL 파일을

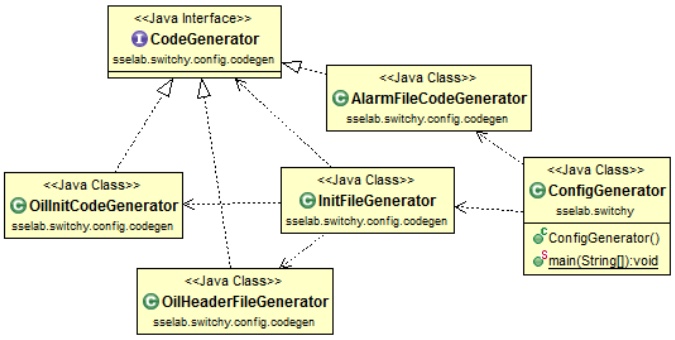


그림 1. 환경 설정 파일 생성 모듈의 구조적 디자인

파싱해 코드를 생성하고 이를 ConfigGenerator 클래스로 돌려준다. 각 클래스가 생성한 코드는 필요한 파일에 출력된다.

3.2.2. 응용 프로그램 코드 수정 모듈

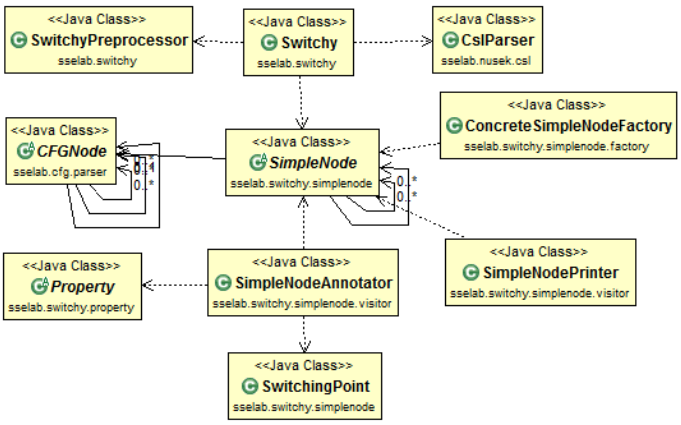


그림 2. 응용 프로그램 수정 모듈의 구조적 디자인

그림 3은 태스크를 포함하고 있는 응용 프로그램 소스 코드를 [2]의 운영체제 C 모듈과 함께 사용할 수 있는 형태로 변환하는 부분을 표현한 클래스 다이어그램이다. 프로그램에 문맥 변환 포인트와 모니터링 코드, assert 문을 삽입하기에 앞서 Switchy 는 일차적으로 [4]의 코드 분석기가 CFG를 생성할 수 있도록 SwitchyPreprocessor 클래스의 기능을 활용하여 전처리를 실시한다. 생성된 CFG는 보다 간략화 된 인터페이스를 제공하고 유연한 수정이 가능하도록 SimpleNode 클래스로 감싸져 다루어진다. SimpleNode 객체들로 구성된 CFG는 SimpleNodeAnnotator가 일차적으로 탐색해 문맥 변환이 일어날 지점, 프로그램 카운터가 관리되는 지점, 검증을 위한 모니터링 코드 및 assert 문이 삽입될 지점을 찾아 SwitchingPoint 와 Property를 추가한다. CFG에 대한 수정이 완료된 후 CFG는 SimpleNodePrinter 클래스에 의해 다시 한 번 탐색되면서 수정된 소스 코드를 생성해 출력한다.

4. 실험

본 장에서는 3의 도구를 활용하여 응용 프로그램 코드를 변환한 후 이를 [2]의 운영체제 모델과 결합하여 기능적 제약조건, 코드 안전 제약조건 및 API 호출 제약 조건을 검증했다. 실험에는 OSEK/VDX를 기반으로 한 차량 전장용 운영체제 Trampoline 2.0[7]과 실시간 운영체제인 Erika Enterprise[6]의 예제 코드가 사용되었다. 검증 환경은 16 Intel(R) Xeon(R) CPU E5-1680 v4 @ 3.4GHz CPU, 및 메모리 128 GB이며 모델 검증 도구로는 C/C++ 대상 바운디드 모델 검증 도구인 CBMC 5.7[8] 이 사용되었다.

표 1. 응용 프로그램의 검증 결과 및 비용

Code	Property	Time(s)	Mem(MB)	Result
EE07.c	functional	40' 04"	8234	Fail
	InPairs(AT, TT)	53' 42"	8481	Fail
	Code-safety	52' 45"	3713	Succ
EE08.c	Functional	10' 43"	1034	Succ
	InPairs(GR, RR) InPairs(AT, TT)	23' 57"	3417	Succ
	Code-safety	37' 15"	3472	Succ
TP01.c	Functional	< 1"	8.4	Succ
	MustEndWith(TT, CT) InPairs(AT, TT)	< 1"	8.6	Succ
	Code-safety	< 1"	13.6	Succ

기능적 제약조건의 경우 Erika Enterprise의 예제 코드(EE07.c, EE08.c)는 내부에 실행 순서를 검사하는 EE_assert 함수와 검사 결과를 저장하는 EE_assertions 행렬이 있으므로 프로그램 종료 시 해당 행렬의 모든 원소가 1인지 확인하는 assert 문을 삽입해 검증하였다. Trampoline 의 예제 코드(TP01.c)에는 기능적 요구사항이 주어지지 않았으므로 API 함수가 호출되면 객체의 상태가 의도한 대로 변경되는지를 확인하는 assert 문을 삽입해 검증하였다.

API 호출 제약조건의 경우 소스 코드의 내용에 따라, InPairs(ActivateTask, TerminateTask), InPairs(GetResource, ReleaseResource), MustEndWith(TerminateTask, ChainTask), InPairs(ActivateTask, TerminateTask)를 검증했다.

코드 안전 제약조건은 코드 내에서 빈번하게 사용되는 포인터, 변수, 행렬의 존재를 고려해 부적절한 포인터 변수의 참조, 변수 오버플로우, 행렬 바운드 외부 침범 여부를 검증했다.

모든 검증은 고정된 바운드 값(반복문 바운드 10) 내에서 수행되었다. 그 결과 표 1과 같은 검증 결과를 얻을 수 있었다. EE07.c의 경우 운영체제 C 모델과의 시뮬레이션에서는 제약 조건 위배가 발견되지 않았지만 모델 검증 결과 잠재적으로 기능적 제약조건 및 InPairs(ActivateTask, TerminateTask)가 위배될 수 있다는 것을 확인할 수 있었다. 실제로 EE07.c의 인터럽트의 주파수를 증가시킨 결과 해당 제약 조건이 위배되는 것이 발견되었다. 그 외의 검증은 모두 주어진 바운드 내에서 제약 조건이 만족됨을 보였다.

5. 결론

본 논문에서는 [2]에서 제시된 OSEK/VDX 운영체제 C 모델을 사용해 응용 프로그램을 검증할 수 있도록 환경 설정을 추출해 초기화 파일을 생성하고 기존 응용 프로그램에 문맥 변환을 위한 추가적 코드와 검증을 위한 모니터링 코드 및 제약조건을 자동으로 삽입하는 도구를 소개하였다. 또한 이 도구를 사용해 코드를 추가 생성 및 변환하여 응용 프로그램을 바운디드 모델 검증 도구 CBMC를 사용해 검증하였다. 그 결과 일부 응용 프로그램의 기능적/API 호출/코드 안전 제약 조건 위배 여부를 확인할 수 있었다.

그러나 현재 도구는 API 호출 제약 조건의 경우 가장 중요한 것으로 간주되는 6가지 제약 조건만을 검증할 수 있다. [5]에서 제시되는 API 호출 제약 조건은 훨씬 다양하므로 이 부분에 대한 추가적인 고려가 필요하다. 또한 더욱 다양한 운영체제 및 하드웨어 환경을 대상으로 하는 응용 프로그램을 처리할 수 있도록 추가적인 연구가 요구된다.

참 고 문 헌

[1] "OSEK/VDX operating system specification," 2005.
[2] Y.Chung, D.Kim, Y.Choi, "Modeling of OSEK/VDX OS Requirements in C", Software Engineering Conference (APSEC), 2017 24th Asia-Pacific. IEEE, 2017. (To appear in)
[3] A. Zahir, "OIL-OSEK implementation language", 1998.
[4]김동우, 박민규, 최윤자, "효과적인 소프트웨어 검증을 위한 코드 자르기 도구의 개발", 한국정보처리학회 춘계학술대회 논문집, pp. 586-589, 2014.
[5] Y. Choi. Constraint specification and test generation for OSEK/VDX-based operating systems. In Software Engineering and Formal Methods, Springer, pp. 305-319, 2013.
[6] "Erika Enterprise", http://erika.tuxfamily.org
[7]"Trampoline open source RTOS project," http://trampoline.rtssoftware.org
[8] D. Kroening and M. Tautschnig, "CBMC-C bounded model checker," in International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, pp. 389-391, 2014.

개발자의 Bad Fix Rate를 고려한 효과적인 체인지 레벨 결함 예측 기법 연구

임형선 ○ 백종문

한국과학기술원 전산학부

loginluv@kaist.ac.kr, jbaik@kaist.ac.kr

An Effective Change Level Software Defect Prediction Considering Developer's Bad Fix Rate

Hyungsun Lim○ Jongmoon Baik

KAIST, School of Computing

291, Daehak-ro, Yuseong-gu, Daejeon, Republic of Korea

요 약

개발자의 Bad fix rate는 개발자의 프로젝트 경험 및 지식을 나타내는 중요한 지표 중 하나다. 본 논문에서는 코드 수정 단위 결함 예측 모델에서 개발자의 Bad fix rate를 정량화하는 방법을 제안하고, 5개의 오픈소스를 대상으로 제안된 메트릭의 성능을 평가한다.

1. 서 론

최근 체인지 기반 결함 예측 연구가 활발해지고 있다. 이 기법은 개발자가 수정한 코드 수정 단위(체인지)에 결함이 있는지를 예측하는 기법이다. 이 기법에서는 체인지의 결함 가능성을 나타내는 다양한 메트릭을 사용한다. Kamei는 과거 결함 예측 분야에서 활발하게 사용되던 메트릭 중 체인지 기반에서 적용할 수 있는 14개의 메트릭을 정리하였다[1]. 하지만 해당 메트릭 집합에서는 개발자의 능력치를 반영하는 메트릭의 숫자가 매우 적다. 뿐만 아니라 개발자 관련 메트릭 간의 연관성이 높아 모델에 학습시킬 수 없는 메트릭이 존재한다. 그러므로 정교하게 개발자의 능력치를 반영하면서 기존 메트릭과 연관성이 적은 메트릭이 필요하다. 그중 개발자의 능력치와 잘못된 픽스(Bad fix) 와의 연관성을 나타낸 연구가 있다[2]. Bad fix란 결함을 수정하는 활동을 통해 새로운 결함이 삽입되는 것을 뜻한다. 해당 논문에서는 프로젝트 경험이 적은 개발자가 bad fix를 유발할 가능성이 높다는 것을 실험적으로 보여주었다. 즉 bad fix가 높은 개발자일수록 지식 및 경험이 낮다는 것을 나타낸다. 이러한 과거 연구를 바탕으로 본 연구는 개발자의 bad fix rate를 체인지 레벨 결함 예측 환경에서 정량화하는 방법을 제안하고 제안된 메트릭의 효과성을 통계적으로 검증한다.

2. 관련 연구

Kamei[1]는 체인지 단위에서 사용 가능한 14개의 메트릭 집합을 표 1과 같이 제안하였다. 6개의 오픈소스 및 5개의 상업용 프로젝트를 대상으로 해당 메트릭 들을 추출하여 로지스틱 회귀 예측 모델을 생성하여 모델을 평가하였다. Yin[2]은 상업용 및 오픈소스 프로젝트를 대상으로 버그를 유발하는 픽스의 특징에 대한 연구를 진행하였다. 해당 연구에서는 프로젝트에 대한 개발자의 지식과 경험이 bad fix와 관련이 있다는 것을 보여주었다.

표 1. 체인지 기반 메트릭[1]

메트릭	설명	메트릭	설명
NS	수정된 시스템의 수	FIX	픽스인지 아닌지 여부
ND	수정된 폴더의 수	NDEV	파일을 수정한 개발자의 수
NF	수정된 파일의 수	AGE	수정 시간 간격
Entropy	수정된 코드의 분포	NUC	수정한 파일 이전 체인지 수
LA	추가된 코드 라인 수	EXP	개발자의 경험
LD	삭제된 코드 라인 수	REXP	최근 개발자의 경험
LT	수정 전 파일의 코드 수	SEXP	서브 시스템에 대한 개발자의 경험

3. 개발자의 Bad fix rate를 고려한 체인지 레벨 결함 예측 방법

3.1 Bad Fix Rate(BFR) 메트릭 정의

개발자의 Bad fix rate(BFR)는 개발자가 수행한 픽스 체인지 대비 결함을 유발한 픽스의 비율로 정의할 수 있다.

$$\text{BFR of developer } x = \text{BFR}(x, change\#) = \frac{The\ number\ of\ x's\ buggy\ fix\ change}{The\ number\ of\ x's\ fix\ change}$$

3.2 체인지 단위 결함 예측 모델 생성 절차

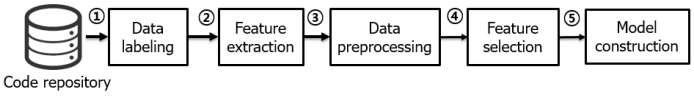


그림 1. 결함 예측 모델 생성 과정

데이터 라벨링 단계에서는 SZZ 알고리즘[3]을 사용하여 코드 저장소에 있는 체인지들을 버그가 있는 체인지와 없는 체인지로 분류한다. 그런 다음, 체인지로부터 특징을 추출하는 작업을 한다. 본 논문에서는 Kamei의 연구에서 사용한

p-value<0.05
>Medium
>Small
No evidence

	Tensorflow				Docker				Cygwin				Django				React			
	P	R	F1	AUC	P	R	F1	AUC	P	R	F1	AUC	P	R	F1	AUC	P	R	F1	AUC
New	33.07	75.06	45.86	77.80	32.94	87.38	47.74	86.25	16.01	68.71	25.92	77.22	64.43	70.43	67.26	77.63	44.01	83.48	57.57	85.38
Kamei	32.43	73.09	44.85	76.90	31.77	87.76	46.62	85.40	16.53	68.54	26.58	77.49	62.38	65.48	63.82	73.57	42.00	82.85	55.62	83.72
p-value	0.26	0.01	0.10	0.10	0.00	0.64	0.00	0.01	0.20	0.91	0.27	0.67	0.00	0.00	0.00	0.00	0.00	0.32	0.00	0.00
Cliff delta	0.13	0.37	0.18	0.21	0.42	-0.04	0.41	0.39	-0.13	0.00	-0.13	-0.04	0.30	0.75	0.61	0.64	0.37	0.11	0.39	0.44

R : Recall P : Precision

그림 2. Kamei 모델과 BFR을 추가적으로 고려한 모델의 성능 비교

14개의 특징[1]과 이 논문에서 제안한 BFR까지 총 15개의 특징을 추출하였다. 추출된 메트릭 들은 데이터의 분포가 한쪽으로 편향되어 있어 성능 저하의 문제가 발생할 수 있다. 이러한 문제를 해결하기 위해 로그 변환 기법을 활용하여 데이터 편향 문제를 해결하였다. 또한 클린 데이터셋과 버그 데이터셋 간의 불균형 문제를 해결하기 위해 랜덤 언더 샘플링 방식을 통하여 데이터셋의 수를 맞춰주었다. 특징 간의 상관관계가 높은 특징을 학습시키거나, 다중공선성이 높은 특징을 모델에 학습시키면 예측 모델에 오버피팅문제를 일으킬 수 있으므로 스피어만 상관계수 값이 0.8 이상인 경우 둘 중 하나의 특징을 제거하였고 분산팽창지수 값이 10 이상인 특징을 찾아내어 다중공선성이 큰 특징을 제거하였다. 이 과정에서 BFR 메트릭은 기존 메트릭과의 연관성이 크지 않아 함께 쓸 수 있다는 것을 확인하였다. 결함 예측 모델로는 로지스틱 회귀 모델을 사용하였다.

4. 실험 및 평가

4.1 실험 목적

새로운 메트릭의 성능을 확인하기 위해, 14개의 특징을 사용한 Kamei의 모델과 개발자의 BFR을 추가적으로 고려한 모델을 비교하였다.

4.2 데이터셋

본 논문에서는 5개의 오픈소스 데이터셋(Tensorflow, Docker, Cygwin, Django, React)을 사용하였다. 실험 결과의 일반화를 위해서 프로젝트 언어, 규모, 픽스의 비율이 서로 다른 데이터셋을 사용하였다

4.3 성능 평가 지표 및 평가 방법

본 논문에서는 정밀도(Precision), 재현도(Recall), F1 점수, AUC(Area Under Curve)를 통해 모델의 성능을 평가하였다. 성능 평가방법은 10겹 교차 검증(10 fold cross validation)방법을 5번 수행하였다. 성능향상의 통계적 유의성을 확인하기 위하여, Wilcoxon signed-rank test 및 Cliff delta effect size 분석을 하였다.

4.4 실험 결과

결과적으로 5개의 데이터 셋 중 4개의 데이터 셋에서는 적어도 하나의 성능 평가 지표에서 통계적으로 유의미한 성능 향상이 일어난 것을 알 수 있다. 통계적 유의성이 없다고 나타난 성능지표에서도 기존 모델과 비슷한 성능을 보여주고 있다. 즉 BFR을 추가적으로 고려한 결함 예측 모델이 기존 모델보다 뛰어난 성능을 보여준다는 것을 알 수 있다. 반면 Cygwin 데이터셋에서는 성능향상이 나타나지 않았는데, 해당 데이터셋에서는 픽스 체인지의 비율이 5.41%로 매우 적어 BFR이 적절히 반영되지 않은 것으로 보인다.

5. 결론

본 논문에서는 개발자의 BFR를 체인지 단위에서 정량화하는 방법을 제안하였다. 결과적으로 개발자의 BFR를 고려하면 기존 방법보다 더 좋은 성능을 보여준다는 것을 확인하였다. 또한 기존 메트릭과의 연관성 분석에서 BFR 메트릭은 기존 메트릭과의 연관성이 크지 않아 함께 쓸 수 있다는 것을 확인하였다. 본 논문의 추후 연구 방향으로는 시간의 흐름에 따라 모델을 새롭게 학습하며 평가하는 Online machine learning 기법을 통해 좀 더 정교한 모델 성능 평가를 해보는 것이다. 또한 SZZ 알고리즘은 필연적으로 노이즈를 포함할 수밖에 없는데, 노이즈 제거 알고리즘[4] 및 개선된 SZZ 알고리즘을 통해 얻어진 데이터셋을 통해서도 실험을 진행할 수 있을 것이다.

Acknowledgement

이 논문은 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2016R1D1A1A09917660).

이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No. 2017M3C4A7066212).

참고문헌

[1] Y. Kamei *et al.*, “A large-scale empirical study of just-in-time quality assurance,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 757–773, 2013.

[2] Z. Yin, D. Yuan, Y. Zhou, S. Pasupathy, and L. Bairavasundaram, “How do fixes become bugs?,” in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering - SIGSOFT/FSE '11*, 2011, p. 26.

[3] J. Śliwerski, T. Zimmermann, A. Zeller, J. Śliwerski, T. Zimmermann, and A. Zeller, “When do changes induce fixes?,” in *Proceedings of the 2005 international workshop on Mining software repositories - MSR '05*, 2005, vol. 30, no. 4, pp. 1–5.

[4] S. Kim, H. Zhang, R. Wu, and L. Gong, “Dealing with noise in defect prediction,” in *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, 2011, p. 481.

SVM: 소프트웨어 제품 라인 개발을 위한 가변성 모델링 도구

한영훈, 강성원, 박연하, 박병훈 ○

한국과학기술원, 티쓰리큐 ○

{younghun.han, sungwon.kang, yyuunn0044}@kaist.ac.kr, warmpark@t3q.co.kr ○

SVM: A Variability Modeling Tool for Software Product Line

Younghun Han, Sungwon Kang, Yeonha Park, Byung-Hoon Park ○

KAIST, T3Q○

요 약

소프트웨어 제품 라인 개발에서 제품군의 공통성과 가변성을 분석하기 위해 사용 되는 대표적인 방법론이 피처 모델링 방법이다. 피처 모델은 처음 등장한 이후 많은 연구자들에 의해 진화 개선 되었음에도 불구하고, 여전히 과학적 원리의 미흡, 추적성 문제, 확장성 문제 등의 문제들이 있다. 이 논문에서는 이러한 문제들을 해결하기 위해 새로운 가변성 모델링 방법인 CFM에 기반한 도구 SVM을 개발한다. 또한 SVM을 사용하여 CFM을 전통적인 피처 모델들과 비교한다.

1. 서론

서로 유사한 일군의 소프트웨어 제품을 개발하는 능력은, 제품 개발의 대량 맞춤 생산(Mass Customization)을 가능하게 하기 때문에 소프트웨어 개발 회사의 글로벌 경쟁력에 필수적인 요소이다. 소프트웨어 제품 라인 개발은 소프트웨어 개발의 전 과정에 걸친 재사용을 통하여 유사한 제품들의 집합, 즉 제품군(群)을 생산함으로써, 개발 비용을 획기적으로 줄이고 제품의 생산성 및 품질을 극도로 높일 수 있는 소프트웨어 개발 패러다임이다[1]. 소프트웨어 제품 라인 개발은 1)제품군의 공통성과 가변성을 분석하여 제품군을 위한 플랫폼을 구축하는 단계와 2)플랫폼으로부터 제품을 도출하는 단계로 구성된다. 따라서, 제품군의 공통성과 가변성을 분석하는 방법은 소프트웨어 제품 라인 개발의 성공 과도 직결되는 매우 중요한 기술이다.

제품군의 공통성과 가변성을 분석하기 위한 대표적인 방법론이 피처 모델링 방법이다. 피처 모델링 방법은 1990년 FODA 피처 모델[2]을 시작으로 1998년 FORM 피처 모델[3]을 거쳐 2009년에는 E-FORM(Extended-FORM) 피처 모델[4]로 발전하였다. 이외에도 많은 연구자들이 1)피처 모델의 표현력을 확장하기 위한 연구들, 2)피처 모델을 분석하는 연구들 및 3)피처 모델을 소프트웨어 개발에 활용하기 위한 연구들을 포함하여 다양한 주제로 활발하게 연구를 진행하고 있다. 피처 모델 기반의 대표적인 도구로는 Pure::variant[5], FeatureIDE[6] 및 SPLOT[7]가 있다.

이와 같이 지난 25여년간 피처 모델링 방법은 많은 연구자들에 의해 발전 되었음에도 불구하고 여전히 해결해야 하는 문제점들이 많이 남아 있다. 이 논문의 저자들은 이전 연구[8]에서 전통적인 피처 모델들을 분석하여, a)과학적 원리의 미흡, b)추적성 문제, c)확장성 문제, d)복잡성 문제, e)가변성 해소 정의 메커니즘의 부재 및 f)피처 그룹핑 메커니즘의 부재라는 문제점을 지적하였다.

본 논문에서는 위에 제시한 문제를 해결하기 위해 가변성 모델링 도구 SVM(Systematic Variability Modeler)를 개발한다. SVM은 [9]에서 제시한 CFM(Compositional Feature Model) 방법론의 일부인 CFM 기본모델 구축 방법 및 CFM 뷰의 구축 방법을 이용한다. CFM 기본모델은 가변성관계, 포함관계, 제약사항으로 구성되어 있으며, 사용자의 요구에 따라 구현 관계, 추적성 관계도 포함 될 수 있다. CFM 방법의 타당성을 검증하기 위해 SVM을 사용하여 전통적인 피처 모델링 방법들과 비교하였다.

2. 전통적인 가변성 모델링 방법의 문제점을 해결하는 SVM의 방법들

이 절에서는 전통적인 피처 모델링 방법의 문제점을 해결하기 위해 SVM에서 사용되는 네 가지 모델링

방법을 각각 2.1절~2.4절에서 소개한다. 이 방법들 중 2.2절과 2.3절은 논문 [9]에서 제시한 CFM 기본모델 구축 방법 및 CFM 뷰의 구축 방법을 활용한다

2.1 피처 모델과 가변성 모델간의 명확한 구별 및 정의

전통적인 피처 모델은 활용방안에 대해서 명확하게 정의되지 않았다. 전통적인 피처 모델은 1)요구사항 도출 전 제품들의 공통성과 가변성을 분석하기 위한 방법으로 제안 된 것인지, 2)소프트웨어 개발 전 주기에 활용하기 위한 방법으로 제안 된 것인지 명확하게 구분하지 않는다. FODA 피처 모델이 도메인 분석을 위해 제안 된 방법임을 비추어 볼 때, 피처 모델 고유의 목적은 전자의 해석에 가깝다. 하지만, 과거 연구진들은 피처 모델의 표현력을 무분별하게 확장하여 소프트웨어 개발 전 주기에 활용하고 있다. 이 경우, 1)피처의 개념이 사용자 관점에서의 공통성과 가변성을 분석하는 단위를 벗어나 요구사항, 컴포넌트, 코드 단위를 지칭하는 개념으로도 활용 될 수 있기 때문에, 피처가 가지는 고유의 특성이 모호해 지는 문제점이 있다. 또한, 2)피처 모델의 기술 기법을 확장하여 소프트웨어 개발 전 과정에 활용하면 특정 개발 단계에서 불필요한 모델링 기능들도 함께 포함되기 때문에, 개발자가 효율적이고 체계적으로 모델링하기 어려워진다.

따라서, SVM에서는 각 개발 단계에서의 이해관계자, 개발 산출물, 가변성 해소 시점 등을 고려하여, 해당 개발 단계에 적합한 가변성 모델들을 제시하고 이들 간의 추적성 관계를 지원함으로써 위 문제들을 해결한다. 전통적인 피처 모델이 소프트웨어 개발 전 과정에 적용할 수 있는 가변성 모델 이기 보다는 사용자 관점에서 제품군의 공통성과 가변성을 분석하기 위해 활용할 수 있는 가변성 모델로 한정한다. 따라서, 전통적인 피처 모델을 SVM에서는 피처 가변성 모델(FVM, Feature Variability Model)이라고 정의한다. 요구공학 단계에서는 요구공학 가변성 모델(RVM, Requirement Variability Model), 설계 단계에서는 설계 가변성 모델(DVM, Design Variability Model), 구현단계에서는 구현 가변성 모델(ImVM, Implementation Variability Model) 및 시험단계에서는 시험 가변성 모델(TVM, Test Variability Model)을 정의한다. 설계 단계에서는 관점에 따라서 다양한 뷰가 도출될 수 있기 때문에, 뷰에도 활용될 수 있는 가변성 모델을 별도로 정의할 수 있다. 각 개발 단계에서의 가변성 모델은 타 개발 단계의 가변성 모델과 추적성 관계를 지원한다. 또한, 해당 개발 단계에서의 가변성 모델과 개발 산출물과의 추적성 관계도 지원한다.

2.2 가변성의 명시적 표현

전통적인 피처 모델에서는 피처와 가변성, 두 개념을 구분하여 사용하지 않는다. 피처의 역할과 가변성을 역할은 서로 다르기 때문에 이들을 서로 분리하지 않고 사용한다면 모델의 명확한 의미 파악이 어렵게 되고, 결과적으로 사용자의 혼란을 초래하는 문제점이 있다. SVM에서는 피처와 가변성의 역할을 명확하게 정의하고, 서로 분리하여 위 문제점을 해결한다. 가변성은 가변점과 가변값으로 구성된다. 가변점은 제품군의 제품들 간의 공통적인 부분을 추상화한 요소로, 제품개발 시 구체화하여 제품의 부분으로 개발하기 위해 도입된 개념이며, 가변값은 제품 수준의 산출물 요소로, 가변점이 가질 수 있는 가능한 구체적인 값을 의미한다.

2.3 관심의 분리(Separation of Concerns) 원리의 적용

소프트웨어 개발은 다양한 이해관계자들이 존재 한다. 제품군의 공통성/가변성을 위한 피처 모델링 단계도 다양한 이해관계자들이 존재할 수 있다. 전통적인 피처 모델은 하나의 모델 안에 피처들 간의 다양한 관계가 함께 포함되어 있다. 예를 들어, FORM 피처 모델의 경우에는 포함 관계, 일반화/특수화(generalization/specialization) 관계, 가변성 관계, 제약 관계 등이 포함되어 있다. 포함 관계는 규모와 확장성 관점의 관심사항의 표현이고, 일반화/특수화 관계는 재사용관점의 관심사항이다. 하나의 모델 안에 다양한 관심들을 표현하면 모델이 복잡해 져서 직관적인 이해라는 피처 모델 고유의 장점이 사라지게 된다. 피처 모델의 복잡성 및 확장성 측면을 고려하여 다양한 관심들을 분리해서 모델링 하는 방법으로 CFM[9]를 사용함으로써 SVM은 이 문제를 해결한다. CFM은 관심 단위로 분리 모델링을 수행하고, 사용자가 관심 있는 단위들을 통합하여 하나의 뷰로 보여줌으로써 피처 모델의 이해관계자들이 직관적으로 이해할 수 있는 모델 구성을 가능하게 한다.

2.4 가변성 모델의 인스턴스 방법의 정의

소프트웨어 제품 라인 개발 전 과정에 가변성 모델이 적용되기 위해서는 가변성 해소 시점에 대한 정의가 필요하다. 전통적인 피처 모델의 경우에는 가변성 해소에 관한 정의가 없다. 피처 가변성 모델의 경우에는 비즈니스 측면에서 가치를 줄 수 있는 인스턴스들을 선별하고 이를 정의해 줄 수 있는 메커니즘이 필요하다. 또한, 구현 가변성 모델의 경우에는 컴파일 시점(compile time)에 가변성을 해소할 수도 있고, 실행 시점(run time)에 가변성을 해소할 수도 있다. 이렇듯, 개발 단계에 따라 가변성 모델의 가변성 해소 시점이 다양하다. 하지만 전통적인 피처 모델링 도구는 다양한 가변성 해소 시점을 반영하지 않았기 때문에 소프트웨어 개발 전 과정에 활용하기 어려운 문제점이 있다. SVM은 각 개발단계에서 요구하는 가변성 해소 시점을 가변성 모델에 적용하게 함으로써 이 문제를 해결한다.

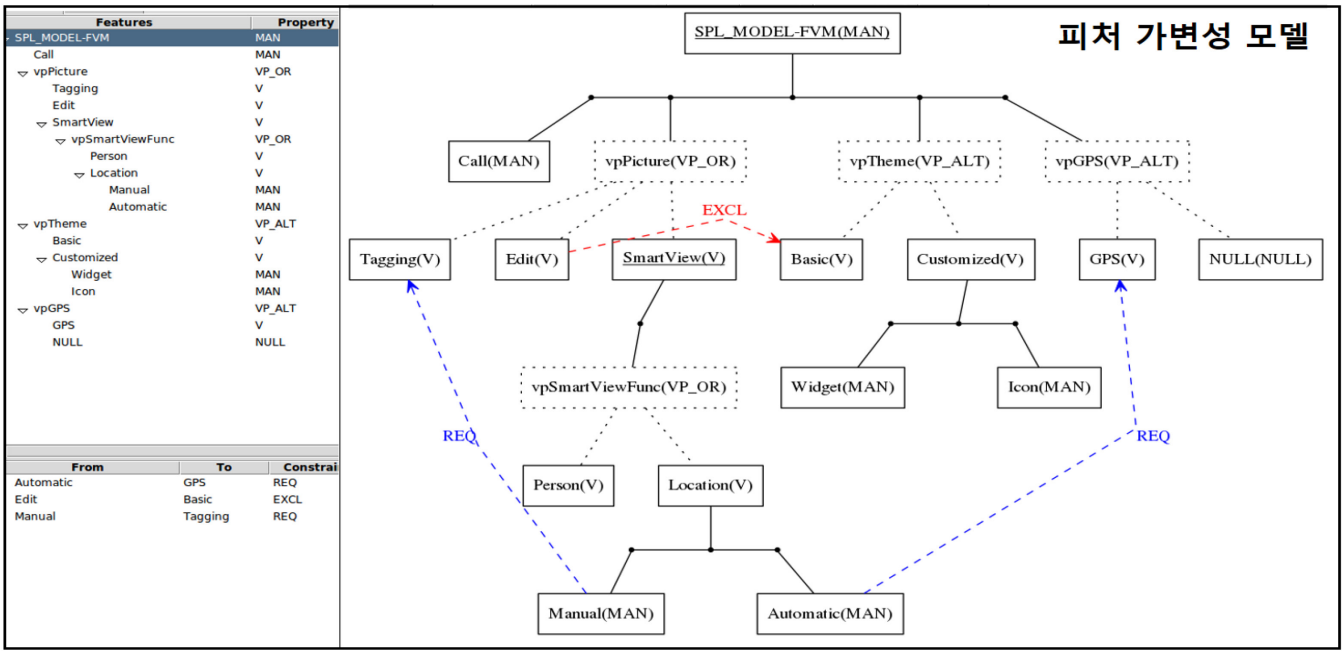


그림 1. 피처 가변성 모델링(FVM) 과정

3. 가변성 모델링 도구 SVM의 개발

이 절에서는 2절에서 기술한 네 가지 모델링 방법을 도구로 구현한다.

3.1 피처 모델과 가변성 모델간의 명확한 구별 및 정의

2.1절에서 FVM, RVM, DVM, ImVM 및 TVM을 제시하였다. RVM에서는 FVM의 피처에 대응되는 것이 요구사항이다. 마찬가지로 DVM에서는 서브시스템 혹은 컴포넌트를 의미하며, ImVM에서는 코드 단위를 의미한다. [그림 1]은 피처 가변성 모델링 과정의 예를 보여준다. [그림 1]과 같이 SVM은 트리뷰(왼쪽 상단), 제약사항뷰 (왼쪽 하단) 및 도식화뷰(오른쪽)로 구성된다. 개발자는 트리뷰와 제약사항뷰에서 모델링을 할 수 있으며, 그 결과는 도식화뷰에서 보여준다. [그림 1]은 피처 가변성 모델을 활용하여 모바일 제품군을 사용자 관점에서 모델링 한 결과이다. Call 피처는 모든 제품들에 포함 된 피처이기 때문에 필수적(Mandatory) 피처로 모델링 하였다. 반면에 GPS는 vpGPS(VP_ALT), GPS(V) 및 NULL(NULL)을 활용하여 모델링 함으로써 선택적(Optional) 피처 모델링이 가능하다. 이 의미는 제품군에 속한 제품들 중 일부는 GPS를 가지고 있고, 다른 일부는 GPS를 가지고 있지 않다는 것을 의미한다. 이를 통해 [표 1]의 '[T1] 소프트웨어 개발주기에서의 피처 모델 역할의 정의' 문제를 해결하였다.

3.2 가변성의 명시적 표현

SVM에서는 가변성을 명확하게 표현 하기 위해 가변점(VP, Variation Point)과 가변값(V, Variant)의 개념을 도입한다. [그림 1]의 도식화뷰에서 가변점은 네모점선박스와 박스 안 이름 앞에 'vp'를 표기함으로써 명확하게 표현한다. 가변값은 실선네모박스와 박스 안 이름 뒤에 '(V)'를 표기함으로써 명확하게 표현한다. 피처와 피처와의 관계 혹은 피처와 가변점 과의 관계를 표현하는 구성 관계(composed of)는 실선으로 표현되고, 가변점과 가변값의 관계는 점선으로 표현 된다. 이를 통해 [표 1]의 '[SP-V] 가변점과 가변값의 구별된 표현 문제'를 해결하였다.

3.3 관심의 분리(Separation of Concerns) 원리의 적용

SVM에서는 이해관계자들의 관심 사항에 적합한 가변성 모델을 지원하기 위해 논문 [9]에서 제시한 분리 모델링 방법 및 뷰 구축 방법을 개발하여 지원한다. SVM에서는 관심사항을 구성 관계, 가변 관계 및 제약 관계로 정의하였고, 분리 단위는 '부모-자식피처 그룹'으로 정의함으로써 [표 1]의 '[S] 확장성 문제 및 [C] 복잡성 문제'를 개선하였다.

3.4 가변성 모델의 인스턴스 방법의 정의

SVM에서는 각 가변성 모델에 적합한 가변성 해소 방법을 정의한다. 피처 모델에서 Valid configuration 분석 방법은 가능한 인스턴스들을 모두 생성하는 분석 기법이다. 이 때의 가변성 해소 시점은 단일하다고 할 수 있다. 도구 상에 가변성 해소 시점들을 명확하게 정의하지 않은 이유는 각 가변성 모델마다 가져야 하는 가변성 해소 시점에 대한 연구가 더 필요하기 때문이다. 따라서, SVM에서는 사용자가 가변점 해소 시점을 정의하여 사용할 수 있도록 개발함으로써 [표 1]의 '[VR] 가변성 해소 정의 메커니즘 부재'를 개선하였다.

표 1. 전통적인 피처 모델링 방법과 제안한 모델링 방법의 비교

피처 모델의 문제점		FORM 피처 모델 [3]	E-FORM 피처 모델 [4]	제안한 방법 (CFM 기반의 SVM)
[SP] 과학적 원리 미흡	[SP-L] 엄밀한 언어			
	[SP-V] 가변점과 가변값의 구별된 표현			○
	[SP-R] 추론 지원			
[T] 추적성 문제	[T1] 소프트웨어 개발 주기에서의 피처 모델 역할의 정의			○
	[T2] 피처 모델과 도메인 요소들간의 추적성 문제	△	△△	△△
	[T3] 피처 모델과 애플리케이션 요소들간의 추적성 문제			△△
[S] 확장성 문제			△	△△
[C] 복잡성 문제		△	△△	△△
[VR] 가변성 해소 정의 메커니즘의 부재			△	△△
[G] 피처 그룹핑 메커니즘의 부재				

△: 부분적으로 처리 됨. △△: 개선되었으나, 여전히 부분적으로 처리 됨. ○: 완전히 처리 됨.

4. 평가

본 논문에서는 제안한 도구를 평가하기 위해, [8]에서 제시한 비교 방법을 활용한다. [표 1]은 [8]에서 제시한 비교 관점을 바탕으로 전통적인 피처 모델들과 제안한 방법을 비교하였다. '피처 모델과 가변성 모델간의 명확한 구별 및 정의'함으로써 '[T] 추적성 문제'가 FORM 피처 모델과 E-FORM 피처 모델에 비해 개선되었다. '가변성의 명시적 표현'을 제공함으로써 '[SP] 과학적 원리 미흡 문제'를 일부 해결하였다. '관심의 분리(Separation of Concerns) 원리의 적용'함으로써 '[S] 확장성 문제'와 '[C] 복잡성 문제'를 일부 해결하였으며, '가변성 모델의 인스턴스 방법의 정의'함으로써 '[VP] 가변성 해소 정의 메커니즘'의 부재 문제를 개선하였다.

5. 결론

본 논문에서는 전통적인 피처 모델의 문제점을 해결하기 위한 새로운 가변성 모델링 도구 SVM을 소개하였다. 또한 SVM 을 이용하여 CFM 가변성 모델링 방법을 전통적인 피처 모델링 방법들과 비교하였다.

후속 연구에서는 SVM을 실제 사례에 작용함으로써 CFM 가변성 모델링 방법을 검증하고, 가변성 모델을 분석하는 연구를 수행할 예정이다.

Acknowledgement

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW중심대학지원사업의 연구결과로 수행되었으며 (2016-0-00018), 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(NRF-2017M3C4A7066210).

참고문헌

[1] 강성원, 체계적인 소프트웨어 제품라인 개발, 홍릉과학출판사, 2017 년 6 월.

[2] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., Peterson, A. S.,"Feature-Oriented Domain Analysis (FODA) Feasibility Study," SEI Technical Report, 1990.

[3] Kang, K. C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M., "FORM: A Feature-Oriented Reuse Method with Domain Specific Reference Architectures," Annals of Software Engineering, vol. 5, pp. 143 - 168, 1998.

[4] Choi, H., Lee, K., Lee, J., J., Kang, K. C., "Multiple Views of Feature Models to Manage Complexity," Workshop on Scalable Modeling Techniques for Software Product Lines (SCALE), 2009.

[5] Pure::variant, https://www.pure-systems.com/

[6] FeatureIDE, https://featureide.github.io/

[7] SPLOT, http://splot-research.org/

[8] Kang, S., Han, Y., Ahn, H., Lee, J., "Critical Analysis of Feature Model Evolution," International Conference on Software Engineering and Service Science(ICSESS), 2017.

[9] 강성원, "CFM: 합성적 피처모델", CS-TR-2017-414, KAIST 전산학부 Technical Report, 2017.

독해력 증진 소프트웨어의 디자인을 위한 매체별 읽기 양상 차이 연구

기경서 ○, 정유진, 권가진

서울대학교 융합과학부

kskee88@snu.ac.kr, yojeong90@snu.ac.kr, ggweon@snu.ac.kr

A Study on Difference of Reading Patterns by Media Types to Design Reading Enhancement Software

Kyung Seo Ki○, You Jin Jeong, Gahgene Gweon

Department of Transdisciplinary Studies, Seoul National University

요 약

본 연구는 디지털 환경 기반에서 독해력 증진을 위한 교육 소프트웨어를 적절하게 디자인하기 위하여 매체에 따라 읽기 양상에서의 차이가 발생하는지를 탐구하였다. 선행 연구 결과에 따르면, 읽기 과정에서 무의식적으로 텍스트를 건너뛰는 현상이 종이에서보다 디지털 매체에서 보다 잘 일어나는 것으로 알려져 있다. 이러한 현상의 발생은 디지털 환경에서 독해력 증진을 시도하는 교육 소프트웨어를 개발하고자 할 때, 인지적 차원에서 종이 환경과는 다른 접근이 필요할지도 모른다는 우려를 환기시킨다. 이에 본 연구는 '인지적 노력'과 '체감 독해 수준', 그리고 '독해 수준'과 같은 인지 요소들에서, 매체에 따라 유의미한 변화가 일어나는지를 검토하였다. 하지만 시선 추적 기법과 설문을 활용하여 30명을 대상으로 종이, 컴퓨터, 태블릿 환경에서 각각 실시된 실험에서, 종이와 디지털 환경 간의 차이는 유의미하게 나타나지 않았으며, 컴퓨터와 태블릿 PC간의 차이만이 유의미하게 나타났다. 이는 디지털 환경에서의 독해 교육도 기존의 종이 환경에서 수행된 교육법의 연장선상에서 접근할 수 있으리라는 시사점을 제공한다.

1. 서 론

교육 분야에서 디지털 매체를 활용한 교육 소프트웨어를 설계한다고 할 때, 매체 자체가 가진 영향력을 고려해야 하는지 아닌지에 대해서는 현재 다소 결론을 내리기 어려운 상황이다. 스마트폰이나 태블릿 PC와 같은 휴대용 디지털 기기가 점차 급속하게 보급되며 디지털 매체의 보편화가 장차 우리의 인지 패턴을 어떻게 바꾸게 될 것인지에 대한 관심이 고조되고 있지만[1], 매체 환경의 변화가 실제로 사람들의 인지 능력에 어떤 영향을 미치는지에 대해서는 의견이 분분하기 때문이다. 본 연구에서는 이런 상황에 주목하여, 디지털 기반의 독해력 증진 소프트웨어를 올바른 방향으로 설계하기 위해 디지털 매체 환경에서의 읽기가 기존의 종이 읽기와 실제로 특기할 만한 차이를 보이는지에 대해 인간-컴퓨터 상호작용 분야의 관점에서 탐구하여 보고자 하였다.

본 연구는 소프트웨어의 적절한 설계 방향을 수립하기 위해 읽기 활동에 영향을 미치는 인지적 요인들인 '인지적 노력'과 '체감 이해 수준', '독해 수준'에 주목하였다. 이를 위해 시선 추적, 설문 및 시험을 활용한 실험을 통한 비교 분석을 수행하였으며, 특히 기존에 시도되지 않았던 종이와 디지털 매체 간 시선 분포 차를 시선 추적 장치를 통해 중점적으로 비교하였다. 논문의 구성은 다음과 같다. 2장에서는 관련 선행 연구를 소개하며, 3장에서는 실험 설계 및 진행 과정에 대해 설명한다. 4장에서는 실험 결과를 분석하고 시사점을 논의하며, 5장에서는 결론 및 향후 후속 연구 방향에 대해 논의한다.

2. 연구 배경

디지털과 종이 매체간의 차이를 탐구한 선행 연구들은 크게 두 가지 유형으로 나누어볼 수 있다. 이는 (1) 시선 추적 기법을 사용하여 디지털 환경에서 읽기 활동을 수행하는 동안의 시선 이동 패턴을 관찰한 연구들[2,3]과 (2) 디지털과 종이 각 매체 환경에서의 수행에서의 차이를 비교한 연구들[4,5,6,7]이다.

우선 (1) 유형의 선행 연구 사례들은 디지털 매체 환경에서 글을 읽을 때 텍스트의 내용 중 일부를 독자가 건너뛰는 듯한 패턴이 나타나고 있다고 보고하고 있다. 대표적으로 Pernice et al. 의 2006년 연구[2]는 이 현상을 'F-Shape Reading Pattern'이라 명명하였다. 이 명칭의 유래는 시선 추적 기법을 통해 드러난 읽기 패턴이 여러 줄로 이루어져 있는 텍스트의 첫번째 줄은 온전하게 다 읽지만, 그 다음 줄부터는 점차 뒷부분을 누락시키는 듯한 모습을 띄기에 알파벳 철자 'F'자와 유사한 모습을 보인다는 데서 기인하고 있다. 이 현상은 2012년 수행된 다른 연구[3]에서도 재현되었으며, 특히 이 선행 연구 사례는 시선 추적을 사용하여 파악한 결과 스마트폰에서도 이 현상이 나타난다고 보고하고 있다. 이러한 사례들은 어떤 사람들이 디지털 매체 환경에서 텍스트를 읽을 때, 종이와는 달리 무의식적으로 텍스트의 세부 내용을 누락시키며 읽기 활동을 하고 있을지도 모른다는 의심을 합리적으로 갖게 한다고 볼 수 있을 것이다. 하지만 종이에서는 위의 선행 연구 사례들과 같은 분석이 수행되지는

않았으므로, 디지털 매체와 종이 매체 간의 시선 패턴의 차이가 있는지의 여부가 아직 명료하게 규명되지는 않았다.

반면 (2) 유형의 선행 연구 사례들은 디지털 매체 환경과 종이 매체 환경에서의 읽기가 실제로 수행 역량에서의 차이를 불러일으키는지에 대한 관심을 중심으로 수행되었다. 일례로 Mangen et al. 2013[4]은 동일한 텍스트를 가지고 실험을 수행한 결과, 종이보다 디지털 매체 환경에서 학생들의 점수가 더 낮았다고 보고하고 있다. Kaufman et al. 2016의 연구[5] 또한 디지털 매체의 경우가 단순하고 구체적인 정보 탐색에 더 적합한 반면, 종이 매체가 텍스트의 요지를 파악하는데 더 적합한 것으로 보인다고 주장하고 있다. 하지만 이와 반대로 디지털 매체 환경에서의 읽기가 건너뛰기를 통해 중요한 정보에 더 효율적으로 접근하므로 더 효과적이라고 주장하는 Duggan et al. 과 같은 연구 사례도 있으며[6], 디지털 매체와 종이 매체 간 차이가 유의미하지 않다고 주장하는 Margolin et al.의 연구 사례[7]도 있다. 이처럼 디지털 매체와 종이 매체의 차이에 대해서는 지금까지 다양한 연구들이 수행되어 왔지만, 디지털 매체와 종이 매체 간의 우열이 존재하는지에 대해서는 아직도 의견이 갈리는 상황이라고 볼 수 있다.

한편 Singer et al. 2017b에 따르면, 디지털 매체 환경과 종이 매체 환경간의 차이를 비교한 연구는 2001년부터 2017년까지 총 36건이 수행되었으며, 시선 추적 기법과 질문지 및 시험 점수를 결합한 연구는 이 중 5건으로 보고되고 있다.[8] 하지만, 이 5건의 연구들 중 디지털과 종이 매체 환경 각각의 시선 고정 분포 밀도 차이를 동시에 비교한 사례는 없었다. 대신 이 연구들은 시선 고정 지속 시간이나 안구 이동 시간 등의 시계열 자료만을 활용하여 비교를 수행하였다. 물론 시선 고정 지속 시간이나 안구 이동 시간은 충분히 독자의 텍스트에 대한 정보 처리를 반영하는 척도로 활용될 수 있다. 시선 고정 지속 시간의 경우는 텍스트의 통합적 이해 과정을 반영하는 것으로 알려져 있으며, 안구의 이동 시간은 단어 처리를 반영하는 것으로 알려져 있기 때문이다.[9] 하지만, 이러한 시계열 정보는 텍스트의 전체 영역에서 사람들이 어느 부분을 더 보고 어느 부분을 덜 보았는지에 대해서는 파악할 수 없다는 한계가 있다.

3. 실험 설계 및 수행

선행 연구 사례들의 연구 결과들을 토대로, 본 연구에서는 기존 연구들의 한계점을 보완하기 위해 시선 추적 기법과 설문을 혼합하여 실험을 설계하였다.

3.1 연구 문제 및 가설 설정

우선 본 연구에서는 종이와 디지털 매체 환경에서의 인지적 노력과 자각에서의 차이가 발생하는지를 탐구하는 것을 연구 문제로 설정하였다. 이와 같이 연구 문제를 설정한 이유는, 선행 연구 사례에서 보고된 바와 같이 디지털 매체 환경에서 건너뛰는 식의 (가령, F-Shape) 읽기 패턴이 실제로 존재한다면, 디지털과 종이 매체 간의 인지적 차이가 발생할 것이라고 보았기 때문이다. 이에 따라 본 연구는 검증을 위해 종이 읽기와 디지털 매체 간의 차이가 발생할 것이라는 세 가지 가설을 설정하고 실험을 설계하였다. 따라서 가설에 따른 실험 결과가 유의미하다면 인지 요소에 따른 매체 간의 차이가 발견되는 것이며, 유의미하지 않다면 매체 간의 차이가 없다고 보아야 할 것이다. 설정된 가설은 아래와 같다.

가설 1. 종이 읽기에서보다 디지털 매체에서의 읽기에서 **인지적 노력**이 덜 들 것이다: 이 가설은 인지적 노력의 차이가 디지털 읽기에서 실제로 발생하고 있는지를 검증하기 위해 설정되었다. 기존 연구 결과에 따르면, 인지적 노력은 일반적으로 독자가 글을 이해하면서 경감되는 것으로 알려져 있다.[10] 하지만 디지털 매체에서 무의식적인 텍스트 누락이 발생한다면, 올바른 방식인 이해를 통해서가 아니라 잘못된 방식인 무의식적 누락을 통하여 인지적 노력이 경감될 것이다. 이러한 상황이 발생한다면 사람들은 자신의 인지적 노력이 줄어들고 있다는 상황만을 근거로 자신이 텍스트를 잘 이해해 가고 있다는 잘못된 믿음을 가질 수 있다.

가설 2. 종이에서보다 디지털 매체에서 **체감 독해 수준**이 더 높을 것이다: 이 가설은 종이와 디지털 매체 사이에서 체감되는 독해 수준의 차이가 발생하는지를 검증하기 위해 설정되었다. Singer et al. 2017a에서 수행된 유사 연구 사례에서, 사람들은 사전 설문에서 일반적으로 자신이 종이에서보다 디지털 환경에서 더 텍스트를 잘 읽을 것이라고 생각한다는 응답을 보인 것으로 보고되었다. [11] 이에 본 연구는 이러한 자신감이 디지털 매체에서 자신이 텍스트를 더 잘 이해한다는 믿음으로부터 발생할 것이라고 가정하였다. 이러한 믿음은 위의 가설 1에서 설정한 상황과 결부되어 있을 수 있다. 즉, 무의식적인 누락 현상이 사람들로 하여금 인지적 노력을 경감시키고, 이에 따라 사람들은 인지적 노력 경감의 결과로 자신이 텍스트를 잘 이해하고 있다는 믿음을 갖게 될 수 있다.

가설 3. 종이보다 디지털 매체 환경에서 **독해 수준**이 더 낮을 것이다: 본 연구에서는 만일 텍스트를 무의식적으로 누락하는 현상이 디지털 매체에서 실제로 발생하였다면, 독자의 주관적 판단과는 달리 실제 독해 점수에서 차이가 나타날 것이라고 가정하였다. 이 가설은 주관적 상태만을 보고자 하는 위의 두 가설과는 달리, 객관적인 독해 점수 차이가 발생하는지를 보기 위해 설정되었다.

3.2 피험자

총 30명(남 13명, 여 17명)의 피험자를 모집하여 within-subject 방식으로 실험을 실시하였으나, 7명의 데이터가 기기 고장 및 소음 등으로 오염되어 총 23명의 데이터로만 분석을 수행하였다. 피험자는 만 19세에서 31세까지의 디지털 매체를 일상적으로 사용하는 대학생이거나 대학원생이었으며, 연령 평균은 25세 정도였다. 실험에 대한 보상으로는 1인당 1만원 상당의 상품권을 지급하였다.

3.3. 측 정

위의 가설들을 검증하기 위해 본 연구에서는 시선 추적

기법과 설문, 그리고 시험의 세 가지 지표를 활용하였다.

우선, 선행 연구들의 한계점들을 보완하기 위해 본 연구는 시선 추적 장치를 활용하여 각 매체별로 텍스트 내에서 시선 분포가 어떻게 이루어지는지를 보고자 하였다. 다음으로, 설문을 활용하여 피험자들의 매체 간 차이에 대한 주관적 상태를 파악하고자 하였다. 마지막으로, 시험을 통해 피험자들의 수행 능력이 매체에 따라 차이를 보이는지를 확인하고자 하였다. 설정된 가설들에 대한 측정 지표들은 아래와 같다.

3.3.1 인지적 노력

가설 1을 검증하기 위해 본 연구에서는 먼저 피험자들이 시선 추적 장치를 착용하고 각 매체 환경에서 읽기를 수행하도록 하여 매체별 시선 고정 분포 데이터를 수집한 뒤, 인지적 노력을 반영하는 것으로 알려져 있는 척도인 체감 난이도를 설문을 통해 측정하여[10] 비교하였다. 시선 추적 데이터는 분석 프로그램을 통해 히트맵으로 변환하여 시선 고정 분포를 파악할 수 있도록 하였다. 측정 장비로는 종이 환경 및 디지털 매체 환경에서의 읽기를 모두 측정할 수 있는 안경형 시선 추적 장치인 Tobii Glasses 2 Pro를 사용하였다.

다음으로 체감 난이도 측정은 5단계의 리커트 척도를 사용하였으며, 문항은 “매우 쉬웠음”에서부터 “매우 어려웠음”까지로 구성되었다.

3.3.2 체감 독해 수준

다음으로 가설 2를 검증하기 위해 체감 독해 수준을 측정하였다. 체감 독해 수준은 설문을 통해 측정하였으며, ‘체감 자신감’과 ‘체감 이해도’의 두 가지 척도를 각각 측정하였다. 우선 체감 자신감은 3가지의 세션이 모두 끝난 뒤 자신이 각 매체별로 텍스트를 얼마나 잘 이해한 것 같은지에 대해 묻는 방식으로 측정되었으며, 체감 이해도는 실험에서의 각 매체별 세션이 끝난 직후 해당 텍스트에 대한 이해도를 묻는 것이었다. 측정은 모두 “잘 이해하지 못함”에서부터 “매우 잘 이해함”까지의 5단계 리커트 척도로 측정되었다.

3.3.3 독해 수준

마지막으로 독해 수준을 판정하기 위해 시험을 실시하였다. 시험은 매체별 각 읽기 세션이 끝난 직후 설문에 앞서 실시되었으며, 5개의 참/거짓 판별 문항으로 구성되었다.

3.3.4 추가 변수

가설 검증과 직접적으로 연관된 변수들 외에도, 본 연구에서는 읽기 과정에 상당한 영향을 미치는 것으로 알려져 있는 변수들[12]인 매체 친숙도와 매체 선호도를 추가로 측정하였다. 특히 매체 친숙도는 실험에서 측정된 독해 수준에 영향을 미칠 수도 있을 것이라 예상되어 중재 변수 역할을 할 것으로 예상되었다. 이 변수들은 모두 “매우 낮음”에서 “매우 높음”까지의 5단계 리커트 척도로 측정되었다.

3.4 실험 절차

본 연구에서는 종이, 컴퓨터, 태블릿 PC의 세 조건을 설정하였다. 실험에서 태블릿 PC를 종이와 컴퓨터 외에 추가로 세팅한 이유는, 태블릿이 종이와 같은 평판 형태이고 이동성이 뛰어나기에 종이와 컴퓨터 간의 중간적인 특성을 보일 것으로 예상되었기 때문이다. 이에 장비로는 Apple iMac 27인치(PC와 Microsoft Surface Pro(태블릿)을 선정하여 실험을 수행하였다. 수행 과정은 아래와 같다.

3.4.1 읽기 자료

각 세션별로 A4 1장(약 420단어 분량)의 한국어 텍스트를 읽기 자료로 활용하였다. 텍스트 출처는 KBS 한국어능력시험 지문으로, 기존의 시험 데이터를 참고하여 오답 비율을 바탕으로 난이도를 고르게 조정하여 선정하였다.

3.4.2 실험 과정

실험 실시 전 피험자는 절차에 대한 설명을 듣고 동의서에 서명한 뒤, 시선 추적을 위한 캘리브레이션(안구 조정)을 수행하였다. 이 과정에서 피험자는 읽기 양상을 측정할 것이며 평소대로 편하게 읽기를 수행하면 되고, 읽기가 끝나면 간단한 참/거짓 문제를 풀게 될 것이라고 전달받았다. 캘리브레이션이 끝나면, 피험자는 장치를 착용하고 곧바로 실험 세션을 시작하였다. 각 세션은 순서 효과를 상쇄하기 위하여 난수 추첨을 통하여 무작위로 배정하였다. 세션별 읽기가 끝난 뒤에는 텍스트를 수거하였으며, 피험자는 곧바로 참/거짓 문제를 푼 뒤 체감 이해도를 묻는 설문에 응답하였다. 설문 응답이 끝나면, 5분 정도의 휴식 시간을 가진 뒤 다음 세션을 똑같은 방식으로 진행하였다. 모든 세션을 완료하는 데에는 45분에서 1시간 정도의 시간이 소요되었으며, 세션이 모두 완료되면 사후 설문을 실시하고 실험을 종료한 뒤 보상을 지급하였다.

3.5. 통계 분석

실험 결과 분석은 시선 추적 데이터에 대해서는 Tobii Lab Pro를, 통계 데이터에 대해서는 SPSS를 사용하였다. 통계 분석에는 Repeated Measures of ANOVA 분석 방법을 사용하였으며, 유의수준으로는 0.05 수준에서 분석을 수행하였다.

4. 결과

시선 추적 기법을 통해 얻은 히트맵 데이터와 설문 및 시험을 통한 측정 데이터 모두 가설에 부합하는 유의미한 결과를 나타내지 않았다.

4.1 히트맵 데이터

그림 1의 히트맵은 23명의 시선 추적 데이터를 각 매체별로 오버래핑하여 제작한 것이다. 히트맵에서 확인할 수 있듯이, 선행 연구 사례에서 언급된 바 있었던 F-Shape는 나타나지 않았으며, 오히려 컴퓨터에서 더 촘촘한 시선 분포가 관찰되었다.

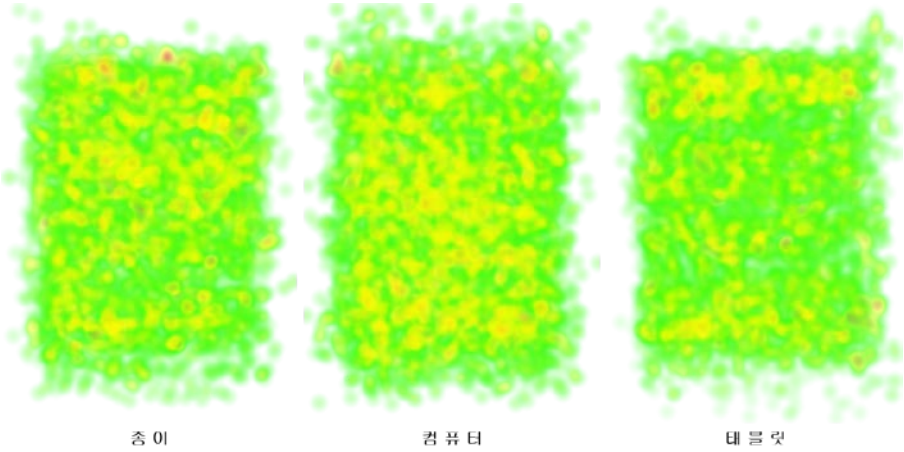


그림 1. 히트맵 분석 결과

4.2 설문 응답 및 독해 점수

설문에서 측정된 체감 난이도($p > 0.583$) 와 체감 자신감($p > 0.723$), 체감 이해도 ($p > 0.445$) 모두 통계적으로 유의미하지 않았다. 다만 컴퓨터와 태블릿 간 체감 이해도의 경우에서만 유일하게 통계적 유의미성($p = 0.048$)이 도출되었다. 이는 매체 선호도에서 피험자 중 단 한 명만이 태블릿을 더 선호한다고 응답한 경향과 관계가 있을 것으로 보인다. 또한, 독해 점수에서도 통계적 유의미성이 도출되지 않았으며, 평균 점수 역시 컴퓨터(8.217), 종이(8.348), 태블릿(8.652)과 같이 나타나 큰 차이가 없었다.

5. 토의 및 결론

'F-Shape' 현상의 발생을 보고했던 선행 연구 결과와는 달리, 히트맵 분석에서 매체에 따른 시선 고정 분포의 현저한 차이는 나타나지 않았으며, 설문 및 독해 점수도 유의미한 차이가 나타나지 않았다. 특히 독해 점수에서의 차이가 유의미하지 않았다는 것은 매체에 따른 독해 차이가 없다는 것을 시사하는 것으로 보인다.

이에 본 연구에서는 텍스트 읽기의 목적이 이러한 결과를 도출하게 만든 것으로 잠정적인 결론을 내렸다. 실험 안내 과정에서 피험자들은 각 세션에서 읽기를 수행하고 간단한 테스트를 수행한다고 전달받았는데, 그 과정에서 상당수의 피험자들이 시험을 본다는 언급에 대해 반응을 하는 모습을 보였다. 핀란드의 한 연구 사례[13]에서도 읽기 과정에서 어떤 과업을 수행하는지가 영향을 미친다는 것이 보고된 적이 있다는 것을 고려한다면, 실험 과정에서 나타난 피험자들의 반응과 최종적으로 도출된 실험 결과는 매체에 따른 효과보다도 읽기의 목적이 읽기 양상에 더 큰 영향을 준다는 것을 간접적으로 드러내고 있는 것으로 보인다.

따라서 본 연구 결과를 고려한다면, 읽기의 목적을 적절하게 조절함으로써 매체에 따른 독해 양상 변화를 고려하지 않고도 충분히 디지털 환경에서 독해 증진 교육을 수행할 수 있을 것으로 판단된다. 이에 본 연구는 후속 연구로 교육학 분야에서 이미 축적되어 있는 기존의 독해 지도법의 성과들에 기반하여, 웹 기반의 독해력 증진을 위한 교육 소프트웨어를 제작하고 그 효과를 검증하여 보고자 한다.

Acknowledgement

이 성과는 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-이공학 개인기초연구 지원사업의 지원을 받아 수행된 연구임(No. 2017R1D1A1B03034511).

참고 문헌

[1] John Harris. 2010. How the internet is altering your mind, *The Guardian*.

[2] Kara Pernice et al. 2006. *How People Read in The Web: The Eyetracking Evidence*, Nielsen Norman Group.

[3] Ralf Biediert et al. 2012. Reading and estimating gaze for smart phones. In *ETRA '12 Proceedings*, pp. 385-388.

[4] Mangen, A. et al. 2013. Reading Linear Texts on Paper Versus Computer Screen: Effects on Reading Comprehension. *International Journal of Educational Research* 58: 61-68,

[5] Kaufman et al. 2016. High-Low Split: Divergent Cognitive Construal Levels Triggered by Digital and Non-Digital Platforms, In *CHI 2016.*

[6] Duggan. Geoffrey B et al. 2011. Skim reading by satisficing: evidence from eye tracking, In *CHI 2011.*

[7] Margolin, S. J. et al. 2013. E-readers, computer screens, or paper: Does reading comprehension change across media platforms? *Applied Cognitive Psychology* 27, pp. 512-519.

[8] Singer. Lauren M, and Patricia A. Alexander. 2017b. Reading on Paper and Digitally: What the Past Decades of Empirical Research Reveal, *Review of Educational Research.*

[9] Rayner, K. 1998. Eye movements in reading and information processing: 20 years of research. *Psychological Bulletin*, 124, 372-422.

[10] Brünken, R. et al. 2003. Direct measurement of cognitive load in multimedia learning. *Educational Psychologist*, 38.

[11] Singer, L. M., & Alexander, P. A. 2017a. Reading Across Mediums: Effects of Reading Digital and Print Texts on Comprehension and Calibration. *The Journal of Experimental Education*, 85.

[12] Lauterman, T, & Ackerman, R. 2014. Overcoming screen inferiority in learning and calibration. *Computers in Human Behavior*, 35.

[13] Kaakinen, J. K., and J. Hyönä. 2010. Task Effects on Eye Movements during Reading. *Journal of Experimental Psychology: Learning Memory and Cognition* 36.

OSEK/VDX 기반 소프트웨어의 API 호출 검증 사례연구: Erika 어플리케이션 검증[1]

김동우, 최윤자
경북대학교 컴퓨터학부
Kdw9242@gmail.com, yuchoi76@knu.ac.kr

API-call Constraint Checking on OSEK/VDX-based Software: a Case Study on Erika Applications

Dongwoo Kim, Yunja Choi
Kyungpook National University, Department of Computer Science

요 약

IoT 기술의 발전과 더불어 많은 분야에서 IoT 소프트웨어를 사용하고 있지만 IoT 소프트웨어의 안전성은 입증되지 않았다. 안전성이 요구되는 분야에 IoT 기술이 사용될 경우 사소한 버그로 인하여 인명피해나 재산 피해를 가져올 수도 있기 때문에 반드시 안전성을 입증하여야 한다. 선행연구에서는 어플리케이션이 운영체제에서 제공하는 API를 사용상의 규칙에 맞지 않게 사용할 경우 운영체제가 가질 수 있는 최악의 상황인 시스템 실패에 도달함을 확인하였다. 본 연구에서는 Erika IoT 운영체제와 어플리케이션을 대상으로 API 사용상의 규칙에 관한 안전성 검증을 수행해 보았으며, 어플리케이션에서 발생한 다수의 제약사항 위반이 운영체제에 어떠한 영향을 미치는지 확인해 보았다. 확인 결과, 다수의 Erika 어플리케이션에서 제약사항 위반이 발견되었으며 이를 운영체제가 임의로 처리함을 확인하였다. 하지만 어플리케이션에서 위반한 표준을 운영체제가 처리하는 것이 안전한지는 고려해 볼 사항이다.

1. 서 론

IoT 산업의 발전과 더불어 금융, 의료, 건축, 운송, 서비스, 제조업 등 많은 분야에서 IoT를 접목하고 있다 [1]. 이에 발 맞추어 저전력, 실시간 지원 등의 다양한 요구사항을 충족시키고자 FreeRTOS, RIOT, Contiki, Zephyr, Erika [2] 등 여러 운영체제가 개발되었다. IoT 산업은 향후 더욱 많은 분야로 확산되어 소비자의 다양한 요구사항에 부응할 것으로 기대된다 [3].

IoT 운영체제는 어플리케이션이 사용할 다양한 시스템 기능들을 API(Application Programming Interface)라는 인터페이스를 통해 제공하고 있으며 각각의 API에는 사용상의 규칙이 있다. 어플리케이션은 사용상의 규칙을 준수하여 API를 사용해야 하지만 프로그램의 크기가 커지고 내용이 복잡해질수록 사람의 눈으로 모든 규칙이 모두 준수되었는지 확인하는 것은 불가능하며, 선행연구에서는 사용상의 규칙이 위반될 경우 시스템 실패와 같은 문제로 이어질 수 있음을 보였다 [4]. 시스템 실패는 소프트웨어가 가질 수 있는 최악의 상황이며 안전 중요 분야에 사용될 경우 인명이나 재산에 막대한 피해를 가져올 수 있으므로 반드시 제거되어야 한다.

본 연구에서는 다양한 IoT 운영체제 중 차량전장용 운영체제 표준인 OSEK/VDX [5] 표준을 준수하고 OSEK/VDX로부터 인증을 획득한 Erika를 사례연구 대상으로 선정하고 Erika 운영체제에서 작동하는 다양한 어플리케이션이 API 사용상의 규칙을 준수하는지 자동 검증기를 [6] 통해 확인하였으며, 자동 검증기의 입력 규칙에 맞게 몇 가지 변환 규칙을 제정하였다. 검증 결과, OSEK/VDX에서 명시한 사용상의 규칙 중 일부를 위배하는 것을 확인하였으나 운영체제에서 해당 사용상의 규칙을 위반할 경우에도 시스템이 안정적으로 작동할 수 있음을 소스코드 레벨에서 확인하였다.

2. 연구배경

2.1 OSEK/VDX와 Erika 운영체제

OSEK/VDX 기반 소프트웨어는 운영체제, 어플리케이션, 환경설정으로 구성된다. 운영체제는 Task, Event, Resource, Alarm, ISR로 구성되어 있으며 소프트웨어가 사용하는 각 구성요소에 관한 정보는 환경설정에 기술되어 있다. 각각의 구성요소는 표준에서 정의하는 기능과 속성들을 가지고 있으며, 운영체제에서 정의하는 API를 통하여 새로운 Task를 활성화하거나 Resource를 할당 받는 작업 등을 수행할 수 있다.

그림 1의 왼쪽은 OSEK/VDX 어플리케이션의 모습을 소개한다. 어플리케이션은 두 개의 Task의 행위를 정의하고 있으며 Task t1은 Task t2를 활성화하고 (줄

[1] 이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보 컴퓨팅기술개발사업의 지원을 받아 수행된 연구임 (No. 2017M3C4A7068175).

02) Resource를 관리하고 (줄 03~05) TerminateTask를 호출함으로써 (줄 06) 종료한다. 그림 1의 오른쪽은 환경설정의 모습을 보여주며 두 개의 Task t1과 t2, Event e1 그리고 Resource r1을 정의한다. 환경설정에 따라 Task t1은 시스템 시작과 동시에 활성화되며 (줄 02) 우선순위는 5이고 (줄 03) Event e1과 Resource r1을 가지고 있으며 (줄 04~05) 최대 활성화 될 수 있는 인스턴스는 3개까지 (줄 06) 허용된다.

어플리케이션	환경설정

01: **TASK**(t1) {
02: ActivateTask(t2);
03: GetResource(r1);
04: ...
05: ReleaseResource(r1);
06: TerminateTask();
07: }
08: **Task**(t2) {
09: TerminateTask();
10: }
	01: **TASK**(t1) {
	02: AUTOSTART = TRUE;
	03: PRIORITY = 5;
	04: EVENT = e1;
	05: RESOURCE = r1;
	06: ACTIVATION = 3;
	07: }
	08: **Task**(t2) { ... }
	09: **EVENT** e1{};
	10: **RESOURCE** r1{};

그림 1 OSEK/VDX 기반 소프트웨어

Erika 운영체제는 OSEK/VDX 표준을 준수하는 IoT 운영체제이며 OSEK/VDX로부터 인증을 획득하였다. Erika 운영체제는 OSEK/VDX에서 명시하는 API를 지원하며 다양한 스케줄링을 제공한다. 또한 주기적 Task를 구현하기 위한 방법으로 OSEK/VDX에서 정의하는 Alarm 외 하드웨어 타이머를 사용한 주기적 Task인 PIT(Programmable Interval Timer)을 지원한다.

2.2 API 호출 제약사항과 안전성 검증

OSEK/VDX는 어플리케이션이 시스템의 다양한 기능을 수행할 수 있도록 26개의 API를 제공하며 각 API에는 사용상의 규칙이 있다. 선행연구 [4] 에서는 OSEK/VDX 기반 운영체제의 강건성을 확인하고자 OSEK/VDX 운영체제와 API 호출 제약사항을 정형 언어로 모델링 하였으며 이로부터 다양한 테스트 케이스들을 생성하여 OSEK/VDX 기반 운영체제 중 하나인 Trampoline에 적용한 결과 사용상의 규칙을 위반할 경우 시스템 실패가 유발됨을 확인하였다.

표 1은 OSEK/VDX에서 정의한 API와 제약사항을 정리한 표이다. 제약사항 C2는 TerminateTask, ChainTask API와 관련된 제약사항이며 이들 API를 호출함으로써 Task가 종료되어야 한다고 명시하고 있다. 제약사항 C4는 SetEvent와 WaitEvent API와 관련된 제약사항으로 Task가 WaitEvent를 호출하여 Event를 기다릴 경우 SetEvent를 호출함으로서 Event를 전달해 주어야 한다고 명시한다.

선행연구 [6]에서는 OSEK/VDX 기반 어플리케이션이 API 호출 제약사항을 준수하였는지 확인하는 자동

표 1 API 호출 제약사항

No	API 함수와 제약사항
C1	**GetResource/ReleaseResource** A critical section shall always be left using ReleaseResource
C2	**ChainTask/TerminateTask** Ending a task function without a call to TerminateTask or ChainTask is strictly forbidden and may leave the system in an undefined state.
C3	**GetResource/ReleaseResource/WaitEvent/TerminateTask/ChainTask/Schedule** It is not allowed to use services that are points of rescheduling for non preemptable tasks (TerminateTask, Chaintask, Schedule, and WaitEvent) in critical sections.
C4	**SetEvent/WaitEvent** An event waited by a task shall eventually be set.
C5	**ClearEvent** The system service ClearEvent is restricted to extended tasks that own the event.
C6	**WaitEvent** This service shall only be called from the extended task owning the event.
C7	**SetEvent** Referenced task shall be an extended task.
C8	**ActivateTask/TerminateTask** An activated task eventually terminated. (legal behavior)

검증도구를 설계 및 구현하였다. 이 도구는 어플리케이션과 환경설정을 입력 받은 후 API 호출 안전성 검증을 수행하고 검증 결과 찾은 API 호출 제약사항 위반을 사용자에게 알려준다. 자동 검증 도구는 지역제약사항과 전역제약사항을 분류하여 수행한다. 지역제약사항 검증은 운영체제의 행위를 고려하지 않고 검증할 수 있는 경우 수행하며 한 번에 하나의 Task만을 대상으로 검증을 수행하기 때문에 검증 속도가 빠르다. 제약사항 C2는 이 조건을 만족하므로 지역제약사항에 속한다. 반면에 전역제약사항의 경우 운영체제의 행위를 고려한 검증이 필요한 경우에 사용이 되며, 전역제약사항 검증 시에는 선행연구 [4] 에서 정의된 운영체제 모델과 어플리케이션을 정형 언어로 나타낸 후 정형 검증한다. 제약사항 C4는 운영체제의 스케줄링에 따라 제약사항 위배여부를 알 수 있으므로 전역제약사항에 속한다.

3. Erika 어플리케이션의 API 호출 제약사항 검증

선행연구 [6]에서 구현한 API 호출 안전성 검증 도구를 Erika 어플리케이션에 적용하기 위해서는 OSEK/VDX 표준과 더불어 Erika에서 추가적으로 지원하는 기능을 다루어주어야 한다. 첫 번째로 OSEK/VDX에서는 어플리케이션의 행위를 각 Task에 정의하며 메인 함수는 커널을 활성화하는데만 사용하지만 Erika에서는 옵션을 통해 메인 함수를 하나의 Task로 사용할 수 있도록 지원하고 있으며 이 경우 메인 함수에서 별도로 커널을 활성화하지 않는다. 이 옵션은 검증 도구에서 지원되지 않는 내용이므로 다루어 주어야 한다. 두 번째로 선행연구에서 만들어진 도구는 컴파일이 가능한 C 코드만 검증할 수 있지만 Erika 어플리케이션에서는 다수의 하드웨어 의존 라이브러리가 사용되고 이

라이브러리에서는 다수 어셈블리가 사용되고 있어 추상화 할 필요가 있다. 세 번째로 Erika에서는 PIT을 이용한 주기적 Task를 지원하지만 이는 OSEK/VDX 표준 및 검증도구에서 지원되는 내용이 아니므로 동일한 의미를 가지는 Alarm으로 변환해 주어야 한다.

3.1 메인 함수 변환

Erika 어플리케이션이 옵션을 통해 메인 함수를 하나의 Task로 사용할 경우 어플리케이션에서 정의된 메인 함수를 하나의 Task로 전환하는 작업을 수행한다.

```
01: int main() {
02:   // job of main
03: }
```

```
01: TASK(MAINTASK) {
02:   // job of main
03: }
04: int main() {
05:   StartOS(DefaultAppMode);
```

그림 2 메인 함수 변환

그림 2는 메인 함수를 변환하는 모습을 나타낸다. 메인 함수에 정의된 작업들은 MAINTASK라는 이름을 가진 Task에 정의하며 메인 함수는 OSEK/VDX 표준과 같이 커널 프로그램을 실행하는 역할을 수행하도록 변환하였다.

```
01: TASK(MAINTASK) {
02:   AUTOSTART = TRUE;
03:   PRIORITY = 0;
04:   // EVENTs & RESOURCES
05:   ACTIVATION = 1;
06: }
```

그림 3 환경설정에 삽입된 MAINTASK

환경설정에는 그림 3과 같이 MAINTASK라는 이름을 가진 Task를 추가하였다. 본 연구에서는 시뮬레이션을 통해 Erika 어플리케이션의 메인 함수가 Task로 사용될 경우 소프트웨어의 시작과 동시에 시작됨을 확인하였으며, Task의 최소 우선순위가 할당됨을 확인하였다.

3.2 하드웨어 의존 코드 생성

하드웨어 의존코드는 Erika 어플리케이션에서 다수 사용이 되며 어셈블리를 사용한 경우가 많다. 어셈블리 언어는 기존 검증 도구를 통해 분석될 수 없으며, 이들을 고려한 검증은 다수 검증비용의 증가를 유발하므로 본 연구에서는 하드웨어 의존 함수 및 변수를 추상화하여 검증하였다.

그림 4는 하드웨어 의존 함수 및 변수를 정의하는데 사용한 템플릿이다. Erika 어플리케이션에서 사용한 변수 중 선언되지 않은 변수와 함수는 hardware_var 템플릿과 hardware_func 템플릿을 이용하여 선언하였다. 각 함수 및 변수는 nondet()을 반환하도록 하였으며, nondet()은 선행 연구에서 개발된 검증 도구에서 "어떤 값이든 할당될 수 있다"로 인식하므로 본 템플릿을 활용할 경우 안전하게 추상화 할 수 있다.

```
int hardware_var = nondet();
int hardware_func { return nondet(); }
```

그림 4 하드웨어 의존 함수 및 변수 정의 템플릿

3.3 PIT을 Alarm의 형태로 변환

Erika에서는 주기적 Task를 Alarm과 PIT을 통하여 구현할 수 있도록 지원하고 있지만 둘은 같은 역할을 수행하므로 본 연구에서는 OSEK/VDX에서 정의되지 않은 PIT을 Alarm의 형태로 변환한다.

그림 5의 윗부분은 PIT을 이용하여 주기적 Task를 구현한 모습이다. EE_pit_handler에는 (줄 01~03) 주기적 Task의 행위가 기술되며 주기적 Task를 초기화하고 (줄 05) 주기를 설정하여 (줄 06) 활성화하고 제거하는 방법을 (줄 08~09) 보여준다.

```
01: void EE_pit_handler() {
02:   // job of periodic task
03: }
04: TASK(SomeTask) {
05:   EE_pit_init();
06:   EE_pit_set_periodic(20);
07:   ...
08:  EE_hal_enableIRQ();
09:  ...
10:  EE_hal_disableIRQ();
11:}
```

```
01: TASK(PIT_HANDLER) {
02:   // job of periodic task
03: }
04: int pit_period = 0;
05: TASK(SomeTask) {
06:   pit_period = scale(1000);
07:   pit_period = scale(20);
08:   ...
09:   SetRelAlarm(PIT_ALARM, 0, pit_period);
10:   ...
11:   CancelAlarm(PIT_ALARM);
12: }
```

그림 5 PIT를 Alarm으로 변환하는 모습

본 연구에서는 그림 5의 아랫부분과 같이 새로운 Task PIT_HANDLER를 추가하여 주기적 행위를 명시하였으며 (줄 01~03) 주기를 설정할 경우 전역변수 pit_period를 이용하여 (줄 04) 주기 값을 저장하도록 하였다. Alarm을 활성화할 때에는 SetRelAlarm을 이용하여

표 2 Erika 어플리케이션에 API 호출 제약사항 검증을 적용한 결과

검증 대상	Loc	Task 수	지역제약사항 검증		전역제약사항 검증	
			시간(초)	위반 사례	시간(초)	위반 사례
cortex_mx/demo_tcp	987	3	3.152	3(C2)	3.221	2(C8)
mico32/demo_tcp	286	2	3.488	1(C2)	0.983	2(C8)
cortex_mx/STM32_FatFs_SD	4276	3	18.064	3(C2)	5.115	2(C8)
s12xs/monostack_resource_demo	347	2	2.493	-	0.712	-
tricore/porting_example	54	2	3.843	-	0.726	2(C8)
tricore/mem_prot_example	162	6	1.853	5(C2)	N/A	-
s12g/ porting_example	720	2	2.991	2(C2)	0.629	-
cortex_mx/ASGPT1	678	2	3.153	2(C2)	2.178	1(C8)

PIT_HANDLER를 pit_period 주기로 실행하도록 하였다 (줄 09). 주기를 나타내는 단위는 PIT에서는 Hz이고 Alarm에서는 Tick이므로 scale 함수를 통하여 두 단위를 맞추어 주었다. (줄 06~07)

4. 실험

소스코드 변환작업을 마친 소스코드를 환경설정과 함께 선행연구 [6]에서 개발된 검증도구에 입력하였다. 실험에 사용된 어플리케이션은 Erika에서 제공하는 예제 어플리케이션 8개를 임의 선별하여 사용하였다. 각 프로그램에 표 1에 제시된 8개의 제약사항을 각각 지역제약사항 및 전역제약사항 검증기를 통해 검증하였다. 모든 실험은 3.5-GHz Intel Xeon E3-1280 CPU와 32 GB RAM을 가진 Fedora Linux 기반 컴퓨터에서 실행되었다.

표 2는 실험에 소요된 시간과 탐지한 제약사항 위반을 보여준다. 첫째 열에는 검증에 사용된 어플리케이션의 하드웨어 아키텍처와 어플리케이션의 이름을 보여주며, 둘째 열과 셋째 열에서는 어플리케이션의 길이와 Task 수를, 넷째 열과 다섯째 열에서는 지역 제약사항 검증 수행 시간과 발견한 제약사항 위반을 보여주며, 여섯 째 열과 일곱 째 열에서는 전역 제약사항 검증 수행 소요된 시간과 탐색한 제약사항 위반을 보여준다.

검증 결과 대부분의 검증은 짧은 시간안에 종료됨을 확인하였으며, 다수의 어플리케이션이 지역제약사항 C2와 전역제약사항 C8을 위반 함을 확인하였다. 수작업으로 확인 결과 발생한 제약사항 위반들은 어플리케이션의 각 Task가 종료될 때 TerminateTask를 호출하지 않음으로부터 발생함을 확인하였다. OSEK/VDX에 따르면 TerminateTask를 호출하지 않고 Task를 종료하는 것은 System을 정의되지 않은 상태로 이끌 수 있음을 명시하였다.

4.1 Erika 운영체제에서의 확인

Erika 운영체제 버전 2.7의 소스코드를 대상으로 TerminateTask를 호출하지 않고 Task를 종료하는 것이 어떤 영향을 미치는지 확인해보았다. 확인 결과, TerminateTask를 호출하지 않더라도 내부적으로 Task의 exit point에 도달할 경우 TerminateTask 호출했을 때와 같이 처리하였음을 확인하였으며, 이 경우 실험 대상 프로그램들은 Erika 운영체제에서 안전하게 작동할 수 있음을 확인하였다.

5. 결론 및 향후 연구

본 연구에서는 API 호출 검증을 Erika 어플리케이션에 적용해 보았다. 적용 결과 다수의 제약사항 위반이 발견되었지만 이들은 Erika 운영체제에서 안전하게 작동할 수 있음을 확인하였다. 하지만 표준을 위반하는 것은 안전하지 않으므로 이를 준수하여 어플리케이션이 개발되어야 한다.

향후 연구에서는 다양한 IoT 운영체제로 범위를 넓혀 실험할 예정이며 이를 위해, API 호출 검증 도구가 다양한 운영체제와 다양한 스케줄링 방식 그리고 다양한 제약사항을 검증할 수 있도록 개선할 예정이며, 실험에서는 하드웨어 의존 코드 추상화로 인하여 오탐이 발생하지는 않았지만 오탐이 발생할 여지가 있으므로 이를 위한 연구를 수행할 예정이다.

참고 문헌

[1]. 한국산업연구원, “산업 패러다임에 따른 미래 제조업의 발전전략”.

[2]. Erika Operating system, http://erika.tuxfamily.org

[3]. 한국전기산업연구원, “사물인터넷(IoT: Internet of Things)의 적용 사례 및 시사점”.

[4]. Y. Choi, “A configurable V&V framework using formal behavior patterns for OSEK/VDX operating systems,” *Journal of Systems and Software*, 2017.

[5]. O. Group, “OSEK/VDX operating system specificaiton,” 2005.

[6]. D.Kim Y. Chung Y. Choi, “Model-based API-call constraint checking for automotive control software,” *2016 Asia-Pacific Software Engineering Conference*, 2016.

A Systematic Review on Variability Modeling in Software Product Line

Aman Jaffari, Jihyun Lee, Jun Hyuk Jo, and Cheol-Jung Yoo

Dept. of Software Engineering, Chonbuk National University

{aman, jihyun30, bionicle73, cjyoo}@jbnu.ac.kr

Abstract

Variability modeling is the core concept characterizing software product line engineering. There are different ways of modeling variability with various modeling techniques. Hence, there is a need to shed light on the current status and direction of the existing research techniques. The objective of this paper is to investigate the status of the existing research, in addition to, identifying hot spots for the future research direction in the area. We performed a systematic review of 43 studies out of 543 studies published between the years 2004 to 2017. We used different screening strategies to obtain the most relevant studies. The findings indicate that the majority of the studies used modeling variability in a separate model, and feature model is the most commonly adapted techniques for variability modeling. The result also indicates that there is a lack of real-world industrial tools and cases, which confines the generalizability of the proposed approaches.

1. Introduction

Product line variability that is documented in so-called variability models is one of the key properties characterizing software product line engineering [1, 2]. Variability can be modeled either as an integral part of the development artifacts or in a separate variability model. There are alternative units of variability in the literature for representing variability models. Among them, feature model, decision model, and OVM (orthogonal variability model) are the most commonly adopted. Modeling variability in SPL is a complex task which requires a good decision about how to select an appropriate technique and tools. Moreover, software products evolve constantly, thus handling the changes in variability model and its evolution is an even more challenging task [3]. Over the years, numerous variability modeling approaches have been proposed. Hence, the SPL practitioners and researchers are always overwhelmed with the massive amount of information. There is a need to shed light on the current status and direction of the research techniques for modeling variability in SPL.

The object of this research is to provide an overview of the status of the existing techniques for modeling variability in SPL. Also, the study aims to identify the most promising hot spots for the future research directions. The result of the study indicates that separate variability modeling along with modeling variability in a single feature model has been used more frequently. Also, more than a half of the studies adopted feature models as units of variability. The result also shows that there is a lack of real-world industrial case studies and tools.

The rest of the paper is organized as follows. Section 2 presents the background of the study and discusses the related works. Section 3 describes the method for our systematic review. The result of the study is presented in Section 4. A brief discussion on our finding is presented in Section 5. And finally, Section 6 concludes the study and presents the future work.

2. Background and Related Work

As, the importance of documenting and managing variability increase in SPL, the research communities are continuously in quest of finding a better solution for modeling variability. Over the last decade, researchers have been proposed a great number of techniques and tools for modeling variability. Some of which are the extension or modification of the existing works while many others introduce or develop their own concepts [4]. There are numerous studies in this area [5, 6, 7, 8, and 9] we omit discussing on each of them due to space limitation. The two most relevant papers to this study are [10], an investigation on classifying variability modeling techniques, and [11] a survey on industrial variability modeling techniques. The first one provides an overview of the variability modeling techniques using a single example to exemplify the concepts of all techniques. The study focuses on identifying the common characteristics of variability modeling techniques. The second one emphasizes on understanding the use of variability modeling in practice and the associated benefits.

The SPL engineering, a broad concept, encompasses many topics and requires constant research on each specific area for discovering new techniques and tools or improving the existing ones. Yet, the relevant studies date back to an average of more

than six to seven years. And so, they are missing most of the recent variability modeling concepts and trends in the research area. In this study, we provide the most up-to-date overview of the status of the existing techniques for modeling variability along with the opportunities for the future research direction.

3. Systematic Mapping Method

A systematic literature review is an exhaustive and systematic method that can be conducted to get an overview of the topic in the area, as well as, synthesize all the existing research related to a specific research question. There are many guidelines for performing systematic reviews. We performed similar steps to a guideline provided by [12].

3.1 Research Questions

This study provides an overview of the status of the existing techniques for modeling variability. To specify our main goal we define the following research questions:

- **RQ1:** Which variability modeling approaches are proposed in SPL?
- **RQ2:** How do the variability modeling approaches in SPL handle the complexity issues?
- **RQ3:** How do the variability modeling approaches in SPL handle the variability model evolution?
- **RQ4:** What is the strength of the evidence in support of the proposed variability modeling approaches?
- **RQ5:** Do the variability modeling approaches in SPL support tools?

3.2 Search and Screening Strategies

The search strategy is performed to extract the most relevant studies out of the selected databases. Similarly, the screening strategy is also applied to exclude the irrelevant studies. As shown in Table 1 we have performed the search in the most commonly used scientific databases. IEEE, another scientific database is missing from the scope of the current study but will be included in our future study.

Table 1. Search string in each database

Databases	*Search String*	*Search results*
ACM	(“variability model” “feature model” “decision model”)	368
ScienceDirect	((“variability model” or “feature model” or “decision model”) and (“product line” or “product lines” or “product family”))	92
Springer	“variability model”, “feature model”, “decision model”	43

We have obtained a total of 543 publications and selected 43 most relevant articles based on the inclusion and exclusion criteria. Due to space limitation, information for the selected publications is omitted. However, an online list including the title, author, publisher and the year of publication is made available at the following link (selected publications). In order to assure that the most relevant papers in the scope of modeling variability in SPL are detected, we have defined a number screening steps for inclusion and exclusion activities as follows: Search each databases using the defined search strings; Exclude the papers that lie outside the variability modeling; Read the titles and abstracts for the additional exclusion; Group together multiple versions of the same studies; and Read the full-text carefully and further exclude the papers that have no contribution.

4. The Results

4.1 The proposed approaches (RQ1)

The pie chart in Figure 1 depicts the distribution of the variability model either presented as an integral part of development artifact or in a separate variability model. When it comes to the units of variability as depicted in Figure 2, more than a half of the studies preferred feature models and features as the units of variability. The 19 percent introduced their own concepts and terminology for expressing the units of variabilities such as clafer [13] and some UML based feature modeling [14].

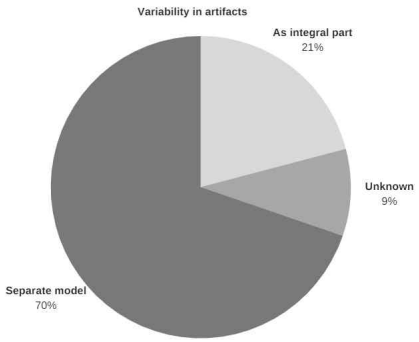


Figure 1. Distribution of the studies by adopting variability method

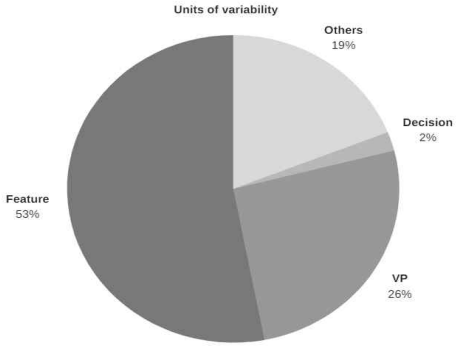


Figure 2. Distribution of studies by units of variability

The column and line graph illustrated in the top compartment of Figure 3 shows the distribution of studies addressing the variability in artifacts (e.g. modeling variability as integral part of development artifacts or vice versa) by publication years. The bottom compartment shows the model multiplicity (e.g. modeling variability in a single feature model or multiple) by publication years.

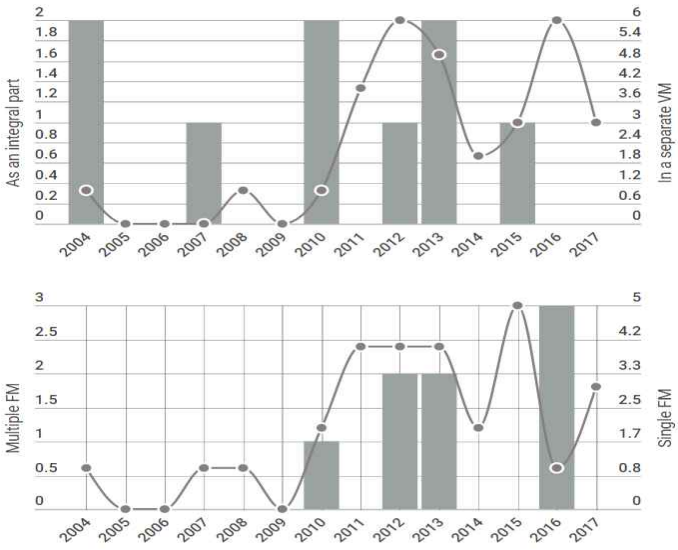


Figure 3. Distribution of studies by different modeling techniques and publication years

Figure 4 illustrates that most of the researchers adopted single feature modeling technique over multiple feature modeling.

4.2 Handling complexity (RQ2)

We examine handling the complexity of variability model from three perspectives: hierarchical model decomposition, modularity, and viewpoints. Compared to modularity and viewpoints, the hierarchical model decomposition is adopted as a more common approach.

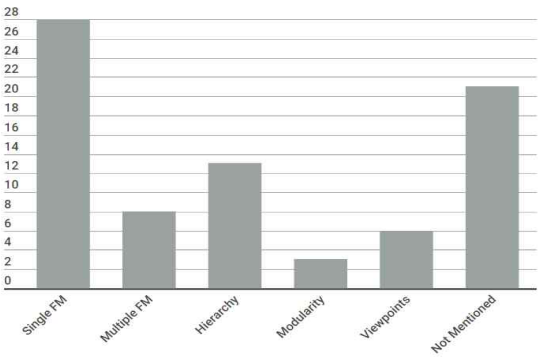


Figure 4. Distribution of studies by handling complexity including model multiplicity

4.3 Variability model evolution (RQ3)

We have investigated the variability model evolution from two perspectives, consistency check and conflicts resolution. A handful of the most remarkable solutions we found in the literature are: first, handling variability model evolution together with related artifacts, by inspecting a sample of the evolution history [15]. The main idea is to analyze the usage frequency of each pattern, and how artifacts are affected when there is a change in the model. Second, handling the variability model evolution by enabling model composition with interfaces [16]. Similar to programming interfaces, a variability model interface is used for the composition of variability models to establish information hiding. Third, handling the variability model evolution by locking specific configurations [17]. The idea is to add temporal variability (feature) models to assess the impact on existing configurations.

4.4 The strength of the evidence and tool support (RQ 4-5)

As depicted in Figure 5, all the selected studies performed some sort of preliminary assessment either using illustrative, academic or industrial examples but mostly toy examples. Figure 5 also depicts that most of the approaches in the literature did not provide a realistic tool for variability model formation and manipulation assessment.

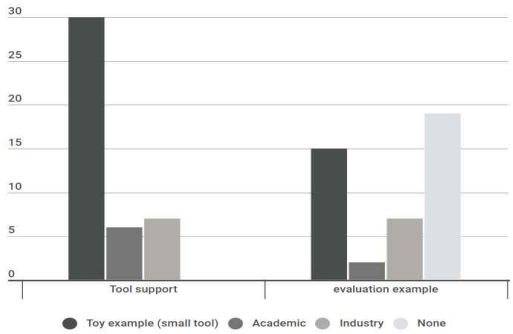


Figure 5. Distribution of papers by strength of the evidence and tool supports

5. Discussion

So far, this systematic review identified a number of approaches that are used for modeling variability in SPL. Recently, many studies have adopted separate variability modeling in contrast to modeling variability as integral part of development artifacts. Hierarchical model decomposition is used as a more common means of handling complexity. Also, the feature is used more commonly for representing the units of variability. In addition, single feature model is chosen over multiple feature models, due to the easiness of validating their proposed approaches. However, modeling an extremely large real-world SPL in a single feature model seems not to be realistic. Lately, BigLever software published its vision about feature models for mega-scale product line [18]. BigLever software proposes a hierarchical feature modeling concepts

composed of primitive standalone feature, feature model as a bundle of primitive standalone features, subsystem product lines as a bundle of feature models, and system product lines by bundling subsystem product lines.

A large portion of the studies in the literature never stated clearly how they handled the complexity. The variability model evolution is repeatedly urged as an important issue. Despite its importance, interestingly, only a few studies have taken it into account as their major challenges. Specifically, we summarize the following issues: Lack of robust tooling support; Lack of commonly accepted mechanism for handling variability model evolution; Great ambiguity on handling the complexity issues; Lack of real-word industrial examples.

6. Conclusion and future work

We have presented a systematic review of variability modeling in SPL. In this systematic review we have investigated a total of 43 articles out of 543 publications. We used different screening strategies to obtain the most relevant studies. Over the years, numerous variability modeling approaches have been proposed. Hence, the SPL practitioners and researchers are always overwhelmed with the massive amount of information. The main contribution of our study is providing the most needed up-to-date information on the status of the existing research for modeling variability in SPL. We hope this paper navigate the research community toward investigating on the right, as well as, the recent trend (discussed in the previous Section) on variability modeling techniques and tools.

Based on aforementioned issues we have discussed earlier several opportunities for the future research direction is highlighted. As part of our future work, we will further investigate on searching IEEE digital library to extend our systematic mapping study.

Acknowledgment. This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2017R1D1A3B03028609)

References

[1] K. Pohl, G. Bockle, F. V. D. Linden, "*Software product line engineering: foundations, principles, and techniques*", Springer, 2010.

[2] A. Metzger, and K. Pohl, "Software product line engineering and variability management: achievements and challenges", Future of software engineering (FOSE), ACM, 2014. pp. 70-84.

[3] S. Kang, Y. Han, H. Ahn, and J. Lee, "Critical analysis of feature model evolution", ICSESS, 2017.

[4] L. A. Zaid, F. Kleinermann, and O. D. Troyer, "Feature assembly: a new feature modeling technique", ER 2010, LNCS, Springer-Verlag Berlin Heidelberg, pp. 233-246.

[5] J. Simmonds, M. Bastarrica, L. Silvestre, and A. Quispe, "Analyzing methodologies and tools for specifying variability in software processes", Universidad de Chile, 2011.

[6] I. R-Berger, K. Figl, and Ø. Haugen, "Investigating styles in variability modeling: Hierarchical vs. constrained styles", Information and Software Technology, 2017, Vol. 87, pp. 81-102.

[7] J. F. Bastos, P. A. da. M. S Neto, E. S. de. Almeida, S. R. de. L. Meira, "Adopting software product lines: A systematic mapping study", 15th annual conference on evaluation & assessment in software engineering (EASE), IET, 2011.

[8] I. S. Santos, R. M. C. C. Andrade, P. A. S. Neto, "How to describe SPL variabilities in textual use cases: A systematic mapping study", 2014 eighth Brazilian symposium on software components, architectures and reuse, IEEE, 2014, pp. 64-73.

[9] R. E. L.Herrejon, S. Illescas, A. Egyed, "Visualization for software product lines: A systematic mapping study", IEEE working conference on software visualization (VISSOFT), IEEE, 2016, pp. 26-35.

[10] M. Sinnema and S. Deelstra, "Classifying variability modeling techniques", Information and Software Technology, ACM, 2007, pp. 717–739.

[11] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker and K. Czarnecki, "A survey of variability modeling in industrial practice", VaMoS'13, ACM 2013.

[12] K. Petersen, S. Vakkalanka, Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update", Information and software technology, ACM, 2015, vol. 64, pp. 1-18.

[13] K. Bak, Z. Diskin, M. Antkiewicz, K. Czarnecki, and A. Wasowski, "Clafer: unifying class and feature modeing", SoSyM, ACM 2016, pp. 811-845.

[14] L. A. Lanceloti, J. C. Maldonado, I. M. S. Gimenes, and E. A. O. Junior, "SMartyParser: A XMI parser for UML-based software product line variability models", VaMoS, ACM, 2013.

[15] L. Passos, J. Guo, L. Teixeira, K. Czarnecki, and A. Wasowski, "Coevolution of variability models and related artifacts: a case study from the Linux kernel", SPLC, ACM, 2013, pp. 91-100.

[16] R. Schroter, S. Krieter, T. Thum, F. Benduhn, and G. Saake, "Feature-model interfaces: the highway to compositional analyses of highly-configurable systems", ICSE, ACM, 2016, pp. 667-678.

[17] M. Nieke, C. Seidl, and S. Schuster, "Guaranteeing configuration validity in evolving software product lines", VaMoS, ACM, 2016, pp. 73-80.

[18] B. Young, J. Cheatwood, T. Peterson, R. Flores, and P. Clements, "Product line engineering meets model based engineering in the defense and automotive industries" SPLC, ACM, 2017, pp. 175-179.

데이터 기반 소프트웨어 신뢰성 모델의 입력 도메인 선정 기술의 필요성 연구

강상모 ○, 백종문
한국과학기술원 전산학부
ksm2456@kaist.ac.kr, jbaik@kaist.ac.kr

Study on the Necessity of Input Domain Selection Method in Data-Driven Software Reliability Model

Sangmo Kang○, Jongmoon Baik
School of Computing, KAIST

요 약

소프트웨어 신뢰성은 소프트웨어가 특정 환경에서 특정 시간 동안 실패 없이 실행될 확률로 중요한 소프트웨어 품질 속성 중 하나이고, 소프트웨어 신뢰성을 추정하기 위해 분석적 소프트웨어 신뢰성 모델이 수십 년 간 연구 되어 왔다. 하지만 분석적 소프트웨어 신뢰성 모델은 비현실적인 가정 및 모델 선택 문제 등의 제한 점이 있어 그를 극복하기 위해 데이터 기반 신뢰성 모델이 제안되었다. 소프트웨어 신뢰성 모델 중 데이터 기반 신뢰성 모델들은 일반적으로 t 시점의 누적 실패 수를 예측하기 위해 t−1의 누적 실패 수부터 t−d의 누적 실패 수 각각을 입력으로 넣는 지연된 다중 입력 단일 출력 구조를 가지는데, 이때 d의 크기를 입력 도메인이라고 한다. 기존 연구에서는 서로 다른 입력 도메인을 가지는 데이터 기반 신뢰성 모델 간의 예측 성능의 차이가 있으므로 예측 성능이 좋은 입력 도메인 선택 방법이 필요하다고 언급하였으나, 그 필요성을 실험을 통해 통계적으로 보이지는 않았다. 본 논문에서는 36개의 프로젝트로부터 수집한 실패 데이터를 사용한 실험을 통해 데이터 기반 소프트웨어 신뢰성 모델의 입력 도메인 선정 기술의 필요함을 보인다.

1. 서론

소프트웨어 신뢰성은 소프트웨어가 특정 환경에서 특정 시간 동안 실패 없이 실행될 확률이며 [1], 중요한 소프트웨어 품질 속성 중 하나이다 [2]. 소프트웨어 신뢰성 모델은 테스트 단계에서 소프트웨어 실패를 기반으로 미래의 소프트웨어 고장을 예측하기 위해 수십 년 동안 연구됐다. 소프트웨어 신뢰성 모델은 분석적 소프트웨어 신뢰성 모델(Analytical Software Reliability Model, ASRM)과 데이터 기반 소프트웨어 신뢰성 모델(Data-Driven Software Reliability Model, DDSRM)로 구분된다. ASRM은 소프트웨어 실패 프로세스를 고정된 수식을 이용하여 나타낸다 [1]. ASRM의 수식에는 해석할 수 있고 가정을 포함하는 매개 변수가 포함된다. 하지만 이 가정이 실제 프로젝트 특성에 맞지 않는 경우에 ASRM의 예측 성능이 감소하게 된다 [3, 4, 5].

반면에, DDSRM은 해석할 수 없고 가정을 포함하고 있지 않은 매개 변수를 가지고, 사용 가능한 오류 데이터를 기반으로 자체적으로 수식을 진화시킨다 [5]. DDSRM은 엄격한 가정이 없으므로 ASRM의 비현실적인 가정 문제를 해결할 수 있고, ASRM 보다 더 높은 예측 성능을 가지는 것으로 여겨진다 [6, 7]. DDSRM은 일반적으로 t 시점의 누적 실패 수를 예측하기 위해 t−1의 누적 실패 수부터 t−d의 누적 실패 수 각각을 입력으로 넣는 지연된 다중 입력 단일 출력(Multiple Delayed Input Single Output, MDISO) 구조를 가지고, d의 크기를 입력 도메인이라고 한다 [4].

기존 연구에서는 서로 다른 입력 도메인을 가지는 DDSRM 간의 예측 성능 차이가 있음을 보임으로 예측 성능이 좋은 입력 도메인을 선택할 방법이 필요하다고 언급하였다 [7, 8]. 그러나 그 필요성을 실험을 통해 보이지는 않았다.

본 연구는 DDSRM의 입력 도메인 선택 방법 연구를 하기 전, 실험을 통해 입력 도메인 선택 방법의 필요 여부를 확인하기 위한 선행 연구이다. 36개의 프로젝트로부터 수집한 실패 데이터를 사용한 실험을 통해 DDSRM의 입력 도메인 선정 기술의 필요함을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구에서 필요한 배경 지식인 DDSRM에 대해 설명한다. 3장에서는 입력 도메인 선택 방법의 필요 여부를 확인하기 위한 실험의 설계에 관해 설명한다. 4장에서는 실험 결과를 보여주고 분석한다. 그리고 마지막 5장에서는 결론에 대해 기술한다.

2. 배경 지식

최근, 소프트웨어 신뢰성 공학 분야에서 머신 러닝 기법을 사용한 DDSRM에 대해 많은 연구가 이뤄지고 있다 [4]. 특히, ε-서포트 벡터 회귀(ε-Support Vector Regression, ε-SVR)를 사용한 DDSRM이 다른 머신 러닝 기법을 사용한 DDSRM에 비해 우수한 성능을 보인다 [9, 10].

DDSRM은 일반적으로 MDISO 구조로 되어 있다. MDISO 구조를 가지는 DDSRM은 식 (1)로 표현된다. x_t 는 t 시점의 누적 실패 수를 의미하고, d 는 입력 도메인을 의미한다 [4].

$$x_n = \mathrm{F}(x_{n-1}, x_{n-2}, \ldots, x_{n-d}) \quad (1)$$

그림 1은 입력 도메인이 1일 때 DDSRM의 훈련 과정과 예측 과정을 예시로 보여준다. x_1 을 입력으로 x_2가 출력, x_2를 입력으로 x_3이 출력, …, 그리고 x_8을 입력으로 x_9 이 출력 되게 DDSRM을 훈련 시킨다. 훈련된 모델에 x_9를 입력으로 넣어 $\hat{x}_{10}$을 예측한다.

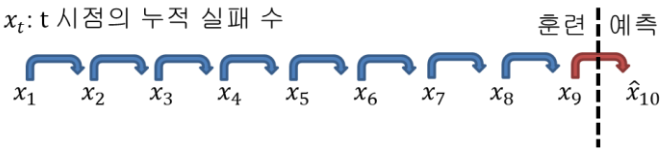


그림 1 입력 도메인이 1일 때의 DDSRM

그림 2는 입력 도메인이 3일 때 DDSRM의 훈련 과정과 예측 과정을 예시로 보여준다. x_1, x_2, x_3 을 입력으로 x_4가 출력, x_2, x_3, x_4를 입력으로 x_5가 출력, …, 그리고 x_6, x_7, x_8을 입력으로 x_9가 출력 되게 DDSRM을 훈련 시킨다. 훈련된 모델에 x_7, x_8, x_9를 입력으로 넣어 $\hat{x}_{10}$을 예측한다.

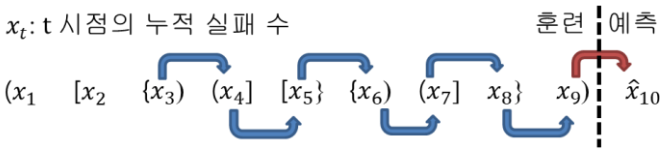


그림 2 입력 도메인이 3일 때의 DDSRM

3. 실험 설계

본 장은 아래의 연구 질문(Research Question, RQ)에 답하기 위한 실험 설계를 설명한다.

데이터 세트 별로 예측 성능이 좋은 입력 도메인을 사용한 DDSRM과 고정된 입력 도메인을 사용한 DDSRM 간의 예측 성능 차이가 있는가?

RQ는 프로젝트별로 예측 성능이 좋은 입력 도메인을 사용한 DDSRM과 고정된 입력 도메인을 사용한 DDSRM 간의 예측 성능의 차이가 통계적으로 의미가 있는지 확인한다. 통계적으로 의미가 있다면 예측 성능이 좋은 입력 도메인을 선정함으로써 통계적으로 유의미한 예측 성능 향상을 기대할 수 있다는 것을 의미하며, 입력 도메인 선정 기술이 필요하다고 할 수 있을 것이다.

실험에서는 기존 연구들에서 [7, 9, 10] 우수한 예측 성능을 보인 ε-SVR을 사용한 DDSRM을 사용한다. 실패 데이터 전처리 과정, 장기 예측 전략, 입력 도메인 크기의 상한 설정, ε-SVR의 매개 변수 선택 방법은 [7]의 연구에서 제안한 방법을 사용한다.

RQ에 대한 답을 하기 위해 36개의 프로젝트로부터 수집한 실패 데이터를 사용하였고, 표 1은 각 데이터 세트에 대한 설명을 보여준다. 이 데이터 세트들은 [11, 20]에서 실험을 위해 수집된 것들이다.

표 1 실험에 사용된 데이터 세트들

#	종류	실패 수	참고문헌
1	실시간 명령 및 제어	136	[12]
2		54	[12]
3		38	[12]
4		53	[12]
5	전자 시스템	461	[13]
6		211	[13]
7	항공 우주 시스템	133	[13]
8		224	[13]
9		351	[13]
10		188	[13]
11		367	[13]
12	상용 소프트웨어	73	[12]
13		100	[14]
14		120	[14]
15		61	[14]
16		42	[14]
17	군용 소프트웨어	38	[12]
18		41	[12]
19		101	[12]
20	무선 네트워크	22	[15]
21		181	[15]
22	의료 시스템	231	[16]
23		245	[16]
24		83	[16]
25	실시간 제어	535	[13]
26		198	[13]

27	데이터베이스 애플리케이션	328	[17]
28	오픈 소스	74	[18]
29		50	[18]
30		58	[18]
31		85	[18]
32		54	[18]
33		54	[18]
34	모니터링 및 실시간 제어	481	[19]
35		266	[19]
36		86	[19]

예측 성능을 평가하기 위한 척도로 [21]가 제안한 예측의 평균 오차(Mean Error of Prediction, MEOP)를 사용한다. MEOP는 식 (2)로 표현된다. x_t 는 t 시점의 누적 실패 수, $\hat{x}_t$는 t 시점의 추정된 누적 실패 수, k는 예측이 시작되는 시점, e 는 예측이 끝나는 시점을 의미한다. MEOP 값이 낮을수록 예측이 잘된 것을 의미한다.

$$\mathrm{MEOP} = \frac{\sum_{i=k}^{e} |x_i - \hat{x}_i|}{e-k+1} \quad (2)$$

입력 도메인은 [7, 8]의 연구를 참고하여 1~5를 사용한다. 각 데이터 세트들을 앞에서부터 60%, 70%, 80%, 90%에서 나누어 앞부분은 DDSRM을 훈련 시키는데 사용하고, 뒷부분은 훈련 시킨 DDSRM의 예측 정확도를 평가하는 데 사용된다.

4. 실험 결과 및 분석

본 실험에서는 입력 도메인 1~5로 고정 하여 DDSRM을 훈련 시켰을 때와 데이터 세트 별로 예측 성능이 가장 좋은 입력 도메인을 선정할 수 있는 기술이 있다고 가정하고 각 데이터 세트의 가장 예측 성능이 좋은 입력 도메인을 선정하여 DDSRM을 훈련 시켰을 때를 비교한다.

표 2는 각 데이터 세트 별로 가장 예측 성능이 가장 좋은 입력 도메인을 사용했을 때(1st)와 입력 도메인 1~5를 고정하여 사용하였을 때(ID 1~5)의 36개 데이터 세트들의 MEOP 평균을 나타낸다. 표 3은 각 기법을 사용하였을 때의 36개 데이터 세트들의 MEOP 중앙값을 나타낸다.

평균과 중앙값의 경우 각 데이터 세트 별로 가장 예측 성능이 좋은 입력 도메인을 사용했을 때가 가장 낮음은 당연하다. 하지만 그 차이가 통계적으로 유의미하지 않다면 입력 도메인 선정 기술을 연구할 필요가 없다고 할 수 있다. 각 기법별 MEOP들 사이의 통계적 차이를 확인하기 위해 Wilcoxon pair-wise 검정을 사용한다 [22].

표 4는 각 데이터 세트 별로 가장 예측 성능이 좋은 입력 도메인을 사용했을 때(1st)의 MEOP들과 입력 도메인을 고정하였을 때(ID 1~5)의 MEOP들이 같은 분포를 가지는 알아보기 위한 Wilcoxon pair-wise 검정을 실행하였을 때의 P-value를 보여준다. Wilcoxon pair-wise 검정의 귀무가설은 “두 집합이 같은 분포를 보인다.”이고, 대립가설은 “두 집합이 서로 다른 분포를 보인다.”이다.

유의 수준을 0.01로 보았을 때, 모든 경우에서 귀무가설이 기각되었으므로 각 데이터 세트 별로 가장 예측 성능이 좋은 입력 도메인을 사용했을 때(1st)의 MEOP들과 입력 도메인을 고정하였을 때(ID 1~5)의 MEOP들은 서로 다른 분포를 보인다고 볼 수 있다. 즉, 예측 성능이 좋은 입력 도메인을 선택함으로써 통계적으로 유의미한 예측 성능 향상을 기대할 수 있고, 입력 도메인 선정 기술이 필요하다고 할 수 있을 것이다.

표 2 MEOP 평균

훈련 비율	ID 1	ID 2	ID 3	ID 4	ID 5	1st
60%	18.16	22.08	22.89	21.99	17.73	9.59
70%	14.17	19.47	14.26	13.52	13.69	6.94
80%	9.91	8.82	5.71	7.13	7.66	3.54
90%	7.00	4.74	3.52	3.90	3.97	1.99

표 3 MEOP 중앙값

훈련 비율	ID 1	ID 2	ID 3	ID 4	ID 5	1st
60%	8.98	11.62	10.83	10.60	9.06	6.08
70%	5.95	6.43	6.55	6.69	7.26	2.74
80%	2.99	3.69	3.09	3.21	4.33	2.10
90%	1.32	1.59	2.54	2.18	1.91	0.75

표 4 Wilcoxon pair-wise test의 P-value

훈련 비율	1st VS				
	ID 1	ID 2	ID 3	ID 4	ID 5
60%	0.0000056	0.0000010	0.0000002	0.0000009	0.0000011
70%	0.0000007	0.0000007	0.0000003	0.0000005	0.0000025
80%	0.0000023	0.0000004	0.0000003	0.0000007	0.0000008
90%	0.0000010	0.0000023	0.0000002	0.0000003	0.0000010

5. 결론

기존 연구에서는 서로 다른 입력 도메인을 가지는 DDSRM 간의 예측 성능 차이가 있음을 보임으로 예측

성능이 좋은 입력 도메인을 선택할 방법이 필요하다고 하였으나, 그 필요성을 실험을 통해 보이지는 않았다. 본 연구는 36개의 프로젝트로부터 수집한 실패 데이터를 사용하여, 입력 도메인을 고정하였을 때와 가장 예측 성능이 좋은 입력 도메인을 선택하였을 때의 성능 차이를 통계적으로 비교하여 DDSRM의 입력 도메인 선정 기술의 필요함을 보였다.

본 연구는 향후 입력 도메인 선정 기술을 연구할 때 연구 동기를 강화해줄 참고문헌으로 써 사용될 수 있을 것이다.

Acknowledgement

이 논문은 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2016R1D1A1A09917660).

이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No. 2017M3C4A7066212).

참고문헌

[1] IEEE Recommended Practice on Software Reliability, IEEE 1633:2016, 2016.

[2] M. R. Lyu, "Software Reliability Engineering: A Roadmap," Future of Software Engineering, 2007. FOSE '07, Minneapolis, MN, 2007, pp. 153-170.

[3] J. H. Lo, "A data-driven model for software reliability prediction," 2012 IEEE International Conference on Granular Computing, Hangzhou, China, 2012, pp. 326-331.

[4] B. Yang, X. Li, M. Xie, and F. Tan, "A generic data-driven software reliability model with model mining technique," Reliab. Eng. Syst. Saf., vol. 95, no. 6, pp. 671-678, 2010.

[5] E. O. Costa, a. Pozo, and S. R. Vergilio, "A Genetic Programming Approach for Software Reliability Modeling," IEEE Trans. Reliab., vol. 59, no. 1, pp. 222-230, 2010.

[6] A. Jaiswal and R. Malhotra, "Software reliability prediction using machine learning techniques,"Int. J. Syst. Assur. Eng. Manag., 2016.

[7] J. Park, N. Lee, and J. Baik, "On the long-term predictive capability of data-driven software reliability model: An empirical evaluation," Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE, pp. 45-54, 2014.

[8] N. Raj Kiran and V. Ravi, "Software reliability prediction by soft computing techniques," J. Syst. Softw., vol. 81, no. 4, pp. 576-583, 2008.

[9] P. Kumar and Y. Singh, "An empirical study of software reliability prediction using machine learning techniques," Int. J. Syst. Assur. Eng. Manag., vol. 3, no. 3, pp. 194-208, 2012.

[10] M. das C. Moura, E. Zio, I. D. Lins, and E. Droguett, "Failure and reliability prediction by support vector machines regression of time series data," Reliab. Eng. Syst. Saf., vol. 96, no. 11, pp. 1527-1534, 2011.

[11] J. Park, An Effective Model Combination Approach based on Adaptive Criteria Tree for Software Reliability Prediction, Ph.D. Dissertation, Korea Adv. Inst. Science, Techn., Daejeon, Republic of Korea, 2016.

[12] J. Musa, A. Iannino, and K. Okumoto, Software Reliability: Measurement, Prediction, Application. 1987. New York, 1987.

[13] M. R. Lyu, Ed., Handbook of software reliability engineering. Los Alamitos, CA: IEEE Computer Society Press, 1996.

[14] A. Wood, "Predicting software reliability," Computer (Long. Beach. Calif)., vol. 29, no. 11, pp. 69-77, 1996.

[15] D. Jeske, X. Zhang, and L. Pham, "Adjusting software failure rates that are estimated from test data," Reliab. IEEE Trans., vol. 54, no. 1, pp. 107-114, 2005.

[16] C. Stringfellow and A. Andrews, "An empirical method for selecting software reliability growth models," Empir. Softw. Eng., vol. 7, no. 4, pp. 319-343, 2002.

[17] J.-H. Lo and C.-Y. Huang, "An integration of fault detection and correction processes in software reliability analysis," J. Syst. Softw., vol. 79, no. 9, pp. 1312-1323, 2006.

[18] X. Li, Y. F. Li, M. Xie, and S. H. Ng, "Reliability analysis and optimal version-updating for open source software," Inf. Softw. Technol., vol. 53, no. 9, pp. 929-936, Sep. 2011.

[19] Y. Tohma and R. Jacoby, "Hyper-geometric distribution model to estimate the number of residual software faults," in Proceedings of the 13th Annual International Computer Software and Applications Conference, 1989, pp. 610-617.

[20] J. Park and J. Baik, "Improving software reliability prediction through multi-criteria based dynamic model selection and combination," J. Syst. Softw., vol. 101, pp. 236-244, 2015.

[21] M. Zhao and M. Xie, "On the log-power NHPP software reliability model," Proc. Third Int. Symp. Softw. Reliab. Eng., pp. 14-22, 1992.

[22] J. Demšar, "Statistical Comparisons of Classifiers over Multiple Data Sets," J. Mach. Learn. Res., vol. 7, pp. 1-30, 2006.

클래스 변경 정보에 기반한 마이크로서비스 단위의 애플리케이션 구성 방법

김 대 호°*, 김 웅 수*, 윤 동 규*, 박 준 석**, 염 근 혁§교신저자
부산대학교 전기전자컴퓨터공학과*§, 부산대학교 물류혁신네트워킹연구소**
{ suitable, kus9010, lodestar692, pjs50, yeom}@pusan.ac.kr

Method to Develop Microservice Based Applications Based on Class Change Information

Daeho Kim°*, Ungsoo Kim*, Donggyu Yun*, Joonseok Park**, Keunhyuk Yeom§corresponding author
Department of Electrical and Computer Engineering Pusan National University*§,
Research Institute of Logistics Innovation and Networking**

요 약

최근 마이크로서비스를 기반으로 애플리케이션의 개발을 지원하는 서버리스 컴퓨팅이 각광받고 있다. 마이크로서비스는 독립적으로 동작 가능한 작은 규모의 서비스를 말하며 개별적인 업데이트와 쉽고 빠른 개발 및 배포가 가능하다는 이점을 가지고 있어 많은 기업들이 기존의 모놀리식 애플리케이션을 마이크로서비스 환경으로 전환을 시도하고 있다. 그러나 마이크로서비스의 구성에 대한 방법 및 기준과 이를 지원하는 연구가 부족한 실정이다. 본 논문에서는 모놀리식 애플리케이션을 마이크로서비스 단위로 구성하기 위한 방법을 제시한다. 모놀리식 애플리케이션의 클래스 정보와 개발 변경 기록을 분석하여 계층화, 그래프화, 마이크로서비스 구성의 3단계를 통해 마이크로서비스 단위로 애플리케이션을 구성한다. 제시된 방법은 모놀리식 애플리케이션을 마이크로서비스 단위로 구성하는 기본 모델로 적용될 것으로 기대된다.

1. 서 론

서버리스 컴퓨팅이란 서버를 관리할 필요가 없이 마이크로서비스 단위로 애플리케이션을 구성하며 지속적인 개발과 배포를 지원하는 새로운 형태의 개발 패러다임이다. 서버리스의 핵심 기술인 마이크로서비스는 독립적으로 동작 가능한 작은 규모의 서비스를 말한다[1]. 다른 마이크로서비스의 내부 구조를 알지 못해도 독립적인 업데이트가 가능하며 이는 기존의 모놀리식 애플리케이션 개발 방식 보다 매우 확장성이 높고 빠른 개발, 배포가 가능하다는 장점이 있다. 시장 조사 기관인 가트너는 향후 5년 이내 세계 IT 기술을 주도할 트렌드 중 하나로 서버리스 컴퓨팅을 선정하였다[2].

최근 많은 기업들이 서버리스 컴퓨팅 환경으로의 전환을 시도하고 있으나 큰 규모의 모놀리식 애플리케이션을 마이크로서비스 단위로 구성하기 위한 방법 및 기준과 이를 지원하는 연구가 부족한 실정이다[3].

따라서 본 논문에서는 모놀리식 애플리케이션을 마이크로서비스 단위로 구성할 수 있는 방법을 제시한다. 모놀리식 애플리케이션의 클래스 정보와 개발 변경 기록을 분석하여 계층화, 그래프화, 마이크로서비스 구성의 3단계를 통해 마이크로서비스 단위로 애플리케이션을 구성한다. 제시된 방법은 모놀리식 애플리케이션을 마이크로서비스 단위로 구성하는 기본 모델로 적용될 것으로 기대된다.

2. 관련 연구

2.1 마이크로서비스

F. Rademacher 등의 연구[4]에서는 마이크로서비스는 기존의 SOA(Service-Oriented Architecture)와 비교하여 보다 독립적인 형태로 구성되어 있다고 정의하였다. 또한 각 마이크로서비스 사이의 결합도가 낮고 Interface가 매우 추상화 되어있음을 제시하였다.

M. Fowler[5]에 따르면, 마이크로서비스에서 데이터베이스는 하나의 데이터베이스가 전체 시스템에서 공유되는 것이 아닌 여러 개의 분산된 데이터베이스로 구성되어 있고 각 데이터베이스가 필요에 따라 업데이트될 수 있는 구조라고 제시하였다.

Y. Yu 등의 연구[6]에서는 엔터프라이즈 환경에서 마이크로서비스를 구성하기 위한 참조 아키텍처 모델을 제시하였다. 이 모델은 마이크로서비스 간 통신을 위한

"이 논문은 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. NRF-2017R1D1A1B03030243)"

API를 중심으로 엔터프라이즈 환경에서 마이크로서비스 환경 구성을 지원하는 모델로서 제시되었다. 그러나 마이크로서비스를 독립적으로 구성하기 위한 기준은 제시되지 않았다.

2.2 마이크로서비스 구성 방법

O. Mustafa 등의 연구[7]에서는 런타임 환경에서 특정 서비스에 대한 접근 빈도를 기준으로 마이크로서비스를 구성해야 한다고 정의하였다. 유사한 시간대에 접근이 자주 발생한 서비스를 마이크로서비스 단위로 분리하여 성능이 향상됨을 검증하였다. 하지만 이와 같은 방법은 런타임 상황에서만 고려되었다는 점에서 한계점이 있다.

D. Escobar 등의 연구[8]에서는 EJB(Enterprise Java Beans)를 기준으로 마이크로서비스를 구성하는 방법을 제시하였다. EJB가 entity에 접근하는 각 경로가 하나의 마이크로서비스 단위로 구성될 수 있음을 보였다. 하지만 이는 EJB를 사용하는 시스템에만 적용 가능하다는 한계점이 있다.

G. Mazlami 등의 연구[9]에서는 클래스 정보를 분석하여 마이크로서비스 단위로 구성하는 방법을 제시하였다. 클래스에 대응하는 정점과 가중치를 가진 선으로 구성된 그래프에서 가중치가 낮은 선을 제거하여 마이크로서비스를 구성한다. 본 연구에서는 모놀리식 애플리케이션의 클래스 유형에 따라 계층을 세분화한 뒤 클래스 변경 정보를 분석하여 그래프를 구성한다는 점에서 차별성이 있다.

3. 클래스 변경 정보를 이용한 마이크로서비스 단위 구성 방법

본 장에서는 모놀리식 애플리케이션을 마이크로서비스 단위로 구성하는 방법을 설명한다. 우선, 구성에 필요한 모놀리식 애플리케이션 클래스 명세 요소들을 표 1과 같이 정의하였다. 또한 모놀리식 애플리케이션을 마이크로서비스 단위로 구성하기 위한 단계를 그림 1과 같이 *계층화, 그래프화, 마이크로서비스 단위 구성*의 3단계로 나눈다.

표 1 모놀리식 애플리케이션 클래스 명세 요소

요소	설명
Name	클래스 이름
Type	클래스 유형
Purpose of change	클래스 변경 목적
Development time	클래스 변경하는데 걸린 시간 정보
Contribution count	클래스 변경 횟수
Developer name	클래스에 접근한 개발자 이름

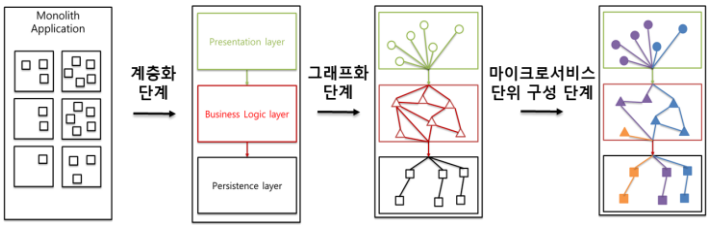


그림 1 마이크로서비스 단위 구성 단계

A) 계층화 단계

모놀리식 애플리케이션의 각 클래스를 유형에 따라 분류한다. 클래스의 유형은 입출력과 관련된 클래스, 애플리케이션의 주요 기능을 실현하는 클래스, 데이터베이스에 접근하는 클래스의 3가지 유형으로 분류한다.

계층화 단계에서는 각 클래스를 비즈니스 애플리케이션의 주요 계층화 방법인 3-Tier 구분 방법에 따라 각각 Presentation, Business Logic, Persistence Layer로 그룹화한다. 아래 그림 2는 계층화 단계를 나타낸 것이다.

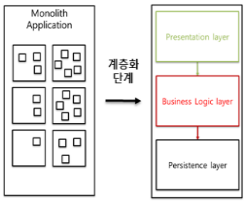


그림 2 계층화 단계

B) 그래프화 단계

클래스 유형별 계층화를 진행한 후 클래스 변경 정보를 분석하여 정점(Vertex)과 가중치 값을 가지는 선(Edge)으로 이루어진 그래프를 그림 3과 같이 구성한다. 정점은 각 클래스에 대응되며, 기본적으로는 1:1로 대응하지만 변경이 자주 발생하는 클래스의 경우 두 개 이상의 정점으로 분리하여 구성한다.

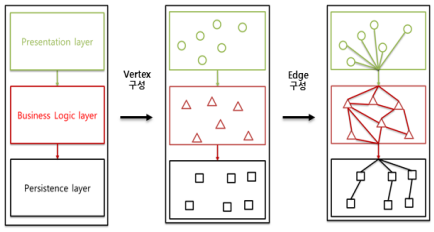


그림 3 그래프화 단계

선은 클래스 간의 연관 관계를 뜻하며, 선에 대한 가중치는 두 정점에 대응하는 클래스 사이의 변경 유사성을 비교하여 계산한다. 개발 변경 정보 중 변경이 동시에 발생한 클래스와 변경 목적이 유사한 클래스들을 비교하여 가중치 값을 구한다. 표 2는 클래스 변경 정보를 명세한 요소이다.

표 2 클래스 변경 정보 명세 요소

요소	설명
Commit Number	변경 번호
Commit Classes	동시에 변경한 클래스 정보
Commit Message	클래스 변경에 대한 목적 또는 메시지 정보

C) 마이크로서비스 구성 단계

그래프화 단계를 거쳐 구성된 그래프에서 정점 사이의 가중치가 낮은 선들을 그림 4와 같이 제거한 뒤 비즈니스 로직을 중심으로 정점을 묶는다. 이때 연관성이 높은 정점들로 구성된 경로를 구하기 위해 프림, 크루스칼, 다익스트라 알고리즘과 같은 경로 탐색 알고리즘을 적용한다. 그 결과로 나온 경로를 구성하는 정점들을 묶어 하나의 마이크로서비스로 구성한다.

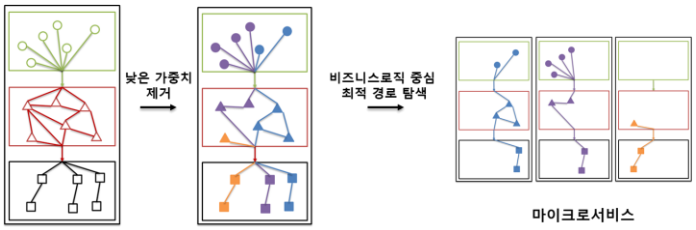


그림 4 마이크로서비스 구성 단계

4. 사례 연구

본 장에서는 3 장에서 제시한 방법을 이용하여 사례 연구를 진행한다. 사례연구를 진행할 모놀리식 애플리케이션은 의료 종합 시스템이다. 시스템의 UML Class Diagram과 Git, SVN을 통해 수집한 클래스 변경 정보를 그림 5와 같이 나타내었다.

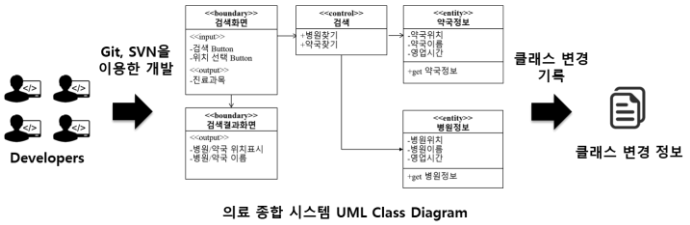


그림 5 UML Class Diagram 및 클래스 변경 정보

A) 계층화 단계 예시

클래스 유형에 따른 계층화를 진행한다. 분류 기준은 UML Class Diagram에서 정의한 클래스의 스테레오 타입이다. 예를 들어, Boundary 스테레오 타입의 경우 Presentation, Control 스테레오 타입의 경우 Business Logic, Entity 스테레오 타입의 경우 Persistence 계층으로 분류된다. 그림 6은 Class Diagram에서 기술된 스테레오 타입 기준으로 클래스들이 각 계층으로 분류된 것을 나타낸다.

Class Diagram의 Classes

Name <<stereo type>>			
약 정보 <<Entity>>	인증 Controller <<Control>>	이용자 정보 <<Entity>>	검색 Controller <<Control>>
병원 예약 Controller <<Control>>	로그인 Controller <<Control>>	개인정보 입력 화면 <<Boundary>>	중상 검색 Controller <<Control>>
회원 정보 <<Entity>>	로그인 화면 <<Boundary>>	저장 완료 화면 <<Boundary>>	질병 정보 <<Entity>>
병원 예약 정보 <<Entity>>>	예약완료 화면 <<Boundary>>	약 정보 조회 Controller <<Control>>	병원 정보 <<Entity>>
메인 화면 <<Boundary>>	병원 예약 화면 <<Boundary>>		

Presentation Layer

Name
메인 화면
로그인 화면
병원 예약 화면
예약완료 화면
개인정보 입력 화면
저장 완료 화면

Business Logic Layer

Name
검색 Controller
중상검색 Controller
약정보조회 Controller
인증 Controller
회원정보저장 Controller
로그인 Controller

Persistence Layer

Name
병원 예약 정보
이용자 정보
회원 정보
약 정보
질병 정보
병원 정보

그림 6 클래스 유형에 따른 분류

B) 그래프화 단계 예시

계층화를 진행한 후 클래스 변경 정보를 분석하여 그래프를 구성한다. 그래프는 정점과 선으로 구성된다. 정점은 하나의 클래스에 대응하여 구성된다. 만약 하나의 정점에 변경이 자주 발생하는 경우 정점을 두 개 이상으로 분리시킨다. 그림 7은 계층화 단계에서 분류된 각 클래스에 대응하는 정점을 구성하는 예시를 보여준다.

Boundary vertex(bv)

Name	vertex
메인 화면	bv1
로그인 화면	bv2
병원 예약 화면	bv3
예약완료 화면	bv4
개인정보 입력 화면	bv5
저장 완료 화면	bv6

Control vertex(cv)

Name	vertex
검색 Controller	cv1
중상검색 Controller	cv2
약정보조회 Controller	cv3
인증 Controller	cv4
회원정보저장 Controller	cv5
로그인 Controller	cv6

Entity vertex(ev)

Name	vertex
병원 예약 정보	ev1
이용자 정보	ev2
회원 정보	ev3
약 정보	ev4
질병 정보	ev5
병원 정보	ev6

그림 7 그래프 정점 구성

선의 가중치 값은 클래스 간 변경 유사성을 비교해 구한다. 그림 8은 개발자의 Commit Classes와 Commit Message를 기술한 정보다. Commit Classes는 하나의 Commit이 발생했을 때 동시에 변경된 클래스를 의미한다. Commit Message는 Commit의 목적이 무엇인지 기록한 내용을 의미한다. 두 요소를 분석하여 변경 유사성을 가진 정점 사이의 가중치를 계산한다. 본 예시에서는 변경 목적이 같으면서 동시에 발생한 Commit 하나당 1의 가중치를 추가한다.

예를 들어 cv1(검색 클래스)과 bv1(메인화면 클래스)이 4번 (C_2, C_6, C_7, C_{10}) 동시에 변경이 발생하였다. 하지만 C_2의 경우 Commit Message를 보면 cv1은 함수 구현 목적으로 변경하였고 bv1은 버그 해결 목적으로 변경하였으므로 목적이 서로 다르다는 것을 알 수 있다. 따라서 가중치는 3으로 계산된다. 그림 9는 이와 같은 방법으로 모든 정점 사이의 가중치를 나타낸 그림이다.

Commit	Commit Classes	Commit Message
C_1	bv15, cv1, ev6	Implement function complete() for Class bv15,cv1,ev6
C_2	bv1, cv1, bv13, ev6	Implement function searchHospital() for Class cv1, ev6 Fix Search bug for Class bv1, ev6
C_3	bv13, cv1, ev6	Implement function complete() for Class bv13,cv1,ev6
C_4	bv1, ev6	Fix Search bug for Class bv1, ev6
C_5	bv14, cv1	Change Parameter Value for Class bv14,cv1
C_6	bv1, bv13, bv14, cv1,ev1	Fix Search bug for Class **bv1,cv1**, bv13 Change Parameter Value for Class bv14, cv1, ev1
C_7	bv1,cv1	Change Parameter Value for Class bv1, cv1
C_8	bv13,bv14	UI part implementation for Class bv13, bv14
C_9	bv14	Fix SearchResult bug for Class bv14
C_{10}	bv1,bv15,cv1, ev5,ev6	Change Parameter Value for Class ev6 Implement function searchHospital() for Class bv1,bv15, cv1,ev5

그림 8 개발자 Commit 정보

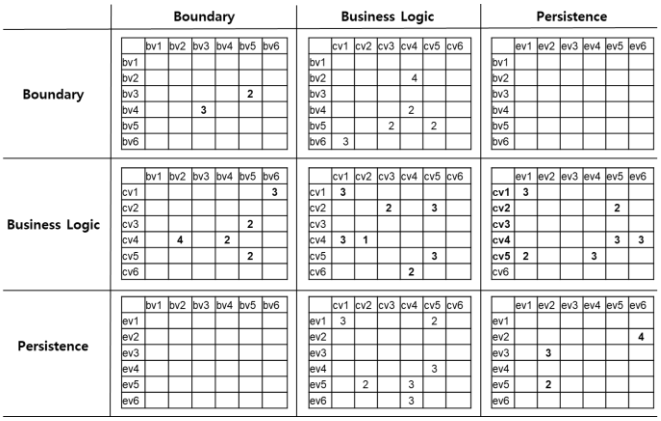


그림 9 정점 간 가중치

C) 마이크로서비스 구성 예시

구성된 그래프에서 그림 10과 같이 가중치가 낮은 선을 제거하고 비즈니스 로직 중심으로 연관성이 높은 정점들로 구성된 경로를 탐색한다. 결과로 도출된 경로를 구성하는 각 정점들을 묶어 하나의 마이크로서비스 단위로 구성한다.

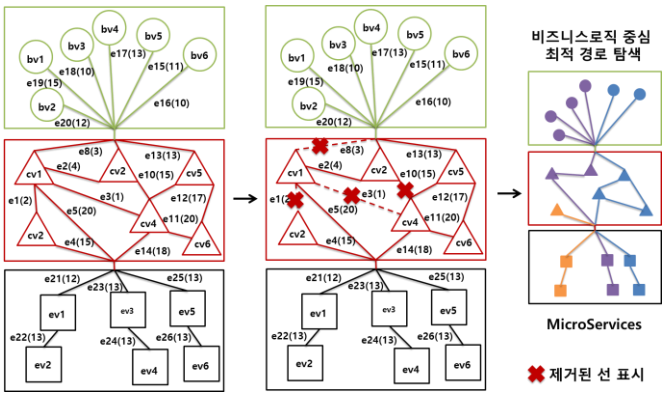


그림 10 마이크로서비스 단위 구성 예시

5. 결론 및 향후 연구

본 논문에서는 모놀리식 애플리케이션의 클래스 변경 정보를 분석하여 마이크로서비스 단위로 구성할 수 있는 방법을 제시하였다. 또한 사례 연구를 진행하여 의료 정보 시스템을 마이크로서비스 단위로 구성하는 과정을 제시하였다. 본 논문에서 제시한 방법은 다양한 형태의 모놀리식 애플리케이션을 마이크로서비스 기반의 서버리스 환경으로 전환할 수 있는 기반 모델이 될 것으로 기대된다. 향후 연구로는 구성된 마이크로서비스를 관리하는 마이크로서비스 저장소에 대한 연구를 진행할 것이다.

참고문헌

[1] J. Thönes, “Microservices”, IEEE Software, Vol. 32, Issue 1, pp. 116-116, Feb. 2015.

[2] Gartner, https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/

[3] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski and P. Suter, “Serverless Computing: Current Trends and Open Problems”, Cornell University, Jun. 2017.

[4] F. Rademacher, S. Sachweh and A. Zündorf, “Differences Between Model-driven Development of Service-oriented and Microservice Architecture”, 2017 IEEE International Conference on Software Architecture Workshops, pp. 38-45, Apr. 2017.

[5] Microservices, https://martinfowler.com/articles/microservices.html

[6] Y. Yu, H. Silveira and M. Sundaram. “A Microservice based Reference Architecture Model in the context of Enterprise Architecture”, IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference, pp. 1856-1860, Oct. 2016.

[7] O. Mustafa, and J. Marx Gómez, “Optimizing economics of microservices by planning for granularity level”, ProWeb 2017 Programming Technology for the Future Web Brussels Belgium, Apr. 2017.

[8] D. Escobar, D. Cárdenas, R. Amarillo, E. Castro, K. Garcés, C. Parra and R. Casallas, “Towards the understanding and evolution of monolithic applications as microservices” In Computing Conference (CLEI), pp. 1-11, Oct. 2016.

[9] G. Mazlami, J. Cito and P. Leitner, “ Extraction of Microservices from Monolithic Software Architecture, Web Services, 2017 IEEE International Conference on, Honolulu, HI, USA, Vol. 40, Issue 11, pp. 524-531, Sep. 2017.

오픈소스 기반 소프트웨어 프로세스 구축 자동화 개선 방안

이진협* 장우성* 박지훈* 박용범** 김영철*O

*홍익대학교 소프트웨어공학연구실 **단국대학교 정보아키텍처연구실

{ ljh* jang* pjh* bob*O}@selab.hongik.ac.kr **ybpark@dankook.ac.kr

Improving Automatic Construction of Software Process based on Open Source

Jin Hyub Lee* Woo Sung Jang* Ji Hoon Park* Yong B. Park** R. Young Chul Kim*O

*Software Engineering Lab, Hongik University

**Information Architecture Lab, Dankook University

요 약

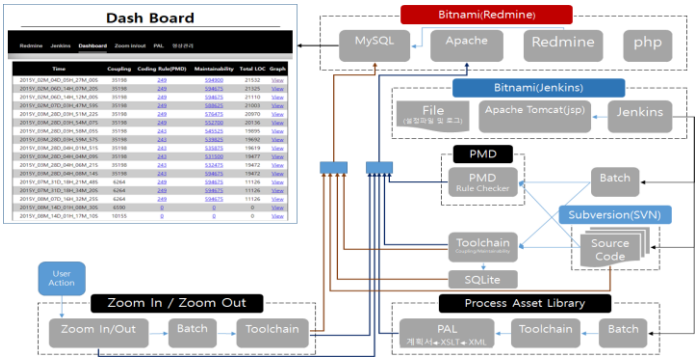
소프트웨어의 대형화와 개발 단계의 비가시성으로 인해 소프트웨어의 품질과 개발 프로세스 품질의 관리가 어렵다. 이를 위해, 가시화 소프트웨어 프로세스 구축 및 적용이 필수적이다. 하지만 국내 소프트웨어 벤처/스타트업/중소기업에서는 자체 소프트웨어 프로세스 구축이 미비하다. 또한 전문인력 부족과 시간 소모가 문제이다. 이를 해결하기 위해, 오픈소스 기반 프로젝트 일정관리, 이슈관리, 지속적 통합, 자동 빌드 등 구축의 자동화 개선을 제안한다. 이를 통해, 쉬운 소프트웨어 프로세스 구축으로 프로젝트 개발 시 내재화가 가능하다. 이는 소프트웨어 제품과 프로세스의 고품질화가 기대된다.

1. 서 론

오늘날 소프트웨어가 대형화되면서 요구사항, 분석, 설계, 구현, 테스팅 전단계에서 관리가 이루어지지 않으면, 이슈 발생 시 처리에 상당한 시간과 비용이 소모된다. 이를 해결하기 위해서는 소프트웨어 프로세스 가시화가 필요하고, 소프트웨어 프로세스 구축과 적용이 필수적이다. 하지만 국내 중소기업에서는 전문인력 부족과 비용문제로 쉽지 않다. 때문에 기존 연구에서 오픈소스 기반의 소프트웨어 프로세스 구축 자동화를 제안하였다[1,2]. 그러나 기존 연구에서는 구축 자동화에 Bitnami에서 제공하는 Redmine과 Jenkins를 활용하였다. 이는 개별적으로 구축하기에는 좋은 방안이지만, Redmine이나 Jenkins 대신 다른 도구를 사용하고자 하였을 때 Plug&Play가 불가능 하였다. 본 논문에서는 기존 소프트웨어 프로세스 구축 자동화를 개선하여 Redmine, Jenkins, Apache, Tomcat, MySQL 등의 오픈소스들을 체인화하고, 필요 시 다른 도구를 대체하여 사용할 수 있도록 Plug&Play가 가능한 구축 자동화를 구현하였다. 이를 활용하여 벤처/스타트업/중소기업에서는 30분 내에 소프트웨어 프로세스 구축이 가능해 프로젝트 기간 내 소모되는 많은 시간과 노력, 비용이 줄어든다. 또한 소프트웨어 제품의 고품질화와 프로세스 고품질화를 기대한다.

본 논문의 구성은 다음과 같다. 2장에서 기존 연구에 대해 언급하고, 3장에서 활용된 오픈소스 도구들과 개선된 소프트웨어 프로세스 구축 자동화에 대해 설명한다. 마지막 4장에서 결론 및 향후 연구를 언급한다.

2. 관련 연구



(그림 1) 기존 소프트웨어 프로세스 구축 자동화 구조

그림 1는 기존 소프트웨어 프로세스 구축 자동화 구조도이다. Redmine, Jenkins, SVN, PMD, Zoom In/Out, PAL이 자동 구축되었다[2,3]. 기존에는 Bitnami를 이용하여, 설치 단계가 짧다. 하지만 Redmine 내에 MySQL과 Apache, php가 모두 연결되어있어, 모든 DB와 파일들은 MySQL과

Apache에 있다. 또한 Jenkins 내에 Apache Tomcat이 있어 Apache 서버가 따로 운영되고, 결과적으로 Jenkins가 없으면 jsp 처리가 불가능한 구조이다. 즉, Plug&Play가 불가능하여, 개선이 필요했다.

3. 개선된 소프트웨어 프로세스 구축 자동화

<표 1> 구축 자동화에 활용된 오픈소스 도구들

도구	역할
Apache	웹 서버로, 웹 상에서 HTML 소스코드를 클라이언트에게 전송하는 역할 수행
PHP	Apache가 해석하지 못하는 php문장 해석, 모듈형태로 제공되어 Apache와 연동
Tomcat	기본적으로는 Apache Tomcat이지만, Tomcat 부분을 따로 Apache와 연동하여 JSP 해석하는 역할 수행
MySQL	Apache와 연동되고 Redmine 데이터 저장 역할 수행
phpMyAdmin	MySQL 데이터 관리를 용이하게 해주는 인터페이스 제공
Redmine	프로젝트/이슈 관리 도구로 자유로운 커스터마이징이 가능
Ruby	Redmine이 기본적으로 Ruby로 작성되어있어 레드마인 설치 시 필요
Jenkins	지속적 통합 도구로, 기본적으로 JSP로 작성되어있어 Tomcat이 필요, 1100 여 개 플러그인 제공
VisualSVN	SVN 서버 역할 수행, 수시로 commit 될 수 있어 독립적으로 사용
TortoiseSVN	SVN Client 역할 수행, 개발자가 Commit, Update 등을 쉽게 가능

표 1은 개선된 소프트웨어 프로세스 구축 자동화에 활용된 도구들과 그 역할들을 나타낸 것이다. Redmine 설치를 위해서는 기본적으로 Ruby, Devkit 설치가 필요하다. Jenkins 설치를 위해서는 Tomcat과 Java 해석을 위한 JDK가 필요하다.

기본적으로 Apache 서버와 MySQL 데이터베이스가 제공되고, Redmine, Jenkins, phpMyAdmin, VisualSVN이 Plug&Play의 형태로 제공된다. 이러한 구축을 위해서는 Apache 내에 php연동, Tomcat 연동, phpMyAdmin 연동이 필요하고, Redmine과 Jenkins를 주소창에서 port 입력없이 '~/redmine 또는 ~/jenkins' 형태로 연결하기 위해 VirtualHosts 설정을 해야한다. Bitnami Redmine이 아닌 본래의 Redmine 설치를 위해서는 Ruby와 Devkit 설치와 명령어 프롬프트에서 Gem, rails 명령어 입력을 해주어야한다. 개선된 소프트웨어 프로세스 구축 자동화에서는 이러한 과정들을 특별한 조치 없이 Next 클릭만으로 30분 내에 자동

```
PHPIniDir "C:/SWV/server/php"
LoadModule php7_module "C:/SWV/server/php/php7apache2_4.dll"
AddType application/x-httpd-php .html .php
AddHandler application/x-httpd-php .php

LoadModule jk_module modules/mod_jk.so

JkWorkersFile conf/workers.properties
JkLogFile "C:/SWV/server/Apache/logs/mod_jk.log"
JkLogLevel info
JkLogStampFormat "[%a %b %d %H:%M:%S %Y]"
JkRequestLogFormat "%w %V %T"

JkMount /* ajp13
JkMount /*.jsp ajp13
JkMount /*.do ajp13
JkMount /servlet/* ajp13
JkMount /url ajp13
JkMount /dita/j_spring_security* ajp13
JkMount /ppspwc/j_spring_security* ajp13
JkMount /mmea/j_spring_security* ajp13
JkMount /irtv/j_spring_security* ajp13
```

(그림 2) Apache 내 http.conf에 추가한 소스코드

설치된다.

그림 2는 Apache 내 http.conf 파일에 추가한 소스코드 일부이다. 윗부분은 Apache와 php 연동을 위한 소스코드이며, 아랫부분은 Apache와 Tomcat을 연동하는 소스코드로, mod_jk.so 파일이 필요하다. 이때, mod_jk.so 파일은 버전 호환이 되지 않으면 Apache 서비스가 실행되지 않는다.

4. 결론 및 향후 연구

소프트웨어 프로세스 가시화는 소프트웨어 개발 전단계의 고품질화를 위해 필요하다. 이를 위해 소프트웨어 프로세스 구축 및 적용이 필수적이지만, 중소기업에서는 쉽지 않은 과정이다. Redmine과 Jenkins는 구글링만으로 충분히 설치 가능하지만, 각 오픈소스 도구 간 버전 호환성으로 인해 많은 시행착오가 수반된다. 본 논문에서 구현한 개선된 소프트웨어 프로세스 구축 자동화를 활용하면 이러한 과정들을 버튼 클릭만으로 30분 이내에 구축 가능하고, 필요 시 Plug&Play도 가능하다. 이는 국내 소프트웨어 중소기업에서 소모하는 시간과 비용을 줄이고, 소프트웨어 제품과 프로세스 품질을 높일 것으로 기대한다.

향후 연구에서는 LDAP을 이용한 계정 통합관리와 통합 DB 구축을 통한 클로즈드 아키텍처 기반 추적성 적용을 연구할 예정이다[4].

ACKNOWLEDGEMENT

이 논문은 2015년 교육부와 한국연구재단의 지역혁신창의인력양성사업(NRF-2015H1C1A1035548)과 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2017R1D1A3B03035421).

참 고 문 헌

[1] 이진협, 손현승, 박지훈, 장우성, 김영철, "요구사항 추적성을 위한 소프트웨어 프로세스 가시화 구축 자동화", 2017 한국정보처리학회 추계학술발표대회 논문집, 제 24권 제 2호, p639-641, 2017

[2] 이진협, 장우성, 김영철, "자동 소프트웨어 프로세스 구축을 위한 공개 소프트웨어 툴 체인 연구", 2017년 한국소프트웨어종합학술대회 논문집, p554-556, 2017

[3] 장우성, 황준순, 김동호, 서채연, 김영철, 박병호, 이상은, 김영수, "프로세스 자산 라이브러리(PAL)위한 XML Data와 XSLT 기반 구축", 한국정보처리학회 2015년 추계학술발표대회 논문집, 제 22권 제 2호, p956-958, 2015

[4] 변은영, 손현승, 문소영, 박지훈, 김영철, "클로즈 아키텍쳐 메커니즘 기반의 요구사항 추적성 매트릭스", 2017 한국정보처리학회 추계학술발표대회 논문집, 제 24권 제 2호, p631-634, 2017

GitHub에서의 이슈 및 풀 요청 관련 활동이 소스 코드 품질에 미치는 영향 분석

조영준, 고인영
한국과학기술원
{yj_cho, iko}@kaist.ac.kr

Analyzing Effects of Issue and Pull Request Related Actions on Source Code Quality in GitHub

Yeongjun Cho, Inyoung Ko
KAIST

요 약

GitHub는 풀 기반 소프트웨어 개발 모델을 기반으로 개발자들이 오픈 소스 프로젝트에 자유로이 참여할 수 있는 플랫폼이다. 본 논문에서는 풀 기반 소프트웨어 개발 모델에서 사용되는 풀 요청 관련 활동들과 GitHub에서 제공하는 이슈 추적 시스템 관련 활동들이 프로젝트의 소스 코드 품질에 어떠한 영향을 끼치는지 분석하였다. GitHub에 호스팅 된 총 39개의 오픈 소스 프로젝트들로부터 수집한 5,642개월간의 지표 값들을 기반으로 다중 회귀 모델을 생성한 결과 풀 요청과 이슈의 생성 빈도와 세부 기능 사용 빈도가 유의미하게 소스 코드 품질에 영향을 미침을 확인하였다. 분석 결과에 기반하여 풀 기반 소프트웨어 개발자에게는 소프트웨어 품질을 위해 풀 요청과 이슈 관련 세부 기능을 적극적으로 사용할 것을, 연구자에게는 결함 예측 분야에서 풀 요청과 이슈의 이벤트를 기반으로 하는 지표 값을 사용할 것을 제안한다.

1. 서론

GitHub는 오픈 소스 프로젝트들의 소스 코드 저장소와 이슈 추적 시스템을 호스팅하고 있으며 개발자들이 소스 코드 변경이나 프로젝트에 대한 논의를 통해 오픈 소스 프로젝트에 자유로이 참여할 수 있는 플랫폼이다. 개발자들은 소스 코드 저장소를 통해 오픈 소스 프로젝트의 소스 코드를 살펴보거나 수정할 수 있으며 이슈 추적 시스템을 통해 프로젝트의 버그를 제보하거나 기능 요청을 요청하는 프로젝트에 대해 의견을 나눌 수 있다. 그리고 사용자들은 이슈 추적 시스템을 더욱 효율적으로 사용하기 위해 각 이슈별로 마일스톤 및 라벨 설정 등의 분류 기능이나 담당자 설정 등의 관리 기능, 구독 및 개발자 호출 등의 알림 기능 등 다양한 이슈 관련 세부 기능들을 제공한다.

GitHub에서는 개발자들에게 풀 기반 소프트웨어 개발 모델(Pull-Based Software Development Model)을 제공한다[1]. 풀 기반 소프트웨어 개발 모델에서 개발자가 소스 코드를 변경하고자 할 경우 소스 코드 저장소에 직접 소스 코드를 변경하는 것이 아니라 소스 코드 저장소를 새로이 복제한 후 복제된 저장소에서 코드를 변경해야 한다. 그 후 개발자는 기존 저장소에 풀 요청(Pull Request)을 하게 되며 프로젝트 관리자들은 풀 요청에 대한 소스 코드 검토를 진행한 후 풀 요청 수락을 결정한다. 풀 요청이 수락될 경우 복제된 저장소의 소스 코드 변경 내용이 프로젝트에 반영된다. GitHub에서는 풀 요청을 특수한 형태의 이슈로 취급하기 때문에 풀 요청 또한 다양한 세부 기능들을 사용할 수 있다.

이전에는 GitHub 상에서의 오픈 소스 프로젝트들에 대해 생성된 이슈 수와 프로젝트의 성공의 상관관계를 분석한 연구가 진행되었다[2]. 위 논문은 등록된 이슈 수와 프로젝트의 성공이 양의 관계가 있음을 보였으나 이슈 생성 외의 이슈와 관련된 세부 활동들은 고려하지 않았으며 프로젝트 소스 코드의 품질 측면에서는 논의가 이루어지지 않았다. 다른 연구에서는 이슈 추적 시스템에서의 사용자 간 논의 내용, 사용자 구조 등이 소프트웨어 품질에 끼치는 영향에 대해 분석을 하였으나 이슈 관련 세부 기능은 고려되지 않았다[3].

본 논문에서는 GitHub 상에서의 오픈 소스 프로젝트들에서 이슈 생성, 이슈에 대한 댓글 작성, 이슈 세부 기능 사용과 같이 이슈와 풀 요청에 대한 활동들이 소스 코드 품질에 미치는 영향을 다중 회귀 모델을 통해 분석하고 분석 결과를 활용할 수 있는 방안들을 제안한다.

2. 관련 연구

Vasilescu et al.[4]은 GitHub 상에서의 오픈 소스 프로젝트들에 대하여 지속적 통합 환경(Continous Integration)[5]이 프로젝트의 핵심 개발자 또는 외부 개발자의 생산성과 프로젝트 품질 관리 활동에 미치는 영향을 분석하였다. GitHub에서의 42개의 프로젝트별 매달 프로젝트의 특성이나 활동을 나타내는 지표 값들을 추출한 후 다중 회귀 모형을 생성하였다. 독립 변수로는 개발팀의 크기, 지속적 통합 환경 사용 여부, 소스 코드 라인 수, 테스트 코드 라인 수 등을 사용하였으며 종속 변수로는 생산성을 대표하는 지표 값으로 내부 및 외부 개발자의 풀 요청 수를, 품질 관리 활동을 대표하는 지표 값으로는 등록된 버그 관련 이슈 수를 사용하였다.

Jarczky et al.[6]은 GitHub 프로젝트의 속성과 프로젝트에 참여한 개발자들의 속성이 프로젝트 품질에 미치는 영향을 2,000여개의 프로젝트로부터 추출한 데이터를 기반으로 생성된 다중 회귀 모델을 통해 분석하였다. 위 논문에서는 프로젝트의 품질을 대표하는 지표 값으로 프로젝트의 인기도와 지원 요청에 대한 응답 기간을 제시하였다. 프로젝트의 속성으로는 프로젝트 생성 일자, GitHub 기능별 사용 여부, 등을 사용하였으며 개발자의 속성으로는 개발자 이름, 가입 일, 구직 여부 등을 사용하였다.

3. 데이터 수집

3.1 대상 프로젝트 및 수집 기간

본 연구에서는 Vasilescu et al.[4] 이 사용한 GitHub에 호스팅 된 오픈 소스 프로젝트 42개 중 39개의 프로젝트[1]에 대해 실험을 진행하였다. 이 프로젝트들은 최소 200개 이상의 풀 요청 활동이 있으며 GitHub에서 제공하는 이슈 추적 시스템을 사용하고 있기 때문에 실험 데이터로 사용하기 적합하다 판단하였다.

데이터 수집 기간의 경우 각 프로젝트별로 프로젝트 생성 시점부터 2017년 6월까지의 데이터를 수집하였다. 수집한 53,911 개의 버그 유발 변경 사항들에 대해 해당 변경 사항에서 유발한 버그를 수정하는 변경 사항 간의 일자 차이[7]를 조사 한 결과 75%의 버그 유발 변경 사항들이 180일 이내에 수정되었음을 확인하였다. 이에 따라 데이터 수집을 진행한 2017년 12월로부터 최근 180일 이내의 변경 사항들에 대해서는 버그 유발 여부가 제대로 분류되지 않을 가능성이 높아 2017년 6월까지의 데이터만을 사용하였다.

3.2 사용한 프로젝트 지표

본 연구에서는 각 프로젝트의 첫 변경 사항 등록 시간을 기준으로 하여 매 30일 주기로 프로젝트의 활동 및 특성을 나타내는 9개의 지표 값들을 총 5,642개를 추출하였다.

먼저 기본적인 프로젝트의 특성과 활동량을 나타내는 지표 값들을 선정하였다. 프로젝트가 생성된 후 추출 시점까지 경과된 개월 수를 나타내는 **age**, 해당 프로젝트에 단위 기간 내 발생한 소스 코드 변경 사항의 수를 나타내는 **commits** 지표 값이 이에 해당한다. 그리고 본 연구에서 분석하고자 하는 이슈 및 풀 요청 관련 활동들을 나타내는 지표 값들을 추출하였다. **prq_count**, **isu_count**는 각각 단위 기간 내 새로 생성된 풀 요청 수와 이슈 수를 의미하며, 이는 프로젝트 내에서 개발자들이 풀 요청 및 이슈 생성 빈도를 나타낸다. **prq_cmts**와 **isu_cmts**는 단위 기간 내 생성된 풀 요청과 이슈에 대한 댓글 수를 의미하며, 풀 요청과 이슈 내에서의 개발자 간의 의사 소통 빈도를 나타낸다. **prq_events**와 **isu_events**는 단위 기간 내 풀 요청과 이슈에 대해 발생한 이벤트의 수를 의미한다. GitHub에서 풀 요청과 이슈에 대한 세부 기능을 사용할 때마다 이벤트가 발생하게 된다. 따라서 발생한 이벤트의 수를 사용한 두 지표 값들은 풀 요청과 이슈의 세부 기능의 사용 빈도를 나타내게 된다.

8개의 독립 변수들에 대해 다중 공선성의 문제가 있는지 확인하기 위해 각 독립변수 별로 분산팽창요인(Variance Inflation Factor, VIF)을 추출하였다. 추출 결과 **prq_cmts**와 **prq_events**가 높은 다중 공선성을 의미하는 10 이상의 VIF값[8]을 보여 **prq_cmts** 독립 변수를 다항 선형 회귀 모델 생성시 제외하였다. **prq_cmts**를 제외한 독립 변수들의 VIF 값들은 아래 **표 1**에서 확인할 수 있다.

표 1 사용된 독립 변수들의 분산팽창요인 값

독립 변수	VIF
age	6.3
commits	4.2
prq_count	6.3
prq_events	8.3
isu_count	4.9
isu_cmts	4.5
isu_events	2.2

[1] adhocracy, autotest-docker, Board3-Portal, cas, distarray, Easy-Digital-Downloads, encore-ui, flot, gaffer, Gandalf, georchestra, Hercules, husky, ilios, invenio, jcabi-github, kuma, mapserver, mrjob, onadata, openfluid, orbisgis, OTM2, phantomas, plottable, qreal, qupzilla, RIOT, scala-js, scipy, SearchWorks, seqan, statsmodels, supercollider, vispy, wdntemplates, webview, WordPress-Android, zeroclickinfo-spice

프로젝트의 품질을 나타내는 지표로는 단위 기간 내에 발생한 버그 유발 변경 사항 수인 **bugs**를 사용하였다. 버그 유발 변경 사항을 찾기 위해서 SZZ 알고리즘[9]을 사용하였다. SZZ 알고리즘은 버그 수정과 관련된 키워드(fix, resolve, ...)가 포함된 소스 코드 변경 사항이 수정한 소스 코드 줄들을 마지막으로 수정했던 변경 사항들을 버그 유발 변경 사항으로 분류한다.

3.3 데이터 전처리

수집한 지표 값들에 대해 히스토그램과 분포 통계를 분석한 결과 8개의 지표 모두 양의 방향으로 분포가 치우친 것을 확인하였다. 각 지표 값들의 분포를 정규 분포 모양으로 변형시키기 위해 모든 분포에 $\mathbf{log(x+e)}$ 변환을 적용하였다. 모든 지표 값들은 음이 아닌 정수이기 때문에 로그 변환 후에도 모두 0 이상의 값을 가지게 된다. 아래 **그림 1**과 같이 로그 변환을 할 경우 값의 분포가 정규 분포에 가깝게 변환되는 것을 확인할 수 있다.

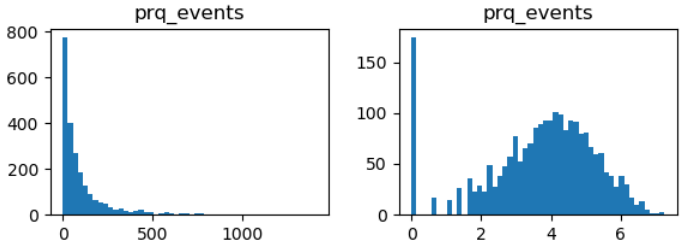


그림 1 prq_events 지표 값에 대해 로그 변환 적용 전(좌)과 적용 후(우)의 히스토그램

4. 실험 및 분석

표 2 단위 기간 내에 발생한 버그 유발 변경 사항 수를 설명하는 다항 선형 회귀 모델

지표	계수	유의 확률
age	-0.052	**0.000**
commits	0.672	**0.000**
prq_count	0.122	**0.000**
prq_events	-0.081	**0.000**
isu_count	0.075	**0.000**
isu_cmts	0.007	0.511
isu_events	-0.060	**0.000**
R^2		**0.930**

표 2는 수집한 데이터를 기반으로 생성한 다항 선형 회귀 모델을 설명하고 있다. 각 행은 사용된 지표별로 종속변수인 단위 기간 내 버그 유발 변경 사항과의 선형 관계를 나타내는 **계수**와 각 지표가 통계적으로 유의미하게 종속 변수를 설명하는지 나타내는 **유의 확률**로 구성되어 있다. 마지막 행은 회귀 식이 데이터를 얼마나 잘 표현하는지 나타내는 결정 계수이며 이 모델의 종속 변수의 93.0%가 독립 변수들로부터 설명된다는 것을 의미한다.

먼저 유의 확률이 0.01 이하인 지표들에 대해 각 계수가 의미하는 바에 대해 분석을 진행하였다. **age** 지표의 경우 다항 선형 회귀 모델에서 음의 계수를 가지고 있다. 이는 프로젝트가 오래 진행될수록 단위 시간당 발생한 버그 유발 변경 사항이 감소함을 의미하며 시간이 흐를 수록 프로젝트가 안정적으로 진행되는 것으로 해석할 수 있다. **commits** 지표는 7개의 지표 중 가장 높은 양의 계수를 가지고 있다. 이는 버그 유발 변경 사항이 발생하기 위해서는 변경 사항이 발생해야 하므로 두 값이 높은 양의 상관관계가 있는 것은 당연함을 알 수 있다.

단위 시간 내에 풀 요청과 이슈의 생성 빈도를 나타내는 **prq_count**와 **isu_count**는 모두 양의 계수를 가지고 있다. 이 2개의 값만 보면 개발 진행에 도움을 주기 위한 풀 요청과 이슈 기능이 오히려 소프트웨어의 품질을 저하시키는 결과로 보일 수 있다. 하지만 단위 시간 내에 풀 요청과 이슈와 관련된 이벤트 수를 나타내는 **prq_events**와 **isu_events**는 음의 계수를 가지고 있다. 이는 풀 요청과 이슈에 대한 세부 기능을 많이 사용할수록 버그 유발 변경 사항이 감소함을 의미한다. 따라서 4개의 값을 동시에 고려한다면 분류, 관리 등의 세부 기능들을 사용하지 않고 풀 요청과 이슈 기능을 이용할 경우 소스 코드 품질에 나쁜 영향을 끼칠 수 있음을 알 수 있다. **prq_count**의 계수는 **isu_count**보다 더욱 큰데, 이는 세부 기능을 이용하지 않고 풀 요청 기능을 사용할 경우 세부 기능을 이용하지 않은 이슈 기능 사용 시보다도 더 큰 악영향을 끼칠 수 있음 또한 확인할 수 있다.

이슈에 대한 댓글 수를 나타내는 **isu_cmts**는 높은 유의 확률을 가지고 있어 본 연구에서 생성한 모델로는 프로젝트에 대한 의사소통이 소프트웨어 품질에 미치는 영향을 설명 할 수 없다고 판단된다. 타 연구에서는 사용자들의 프로젝트에 대한 의사소통이 소스 코드 품질과 양의 선형 관계가 있다는 것을 단위 기간 내의 이메일 리스트에서 교환된 메일 수와 발생한 버그 유발 변경 사항 수의 상관관계를 통해 확인된 바가 있다[10].

5. 결론

본 논문에서는 39개의 프로젝트로부터 총 5,642개월의 데이터를 사용하여 이슈 및 풀 요청 활동들이 소스 코드 품질에 미치는 영향을 다중 선형 회귀 모델을 통해 설명하였다. 본 논문이 분석한 이슈 및 풀 요청 활동들의 소스 코드 품질에 대한 영향은 다음과 같다:

- Github에서 제공하는 풀 요청과 이슈에 대한 세부 기능을 사용하는 것은 소스 코드 품질에 긍정적인 영향을 끼친다.
- 세부 기능을 사용하지 않는 풀 요청과 이슈들은 소스 코드 품질에 부정적인 영향을 끼친다.

- 이슈 보다 풀 요청 기능이 소프트웨어 품질에 더 큰 영향을 끼친다.

분석 결과에 따라 GitHub 상의 프로젝트에 대해 풀 요청 기능을 이용할 경우 소스 코드 품질 향상을 위해 풀 요청 등록 및 수락 외에 다양한 세부 기능을 적극적으로 사용할 것을 제안한다. 또한, 프로젝트의 지표 값들을 기반으로 하여 소프트웨어 산출물의 결함 여부를 예측하는 결함 예측 연구 분야[11]에서 GitHub를 사용하는 프로젝트를 대상으로 결함 예측을 진행할 경우 풀 요청 및 이슈 관련 이벤트를 활용한 지표 값의 사용을 제안한다.

향후 연구에서는 풀 요청 및 이슈 관련 이벤트의 세부 종류별로 소스 코드 품질에 미치는 영향을 분석하고자 한다. 또한, 키워드 기반 버그 수정 변경 사항 분류와 SZZ 알고리즘의 정확도가 높지 않아[12], [13] 버그 유발 변경 사항 분류에 오류가 포함될 수 있다는 한계점을 극복하기 위해 개선된 SZZ 알고리즘[14]을 적용하고자 한다.

사사

이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보・컴퓨팅기술개발사업의 지원을 받아 수행된 연구임. (NRF-2017M3C4A7066210)

참고문헌

[1] G. Gousios, M. Pinzger, and A. van Deursen, “An Exploratory Study of the Pull-based Software Development Model,” in *Proceedings of the 36th International Conference on Software Engineering*, New York, NY, USA, 2014, pp. 345–355.

[2] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillère, J. Klein, and Y. L. Traon, “Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub,” in *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, 2013, pp. 188–197.

[3] N. Bettenburg and A. E. Hassan, “Studying the Impact of Social Structures on Software Quality,” in *2010 IEEE 18th International Conference on Program Comprehension*, 2010, pp. 124–133.

[4] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, “Quality and Productivity Outcomes Relating to Continuous Integration in GitHub,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, New York, NY, USA, 2015, pp. 805–816.

[5] M. Fowler, “Continuous integration,” *Martin Fowler*, 2006. [Online]. Available: https://martinfowler.com/articles/continuousIntegration.html.

[6] O. Jarczyk, B. Gruszka, S. Jaroszewicz, L. Bukowski, and A. Wierzbicki, “GitHub Projects. Quality Analysis of Open-Source Software,” in *Social Informatics*, 2014, pp. 80–94.

[7] S. Kim and E. J. Whitehead Jr., “How Long Did It Take to Fix Bugs?,” in *Proceedings of the 2006 International Workshop on Mining Software Repositories*, New York, NY, USA, 2006, pp. 173–174.

[8] R. M. O’brien, “A Caution Regarding Rules of Thumb for Variance Inflation Factors,” *Qual. Quant.*, vol. 41, no. 5, pp. 673–690, Oct. 2007.

[9] J. Śliwerski, T. Zimmermann, and A. Zeller, “When Do Changes Induce Fixes?,” in *Proceedings of the 2005 International Workshop on Mining Software Repositories*, New York, NY, USA, 2005, pp. 1–5.

[10] R. Abreu and R. Premraj, “How Developer Communication Frequency Relates to Bug Introducing Changes,” in *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops*, New York, NY, USA, 2009, pp. 153–158.

[11] Y. Kamei and E. Shihab, “Defect Prediction: Accomplishments and Future Challenges,” in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016, vol. 5, pp. 33–45.

[12] T. Hall, D. Bowes, G. Liebchen, and P. Wernick, “Evaluating Three Approaches to Extracting Fault Data from Software Change Repositories,” in *Product-Focused Software Process Improvement*, 2010, pp. 107–115.

[13] D. A. da Costa, S. McIntosh, W. Shang, U. Kulesza, R. Coelho, and A. Hassan, “A Framework for Evaluating the Results of the SZZ Approach for Identifying Bug-Introducing Changes,” *IEEE Trans. Softw. Eng.*, vol. PP, no. 99, pp. 1–1, 2016.

[14] S. Kim, T. Zimmermann, K. Pan, and E. J. J. Whitehead, “Automatic Identification of Bug-Introducing Changes,” in *21st IEEE/ACM International Conference on Automated Software Engineering (ASE’06)*, 2006, pp. 81–90.

모바일 클라이언트와 서버간 통합테스트 애스팩트 구성 방법

최윤석 ○, 최은만
동국대학교 컴퓨터공학과
ysmanse@gmail.com, emchoi@dgu.ac.kr

A Construction Method of Integration Testing Aspect Between Mobile Client and Server

Yun Seok Choi○, Eun Man Choi
Dongguk University

요 약

모바일 클라이언트와 서버의 개발이 동시에 진행될 때 모바일 클라이언트는 사용자에게 기능을 제공하기 위해 필요한 데이터가 서버에 의존적이다. 이러한 이유로 모바일 클라이언트의 개발 초기부터 완료까지 서버와의 통합테스트는 필수 적이다. 본 논문에서는 Google Spreadsheet API와 AOP(Aspect-Oriented Programming)를 활용하여 소프트웨어의 코드 수정없이 구성된 애스팩트(Aspect)의 이식만으로 가능한 모바일 클라이언트와 서버간 통합테스트 방법을 제안한다.

1. 서 론

소프트웨어 통합테스트(Software Integration Testing)는 둘 이상의 시스템 구성요소가 통합되었을 때 기대한 대로 기능이 동작하는지 검사하는 테스팅 방법이다. 실시간으로 변화하는 데이터를 제공하는 페이스북이나 트위터와 같은 많은 모바일 애플리케이션은 중앙 서버와 통신하여 다양한 서비스를 제공한다. 이러한 형태의 모바일 애플리케이션 개발은 대부분 서버와 동시에 개발이 이루어 진다. 사용자에게 제공되는 대부분 기능에 필요한 데이터가 서버에 저장되어 있기 때문에 클라이언트와 서버간 통합 테스트는 필수적이다. 하지만 필요한 API 개발이 완료되었다 하더라도 응답 데이터가 없거나 테스트에 필요한 충분한 데이터가 아니라면 다양한 조건에 따라 제공되는 기능과 UI를 테스트할 수 없게 된다. 또한 개발 일정의 차이로 인해 서버에 API 자체가 개발이 되지 않은 경우가 있다.

현재는 대부분 모바일 애플리케이션과 서버간 통합테스트를 위해 실제 서버를 흉내 내는 가상의 모크 서버를 구성하여 테스트를 진행하는 방법이 일반적이다[1,2,3]. 그러나 모크 서버 또한 하나의 작은 서버이기 때문에 독립적인 시스템으로 구성해야 하는 물리적, 시간적 비용이 발생한다. 또한 모바일 애플리케이션에서 구성된 모크 서버로 접속하기 위해 연결할 도메인을 변경해야 하는데 이로 인해 모든 통신이 모크 서버로 연결된다. 테스트가 완료되거나 개발이 완료된 API는 실제 서버로 연결하도록 하고 그 외 테스트가 필요한 API 대해서만 테스트를 진행하는 것은 불가능 하다.

본 논문에서는 Google Spreadsheet로 작성된 API 연동 규격 문서를 Google Spreadsheet API를 활용하여 응답 테스트 데이터 구성한다. 그리고 기능 요구사항을 핵심 모듈과 별개로 보조적인 횡단 모듈로 분리할 수 있도록 지원하는 AOP(Aspect-Oriented Programming)[4]를 기반으로 애스팩트 라이브러리를 구현하여 별도의 서버 구성 비용 없이 통합테스트를 가능하게 한다. 애스팩트가 iOS와 안드로이드 OS의 코어 네트워크 라이브러리를 Point Cut 하여 애스팩트의 수정 없이 모든 요청에 대해 모니터링이 가능하게 하고 테스트로 선언한 API를 제외한 요청은 실제 서버로 연결하도록 하여 유연한 테스트 진행을 가능하게 한다. 마지막으로 제안한 방법을 토대로 iOS 환경에서 구동 가능한 애스팩트를 구현하여 실험하였다.

2. 모바일 클라이언트의 코어 네트워크 라이브러리

대표적인 모바일 애플리케이션 OS인 iOS와 Android OS는 각각 Objective-C와 JAVA를 사용하여 애플리케이션을 개발 한다. AOP(Aspect-Oriented Programming)를 사용하기 위해 JAVA에서는 AspectJ 라는 라이브러리를 통해 AOP의 핵심 기능인 Point Cut, Join Point, Advice 등의 기능을 사용할 수 있다[5][6]. Objective-C는 C언어에서 스몰 토크 스타일의 메시지 구문을 추가한 객체 지향 언어 이다[7]. C언어 기반의 Objective-C의 경우 런타임의 동적인 특성을 갖고 있다. 그러한 특성 때문에 런타임상에서 실행되는 메소드를

교체 혹은 확장하는 것이 가능하다. Objective-C에서 이러한 방법을 Method Swizzling이라 부르며, Method Swizzling을 활용해 Objective-C에서도 AOP를 활용할 수 있다[8,9].

모바일 클라이언트와 서버간 통합테스트를 위해 모바일 클라이언트에서 발생하는 통신을 모니터링 해야 한다. 그림 2와 같이 모바일 클라이언트 SDK에서 제공하는 HTTP 통신 모듈을 직접 Point Cut 함으로써 해결할 수 있다. SDK 에서 제공하는 통신 모듈을 직접 Point Cut 하면 한번에 모바일 애플리케이션에서 발생하는 모든 HTTP 통신 요청을 모니터링 할 수 있다[10]. 이와 같이 애스팩트 포멧을 구성함으로써 같은 플랫폼의 모든 프로그램에 별다른 수정 없이 적용이 가능하며 적용 이후에 발생하는 프로그램의 수정 또는 확장에 영향을 받지 않게 된다. 본 논문에서는 iOS 환경에서 Method Swizzling을 활용해 구현한 애스팩트 라이브러리를 통해 실험을 진행한다.

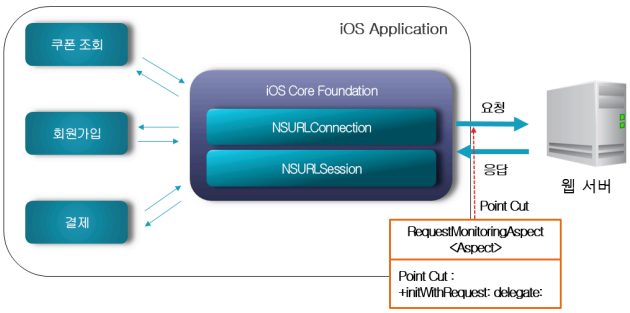


그림 2 모바일 애플리케이션의 모든 통신 요청 모니터링을 위한 AOP 적용 방법

3. Google Spreadsheet를 활용한 통합테스트 애스팩트

3.1 Google Spreadsheet API 연동을 위한 config.json 파일의 구성

Google Spreadsheet API를 연동하기 위해 구글에서 API 접근을 위해 제공하는 키 관리를 해줄 도구가 필요한데 본 연구에서는 테스트 API 선언과 관리를 위해 config.json 파일을 활용한다. 표 1과 같이 JSON 형태의 간단한 룰 셋을 미리 정의하여 애스팩트 함께 제공한다. “clientId”는 Google Spreadsheet API를 사용하기 위해 사용자 등록을 하면 발급하는 키를 뜻한다. “documentId” 는 공유된 연동 규격서 문서의 Google Spreadsheet 문서 고유 아이디이다. 2가지 항목을 입력하여 손쉽게 사용중인 연동 규격서 문서에 접근이 가능하게 한다.

표 1 테스트 API 설정 JSON 노드 정의

Type	Key	Description
S	clientId	Google API 사용자 고유 ID
S	documentId	문서 ID

```
{
 “clientId”: 373700882678-sj47ft4
 “documentId”: 1BxiMVs0XRA5nFMdKvBdBZjgmU
}
```

그림 3 config.json 파일의 설정 예

3.2 통합테스트 애스팩트의 흐름

앞서 설명한 네트워크 코어 라이브러리 Point Cut 방법과 Google Spreadsheet API를 통해 그림 4와 같은 흐름으로 통합테스트 애스팩트를 구현한다. 애스팩트 클래스에서 모든 HTTP 통신 요청에 대한 모니터링을 하고 있기 때문에 모바일 클라이언트에서 서버 요청이 발생하면 이를 감지하게 된다. 감지된 요청은 Google Spreadsheet 폼 내의 테스트 시행 선언부를 확인하여 테스트 여부를 판단을 한다. 테스트 API로 선언된 요청이라면 함께 작성된 반환 데이터 JSON 파일의 내용을 응답하고 테스트 선언이 되지 않은 요청에 대해서는 실제 서버로 연결하여 응답 데이터를 받게 한다.

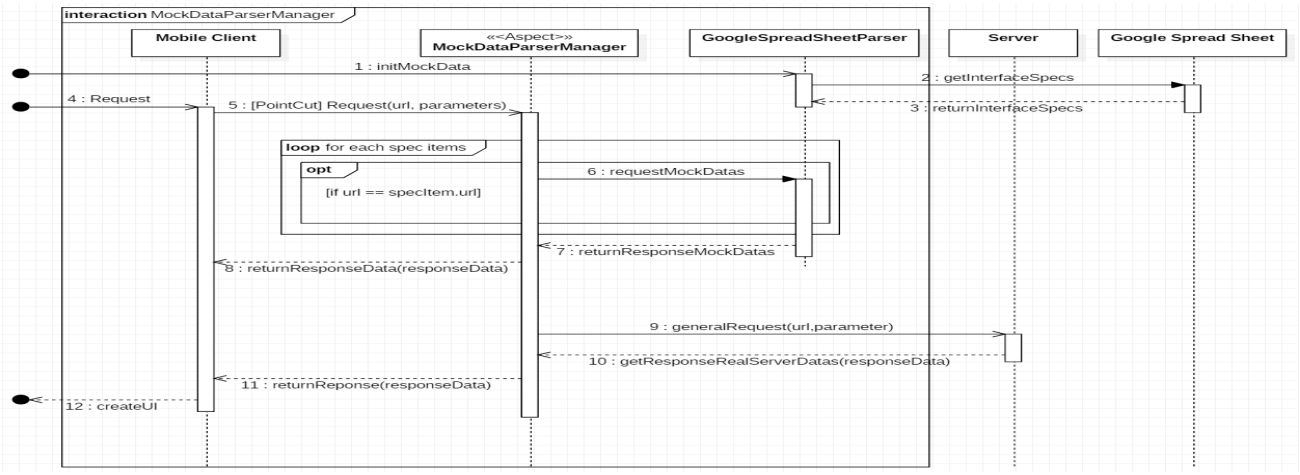
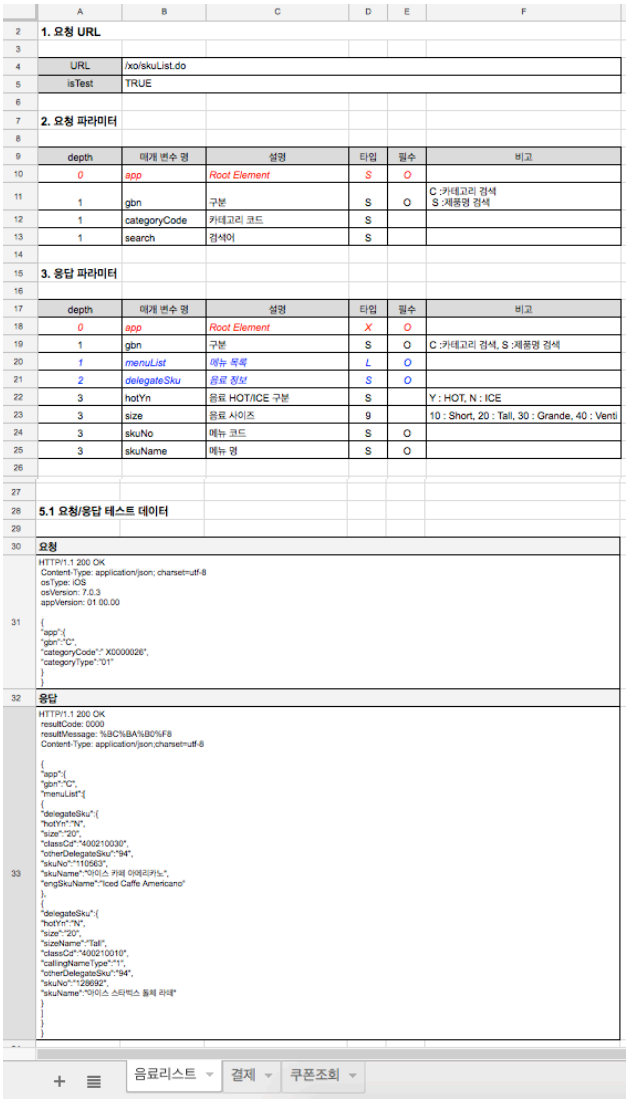


그림 4 통합테스트 애스팩트의 흐름

3.3 Google Spreadsheet 폼 정의

모바일 애플리케이션 개발 착수 단계에서 서버 연동 규격서를 모바일 애플리케이션 개발팀과 서버 개발팀간 어떻게 공유할 것인지를 결정한다. 최근에는 팀원 등록만으로 문서 공유가 가능하고 실시간으로 업데이트된 내용을 확인 가능한 Google Document를 많이 활용한다. 통합테스트 애스팩트에서 공유된 Google Spreadsheet를 활용하여 테스트 여부 판단과 응답 데이이터를 생성하기 때문에 미리 그림 5와 같이 폼의 형태를 정의하였다. 문서를 배포하는 서버 개발팀에서는 해당 폼에 맞게 문서를 배포하고 공유받은 모바일 애플리케이션 개발팀은 테스트가 필요한 API에 대해 isTest 항목에 TRUE, FALSE 값을 입력하여 API 에 대한 테스트 선언을 한다. 테스트에 필요한 응답 데이터를 폼 아래의 응답 필드에 미리 작성하면 모바일 애플리케이션에서 해당 API 에 대한 요청이 발생했을 때 통합테스트 애스팩트는 이 문서를 토대로 미리 작성된 응답데이터를 반환한다.



1. 요청 URL

URL	/xo/skuList.do
isTest	TRUE

2. 요청 파라미터

depth	매개 변수 명	설명	타입	필수	비고
0	app	Root Element	S	O	
1	gbn	구분	S	O	C :카테고리 검색 S :제품명 검색
1	categoryCode	카테고리 코드	S		
1	search	검색어	S		

3. 응답 파라미터

depth	매개 변수 명	설명	타입	필수	비고
0	app	Root Element	X	O	
1	gbn	구분	S	O	C :카테고리 검색, S :제품명 검색
1	menuList	메뉴 목록	L	O	
2	delegateSku	음료 정보	S	O	
3	hotYn	음료 HOT/ICE 구분	S		Y : HOT, N : ICE
3	size	음료 사이즈	9		10 : Short, 20 : Tall, 30 : Grande, 40 : Venti
3	skuNo	메뉴 코드	S	O	
3	skuName	메뉴 명	S	O	

5.1 요청/응답 테스트 데이터

요청

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
osType: iOS
osVersion: 7.0.3
appVersion: 01.00.00

{
"app":{
"gbn":"C",
"categoryCode":" X0000026",
"categoryType":"01"
}
}
```

응답

```
HTTP/1.1 200 OK
resultCode: 0000
resultMessage: %BC%BA%B0%F8
Content-Type: application/json;charset=utf-8

{
"app":{
"gbn":"C",
"menuList":[
{
"delegateSku":{
"hotYn":"N",
"size":"20",
"classCd":"400210030",
"otherDelegateSku":"94",
"skuNo":"110563",
"skuName":"아이스 카페 아메리카노",
"engSkuName":"Iced Caffe Americano"
},
{
"delegateSku":{
"hotYn":"N",
"size":"20",
"sizeName":"Tall",
"classCd":"400210010",
"callingNameType":"1",
"otherDelegateSku":"94",
"skuNo":"129892",
"skuName":"아이스 스타벅스 돌체 라떼"
}
]
}
}
```

음료리스트 | 결제 | 쿠폰조회

그림 5 Google Spreadsheet 폼 정의

4. 애스팩트 라이브러리의 구성

그림 6과 같이 통합테스트 애스팩트 라이브러리를 구성 한다. iOS 환경에서 AOP를 활용할 수 있도록 제공되는 Aspects 라이브러리와 Google Spreadsheet API에 접근하기 위해 구글에서 제공하는 Google Spreadsheet API 라이브러리가 포함된다[11,12]. AspectManager 폴더에는 구현된 애스팩트 클래스와 애스팩트 클래스에서 사용될 Helper 매소드를 갖는 Helper 클래스가 함께 구성되며 GoogleSpreadsheetAPIManager 폴더에는 작성된 연동 규격 문서를 파싱하기 위한 클래를 포함한다. config.json 파일을 제공하여 공유된 Google Spreadsheet 접근을 위한 키 관리를 할 수 있게 한다. 이와 같은 내용물을 패키지 형태로 제공한다.

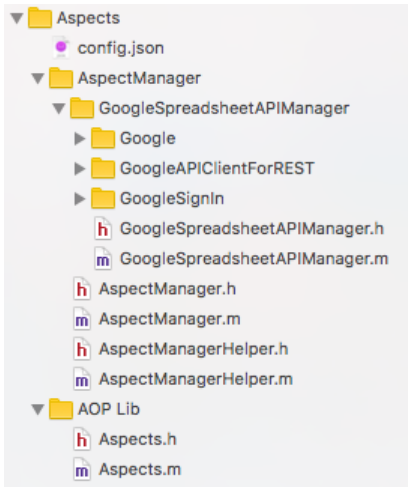


그림 6 통합테스트 애스팩트의 패키지 구성

5. 실험 결과

본 연구의 실험은 현재 널리 사용중인 커피전문점 애플리케이션에 구현된 통합테스트 애스팩트를 추가하여 진행하였다. 먼저 그림 7 의 첫번째 화면과 같이 현재 보유한 쿠폰이 없는 경우 화면에 표시되는 항목이 없다. 이런 경우 쿠폰리스트 화면에 대한 테스트가 불가능할 뿐만 아니라 쿠폰리스트 화면을 통해 진행되는 쿠폰상세보기, 쿠폰사용하기 등 연계되는 기능들도 테스트가 불가능하다.

테스트를 위한 쿠폰 리스트를 만들기 위해 두번째 화면과 같이 Google Spreasheet 에 작성된 문서의 해당 API 시트에 isTest 를 true 로 변경하고 응답 받을 데이터를 입력하였다. 이와 같이 변경 후 다시 쿠폰 리스트를 로드하면 3번째 화면과 같이 테스트 응답데이터를 받아 쿠폰 리스트가 표시 되어 리스트 기능 및 화면 테스트 뿐만 아니라 관련되는 쿠폰상세보기, 쿠폰사용하기 등의 기능도 테스트가 가능하게 되었다.

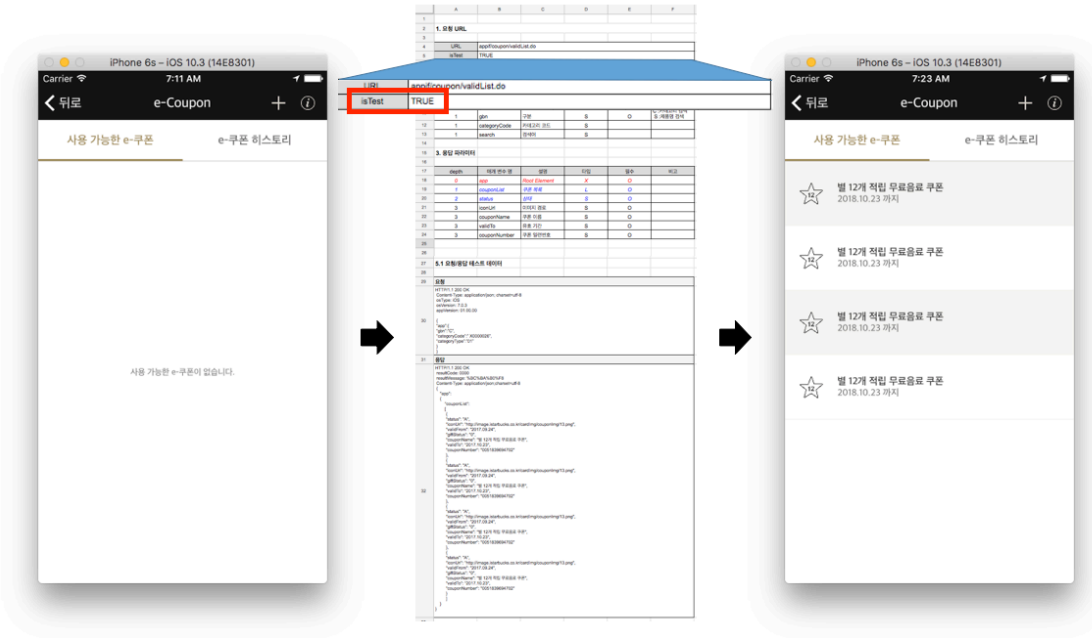


그림 7 커피전문점 애플리케이션 통합 테스트 애스팩트 적용

5. 결론 및 향후 연구

본 논문에서는 AOP와 Google Spreadsheet를 활용하여 모바일 클라이언트와 서버 간의 통합 테스트를 위한 방법을 제안하고 구현하였다.

AOP를 활용하여 애스팩트 클래스를 구성하였기 때문에 기존의 프로그램의 수정 없이 적용이 가능하여 유지보수성이 향상되었다. 또한 동일한 플랫폼의 어떠한 프로그램에 구현된 애스팩트 클래스를 추가하는 것만으로 동작되어 이식성이 개선된다.

일방적으로 연결되는 모크 서버를 사용하는 방법과 달리 테스트로 선언된 API가 아닌 요청은 자동으로 실제 서버로 연결하여 선택적인 테스트가 가능하다. 또한 널리 사용되는 스프레드시트 형태의 연동규격서를 테스팅 도구로 활용하여 프로그래밍을 다룰 줄 모르는 인원도 손쉽게 테스팅을 진행할 수 있다. 또한 실시간으로 웹 상의 Google Spreadsheet에 작성된 내용을 기반으로 테스트 데이터를 불러오기 때문에 테스트 데이터의 변경이 일어나도 프로그램을 다시 빌드할 필요가 없어 시간적비용이 감소하였다.

향후 진행할 연구로는 안드로이드 OS에 적용 가능한 JAVA 기반의 통합테스트 애스팩트를 개발할 것이다. 현재는 Google Spreadsheet 내에 필요한 데이터를 입력하는 형태였지만 추후 필드의 형태를 토대로 자동으로 응답데이터를 만들 수 있도록 하여 시간적비용을 최소화할 계획이다.

참 고 문 헌

[1] Jame D Bloom, “MockServer”, http//www.mockserver.com/

[2] CyberAgent, “EasyMock”, https://github.com/CyberAgent/node-easymock

[3] Blake Mizerany, “Sinatra”, https://www.sinatrarb.com

[4] Kiczales, Gregor, G Lamping A Mendhekar, C Maeda, C Lopes, J Loingrier, J Irwin, Aspect-Oriented Programming, Proc. Of ECOOP, Vol.1241, pp.220-242, 1997.

[5] Kiczales, Gregor, et al,. “An overview of AspectJ”, ECOOP 2001, Object-Oriented Programming, Springer Berlin Heidelberg, p.327-354, 2001.

[6] Ron Bodkin, Adrian Colyer, Mik Kersten, “AspectJ”, https://www.eclipse.org/aspectj

[7] Stephen G. Kochan, Programing Objective-C,Addiison-Wesley Professional, 2013.

[8] Kyle Richter, Joe Keeley, iOS Components and Frameworks : Understanding the Advanced Feateres of the iOS SDK, pp.300-312, 2013.

[9] NSHipster, “Method Swizzling”, http://nshipster.com/method-swizzling/

[10] 최은만, 최윤석, “AOP를 활용한 모바일 클라이언트와 서버간 민간 정보 전송의 동적 모니터링 방법”, 한국소프트웨어공학 학술대회, 19권 1호, 55-58.

[11] Steipete, Aspects, https://github.com/steipete/Aspects.

[12] Google Inc., “Google Spreadsheet API”, https://developers.google.com/sheets/api

유닛 테스트 케이스 반복 실행을 통한 자바 프로그램에서의 메모리 블롯 오류의 효과적 검출: Apache Commons VFS 대상 적용 사례

이재훈 홍신
한동대학교 전산전자공학부
{21400575, hongshin}@handong.edu

Detecting Memory Bloats of Java Programs by Monitoring Repeated Unit Test Executions: A Case Study with Apache Commons VFS

요 약

본 논문은 Java 프로그램에서 발생할 수 있는 메모리 누실 오류의 한 종류인 메모리 블롯을 효과적으로 탐지하는 새로운 동적 분석 기법을 제안하고, 이를 Apache Commons VFS에 적용하여 실제 결함을 검출한 사례 연구를 소개한다. 제안한 동적 분석 기법은 기존에 개발된 유닛 테스트 케이스를 활용하여, 같은 동작을 반복 수행 하는 가운데 메모리 증가가 발생하는 지 자동으로 검사함으로써 Java 프로그램에서의 메모리 블롯을 효과적으로 탐지한다. 제안한 기법을 Apache Commons VFS에 적용한 결과, 실제적인 메모리 블롯을 자동으로 검출할 수 있었으며, 이를 분석하여 3개의 신규 메모리 블롯 결함을 발견하였다.

1. 서론

메모리 블롯(memory bloat)이란 프로그램이 특정 시점 이후에 기능(functionality) 제공에 불필요한 메모리 자원을 반환하지 않음으로써 실행 중 메모리 사용량이 불필요하게 증가하는 문제를 일으키는 일종의 *메모리 누실*(memory leak) 오류를 뜻한다[1, 2]. 메모리 블롯 발생으로 인한 메모리 누실은 일차적으로 메모리 자원 관리 비용 증대로 인한 성능저하를 일으킬 수 있으며, 장기간 누적될 경우, 메모리 부족 현상을 발생시켜 서비스 실패 등 심각한 소프트웨어 안정성 문제를 야기할 수 있다. 할당된 메모리 주소의 상실로 발생하는 메모리 상실(memory loss) 결함과 달리, 메모리 블롯 결함은 Java와 같이 불용 메모리 수거(garbage collection)가 제공되는 언어/실행환경에서도 발생할 수 있으며, 일반적인 테스팅에서 오류 증상을 명확히 식별하기 어려워 디버깅이 난해한 결함으로 알려져, 효과적인 메모리 블롯 검출 기법의 개발이 요구되는 상황이다.

기존 연구에서는 시스템 테스트(system testing) 중 메모리 사용을 프로파일링(profiling)함으로써, 메모리 블롯 발생을 통계적으로 추정하는 방식이 제안되어 왔다[2, 3, 4]. 이러한 프로파일링 기반 기법은 개발자가 메모리 블롯이 두드러지게 발생하는 시스템 테스트를 제공해 주지 않을 경우, 효과적인 오류의 탐지가 불가능하므로 오류 검출 효용성과 민감성에 한계를 가지고 있다. 뿐만 아니라, 프로파일링 기반 방식은 실행 과정에서 어떠한 메모리가 기능에 불필요한지 판별하기 어려우므로, 정확한 오류 분석에 한계가 있다는 점에서 또 다른 한계점을 가진다.

본 논문은 Java 프로그램 실행에서 발생한 메모리 블롯을 효과적으로 탐지하기 위해 유닛 테스트 케이스를 활용하여 메모리 블롯 검출에 유용한 프로그램 실행을 자동으로 생성하고, 실행 중 체계적으로 메모리 사용을 추적한 기록을 분석함으로써 메모리 블롯을 민감하게 탐지, 분석하는 새로운 동적 분석 기법을 제안한다. 제안하는 동적 분석 기법은 유닛 테스트 케이스에 따른 실행에서 메모리 블롯을 효과적으로 관찰하기 위해, 각 유닛 테스트 케이스를 의도적으로 수차례 반복 재생하고, 각 재생마다 메모리 사용량과 메모리 상태를 비교하여, 동일한 기능을 수행함에도 메모리 소비가 지속적으로 증가하는 현상을 메모리 블롯 오류로 검출한다. 본 연구에서 제안한 기법을 실제적인 Java 프로그램에 대해 동작하는 도구로 구현하여, Apache Commons VFS [5] 에 적용한 사례 연구 결과, 이전에 발견되지 않은 3개의 실제적인 메모리 블롯 결함을 검출할 수 있었다.

본 논문이 제안하는 기법은 사용자가 설계한 시스템 테스팅 대신, 유닛 테스트 케이스로부터 메모리 블롯 검출에 효과적인 실행을 능동적으로 생성하여 활용한다는 점에서, 수동적 프로파일링에 의존하는 기존 연구와 차별성을 갖는다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구의 주제와 접근 방법에 대한 이해를 돕기 위해 유닛 테스트 케이스의 기능과 구조, 메모리 블롯 결함의 예제를 소개한 후 Apache Commons VFS 프로젝트에서 발견된 실제 결함 사례를 소개한다. 3장에서는 본 논문이 제안하는 새로운 메모리 블롯 검출 기법을 소개하고, 이어서 4장에서는 Apache Commons VFS를 대상으로 제안한 기법을 적용하여 메모리 블롯을 실제적으로 검출하고 3개의 신규 결함을 발견한 사례 연구를 소개한다. 5장에서는 향후 연구 계획과 함께 논문을 마무리한다.

2. 연구배경(Background)

2.1 유닛 테스트 케이스의 기능

*유닛 테스팅*은 검증대상 프로그램의 특정 모듈 혹은 객체를 테스트 프로세스 내에서 생성한 후 일련의 명령/메소드를 호출하고 그 반환 값을 검사하는 과정들, 즉 *유닛 테스트 케이스*들의 집합으로 구성된다. 검증대상 프로그램의 완성된 코드를 실행시킨 후 실제 사용자가 발생시킬 수 있는 데이터를 입력하여 여러 기능 간의 상호작용을 검사하는 시스템 테스트 케이스와 달리, 각 유닛 테스트 케이스는 검증대상 프로그램의 구체적인 기능을 특정하여 수행하고 검사하는 것을 목표로 설계되는 것이 일반적이다. 유닛 테스트 케이스는 개발 과정에서 지속적인 회귀 테스팅(regression testing)을 가능하게 하고 각 모듈 API의 용례로 사용될 수 있기 때문에, 오늘날 소프트웨어 개발 프로젝트에서 품질관리와 협업의 중요한 매체로 인식되고 있으며, 대부분의 실제 프로젝트에서 소프트웨어 주요 요소로 체계적으로 개발/관리되고 있다.

본 연구는 한국연구재단 신진연구자지원과제(NRF-2017R1C1B1008159) 지원으로 수행됨.

```
01 class Stack {           08 public Object pop() {
02  Object [] arr;        09  if (!isEmpty()) {
03  int size;             10   Object e=arr[--size];
04  Stack() {             11 /*FIX: arr[size]=null;*/
05   arr=new Object[MAX]; 12   return e;
06  }                     13  }
07  ...                   14 }
```

그림 1. 메모리 블롯 결함 예제

본 연구에서는 이와 같은 유닛 테스트 케이스에 주목하여, 검증대상 프로그램에 대해 개발자가 올바르게 작성한 유닛 테스트 케이스가 충분히 제공되는 상황을 가정하였다. 또한 각 유닛 테스트는 특정 기능을 올바르게 실행하는 코드로 작성되었으므로, 이를 반복 실행할 경우, 항상 동일한 기능/동작이 수행되게 되며, 그 과정에서 불필요한 메모리의 증대는 없어야 한다고 추정하였다.

2.2. 유닛 테스트 스위트의 구조

일반적으로, 특정 모듈에 관련된 유닛 테스트 케이스들은 하나의 *유닛 테스트 스위트*(unit test suite) 형태로 작성된다. 유닛 테스트 스위트는 해당 모듈의 유닛 테스트 케이스들과 유닛 테스트 케이스의 구동에 필요한 일체 *테스트 구동 코드*로 구성된다. 테스트 구동 코드는 각 유닛 테스트 케이스의 실행 전후에 호출되어 테스트 케이스 실행에 필요한 객체를 생성/소멸하거나, 자원을 할당/해제함으로써 유효한 실행 환경을 구성하는 역할을 한다.

유닛 테스팅 프레임워크는 개발자가 유닛 테스트 스위트를 체계적으로 작성할 수 있는 구조를 제공하고, 구조에 따라 작성된 테스트 구동 코드와 테스트 케이스를 일련의 순서로 실행시켜, 테스트 케이스가 올바르게 구동되도록 지원한다. Java에서 표준적으로 사용되는 JUnit 프레임워크의 경우, 하나의 유닛 테스트 스위트를 한 개의 TestClass 클래스로, 각 테스트 케이스를 한 개의 메소드로 작성하도록 지원한다. 또한, @BeforeClass, @AfterClass, @Before, @After와 같은 메소드 속성 선언을 통해서 테스트 스위트 전체 초기화, 전체 마무리, 각 테스트 케이스 실행 직전 초기화, 실행 직후 마무리에 필요한 테스트 구동 코드를 개발자가 작성하도록 지원한다.

2.3 메모리 블롯 결함

메모리 블롯은 실행 중 특정 시점 이후로 다시 사용하지 않을 객체를 제거하지 않는 결함으로 인하여, 소멸(reclaim)되지 않는 불필요한 객체가 지속적으로 발생하게 되는 오류적 상황을 뜻한다. Java 프로그램에서는 불필요한 객체에 대한 참조(hard reference)를 모두 소멸함으로써, 해당 객체를 불용 객체(eligible object)로 선언하여 불용 메모리 수거 루틴에 의해 자동으로 소멸되도록 할 수 있다. Java 프로그램에서 메모리 블롯은 결함으로 인해 불필요해진 객체를 향한 모든 참조 제거를 놓치거나 실패할 경우 발생하며, 주로 배열이나 리스트(list), 맵(map)과 같은 *컨테이너*(container) 관리를 잘못 처리 하여 발생하는 경우가 빈번하다.

그림1은 Java로 스택을 구현한 코드에서 발생한 메모리 블롯의 예이다. 본 프로그램은 배열을 통해 스택 데이터 구조를 구현하고 있다. `Stack`은 데이터 객체를 입력 받아 `arr` 배열(2행)에 저장한다. `Stack`은 `pop()` 명령이 실행될 경우, `arr` 배열에 가장 최근에 저장된 데이터를 반환하고, `size`를 줄여 더 이상 스택에서 사용할 수 없게 한다. 이때, `arr[size]`에 불필요한 객체에 대한 참조가 남아있으므로, 해당 객체가 메모리에서 사라지지 않고 남아있게 된다. 이는 결과적으로 필요이상의 메모리 사용을 발생시키는 이상 증상, 즉 메모리 블롯을 야기한다. 이 결함은, 코드의 11행의 주석에 표시한 바와 같이, 불필요한 참조를 제거하는 코드를 추가함으로써 수정(bug fix)할 수 있다.

표1. Apache Commons VFS에서 보고된 11개 메모리 블롯 사례

Bug ID	Severity	오류 발생 상황	결함 원인
VFS-134	Major	컨테이너에 포함 (가)	잘못된 컨테이너에서 제거 (C)
VFS-140	Minor	컨테이너에 포함 (가)	소유자 객체의 정리 실패 (A)
VFS-142	Major	컨테이너에 포함 (가)	쓰레드 종료 전 미제거(B)
VFS-143	Major	컨테이너에 포함 (가)	약한 참조 미사용(D)
VFS-227	Major	컨테이너에 포함 (가)	쓰레드 종료 전 미제거(B)
VFS-287	Major	다른 객체가 참조 (나)	소유자 객체의 정리 실패 (A)
VFS-309	Blocker	컨테이너에 포함 (가)	쓰레드 종료 전 미제거(B)
VFS-480	Critical	컨테이너에 포함 (가)	소유자 객체의 정리 실패 (A)
VFS-544	Minor	컨테이너에 포함 (가)	잘못된 컨테이너에서 제거 (C)
VFS-545	Major	컨테이너에 포함 (가)	소유자 객체의 정리 실패 (A)
VFS-633	Major	컨테이너에 포함 (가)	소유자 객체의 정리 실패 (A)

2.4 Apache Commons VFS에 발생한 실제 메모리 블롯 사례

본 연구에서는 실제 소프트웨어 발생한 메모리 블롯 실태를 이해하기 위해, 오픈소스 프로젝트인 Apache Commons VFS 저장소를 탐색하여, 이전에 실제 발생한 메모리 블롯 관련 사례를 수집하고 정리하였다. Apache Commons VFS(이후 VFS)는 Java 어플리케이션에서 여러 파일 시스템에 접근할 수 있는 API를 제공하는 라이브러리 프로그램으로, 복잡한 파일 관련 객체를 지속적으로 생성/해체하는 코드가 복잡한 편이다.

VFS 프로젝트의 이슈 관리 시스템(issue tracker)에서 해결된 결함(bug)으로서 'memory leak' 혹은 'memory bloat' 이라는 키워드와 연관된 이슈를 모두 검색하였다. 검색 결과, 2005년 1월부터 2017년 12월까지(총 13년간) 보고된 총 406개의 해결된 결함 중 관련 키워드를 포함하는 이슈는 총 15개였으며, 각 사례를 정성적으로 분석한 결과. 총 11개의 메모리 블롯 실제 사례를 파악할 수 있었다.

표1은 11개 사례에 대해 관리자(maintainer)가 부여한 심각도(severity)와, 정성적 분석을 통해 파악한 메모리 블롯 발생 상황과 결함 원인을 분류한 결과다. 표에서 알 수 있듯이 11개 중 9개 결함이 Critical, Blocker, Major와 같이 주요 결함으로 개발자들에게 인식되었음을 확인할 수 있다.

정성적으로 분석 결과, 11개 메모리 블롯 결함은 '오류 발생 상황'에 따라 다음 두 가지로 종류로 분류 할 수 있다.

가. 맵(Map), 리스트(List)와 같은 컨테이너에서 불필요한 객체 참조 (총 10건)

나. 다른 객체의 멤버가 불필요한 객체를 참조 (총 1건)

11개 사례에 대해 오류를 일으킨 결함의 원인을 정성적으로 조사한 결과, 다음 4개의 유형이 있음을 파악할 수 있었다.

A. 소유자 객체(owner)가, 재사용의 목적으로 소유하고 있던 데이터 객체를 정리(clear) 과정에서 실수로 특정 데이터 객체에 대한 참조를 지우지 않아, 불필요한 데이터 객체가 메모리 블롯을 일으키는 경우. (총 5건)

B. 쓰레드(thread)가 기능 수행을 위해 공역 데이터에 등록하였으나, 종료 시점에 불필요해 진 객체를 제거하지 않고 쓰레드를 종료하게 됨. (총 3건)

C. 불필요하게 된 객체를 컨테이너에서 제거하기 위한 코드가 있으나, 실수로 해당 객체와 상관없는 컨테이너에 대해 제거 명령을 수행 한 경우. 결과적으로 불필요한 객체가 제거되지 않음. (총 2건)

D. 약한 참조(weak reference)를 사용 해야 하는 상황에서 실수로 해당 객체에 대한 일반 참조(strong reference)를 사용해 예기치 않게 해당 객체에 대한 소멸이 발생하지 않게 된 경우. (총 1건)

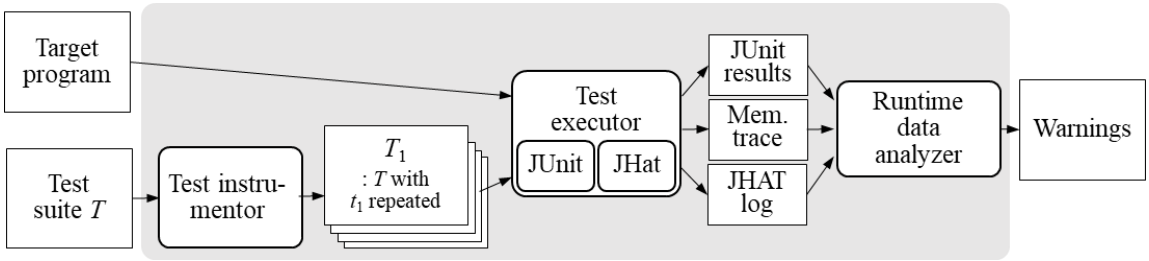


그림 2. 동적 분석 기법의 구성과 실행 흐름

3. 메모리 블롯 검출 기법

3.1 동적 분석 과정

메모리 블롯 결함을 효과적으로 탐지하기 위하여, 본 논문에서는 유닛 테스트 케이스를 반복 실행 시켜 동일한 동작을 지속적으로 발생시키고, 이 때 프로그램이 사용하는 메모리 사용량의 추세를 분석하는 동적 분석(dynamic analysis) 기법을 제안한다. 일반적으로 개발자가 작성한 유닛 테스트 케이스는 분석대상 프로그램의 특정 모듈을 시동한 후 특정 기능을 수행시키고, 해당 모듈을 종료하는 일련의 과정으로 구성된다. 따라서, 올바르게 작성된 유닛 테스트 케이스의 경우, 여러 차례 반복 시행을 할 경우 동일한 기능을 시연하게 된다. 본 논문에서 제안하는 동적 분석 기법은 유닛 테스트 케이스의 반복 실행에서 동일한 동작이 발생함에도, 반복에 따라 메모리 사용량이 지속적으로 증가하는 유닛 테스트 케이스를 메모리 블롯 의심 증상으로 추정하고, 해당 메모리 사용량 증가에 기여한 코드 요소를 결함 의심 요소로 판별하여 경보 정보를 사용자에게 제시한다.

그림 2는 제안하는 동적 분석 기법의 구성 요소와 동작 흐름을 보여준다. 본 기법은 사용자로부터 분석대상 프로그램(target program) 코드와 JUnit 상에서 작성된 유닛 테스트 스위트(test suite) 소스코드를 입력으로 받아 분석을 수행하며, 그 결과는 메모리 블롯을 발생시키는 것으로 의심이 되는 객체 정보(warnings)다. 테스트 스위트 T 에는 테스트 케이스 집합 $testcases(T)=\{t_1, t_2, \ldots, t_n\}$과 테스트 케이스의 실행에 필요한 일체의 부속코드(test fixture)로 구성된다. 이 때 모든 테스트 케이스 결과는 성공(pass)으로 정상적으로 동작 해야 하며, 실패(fail) 테스트 케이스는 분석에서 제외하고 사용해야 한다.

테스트 편집기 모듈(test instrumentor)은 입력 받은 테스트 스위트에서 각 테스트 케이스 t_i별로 (1) t_i가 k회 반복 실행되도록 테스트 케이스 구동코드를 추가하며, (2) 테스트 케이스 반복에 메모리 사용량 기록과 JHAT 을 통해 수집한 메모리 상태 정보를 출력하도록 탐침(probe)를 삽입한 n개의 수정된 테스트 스위트 집합 $\{T_1, T_2, \ldots, T_n\}$을 생성한다. T_i 는 T와 동일한 순서로 테스트 코드를 실행하되, i 번째 테스트 케이스 t_i와 관련된 코드는 k번 연속 반복 수행하게 된다.

테스트 실행 모듈(test executor)은 JUnit을 각각의 수정된 테스트 스위트 T_i 를 입력 받아 테스트 스위트를 실행하고, 그 결과 생성되는 JUnit 결과, 메모리 사용량 기록, JHAT (Java Heap Analysis Tool) [6]의 메모리 덤프 해석을 수집한다. JUnit 실행 결과는 각 테스트케이스 실행의 성공/실패 여부를 나타내며, 메모리 사용량 기록은 t_i가 최초 실행되기 직전, 매번 t_i실행이 마친 직후, 그리고 T_i 전체의 종료 직전에 테스트 실행 프로세스의 전체 메모리 공간과 해당 시점에서 가용(free memory) 메모리 양의 차를 구한 k+2 길이의 순열이다. 메모리 상태 기록(JHAT logs)은 테스트 스위트 종료 시점에 메모리 덤프를 통해 프로세스 공간에 존재하는 객체의 종류, 개수, 주소, 참조관계를 포착한 정보다.

런타임 데이터 분석 모듈(*runtime data analyzer*)은 테스트 실행에서 추출한 결과로부터 다음과 같은 분석을 수행한다:

- **분석1. 테스트 케이스 반복 실행의 유효성 판별**: 각 T_i 의 JUnit 결과를 분석하여 모든 테스트 케이스 결과가 성공인지 확인한다. 만약 실패한 테스트 케이스가 있을 경우, 테스트 케이스의 재실행으로 인해 동일한 동작이 반복되지 않고 의도하지 않은 다른 동작이 발생한 경우로 볼 수 있으므로, 동적 분석에서 제외한다.
- **분석2. 메모리 블롯 발생 유무 판별**: 분석1을 통과한 각 T_i 에 대하여, 메모리 사용량 측정 결과로 얻은 메모리 사용량 순열에 선형 회기(linear regression)를 사용해 추세선을 얻고, 추세선의 계수가 1000 byte/time이상이며, R^2 결정계수가 0.9 이상인 경우 메모리 블롯이 발생한 것으로 판별한다. 그 이외의 경우는 메모리 블롯이 발생하지 않은 것으로 이후 분석에서 제외한다.
- **분석3. 메모리 블롯 객체 판별**: 분석2에서 메모리 블롯이 발생한 것으로 판별된 각 T_i의 메모리 상태 기록을 분석하여 메모리 블롯에 기여한 것으로 의심되는 순서로 객체를 판별한다. JHAT으로 포착한 메모리 상태로부터 측정 시점의 프로세스 내에 유효하게 존재하는 객체, 각 객체가 생성(instantiation) 된 시점의 스택(stack trace), 그리고 각 객체의 멤버가 다른 객체를 참조하는 관계를 파악할 수 있다. 메모리 블롯에 기여한 객체를 파악하기 위해서, 메모리 상태 기록에 존재하는 모든 객체를 클래스가 같고 생성 시점의 스택(stack trace)이 같은 객체끼리 묶어서 객체군 집합을 정의하고, 각 객체군에 속한 객체의 수를 구하고, 객체군에 속한 객체 수가 많은 순서대로 정렬한다.

런타임 데이터 분석 모듈은 메모리 블롯 발생이 탐지된 각 테스트케이스에 의심도에 따라 정렬한 객체 정보를 분석의 최종 결과로 사용자에게 제공하여, 사용자가 분석 결과 결함의 기여가 큰 것으로 보이는 객체와 객체 관련 코드를 우선적으로 검토하도록 지원한다.

3.2 구현

제안한 기법에서 간단한 소스코드 편집과 JHAT 덤프 분석, 메모리 사용량 기록 분석은 Python 스크립트로 구현했고, 구조적인 소스코드 편집에는 JavaParser, 선형 회귀 분석에서는 SciPy 라이브러리를 활용해 계산하였다. 테스트 편집기가 삽입하는 탐침 코드에는 매 테스트 케이스 실행 직후 메모리 사용량을 정확히 측정하기 위해 `System.gc()`와 `System.sleep()`을 각각 호출하여 불용 메모리 수거를 유도 후, `Runtime.totalMemory()`, `Runtime.freeMemory()`를 호출하여 메모리 사용량을 측정하였다. 메모리 상태 기록을 얻기 위해서 테스트 실행 모듈에서는 JUnit을 구동할 때 JVM 옵션으로 종료시점에 메모리 덤프를 출력하도록 설정하였고, 테스트 실행 직후 JHAT을 호출해서 덤프를 해석하게 했다.

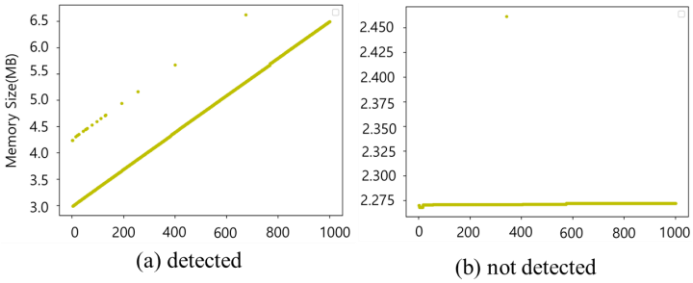


그림3. 메모리 블롯이 탐지된 실행(a)과 메모리 블롯이 탐지되지 않은 실행(b)에서 메모리 사용 추세 비교

4. Apache Commons VFS 대상 사례 연구

제안한 기법을 Apache Commons VFS에 적용하여 실제 결함을 검출하는 사례 연구를 수행하였다. 적용대상으로는 사례연구 수행 당시 최신 버전인 Apache Commons VFS 2.2-rc2가 사용됐다. 해당 프로젝트는 총 382개의 Java 파일로 구성 되어 있으며, Java 소스코드는 총 26376 LOC 규모이다. 프로젝트 내에는 JUnit으로 작성된 총 2341개 테스트 케이스(총 106개의 테스트 스위트)가 작성 돼 있는데, 이들은 정상적인 빌드 상황에서 모두 성공하는 테스트 케이스이다. 본 사례 연구에서는 2341개의 테스트 케이스를 각각 총 1000번 반복 실행하도록 설정하여 제안한 기법을 적용하였다. 실험은 Intel Core i5-6600 CPU와 64G DDR4 RAM이 설치된 Ubuntu OS환경에서 수행됐으며, 실행에는 OpenJDK 1.8를 사용하였다.

총 2341개의 테스트 케이스에 대해 동적 분석 기법을 적용한 결과, 총 3개의 테스트 케이스에 대해서 메모리 블롯 발생이 검출됐다. 나머지 2338 개의 테스트 케이스의 경우, 모두 반복 실행은 유효하게 실행 되었지만 메모리 블롯 발생이 탐지되지는 않았다. 그림3은 메모리 블롯이 검출된 `JunctionTests.testAncestor()` 실행의 경우와, 이와 반대로 실행 과정에서 메모리 블롯 발생이 검출 되지 않은 `UserAuthenticationDataTestCase.testCustomType()` 경우를 예시로 하여 반복 횟수에 따른 메모리 사용량 기록을 그래프로 표현한 결과다. 그림3-(a) 에서는 반복실행에 따라 메모리 사용량 증가가 뚜렷한 반면, 그림3-(b)에서는 사용량이 일정하여, 오류 발생 여부가 분명히 구별됨을 볼 수 있다.

표2는 메모리 블롯이 탐지된 3개의 테스트 케이스의 이름과, 1000번의 반복 실행 전 후 메모리 사용량 증가량, 선형 회귀 계수와 R^2 값을 나타낸다. 표2의 결과로 볼 때, 메모리 블롯이 탐지된 3개 경우에서는 테스트 케이스 반복에 따른 메모리 사용량의 선형적 증가가 분명하게 관찰됨을 확인할 수 있다.

탐지된 3개의 경보가 실제 결함에 의한 것인지 판별하기 위하여, 제시된 결함군 정보와 관련 소스코드를 정성적으로 분석하여 결함 유무를 분석하였다. 그 결과, `JunctionTets.testAncestor()` 케이스와 관련된 결함은 실제 VFS 라이브러리 코드 내에 `LocalFileSystem` 클래스에서 소유자 객체가 정리되는 과정에서 컨테이너 내에 불필요한 `WeakRefFileListener` 객체를 제거하지 않아 발생하는 결함임을 확인할 수 있었다. `JunctionTests.testAncestor()` 분석 결과로 도출된 180개의 객체군에서 `WeakRefFileListener`는 4번째로 가장 높은 의심에 위치하였다. `WeakRefFileListener`보다 의심도가 높은 3개의 객체군은 테스트 케이스의 특성 상 반복실행에 따라 정상적으로 추가 할당되는 자원에 해당하는 객체로, 검토 결과 메모리 블롯과 관련이 없는 것을 확인했다.

메모리 블롯 객체가 발생하는 원인을 분석하기 위해, 메모리 덤프로부터 `WeakRefFileListener` 객체군에 속한 객체의 참조 관계를 분석한 결과, `WeakRefFileListener` 객체가 `LocalFileSystem` 내 컨테이너에 한번 추가가 된 이후 테스트 종료 전까지 제거되지 않음을 확인할 수 있었다. 해당 객체가 불필요한 객체인지 판별하기 위해, `JunctionTests.testAncestor()`가 검사하는 시나리오를 분석한 결과, 테스트 케이스 구동 코드에서 `WeakRefFileListener` 객체와 관련된 다른 객체는 제거되는 반면 `WeakRefFileListener` 객체가 제거되지 않음을 알 수 있었다. 메모리 블롯 여부를 확인하기 위해, 관련 객체의 제거 시 `WeakRefFileListener` 객체가 컨테이너에서 제거되도록 코드를 수정하자, 테스트 케이스는 여전히 성공하면서도 메모리 블롯 증상이 사라짐을 확인할 수 있었고, 이를 바탕으로 분석 결과가 실제 메모리 블롯 문제를 탐지하였음을 확인할 수 있었다.

표2. 메모리 블롯 검출 결과

Test case	Mem증가	선형회귀계수	R^2
JunctionTets.testAncestor()	2.32MB	2370.4	0.979
CustomRamProviderTest. testReadyEmptyFileByByte()	2.26MB	3605.4	0.969
FileLockTestCase. testResolveAndOpenCloseContent	50.00MB	50952.2	0.999

`CustomRamProviderTest.testReadyEmptyFileByByte()`와 `FileLockTestCase.testResolveAndOpenCloseContent()` 탐지 결과, 역시 테스트 케이스 내에서 불필요한 메모리 사용을 발생 시키는 메모리 블롯과 연관되어 있음을 확인하였다. `JunctionTets.testAncestor()` 경우와 달리, 이 2개 경우는 결함이 메모리 블롯이 테스트 케이스 코드에 위치하므로, 상대적으로 위험도가 낮은 결함으로 볼 수 있다. 다만, 테스트 케이스 코드는 개발자들에게 표준적 용례를 나타내는 용도로 쓰인다는 측면에서, 해당 결함 역시 수정되어야 하는 실제적 결함으로 볼 수 있다.

5. 결론

본 논문에서는 Java 프로그램의 메모리 누실 오류의 한 종류인 메모리 블롯의 발생을 효과적으로 탐지하기 위해 유닛 테스트 케이스 반복 실행을 통한 새로운 동적 분석 기법을 제안 하였고, Apache Commons VFS를 대상으로 한 사례연구를 통해 제안한 기법의 결함 검출 효용성을 논의하였다. 제안한 기법을 Apache Commons VFS에 적용한 결과, 라이브러리 코드에서 1개의 메모리 블롯 결함과 테스트 코드에서 2개의 메모리 블롯 결함을 신규로 탐지할 수 있었다.

향후에는 Apache Commons VFS 이외에 다양한 Java 프로그램에 개발한 동적 분석 기법을 적용하여 일반적 유용성을 실험할 계획이다. 또한 본 기법을 유닛 테스트 케이스 자동 생성과 연계하여, 개발자로부터 주어진 유닛 테스트 케이스가 없거나 부족한 경우에도 능동적으로 테스트 케이스를 생성하며 메모리 블롯을 검출하는 방법을 연구할 계획이다.

참조문헌

[1] N. Mitchell et al., Four Trends Leading to Java Runtime Bloat, IEEE Software, Jan/Feb, 2010
[2] G. Xu et al., Go with the Flow: Profiling Copies to Find Runtime Bloat, PLDI, 2009
[3] M. D. Bond and K. S. McKinley, Bell: Bit-Encoding Online Memory Leak Detection, ASPLOS, 2006
[4] M. Jump and K. S. McKinley, Cork: Dynamic Memory Leak Detection for Garbage-collected Languages, POPL, 2007
[5] Commons Virtual File System, Apache Software Foundation, https://commons.apache.org/proper/commons-vfs
[6] JHat – Java Heap Analysis Tool, Oracle https://docs.oracle.com/javase/7/docs/technotes/tools/share/jhat.html

가중치 기반 센서 데이터 획득 효율성 향상 기법

송명호 [1,O], 김수동 [2,*]

숭실대학교 컴퓨터학과 [1], 숭실대학교 소프트웨어학부 [2]

{songmho, sdkim777}@gmail.com

Method for Weight based Sensor Data Acquisition Efficiency Improvement

Myeong Ho Song[1,O], Soo Dong Kim[2,*]

Department of Computer Science and Engineering, Soongsil University[1],

School of Software, Soongsil University[2]

요 약

현재 가정, 산업 등 여러 분야에서 수많은 센서 간의 통신을 이용하고 있다. 이 센서 간 통신에서 얻게 되는 효율성들과 가중치를 이용하여 데이터 획득 효율성을 구하게 된다. 이 때, 고정된 가중치를 이용할 경우 데이터 획득 방식, 시간의 흐름, 낮은 효율성 값에 대해 고려가 되지 않는 문제가 있다. 본 논문에서는 가중치를 자율적으로 변경하고 최적화할 수 있는 기법을 제안한다. 제안된 기법을 이용하여 고정된 가중치로 생기는 문제점을 해결하고 항상 최적의 가중치를 유지할 수 있는 효과를 얻을 수 있다.

1. 서 론

현재 가정, 산업 등 여러 분야에서 수많은 센서 간의 통신을 이용하고 있다 [1][2][3]. 이때 센서 간의 데이터 획득에 대한 효율성은 생산성, 보안, 안전 등에 직접적으로 연관되므로 매우 중요한 부분이다.

데이터 획득에 쓰이는 센서들은 데이터를 생성하는 데이터 소스(Data Source)와 데이터를 획득하는 데이터 싱크(Data Sink)로 나뉜다. 획득 효율성은 데이터 소스가 생성한 데이터를 데이터 싱크에서 획득한 데이터양, 데이터를 획득하는데 소요된 시간 또는 소요된 리소스 등 여러 부분을 통해 효율성을 얻을 수 있다 [4][5]. 데이터 획득 방식이나 통신하는 센서의 개수, 사용하는 분야 등에 따라 추가적인 부분을 통해 효율성을 얻는 것이 가능하다. 측정된 효율성들과 가중치를 가지고 현재 센서 간의 데이터 획득 효율성에 대한 메트릭(Metric)을 구할 수 있다.

가중치는 효율성에 일대일로 할당하여 구해진 효율성을 가중치를 통하여 스케일링(Scaling) 과정을 거치게 된다. 이 과정의 결과로 최종 효율성 결과치를 얻게 된다.

이때 가중치가 고정으로 설정이 될 시, 아래와 같은 이슈가 생기게 된다.

- 센서 간 데이터 획득 방식이 고려되지 않는 이슈
- 한가지 데이터 획득 방식에서 시간의 흐름을 반영하지 못하는 이슈
- 낮은 값으로 측정된 효율성에 대한 이슈

본 논문에서는 데이터 획득 효율성에서의 가중치가 고정일 때 생겼던 이슈들에 대한 해결법을 제안한다. 또한 가중치에 대한 데이터 분석을 통해 자율적으로 설정, 항상 최적의 가중치를 유지하는 기법을 제시한다.

2. 센서 측정치 획득 효율성의 가중치

서론에서 제시한 것과 같이 센서 측정치 획득 효율성은 획득한 데이터양, 소요시간, 소요 리소스와 데이터 획득 방식이나 통신하는 센서의 개수 등에 따라 증가할 수 있다. 효율성은 0에서 1 사이의 값을 갖고 1에 가까울수록 높은 효율성을 나타내고 0에 가까울수록 낮은 효율성을 나타낸다.

메트릭에서 사용되는 가중치는 효율성에 대한 중요도를 나타낼 수 있으며 모인 효율성을 가지고 스케일링을 하여 최종 효율성이 0에서 1사이의 값을 갖도록 하는 역할을 한다. 한 가중치 w_x 는 0에서 1사이의 값을 갖고, 수식 1과 같이 모든 가중치의 합은 1이 되어야 한다.

$$\sum_{x=1}^{n} w_x = 1 \quad (0 \le w_x \le 1) \qquad \text{수식 } 1$$

가중치와 효율성을 통해 얻은 데이터 획득

* 교신저자 (Corresponding Author)

효율성(Data Acquisition Efficiency, DAE)은 수식 2와 같이 0에서 1사이의 값을 갖고 각 가중치(w_x)와 효율성(a_x)의 합의 곱으로 나타낸다.

$$\mathrm{DAE} = \sum_{x=1}^{n} w_x \times a_x (0 \le DAE \le 1) \qquad 수식/2$$

수식 2를 통해 얻은 DAE는 다양한 데이터 획득 방식과 여러 범위에 속하는 효율성을 통하여 얻게 된다. 따라서 고정된 가중치를 모든 효율성에 적용하여 연산을 진행할 경우, 효율성에 대한 중요도나 특정 효율성 값들을 DAE에 반영하기 어렵게 된다. 이를 통해 고정된 가중치에 대한 이슈는 아래와 같다.

첫째, 센서 간 데이터 획득 방식이 고려되지 않는다. 다양한 센서 간 데이터 획득 방식이 존재한다. 이 방식들은 각각의 고유 특징을 갖고 있고 이에 따라 다른 효율성보다 영향을 많이 받는 효율성이 존재한다. 예를 들어 Pulling 방식의 경우, 데이터 싱크가 필요한 시점에 데이터를 요청, 획득하므로 데이터 획득 유무에 대해 크게 영향을 받는다. 따라서 Pulling 방식은 데이터 획득 량에 대해 영향을 많이 받는다. 하지만 고정된 가중치를 사용하게 될 경우, 이러한 특징을 반영하지 못한다.

둘째, 한 획득 방식에서 시간의 흐름이 고려되지 않는다. 한 획득 방식이 지속되면 누적된 결과로 인해 어떠한 특징이 부각된다. 예를 들어 Pushing 방식의 경우, 데이터 소스에서 변경된 값을 계속하여 보내주는 형식으로 이 방식을 계속하게 되면 데이터 싱크에서의 리소스 사용량이 증가하게 된다. 하지만 고정된 가중치를 쓰게 될 경우, 획득 방식이 계속 사용되어도 변화가 없으므로 특징을 반영하지 못한다.

셋째, 낮게 측정된 효율성에 대한 처리가 미비에 대한 점이다. 효율성 측정 중 일부 효율성이 낮게 나왔음에도 불구하고 다른 효율성이 높게 나와 결과는 좋게 나오는 경우가 있다. 하지만 특정 효율성이 낮게 나온다는 것은 현재 측정 방식이 좋은 효율성을 갖지 않는다는 것을 나타낸다. 고정된 가중치를 쓰게 될 경우, 이러한 낮게 측정된 효율성에 대해 대처를 하지 못한다.

고정된 가중치를 사용할 경우, 위의 이슈를 반영하지 못한 DAE을 얻게 된다. 따라서 상황에 따라 자율적으로 설정이 되는 가중치가 필요하다.

3. 가중치 자율 최적화 알고리즘

본 장은 2장에서 제시된 이슈를 해결하기 위한 가중치 자율 최적화 알고리즘을 제안한다. 일반적인 케이스를 고려하여 데이터 소스가 생성한 데이터를 데이터 싱크에서 획득한 양(Data Acquisition Rate, DAR), 데이터를 획득하는데 소요된 시간(Acquisition Time Efficiency, ATE), 데이터를 획득하는데 소요된 리소스 양(Acquisition Resource Efficiency, ARE)을 효율성에 대한 메트릭 연산에 사용한다. 따라서 DAR의 가중치(wDAR), ATE의 가중치(wATE), ARE의 가중치(wARE)를 사용한다. 이 세 가중치의 합은 수식 1에 의해 1이 된다. 또한 데이터 획득 방식은 Pulling, Pushing, Notify & fetch을 사용한다. 이 알고리즘은 센서에서 데이터를 획득하는 전처리 컴포넌트에서 획득된 데이터를 가지고 효율성을 연산하는 컴포넌트에서 사용이 된다.

획득 방식의 특징을 드러내는 가중치를 적용하기 위해 3가지 효율성 중 1가지를 설정한다. Pulling은 데이터 싱크(Data Sink)가 직접 데이터를 요청 획득하므로 DAE는 DAR에 영향을 많이 받아야 한다. Pushing의 경우, 데이터 소스(Data Source)가 생성된 데이터를 데이터 싱크(Data Sink)의 상태를 상관하지 않고 전송하게 된다. 따라서 Pushing은 데이터 획득이 계속될수록 리소스에 대한 영향을 많이 받으므로 DAE는 ARE에 영향을 많이 받아야 한다. Notify & fetch는 일정 fetch rate 안에서의 데이터만 사용한다. 따라서 이 방식의 데이터 획득은 데이터 획득 시 소요된 시간에 영향을 받게 된다. 그러므로 Notify & fetch의 DAE는 ATE에 영향을 많이 받아야 한다.

각각의 방식에서 가장 영향을 미치는 효율성과 나머지의 관계를 식 3과 같이 표현한다.

$$\beta < \alpha \le (\beta + \gamma)\ (\beta = \gamma) \qquad 수식/3$$

가장 큰 영향을 미치는 효율성(α)은 나머지 효율성(β, γ)의 합과 같거나 작지만 나머지 효율성 하나보단 항상 큰 값을 유지한다. 이때 나머지 효율성은 같은 값으로 설정한다. 수식 1과 3을 통해 가장 큰 영향을 미치는 효율성은 1/3에서 1/2 사이의 범위를 나머지 효율성은 1/4에서 1/3 사이의 범위를 갖게 된다.

$$\frac{1}{3} < \alpha \le \frac{1}{2} \qquad 수식/4$$

$$\frac{1}{4} < \beta \le \frac{1}{3}\ (\beta = \gamma) \qquad 수식/5$$

수식 4, 5의 결과를 통해 획득 방식 별로 가장 큰 영향을 미치는 효율성의 가중치를 0.4 나머지 효율성의 가중치를 0.3으로 할당한다.

표 1. 초기 가중치 설정 알고리즘

알고리즘 1. 데이터 획득 방식에 따른 초기 가중치 설정 알고리즘

```
1  BEGIN
2  function set_init_weight(scheme)
3  Input: a string scheme
4  Output: None
5     switch (scheme)
6        case "Pulling":
7           wDAR = 0.4
```

```
8           wATE = 0.3
9           wARE = 0.3
10          break
11      case "Pushing":
12          wDAR = 0.3
13          wATE = 0.3
14          wARE = 0.4
15          break
16      case "NotifynFetch":
17          wDAR = 0.3
18          wATE = 0.3
19          wARE = 0.4
20          break
21  END
```

알고리즘 1은 데이터 획득 방식에 따른 초기 가중치 설정 방식에 대한 알고리즘이다. 이 중 Pulling 방식을 예시로 들자면, Pulling 방식에서 가중치 설정하는 방법의 가장 큰 영향을 미치는 효율성인 DAR의 가중치를 0.4로 할당, 나머지 효율성의 경우 0.3을 할당한다.

수식 4, 5의 범위를 통해 효율성 메트릭 연산을 진행하면서 가중치를 변화시켜준다. 가장 큰 영향을 미치는 효율성을 0.4, 나머지는 0.3씩 적용하여 초기 가중치를 설정함으로써 초기 가중치부터 효율성 별 특징을 드러내게 한다. 또한 가중치를 일정 주기 동안 일정량 증가 또는 감소를 시킨다. 여기에서의 주기(cal_count)는 획득 방식에서 진행된 메트릭 연산의 횟수의 평균을 의미한다. 따라서 가중치의 증감 양은 해당 가중치의 최댓값과 최솟값의 차이를 주기로 나눈 값을 의미하게 된다.

표 2. Pulling 방식 초기 가중치 산정 알고리즘

알고리즘 2. Pulling 방식에서 가중치 초기화와 연산에 따른 증감률 적용 알고리즘

```
1   BEGIN
2   function set_pulling_weight()
3   Input: None
4   Output: None
5     if wDAR >= 0.4 and wDAR < 0.5    then
6       if DAR > (threshold/2) and ATE>
            (threshold/2) and ARE>(threshold/2) then
7         wDAR = wDAR + (0.5-0.4) / cal_count
8         wATE = wATE - (0.3-0.25) / cal_count
9         wARE = wARE - (0.3-0.25) / cal_count
10  …
11  END
```

알고리즘 2은 Pulling 방식일 때 가중치 초기화 및 데이터 획득 메트릭 연산에 따른 증감률을 적용하는 방식을 나타낸다. 7~9줄을 보면 5번째 줄의 조건을 만족하는 범위에서 매 연산 때 일정 증감률을 가중치에 더해주고 있다.

데이터 획득 효율성 획득 과정 중 데이터 획득 효율성은 기준점(Threshold) 이상의 값을 나타내지만 한 가지 또는 두 가지의 효율성에서 낮은 값을 나타내는 경우가 있다.

낮은 효율성에 대한 기준을 데이터 획득 효율성에 대한 기준점(Threshold)의 절반보다 낮음으로 한다. 이 때 낮은 효율성에 대한 경우는 영향을 많이 주는 효율성의 포함 유무에 따라 총 4가지 경우가 생긴다. 수식 4, 5의 범위를 초과하지 않는 조건에서 가중치의 증감률을 수정한다.

첫째, 영향을 많이 주는 효율성이 낮은 데이터 효율을 나타내는 경우다. 이 경우에는 일반적인 경우와 같은 증감률을 적용한다.

둘째, 이 외의 효율성 중 하나가 낮은 데이터 효율을 나타내는 경우이다. 이 경우에는 해당 가중치에 대한 감소율을 0으로 바꾼 후, 큰 영향을 미치는 효율성의 가중치를 절반으로 감소시켜 적용한다.

셋째, 영향을 많이 주는 효율성과 이 외의 효율성 중 하나가 낮은 데이터 효율을 나타내는 경우이다. 해당 이 외의 효율성의 감소율을 0으로 변경 후, 나머지 이 외의 효율성의 감소율을 2배 증가시켜 적용한다.

넷째, 이 외의 효율성 2가지가 모두 낮은 데이터 효율을 나타내는 경우이다. 이 경우 각각 일반적인 경우에서 증가율에 해당하는 값을 감소율로, 감소율에 해당하는 값을 증가율로 하여 이 외의 효율성 2가지의 가중치가 증가하도록 한다.

표 3. 효율성 증가를 위한 Pulling 방식 가중치 산정 알고리즘

알고리즘 3. Pulling 방식에서 효율성이 낮은 데이터 효율에 가중치 적용 알고리즘

```
1   BEGIN
2   // 모든 효율성이 (threashold/2)보다 높을 경우
3   …
4     if DAR<(threshold/2) then
5       if ATE<(threshold/2) then
6         wDAR = wDAR+((0.5-0.4)/cal_count)
7         wATE = wATE
8         wARE = wARE-((0.3-0.25)/cal_count)*2
9       else if ARE<(threshold/2) then
10        wDAR = wDAR+((0.5-0.4)/cal_count)
11        wATE = wATE-((0.3-0.25)/cal_count)*2
12        wARE = wARE
13      else
14        wDAR = wDAR + (0.5-0.4) / cal_count
15        wATE = wATE - (0.3-0.25) / cal_count
16        wARE = wARE - (0.3-0.25) / cal_count
17      endif
18    else
19      if ATE<(threshold/2) then
20        if ARE<(threshold/2) then
21          wDAR = wDAR-(0.5-0.4)/cal_count
22          wATE = wATE+(0.3-0.25)/cal_count
22          wARE = wARE+(0.3-0.25)/cal_count
23        else
24          wDAR = wDAR+((0.5-0.4)/cal_count)
25          wATE = wATE
```

```
26              wARE = wARE-((0.3-0.25)/cal_count)*2
27            endif
28          else if ARE<(threshold/2) then
29            wDAR = wDAR+((0.5-0.4)/cal_count)
30            wATE = wATE-((0.3-0.25)/cal_count)*2
31            wARE = wARE
32          endif
33        endif
34      END
```

알고리즘 3는 Pulling 방식일 때 효율성이 낮은 데이터 효율성에 대한 가중치 부여를 나타낸다. 5~9번째 줄은 DAR과 ATE 가 낮은 값을 가질 때이므로 위에 제시한 4가지 경우 중 세 번째에 해당된다. 14~17번째 줄의 경우 DAR만 낮은 값을 가질 때이므로 4가지 경우 중 첫 번째에 해당된다. 20~23번째의 경우는 ATE와 ARE가 낮은 경우로 네 번째 경우에 해당되고 32~35번째 줄은 ATE 하나의 값에 대해서만 낮은 경우로 두 번째 경우에 해당된다.

4. 실험 및 평가

본 장에서는 3장의 알고리즘에 대한 실험을 수행하였다. 2장에서 제시한 3가지 문제점에 대해 실험을 진행하였다.

데이터 획득 방식이 설정되거나 변경될 때 해당 데이터 획득 방식에 맞춰 영향을 많이 받는 효율성의 가중치에 0.4를 나머지 가중치에는 0.3을 부여하는 것을 볼 수 있었다.

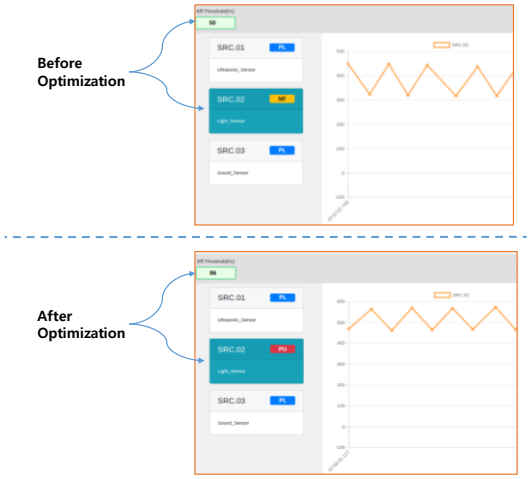


그림 1. 데이터 획득 방식에 변경 및 효율성 향상 결과

그 후, 모든 효율성이 기준점(Threshold)의 절반 이상일 때 설정된 주기 동안 각각의 가중치들이 증가 또는 감소함을 볼 수 있었다. 이렇게 구하여진 가중치를 적용하여 그림 1과 같이 데이터 획득 효율성이 증가하게 되었다.

낮은 효율성을 보이는 일부 효율성이 존재할 경우 기준점(Threshold)의 절반 이하로 설정된 효율성이 어떤 종류인지 와 몇 개인지에 따라 증감률이 변경이 되어 적용이 되었다. 이를 통해서 일부 효율성이 낮은 값을 갖는 경우, 이 효율성이 데이터 획득 효율성에 끼치는 영향이 정상적인 경우보다 커지는 것을 볼 수 있었다.

5. 결 론

본 논문에서는 센서 측정치 효율성의 가중치가 고정적일 때 생기는 문제점을 제시하고 이러한 문제점을 해결하기 위해 가중치의 자율적 설정이 필요성과 그에 따른 가중치 자율 최적화 알고리즘을 제안하였다.

제안한 알고리즘을 사용하여 센서 측정치 획득 시, 데이터 획득 방식에 따라, 데이터 획득 효율성 연산이 계속됨에 따라, 낮은 효율성을 나타내는 일부 효율성에 따라 가중치를 자율적으로 최적화시켜 현재 상황에 맞게 효율성을 적용, 알맞은 데이터 획득 효율성을 구할 수 있음을 확인하였다.

6. Acknowledgement

이 논문은 2017 년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행한 연구임(NRF-2015R1A2A2A01004078).

7. 참고문헌

[1] C. Beck, H. J. Ng, and et al, “Battery-free wireless sensors for industrial applications based on UHF RFID technology,” *IEEE Sensors Journal*, vol. 16, no. 17, pp. 6566-6578, July 2016.

[2] H. Elayan; R. M. Shubair; and A. Kiourti, “Wireless sensors for medical applications: Current status and future challenges,” *In Proceedings of the 11th European Conference on Antennas and Propagation (EUCAP 2017)*, pp. 2478-2482, March 2013.

[3] A. K. Triantafyllidis, C. Velardo, and et al, “A Survey of Mobile Phone Sensing, Self-Reporting, and Social Sharing for Pervasive Healthcare,” *IEEE Journal of Biomedical and Health Informatics*, vol. 21, no. 1, pp. 218-227, September 2016.

[4] J. Lee and S. Kim, “IoT Contexts Acquisition with High Accuracy and Efficiency,” *In Proceedings of IEEE 11th World Congress on Services 2015 (SERVICES 2015)*, pp. 9-16, June 2015.

[5] A. Gomez, M. Magno, and et al, “Precise, Energy-Efficient Data Acquisition Architecture for Monitoring Radioactivity Using Self-Sustainable Wireless Sensor Nodes,” *IEEE Sensors Journal*, vol. 18, no. 1, pp. 459-469, June 2017.

An Observation of Use Case and Misuse Case in System Theoretic Process Analysis

Anit Thapaliya, Gihwon Kwon

Department of Computer Science, Kyonggi University

{anitdgret, khkwon}@kgu.ac.kr

Abstract

Use case diagram is a representation of user's interaction which can recognize the impact on the use cases and involved actors. In this paper, use case and misuse cases are used to elaborate system theoretic process analysis (STPA). We generate a set of basic software safety requirements from misuse cases based STPA. This provides a pathway where system requirements can be identified as misuse cases and analyze their connection in terms of failure. The results of misuse cases analysis describe the basic inputs for traditional STPA. Unlike to traditional STPA inputs where hazards and accident are taken from real world scenarios, brainstorming and domain knowledge, but this paper present a framework to generate hazard and accident from misuse case diagrams. A case study of railway level crossing system is also provided to verify the applicability of proposed method.

1. Introduction

Software safety doesn't have a clear and concise definition in nature, the portion of software in safety critical system is deep-rooted. Safety related problems are both manually and automatically handled in safety engineering process. This inherits the difficulties in verifying the safety aspect of the software system. In other word, both manual and automatic safety analysis are more based on the analyst domain knowledge and experiences. This is the reason why there is no guarantee of fully safe software system.

In the modern computerized era of science and technology, the safety aspect of critical systems is normally handled by using reliability and system theory-based safety analysis. These analysis techniques are based on the informal specification of the system [1].

In addition, the design and development of safety critical system requires a joint effort of system engineering, software engineering as well as safety engineering. To be more specific in software and safety engineering, safety analysis techniques requires system descriptions, general assumptions and requirements from the stakeholders. This information provides the basic inputs for safety analysis.

This research was supported by the MSIP (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2017-2015-0-00445) supervised by the IITP (Institute for Information & communications Technology Promotion)

In traditional techniques such as fault tree analysis (FTA), failure mode and effect analysis (FMEA) the input parameters are taken from the system description along with requirements, but in modern techniques such as system theoretic process analysis (STPA) uses hazards and accident as input parameter [2]. These hazards and accident are derived manually from the system description using the domain knowledge and expertise of analyst.

In this paper, we present misuse case-based safety analysis through system theoretic process analysis. This process includes misuse case diagram (MUC) generated from the system description, requirements and assumptions made before the safety analysis. Use case diagrams can describe the reliability of software system through a sequence of actions that required in examining functional requirements [3].

Misuse case diagrams are an extension of use care diagrams which traditionally used to generate security requirements in software system [4]. This is because UML use case diagrams are not suitable for eliciting non-functional requirements such as security. This paper presents an experimental use of misuse case diagrams in safety analysis. Similarly, the outputs of misuse case diagrams are considered as an input to STPA based analysis.

In other word, this paper aims to generate inputs for STPA analysis from misuse case diagrams. Misuse case requirements modeling includes the all the function that the system should not allow, all the

miss-actor who is responsible for the misuse cases etc. In other word, traditional STPA uses system requirements directly as an input, but this paper focuses on requirements modeling using misuse case diagram before the actual STPA analysis. We investigate how system requirements can be modeled into STPA inputs to generate software safety requirements.

The rest of the paper is structured as follows: Section 2 reviews background of proposed concepts and related work. Section 3 presents the proposed method for the experiment in details. Section 4 presents the experimental case study, and section 5 presents the discussion about the proposed method from the experiment. Section 7 concludes the paper and indicates some directions for future work.

2. Background

Unified modeling language (UML) is a standardized language for modeling object-oriented software. UML can describe two aspects of software system such as static and dynamic structure [5]. Use case diagram, class diagram is usually used for describing the static structure of the system including the characteristic of the software system. Activity diagrams and interaction diagram is usually used for describing the dynamic structure of the system.

2.1 Misuse case

Being an extension of UML use case diagram, misuse case diagrams are widely used for eliciting security requirements in software system. A misuse case is triggered by the actor hostile to the system. It describes the threats posed by a hostile actor to the system including the user errors and faults [4, 6].

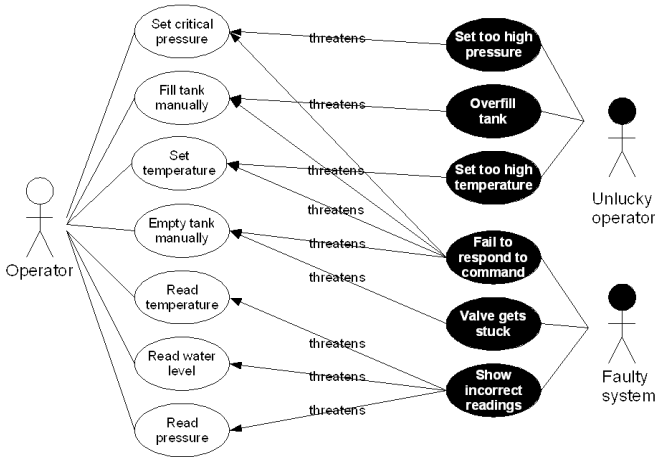


Fig. 1 Example of Misuse case

The above Fig. 1 represents the overview of misuse case diagram [4]. In the figure, the normal use cases are represented in oval shape without color, whereas the misuse cases are represented by color oval with the corresponding actors who are responsible for the misuse cases. The normal use cases are very effective in modeling the user requirements in software development. The misuse cases represent the threat or damage imposed by the faulty actors in the system.

2.2 System theoretic process analysis

System theoretic process analysis (STPA) is a modern safety analysis technique originally proposed by Leveson to identify the causal factors leading to possible accident in system theoretic accident model (STAMP). It involves identification of accidental scenarios based on the design error, requirements fault and other human related factors.

The major goals of STPA is to address the limitation imposed by traditional hazard analysis techniques in the field of modern complex systems. STPA is a systematic approach which focuses on the view of safety control issues [7]. In other word, STPA identify all the potential inadequate control of system which could lead to the possible accident in the future. This is because a hazardous state occurs because of poor system level thinking such as poorly performed control actions for safety, unsafe control actions provided [8]. Fig. 2 describes the foundation of STPA analysis based on control theory with process model [9].

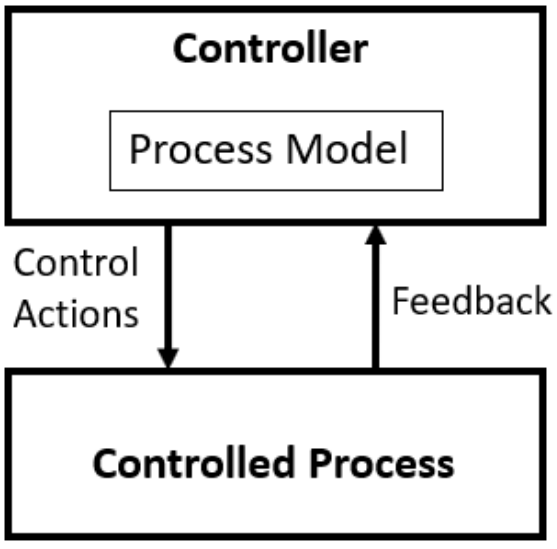


Fig. 2 STPA Overview

3. Proposed Methodology

Generating software safety requirements always starts with hazard analysis followed by risk assessment. In safety software development life cycle, various methods are proposed for hazard identification and risk assessment. Yet, the basics of these methods are based on historical data, brainstorming and expert domain knowledge. This paper presents a methodology to generate software safety requirements from system specification using misuse case diagrams and system theoretic process analysis.

Traditionally, safety analysis technique such as FMEA, FTA starts with system requirements in addition with several assumptions, experiences and domain knowledge. Similarly, STPA also uses hazards and accidents from domain knowledge. But, this paper aims to introduce misuse case diagrams before the actual safety analysis. From the misuse case, we generate a list of hazards and accident as an input for STPA analysis.

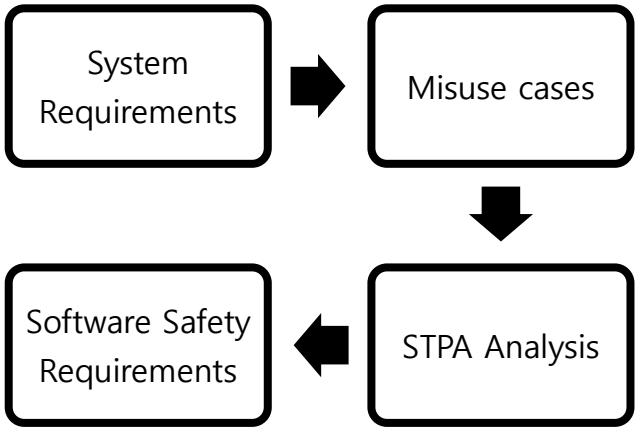


Fig. 3 Proposed Methodology

The above Fig. 3 describes the proposed methodology of safety analysis. We suggest four different steps to generate software safety requirements starting from system requirements. In step 1, we describe the system requirements of proposed system which includes user's requirement, safety plans, general guidelines and standards etc. In step 2, we generate misuse case diagram from system specification. This diagram reflects two different aspect of requirements such as normal use cases for describing functional characteristic of system and misuse cases for describing safety related characteristic of system. This safety related characteristic includes those factors responsible for the misuse cases. For example, those behaviors that the system is not intended to do triggered by some errors or faulty system or external users. We evaluate misuse case diagram to generate hazards and accident. MUC is more useful to analyze failure mode related to user's interaction in the system [10].

In addition, MUC enforce the possible accident in the future if specific control mechanism doesn't exist in the system. Hence, the outputs from MUC concentrates on system safety and high-level hazards or risks. In step 3, the actual STPA process starts with hazardous behaviors by identifying unsafe control actions, followed by the hierarchical safety control structure. This process includes modeling the original control structure diagrams with process model. To be more specific, we select a set of unsafe control actions applicable in railway system to identify hazardous control actions root causes. After analyzing all the hazardous control action and their corresponding accidental scenario's software safety requirements are generated in step 4.

4. Case Study

In this paper, we demonstrate our approach on the case study of railway level crossing system. Level crossing is an intersection point between a railway line and a road or a path. In other words, level crossing is a term used for describing a section where railway track crosses the roadway.

This paper case study is more focused on the system requirements of railway level crossing system. Among the different types of level crossing accidents in the world, like train – vehicles collision, train – pedestrian's collision, this case study only focuses on the accidental scenarios of train and vehicles collision. This case study is concerned with a sensor-based crossing controlling system used in the intersection area between the track and the road as illustrated in Fig. 4.

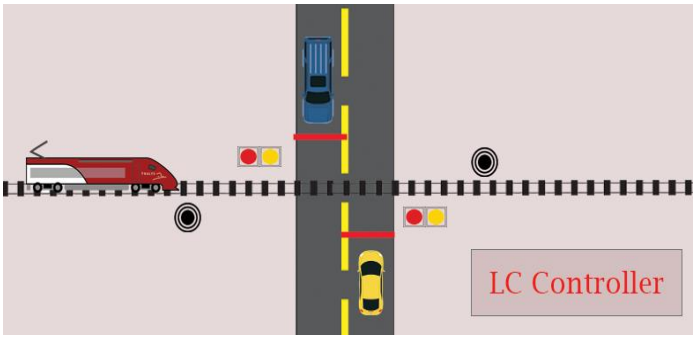


Fig. 4 Level Crossing System Overview

This case study consists of two sensors, two gates and one controller which control the overall working mechanism of the level crossing system. Sensor is used to detect the train in the level crossing area and the gates are used as the barriers for the roadway to block the entry of vehicles in the track. Similarly, the lights, red and yellow, are for the roadway traffic indicating track busy (red) and track free (yellow). Additionally, there are some basic considerations for the case study in terms of requirements such as the reliable configuration of level crossing gate and lights, reducing the accidental scenarios through the reliable safety control mechanism etc. With these generation system specification, we draw a misuse case diagram for level crossing system as mentioned in Fig. 5. This diagram consists of two different aspects of system, one as a normal requirement modeling based on the functionalities and second as misuse case-based requirement modeling with possible failure in the system.

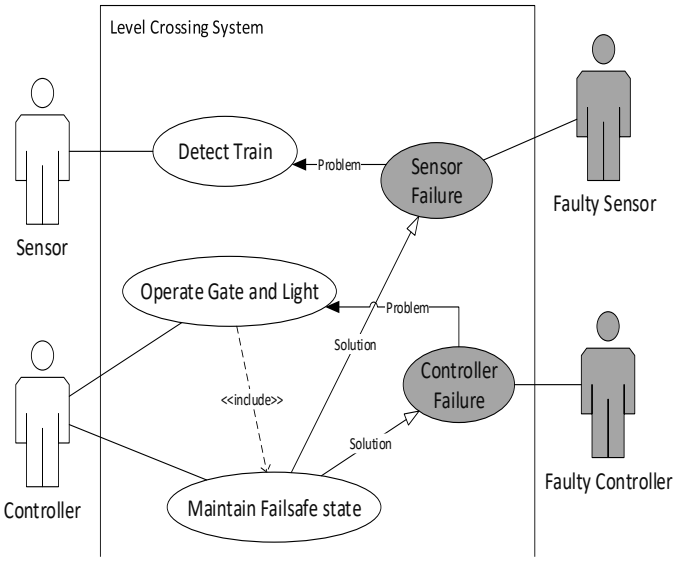


Fig. 5 Misuse case Diagrams

In the left-hand side of the figure, we can see major actors responsible for the normal operation i.e. uncolored use case in Fig. 5. On the other hand, there are faulty actors responsible for possible failure in the system. In general, our consideration focuses on the statement that a major actor in the use case diagrams will eventually be a faulty actor due to internal or external factors in the system. Similarly, the light black coloured misuse case imposed a threat to normal use cases which could eventually lead to system failure. During misuse case-based requirement modeling, we consider a failsafe use case as a mitigation for misuse case. Moreover, a failsafe use case diagram acts as a safety constraint for level crossing system.

S.N.	Use case Name	Description
1	Detect train	Whenever the train approach, sensor is activated and forward the train status to controller
	Precondition: sensor ready and train approaching	
	Post condition: forward train status to controller	
2	Operate gate and light	Whenever the controller receives train status, the controller operate gate accordingly
	Precondition: receive train status	
	Post condition: open/close the gate and turn light red/yellow	

Tab. 1 Use case Specification

In Tab. 1 we explain the first aspect of misuse case diagram of Fig. 5 using use case specification. This table describes the normal scenarios of level crossing system. Similarly, Tab. 2 describes the second aspect of failure propagation and its effects in Fig. 5. From the table, we investigate misuse cases such as sensor failure and controller failure using use case specification. In general, use case and misuse case specification contain actors for responsible for the corresponding use cases, pre and post condition of related use cases with simplified description.

S.N.	Misuse case Name	Description
1	Sensor failure	In case of sensor problem, this misuse case provides a threat to detect train
	Precondition: faulty sensor or sensor not ready	
	Post condition: unable to detect the train [Hazard 1]	
2	Controller failure	In case of controller failure, this misuse case provides threat to operate gate and light safely
	Precondition: faulty controller or controller not ready	
	Post condition: unable to operate gate and light safely [Hazard 2]	

Tab. 2 Misuse case Specification

From Tab. 2 we identify hazards from post condition of each misuse cases. In fact, these hazards are the outcomes of each faulty actors of the system. The identified hazards are mentioned in Tab. 3 below.

S.N.	Hazards
H1	Unable to detect train
H2	Unable to operate gate and light

Tab. 3 List of Identified Hazards

This identified hazards list works as an input for next STPA analysis. After identifying hazards, we evaluate failsafe use case diagrams to generate initial safety constraints for those hazards identified in misuse case specification. The precondition at this point those misuse cases based on faulty actors. And the post condition imposed the required safety actions for those misuse cases.

S.N.	Use case Name	Description
3	Maintain Failsafe state	As a solution for faulty sensor's and controller's threats, this use case prevents the system from level crossing accident i.e. Train and Vehicle collision [Accident 1]
	Precondition: faulty sensor or faulty controller related misuse cases	
	Postcondition: If unable to detect the train or the case with controller failure, close gate and turn light red [Safety Constraint]. Else, open gate and turn light yellow accordingly when train is not approaching.	

Tab. 4 Use case Specification for Failsafe

From this failsafe use case specification in Fig. 4, we identify accident from the description and safety constraints from the post condition. This safety constraints further explain in terms of control action in the next steps of proposed methodology. In fact, safety constraint aims to prevent the accidental scenarios defined in the case study.

Considering the viewpoint of proposed methodology, the outputs of misuse case diagram is an input for STPA analysis. In general, system theoretic analysis uses hazards list and accidental scenarios because of their past experiences and domain expertise. Here, we integrate use case and misuse specification in order to generate a list of hazards their consequences in terms of accident and preventive safety constraints.

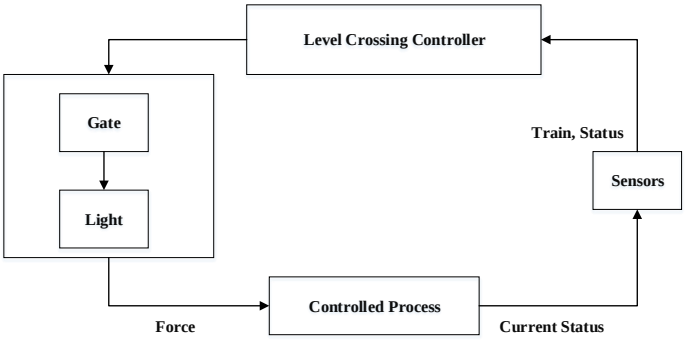


Fig. 6 Control Structure Diagram

Fig. 6 presents the control structure for level crossing system. The inputs such as hazards and accidental scenarios are taken from earlier misuse case analysis from the case study. Again, we model the functional control of the level crossing system using the control structure diagram having controller, sensors, actuators and control actions such as "open/close gate" and "turn red/yellow lights" are generated using failsafe use case specification.

In fact, these control actions are general for normal railway level crossing systems but in this paper, we are more focused on safety related control actions to prevent the accident. Similarly, after control structure diagrams we further analyze the unsafe control actions using the lead words from system thinking. The STPA provides 4 different lead worlds to analyze the unsafe control actions such as 'wrongly provided', 'not provided', 'wrong time' and 'stop too soon'. The analysis results are mentioned in Tab. 5 with the lead words from STPA.

Control Actions	Wrongly Provided	Not Provided	Wrong Time	Stop too Soon
Open Gate	Hazard	Safe	Hazard	N/A
Close Gate	N/A	Hazard	N/A	N/A
Turn Red Light	N/A	Hazard	N/A	N/A
Turn Yellow Light	Hazard	Safe	Hazard	N/A

Tab. 5 Unsafe Control Actions

According to the table, the safety of level crossing system depends of inadequate control of 4 different control actions identified using misuse case diagram.

The identified control actions can be critical for the system in following ways:

1. Wrongly provided: The level crossing system open the gate when the train is approaching or crossing. And, the system turns yellow light on when the train is approaching or crossing. Similarly, whenever the system closes the gate and turn red light on, then there will no hazard as the vehicles cannot are not permitted to pass the crossing. In fact, in this paper, we consider N/A inadequate control for this scenario but in general if the vehicles are struck in between the gates then it will be the critical condition.
2. Not Provided: The level crossing system does not close the gate whenever the train is approaching or crossing. This is another unsafe control for level crossing system which is the same to 'turn red light" control action in the case study.
3. Wrong time: The level crossing system open the gate before the train is leaving. Similarly, whenever the system 'turn yellow light' before the train is leaving is another unsafe control.
4. Stop too soon: The unsafe control generated by this lead word is quite deviated from our case study and it's not applicable. But still if the system closes the gate too soon or for the short time and then again open the gate then it will be critical for level crossing system.

The results of inadequate control are transferred in the following Tab. 6, which describe the applicable unsafe controls of the level crossing system. In the table, we extract only applicable unsafe controls that could lead to the potential accident.

In other word, unsafe control related to control actions 'close gate' and 'turn red light' are the same in meaning as control actions 'open gate' and 'turn yellow light'. For example, the unsafe control related to control action 'open gate' i.e. the system opens the gate when the train is crossing is directly equivalent to the unsafe control related control action 'close gate' i.e. system does not close the gate when the train is crossing.

S.N.	Control Actions	Unsafe Control Action (UCA)
1	Open gate	UCA1. The system opens the gate when the train is approaching or crossing [Hazard 1,2] UCA2. The system opens the gate before the train is leaving [Hazard 1, 2]
2	Turn yellow light	UCA3. The system turns yellow light on when the train is approaching or crossing [Hazard 1, 2] UCA4. The system turns yellow light before the train is leaving [Hazard 1, 2]

Tab. 6 Unsafe Control Related to Case Study

Likewise, unsafe control actions and their corresponding safety constraints are generated as mentioned in Tab. 7, which describes the unsafe controls based on the external factors like train status.

Unsafe Control Actions	Safety Constraints
UCA1	The gate must not be opened while the train is approaching or crossing
UCA2	The gate must not be opened before the train leaving
UCA3	The yellow light must not turn on while the train is approaching or crossing
UCA4	The yellow light must until the train is leaving

Tab. 7 UCAs and Safety Constraints

After identifying unsafe controls and safety constraints, the control structure diagram presented in figure 6, can be transferred into safety control structure diagram with process model variables. These process model variables and basic feedback from the safety constraints can model a safety control structure diagram [8]. Fig. 7 describes the safety control structure for the proposed case study. It contains 5 different process model variables generated from unsafe controls and safety constraints such as Train containing 3 different variables approaching, crossing and leaving the level crossing area, gate containing 2 variables up and down, light with containing 2 variables red and yellow, sensor with activated and deactivated to identify the train status and lastly, process model status with safe and failsafe. Safe denotes the normal operation of level crossing system

whereas failsafe denotes the emergency operation of the system. On the other hand, safety control structure diagrams also contain the control actions generated from misuse case diagrams.

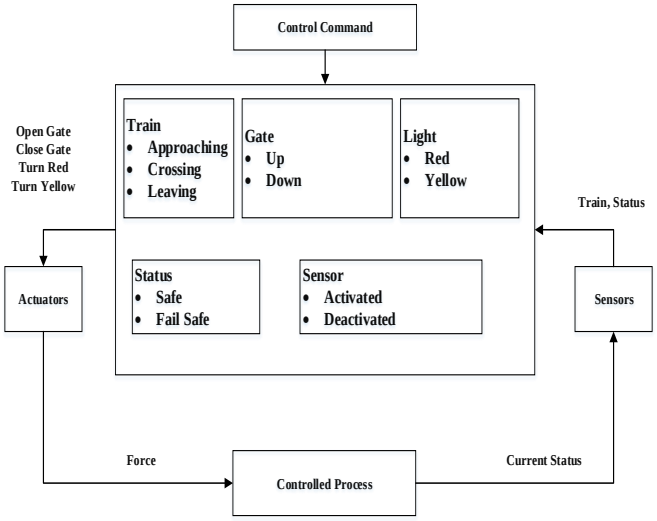


Fig. 7 Safety Control Structure Diagram

Similarly, after generating safety control structure for level crossing system, UCAs their corresponding casual factors are generated along with relevant controls as mentioned in Tab. 7

UCA	Casual Factors	Controls
1-2-3-4	a) Incorrect sensor inputs b) Sensor malfunction c) Faulty control algorithms d) Controller malfunction	a) Provide additional safety inspection to ensure the reliability of sensors and controller b) Provide failsafe mechanism to prevent and protect from the possible accident in the future

Tab. 8 Casual Factors and Controls

At the end of this safety analysis, detailed software safety requirements are generated in the system level of the level crossing system. These software's safety requirements are the results of different aspect of STPA analysis including the analysis of unsafe control actions, safety control structure, safety constraints, casual factors and their controls. The generated software safety requirements are mentioned as follows.

S.N.	Software Safety Requirements (SSR)
SSR1	Open gate command must not be provided while the train is approaching or crossing.
SSR2	Open gate command must not be provided before the train is leaving.
SSR2	Close gate command must be provided while the system status is failsafe.
SSR3	Turn light yellow must not be provided while the train is approaching or crossing.
SSR4	Turn light yellow must not be provided before the train is leaving.
SSR5	Turn light red must be provided while the system is in failsafe mode.

Tab. 9 Generated Software Safety Requirements

5. Discussion

By applying this misuse case-based framework for safety analysis, it is feasible to extract hazards from the faulty actors in the use case diagrams. On the other hand, it is possible to reduce the theatrical gap in STPA analysis input parameters. In fact, identifying hazards, accident and initial safety constraints before is very useful to quantify the result of STPA. From the experiment, we present one accident scenarios of level crossing accident using the misuse case-based system requirements modeling and generate 5 different software safety requirements.

Similarly, the proposed method supports the early safety consideration during requirement engineering phase in safety software development life cycle. The transition of traceability in proposed method make it more comprehensive while interpreting the failure in system level. Likewise, the early focus on the safety maintain the balance in the traditional approach where safety considerations are usually taken at the design phase and are more focused in the failure of physical components rather than the user related interaction and software requirements faults. Using STPA to describe human related and software requirements related faults in misuse case diagram is another strong direction of this research. The usual STPA process are followed only after the misuse case analysis which usually covers both physical component failure and the involved actors. Hence, the usual STPA approach is completely different than the proposed framework. This is because, the traditional misuse case diagrams are used to reflect the human interaction related failure and threats in refining the security requirements.

But, we present our proposed technique in the

laboratory case study where some scope of safety analysis is limited. This results the practical deviation of this method in the real time framework.

6. Conclusion and Future Work

From the experiment, we present a misuse case based STPA analysis for safety critical level crossing system. The proposed method starts with general system requirements specification followed by use case and misuse case-based requirements modeling. The use of UML use case diagram and misuse case is to generate basic list of hazards and their possible accident in the safety critical system. Normally, STPA analysis uses hazards and accident as the input for safety analysis but there is not a clear direction on how to generate a list of hazards and their related accident. This paper aims to provide a use case and misuse case-based hazards generate framework for STPA analysis. As a result, this paper concludes that it is possible to use an extension of UML case diagram to generate the basic parameter for STPA analysis. This is because use case diagrams can model the requirement of the system as well as misuse case can represent the actual failure flow in the system due to the faulty actors.

Similarly, from this research it can also be concluded that the basic UML use case diagrams and its extension can be used for safety analysis in critical system. The pictorial representation of use case diagrams is easy to understand and is very useful to represent the threat caused by the faulty actors in the normal use case representing the functional behavioural of the system.

Lastly, in addition to this the future research dimension might be the use of other UML diagrams in STPA analysis. On the other hand, another direction is to study the analysis and comparison of misuse case in reliability-based safety analysis techniques such as fault tree analysis (FTA), failure and effects analysis (FMEA) and investigating the generated software safety requirements.

References

[1] E. Troubitsyna. Integrating Safety Analysis into Formal Specification of Dependable Systems. Proceedings International Parallel and Distributed Processing Symposium, pp.8, 2003

[2] Nancy Levenson, Engineering a Safer World. MIT Press, 2011

[3] Neledath Damodarn, Secure Software Development Using Use Cases and Misuse Cases. Issues in Iformation Systems, pp.1-2, Vol. VII, 2006

[4] Guttorm Sindre and Andreas L. Opdahl. Capturing Security Requirements Through Misuse Cases. Norsk Informatikkonferanse, 2011

[5] Jacobson. Object Oriented Software Engineering A Use Case Driven Approach, Boston, 1992

[6] Ian Alexander. Misuse Cases: Use Cases with Hostile Intent. IEEE Software, 2003

[7] Fierab Mason Blakley and Ryan Habibi. Prospective Hazard Analysis for Information Systems. IEEE International Conference on Healthcare Informatics, 2014

[8] Nancy Levenson, A New Accident Model for Engineering Safer Systems. MIT Press, 2004

[9] John Thomas. Extending and Automating a Systems-Theoretic Hazard Analysis for Requirements Generation and Analysis, Ph.D. Dissertation, MIT, Boston, 2013

[10] Tor Stalhane and Guttorm Sindre. A Comparison of Two Approaches to Safety Analysis Based On Use Cases. International Conference on Conceptual Modeling, 2007

안전 필수 시스템의 개념단계 해저드 분석 프로세스

김은비 ○, 전은진, 도성룡†, 한혁수
상명대학교 컴퓨터과학과, 소프트웨어 안전성 보증 연구센터†
aqua9131@gmail.com, dmswls3030@gmail.com, imdosungryong@gmail.com†, hshan@smu.ac.kr

A Hazard Analysis Process at Concept Phase of Safety Critical System

김은비 ○, 전은진, 도성룡†, 한혁수
Dept. of Computer Science, Sangmyung University, SW Safety Assurance Research Center†

요 약

오늘날의 시스템들은 많은 컴포넌트들로 구성되어있다. 특히, 전기, 전자, 소프트웨어 등이 시스템의 많은 부분을 차지하게 되면서, 이들로 인한 사고를 예방하기 위한 안전성 확보 노력이 필요해졌다. 그러므로 시스템의 안전성을 확보하기 위해서는 개념 단계부터 이러한 문제들에 대한 잠재적인 해저드를 파악하고, 그 리스크를 완화하는 방안을 설계와 구현에 반영하는 것이 요구된다. 이를 위해서는 요구사항을 명세하고 모델링하여 해저드를 분석하는 방법이 필요하다. SysML의 요구사항 다이어그램은 이러한 목적에 적합한 모델링 방법으로 인정받아 왔다. 하지만, 파악된 해저드를 기반으로 도출된 안전 요구사항을 모델링에 반영하기 위한 연구는 많이 이루어지지 않았다. 본 논문에서는 SysML의 다이어그램들을 활용한 해저드 분석 방법과 이를 통해 도출한 안전 요구사항을 요구사항 다이어그램에 반영하기 위한 프로세스를 제안한다. 그리고 제안한 프로세스의 효용성을 입증하기 위해 엘리베이터 시스템에 적용해 보았다.

1. 서론

시스템 내 전기, 전자 및 소프트웨어의 비중이 높아지고, 소프트웨어가 시스템들을 제어하기 시작하면서, 기계적 고장 뿐만 아니라, 소프트웨어 결함 및 시스템 컴포넌트들 간의 상호작용에 따른 오작동(Malfunction)으로 인한 사고의 위험이 커지고 있다[1].

자동차, 의료, 항공 등 안전 필수(Safety Critical) 산업분야에서는 기능 안전(Functional Safety)의 메타 모델인 IEC 61508을 기반으로 각 산업 분야의 특성에 맞춘 기능 안전 표준을 제정하였다. 이러한 표준들은 시스템의 안전 확보를 위해 전체 개발 생명주기에서 준수해야 할 안전 활동을 정의하고 있다. 그림 1은 자동차 분야의 기능 안전 국제 표준인 ISO 26262에 정의된 안전 활동의 일부를 나타낸다[2].

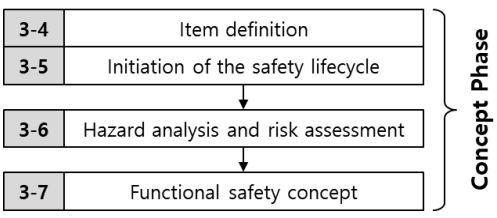


그림 1. ISO 26262의 안전 활동의 일부(개념 단계)

안전 활동의 중심에는 안전에 위협이 되는 잠재적인 해저드(Hazard)를 파악하기 위한 해저드 분석 작업이 있다. 파악된 해저드는 리스크(Risk)를 평가하고, 수용할 수 없는 해저드에 대해서는 제거 또는 완화하기 위한 안전 요구사항(Safety Requirement)으로 전환해야 한다. 해저드를 분석하는 활동은 전체 개발 생명주기 동안 지속적으로 해야한다. 특히, 개념 단계에서는 시스템의 목적과 요구사항을 정의하고 이를 기반으로 개발을 수행하기 때문에, 이 단계에서의 해저드 분석 활동은 중요하다.

일반적으로 개념 단계의 해저드 분석 기법으로는 PHL(Preliminary Hazard List)와 PHA(Preliminary Hazard Analysis)등을 실시한다. 하지만 이러한 해저드 분석 방법들은 시스템 개발 활동들과 분리해서 수행하는 경향이 있으므로, 안전 분석가와 개발자들 간에 소통이 어려운 경우들이 있다. 그러므로 소프트웨어의 비중이 높아진 안전 필수 시스템에서는 해저드 분석 활동과 시스템 개발 생명주기를 연계해야 한다.

또한 시스템을 체계적으로 개발하기 위해서는 개념 단계부터 유지보수 단계까지의 모든 내용들을 명확하게 정의해야 하는데, 텍스트 만을 활용하면 이해하고 분석하는데 어려움이 있다. 그러므로 표준으로 인정받는 다이어그램들을 활용하는 것이 바람직하다.

소프트웨어 개발에서는 UML(Unified Modeling Language)이 업체 표준으로 채택되어 왔고, 시스템 개발에서는 UML의 확장 모델인 SysML(System Modeling Language)이 사용되고 있다. SysML은 그림 2와 같이 시스템을 다양한 관점으로 분석할 수 있는 다이어그램들을 제공하고 있다[3].

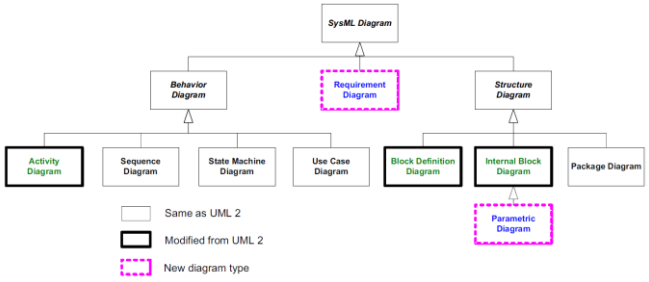


그림 2. SysML의 다이어그램

특히 요구사항 다이어그램(Requirement Diagram)은 Containment, Satisfy, Verify, Derive, Refine, Trace, Copy 등의 관계 속성을 이용하여, 텍스트 기반의 요구사항 명세를 여러 다이어그램들과 연결시킴으로써 구체적이고, 통합적인 분석과 추적을 용이하게 한다[4].

하지만 파악된 해저드를 기반으로 리스크를 제거하거나 완화하기 위한 안전 요구사항을 모델링에 반영하기 위한 연구는 많이 이루어지지 않고 있다.

본 논문에서는 SysML 다이어그램들을 활용한 시스템 개념 단계에서의 해저드 분석 방법을 제시한다. 특히, 이를 통해 도출한 안전 요구사항들을 요구사항 다이어그램에 반영하는 방안을 제안한다. 본 논문에서 제안한 방법의 효용성을 검증하기 위해 엘리베이터 시스템에 적용해 보았다.

2. 관련 연구

2.1 개념 단계의 해저드 분석 활동

안전 필수 시스템의 개발 단계는 개념(Concept), 제품 개발(Product Development), 운영&유지(Operation and Maintenance) 단계로 구분한다.

개념 단계에서는 안전 요구사항을 정의하고 분석한다. 제품 개발 단계에서는 안전성을 고려하여 하드웨어 및 소프트웨어에 대한 개발 활동을 수행한다. 마지막으로 운영&유지 단계에서는 시스템의 운영을 모니터링하고 그 결과를 피드백하여 시스템을 개선한다.

개념 단계에서 안전 요구사항을 도출하기 위한 대표적인 해저드 분석 기법으로는 PHA가 있고, 이를 지원하기 위한 기법으로는 PHL과 상위 수준 구조도 기반의 분석 방법인 SHA(System Hazard Analysis)가 있다. 이 단계에서 파악한 해저드는 제품 개발 단계에서 제거하거나 완화하기 위한 조건들을 제공하게 된다.

System: Ace Missile System Subsystem: ***Missile Structure Subsystem***			**Preliminary Hazard Analysis**				Analyst: Date:		
No.	Hazard	Causes	Effects	Mode	IMRI	Recommended Action	FMRI	Comments	Status
PHA-1	Missile structure fails, resulting in unstable missile flight and missile crash	Manufacturing defect; design error	Unstable flight, resulting in crash causing death/injury; incorrect target	Flight	1D	Use 5× safety factor on structure design	1E		Open
PHA-2	Missile body breaks up, resulting in fuel leakage; and ignition source, causing fire	Manufacturing defect; design error	Missile fire, causing death/injury	Ground operations	1D	Use 5× safety factor on structure design	1E		Open

그림 3. PHA의 산출물 양식

그림 3은 PHA의 산출물 양식을 보여주고 있다. 하지만 PHA는 주로 환경문제, 부품 고장, 휴먼 에러 측면의 체크리스트만 고려한다는 한계가 있다.

2.2 기존 연구 분석

과거 기계 중심 시스템들에서는 컴포넌트의 고장이 사고의 주 원인이었다. PHA, FTA(Fault Tree Analysis), FMEA(Failure Mode and Effect Analysis) 등의 전통적인 해저드 분석 기법들은 이를 가정하고 해저드를 파악하였다. 현재의 전기, 전자, 소프트웨어 기반의 시스템들에서는 컴포넌트들 간의 예상치 못 하거나 의도치 않은 상호작용이 사고의 더 큰 원인이기 때문에 여러 관점을 고려한 분석이 필요하다[5].

Leveson은 Chain of Event 방식을 보완하여 시스템 이론에 기반을 둔 분석 방법인 STPA(Systems Theoretic Process Analysis)를 개발하였고, 이 기법은 여러 분야에서 활용되고 있다[5]. 하지만, UML, SysML 등의 공학적 모델링 활동들과 별도로 수행해야 한다는 어려움이 있다. Martin-Guillerez 등은 UML 기반의 모델링과 PHA, HAZOP, FTA, FMEA를 접목시키는 방안을 제시하고 있다[6]. Zhong 등은 UML을 기반으로 STPA를 확장하려고 시도했다[7]. 김영민 등은 SysML을 기반으로 전체 해저드 분석 프로세스를 제안했다[8].

하지만, 이 연구들은 시스템 개념 단계의 해저드 분석과 이를 통해 도출된 안전 요구사항을 시스템 모델링에 반영하기 위한 체계적인 방안을 제공하지 않고 있다.

3. SysML 기반의 개념 단계 해저드 분석 및 안전 요구사항 모델링 프로세스

3.1 SysML 다이어그램을 활용한 개념 단계 해저드 파악

개념 단계에서 작성되는 산출물과 적용 가능한 SysML 다이어그램들은 그림 4와 같다.

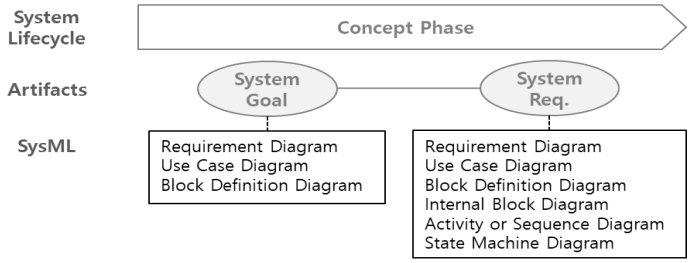


그림 4. 개념 단계의 산출물과 SysML 다이어그램

PHL, PHA 등의 해저드 분석 지원하기 위하여 SysML이 제공하는 요구사항 다이어그램과 Block Definition 등의 구조 다이어그램 그리고 Activity 등의 행위 다이어그램들을 활용하는 것이 필요하다.

3.2 SysML 요구사항 다이어그램과 관계 속성

요구사항 다이어그램의 관계 속성들과 근거(Rationale)는 요구사항을 통합적으로 분석, 이해, 검증하는데 도움이 된다.

표 1. 요구사항 다이어그램의 관계 속성

관계	Stereotype	의미
Containment		요구사항 간의 포함 관계
Satisfy	satisfy	요구사항을 만족하는 설계 컴포넌트 등과의 관계
Verify	verify	요구사항을 확인하기 위한 테스트 케이스와의 관계
Derive	deriveReqt	요구사항 분석을 통해 도출된 요구사항과의 관계
Refine	refine	요구사항을 보다 구체적으로 설명하기 위한 Use Case와 State 등과의 관계
Trace	trace	요구사항의 근거가 되는 다른 문서와의 관계
Copy	copy	요구사항 간의 복사 관계

요구사항 다이어그램의 관계 속성별 Stereotype과 의미는 표 1과 같으며, 이를 다이어그램으로 표현하면 그림 5와 같다.

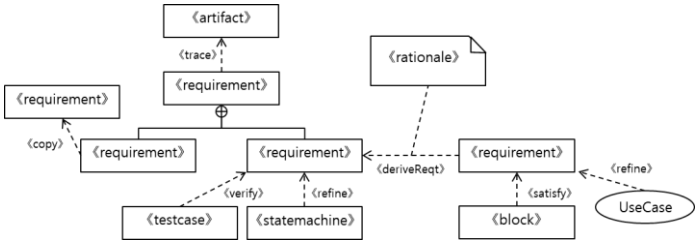


그림 5. 요구사항 다이어그램의 관계

기존 요구사항을 기반으로 해저드 분석을 통해 도출된 안전 요구사항은 이 관계 속성 중 Derive 관계로 표현 가능하다.

3.3 제안 프로세스

SysML의 다이어그램들을 기반으로 기존의 해저드 분석 기법의 목표에 보다 체계적으로 접근하기 위한 프로세스는 그림 6과 같다.

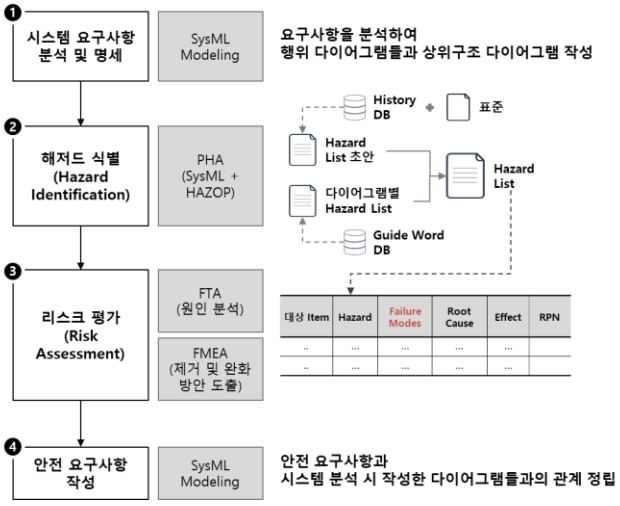


그림 6. SysML 기반의 개념단계 해저드 분석 및 안전 요구사항 모델링 프로세스

안전 요구사항과 시스템 분석 시 작성한 다이어그램들과의 관계는 그림 7과 같이 정립한다.

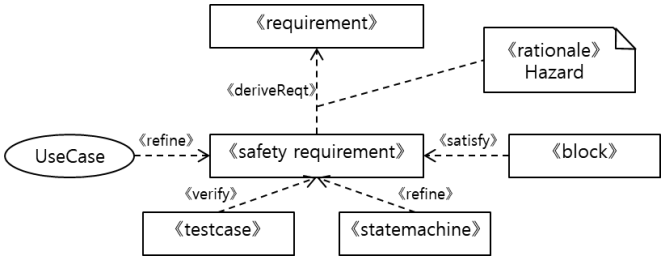


그림 7. 안전 요구사항을 반영한 요구사항 다이어그램

안전 요구사항은 기존 요구사항을 기반으로 해저드 분석을 통해 도출하였기 때문에 Derive 관계로 연결하고, 이와 관련된 행위 다이어그램은 Refine 관계, 구조 다이어그램은 Satisfy 관계로 연결한다. 또한, 안전 요구사항을 검증하기 위한 테스트 케이스는 Verify 관계를 사용하며, Rationale을 활용하여 안전 요구사항이 도출된 근거를 제시한다.

4. 사례 적용: 엘리베이터 시스템

본 논문에서 제안하는 해저드 분석 프로세스의 효용성을 검증하기 위해 엘리베이터 시스템에 적용해 보았다. 엘리베이터 시스템의 요구사항은 표 2와 같다.

표 2. 엘리베이터 시스템의 요구사항

목적(Goal)	엘리베이터 시스템의 모든 구성요소를 안전하고 효율적으로 제어
ID	내용
REQ01	승객의 요청에 응답하여 층간 엘리베이터를 탐색하고, 요청을 처리해야 함
REQ02	엘리베이터 시스템의 모든 구성요소의 입/출력 신호를 처리해야 함
REQ03	사고 발생 시 승객에게 알려야 함

본 요구사항을 분석하여 작성한 요구사항 다이어그램은 그림 8과 같다.

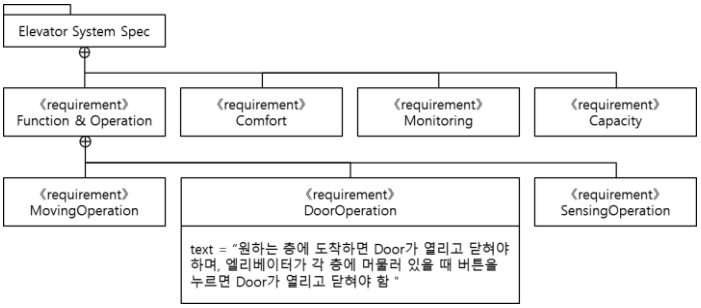


그림 8. 엘리베이터 시스템의 요구사항 다이어그램

요구사항을 기반으로 Use Case, State Machine, Sequence 다이어그램을 작성하여 시스템의 기능과

상태, 작동 흐름을 파악한다. 그림 9는 엘리베이터 시스템의 DoorOperation 요구사항에 대한 State Machine 다이어그램을 나타낸다.

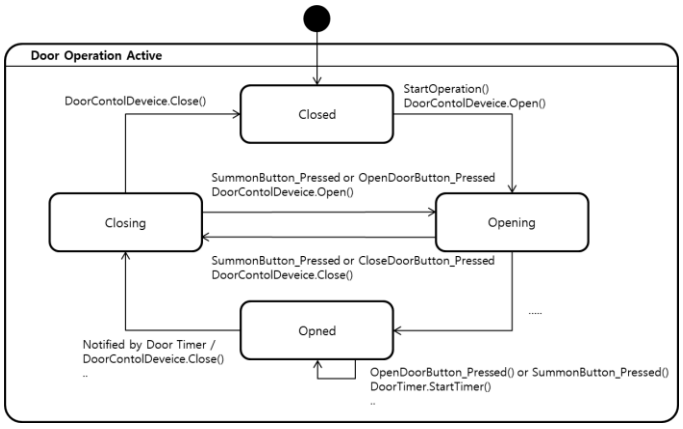


그림 9. DoorOperation의 State Machine 다이어그램

엘리베이터 시스템의 안전 요구사항을 도출하기 위해 각 다이어그램에 적합한 HAZOP 가이드워드를 적용하여 해저드를 파악하였다. 표 3은 DoorOperation에 대한 State Machine 다이어그램의 속성(Attribute) 일부로부터 도출된 해저드의 일부를 보여준다.

표 3. 엘리베이터 시스템의 해저드

속성	가이드워드	해저드
DoorTimer. StartTimer()	No/None	Door Timer가 작동하지 않음
	Early	Timer의 시간보다 더 빨리 Door가 닫힘
	Other than	Door Timer의 이상으로 인해 의도하지 않은 시점에서 Door가 닫힘

식별한 해저드를 기반으로 안전 요구사항을 작성하고, 그림 10과 같이 요구사항 다이어그램에 반영한다.

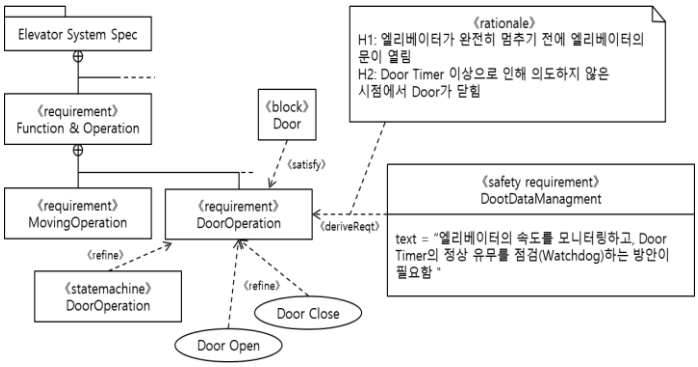


그림 10. 안전 요구사항을 반영한 엘리베이터 시스템의 요구사항 다이어그램

5. 결론

본 논문에서는 SysML 다이어그램들을 활용한 시스템 개념 단계에서의 해저드 분석 방법을 제안하였다. 제안한 방법의 장점은 다음과 같다.

첫째, SysML 다이어그램을 기반으로 개발 프로세스와 PHA, SHA 등의 해저드 분석 활동을 통합시킴으로써 개발자들과 안전 분석가들의 소통을 원활하게 하였다.

둘째, 시스템을 SysML의 요구사항과 행위, 구조 관점의 다이어그램으로 표현함으로써, 종합적인 해저드 분석이 가능하였다.

셋째, 도출된 안전 요구사항을 SysML 요구사항 다이어그램의 Derive 관계로, 상세화를 위해 Refine 관계로, 검증을 위해 Verify 관계로 표현함으로써 잠재적인 해저드에 대한 제거 또는 완화 방안을 개념 단계에서 정의하였다.

끝으로 본 논문에서 제안한 방법을 엘리베이터 시스템에 적용하여, 그 효용성을 입증하였다.

사 사 (Ack)

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터육성 지원사업의 연구결과로 수행되었음 (2015-0-00445)

참 고 문 헌

[1] Leveson, N., "A new accident model for engineering safer systems", Safety science, 42(4), 237-270, 2004

[2] ISO, “ISO 26262 : Road Vehicles-Functional Safety,” 2011

[3] OMG, "OMG System Modeling Language", http://www.omg.org/spec/SysML/1.5/, 2017

[4] Friedenthal, S., Moore, A., & Steiner, R., "OMG Systems Modeling Language (OMG SysML™) Tutorial", In INCOSE Intl. Symp., 2006

[5] Leveson, N., "Engineering a safer world: Systems thinking applied to safety", MIT press, 2011

[6] Martin-Guillerez, D., Guiochet, J., & Powell, D., "Experience with a Model-based Safety Analysis Process for Autonomous Service Robot", In 7th International Workshop on Technical Challenges for Dependable Robots in Human Environments (DRHE), 2010

[7] Zhong, D., Wu, N., Wang, Q., & Sun, R., "A multi-view extended software control structure modeling and safety analysis method" In Prognostics and System Health Management Conference (PHM), 2015(pp. 1-5), IEEE, 2015

[8] 김영민, & 이재천, "안전중시 시스템의 설계프로세스 구축에서 SysML 모델의 활용에 관한 연구", 대한안전경영과학회지, 15(3), 7-17, 2013

객체지향 재사용 매트릭스

변은영 O1 박지훈 2 김영철 3

홍익대학교 소프트웨어공학연구실

{eybyun[1], pjh[2], bob[3]}@selab.hongik.ac.kr

Object Oriented Reusability Metrics

Eun Young Byun O1, Ji Hoon Park[2], R. Young Chul Kim[3]

SE Lab, Dept. of Computer Information Communication, Hongik University

요 약

소프트웨어 산업에서의 잦은 업데이트와 유지보수 시 소모되는 시간/비용 절감 및 고품질화를 위한 방안으로 재사용은 매우 중요한 이슈이다. 이 논문은 레거시 시스템을 기반으로 확장 또는 기능이 개선된 새로운 시스템 개발에서 재사용 메커니즘을 고품질을 위한 하나의 방안으로 제안한다. 기존에 연구된 객체지향 패러다임에서 재사용성 매트릭스의 문제점(①동적 매트릭스 미비, ②CK 매트릭스, ③순환 복잡도, ④상속성, ⑤다형성)을 도출하고, 이를 개선 및 확장한 정적/동적 매트릭스를 정의한다. 또한, 정의한 매트릭스를 측정하여 재사용 모듈을 자동 식별하는 시스템을 구축한다. 이는 기존 레거시 시스템 모듈의 재사용성을 높임으로써, 신뢰성 및 생산성을 개선시키고 품질을 향상시킬 수 있다.

1. 서 론

최근 소프트웨어의 적용 분야와 규모가 다양화 및 대규모화 되면서 시장 환경의 급속한 변화, 빈번한 사용자 요구사항 변경 요청 등이 야기되고 있다. 이로 인한 부족한 시간, 예산, 인력 문제들로 인해 고품질 소프트웨어 개발은 더욱 어려운 실정이다. 고품질의 소프트웨어를 위해, 국내 중소/중견/벤처/스타트업 기업들은 SW 테스팅에 초점을 두지만, 이 논문에서는 레거시 시스템의 모듈을 재사용하여 고품질화시키기 위한 재사용 매트릭스 및 자동식별 시스템을 제안한다.

기존 재사용 매트릭스 연구[1,2,3]는 정적 분석 기반의 재사용 모듈 식별이 이루어졌지만, 동적 분석 기반에서의 재사용성 식별은 미비하다. 또한, 객체지향 프로그램은 절차지향 프로그램과 차이를 갖지만 상속성, 다형성 등의 특성이 반영되지 않았다. 이를 위해, 확장 및 개선된 재사용 매트릭스를 정의하고 이를 자동 측정하는 시스템을 구축한다. 식별된 모듈을 재사용함으로써 레거시 시스템을 개선 및 확장하는 새로운 시스템의 도메인에서 빠르고 효율적인 개발이 가능하다.

본 논문의 구성은 다음과 같다. 2장은 관련연구로 기존 재사용 매트릭스의 문제점을 언급한다. 3장은 확장 및 개선된 정적/동적 재사용 매트릭스를 정의한다. 4장은 정의한 매트릭스 기반 자동 식별 시스템을 설명한다. 5장에서는 결론 및 향후 연구를 언급한다.

2. 관련연구

Marko[4]의 연구에서 언급된 기존 연구에서 다음의 문제점을 도출하였다. 1) 정적 관점에 편향되어 있고 동적 관점은 매우 미비하다. 동적 매트릭스는 Component Dynamic Complexity (CDC) 단 한가지만이 언급된다. 2) 재사용성 측정을 위해 대표적으로 사용되는 CK 매트릭스는 결합도와 응집도의 종류를 고려하지 않는다. 3) 순환 복잡도는 객체지향 프로그램의 복잡도를 측정하는데 한계가 있다. 이 논문에서는 이를 개선하기 위해, 성숙 결합도/응집도를 측정함으로써 재사용성을 측정하고자 한다. 객체지향 프로그램과 절차지향 프로그램의 주요 차이점은 상속성이다. 4)이로 인해, 객체지향 메커니즘에서의 결합도는 단순한 호출 관계만으로는 식별이 불가능하다. 특히 레거시 시스템의 재사용에서의 상속성과 같은 클래스간의 관계는 고려해야 하는 문제이다. 다형성도 객체지향의 중요한 특성이다. 5)오버라이딩에서의 동적 바인딩은 동적 분석으로만 식별이 가능하다.

3. 확장 및 개선된 정적/동적 재사용 매트릭스

기존의 문제점을 해결하고자, 확장 및 개선된 정적/동적 재사용 매트릭스를 정의한다. 레벨은 메소드, 클래스, 객체, 클래스-객체로 구분한다. 클래스-객체 레벨은 정적인 클래스와 동적인 객체 간의 동적 바인딩으로 현재 연구 중에 있다. 기존 매트릭스는 제외하고, 확장 및 개선된 매트릭스에 대해 설명한다.

3.1 메소드 레벨

- Coupling Between Method(CBM): 클래스 레벨에서 클래스간의 호출 횟수를 계산한 CBO(Coupling Between Object Classes)를 메소드 레벨로 확장한다.

▪ Method Coupling(MCO): 자료/스탬프/제어/외부/공유/내용 결합도를 구분하고 평균을 측정한다.
▪ Lack of Cohesion of Statement(LCOS): 클래스 레벨에서 클래스 내부 전역 변수에 접근성을 측정한 Lack of Cohesion of Methods(LCOM)을 메소드 레벨로 확장한다.
▪ Method Cohesion(MCE): 우연/논리/시간/절차/교환/순차/기능적 응집도를 구분하고 평균을 측정한다.
▪ Dynamic Coupling Between Method(DCBM): CBM을 동적 환경에서 측정한다.
▪ Dynamic Method Coupling(DMCO): MCO를 동적 환경에서 측정한다.

3.2 클래스 레벨

▪ Class Coupling(CCO): 클래스 내부 메소드가 외부 메소드들과 갖는 결합도 MCO를 합산한다.
▪ Class Cohesion(CCE): 클래스 내부의 메소드 간의 결합도 MCO를 합산한다.

3.3 객체 레벨

▪ Dynamic Coupling Between Object Classes (DCBO): CBO를 동적 환경에서 측정한다.
▪ Dynamic Object Coupling(DOCO): 클래스 내부 메소드가 외부 메소드들과 갖는 동적 결합도 DMCO를 합산한다.
▪ DORC(Dynamic Object Relation Coupling): 동적 환경에서 클래스의 상속, 참조, 호출 관계를 기반으로 동적 결합도를 측정한다.
▪ DOCE(Dynamic Object Cohesion): 클래스 내부의 메소드 간의 동적 결합도 DMCO를 합산한다.

4. 자동 재사용성 식별 시스템

4.1 시스템 구성도

재사용성을 측정하여 재사용 모듈을 자동 식별하는 시스템을 구축한다. 그림 1은 시스템 구조도이다. 이는 자바 프로그램을 대상으로 개발되었으며, 정적 분석 도구 xCodeParser를 사용하여 ASTM파일을 추출하고, 동적 분석 도구 HPROF를 사용하여 프로그램 프로파일링 결과 텍스트 파일을 추출한다. 분석 결과를 데이터베이스에 저장하고 재사용 매트릭스를 측정하는 Generate DB from ASTM/Hprof를 구현한다. 그 결과는 Tabulator를 사용하여 웹 기반으로 가시화된다. 측정된 수치 및 재사용 모듈 식별 결과를 출력한다.

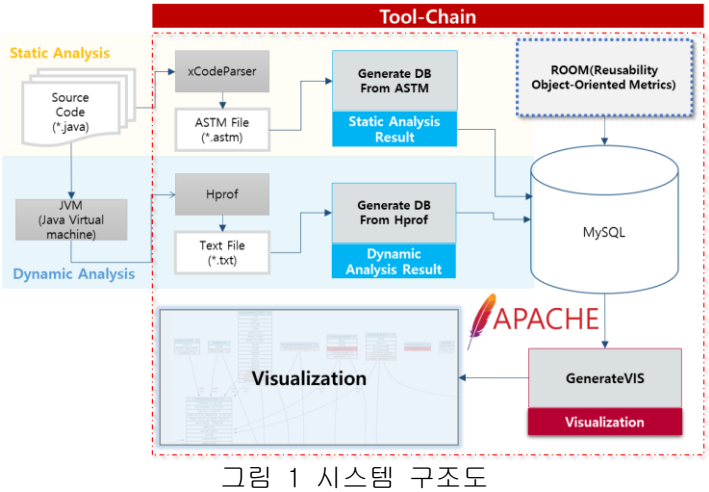


그림 1 시스템 구조도

4.2 적용 사례

가시화 타겟 코드는 연구실에서 개발한 '신재생에너지 발전소 모니터링 시스템'으로 로컬과 서버의 통신을 위해 이중 데이터의 메타모델을 사용하는 시스템이다[5]. 그림 2는 이 시스템의 재사용 모듈 자동 식별 결과이다. 자동 재사용성 식별 시스템에 의해 메소드, 클래스, 객체 레벨의 매트릭스가 계산되고 그 결과는 바 형태로 나타내어 어떤 수치가 높은지를 구분할 수 있다. 또한, 이 수치를 기반으로 해당 모듈이 재사용에 적합한지 마지막 열에 표기한다. 이 기준은 사용자가 임의로 설정할 수 있으며, 이 사례에서도 임의로 사용자의 요구를 정의하여 결과를 식별했다.

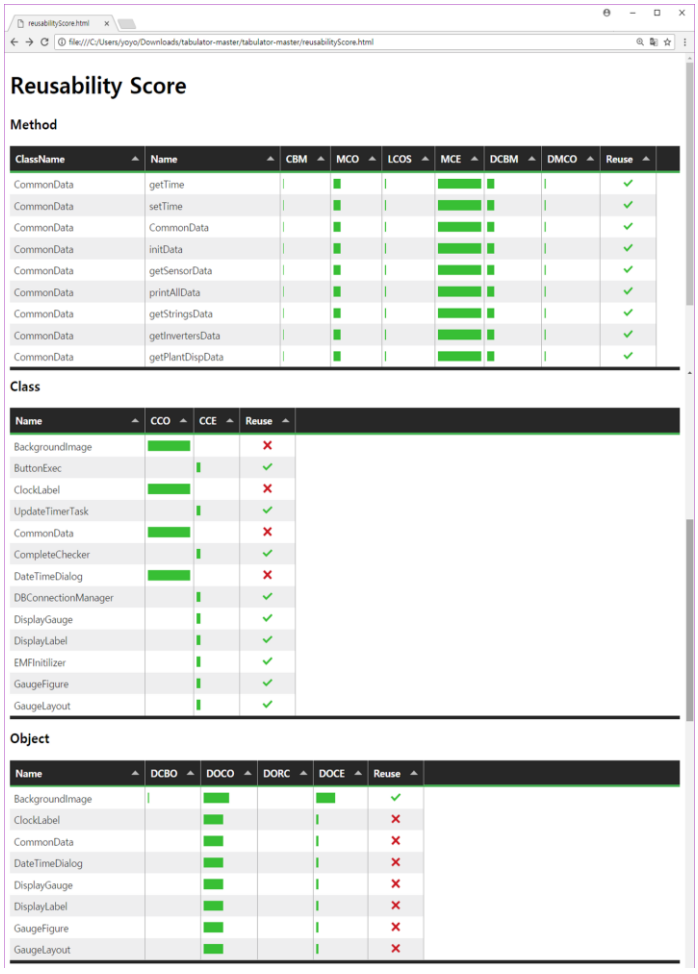


그림 2 재사용 모듈 자동 식별

5. 결론 및 향후 연구

최근 급속한 시장 환경 변화와 빈번한 요구사항 변경으로 소프트웨어의 생산성을 높이지 못하고 있다. 이는 소프트웨어 품질의 문제로 직결된다. 이 문제의 방안들로 재사용성, 유지보수성, 품질, 생산성, 효율성 등에 대한 다양한 연구가 이루어졌다. 그 중에서도 재사용성은 소프트웨어 시장의 잦은 유지보수로 인한 품질 저하 문제를 해결할 수 있다.

기존 연구들은 재사용성의 정적 지표에 대해 초점이고 동적 지표의 연구는 부족했다. 이를 해결하고자, 기존 지표를 기반으로 확장/개선된 정적/동적 지표를 제안하였다. 이를 위해, 오픈 소스 기반으로 구축된 재사용 모듈 자동 식별 시스템을 구축하였다. 이는 동적 분석 도구를 사용하여 정적 분석에서는 얻을 수 없는 데이터를 수집하여 재사용성을 측정한다. 레거시 시스템의 확장 및 기능 개선으로 새로운 시스템을 구축할 경우, 식별된 모듈을 재사용하여 개발 비용 및 시간 절감과 품질 개선을 기대한다.

Acknowledge

본 논문은 2015년 교육부와 한국연구재단의 지역혁신창의인력양성사업의 지원을 받아 수행된 연구임(NRF-2015H1C1A1035548).

참고문헌

[1] W Frakes and C. Terry, “Software reuse: metrics and models, “ACM Computing Surveys, Vol. 28, No. 2, pp. 415-435, June. 1996.

[2] Chidamber Shyam and kemerer Chris, “A metrics suite for object oriented design,” IEEE Transactions on Software Engineering, Vol. 20, No. 6, June. 1994.

[3] DocForge, (Dec. 2017.) “Code reuse”.

[4] Marko Mijac and Zlatko Stapic, “Reusability Metrics of Software Components: Survey” in Central European Conference on Information and Intelligent, Varazdin, Croatia, pp. 221-230, September, 2015.

[5] Hyun Seung Son, So young Moon, R. Young Chul Kim, “Replacing Source Navigator with Abstract Syntax Tree Metamodel(ASTM) on the open source oriented tool chains SW Visualization,“ In International Conference on Convergence Technology, Vol. 5, No. 1, pp. 366-367, 2015.

DEVS 기반 에이젼트 시뮬레이션 모델 생성기

정준영 ○ 배장원 이천희 백의현
한국전자통신연구원
Jyjung21@etri.re.kr

DEVS based Agent Simulation Model Generator

Joonyoung Jung ○, Jang Won, Bae, Chunhee Lee, Euihyun Paik
Electronics and Telecommunications Research Institute

요 약

본 연구에서는 DEVS(Discrete Event System Specification) 기반의 에이전트 시뮬레이션 모델 생성기를 제안한다. 에이전트 시뮬레이션을 사용하여 사회 변화 예측등과 같은 장기간 시뮬레이션을 수행하면 오류가 누적되어 시뮬레이션 결과가 부정확하게 되며, 이러한 오류 누적을 극복하기 위해서 자가 진화를 통한 에이전트 시뮬레이션을 실행할 필요가 있다. 에이전트 시뮬레이션 플랫폼에서 자가진화를 수행하기 위해서는 시뮬레이션 모델이 재구성되어야 하므로, 에이전트 모델링이 컴포넌트 기반으로 구현되어야 한다. 이를 위해 본 논문에서는 DEVS 기반의 시뮬레이션 컴포넌트 모델(원자모델, 결합모델)을 GUI 로 생성하고, 생성된 컴포넌트 모델에 대한 메타데이터를 생성하고. 생성된 메타데이터를 이용하여 시뮬레이션 코드를 생성하는 시스템을 제안한다. 제안된 방법을 통하여 형식론에 익숙하지 않은 사용자도 재구성이 가능한 DEVS 기반의 에이전트 시뮬레이션 모델을 구현할 수 있다.

1. 서 론

사회/복지를 포함한 다양한 분야에서 미래에 발생할 수 있는 문제에 대해 적절한 조치를 취하기 위해 시뮬레이션이 수행될 수 있으며, 시뮬레이션을 통해 사회 구성원의 다양한 특징이 예측될 수 있으므로 저출산, 고령화와 같은 사회 문제를 예방하는데 도움이 될 수 있다. 사회적 이슈 예측을 위해 거시적 시뮬레이션과 미시적 시뮬레이션을 사용할 수 있으며, 거시적 시뮬레이션은 전체 사회 구조와 특성의 확률론적 접근법을 사용하여 사회적 이슈의 전반적인 경향을 예측하고, 미시적 시뮬레이션은 사회 문제와 관련된 각 구성원의 행동과 특성을 예측할 수 있다. 따라서 미시적 시뮬레이션은 사회 현상의 원인을 미세하고 다양하게 분석 할 수 있는 장점이 있으며, 마이크로 시뮬레이션 모델링(MSM)과 개인과 개인간의 상호 작용을 모델링하는 에이전트 기반 모델링(ABM)이 사용된다[1-7]. ABM은 대부분 규칙에 따라 행동을 정의하며, 이러한 기술을 사용하여 장기간 시뮬레이션을 수행하면 오류 누적으로 인해 시뮬레이션 결과가 부정확해 질 수 있다. 이러한 오류 누적을 극복하기 위한 방법으로, 변화하는 데이터를 에이전트 모델과 동화하고, 그 결과를 반영하여 모델링 및 시뮬레이션 구성이 최적화될 수 있는 자가 진화 에이전트 기반 시뮬레이션을 실행할 수 있다[8,9]. 에이전트 시뮬레이션 플랫폼에서 시뮬레이션 모델이 자가진화를 하기 위해서는 모델을 컴포넌트화 되어 효과적으로 재구성 될 수 있어야 한다. 모델을 컴포넌트화 하기 위한 방법으로, 에이전트 모델링 및 시뮬레이션 모델을 DEVS(Discrete Event System Specification) 기반으로 구현할 수 있다. 본 논문에서는 자가진화 에이전트 시뮬레이션 구축을 위해 DEVS 기반의 컴포넌트 모델을 GUI 방식으로 생성하고, 메타데이터를 생성하고, 모델 코드를 체계적으로 생성할 수 있는 코드 생성기 시스템을 제안한다.

본 논문은 2 절에서는 모델 생성기 구성에 대하여 살펴보고, 3절에서는 모델 생성기 세부 모듈의 기능과 구현에 대하여 설명하고, 4절에서는 결론을 맺는다.

2. 모델 생성기 구성

아래 그림 1은 모델 생성기의 구성을 나타내며, 모델저작도구 GUI, DEVS 모델 생성, 중간코드 생성, DEVS 코드 생성 및 DEVS 코드 컴파일 모듈로 구성되어 있다.

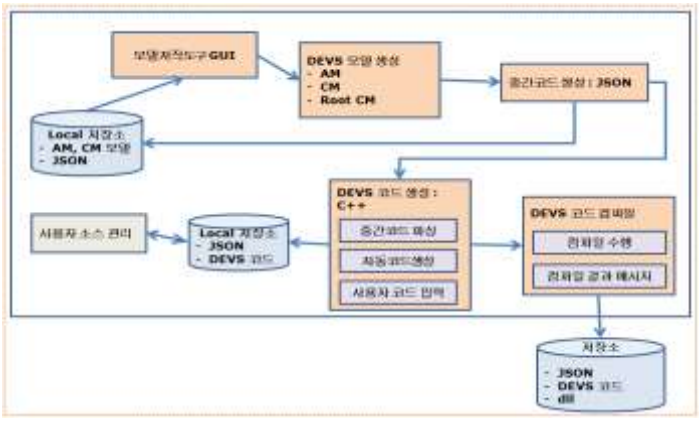


그림 1. DEVS 모델 생성기 구성

DEVS 모델저작도구 GUI를 통해 그래픽 기반으로 DEVS 모델을 생성할 수 있으며, DEVS 모델 생성 모듈을 통해 원자모델(AM) 및 결합모델(CM)을 생성한다. 중간코드 생성 모듈은 그래픽으로 생성한 AM 및 CM 을 이용하여 메타데이터인 JSON을 생성한다. DEVS 코드 생성 모듈은 JSON 코드의 구문 및 어휘를 분석하여 자동으로 템플릿 코드를 생성하고, 추가적인 사용자 코드를 작성할 수 있도록 한다. 자동 생성된 소스코드를 사용자가 다른 개발환경에서 사용할 수 있도록 하기 위해 소스코드를 다운로드를 받고, 사용자 개발환경에서 수정한 후 저장할 수 있는 기능을 추가한다. AM 및 CM에 대한 DEVS 코드가 완성된 후 DEVS 코드 컴파일 모듈에서 컴파일을 수행하고, 자가진화 에이전트 시뮬레이션 플랫폼의 다른 모듈이 사용할 수 있도록 저장소에 JSON, DEVS 코드 및 실행코드를 저장한다.

3. 모델 생성기 구현

자가진화 에이전트 시뮬레이션을 위한 DEVS 기반의 모델 생성기를 아래와 같이 구현하였다.

3.1 DEVS 모델생성

그림2와 그림3에서 보는 바와 같이 원자모델 및 결합모델을 GUI 기반으로 생성할 수 있도록 한다.

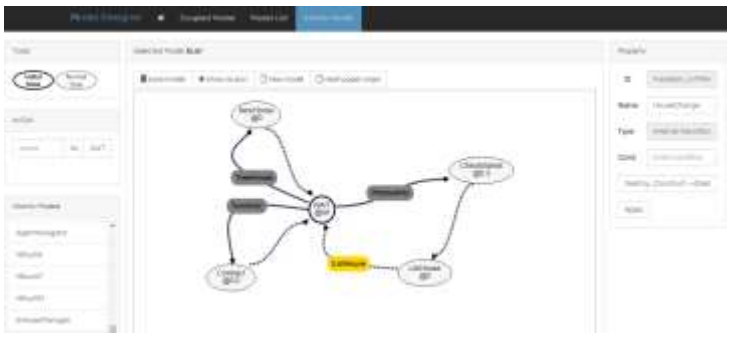


그림 2. DEVS 기반 에이전트 시뮬레이션 원자모델 생성

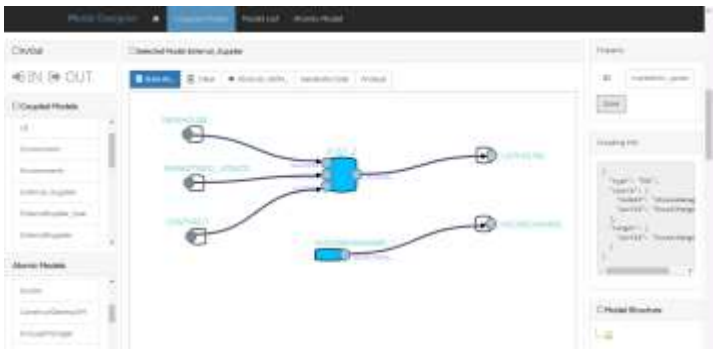


그림 3. DEVS 기반 에이전트 시뮬레이션 결합모델 생성

DEVS 모델생성 모듈은 컴포넌트와 컴포넌트간의 상호작용(에이전트간 상호 작용, 환경 간 상호작용, 에이전트와 환경의 상호작용)을 그래픽 기반으로 생성할 수 있으며, Drag&Drop 형식으로 노드(DEVS 구성요소 및 컴포넌트)를 연결할 수 있다. 생성된 컴포넌트는 로컬 저장소에 저장된다.

3.2 중간코드 생성

생성된 컴포넌트 모델(AM, CM, Root CM)에 대하여 JSON 코드를 생성한다. DEVS 기반의 컴포넌트를 표현하기 위한 JSON 스키마는 그림 4에서 보는 바와 같다.

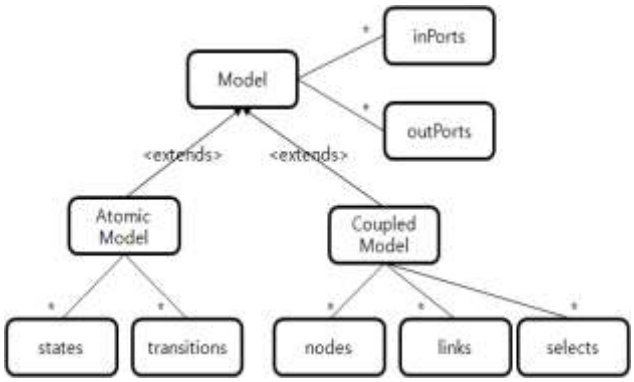


그림 4. DEVS 기반 JSON 스키마

원자모델과 결합모델로 구성되는 모델은 공통 속성을 상위 모델 객체로부터 상속 받으며, 결합모델은 자식 요소로서 결합모델 혹은 원자모델을 1개 이상 가질 수 있다. 모델 객체는 기본 속성(아이디, 이름)과 입력 및 출력을 기술하는 속성을 가지고, 원자모델은 모델 객체를 상속하며 상태를 기술하는 states 속성과 상태천이를 기술하는 transitions 속성을 가지고, 결합모델은 모델 객체를 상속하며 자식 모델을 기술하는 nodes 속성과 모델간의 연결을 기술하는 links 속성 그리고 select 속성을 가진다. 생성된 컴포넌트에 대한 JSON 코드는 아래 그림 5에서 보는 바와 같이 자동으로 생성되고 저장된다.

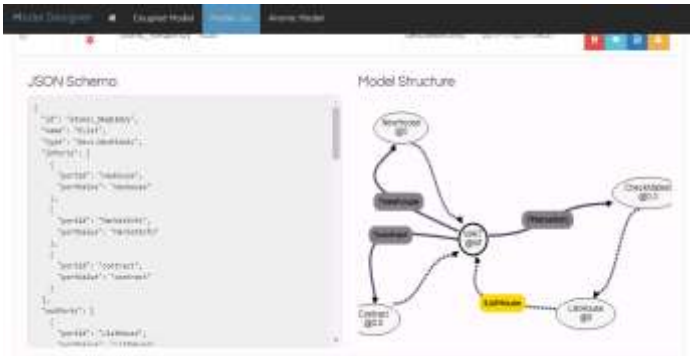


그림 5. JSON 코드 생성

3.3 DEVS 코드 생성

생성된 컴포넌트 모델에 대한 JSON 코드를 이용하여 컴포넌트 모델 코드를 생성한다. 그림 6은 원자모델과 소스코드의 생성 관계를 보인다. 원자 모델에 대한 클래스가 생성되며, 상태를 저장하기 위한 상태변수와 입/출력을 위한 변수들이 생성된다. 입력은 delta_x 함수로 기술되고, 입력 값에 따라서 상태천이가 이루어지고, 출력은 delta_y 함수로 기술되며 상태에 따라서 적절한 출력을 발생 시키도록 코드가 생성된다.

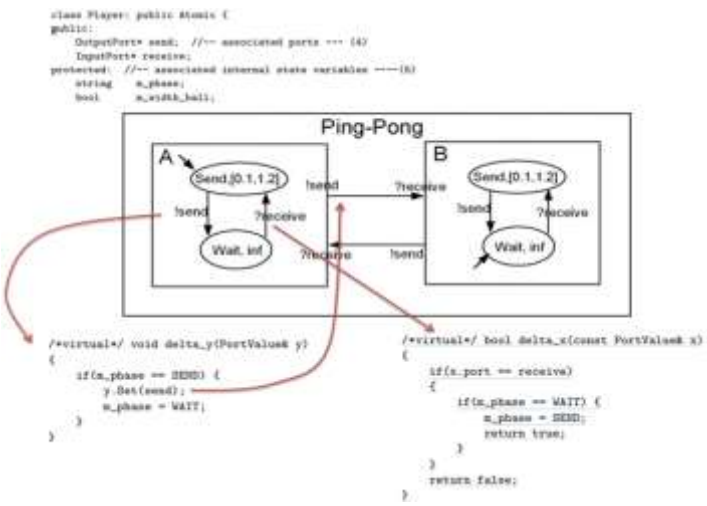


그림 6. 원자모델 코드 생성

그림 7은 결합 모델과 소스 코드의 생성 관계를 보여준다. 결합 모델에 대한 클래스가 생성되며, 결합 모델의 주요 특징은 내부 모델들의 연결 관계를 표현하는데 있으며, Internal Coupling(IC), External Input Coupling(EIC) 및 External Output Coupling(EOC) 세 가지 형태의 연결 관계에 대해서 자동으로 생성한다.

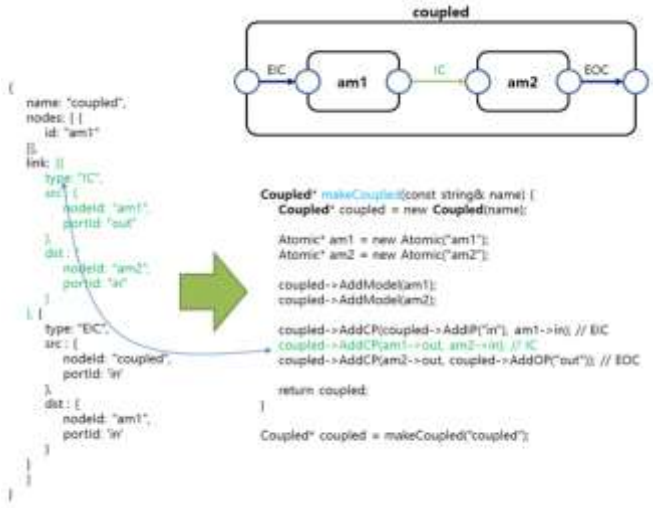


그림 7. 결합모델 코드 생성

그림 8은 DEVS 코드 생성기를 보인다. 모델의 특정버전에 해당하는 파일을 선택 후, 자동으로 생성된 DEVS 코드에 사용자 코드를 추가한다. 파일 목록에서 편집하고자 하는 파일을 선택하면 에디터 창에 소스파일이 나타나고, 소스 편집을 끝내고 저장 버튼을 누르면 DEVS 코드가 저장된다.

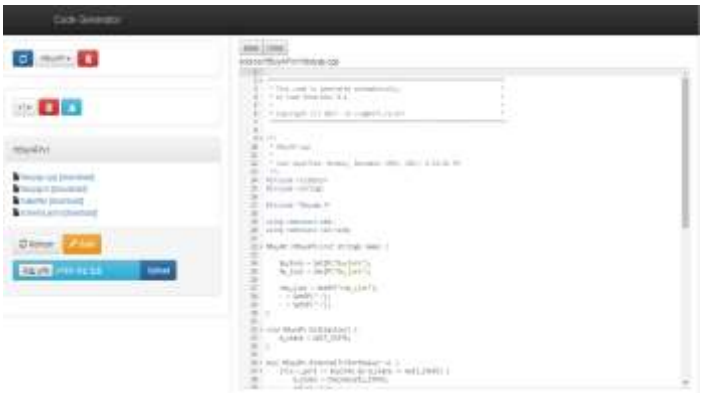


그림 8. DEVS 코드 생성기

3.4 DEVS 코드 컴파일 및 DB 저장

생성된 소스코드를 Makefile을 이용하여 컴파일을 수행하고, 그 결과로 .so 파일이 생성된다. 컴파일을 위한 Makefile은 소스코드 생성시점에 자동으로 생성되고, 개별 모델의 특정 속성에 대한 내용만 달라지고 공통 속성은 모델간에 동일하다. 실행결과는 stdout 및 stderr로 출력되고, 만약 stderr에서 출력이 발생하는 경우는 컴파일 실패로 인지하고 그 결과를 사용자에게 보여준다.

컴파일이 에러 없이 수행된 경우, JSON 파일, DEVS 소스코드 및 실행 화일을 저장소에 저장하여, 자가 진화 에이전트 시뮬레이션의 다른 모듈이 사용할 수 있도록 한다.

4. 결론

에이전트 시뮬레이션을 사용하여 장기간 시뮬레이션을 수행하면 오류가 누적되므로 시뮬레이션 결과가 부정확하게 되며, 오류 누적을 극복하기 위해서 자가 진화 에이전트 기반 시뮬레이션을 실행될 필요성이 있다. 에이전트 시뮬레이션 플랫폼에서 자가진화를 수행하기 위해서는 시뮬레이션 모델이 재구성되어야 하므로, 에이전트 모델링 및 시뮬레이션이 컴포넌트 기반으로 구현될 필요가 있다. 이를 위해 본 논문에서는 DEVS 기반의 시뮬레이션 모델을 GUI 로 생성하고, 생성된 컴포넌트 모델에 대한 메타데이터를 JSON으로 생성하고. 생성된 JSON 코드를 이용하여 시뮬레이션 코드 및 실행코드를 생성하는 시스템을

설계하고 구현하였다. 제안된 방법을 통하여 형식론에 익숙하지 않은 사용자도 재구성이 용이한 DEVS 기반의 에이전트 시뮬레이션 모델을 구현할 수 있다.

Reference

[1] A. Harding, "Challenges and opportunities of dynamic microsimulation modelling," Plenary paper presented to the 1st General Conference of the International Microsimulation Association, Vienna, Aug. 2007.

[2] M. Favreault and K. Smith, "A primer on the dynamic simulation of income model (DYNASIM3)," Urban Institute, Feb. 2004.

[3] A. Getchell, "Agent-based modeling," Physics, pp. 757-767, Jun. 2008.

[4] N. Geard, J. M. McCaw, A. Dorin, K. B. Korb, and J. McVernon, "Synthetic population dynamics: A model of household demography," Journal of Articial Societies and Social Simulation, 16(1) 8, Jan. 2013.

[5] K. Singh, M. Sajjad, and C. W. Ahn, "Towards full scale population dynamics modelling with an agent based and micro-simulation based framework," International Conference on Advanced Communication Technology (ICACT), pp. 495-501, Jul. 2015.

[6] J. W. Bae, B. W. Nam, K. E. Kim, J. S. Lee, and I. C. Moon, "Hybrid Modeling and Simulation of Tactical Maneuvers in Computer Generated Force," IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 942-947, Oct. 2017.

[7] J. W. Bae, E. H. Paik, and K. Singh, "Simulation-Based Population Dynamics Analysis: Korean Population Aging," Asian Simulation Conference (AsiaSim), pp. 358-367, Sep. 2016.

[8] J. Y. Jung, J. W. Bae, C. H. Lee, and E. H. Paik, "Component based Agent Simulation Modeling for Self-Evolving House Market Prediction," International Conference on Human and Social Analytics (HUSO), pp. 35-38, Jul. 2017.

[9] D. O. Kang, J. W. Bae, C. H. Lee, and E. H. Paik, "Data-driven Automatic Model Validation using Genetic Algorithm for Self-evolving Agent-based Social Simulation," International Symposium on Advanced Intelligent Systems(ISIS), pp. 25-30. Oct. 2017.

인공신경망을 이용한 코드 변경 추천 시스템

이준범*, 강성원*, 안휘*, 정필수*, 이선아**

*한국과학기술원
{leebduk, sungwon.kang, ahnhwi, psjung}@kaist.ac.kr
**경상대학교
saleese@gnu.ac.kr

Code Change Recommendation System using Artificial Neural Network

JunBeom Lee*, Sungwon Kang*, Ahn Hwi*, Pilsu Jung*, Seonah Lee**

* School of Computing, KAIST
**Gyeong Sang National University

요 약

코드 변경 추천 시스템은 프로그래머의 코드 탐색 시간을 감소시켜, 소프트웨어 진화 시 소프트웨어 개발 생산성을 증대시킨다. 기존 변경 추천 시스템들은 리비전 히스토리 또는 동작 히스토리로부터 연관 규칙을 찾아내어 변경할 파일 또는 함수를 프로그래머에게 추천한다. 하지만 기존 방법들이 활용하는 연관 규칙 마이닝 방법은 추천해야 하는 파일들이 있음에도 불구하고 추천하지 못하는 한계를 가진다. 본 연구에서는 코드 변경 추천 시스템에 최초로 인공신경망을 적용하여, 기존 방법들이 발견하지 못했던 새로운 연관 규칙들을 발견하고 이를 추천에 활용하는 방법을 제안한다. 제안 방법을 구현하여 이를 Eclipse Bugzilla 에 저장된 1,426 개의 동작 트레이스를 대상으로 실험한 결과, 기존 방법에서 발생하지 않았던 127 개의 새로운 추천이 발생하였다.

1. 서론

프로그래머들은 소프트웨어 진화 작업을 수행 할 때, 변경 해야 할 파일을 찾는 데 많은 시간을 소비한다 [1]. 코드 변경 추천 시스템(Code Change Recommendation System)은 프로그래머가 소프트웨어 진화 작업을 수행할 때 프로그래머에게 변경해야 할 파일을 추천해주는 시스템이다 [2]. 이 시스템은 변경해야 할 파일들을 추천하여 프로그래머들이 변경해야 할 파일을 탐색하는 시간을 줄일 수 있도록 돕는다. 이를 통해 소프트웨어 진화 작업에 소요되는 시간을 감소시킬 수 있다.

기존의 코드 변경 추천 시스템들은 입력 데이타인 코드 변경 히스토리에 따라 크게 두 가지 종류로 구분될 수 있다. 첫 번째는 리비전 히스토리를 이용하는 코드 변경 추천 시스템이다. Ying et al. [3]와 Zimmerman et al.[4]는 버전 관리 시스템에 저장된 리비전 히스토리에서 과거에 함께 자주 변경되었던 파일들 사이의 연관 규칙을 마이닝하여 변경 추천에 이용하였다. 두 번째는 프로그래머의 동작 히스토리를 이용하는 코드 변경 추천 시스템이다. Lee et al. [1]은 Eclipse Bugzilla 시스템에 저장된 동작 히스토리에서 프로그래머가 과거에 함께 열람하거나 변경한 파일에 관한 연관 규칙을 마이닝하여 변경할 파일을 추천한다. 하지만 이러한 기존 방법들은 함께 열람하지 않거나 함께 변경이 일어나지 않은 파일들일 경우 추천하지 못하는 한계를 가지고 있다. 그 이유는 기존 연구 방법들이 활용하는 연관 규칙 마이닝 방법[5]은 함께 열람되거나 변경된 파일들 만의 연관성을 고려하기 때문이다.

추천의 성능을 향상 시키기 위해, Alvarez et al.[6], Felden [7]은 인공신경망을 적용하였다. 이 연구들은 영화, 인터넷 쇼핑에 인공신경망을 적용하여 추천 시스템의 성능을 크게 향상 시켰다. 하지만 본 연구진의 조사에 의하면, 코드 변경 추천에 인공신경망을 적용한 연구는 지금까지 이루어지지 않았다.

본 연구에서는 인공신경망의 개념을 최초로 코드 변경 추천 시스템에 적용하여, 기존 방법들에서는 발견하지 못했던 새로운 연관 규칙들을 발견하고 이를 추천에 활용하는 방법을 제안한다. 제안 방법을 검증하기 위해, Eclipse Bugzilla 에 저장된 1,426 개의 동작 트레이스를 이용하여 실험한 결과, 기존의 방법 중 가장 높은 성능을 가진 MI[1]에서 추천하지 못했던 127 개의 새로운 추천이 발생하였다. 이 때 발생한 추천의 정확도와 재현율은 각각 37%, 20% 이다.

본 논문은 다음과 같이 구성 된다. 2 절에서는 관련 연구들을 소개한다. 3 절에서는 인공신경망을 이용한 코드 변경 추천 방법을 제안한다. 4 절에서는 실험 수행 결과를 설명한다. 마지막으로 5 절에서는 본 연구의 결론을 제시한다.

2. 관련 연구

소프트웨어 공학을 위한 추천 시스템은 “주어진 상황에서 소프트웨어 공학 작업에 귀중한 정보를 제공해주는 어플리케이션이다” [2]. 연구자들은 프로그램의 행동 히스토리를 마이닝하는 기술을 사용해 소프트웨어 진화 작업 수행과 관련된 파일을 추천하기 위한 코드 변경 추천 시스템들을 개발했다. 코드 변경 추천 시스템들은 입력값으로 사용하는 히스토리의 종류에 따라 크게 두 가지 종류로 구분될 수 있다. 또한 다른 영역의 추천 시스템에서는 인공 신경망을 적용하고 있다.

2.1 리비전 히스토리를 추천에 이용하는 연구

리비전 히스토리를 추천에 이용하는 연구들 [3], [4]은 과거에 함께 자주 변경(Edit)되는 파일 혹은 메소드들의 발생빈도를 고려하는 연관 규칙 마이닝(associati-on rule mining)을 사용했다 [5]. Zimmermann et al.는 연관 규칙 마이닝을 소프트웨어 리비전 히스토리에 적용하여 ROSE 라는 도구를 개발하였다 [4]. ROSE 는 어떤 파일에서 어떤 메소드가 변경 되었는지를 담고 있는 리비전 히스토리로부터 연관 규칙을 마이닝하여 추천 모델을 생성한다. 사용자가 어떤 메소드를 변경할 경우 ROSE 는 추천 모델로부터 해당 메소드와 연관된 연관 규칙을 통해 변경해야 할 메소드를 추천한다. Ying et al. [3]는 웹 브라우저 중 하나인 Mozilla 에서 제공하는 리비전 히스토리로부터 함께 자주 변경(Edit)되는 파일들의 변경 패턴을 마이닝하였다. 마이닝된 패턴은 개발자가 코드 변경 작업을 수행할 때 적절한 파일을 추천하기 위해 사용된다.

MI[1]의 연구결과에 따르면 리비전 히스토리는 오로지 변경(Edit)된 파일들의 정보만을 가지고 있기 때문에, 리비전 히스토리를 이용한 기존 코드 추천 시스템들은 소프트웨어 진화 작업에서 중요한 파일 열람(View)기록을 이용하지 못하는 한계가 있다.

2.2 동작 히스토리를 추천에 이용하는 연구

동작 히스토리를 추천에 이용하는 연구들 [1], [10]도 리비전 히스토리를 추천에 이용하는 연구들과 동일하게 연관 규칙 마이닝 기법을 사용하였다. Lee et al. [1] 연구진이 개발한 MI 는 ROSE[4]를 확장시킨 코드 변경 추천 시스템으로, 파일 변경(Edit)기록 뿐만 아니라 파일 열람(View)기록도 포함하고 있는 동작 히스토리를 사용하였다. 게다가 추천을 발생시키는 문맥을 구성하는 동작의 종류와 개수에 대한 조건을 자유롭게 설정하였다. 이를 통해, MI 는 ROSE 에 비해 높은 정확도와 재현율을 보였다. Maalej et al. [10]는 사용자 동작 데이터를 수집할 때 사용자가 사용하는 도구에 대한 정보도 함께 수집하여 새로운 작업을 할 때 어떤 도구를 사용하는 것이 적절한지 추천해주는 방법을 제안하였다.

위의 연구들은 동작 데이터를 정의하는 방법과 추천을 발생시키기 위한 문맥의 구성 방법에는 차이가 있지만, 연관 규칙 마이닝 방법을 사용해 문맥으로부터 추천 대상을 추천한다는 공통점이 있다. 하지만 연관 규칙 마이닝 방법[5]은 추천해야 하는 파일들임에도 불구하고, 함께 열람(View)하지 않은 파일들이거나 함께 변경(Edit)이 일어나지 않은 파일들일 경우 추천하지 못하는 한계를 가지고 있다.

2.3 인공 신경망을 추천 시스템에 이용하는 연구

인공 신경망을 다른 영역의 추천시스템에 적용한 여러 연구들 [6], [7]이 수행되었다. Alvarez et al. [6] 는 영화 추천 시스템의 성능을 향상시키기 위해 인공신경망을 기반으로 내용 정보와 관계 정보를 모두 고려하는 연구를 수행하였다. Felden et al. [7]는 인터넷 쇼핑 영역에서 제품 설명이나 테스트 보고서와 같은 의사결정 관련 텍스트 문서를 추천하는 시스템에 대해 인공신경망 뿐만 아니라 여러 머신러닝 기법을 적용한 연구를 수행하고 비교하였다. Lee et al [8]은 이슈의 개발자 할당에 딥 러닝 기법인 CNN 을 적용하여 기존보다 향상된 결과를 보였다. 하지만 본 연구진의 조사에 의하면, 코드 변경 추천에 인공신경망을 적용한 연구는 지금까지 이루어지지 않았다.

3. 인공 신경망 기반 변경 추천 방법

이 절에서는 사용하는 동작 히스토리의 구성 요소인 동작 트레이스와 추천을 발생시키는 문맥에 대해 살펴보고, 인공 신경망[10]을 이용한 코드 변경 추천 방법에 대하여 설명한다.

3.1 동작 트레이스

본 연구에서 동작 트레이스는 소프트웨어 진화 작업을 하는 동안 기록된 로그 파일 하나를 의미하며, 프로그래머의 동작들로 구성된다. 하나의 동작은 열람(View) 또는 변경(Edit)이 일어난 파일을 나타낸다. 동작 트레이스들이 모여 전체 동작 히스토리를 구성한다. 그림 1 는 사용자의 동작이 어떻게 동작 트레이스로 표현되는지를 보여준다.

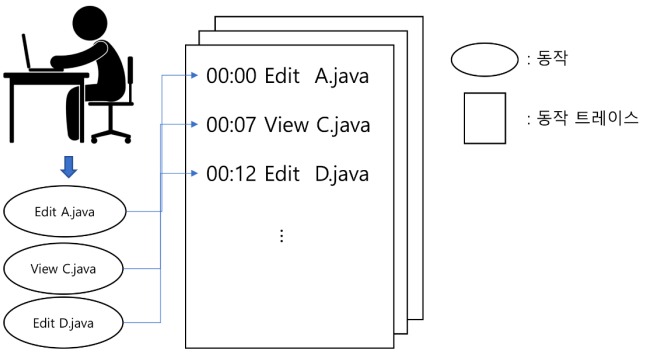


그림 1 동작 트레이스 생성 과정

3.2 문맥 형성

본 연구에서 문맥(Context)은 '추천을 발생시키는 질의문' 으로, 현재 사용자의 동작들로부터 형성된다. 현재 사용자가 어떤 작업을 진행 중 일 때, 문맥은 현재 사용자가 마지막으로 열람(View)하거나 변경(Edit)한 파일들로 구성된다. 따라서 현재의 사용자가 계속 파일들을 열람하거나 변경하면, 문맥은 변한다. 다시 말해, 문맥 C는 프로그래머가 마지막으로 열람(View)한 v개 파일의 집합과 마지막으로 변경(Edit)한 e개 파일의 집합의 쌍이다. 그림 2 는 3 개의 열람(View)파일과 1 개의 변경(Edit)파일로 구성된 문맥이 생성되는 과정을 나타낸다.

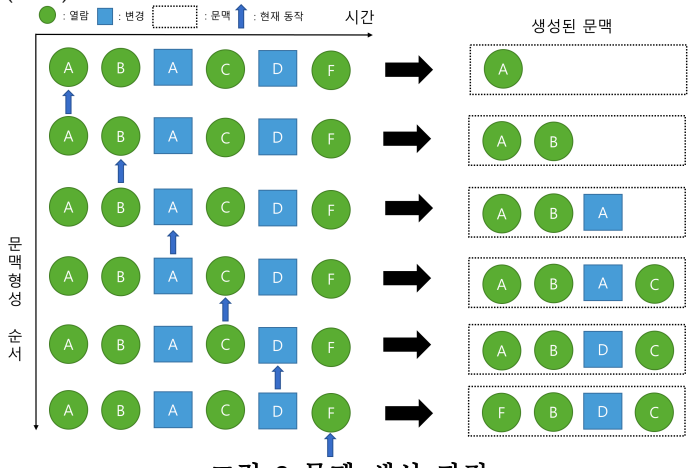


그림 2 문맥 생성 과정

3.3 인공 신경망을 이용한 학습 방법

본 연구에서 코드 변경 추천을 위해 사용하는 인공신경망은 학습 데이터로 문맥(Context)과 변경 추천 대상 파일들을 이용한다. 변경 추천 대상파일들은 문맥이 발생한 동작 트레이스에서 변경된 모든 파일들 중에 해당 문맥에 포함되어 있지 않은 파일들의 집합이다. 그림 3 은 6 개의 동작으로 이루어진 동작 트레이스로부터 학습 데이터를 생성하는 과정을 나타낸 것이다. 동작 트레이스에서 시간 순서대로 가장 최근 3 개의 열람(View) 동작과 1 개의 변경(Edit) 동작으로 구성된 문맥(Context)과 해당 동작 트레이스에서 발생한 변경 추천 대상을 쌍으로 학습 데이터들을 생성한다. 생성된 M 개의 학습 데이터 중에서 i 번째 학습 데이터의 문맥을 x_i, 변경 추천 대상을 y_i 로 나타낸다.

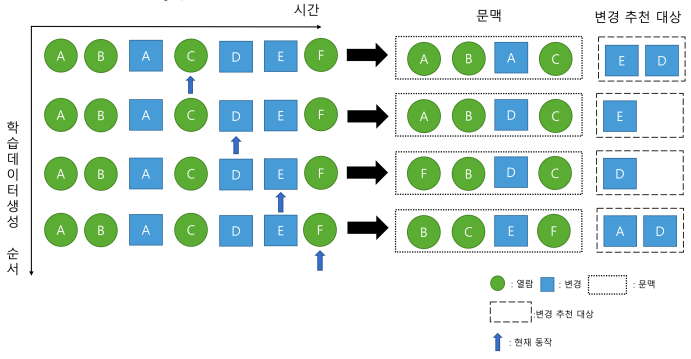


그림 3 학습 데이터 생성 방법

생성된 학습 데이터를 파일들의 고유 번호(N 개)를 가지고 있는 파일 사전을 숫자로 변환 한다. x_i는 임베딩 매트릭스 A(d×N 사이즈)를 통해 $v_i = \sum Ax_i$로 변환하고, 변환된 벡터 v와 가중치 행렬 W 그리고 바이어스 b를 이용해 추천 대상 벡터 a를 다음과 같이 생성한다.

$$a = \text{Sigmoid}(W*v+b).$$

a 는 0 에서 1 사이의 실수로 구성된 크기 N의 벡터로 표현되며, 벡터 a 의 원소 값이 0.5 이상이면 해당 원소 인덱스의 번호를 가진 파일이 변경 추천 파일로 추천된 것을 의미한다. 정답 추천 대상인 y_i는 one-hot 인코딩을 통해 크기 N의 벡터로 변환하고 변환된 벡터를 모두 더해 정답 추천 벡터로 만든다. 그림 4 은 인공신경망 모델의 전체 학습 과정을 나타낸다. 훈련 동안 임베딩 매트릭스 A와 W 그리고 b 는 정답 추천 대상과 추천 대상 사이의 시그모이드 교차 엔트로피 손실(Sigmoid cross-entropy loss)을 최소화함으로써 공동으로 학습된다. 학습은 아담(Adam) 기울기 하강을 사용하여 수행된다.

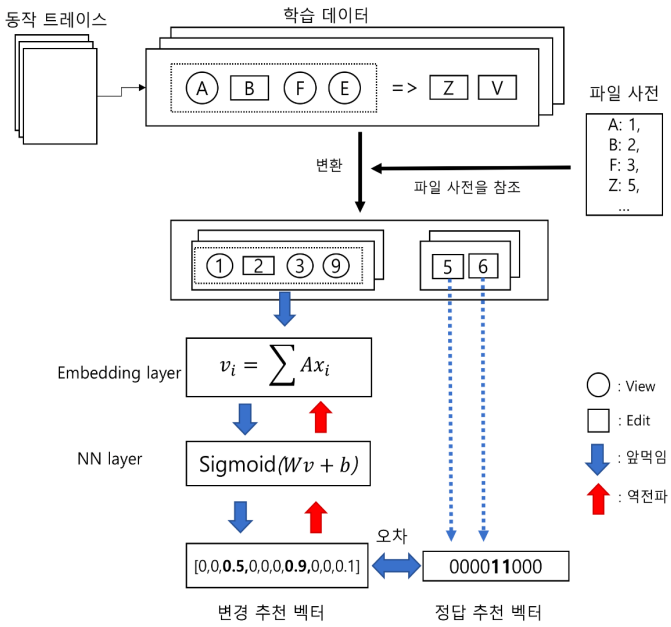


그림 4 인공신경망을 이용한 학습 방법

4. 실험

이 절에서는 본 연구에서 제안한 코드 변경 추천 시스템이 연관 규칙 마이닝 방법을 활용한 기존의 방법에서 발견하지 못했던 새로운 추천을 발생시키는지를 평가 하기 위해 실험을 수행한다.

실험 데이터로 Eclipse Bugzilla 에 저장된 1,426 개의 동작 트레이스를 사용하였다. 표 1 은 실험에 사용된 3 개의 프로젝트의 동작 트레이스의 개수를 보여준다.

표 1 Eclipse Bugzilla 프로젝트의 동작 트레이스

프로젝트	기간	동작 트레이스의 개수
Platform	2007-10-18 ~ 2011-05-25	582
PDE	2007-11-11 ~ 2011-02-25	536
ECF	2007-04-06 ~ 2011-05-20	308

실험에서 사용된 문맥은 3 개의 관찰 행동과 1 개의 변경 행동으로 구성되며, 변경 파일 추천의 개수는 최대 10 개로 높은 값부터 추천하며 비교 대상인 MI 도 동일하게 수행되었다. 새로운 변경 행동이 일어났을 때 현재 문맥을 이용하여 추천을 발생시킨다. 동작 트레이스 1 번을 제외하고 동작 트레이스 2 번부터 시작하여 직전까지의 동작 트레이스를 학습 데이터로 사용하고, 바로 다음 동작 트레이스를 검증 데이터로 사용하였다. 모든 동작 트레이스는 시간순서대로 순차적으로 실험하였다. 표 2 는 제안한 방법을 통해 발생한 추천 중에서 MI 에서 발생한 추천을 제외한, 제안된 방법에서만 발생한 추천의 정확도와 재현율을 각 프로젝트 별로 나타낸 표이다.

표 2 제안한 방법에서만 발생한 추천

프로젝트	정확도	재현율	새로운 추천의 개수
Platform	0.35	0.14	97
PDE	0.50	0.47	16
ECF	0.36	0.29	14
평균/합계	**0.37**	**0.20**	**127**

본 연구에서 발생한 모든 추천 중 MI 에서도 공통적으로 발생한 추천을 모두 제외하고 총 127 번의 추천이 존재하였다. 이는 MI 에서 발생한 모든 추천수 919 번의 15%에 해당하며, 발생한 추천의 평균 정확도와 재현율은 각각 37%과 20% 이다.

결론적으로, 제안된 방법에서만 발생한 추천의 정확도와 재현율이 기존 방법에 비해 낮지만, 기존의 연관 규칙 마이닝 방법에서는 전혀 찾아내지 못한 연관 규칙을 새로 제안한 방법으로 찾을 수 있었고 이를 활용해 새로운 추천이 발생되었다. 이는 같이 열람(View)하거나 변경(Edit)되지 않은 파일들 사이에도 연관 규칙이 존재하고, 이 연관 규칙이 추천에 사용될 수 있음을 보여준다.

5. 결론

본 연구에서는 인공신경망의 개념을 최초로 코드 변경 추천 시스템에 적용하여, 새로운 연관 규칙을 찾아내고 이를 추천에 활용하는 방법을 제안하였다. 제안된 방법을 Eclipse Bugzilla 에 저장된 1,426 개의 동작 트레이스를 대상으로 실험한 결과, 기존 방법에서 발생하지 않은 새로운 127 개의 추천이 발생하였고 이 때 발생한 추천의 정확도와 재현율은 각각 37%과 20% 이다. 제안된 방법에서만 발생한 추천의 횟수는 MI 에서 발생한 모든 추천수 919 번의 약 15%에 해당한다. 결론적으로, 기존 방법에서 사용된 연관 규칙 마이닝 방법에서는 발견 할 수 없었던 연관 규칙을 새로 제안한 방법으로는 찾을 수 있었으며, 이를 활용해 새로운 추천이 발생되었다.

실험에서 사용한 3 개의 프로젝트는 모든 소프트웨어 프로젝트를 대표할 수 없기 때문에 향후 연구로 3 개의 프로젝트 이외의 다른 소프트웨어 프로젝트를 대상으로 실험을 수행하고 결과를 비교할 계획이다. 또한 제안된 모델의 인공신경망 레이어를 단일 레이어로 사용하였지만 향후에 여러 개의 레이어로 확장하고, 단순 고유 번호를 통한 임베딩 기법이 아닌 파일 내용을 기반으로 임베딩 기법을 실험할 계획이다.

6. 참고 문헌

[1] S. Lee, S. Kang, S, Kim and M. Staats, “The Impact of View Histories on Edit Recommendations,” in IEEE TSE., vol 41, no. 3, pp. 314-330, 2015.

[2] M. Robillard, R. Walker, and T. Zimmermann, “Recommendation systems for software engineering,” IEEE Software., vol. 27, no. 4, pp. 80–86, Jul./Aug. 2010.

[3] A. T. T. Ying, G. C. Murphy, R. Ng and M. C. Chu-Carroll, “Predicting source code changes by mining change history,” IEEE Transactions on Software En gineering, 30, 2004, pp. 574–586

[4] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, “Mining version histories to guide software changes,” Proc. ICSE, vol. 31, no. 6, pp. 429–445, 2005.

[5] Karthikeyan, T., & Ravikumar, N. (2014). A survey on association rule mining. International Journal of Advanced Research in Computer and Communication Engineering, 3(1), 2278-1021.

[6] Alvarez, S. A., Ruiz, C., Kawato, T., & Kogel, W. (2011). Neural expert networks for faster combined collaborative and content-based recommendation. Journal of Computational Methods in Sciences and Engineering, 11(4), 161-172.

[7] Felden, C., & Chamoni, P. (2007, January). Recommender systems based on an active data warehouse with text documents. In System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on (pp. 168a-168a). IEEE.

Sun-Ro Lee, Min-Jae Heo, Chan-Gun Lee, Milhan

[8] Kim, and Gaeul Jeong. (2017). Applying deep learning based automatic bug triager to industrial projects. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017). ACM, New York, NY, USA, 926-931.

[9] W. Maalej, T. Fritz, and R. Robbes, “Collecting and processing interaction data for recommendation systems,” in Recommendation Systems in Software Engineering, M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, Eds. Springer BerlinHeidelberg, 2014, pp. 173–197.

[10] Rumelhart, D. E., Widrow, B., & Lehr, M. A. (1994). The basic ideas in neural networks. Communications of the ACM, 37(3), 87-93.

지속적 통합의 빌드 및 테스트 성능 향상을 위한 동적 스케줄링 기법

이장수, 권영우

경북대학교 컴퓨터학부

dellhartsmailbox@gmail.com, ywkwon@knu.ac.kr

Dynamically Scheduling Continuous Integration Services to Improve Their Build and Test Performance

Jangsoo Lee and Young-Woo Kwon

School of Computer Science and Engineering, Kyungpook National University

요 약

최근 각광받고 있는 지속적 통합(Continuous Integration)은 변경된 코드를 주기적으로 컴파일, 테스트 및 검증 후 형상 관리 시스템으로 통합하는 자동화된 소프트웨어 개발 프로세스로써 통합 과정에서 발생하는 문제를 조기에 발견 및 수정을 가능하게 하여 소프트웨어의 품질과 프로그래머의 생산성 향상에 큰 기여를 하고 있다. 하지만 다수의 CI 작업이 동시에 요청될 경우 CI 작업들 간의 자원 점유 문제로 인하여 CI 성능이 저하되는 문제가 발생함에 따라 본 논문에서는 다수의 CI 작업들을 위한 스케줄링 기법과 이를 채택한 CI 프레임워크를 소개한다. 제안된 CI 프레임워크는 동시에 여러 CI 작업 요청이 발생할 때 이전의 CI 작업 시간과 자원 사용 패턴을 파악하여 새로운 CI 작업의 실행 시간을 예측하고 기존의 CI 작업들 과의 자원 점유 경쟁을 피하여 CI 작업을 배치함으로써 CI 작업 성능을 최대 37% 향상시킬 수 있음을 보였다.

1. 서론

지속적 통합(Continuous Integration, 이하 CI)은 애자일 (Agile) 소프트웨어 개발 방법론과 함께 급격히 확산되고 있는 소프트웨어 개발 프로세스 중 하나이다. 애자일 개발 방법론에서는 프로젝트의 생명주기 동안 반복적인 개발을 통해 동작하는 프로토타입(Working Prototype)을 끊임없이 만들어내어 요구사항을 확인하고 수정하며 전체의 소프트웨어를 만든다. 애자일 개발 방법론의 등장으로 인해 여러 가지 개발 프로세스가 등장하였는데 지속적 통합은 변경된 코드를 주기적으로 컴파일, 테스트 및 검증 후 형상 관리 시스템으로 통합하고 통합 과정에서 발생하는 문제를 조기에 발견 및 수정을 가능하게 한다는 장점으로 최근 널리 사용되고 있다 [1, 2].

지속적 통합 기반의 개발에서는 프로그래머가 코드를 변경(추가 및 수정)하고 형상 관리 시스템에 커밋을 하면, 형상 관리 시스템은 CI 작업을 호출하여 컴파일 및 테스트 작업을 수행하고 코드 작성자와 팀원들은 CI 결과를 바탕으로 코드 리뷰 및 토의를 거쳐 통합을 결정하게 된다. CI가 구축되지 않은 개발 환경에서는 프로그래머가 잘못된 코드를 커밋할 경우 개발 후반에 장애로 발견되어 문제 해결 시간이 길어진다는 문제가 발생하지만 CI 기반의 개발 환경에서는 프로그래머들이 각자의 코드를 형상 관리 서버에 커밋하면 컴파일, 테스트, 코드 검증 및 분석이 이루어져 프로그램의 결함을 일찍 확인하여 소프트웨어의 품질을 지속적으로 관리할 수 있다는 장점이 있다.

지속적으로 코드를 통합함으로써 얻는 장점으로 인하여 많은 오픈소스 및 상용 프로젝트에서 CI 기반의 개발 환경을 도입하였으며 Software as a Service (SaaS) 형태의 CI 서비스들이 개발되어 기존의 형상 관리 시스템 (예, GitHub)과의 연동도 쉽게 이루어져 CI의 도입은 급격히 늘어나고 있다 [1]. 최근에는 CI 서비스를 통하여 컴파일에서 배포까지 한 번에 이루어지는 것이 가능해짐에 따라 CI의 미래는 더욱 밝을 것으로 전망된다. CI로 인한 여러 장점에도 불구하고 최근의 소프트웨어 개발에서는 여전히 많은

제약사항이 존재하는데, 본 논문에서는 여러 CI 작업이 동시에 요청될 경우 발생하는 CI 작업의 성능 저하 문제를 다루고 새로운 CI 프레임워크를 제안한다.

CI 작업은 컨테이너(Container)를 기반으로 동작되며 하나의 CI 작업은 하나의 컨테이너를 가짐으로써 서로 다른 CI 작업들 간의 영향을 최소화 한다는 장점이 있지만 여러 CI 작업들이 한 번에 요청될 경우 CI 작업들 간의 자원 점유 경쟁으로 인하여 CI 서비스의 성능 저하 문제가 발생한다. 특히, 충분한 컴퓨팅 자원을 가질 수 없는 1인 개발자 혹은 소규모의 스타트업의 경우 CI의 성능 저하는 생산성과 직결되므로 시급히 해결되어야 할 문제이다.

본 논문의 구성은 다음과 같다. 2장에서 문제점과 관련 연구에 대해 설명하고, 3장에서 제안하는 스케줄링 기법에 대해 설명한다. 4장에서 제안하는 기법에 대한 실험결과에 대해 논하고, 5장에서 향후 연구에 대해서 논하면서 본 논문을 마무리한다.

2. 문제점 분석 및 관련 연구

이번 장에서는 본 논문에서 풀고자 하는 문제와 관련 연구 동향에 대해서 설명한다.

2.1 문제점 분석

최근의 CI 환경은 컨테이너(Container)를 기반으로 구축되어 하나의 CI 작업은 하나의 컨테이너에서 동작한다. 컨테이너기반의 CI 환경에서는 서로 다른 CI 작업들 간의 영향을 최소화 하여 독립된 빌드 및 테스트 환경을 구축할 수 있다는 장점이 있지만 여러 CI 작업들이 동시에 요청될 경우 CI 작업들 간의 자원 점유 경쟁으로 인하여 성능 저하 문제가 발생할 수도 있다. 그림 1은 앞서 설명한 컨테이너 기반의 지속적 통합에서 발생 가능한 시나리오를 보여주고 있다.

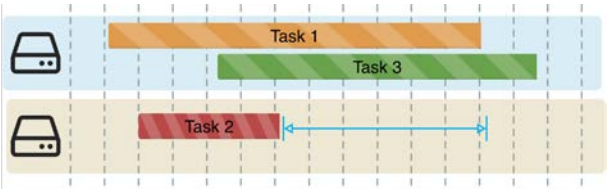


그림 1. CI 작업의 배치.

특히, 상용 CI 서비스를 사용하지 않고, 자체적으로 CI 기반의 개발환경을 구축할 때 충분한 컴퓨팅 자원을 가질 수 없는 경우 CI 작업은 제한된 컴퓨팅 자원을 놓고 경쟁을 하는 문제가 발생하기 쉽다. 그림 1에서 볼 수 있듯이, 두 개의 서버를 사용하여 CI 환경을 구축하고, 세 개의 CI 작업이 요청되었을 때 어떤 서버에 CI 작업을 배치하느냐에 따라 CI 작업 성능이 달라진다. CI의 성능 저하는 생산성과 직결되므로 시급히 해결되어야 할 문제이다.

그림 2는 앞서 기술한 문제점이 실제 개발환경에서 발생하는 경우를 시뮬레이션 한 것으로 오픈소스 프로젝트인 PHP-SRC에 대한 최근 30개의 커밋에 대한 CI 작업 결과를 보여주고 있다. 그림에서 볼 수 있듯이 각 커밋 간에는 실행 시간의 변화가 크게 없음을 볼 수 있다. 이는 최근의 소프트웨어 개발 방법이 작은 단위의 커밋을 장려하고 있으며 이를 잘 따른 프로젝트의 경우 커밋 간의 변화가 크지 않아 CI 작업 시간도 큰 차이가 없음을 확인할 수 있었다. 이러한 발견을 바탕으로 요청되는 CI 작업 시간 및 자원을 예측하여 CI 작업들을 스케줄링하고자 한다.

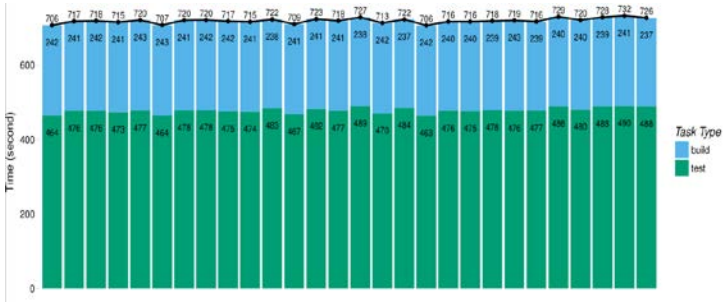


그림 2. PHP-SRC 프로젝트의 최근 30개의 커밋별 CI 작업 시간 (빌드 및 테스트 시간).

2.2 관련 연구 동향

소프트웨어 개발에 있어 테스팅 프로세스를 개선하기 위한 많은 노력들이 있었다. 특히 코드 변경에 따른 테스트 케이스 선택과 테스트 케이스의 우선 순위화는 짧은 시간에 테스트 작업을 더욱 효율적으로 하기 위한 것으로 오랫동안 연구되어 왔다. [1, 3, 5]. 코드의 커버리지를 분석하거나 [6], 이전의 데이터를 사용하여 [4] 테스트 케이스를 선택하고 이들을 정렬하여 테스트 하는 기법은 오래된 기법이지만 여전히 많이 사용되고 있다. 특히 Gligoric et al [2]은 CI 서비스와 테스트 케이스 선택을 통합하여 효율적인 CI 작업을 제공한다. 상용 소프트웨어에서 CI 도구의 성능 향상을 위한 작업들이 많이 이루어지고 있지만 본 연구에서 제안하는 스케줄링 기법에 대한 연구 및 논의는 아직 제한적으로만 이루어지고 있다. 본 논문에서 다루는 문제는 CI 작업의 효율적인 배치를 통한 성능 개선으로 테스트 케이스 선택을 통한 테스팅 프로세스 개선과는 서로 다르며, 상호 보완적으로 적용될 수 있다.

3. 시간 기반 스케줄링을 사용한 지속적 통합

이번 장에서는 개선된 CI 프레임워크에 대해서 소개한다. 개선된 CI 프레임워크는 스케줄링 기법을 도입하여 CI 작업들 간의 실행 순위를 정하여 각 CI 서버에 배치한다. 그림 3은 제안된 CI 프레임워크가 동작하는 순서를 보여주고 있다. 동일한 프로젝트에 대한 CI 수행 시간은 커밋 별로 큰 변화가 없음을 앞선 테스트로 알 수 있었으므로 새로 요청되는 CI 작업에 대한 시간 예측도 이를 바탕으로 이루어진다. CI 작업 시간에 대한 예측이 끝나면 CI 작업들 간의 우선순위를 조정하고 CI 작업이 할당되는 물리적 서버의 위치를 결정한다.

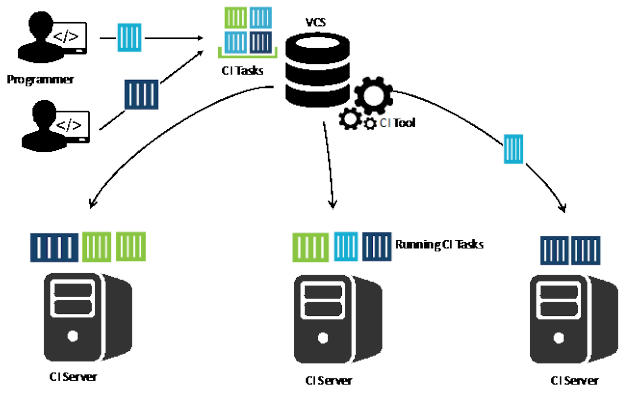


그림 3. 제안하는 CI 프레임워크의 동작.

그림 4는 앞서 설명한 스케줄링에 대한 것으로 이전의 CI 작업 시간 (빌드 및 테스트 시간)을 예측하고 CI 작업들간의 자원 점유 경쟁을 피하기 위하여 비어 있는 서버에 CI 작업을 할당한다. 비어있는 서버가 없는 경우에는 현재 수행 중인 CI 작업이 가장 빨리 끝나는 곳에 새로운 CI 작업을 할당한다. 아래 스케줄링 알고리즘은 FCFS (First Come First Served)에 기반한 것으로 우선 요청된 작업은 우선 배정하여 작업의 순서는 바꾸지 않는다.

Algorithm 1 Time-based Dynamic Selection (TDS)

```
1:  procedure TIME-BASED DYNAMIC SELECTION(a, b)
2:      nodes                                          ▷ all nodes
3:      min_task ← min_tasks(nodes)
4:      if min_task_num == limit then                  ▷ all servers are used
5:          return none                ▷ wait until there is an available node
6:      selected_node ← none                           ▷ selected node
7:      eTime ← ∞                                      ▷ estimated completion time
8:      for n ← nodes do
9:          if eTime <= time(nodes) then
10:             selected_node ← n
11:             eTime ← estimate_time(n)
12:     return selected_node
```

그림 4. 스케줄링 알고리즘

4. 실험 결과

제안된 CI 프레임워크의 검증을 위하여 두 개의 오픈소스 프로젝트-Git, PHP를 사용하여 실험을 진행하였다. 먼저 CI 서버는 아마존의 EC2 인스턴스 (c3.xlarge)에 Ubuntu 12.04 64 bits를 설치하고 Docker 1.12를 사용하여 컨테이너 기반의 CI 작업 환경을 구성 구성하였다. CI 프레임워크는 Travis CI[1] 사용하였으며 Git, PHP의 CI 작업들과 자원 점유 경쟁을 벌이는 CI 작업은 FFMpeg 프로젝트로, 상당히 높은 CPU 점유율을 차지하는 테스트들로 구성되어 있다.

표 1은 여러 스케줄링 기법들은 사용하여 CI 작업 시간을 측정한 것으로, LRU (Least Recently Used), MRU (Most Recently Used), Random, 그리고 본 논문에서 제안하는 시간 기반 스케줄링(TDS)의 CI 작업 성능을 비교하고 있다. 모든 작업들은 FFMpeg과 함께 경쟁을 하고 있으며, 프로젝트명-1에서는 컨테이너의 CPU 점유에 제한을 걸지 않은 것이며 프로젝트명-2에서는 각 컨테이너마다 CPU 자원의 점유에 제한을 둔 것이다 (예, CPU 코어 갯수 제한).

표 1. 여러 스케줄링 기법들을 사용한 작업시간 비교 (단위 초).

	PHP-1	PHP-2	GIT-1	GIT-2
LRU	877.9614	906.2550	570.4184	700.1800
MRU	916.1221	929.4899	699.7182	702.8452
Random	909.7817	909.5335	594.4979	634.9804
TDS	**888.4284**	**908.599**	**560.3035**	**691.0834**

표1 에서 볼 수 있듯이 제안한 시간 기반 스케줄링 기법은 다른 스케줄링 기법에 비해서 최대 약 24%가 더 빠름을 알 수 있었고, 스케줄링 기법이 없는 변경되지 않은 기존 Travis CI 보다는 37%의 성능 향상이 있음을 알 수 있었다. 하지만 제안하는 스케줄링 기법은 항상 최적의 결과를 보장하지는 않는데, PHP-1, PHP-2 의 경우에는 LRU 기법이 더 나은 결과를 보였고, GIT-2의 경우는 Random 기법이 더 나은 결과를 보였다. 그러나 이러한 경우에도 제안하는 스케줄링 기법과 LRU 및 Random 기반 기법들과 성능 차이는 미미한 수준이며, 이 경우를 제외한 나머지 경우에는 성능이 제안하는 기법보다 더 떨어짐을 알 수 있다. 그러므로 제안하는 기법은 항상 최적의 성능을 보장하지는 않지만 항상 우수한 성능을 보임을 알 수 있다.

5. 결론 및 향후 연구

본 논문에서는 CI 작업들의 CI 서버 배치 시 발생할

[1] https//hub.docker.com/r/travisci/

수 있는 CI 작업들간 자원 점유 경쟁 문제를 해결하기 위하여 시간에 기반한 스케줄링 기법을 제안하고 이를 실제 CI 프레임워크에 구현하여 효용성을 보였다. 향후 연구에서는 정확한 CI 작업 시간의 예측을 위하여 추가되는 코드가 미치는 영향을 고려할 예정이며, 자원 사용 패턴(예, CPU 위주의 프로그램, 메모리 사용이 높은 프로그램, 네트워크 사용이 높은 프로그램)을 함께 파악하여 동일한 자원에 대한 점유 경쟁이 발생하는 경우를 줄일 수 있도록 할 것이다.

참고문헌

[1] S. Y. Hea and E. G. Kim, “Design and implementation of the differential contents organization system based on each learner's level,” *The KIPS Transactions: Part A*, Vol.18, No.6, pp.19-31, 2011.

[2] S. Russell, P. Norvig, “Artificial Intelligence: A Modern Approach,” 3th ed., New York: Prentice Hall, 2009.

[3] J. L. Hennessy and D. A. Patterson, “Instruction-level parallelism and its exploitation,” in *Computer Architecture: A Quantitative Approach*, 4th ed., San Francisco, CA: Morgan Kaufmann Pub., ch. 2, pp.66-153, 2007.

[4] D. B. Lenat, “Programming artificial intelligence,” in *Understanding Artificial Intelligence*, Scientific American, Ed., New York: Warner Books Inc., pp.23-29, 2002.

[5] A. Stoffel, D. Spretke, H. Kinnemann, and D. A. Keim, “Enhancing document structure analysis using visual analytics,” in *Proceedings of the ACM Symposium on Applied Computing*, Sierre, 2010, pp.8-12.

[6] J. Y. Seo, “Text driven construction of discourse structures for understanding descriptive texts,” Ph.D. Dissertation, University of Texas at Austin, TX, USA, 1990.

피처 간 상호작용 패턴 정의[1]

이영우, 이선열, 채흥석
부산대학교 전기전자컴퓨터공학과
{amorepooh, sun302412, hschae}@pusan.ac.kr

Definition of Feature Interaction Pattern

Youngwoo Lee, Seonyeol Lee, Heung-Seok Chae
Dept. of Electrical & Computer Engineering, Pusan National University

요 약

필수 안전 시스템은 결함이 발생할 경우 인명, 재산 피해와 같은 치명적인 사고로 이어질 수 있다. 다양하고 많은 기능을 제공하는 시스템은 개별적인 기능들이 정상적으로 동작함에도 불구하고, 여러 기능들이 함께 수행될 때 예상치 못한 동작을 수행할 수 있다. 기존 피처 간 상호작용에 대한 연구들은 구성 요소 간의 상관관계나 시간에 따른 값의 변화로 인한 상호작용을 고려하지 않는다. 본 연구에서는 피처와 변수에 대해 정의하고, 변수 간의 상관관계와 시간에 따른 변수 값의 변화를 고려한 피처 간 상호작용 패턴을 정의한다.

1. 서 론

현대에서 사용되는 시스템들은 다양하고 많은 기능을 제공한다. 이러한 시스템들은 설계와 구현이 복잡하여 결함이 발생할 가능성이 높다. 필수 안전(safety critical) 시스템은 결함이 발생할 경우 인명, 재산 피해와 같은 치명적인 사고로 이어질 수 있다.

다양한 기능을 제공하는 시스템은 개별적인 기능들이 정상적으로 동작함에도 불구하고, 여러 기능들이 함께 동작할 때 예상치 못한 동작을 수행할 수 있으며[1], 이를 피처 간 상호작용이라 한다. 피처 간 상호작용이 발생하는 조건은 분야 별로 상이하며, 시스템의 목적과 상황에 따라 다르다.

기존 피처 간 상호작용에 대한 연구들은 구성 요소 간의 상관관계나 시간에 따른 값의 변화로 발생하는 피처 간 상호작용을 고려하지 않는다. 본 연구에서는 상호작용이 발생하는 요소들을 분석하고, 상관관계와 시간에 따른 값의 변화를 고려한 피처 간 상호작용 패턴을 정의한다. 이를 바탕으로 추후 상호작용에 대한 연구 방법과 방향을 제시하고자 한다.

2. 관련 연구

피처 간 상호작용에 대한 연구는 스마트 폰, 지능형 관리 시스템, 자동차 시스템 등 다양한 분야에서 연구가 수행되고 있다[2]. 각 분야마다 피처 간 상호작용에 대한 정의가 상이하며, 시스템의 상황과 목적에 따라 다르다.

Metzger and Webel의 연구[3]는 빌딩 제어 시스템의 요구사항을 독립적인 수행 단위인 작업으로 상세화하고, 시스템을 구성하는 센서와 액추에이터를 객체로 식별하여, 작업과 객체 간의 의존성을 분석한다. 분석된 정보에서 서로 다른 두 작업이 하나의 객체와 의존성이 있을 때, 두 작업은 상호작용이 있는 것으로 간주한다.

Kwon et al.의 연구[4]는 지능형 빌딩 관리 시스템의 행위와 각 행위가 사용하는 자원을 분석하며, 자원은 행위 목적에 따라 참 또는 거짓의 상태를 가진다. 이 연구는 서로 다른 두 행위가 공통적인 하나의 자원과 의존성이 있으며, 두 행위가 자원의 상태를 상반되도록 할 때 상호작용이 있는 것으로 간주한다.

Le Guilly et al.의 연구[5]는 스마트 홈을 대상으로 발생 가능한 이벤트, 이벤트가 발생하기 위한 조건, 조건이 만족되었을 때 수행하는 행위를 식별한다. 이벤트 조건은 만족해야하는 특정 값이나 값의 범위, 시간을 명시한다. 이 연구는 여러 이벤트가 동시에 수행될 때, 이벤트의 조건들이 상충되는 경우 이벤트 간에 상호작용이 존재하는 것으로 간주한다.

Juarez-Dominguez et al.의 연구[2]는 자동차 시스템을 대상으로 피처와 액추에이터, 그리고 시스템 환경 요소를 식별하고, (I)두 피처에 의해 공유되는 액추에이터와 (II)같은 환경 요소를 공유하는 액추에이터를 식별한다. 그리고 액추에이터에 할당 가능한 값의 타입과 값을 할당하는 시점을 정의한다. (I)의 경우는 액추에이터에 할당되는 두 값의 차이가 허용 범위 이상이거나, 값을 할당하는 시점의 차이가 허용 범위 이하일 때 피처 간 상호작용으로 간주하며, (II)의 경우는 환경 요소에 할당하는 값이 허용 범위 이상이거나, 값을 할당하는 시점의 차이가 허용 범위 이하일 때 피처 간 상호작용으로 간주한다.

기존 연구들은 시스템을 구성하는 요소와 요소들 간의 의존성을 파악하고, 요소들의 상태나 값에 대한 조건이나 시간 제약을 기준으로 피처 간 상호작용을 식별하였다. 그러나 위의 기준들 이외에 요소들 간의 상관관계, 시간에

[1] 이 논문은 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No.2016R1D1A1B03934344)

따른 요소들의 값의 변화에 대해서는 고려되지 않고 있다. 예를 들어 난방과 가습이 동시에 수행될 경우 난방은 공기 중의 습도를 낮추기 때문에 가습이 제대로 수행되지 않을 수 있다. 다른 예로 차량이 코너를 돌 때, 스티어링 휠이 코너 방향으로 회전하였다가 일정 시간 후에 반대 방향으로 회전해야 하지만 실제로 수행되지 않을 수 있다.

기존 연구들의 상호작용 식별 방법은 상관관계로 인한 상호작용과 시간에 따른 값의 변화로 인한 상호작용을 식별할 수 없다. 따라서 본 연구에서는 상관관계와 시간에 따른 변수 값의 변화를 고려한 피처 간 상호작용 패턴을 정의한다.

3. 시스템 구성요소

본 절에서는 피처 간 상호작용 패턴을 정의하기 위해 시스템의 구성요소로써 피처와 변수에 대해 정의한다.

정의 1. 피처.

피처란 시스템을 구성하는 독립적인 단일 기능으로써, 상태 기계 다이어그램을 이용하여 피처를 표현한다. 피처에 대한 정의는 아래와 같다.

$$S = (S, s_0, V, \delta, F)$$

- S: 피처를 구성하는 상태 집합이며, 공집합이 아니다.
- s_0: S의 구성요소이며, 피처의 초기 상태를 나타낸다.
- V : 상태 기계의 입력으로 사용되는 변수의 집합이며, 유한하고 비어있지 않은 집합이다.
- δ: 상태 전이 함수이며, $S \times V \rightarrow S$로 표현한다,
- F: 최종 상태의 집합이며, S의 부분 집합이다.

정의 2. 변수

변수는 피처의 입력으로 사용되며, 전이 함수에서 정의(define)되거나 참조(use)된다. 변수의 정의는 아래와 같다.

$$V = (n, T)$$

- n: 변수의 이름을 나타낸다.
- T : 변수의 형식을 나타내며, 변수의 형식은 Discrete와 Continuous 두 가지가 있다.

$$T = \{Discrete, Continuous\}$$

그림 1은 엘리베이터 문을 제어하는 피처의 상태 기계 다이어그램을 보여준다.

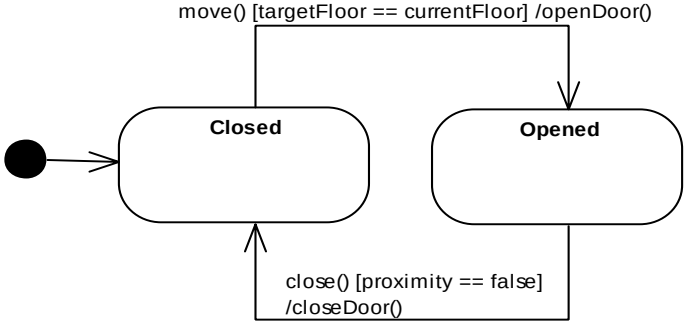


그림 1 엘리베이터 문 제어 피처

엘리베이터 문 제어 피처에는 Closed와 Opened 상태가 있으며, 두 개의 상태 전이 함수가 있다. 사용되는 변수로는 층을 나타내기 위한 Continuous 형식의 targetFloor, currentFloor가 있으며, 문 근처의 사람이나 물체를 감지하기 위한 Discrete 형식의 proximity가 있다.

4. 피처 간 상호작용 패턴

본 절에서는 피처 간 상호작용이 발생하기 위한 요소들을 정의하고, 요소들 간의 관계를 통해 피처 간 상호작용 패턴을 정의한다. 그림 2는 피처 간 상호작용 요소를 보여준다.

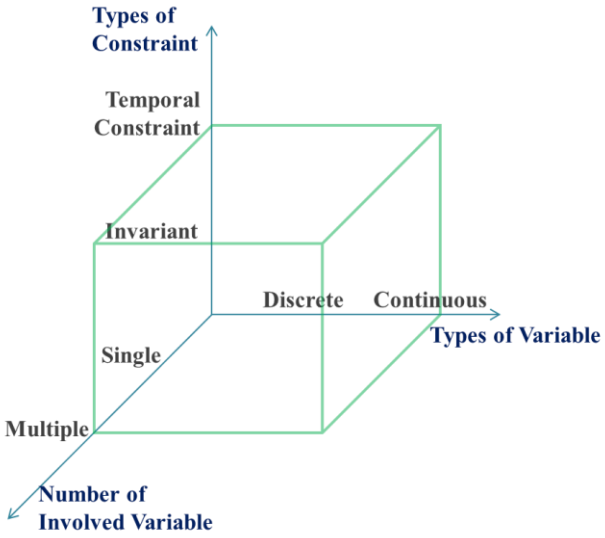


그림 2 피처 간 상호작용 요소

피처 간 상호작용이 발생하기 위한 요소로써 연관된 변수의 개수(Number of Involved Variable), 변수의 형식(Types of Variable), 그리고 제약 사항의 형식(Types of Constraint) 세 가지가 있다. 피처 간 상호작용 패턴의 정의는 아래와 같다.

정의 3. 피처 간 상호작용 패턴

피처 간 상호작용은 변수의 개수와 변수의 형식, 그리고 제약 사항의 형식에 따라 결정된다. 따라서 상호작용 패턴의 정의는 다음과 같다.

$$FIP = (NIV, ToV, ToC)$$

- NIV: 피처 간 상호작용에 연관된 변수의 개수를 의미한다. 상호작용은 단일 변수에 대해 발생하거나, 서로 상관관계가 있는 여러 변수들 간에 발생한다. 따라서 NIV의 정의는 다음과 같다.

$$NIV = \{Single, Multiple\}$$

차량에서 가속 기능과 감속 기능은 속도라는 하나의 변수에 대해 상호작용이 발생할 수 있으며, 온도와 습도 변수는 서로 간의 상관관계로 인해 상호작용이 발생할 수 있다.

- ToV: 피처 간 상호작용에 연관된 변수의 형식을 의미한다. 변수의 형식은 이산 변수와 연속 변수가 있다. 따라서 ToV는 다음과 같이 정의한다.

$$ToV = \{Discrete, Continuous\}$$

이산 변수는 참/거짓이나 상태를 나타내는 열거형 값을 나타내며, 연속 변수는 정수나 실수와 같은 연속적인

값을 나타낸다.

- ToC : 피처 간 상호작용의 제약 조건 형식을 의미한다. 제약 조건에는 시스템이 동작하는 모든 시점에서 만족해야하는 불변식(Invariant)과 특정 시점에만 만족해야하는 시간 제약(Temporal Constraint)이 있다. 따라서 ToC는 다음과 같이 정의한다.

$$ToC = \{Invariant, Temporal\ \ Constraint\}$$

차량의 속도는 최저 속도와 최대 속도를 불변식으로 명시할 수 있으며, 차량을 추월하는 경우를 시간 제약으로 명시할 수 있다. 앞 차량을 추월하는 경우 가속하여 앞 차량을 추월하고, 추월 후에는 이전 주행 속도로 감속해야 한다. 불변식과 시간 제약에 대한 정의는 다음과 같다.

정의 4. 불변식(Invariant)

불변식이란 시스템에서 사용되는 변수가 모든 시점에서 만족해야하는 값이며, $inv(V)$ 로 표기한다. 불변식은 Object Management Group에서 제정한 Object Constraint Language를 준수하여 정의한다.

정의 5. 시간 제약(Temporal Constraint)

시간 제약이란 시스템에서 사용되는 변수가 특정 시점에서 만족해야하는 패턴의 집합이며, $Temp(V)$ 로 표기한다. 시간 제약의 정의는 다음과 같다.

$$Temp(V) = \{temp(V)\}$$

하나의 패턴을 의미하는 $temp(V)$ 는 변수의 형식에 따라 (I)이산 변수인 경우와 (II)연속 변수인 경우로 구분한다.

(I) 이산 시간 제약(Discrete Temporal Constraint)

변수 V 가 가질 수 있는 값의 집합이 V_d 이고, 값을 유지하는 시간이 $T(s)$ 일 때 이산 시간 제약을 다음과 같이 정의한다.

$$temp(V) =< (E, T) *>, where\ \ E \in V_d$$

예를 들어 히터(V)가 On, Off의 값을 가질 때($V_d = \{On, Off\}$), 10분간 동작하고, 2분간 동작하지 않는 패턴은 아래와 같이 정의할 수 있다.

$$temp(V) =< (On, 600), (Off, 120) >$$

(II) 연속 시간 제약(Continuous Temporal Constraint)

변수 V 가 가질 수 있는 값의 범위가 V_c 이고, 값을 유지하는 시간이 $T(s)$ 일 때 연속 시간 제약을 다음과 같이 정의한다.

$$temp(V) =< (min, max, T) *>, where\ \ min, max \in V_c$$

예를 들어 차량의 속도(V)가 0km/h에서 200km까지 범위를 가질 때($V_c = [0, 200]\text{km/h}$), 10초간 속도를 110~120km/h로 유지하고, 5초간 속도를 90~100km/h로 유지하는 패턴은 아래와 같이 정의할 수 있다.

$$temp(V) =< (110, 120, 10), (90, 100, 5) >$$

[3]는 변수의 상태를 Read와 Write로 구분하고, 변수의 값에 대해 제약 사항을 불변식으로 정의하였으며, [4]는 변수의 상태를 True와 False로 구분하고, 변수의 값에 대한 제약 사항을 불변식으로 정의하였다. 그리고 [5]과 [2]은 이산 변수와 연속 변수에 대해 불변식과 시간 제약을 정의하였다. 표 1은 기존 연구들을 본 장에서 정의한 피처 간 상호작용 패턴으로 분류한 결과를 보여준다.

표 1 피처 간 상호작용 패턴에 따른 기존 연구 분류

FID 요소 \ 관련 연구		[3]	[4]	[5]	[2]
NIV	*Single*	O	O	O	O
	Multiple	-	-	-	-
ToV	*Discrete*	O	O	O	O
	Continuous	-	-	O	O
ToC	*Invariant*	O	O	O	O
	Temporal Constraint	-	-	O	O

5. 스마트 홈 시스템

본 절에서는 스마트 홈 시스템을 대상으로 피처 간 상호작용 패턴을 적용하여 각 요소에 따른 피처 간 상호작용을 설명한다. 그림 3은 스마트 홈 시스템의 개요 와 제약 사항을 보여준다.

스마트 홈 시스템은 네 개의 피처가 있으며, 변수로 냉방기, 난방기, 가습기, 온도, 습도가 있다. 각 변수는 형식과 불변식, 그리고 시간 제약이 정의되어 있으며, 냉방기와 난방기, 온도와 습도 사이에는 상관관계가 존재한다.

I. Single Discrete Invariant

가습기는 f_3 과 f_4 에 의해 공유되며, Boolean 형식이므로 True와 False의 값을 가진다. 가습기의 불변식은 희망 습도가 항상 40 초과에서 75미만이어야 한다. 그러나 f_3과 f_4에 의해 희망 습도가 40이하가 되거나 75이상이 되는 경우 Single Discrete Invariant 상호작용이 발생하는 것으로 간주된다.

II. Single Discrete Temporal Constraint

가습기의 시간 제약은 희망 습도를 45이상 50이하로 20분간 유지 후, 60 이상 70 이하로 40분간 유지해야 한다. 그러나 f_3과 f_4에 의해 희망 습도의 변화가 정의된 패턴대로 동작하지 않는 경우 Single Discrete Temporal Constraint 상호작용이 발생하는 것으로 간주된다.

III. Single Continuous Invariant

온도는 냉방기와 난방기에 의해 공유되며, Float 형식이므로 실수의 값을 가진다. 온도의 불변식은 실제 온도가 항상 10 초과 35 미만이어야 한다. 그러나 냉방기와 난방기에 의해 실제 온도가 10 이하 또는 35이상이 되는 경우 Single Continuous Invariant 상호작용이 발생하는 것으로 간주된다.

IV. Single Continuous Temporal Constraint

온도의 시간 제약은 실제 온도를 20 이상 22 이하로 30분간 유지 후, 25 이상 28 이하로 10분간 유지하거나, 실제 온도를 27 이상 29 이하로 30분간 유지 후, 23 이상 26 이하로 10분간 유지해야한다. 그러나 냉방기와 난방기에 의해 실제 온도의 변화가 정의된 패턴을 위반하는 경우 Single

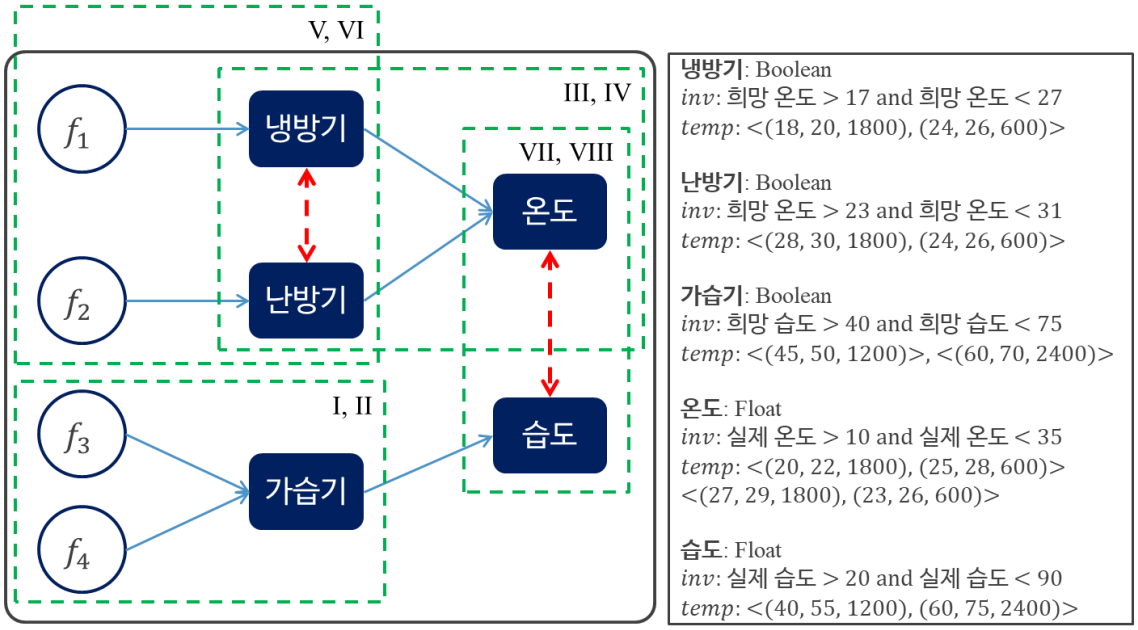


그림 3 스마트홈 시스템 개요 및 제약 사항

Continuous Temporal Constraint 상호작용이 발생하는 것으로 간주된다.

V. Multiple Discrete Invariant

냉방기와 난방기는 모두 Boolean이며, 둘 다 희망 온도를 조절하는 장치이므로 상관관계가 존재한다. 따라서 희망 온도가 27 이상이 되면 냉방기의 불변식을 만족하지 못하게 되고, 희망 온도가 23 미만이 되면 난방기의 불변식을 만족하지 못하게 된다. 이러한 경우는 Multiple Discrete Invariant 상호작용이 발생한 것으로 간주한다.

VI. Multiple Discrete Temporal Constraint

냉방기와 난방기가 함께 동작할 때, 상관관계로 인해 서로의 시간 제약 패턴을 만족하지 못하는 경우가 발생할 수 있으며, 이를 Multiple Discrete Temporal Constraint 상호작용이 발생한 것으로 간주한다.

VII. Multiple Continuous Invariant

온도와 습도는 모두 Float이며, 과학적으로 온도가 높아지면 습도가 낮아지고, 온도가 낮아지면 습도가 높아진다. 따라서 두 변수의 불변식을 모두 만족하지 못하는 경우가 발생할 수 있으며, 이러한 경우 Multiple Continuous Invariant가 발생한 것으로 간주한다.

VIII. Multiple Continuous Temporal Constraint

온도와 습도는 과학적인 상관관계로 인해 서로의 시간 제약 패턴을 만족하지 못하는 경우가 발생할 수 있으며, 이를 Multiple Continuous Temporal Constraint가 발생한 것으로 간주한다.

6. 결론 및 향후 연구

피처 간 상호작용에 대한 기존 연구들은 구성 요소들 간의 의존성을 분석하고, 값의 상태나 범위, 또는 시간에 대하여 조건을 명시하고 이를 위반하는 경우 피처 간 상호작용으로 간주하였다. 그러나 구성 요소들 간의 상관관계나 시간에 따른 값의 변화로 인한 상호작용은 식별하지 못하였다.

본 연구에서는 피처 간 상호작용이 발생하기 위한 요소들을 분석하고, 변수 간의 상관관계와 시간에 따른 변수 값의 변화를 고려한 피처 간 상호작용 패턴을 정의하였다. 향후 연구에서는 피처 간 상호작용 패턴을 기반으로 상호작용을 식별하고, 테스트 케이스를 생성하는 연구를 수행할 계획이다.

참 고 문 헌

[1] Calder, M., & Magill, E., Feature Interactions in Telecommunications and Software Systems VI, IOS Press, 2000.

[2] Juarez-Dominguez, A. L., Day, N. A., & Joyce, J. J., “Modelling feature interactions in the automotive domain,” In Proceedings of the 2008 international workshop on Models in software engineering, pp. 45-50, 2008.

[3] Metzger, A., & Webel, C., “Feature Interaction Detection in Building Control Systems by Means of a Formal Product Model,” Feature Interaction Workshop, pp. 105-122, 2003.

[4] Kwon, J. J., Hong, J. E., & Chung, L., “Collision detection and resolution of hazard prevention actions in safety critical systems,” Journal of Systems and Software, Vol. 118, pp. 1-18, 2016.

[5] Le Guilly, T., Nielsen, M. K., Pedersen, T., Skou, A., Kjeldskov, J., & Skov, M., “User constraints for reliable user-defined smart home scenarios,” Journal of Reliable Intelligent Environments, Vol. 2, No. 2, pp. 75-91, 2016

지식베이스 기반 L-V-C 매핑 방법 연구

이동재 ○, 손윤희, 이규철 †

충남대학교 컴퓨터공학과

ehdwo115@gmail.com, {mellow211,kclee}@cnu.ac.kr

Research on Knowledge base L-V-C Mapping Method

Dong-Jae Lee ○, Yun-Hee Son, Kyu-Chul Lee †

Department of Computer Science and Engineering, Chungnam National University

요 약

국방 훈련 체계는 실기동 모의 훈련을 위한 L(Live)체계, 가상 훈련을 위한 V(Virtual)체계, 전투지휘 훈련을 위한 C(Constructive)체계가 사용되고 있다. 최근 실제 전장과 동일한 환경을 구현하기 위해 L-V-C 통합 훈련 체계에 대한 연구가 진행되고 있다. 하지만 L-V-C 통합훈련체계는 각각의 L, V, C 특성에 따라 서로 다른 미들웨어가 사용되기 때문에 미들웨어 간 연동 시 문제점이 발생한다. 각 체계에서 사용되는 미들웨어는 HLA, DDS, DIS로 이들은 서로 다른 API 및 데이터 형식(FOM, PDU, TOPIC)을 사용하고 있다. FOM, PDU 같은 경우 데이터 형식의 표준이 있지만, TOPIC은 데이터 형식 표준이 정해져 있지 않아 이종 미들웨어 간 상호 연동 시 문제가 발생한다. 본 논문에서는 이종 미들웨어의 데이터 연동 문제를 해결하기 위해 HLA FOM 데이터 형식을 기반으로 키워드를 추출하고, 확장하여 온톨로지 모델링을 통해 지식베이스를 구축하였다. 또한 구축한 지식베이스와 가중치 기반의 검색 알고리즘을 이용하여, FOM과 TOPIC간의 연동을 지원해주는 지식베이스 처리 엔진을 개발하였다.

1. 서 론

[1]L-V-C 통합훈련체계는 실제 전장과 같은 가상 환경을 이용하여 다양한 전투 모의훈련을 실시하기 위해 L(Live), V(Virtual), C(Constructive) 체계를 통합한 훈련 체계를 의미한다.

L 체계는 실기동 모의훈련으로 가상의 공간이 아닌 현실세계에서 병사, 군사장비 등을 실제로 운용하는 것을 의미한다. L체계에서는 실시간성이 중요하기 때문에 미들웨어로 DDS(Data Distribution Service)를 사용한다. DDS는 OMG(Object Management Group) 표준 출판/구독 네트워크 커뮤니케이션 미들웨어이면서 실시간을 지원할 수 있는 특성을 가지고 있다[1]. V 체계는 가상 훈련을 의미하며 병사가 실제 환경에서 훈련하는 것이 아니라 가상환경에서 장비 등을 가지고 훈련하는 것을 의미한다. V체계에서는 시뮬레이션 미들웨어로 HLA(High-Level Architecture)를 사용한다. HLA는 분산 처리 컴퓨터 시뮬레이션 시스템 위한 범용 아키텍처를 의미한다[2]. 마지막으로 C 체계는 전투지휘훈련을 의미하며 가상의 환경에서 가상 장비, 가상 병력을 가지고 훈련하는 것을 의미한다. C 체계에서는 워 게임에서 운용되는 미들웨어인 DIS(Distribute Interactive Simulation)를 사용한다. DIS는 분산 네트워크상에서 실시간으로 데이터 메시지를 송수신 하면서 분산된 위치에서 워 게임을 수행하기 위한 IEEE 표준이다[3]. 최근 실제 전장과 동일한 환경을 구현하기 위해 L-V-C 통합 훈련 체계에 대한 연구가 진행되고 있다. 하지만 L-V-C 체계를 통합 할 때, 각 체계가 서로 다른 미들웨어를 사용하고 있어 미들웨어들 간의 연동 시 문제점이 있다. 각 체계에서 사용되고 있는 미들웨어인 DDS, HLA, DIS 는 서로 다른 데이터 형식인 TOPIC, FOM, PDU 를 사용한다. TOPIC은 DDS에서 발간/구독을 연결해주고, 데이터 자체와 관련된 이름, 데이터 유형 등을 연관시키는데 사용된다[1]. FOM(Federation object model)은 Federation에 대한 공유 객체, 속성, 상호작용 등을 설명해 주는 문서이다[2]. PDU(Protocol Data Unit)는 프로토콜 데이터 단위로 정의된 프로토콜에 따라 시뮬레이션 응용 프로그램간에 전달되는 DIS 데이터 메시지이다[3].

FOM과 PDU는 데이터 형식 표준이 정해져 있기 때문에 서로 간 데이터 매핑 시 어려움이 없으나, TOPIC은 데이터 형식 표준이 정해져 있지 않아 사용자가 임의로 정의한 TOPIC을 발간/구독 하기 때문에, 임의로 정의하여 발간한 TOPIC과 데이터 형식 표준이 정해져 있는 FOM, PDU와 매핑 시 문제점이 생긴다. 본 논문에서는 온톨로지와 지식베이스를 활용하여 이 문제를 해결한다. 온톨로지를 이용하여 FOM, PDU의 데이터를 분석하여

*이 성과는 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.NRF-2015R1A2A2A01004879)

† 교신저자

지식베이스를 구축해 놓으면, 사용자가 정의한 TOPIC을 지식베이스에 질의하여 매핑하는 과정을 통해 FOM, PDU와 연동이 가능하다. 또한 온톨로지를 활용하면 복잡도가 증가해도 오버헤드가 증가하지 않는다는 장점이 있다.

본 논문에서는 L-V-C에서 사용되는 이종 미들웨어 HLA, DDS, DIS 중 HLA와 DDS의 연동을 위한 지식베이스 처리 엔진을 개발하였다. 본 논문의 연구내용은 다음과 같다.

1) HLA FOM과 DDS TOPIC사이의 연동을 위해 표준이 정해져 있는 HLA FOM의 시맨틱 키워드를 분석, 확장한 후 온톨로지 모델링을 통해 지식베이스를 구축하였다.

2)가중치 기반 검색 알고리즘과 구축한 지식베이스를 사용하여 질의를 통해 HLA FOM과 DDS TOPIC을 연동한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 관련연구를 비교하고, 3장에서는 필요한 배경지식을 설명하며, 4장에서는 본 논문에서 개발한 지식베이스 처리 엔진을 설명한다. 마지막으로 5장에서는 결론을 맺는다.

2. 관련 연구

본 논문의 관련연구로는 [4], [5]가 있으며 [4]는 HLA와 DDS를 결합하여 하나의 미들웨어로 만들어주기 위한 연구이고 [5]는 HLA와 DDS를 모두 활용할 수 있는 시스템 구축을 위한 연구이다. 두 연구 모두 HLA와 DDS를 연동해 주기 위해 HLA 표준에 정의되어 있는 데이터와 DDS 명세에 나와 있는 데이터만을 매핑한다. 그렇기 때문에 사용자가 새로운 Topic을 추가할 때 매핑의 어려움이 생기며, 본 연구에서는 사용자가 새로운 Topic을 추가하여도 HLA FOM 데이터와 매핑을 지원한다.

3. 배경지식

3.1 온톨로지 모델

본 논문에서는 온톨로지 모델 구축을 위해 N-Triple형식의 RDF를 사용하였다. RDF는 W3C에서 제정한 표준 기술이며, 웹에서 정보를 표현하기 위한 범용 언어이다[6]. 또한 구축한 지식베이스의 질의를 위해 SPARQL(Simple Protocol and RDF Query Language)을 이용하였으며, SPARQL은 RDF에 저장된 데이터를 검색하고 조작할 수 있는 데이터베이스의 질의 언어이다. W3C에서 제정한 표준 기술로 시맨틱 웹의 주요 기술 중 하나로 인정받고 있다[7]. 본 논문에서는 RDF와 SPARQL을 이용하기 위해 Jena를 사용하였다. Jena는 Java기반 오픈소스 시맨틱 웹 프레임 워크이며, RDF, RDFS, OWL, SPARQL, Rule 기반 추론 엔진을 위한 프로그램 환경을 제공한다[8]. 마지막으로 본 논문에서는 구축한 모델을 지식베이스에 저장하기 위해 Jena TDB를 사용하였으며, Jena TDB는 RDF 형식의 Data를 저장하고 관리하기 위한 기능을 제공한다[9].

3.2 키워드 추출, 확장

본 논문에서는 HLA FOM의 시맨틱에서 명사인 키워드를 추출 하기 위해 OpenNLP를 사용하였다. OpenNLP는 자연 언어 텍스트 처리를 위한 기계학습 기반도구이다. 토큰화, 문장 분할, 품사 태깅, 엔티티 추출, 파싱 등 가장 일반적인 NLP작업을 지원하며, 보다 고급적인 텍스트 처리 서비스를 지원한다[10]. 또한 추출된 키워드를 확장하기 위해 WordNet을 사용하였다. WordNet은 약 15만개의 명사, 동사, 형용사, 부사 등의 단어가 11만 5천개의 동의어 집합으로 저장되어있는 데이터베이스를 의미한다[11].

4. 지식베이스 처리 엔진

4.1 지식베이스 구성도

그림 1은 LVC 지식베이스 처리 엔진의 전체 구성도이다. FOM과 사용자가 임의로 정의한 TOPIC을 매핑 해주기 위하여 지식 베이스 처리 엔진에서는 총 두 가지 과정을 처리한다. 하나는 지식베이스 구축 과정으로 키워드 추출 과정을 통해 FOM의 키워드를 추출하고, 키워드 확장 과정을 통해 추출된 키워드를 확장한다. 확장된 키워드를 온톨로지 모델링 과정을 통해 N-Triple 형식으로 변환 후 지식베이스(Triple Store)에 저장한다.
다른 하나는 키워드 검색 과정으로 사용자가 입력한 키워드를 키워드 확장 과정을 통해 확장한 후, 키워드 매핑 과정을 거친다. 키워드 매핑 과정에서는 확장된 키워드를 가중치 검색 알고리즘을 사용하여 지식베이스에 질의한 후, 매핑이 이루어진 오브젝트클래스의 목록을 반환한다.

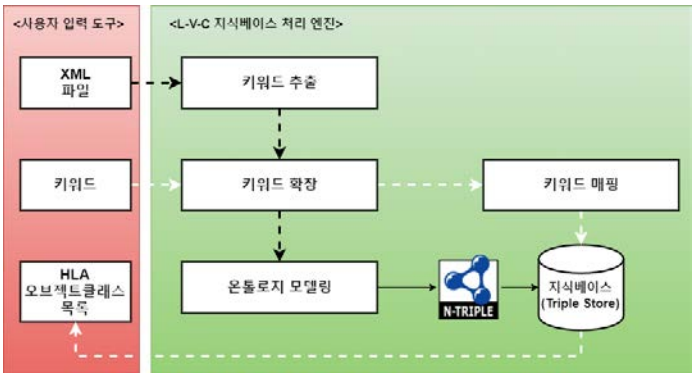


그림 1. LVC 지식베이스 처리 엔진 전체 구성도

4.2 지식베이스 구축

기존의 HLA와 DDS의 매핑 방식은 사전에 정의된 것들

로만 매핑이 가능했지만 사용자가 임의로 정의한 TOPIC과 HLA를 매핑하기 위해서는 지식베이스 구축이 필요하다. 지식베이스 구축 시 동의어를 기반으로 구축하여 사용자의 편의를 지원하였으며, 지식베이스 구축은 키워드 추출, 키워드 확장, 온톨로지 모델링으로 총 세 단계로 나누어진다.

4.2.1 키워드 추출

키워드 추출 과정에서는 먼저 HLA FOM(xml) 파일을 사용자 입력을 통해 받는다. FOM 안에 있는 오브젝트클래스 이름과 오브젝트클래스를 설명해주는 시맨틱 문장을 파싱 한 후 OpenNLP를 사용하여 오브젝트클래스 이름과 시맨틱 문장에서 명사를 추출해 낸다.

4.2.2 키워드 확장

동의어 기반의 키워드 확장을 위해 추출한 명사를 WordNet에서 구축한 별도의 사전에 검색한다. 검색 후 나온 명사에 대한 유의어 결과를 가지고 데이터를 확장한다.

4.2.3 온톨로지 모델링

[그림 2] 는 온톨로지 모델을 그림으로 나타낸 것이다. 오브젝트클래스는 서브클래스로 키워드를 가지고 있고, 키워드는 서브클래스로 WordNet을 통해 확장된 키워드를 가지고 있다. 또한 키워드와 확장된 키워드는 우선순위를 나타내는 가중치를 가지고 있다.

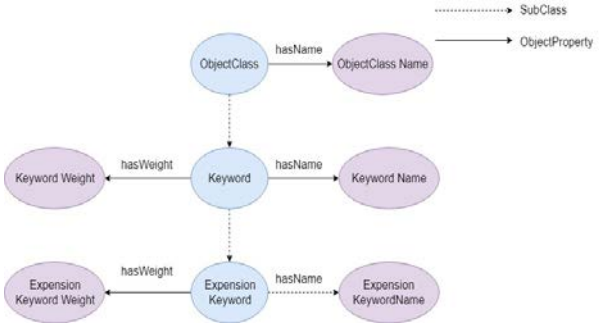


그림 2. 지식베이스 온톨로지 모델

온톨로지 모델링을 통해 만들어진 데이터는 N-Triple형태로 저장되고, 만들어진 N-Triple을 Jena TDB를 이용하여 데이터베이스에 저장한다.

4.3 키워드 검색

키워드 검색은 앞서 구축된 지식베이스를 기반으로 온톨로지 질의에 필요한 키워드(예 : craft)를 전달받아 SPARQL 질의를 수행한다. 질의를 수행 할 때 사용자가 찾고자 하는 오브젝트클래스 이름을 정확히 알지 못하더라도, 지식베이스는 동의어 확장을 통해 구축되어있기 때문에, 정확한 키워드를 입력하지 않아도 연관된 오브젝트클래스 이름을 검색 할 수 있다. 또한 키워드만으로 결과가 나오지 않을 때에는, 입력한 키워드를 다시 동의어 확장하여 구축된 지식베이스에 질의하여 결과를 도출한다. 이 과정을 위해 가중치 알고리즘을 새롭게 구현하여 사용하였다.

4.3.1 가중치 검색 알고리즘

그림 2를 보면 키워드와 확장된 키워드는 속성으로 가중치를 가지고 있다. 가중치를 부여하는 방법은 아래와 같다.

1) 오브젝트클래스 이름에서 추출된 키워드는 가중치 1을 부여하고, 시맨틱 문장에서 추출된 키워드는 가중치 2를 부여한다.
2) 위의 과정에서 추출된 키워드를 다시 WordNet을 통해 확장된 단어 순서대로 1부터 1씩 더하여 가중치를 구성한다. WordNet은 연관성이 높은 순서대로 결과가 출력된다.

이 가중치를 사용하여 오브젝트클래스를 트리 형태로 구성하였다. 그림 3은 트리를 구성한 예이다.

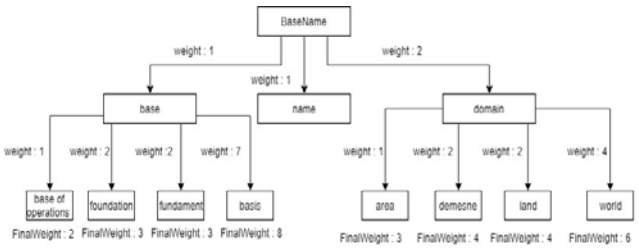


그림3. BaseName 트리 예

이 트리는 오브젝트클래스의 수만큼 존재하며, 구성된 트리를 이용하여 아래와 같이 3가지의 경우로 나누어 순차적으로 검색한다.

1)키워드(지식베이스)와 키워드(사용자 검색) 매핑
지식베이스의 키워드와 사용자가 입력한 키워드가 매핑 되는 경우이다.

2) 확장된 키워드(지식베이스) 와 키워드(사용자 검색) 매핑
지식베이스의 확장된 키워드와 사용자가 입력한 키워드가 매핑 되는 경우이다.

3) 확장된 키워드(지식베이스)와 확장된 키워드(사용자 검색) 매핑
지식베이스의 확장된 키워드 와 사용자가 입력한 확장된 키워드가 매핑 되는 경우이다.

각각의 단계에서 결과가 도출 되면 다음 단계로 넘어가지 않고 종료한다. 만약 각각의 검색 단계에서 사용자가

입력한 키워드가 여러 개의 오브젝트클래스 트리에 저장되어있는 키워드와 일치한다면, 가중치를 비교하여 가장 낮은 가중치의 키워드가 포함되어 있는 트리의 키워드부터 오름차순으로 출력된다.

5. 결 론

본 연구에서는 지식베이스 처리 엔진을 개발하기 위하여 필요한 관련 기술들을 분석하였다. LVC 지식베이스 처리 엔진의 키워드 추출 및 확장 기술을 위해 OpenNLP, WordNet을 분석하였다. 또한 온톨로지 표현 언어인 RDF, N-triple, Jena 및 SPARQL을 분석하였고, 지식베이스 저장을 위해 Jena TDB에 대해 분석하였다. 이러한 관련 기술 분석을 바탕으로 이종 미들웨어간 상호 연동을 지원해주는 지식베이스 처리 엔진을 개발하였다. 본 연구를 통해 온톨로지 기반의 지식베이스를 사용하면 보다 유연한 연동 및 확장이 가능하다는 것을 확인하였고, 지식 베이스 처리 엔진의 개발을 통해 L-V-C 체계에 사전 지식이 없더라도 키워드 기반의 검색을 통해 데이터 연동이 가능하기 때문에 사용자들의 진입장벽 또한 낮출 수 있을 것이며, 다양한 분야에서 지식베이스가 활용 될 수 있을 것으로 기대된다.

참 고 문 헌

[1] Pardo-Castellote, Gerardo. "Omg data-distribution service: Architectural overview." Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on. IEEE, 2003

[2] High-level architecture (simulation), http://en.wikipedia.org/wiki/High_level_architecture_(simulation).

[3] IEEE Standard for Distributed Interactive Simulation—Communication Services and Profiles

[4] Park, Yunjung, and Dugki Min. "Development of HLA-DDS wrapper API for network-controllable distributed simulation." Application of Information and Communication Technologies (AICT), 2013 7th International Conference on. IEEE, 2013.

[5] Joshi, Rajive, and Gerardo-Pardo Castellote. "A comparison and mapping of data distribution service and high-level architecture." Technology, The Netherlands. His research interests include parallel and distributed computing, component based architectures, and embedded systems (2006).

[6] Lassila, Ora, and Ralph R. Swick. "Resource descryyptti on framework (RDF) model and syntax specification." (1999).

[7] Prud, Eric, and Andy Seaborne. "SPARQL query language for RDF." (2006).

[8] Jena, Apache. "semantic web framework for Java." (2007).

[9] Jena, Apache(2004). "JenaTDB" jena.apache.org/documentation/tdb/index.html (2017.12.20)

[10] OpenNLP, Apache. "Apache software foundation." URL http://opennlp. apache. org (2011).

[11] Fellbaum, Christiane. WordNet. John Wiley & Sons, Inc., 1998.

프로세스 간 공유 메모리 데이터 기록 및 재현

윤기현 ○, 유민수
한양대학교 컴퓨터소프트웨어학과
khyun@rtcc.hanyang.ac.kr, msryu@hanyang.ac.kr

Record and replay for interprocess shared memory data

Kihyun Yun○, Minsoo Ryu
Department of Computer Software, Hanyang University

요 약

본 논문에서는 프로세스 간 통신 방법 중 공유 메모리를 사용하는 프로그램을 기록 및 재현(Record-Replay)하는 방법을 제안한다. 그 방법으로는 공유 메모리에 접근하여 사용되는 모든 공유 데이터를 순차적으로 기록하고, 재현한다. 그러기 위해 각 프로세스 메모리에서 물리 메모리에 할당된 공유 메모리 프레임을 가리키는 가상 주소의 페이지 테이블 엔트리의 접근 권한(Access Permission)을 접근 제한으로 설정한다. 이 상황에서 해당 가상 주소에 접근을 하면 데이터 접근 예외처리(Data abort exception)가 의도적으로 발생하게 되고, 데이터 접근 예외처리 핸들러(Data abort exception handler)에서 공유 메모리 데이터를 기록 및 재현을 한다. 프로그램을 실행할 때는 기록을 위한 모드나 재현을 위한 모드로 실행한다. 기록을 위한 모드일 때는 공유 데이터를 로그에 기록하고, 재현을 위한 모드일 때는 로그에 기록된 공유 데이터를 메모리에 대체한다. 이렇게 공유 메모리를 사용하는 프로그램의 동작 상태나 흐름을 기록하고, 완벽하게 재현해 줄 수 있다.

1. 서 론

기록 및 재현 방법에서 결정적이게 프로그램 동작을 재현하려면 프로그램의 동작 상태나 흐름을 변경할 수 있는 비결정적인 사건을 기록해야 한다[2,3]. 공유 메모리를 사용하는 프로그램에서 사용자에 의해 공유 메모리의 데이터로 입력되는 입력 값이나 프로그램 구동 중 경우에 따라 달라지는 공유 데이터는 프로그램의 동작을 비결정적이게 바꾼다. 이러한 비결정적인 사건이 발생하는 공유 메모리 사용 프로그램을 기록 및 재현하기 위해서 공유 메모리에 접근하여 읽기와 쓰기 동작에 관여한 모든 데이터를 순차적으로 로그에 기록할 수 있으면 프로그램의 동작 상태나 흐름을 완벽하게 재현해 줄 수 있다.

유저 프로그램에서 공유 메모리의 생성과 할당하는 시스템 콜을 사용하면 커널은 할당되지 않은 영역을 찾아 공유 메모리를 할당한다. 그 때 각 프로세스 메모리에서 물리 메모리에 할당된 공유 메모리 프레임을 가리키는 가상 주소의 페이지 테이블 엔트리의 접근 권한을 접근 제한으로 설정한다. 그러면 해당 공유 메모리 가상 주소 영역에 접근할 때마다 의도적으로 데이터 접근 예외처리를 발생시켜 데이터 접근 예외처리 핸들러에서 프로세스 간 공유하려는 데이터를 모드에 따라 기록 및 재현을 한다. 기록 모드일 때는 순차적으로 공유 데이터를 로그에 기록하고, 재현 모드일 때는 로그에 기록된 공유 데이터를 메모리에 대체하여 데이터를 재현해 줌으로써 기록 모드일 때와 동일하게 프로그램을 동작시킨다.

2. 관련 연구

2.1 페이지 테이블 엔트리의 접근 권한 필드 설정

그림 1과 같이 물리 메모리에 할당된 공유 메모리 프레임을 가리키는 가상 메모리의 영역은 각 프로세스마다 다르다. 해당 가상 주소 페이지 테이블 엔트리의 접근 권한 필드의 값을 조정하여 모든 읽기와 쓰기 권한을 제한한다[1]. 그럼으로써 해당 가상 주소에 접근하면 데이터 접근 예외처리가 의도적으로 발생한다.

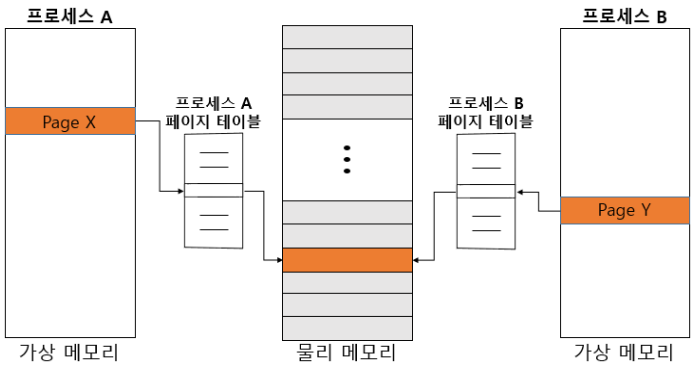


그림 1. 각 프로세스의 공유 메모리 할당

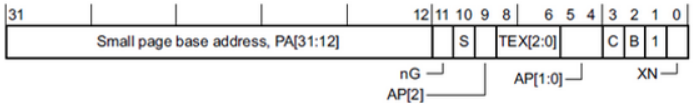


그림 2. 페이지 테이블 엔트리의 포맷

2.2 데이터 접근 예외처리 시점과 처리 핸들러

그림 3과 같이 인스트럭션들의 흐름에서 접근 권한이 접근 제한으로 되어 있는 공유 메모리 가상 주소에 접근하게 된다. 그러면 데이터 접근 예외처리가 의도적으로 발생하여 데이터 접근 예외처리 핸들러로 들어가게 되고, 데이터 접근 예외처리 핸들러 내에서 기록 및 재현을 한다. 공유 데이터를 기록 및 재현하려면 공유 데이터에 접근해야 하기 때문에 해당 가상 주소의 페이지 테이블 엔트리의 접근 권한을 읽기와 쓰기 모두 가능하도록 설정한다. 그 후에 기록 모드인지 재현 모드인지 확인하여 기록 모드라면 해당 메모리의 데이터를 로그에 기록하고, 재현 모드라면 로그에 기록된 공유 데이터를 해당 메모리에 대체한다.

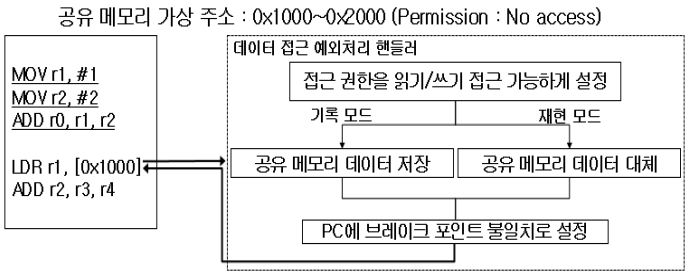


그림 3. 데이터 접근 예외처리 시점과 처리 핸들러

2.3 브레이크 포인트를 이용한 접근 권한 재설정

그림 3에서 접근 권한이 접근 제한으로 되어있는 공유 메모리 가상 주소에 접근하면 데이터 접근 예외처리가 발생되고, 데이터 접근 예외처리 핸들러에서 접근 권한을 읽기와 쓰기 모두 접근 가능하게 접근 권한을 설정하여 기록 및 재현이 수행된다. 그 후에 다음 공유 메모리 데이터를 기록하거나 재현하려면 다시 공유 메모리의 접근 권한을 접근 제한으로 설정해야 한다. 그러기 위해 프로그램 카운터(Program Counter)에 불일치(Mismatch)로 브레이크 포인트(Breakpoint)를 설정한다[3]. 그러면 그림 4와 같이 다음 인스트럭션이 수행되면 브레이크 포인트가 불일치로 걸려 있는 프로그램 카운터에 의해 브레이크 포인트 예외처리가 발생되어 브레이크 포인트 불일치 핸들러로 들어가게 된다. 브레이크 포인트 불일치 핸들러에서 공유 메모리 가상 주소의 페이지 테이블 엔트리 접근 권한을 접근 제한으로 바꿔준다.

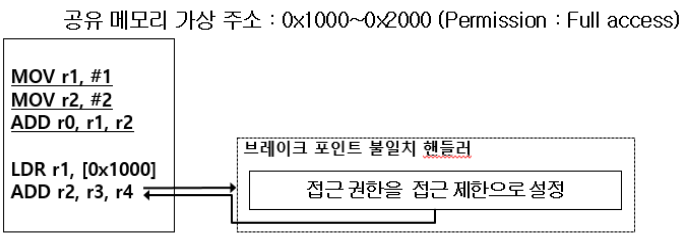


그림 4. 브레이크 포인트 예외처리와 처리 핸들러

3. 구현

데이터 접근 예외처리가 발생하면 발생 원인과 발생된 명령어 주소 정보가 DFAR(Data Fault Address Register)과 DFSR(Data Fault Status Register)에 저장된다. DFAR은 데이터 접근 예외처리를 발생시킨 명령어의 가상주소를 가지고, DFSR은 마지막으로 발생한 데이터 접근 예외처리에 대한 상태 정보를 가진다. 그림 5와 같이 DFAR이 페이지 테이블 접근 제한으로 설정된 영역인지 확인하고, DFSR은 접근 제한으로 인해 데이터 접근 예외처리가 발생했는지 확인한다. 해당 공유 메모리의 접근 제한 설정으로 인해 의도된 데이터 접근 예외처리 발생으로 확인되면, DFAR이 포함된 페이지 테이블에 대해 읽기와 쓰기가 가능하도록 접근 권한을 변경한다. 기록 모드로 실행되었다면 공유 데이터를 로그에 기록한다. 재현 모드일 때는 기록했던 데이터를 실제 메모리에 대체하는 작업을 하여 기록 모드에 동작되었던 프로그램처럼 똑같이 재현한다. 그 후 현재 프로그램 카운터에 브레이크 포인터를 불일치로 설정한다. 그러면 불일치로 설정된 브레이크 포인트 때문에 다음 인스트럭션이 실행되면 브레이크 포인트 불일치 예외처리가 발생되고, 브레이크 포인트 불일치 핸들러에서 다시 공유 메모리를 가리키는 가상 주소의 페이지 테이블 엔트리 접근 권한을 접근 제한으로 바꿔준다.

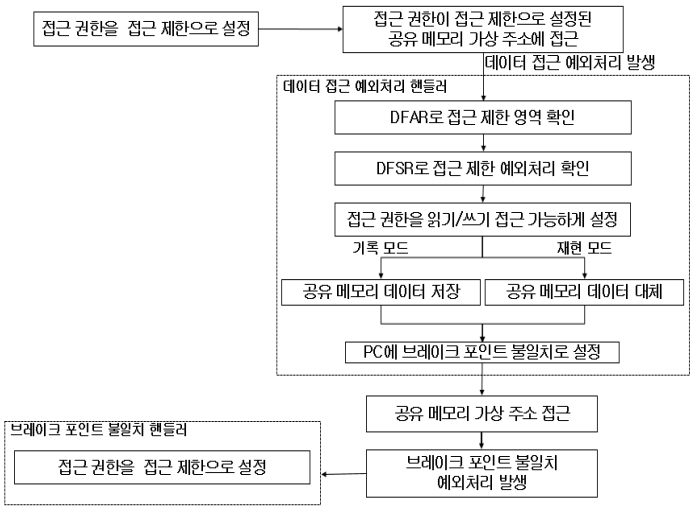


그림 5. 공유 메모리 데이터 기록 및 재현 흐름도

4. 결론

프로세스 간 통신 중 공유 메모리를 사용하는 프로그램에서 비결정적 사건의 주체인 공유 데이터를 순차적으로 기록하고, 재현할 수 있어 프로그램을 결정적이게 동작할 수 있다. 이를 통해 상용 프로그램에서 비결정적 요소로 인해 발생되는 문제점을 효과적으로 찾고 해결할 수 있다. 향후 연구로는 다른 방법으로 공유 메모리를 생성하고 할당하는 프로그램에도 기록 및 재현을 적용해야 하며, 공유 메모리의 데이터 레이스까지 재현하는 연구가 요구된다.

Acknowledgment

이 논문은 2017년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원(No. R0114-16-0046, (SW-창조씨앗2단계) 고신뢰 컴퓨팅을 위한 소프트웨어 블랙박스 기술 개발)과 과학기술정보통신부 및 정보통신기술진흥센터의 SW중심대학 지원사업의 연구결과로 수행되었음(2016-0-00023)

참 고 문 헌

[1] ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition

[2] H.J.Shin, S.Y.Jung, and M.S.Ryu, “User Application Record-Replay Mechanism via Detecting Kernel-to-User Data Transfer”, Foundations of Computer Science (FCS), pp 126-128, Las Vegas, USA, July 2016.

[3] 박성현, 윤호상, 유민수, “운영체제 시그널 동작의 정확한 기록과 재현 방법”, 한국통신학회 하계종합학술발표회, Vo1.63, 15E-23, pp.1292-1293, 2017.

역공학 기반 코드 가시화를 통한 기능점수의 비용 추정 검증 방안

문소영 O, 김영철
홍익대학교 소프트웨어공학 연구실
msy@selab.hongik.ac.kr O, bob@hongik.ac.kr

Validating the Cost Estimations of Function point through Code Visualization Mechanism based on Reverse engineering

So Young Moon O, R. Young Chul Kim
SELab., Hongik University

요 약

소프트웨어 개발 프로젝트는 비가시성이라는 소프트웨어의 특성으로 인해 규모 추정에 어려움이 있다. 또한 개발되면서 요구사항도 수시로 변경 및 추가되어 처음의 규모와 다르게 프로젝트가 진행되어 규모에 따른 비용의 차이가 발생한다. 본 논문에서는 위와 같은 문제를 해결하기 위해, 역공학 기반 소스코드 가시화를 통해 IFPUG에서 권고하는 기능점수 산정 방식으로 기능점수를 검증하였다. 그 결과 소프트웨어 개발 결과물 중, 최신이며 최종인 소스코드를 통해 기능점수를 검증할 수 있었다. 본 연구를 통해 프로젝트 발주 시 규모와 완료 시 규모에 대한 검증이 가능하고, 비가시성이라는 특성을 극복할 것으로 기대한다.

1. 서 론

소프트웨어가 대부분의 영역에서 사용되고, 소프트웨어 개발 비용이 차지하는 비중이 높아지면서 소프트웨어 개발을 위한 비용 추정과 효과적 비용 관리가 중요한 이슈이다. 소프트웨어공학은 빈번히 발생하는 요구사항 속에서 추정된 기간, 추정된 비용 안에서 고품질 소프트웨어를 개발할 수 있도록 돕는 기술, 방법론, 도구 등을 사용하여 소프트웨어를 개발하도록 돕는 학문이다. 관리자 측면에서 소프트웨어 규모에 따라 기간, 비용, 인력을 추정해야 하는 것은 성공적 프로젝트 수행과 고품질 소프트웨어 개발을 위해 중요하다. 그러나 소프트웨어 개발 비용은 막연한 요구사항, 과거 소프트웨어 개발 데이터 부족과 같은 이유로 비용 추정에 어려움이 있다[1].

본 논문에서는 소프트웨어 개발에서 산출되는 가장 최신의 실제 운영되는 소스코드를 중심으로 기능점수를 검증한다. 역공학 기법을 적용하여 소스코드를 가시화하고, 기능점수 계산에 필요한 기능유형들을 찾고, 참조파일유형 개수, 데이터요소유형 개수를 찾아 기능 복잡도를 측정하여 전체 기능점수를 검증한다.

2. 역공학 기반 기능점수 검증 방법

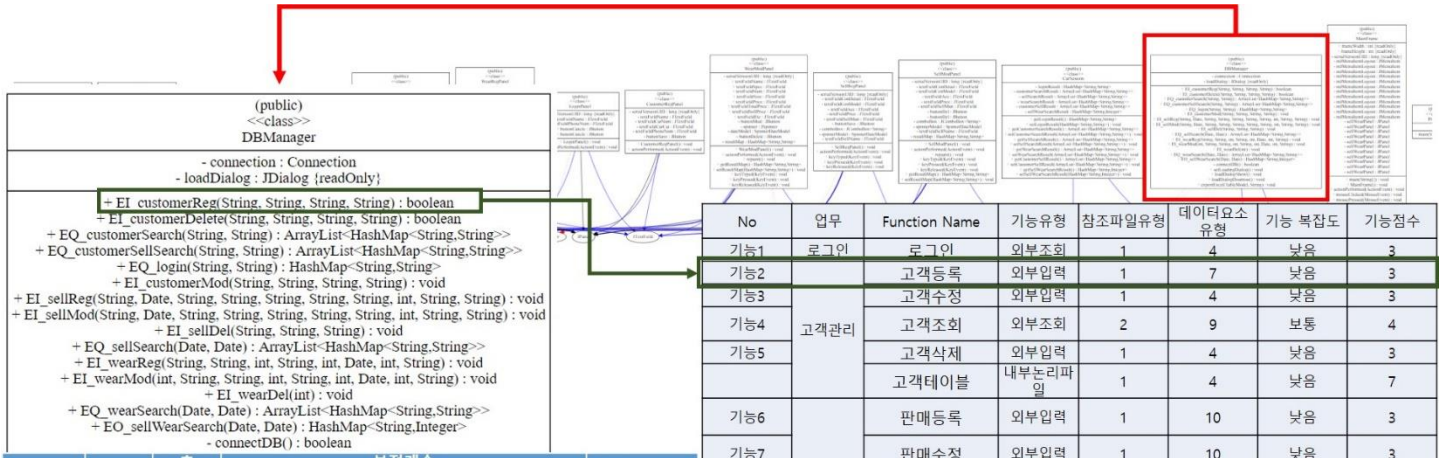
본 논문에서는 IFPUG에서 권고하는 기능점수 측정 절차 중에서 소스코드 가시화를 통해 데이터 기능, 트랜잭션 기능 유형을 구분하여 각각의 기능 복잡도를 계산하는 것을 시도하였다.

데이터 기능, 트랜잭션 기능을 소스코드에서 찾아내기 위해 ①고객의 요구사항에 대한 기능 정의와 분류가 되어 있어야 하고, ②코딩 규칙을 적용하여 각 기능이 적용된 소스코드에 “트랜잭션기능명_기능명”으로 메서드를 명명한다.(예: EQ_login) ③기존 연구에서 제안한 소스코드 가시화를 위한 툴체인[2]을 사용하여 소스코드 가시화를 수행한다. ④ 적용된 코딩 규칙을 통해 기능을 찾고 그 기능이 외부입력, 외부조회, 외부출력 중 어떤 트랜잭션 기능에 해당하는지 판별 후, 사용하는 쿼리문을 기본으로 내부논리파일, 외부연계파일을 찾는다. 그 다음 쿼리문에서 필드의 개수를 구하여 복잡도를 계산한다. ⑤추출한 데이터 기능, 트랜잭션 기능에 대해 기능 복잡도를 계산하여 전체 기능점수를 측정한다.

제안 방법을 통해 자바 언어로 개발된 GUI 환경의 S썬팅샵 판매관리 프로그램의 기능점수를 검사하였다. 이 프로그램은 오라클 데이터베이스와 연동하여 작동하는 어플리케이션이다. 툴체인을 통해 S썬팅샵 판매관리 프로그램을 분석하여 클래스 다이어그램으로 가시화한 내용이 그림 1이고, 그림 1에서 소스코드를 통해 기능점수를 계산하여 검증한 내용을 확인할 수

* 본 논문은 2015년 교육부와 한국연구재단의 지역혁신창의인력양성사업(NRF2015H1C1A1035548)과 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2017R1D1A3B03035421).

있다. 하나의 경우를 예를 들어 설명하면, 기능 수는 17개로 고객관리 업무에 고객등록 기능이 있는데, 외부입력에 해당되고 소스코드는 DBManager 클래스에 EI_customerRegI 메서드로 구현되어 있다. 메서드 내부의 소스코드를 분석한 결과 customer 고객 테이블을 내부논리파일로 참조하고, 이름, 차량번호, 차량종류, 전화번호, 기능키1, 메시지2개의 데이터요소 유형 7개를 갖고 있다는 것을 확인할 수 있다. 추출된 값으로 IFPUG 기능점수 계산 방법을 적용하여 본 논문에서 개발한 가시화 툴체인을 통해 기능복잡도는 낮음, 기능점수는 3으로 계산되었다. 다른 기능들에 대해서도 같은 방법으로 측정한 결과 적용 프로그램의 데이터 기능, 트랜잭션 기능에 대한 총 기능점수는 77점으로 계산되었다. 그림 1의 하단 왼쪽 표는 S썬팅샵 프로그램에 대한 보정 후 개발원가 산정을 나타낸다. 본 논문에서는 2017년 개정된 소프트웨어 사업 대가산정 기준을 적용하였다. 소프트웨어 개발 생명주기의 분석, 설계, 구현, 시험 단계에 대한 각각의 단가를 적용하여 계산한다. 본 논문에서 제안한 역공학 기반 코드 가시화를 통해 개발원가 31,183,330원이 추정되었다.



단계	단계별 단가	총 기능점수	보정계수				개발원가
			언어	어플리케이션유형	규모	품질 및 특성	
분석	98,648	77	1.2000	1.00	0.6500	1.0000	5,924,798
설계	124,609						7,484,016
구현	166,145						9,978,668
시험	129,801						7,795,848
합계	519,203						31,183,330

보정계수 결정	구분	적용기준	보정계수
	규모 보정계수	0.108 * log e(기능점수) + 0.2229	0.65
	유형 보정계수	업무처리용 소프트웨어	1.00
	언어 보정계수	JAVA	1.20
	품질 및 특성 보정계수	0.025 * 총영향도(0) + 1	1.00

No	업무	Function Name	기능유형	참조파일유형	데이터요소 유형	기능 복잡도	기능점수
기능1	로그인	로그인	외부조회	1	4	낮음	3
기능2	고객관리	고객등록	외부입력	1	7	낮음	3
기능3		고객수정	외부입력	1	4	낮음	3
기능4		고객조회	외부조회	2	9	보통	4
기능5		고객삭제	외부입력	1	4	낮음	3
		고객테이블	내부논리파일	1	4	낮음	7
기능6	판매관리	판매등록	외부입력	1	10	낮음	3
기능7		판매수정	외부입력	1	10	낮음	3
기능8		판매조회	외부조회	1	12	낮음	3
기능9		판매삭제	외부입력	1	6	낮음	4
기능10		판매관리엑셀저장	외부출력	2	12	보통	4
		판매테이블	내부논리파일	1	8	낮음	7
기능11	입고관리	입고등록	외부입력	1	13	낮음	3
기능12		입고수정	외부입력	1	13	낮음	3
기능13		입고조회	외부조회	1	12	낮음	3
기능14		입고삭제	외부입력	1	11	낮음	3
기능15		입고관리엑셀저장	외부출력	1	11	낮음	3
		입고테이블	내부논리파일	1	9	낮음	7
기능16	수입/지출관리	수입지출조회	외부출력	2	10	보통	4
기능17		수입지출엑셀저장	외부출력	2	11	보통	4

그림 1 S썬팅샵 판매관리 프로그램 가시화 및 기능점수 검증

4. 결론 및 향후 연구

소프트웨어 개발 프로젝트에 대한 비용 산정 또는 규모 산정은 프로젝트를 성공적으로 수행할 수 있도록 하는데 중요한 요소가 된다. 규모 산정이 잘 되어야 조직 인력을 체계적으로 관리하도록 계획을 할 수 있고, 개발자들의 성과와 계획의 차질에 대한 원인분석이 가능하다.

본 논문에서는 프로젝트 규모 추정에 대한 검증을 시도하였다. 역공학 기반 소스코드 가시화를 통해 소스코드에서 기능 유형을 찾아내어 각 기능에 대한 기능점수를 측정하여 데이터 기능과 트랜잭션 기능에 대한 총 기능점수를 산출했다. 향후에는 본 논문에서 제안한 방법으로 실제 프로젝트에 적용하여 비교 및 검증을 해보고, 그 데이터들을 축적하여 조직의 프로젝트 성과와 프로세스 개선에 도움을 주고자 한다.

5. 참고문헌

[1]Vidger, M.R., A. W. Kark, Software cost estimation and control, Institute for Information Technology, National Research Council Canada, 1994.

[2]So Young Moon, R. Youngchul Kim, Chae Yun Seo, “A Survey for Software Visualization”, Journal of Platform Technology, Vol.4, No.3, pp.22-29, 2016.

요구사항에 따른 이슈 분류

심재경[o1], 정세린[2], 이선아[1, 2]
[1]항공우주 및 소프트웨어공학전공, 경상대학교
[2]정보과학과, 경상대학교
vxeimv@naver.com, {rin0608, saleese}@gnu.ac.kr

Issue Classification according to Requirements

Jaekyeong Sim[o1], Serin Jeong[2], Seonah Lee[1, 2]
[1]Department of Aerospace and Software Engineering, Gyeongsang National University
[2]Department of Informatics, Gyeongsang National University

요 약

과거 연구에서는 사용자 피드백을 체계적으로 반영하기 위하여 앱 스토어에 저장된 사용자 피드백을 정의된 범주로 자동 분류하는 기법을 제시하여 왔다. 본 연구에서는 오픈소스 개발체계에서 일반적으로 활용하는 이슈 추적 시스템에도 요구사항 관련 제안들이 있음에 주목하여, 해당 이슈들을 요구사항의 범주별로 분류하는 기법에 대해서 제안한다. 또한 하나의 오픈소스 프로젝트를 대상으로 기법을 적용하여 초기 실험 결과를 보고한다. 이러한 제안과 실험 결과는 향후 소프트웨어 요구사항의 진화에 있어 자동화를 통한 체계적인 분류와 반영에 도움을 줄 것으로 기대한다.

1. 서 론

개발자들은 소프트웨어의 개선을 위해 사용자의 리뷰를 활용한다. 그러나 사용자 리뷰가 많을 경우 이를 일일이 읽어서 검토하기 힘들다. 이를 위하여 과거 연구[1][2]에서는 앱 스토어에 있는 많은 양의 사용자 피드백을 식별하고자 분류 기법을 적용하였다. 먼저 Panichella과 동료들은 자연어 처리, 텍스트 분석, 감정 분석을 통해 앱 리뷰를 정의된 범주로 자동 분류하는 방법을 제안하였다 [1]. 또한 Guzman과 동료들은 개별 기계학습 접근법과 이들을 조합한 앙상블 방법을 사용하여 사용자의 리뷰를 소프트웨어 진화와 관련된 범주로 자동 분류하는 방법을 제안하였다 [2]. 그러나 [1][2]에서는 앱 스토어의 사용자의 리뷰를 범주로 분류하는 부분으로 그 대상과 기법을 제약한다.

반면 앱 스토어의 리뷰뿐만 아니라 Github나 Bitbucket 등과 같은 오픈소스 저장소에도 방대한 양의 사용자 이슈가 존재한다. 이러한 이슈들도 사용자의 요구 사항을 포함하고 있으며, 개발자에게 유용한 정보를 제공하여 사용자들의 만족도를 높이는데 기여할 수 있다. 그러나 많은 양의 이슈 중에서 유용하고 특정한 요구사항과 관련된 이슈를 추출하는 것은 어려운 작업이다. 이를 위해 본 논문에서는 사용자의 이슈를 요구사항에 따라 자동으로 분류하는 방법을 제안한다.

본 논문에서 제안하는 방법은 다음과 같다. 먼저, 사용자 이슈를 입력 데이터로 받아 들여서 전처리를 수행 후, 트레이닝 모델을 생성한다. 이 때 트레이닝 모델은 항목별 단어의 빈도수이다. 다음 트레이닝 과정에서 만들어진 트레이닝 모델을 이용하여 Naïve Bayesian 분류법을 사용하여 사용자의 이슈를 분류하는 작업을 수행한다. 또한 제안하는 방법을 검증하기 위하여 하나의 오픈소스 프로젝트의 이슈들을 추출하여 분류의 정확도를 측정하였다. 초기의 정확도는 73%이며 요구사항 항목마다 정확도는 달랐다. 따라서 정확도를 높이고 요구항목의 정확도를 일정하게 유지하는 분류 기법 개발이 필요함을 알 수 있었다.

본 논문의 구성은 다음과 같다. 먼저 2절에서는 제안 방법을 설명한다. 다음 3절에서는 제안 방법을 검증할 실험 계획을 설명한다. 4절에서는 실험의 결과를 논의한다. 5절에서는 결론과 향후 연구에 대해 논의한다.

2. 제안 방법

제안 방법은 Naïve Bayesian 방법을 채택하였다. 이를 기반으로 우리의 제안 방법을 설명한다.

2.1 Naïve Bayesian 방법

Naïve Bayesian 분류법은 단어의 등장 확률을 이용하여 문서를 분류하는 기법이다. 문서가 특정 항목에 속할 확률을 계산하기 위해 다음 bayes' rule을 이용한다.

$$P(c_i|d) = \frac{P(d|c_i)P(c_i)}{P(d)}$$

$$P(c_i|d) \propto P(d|c_i)P(c_i)$$

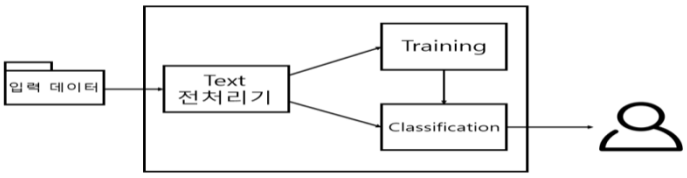
해당 식에서 d는 분류할 입력 문서, c_i는 분류 항목 중 하나이다. 문서 d가 단어 w_1, w_2, ..., w_i로 구성되어 있고 각 단어가 독립임을 가정하면 식은 다음과 같다.

$$P(c_i|d) = P(c_i|w_1, w_2, \dots, w_i) \propto P(w_1, w_2, \dots, w_i|c_i)P(c_i)$$

$$P(w_1, w_2, \ \dots, w_i|c_i) = P(w_1|c_i) \cdot P(w_2|c_i) \cdot \ \dots \ \cdot P(w_i|c_i)$$

Naïve bayesian 분류법은 이 식을 이용한다. 각 단어 별 확률을 구한 후 이를 통해 사용자의 이슈를 분류할 수 있는 모델을 생성한다.

2.2 제안 방법



[그림 1] 제안 방법의 구조

우리 제안 방법의 작업은 크게 3가지로 나눌 수 있다. Text 전처리기와 Training, Classification 이다. 각 작업의 설명은 다음과 같다.

1) **Text 전처리기:** Text 전처리기는 분류의 정확도를 높이기 위해 입력된 텍스트 데이터를 가공한다. 이 작업에서는 형용사, 명사, 동사, 부사의 단어만 남긴 후 변형된 형태의 단어는 원형으로 만들어 다른 형태의 동일한 단어가 다른 단어로 처리되지 않도록 한다. 이 기능은 nltk 라이브러리 [1] 를 이용하였다. 품사를 판별하는 데에 해당 라이브러리의 tag 기능을 사용하였고, 단어의 형태를 원형으로 바꾸는데에 해당 라이브러리의 Lemmatize 기능을 사용하였다.

2) **Training:** Training은 Naïve Bayesian 분류법을 사용하기 위해 항목별 단어의 빈도수를 세는 작업이다. Training 기능에서는 2가지를 저장한다. 하나는 사용되는 단어 목록이고 다른 하나는 단어의 빈도수이다. Training 동작 중 단어 목록에 없는 단어가 나오면 단어 목록에 추가한다.

3) **Classification:** Classification은 Naïve Bayesian 분류법에 의한 확률로 이슈를 분류하는 작업이다. Training 기능으로 만들어진 단어의 빈도수를 이용하여 입력된 이슈를 naive Bayesian 분류법으로 확률을 구한다. 그 중 가장 확률 값이 큰 항목으로 분류한다. 이는 scikit-learn 라이브러리를 이용하였다.

3. 실험 계획

실험에 있어서는 이슈를 요구사항 별로 분류할 수 있는지에 대해 앞에서 제시한 기법을 적용하여 보았다.

1) **실험 대상:** 본 논문에서 제안하는 분류법의 성능을 측정하기 위해 Github에 있는 오픈소스 프로젝트인 ATOM 프로젝트를 대상으로 실험하였다. ATOM은 Github에서 만든 에디터로 UI 편집기이다. 실험대상으로 ATOM을 선정한 이유는 다음과 같다. 첫번째, Github에 업로드되어 있기 때문에 사용자들의 이슈를 수집할 수 있다. 두번째, GUI 에디터이기 때문에 사용자들의 요구사항을 명확히 알기 쉽다.

2) **실험 방법:** Github에서 이슈의 레이블은 사용자가 지정한다. 이러한 점에서 레이블은 요구사항과 관련 있으므로 레이블을 분류 항목으로 정한다. 실험에는 레이블 중 ui, typography, proposal, network, auto-indent, enhancement을 선택하였다. 이러한 레이블이 붙은 이슈를 총 100개를 수집하였다. 그 중 63개는 Training에 사용하였고, 37개는 분류를 하였다.

3) **척도:** 분류 결과는 선택한 6가지 레이블 중 하나이다. 분류한 이슈 중 실제 이슈에 붙은 레이블과 분류 결과가 일치한 이슈 개수로 정확도를 측정한다.

$$\text{정확도} = \frac{\textit{일치한 개수}}{\textit{분류한 이슈 개수}}$$

4. 실험 결과

사용된 데이터 개수와 일치 개수, 정확도는 다음과 같다.

표 1 실험한 항목별 이슈 개수와 분류가 일치한 개수

항목	Training + Test	Training	Test	일치 개수	정확도
ui	20	12	8	3	0.38
typography	13	9	4	2	0.50
proposal	13	9	4	4	1.00
network	21	12	9	8	0.89
auto-indent	12	8	4	3	0.75
enhancement	21	13	8	7	0.88
Total	100	63	37	27	0.73

적은 데이터 수인 100개의 이슈에 대해 실험한 결과의 정확도가 73%였다. 이는 더 많은 이슈를 사용하면 더 높은 정확도를 기대할 수 있음을 의미한다. 하지만 단어의 빈도수만 이용하는 Naïve Bayesiann 분류법에는 한계 있다. 이 분류법은 특정 항목에 주로 나타나는 단어의 종류와 개수에 영향을 받는다. 표 1에 따르면 ui 항목에 대한 정확도가 떨어짐을 확인할 수 있다. 이 항목 관련된 이슈는 이미지로 작성된 경우가 많아 특징적인 단어가 잘 나타나지 않는다. Proposal 항목의 결과는 데이터가 적음에도 모두 일치하는 것을 확인할 수 있다. 이 항목은 작성된 이슈 내용의 양이 평균적으로 많기 때문에 정확도가 높아졌다.

5. 결론

오픈소스 프로젝트에서 100개의 이슈를 선정하여 요구사항별로 자동 분류를 실험하였다. 그 결과 73%의 정확도로 분류할 수 있음과 항목에 따라 정확도에 차이를 보임을 확인하였다. 본 논문에서 적은 양의 실험 데이터를 사용하였다. 향후에는 많은 데이터를 수집하여 정확도를 측정할 예정이다. 또한 좀 더 높은 정확도를 위한 추가적인 기법 개발을 하고자 한다.

6. 참고문헌

[1] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora and H. C. Gall, " "How can i improve my app? classifying user reviews for software maintenance and evolution." ICSME, IEEE, 2015

[2] E. Guzman, M. El-Haliby and B. Bruegge, "Ensemble methods for app review classification: An approach for software evolution (n)." ASE, IEEE, 2015

[1] Natural Language Toolkit, http://www.nltk.org/

코드 변경에 따른 테스트 케이스 갱신 분석

강 혜진 ○, 이 선아
정보과학과, 경상대학교
{master6280,saleese}@gnu.ac.kr

Analyzing Updates of Test Cases according to Code Changes

Hyejin Kang ○, Seonah Lee
Department of Informatics, Gyeongsang National University

요 약

단위테스트를 자동으로 생성하는 기술들이 다양하게 연구되고 있다. 이러한 기술들은 단위 테스트를 위한 테스트 케이스 생성 기능을 제공하여 사용자에게 편의성을 제공한다. 그러나 코드에 대한 테스트케이스 재생성에 초점을 두고 있다. 따라서 코드 변경 시 기존의 테스트케이스를 사용자가 변경했다면, 이러한 수작업 변경은 새로운 테스트 케이스 생성에 반영되지 않고 삭제되는 문제점을 가지고 있다. 본 논문에서는 단위테스트를 생성하지 않고 자동 갱신하는 방향으로 나아가기 위하여 코드 변경과 그에 따른 테스트 케이스 갱신을 조사하여 이들간의 연관성을 찾는다. 이러한 연관성을 바탕으로 진화적 관점에서 코드변경에 따른 테스트 케이스 자동 갱신에 도움을 줄 수 있는 자동화 연구 방향에 대해서 논의한다.

1. 서 론

요즈음 단위 테스트를 위한 자동화 기술을 다양하게 연구하고 개발하고 있다. 이러한 기술은 도구 또는 플러그인으로 개발되어 상용화하거나 오픈소스로 제공한다. 그 중 대표적으로 Junit을 이용하는 오픈소스 플러그인으로 Evosuite가 있으며 상용도구로는 슈어소프트사의 Controller Tester가 있다. 이러한 테스트 자동화 기술은 테스트 케이스를 자동으로 생성하여 준다. 하지만 이러한 생성 기술을 기존의 테스트 케이스에 사용자가 코드를 추가한 테스트 케이스를 새로운 버전에 반영하지 않고 무시한다는 문제점을 가지고 있다.

기존 연구들은 단위 테스트 자동 생성 기술과 효율적인 회귀 테스트 방법 연구로 구분할 수 있다. 먼저 단위 테스트 자동 생성의 대표적인 연구로는 Andrea Arcuri et al.가 제안한 Evosuite 플러그인이 있다. 해당 도구는 객체지향 클래스에 대한 단위 테스트를 자동으로 생성한다. 개발된 플러그인은 다양한 소프트웨어 개발 환경에 통합하여 사용할 수 있고, 이 때 생성된 Evousuite 아키텍처는 다른 테스트 데이터 생성 도구에도 재사용 할 수 있어 보다 효율적인 단위 테스트를 지원한다[2] 효율적인 회귀 테스트 방법 연구 중 최근 연구 중 히스토리를 기반으로 한 연구는 Edward et al.가 진행한 연구로 테스트 이력을 기반으로 한 효율적인 회귀 테스트 도구를 개발하였다 [3]. 이 논문에서는 선택적 회귀테스트에 대한 정확성, 정밀도, 리콜, 효율성에 대하여 산업 평가를 수행하였고 평균적으로 변경된 코드와 관련된 테스트데이터의 80%를 식별하는 높은 효율성 및 정확성과 시간 절약을 통한 비용을 절감할 수 있는 효과를 보고하였다[3]. 그러나 이러한 기존 기술들은 테스트 케이스 재생성에 초점을 둔 기술이라 사용자가 수작업으로 테스트 케이스를 수정했을 때 이를 감안하여 테스트 케이스를 자동 갱신하지 못하는 제약이 있다.

본 논문에서는 코드 변경에 따른 테스트 케이스 자동 갱신으로 나아가기 위하여 오픈소스 프로젝트 하나를 선정하여 커밋 이력을 분석하고 코드 변경에 따른 테스트 케이스 변경의 연관성을 조사하는 실증적 연구를 수행한다. 이 연구에서 코드 변경에 따라 테스트 갱신이 어떻게 이루어 지는지를 조사하고, 패턴화와 자동화가 가능한지를 타진한다. 이 실증적 연구를 통하여 얻은 결과는 향후 테스트 케이스 자동 갱신 연구에 적용하여 진화적인 관점에서의 단위테스트 연구에 공헌 하고자 한다.

본 논문의 구성은 다음과 같다. 2절에서는 관련된 연구의 문헌조사 내용을 정리한다. 3절에서는 본 연구의 실증적 조사 방법을 설명하고 4절에서는 3절에서 제안한 방법을 활용하여 나온 조사 결과를 설명한다. 5절에서는 본 논문의 결론을 내린다.

2. 관련연구

기존 연구들은 Hilton et al과 같이 오픈소스 프로젝트에서의 지속적인 통합환경(CI)의 도입에 대한 중요성을 강조하였다. 하지만 도입을 위한 자료의 부족으로 인하여 연구 커뮤니티의 관심이 부족하다는 문제점을 지적하였고, 이를 해결하기 위하여 CI환경의

도입 효과에 대한 성장성 및 향후 전망을 연구하는 것을 제안하였다. 이를 위하여 GitHub를 이용하여 오픈소스 프로젝트의 분석, CI시스템의 빌드수집, 사용자 설문조사를 통하여 CI의 사용실태를 분석하였다. 이 분석 결과 향후 CI는 오늘날 보다 더 큰 성장력을 가질 것으로 예측하였다. 이 연구는 GitHub에 업로드 된 방대한 자료를 수집,분석을 하여 CI의 향후 전망성을 밝혀냈다.[1] 이 논문과 같이 CI환경의 도입에 대한 전망을 밝히는 논문 외에 전통적인 지속적인 통합환경에서의 단위 테스트에서 발생하는 문제점을 분석하고 효과적으로 단위 테스트를 할 수 있는 방법을 제안하는 문헌도 많이 볼 수 있다.

먼저 Andrea Arcuri et al.은 객체지향 클래스에 대한 단위 테스트를 자동으로 생성하는 기술을 접목한 도구를 SW개발 활동에 도입하는 관점에 대하여 조사가 필요하다고 판단하여 Evosuite라는 플러그인을 개발하였다. 하지만 이 방법은 테스트 케이스를 자동으로 생성은 해주지만 재생성만 가능하여 기존의 테스트 케이스에서 사용자가 추가한 부분에 대하여 새로운 버전에 반영은 되지 않아 진화적인 부분에 대한 연구가 더 필요한 연구이다.[2]

다음으로 회귀시험에 관련한 연구도 많이 존재한다. Edward Dunn Ekelund et al.은 CI환경에서의 회귀테스트에 있어 기존의 피드백의 반영보다 더 빠르게 반영될 필요가 있다고 주장하였다. 이를 위하여 테스트 기록을 기반으로 하는 정확한 선택적 회귀테스트 도구인 Difference Engine을 구현하고 경험적 평가를 수행하였다. 이 도구는 기존의 테스트 기록을 이용하여 상관관계를 파악하고 자동으로 높은 상관관계를 가진 테스트를 선택하여 회귀 테스트를 수행한다.[3] 이와 같이 효율적인 회귀테스트를 위하여 Elbaum과 Dusica Marijan의 연구가 있다.

Elbaum의 연구는 코드 변경 및 신규 코드 생성 후 이루어지는 회귀테스트에서의 과다한 비용 문제 발생 및 대규모 프로젝트에서 빌드 전 과도한 리소스 사용으로 인한 병목 현상 발생 등의 기존의 CI 통합 개발 환경에서의 회귀테스트의 비용 및 리소스 사용의 문제가 발생하고, 변경이 매우 잦은 프로젝트에서는 테스트 데이터를 최신으로 유지하는 것이 불가능하다는 기존의 CI통합 개발환경에서의 회귀테스트는 잦은 변경에 취약한 문제가 있음을 발견하였다. 이를 해결하기 위하여 비용 효율적(Cost-effectiveness)으로 CI환경에서의 회귀테스트를 적용하기 위한 전략 수립 및 제안하였다. 즉, 새로운 RTS(Regression Testing Selection) 및 TCP(Test Case Prioritization) 기술기반 접근법 개발하였다. 새로운 RTS방법은 사전 제출(Pre-Submit)테스트에서 사용하는 접근법으로 변경 또는 신규 코드의 테스트를 위하여 사용한다. 그리고 TCP방법은 사후제출(Post-Submit)테스트로 코드 제출(Submission)이후 수행되는 회귀테스트에 이용하기 위해 제안된 방식이다. 제안된 방법을 평가하기 위하여 구글의 데이터에 적용한 실증적 연구를 수행하였다. 이 방법은 Failure window와 Execution Window를 사용하여 변경 또는 새롭게 개발된 테스트 스위트를 선택하고 추후 오류가 발생한 테스트 스위트의 우선순위를 선정하여 회귀테스트를 수행하는 방법이다.[4]

Dusica Marijan의 연구는 지속적인 통합 환경(CI)에서의 회귀테스트는 엄격한 시간 제약이 존재하므로 시간 제약조건을 만족시키고 테스트 목표 달성을 위한 효율적인 테스트 케이스의 실행이 필요하며, 테스트 케이스를 효율적으로 실행하기 위하여 우선순위 기술을 많이 사용한다.
하지만 기존의 많이 제안된 우선순위 기술은 시간 제약조건을 고려하지 않는다는 문제점이 있다는 것을 발견하였다.

이 문제를 해결하기 위해 산업파트너인 VCS의 테스트 부서와 협력하여 연속회귀테스트 우선순위 지정 접근방식인 ROCKET을 개발하였다. ROCKET을 실제 VCS산업에 적용하여 연속 회귀 테스트의 효율성 및 시간 향상을 입증하는 경험적 평가를 수행하였다.[5]

3. 조사 계획

조사 계획은 다음과 같이 연구질문과 질문에 따른 해답으로 구성하였다.

3.1 연구 질문

본 논문 연구를 위한 질문은 아래와 같이 선정하였다.

RQ1. 코드 변경과 테스트 케이스 갱신간의 연관성이 있는가?

RQ2. 코드 변경과 테스트 케이스 갱신간의 일관된 패턴이 있는가?

RQ3. 코드 변경에 따라 테스트 케이스의 자동갱신이 가능한가?

3.2 조사 대상

3.1절의 연구질문에 대한 답변을 위하여 분석 대상을 선정하였다. 본 연구를 위한 대상선정요건은 첫째, Java maven 프로젝트로 구성되어있고, 둘째, Release가 10회 이상이고, 셋째, 커밋 이력의 수가 최소 1000회 이상인 프로젝트여야 한다. 이에 따라 GitHub를 통해 spring-framework를 선정하였다. 이 프로젝트는 Java maven 프로젝트 내부에 main과 Test로 구성되어있는 프로젝트이다. 또, Spring-framework는 Release가 118회 Commit이 15,732회인 프로젝트로서, 위의 선정 요건을 충족시킨다.

3.3 조사 방법

조사 방법은 spring-framework의 15000여개가 넘는 커밋 이력중에서 가장 최근에 변경된 이력부터

조사하며 코드의 변경에 따라 테스트파일도 함께 변경된 커밋 이력을 찾는다. 다음 코드의 변경과 그에 따른 테스트 변경을 수집한 후, 변경에 따른 테스트케이스 변경을 분석하여 세 가지 연구 질문의 해답을 찾는다.

4. 실험 결과

변경된 커밋 중에서 1610개의 커밋 이력을 조사하고 그중 41개의 코드와 테스트 케이스 간의 변경 이력 데이터를을 찾을 수 있었다. 이를 바탕으로 분석한 결과 코드를 기준으로 다음과 같이 4 가지의 공통된 이력을 발견하였다.

1. 클래스의 추가 및 삭제
2. 메소드 추가, 삭제, 이름 변경,리턴 타입 변경
3. 매개변수 추가, 삭제, 변경
4. 코드 내 문장 변경

각각에 대한 이력을 표 1에 정리하였으며, 표1은 41개중 공통된 이력 하나만 예제로 표현하였다. 표1의 분류에서 찾은 공통된 이력을 코드 변경을 기준으로 테스트 케이스가 어떻게 변경하는지 정리하였으며, 본 조사 결과는 4.1절의 실험 질문에 대하여 다음과 같이 정리하였다.

4.1 RQ1 – 코드 변경과 테스트 케이스 갱신간의 연관성이 있는가?

표1을 보면 코드 변경과 테스트 케이스 갱신 간에 연관성을 찾을 수 있으며, 또한 어떤 방식으로 바뀌었는지를 알 수 있다. 연관성의 기준은 코드의 변경을 기준으로 테스트 케이스가 일관되게 변경이 되면 코드의 변경과 테스트 케이스 갱신 간의 연관성이 있다고 분류한다.

먼저 클래스에서의 코드 변경과 테스트 갱신에 대한 예제 코드를 보면 다음과 같다.

- 추가: DataBufferDecoder.java파일이 추가 되어 DataBufferDecoderTests.java라는 테스트 코드가 추가 되었다.
- 삭제: PortletRequestDataBinder.java라는 파일이 삭제되면 PortletRequestDataBinder-Tests.Java 파일이 전체 삭제 되었음을 알 수 있었다.

이 결과 코드에서 클래스의 추가 및 삭제가 되면, 테스트코드에서 역시 테스트 케이스를 위한 클래스가 추가 또는 삭제 되었다.

다음으로, 메소드 관련 코드와 테스트 케이스 간의 예제 코드를 살펴보면 메소드의 추가, 삭제, 이름변경, 리턴 타입의 변경으로 분류된다.

- 추가: 코드에서 public boolean isAssignableFrom 이라는 메소드가 추가되었을 경우 테스트 코드에서는 assertTrue(collectionClass.isAssignable-From)라는 assert구문이 추가 되었다.
- 삭제: encodeHttpHeaderFieldParam라는 메소드가 코드에서 전체 삭제되었을 때, 테스트 코드에서도 encodeHttpHeaderFieldParam 메소드가 전체 삭제되었다.
- 이름 변경: 코드에서 Mono라는 메소드가 Flux라는 이름의 메소드로 변경될 경우 테스트 코드에서도 Mono대신 Flux로 변경되어 사용되었다.
- 리턴 타입 변경: public static Mono<Void>-write(…)에서 Mono(Void)리턴 타입의 write함수가 Flux<DataBuffer> 리턴 타입인 public static Flux<DataBuffer>write(…)으로 변경되면서 테스트 코드에서 Mono<Void>writeResult = DataBuffer-Utils.write(flux,os);가 Flux<DataBuffer>write-Result = DataBufferUtils.write(flux,os); 로 변경되었다.

세번째는 메소드 매개변수의 추가, 삭제, 변경에 대한 예제 설명이다.

- 추가: StringDecoder(boolean splitOnNew-line, MimeType...mimeTypes)에서 MimeType...mimeTypes 이라는 매개변수가 추가되어 테스트 코드의 public class StringDecoderTests extends AbstractDataBufferAllocatingTestCase{…StringDecoder.allMimeTypes(true);…}에서 allMime Types(true)이 추가되었다.
- 삭제: public boolean canDecode(ResolvableType elementType, MimeType mimeType, Map<String, Object> hints)의 Map<String, Object> hints가 코드에서 삭제되었을 때, 테스트 코드의 public void canDecode(){…… MimeTypeUtils.TEXT_PLAIN, Collections.emptyMap()));}의 Collections.empty-Map이 삭제 되었다.
- 변경: public static final ResolvableType NONE = new ResolvableType(EmptyType.INSTANCE, null, null, 0);에서 EmptyType.INSTANCE라는 이름으로 매개변수의 이름이 변경되어 테스트 코드에서 assertThat(none.getType(), equalTo(Resolvable-Type.EmptyType.INSTANCE));로 ResolvableType.Em-ptyType.INSTANCE라는 이름으로 변수 이름이 변경되었다.

마지막으로 코드 내 문장이 변경될 경우이며,

- public boolean canEncode{……} 에서 리턴문이 변경된 예제이다. 코드 내의 리턴문이 변경되면서 테스트 코드의 canEncoder에서 assertFalse(th-is.encoder.canEncode(ResolvableType.NONE,null)); 이라는 assert구문이 추가 되었다.

위의 결과를 보았을 때, 코드의 변경과 테스트 케이스 갱신 간의 연관성이 보이므로 연구 질문 1에 대하여 긍정적인 해답을 얻었다.

[표 1] spring-framework 프로젝트의 코드 변경에 따른 테스트 케이스 변경 타입 분석표

Type		Commit	Code변경	Test변경
C	추가	88c5f59	DataBufferDecoder.java 추가	DataBufferDecoderTests.java 추가
	삭제	2b3445d	~~PortletRequestDataBinder.java~~	~~PortletRequestDataBinderTests.java~~
M	추가	1a30252	public boolean isAssignableFrom(ResolvableType other) { …… }	public void testSpr14648() throws Exception { … assertTrue(collectionClass.isAssignableFrom(fromReturnType)); …… }
	삭제	a8f7f75	~~public static String encodeHttpHeaderFieldParam(String input, Charset charset) { …… }~~	~~@Test public void encodeHttpHeaderFieldParam() {……}~~
	이름 변경	59e9094	public boolean ~~processRequest~~process(@Nullable CorsConfiguration config, ServerWebExchange exchange) {...}	public void actualRequestWithOriginHeader() throws Exception { …. this.processor.~~processRequest~~process(this.conf, exchange) …}
	리턴 타입 변경	c802827	public static ~~Mono<Void>~~Flux<DataBuffer> write(Publisher<DataBuffer> source, OutputStream outputStream) {...}	public void writeOutputStream() throws Exception { ..... ~~Mono<Void>~~ Flux<DataBuffer> writeResult = DataBufferUtils.write(flux, os);……}
P	추가	0662dbf	private StringDecoder(boolean splitOnNewline, MimeType... mimeTypes) {……}	public class StringDecoderTests extends AbstractDataBufferAllocatingTestCase { private StringDecoder decoder = StringDecoder.allMimeTypes(true);……}
	삭제	38f3d12	public boolean canDecode(ResolvableType elementType, MimeType mimeType, ~~Map<String, Object> hints~~)	public void canDecode() { assertTrue(this.decoder.canDecode(ResolvableType.forClass(ByteBuffer.class) MimeTypeUtils.TEXT_PLAIN, ~~Collections.emptyMap())~~);
	변경	fd53d2a	public static final ResolvableType NONE = new ResolvableType(~~null~~EmptyType.INSTANCE, null, null, 0);	assertThat(none.getType(), equalTo(~~nullValue()~~ResolvableType.EmptyType.INSTANCE));
S	변경	3a1d431	public static String replace(String inString, String oldPattern, String newPattern) { ..... ~~StringBuilder sb = new StringBuilder();~~ ~~int pos = 0;~~ ~~int index = inString.indexOf(oldPattern);~~ ~~int patLen = oldPattern.length();~~ if (index == -1) { return inString; } StringBuilder sb = new StringBuilder(inString.length()); int pos = 0; int patLen = oldPattern.length(); ..... return sb.toString(); }	public void testReplace() throws Exception { ..... s = StringUtils.replace(inString, "qwoeiruqopwieurpoqwieur", newPattern); ~~assertTrue("Replace non matched is equal", s.equals(inString));~~ assertSame("Replace non-matched is returned as-is", inString, s); s = StringUtils.replace(inString, oldPattern, null); ~~assertTrue("Replace non matched is equal", s.equals(inString));~~assertSame("Replace non-matched is returned as-is", inString, s); s = StringUtils.replace(inString, null, newPattern); ~~assertTrue("Replace non matched is equal", s.equals(inString));~~assertSame("Replace non-matched is returned as-is", inString, s); }

범례 : C- Class / M-Method / P- Parameters / S-Statements

[표 2] 코드와 테스트 케이스 분석표

Type		개수	자동갱신
클래스	추가	6	가능
	삭제	4	가능
메소드	추가	9	가능
	삭제	2	가능
	이름 변경	2	가능
	리턴 타입 변경	2	가능
메소드 매개변수	추가	2	가능
	삭제	1	가능
	변경	6	가능
문장	변경	7	불가능

4.2 RQ2 - 코드 변경과 테스트 케이스 갱신 간의 일관된 패턴이 있는가?

코드의 변경과 테스트 케이스 갱신 간의 일관된 패턴이 존재하는지 알아보기 위하여 표1의 항목에 대하여 조사한 커밋 이력에서 표2의 개수항목으로 정리하였다. 이 때, 표2의 개수항목에서 2개 이상의 공통된 변경이 발견되면 코드와 테스트 케이스 간의 일관된 패턴이 존재하는 것으로 분류의 기준으로 정의하였다.

먼저, 클래스의 추가는 코드에서 클래스가 추가될 경우 테스트 케이스에서도 동일하게 클래스가 추가되었다. 예를 들어, `DataBufferDecoder`라는 클래스가 코드에서 추가될 경우 `DataBufferDecoderTests`라는 클래스가 테스트 케이스에서도 추가 된다. 반대의 경우 코드에서 클래스가 삭제될 경우 테스트 케이스에서도 삭제된다. 이러한 예제와 같이 클래스의 공통된 변화가 추가는 6회 삭제는 4회가 있었다..

다음으로 메소드와 관련하여 코드에서 `public boolean isAssignableFrom()` 이라는 메소드가 추가 및 삭제되면 테스트 케이스에서도 `public boolean isAssignableFrom()`라는 메소드가 추가 및 삭제되었고, 조사한 데이터에서 이런 변화는 추가 9회, 삭제는 2회였다. 이름변경은 `processRequest`라는 이름의 메소드가 코드에서 `process`로 변경되었을 경우 테스트 케이스에서도 동일한 형태로 `process`라는 이름으로 변경이 2회 발생하였다. `Mono<Void>`가 `Flux<DataBuffer>`로 변경되는 형태의 리턴 타입의 변경은 2회가 발견하였다.

세 번째로 메소드 매개 변수에서 `StringDecoder` 메소드에서 `MimeType... mimeTypes`이라는 매개 변수 추가와 같은 형태의 변화는 2회 `canDecode` 메소드에서 매개 변수 `Map<String, Object> hints` 삭제와 같은 변경은 1회, `ResolvableType` 의 매개 변수 `null`이 `EmptyType.INSTANCE`과 같이 변경되는 매개변수의 변경은 6회였다. 마지막으로 public static String replace 코드 내 대입 문장이 if문으로 변경되는 것과 같은 형태의 문장의 변경과 관련되어 7회의 변경이 발견되었다. 비록 메소드 매개변수와 관련된 커밋 이력에서 1회만 발견된 데이터가 존재하였지만 이는 패턴이 없다고 단정지을 수 없다. 왜냐하면 15000개가 넘는 커밋 이력 중에서 일부만을 조사한 것이기 때문에 아직 조사하지 못한 커밋에 이와 관련된 패턴이 존재할 수 있기 때문이다.

위와 같은 변경 이력을 조사한 결과 코드의 변경에 따라 테스트 코드의 변경이 동시에 발생하는 변경 이력이 2회 이상 지속적으로 발생 하는 것을 알 수 있다. 즉, 코드의 변경에 따른 테스트 케이스 간의 일관된 패턴이 존재한다.

4.3 RQ3 - 코드 변경에 따라 테스트 케이스의 자동 갱신이 가능한가?

표2의 자동갱신 열에 본 연구 질문의 결과를 정리하였다. 코드의 변경에 따른 테스트 케이스의 자동 갱신이 가능할 것으로 판단되는 항목과 자동 갱신이 불가능 항목으로 분류하였다. 분류의 기준은 코드 변경에서 테스트 케이스 갱신을 유추할 수 있으면 자동화가 가능한 쪽으로, 아니면 자동화가 불가능한 쪽으로 분류하였다.

먼저 자동으로 갱신이 가능 할 것으로 판단되는 항목은 클래스의 추가 및 삭제, 메소드의 추가, 삭제, 이름변경, 리턴타입 변경 그리고 메소드 매개변수의 추가, 삭제, 변경이다. 이 데이터들의 코드 변경에 따른 테스트 케이스의 변경 형태를 살펴본 결과 코드가 변경을 기준으로 테스트 케이스의 변경을 충분히 유추할 수 있어 자동 갱신이 가능하다 판단하였다.

문장 변경에 따른 테스트 케이스는 자동 갱신이 불가능하다고 판단하였다. 문장의 변경은 코드가 변경될 때마다 테스트 케이스는 다양한 패턴으로 변경된다. 즉, 코드를 기준으로 테스트 케이스가 어떻게 변경될 지 유추가 불가능 하므로 자동화 불가능으로 판단하였다.

5. 결 론

단위테스트에서 사용자에게 편의를 제공하는 자동화된 테스트케이스 생성 기술이 많이 개발되었다. 그러나 이러한 기술은 사용자의 코드변경에 따라 자동으로 테스트 케이스를 갱신 하거나 코드 변경 시 기존의 테스트 케이스의 사용자 추가데이터는 새로운 버전에 반영되지 않고 삭제 된다는 문제점이 있다. 이를 해결하기 위한 방법을 연구하기 위하여 본 논문에서 실증적 연구를 수행하였다. 하나의 프로젝트를 선정하여 코드와 테스트 케이스 간의 변경을 조사할 수 있는 41개의 커밋 이력을 찾았다. 이를 분석한 결과 클래스, 메소드, 매개변수, 코드 문장의 변경으로 분류할 수 있었으며, 이 실증적 연구를 통하여 코드와 테스트 케이스는 서로 연관성이 존재하고 일괄된 패턴이 존재함을 찾게 되었다. 또한 코드의 변경에 따라 테스트 케이스의 자동 갱신의 가능 여부를 판단하였다.

하지만 좀 더 많은 샘플 코드와 분석을 통하여 코드의 변경에 따른 테스트 케이스의 자동갱신이 가능할 수 있는 좀 더 체계적인 연구가 필요하다. 향후 연구는 실제로 테스트 케이스의 자동갱신에 대한 기법을 개발하여 단위테스트에서의 테스트 케이스 진화연구에 기여하고자 한다.

6. 참고문헌

[1] “Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects”,
Hilton, Michael, et al. Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on. IEEE, 2016

[2] “Unit Test Generation During Software Development: EvoSuite Plugins for Maven, IntelliJ and Jenkins”, Andrea Arcuri,José Campos,Gordon Fraser, 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)

[3] "Efficient regression testing based on test history : An industrial evaluation",Edward Dunn Ekelund; Emelie Engström,2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)

[4] Elbaum, Sebastian, Gregg Rothermel, and John Penix. "Techniques for improving regression testing in continuous integration development environments." Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014.

[5] Dusica Marijan; Arnaud Gotlieb; Sagar Sen. "Test case prioritization for continuous regression testing: An industrial case study." Software Maintenance (ICSM), 2013 29th IEEE International Conference on. IEEE, 2013

데이터 기반의 강화학습을 통한 선박 자율 운항에 관한 연구

임지수 ○ 주재걸

고려대학교

js282kk@korea.ac.kr, jchoo@korea.ac.kr

Study on Data-Driven Autonomous Ship Navigation based on Reinforcement Learning

Jisu Leem○ Jaegul Choo

Department of Computer Engineering, Korea University

요 약

자율 주행 연구가 급증하는 상황 속에서 본 논문에서는 자율 선박 운항에서의 가상 시뮬레이션에 대한 연구를 하였다. 자동차, 선박, 비행기와 같은 운송 수단에서 발생하는 사고의 주요 원인은 인간의 미숙한 수행 오류가 큰 비중을 차지한다. 선박은 다른 운송수단과는 달리 선박의 종류에 따라 선체의 규모가 큰 차이를 보이며, 선체에 따른 동적 특성 정보가 다르므로 자율 운항 시 외부 선박 및 기타 물체와의 충돌을 예측 및 대비하기가 어렵다. 본 연구에서는 선박의 선체 크기에 대한 선박의 동역학 모델을 적용하여 추출된 선박의 6자유도 운동을 구성하고, 선체 크기에 따른 충돌 회피 알고리즘을 구현하였다. 구체적으로, 선박의 자율 운항을 하기 위하여 강화 학습 기법인 Q-learning 알고리즘을 활용하여 가상의 시나리오에 따른 시뮬레이션을 구현하였다. 마지막으로, 본 시뮬레이션을 활용하여 선박 충돌 회피 알고리즘의 효율성을 입증하였다.

1. 서 론

인공지능에 대한 연구가 발전함에 따라 자율 운행과 같이 자동차, 선박, 비행기와 같은 운송수단에 인공지능을 적용하는 연구가 활발히 진행되고 있다. 기존에는 사람과 운송수단 간의 원격 제어를 통한 주행 제어에 대한 연구가 많이 수행되었다. 하지만, 최근에는 인공지능 기술이 발전됨에 따라 사람이 개입하지 않고 운송수단 시스템만을 활용하는 자율 주행에 대한 연구가 활발히 진행되고 있다 [1,2]. 하지만, 사람의 개입을 필요로 하지 않는 자율 주행까지는 아직 많은 연구가 더 필요하며, 안전한 자율 주행을 하기 위하여 사람의 개입 정도에 따른 단계적인 연구가 이루어 지고 있다 [3].

본 논문에서는 선박의 데이터를 활용하여 해양에서의 자율 선박 운항에 대한 시뮬레이션 실험을 하였다. 선박의 데이터를 활용하기 위하여 선박 동역학 모델 알고리즘을 통하여 출력된 선박 6자유도 운동 데이터를 적용하였다 [4]. 사람이 개입하지 않는 상황 속에서 자율 선박 운항 시 해상의 부유물이나 유인 선박 등의 장애물에 대한 충돌 회피 알고리즘에 강화 학습 기법인 Q-learning 기법을 적용하였다 [5]. 가상 시뮬레이션 환경에 본 충돌 회피 알고리즘을 적용하여 선박 운항 시 발생할 수 있는 충돌을 회피하고, 선박 운항에 대한 최적화된 경로에 도달할 수 있었다.

2. 관련 연구

본 논문에서는 자율 주행 중 선박 데이터를 활용한 선박 자율 운항에 대한 연구 주제를 다룬다. 기존의 자율운항제어 시스템은 사람의 개입을 최소화하여 선박 자율 운항 로직에 의해 운항을 하는 무인 잠수정 (autonomous underwater vehicle)에 대한 연구로 진행되어오고 있다. 자율 무인 잠수정에는 관성 항법 시스템 (Inertial Navigation System) 중 관성 측정 장치 (inertial measurement unit)을 사용하여 선박의 위치 및 속도 등 선박운항에 대한 정보를 제공받을 수 있으며, 고가인 INS를 대신하여 자세 측정장치 (attitude heading reference

system)를 사용하는 연구도 진행되어왔다 [6,7]. 또한, 단안 카메라 영상을 기반으로 해상에서의 물체를 탐지하고 [8], 상대선박의 경로를 추정하여 운항하는 방식의 연구들이 진행되고 있다 [9]. 기존의 연구들은 규칙 기반으로 자율운항을 연구하고 있다 [8,9]. 본 연구는 출발점과 도착점, 그리고 장애물의 위치 정보만을 제공하여 선박의 직선운동의 데이터 기반으로 강화학습을 적용하여 자율 운항을 하는 데에 초점을 두었다.

3. 본 론

3.1 강화 학습 기법

본 연구에서는 선박의 6자유도 운동 데이터를 활용하고, 장애물과의 충돌을 회피하여 선박 운항 시 최적화된 경로로 자율 운항 하기 위해, 강화 학습 기법을 활용하였으며, Markov Decision Process (MDP) 를 활용한 Q-learning 알고리즘을 적용하였다. 본 논문에서는 Q-learning 에서 현재 상태에서 최대의 보상 (reward)을 획득하는 최적의 행동 정책을 학습하였다. 또한, 보상을 계산할 때 할인된 보상의 총계 (sum of discounted rewards)의 기대값으로 계산하였다.

Q-learning 알고리즘이 시작되기 전에 Q 함수는 임의의 값으로 초기화된다. 각 시간 t에 에이전트는 어떠한 상태 s_t에서 행동 a_t를 취하고 새로운 상태 s_{t+1}로 전이한다. 이 때 보상 r_t가 얻어지며, Q 함수가 갱신된다. 또한, α는 learning rate의 값이다. 알고리즘의 핵심은 이전의 값과 새 정보의 가중 합 (weighted sum)을 이용하는 간단한 값 반복법이다. 수식1에서 확인할 수 있다 [10].

$$Q(s,a) \leftarrow (1-\alpha)\cdot Q(s_t,a_t) + (r_t + \gamma \cdot \max_a(s_{t+1},a))$$

수식 1 Q-learning 알고리즘

3.2 가상 시뮬레이션 환경

본 연구에서는 최적화 된 경로를 탐색하기 위하여 두가지 가상 시나리오를 구성하였다.

첫번째 시나리오는 선박 운항 시 출발점과 도착점만을 주고, 다른 외부 장애물이 없는 설정이며, 두번째 시나리오는 선박 운항 시 출발점과 도착점과 함께, 부가적인 장애물이 경로 중에 함께 주어지는 설정이다.

4. 실험 결과

4.1 실험 환경

4.1.1 선박 환경 구축

본 논문에서는 임의의 컨테이너선을 설정하여 실험하였으며, 상세 구성은 표1과 같다.

Length	175.00m
Breadth	25.40m
Draft(fore)	8.0m
Draft(aft)	9.0m
Draft(mean)	8.5m
Displacement volume	$21{,}222m^3$
Propeller diameter	6.533m

표 1 선박 환경 구축

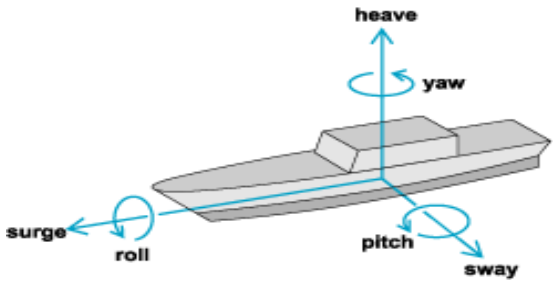


그림 2 선박의 6자유도 운동

본 연구에서는 선박의 움직임에 필요한 RPM과 타각의 변화를 통해 선박의 6자유도 운동에서의 출력 값을 확인하기 위한 실험을 하였다.

4.1.2 강화 학습 환경 구축

선박 운항을 위해서는 방향을 조정하기 위한 타각과앞으로 나아가기 위한 추진력을 조정하기 위한 RPM 이 필요하다. 또한, 배의 상태는 선박의 6자유도로 표현할 수 있다. 따라서, 본 실험에서는 강화학습 실험 설정을 위해 시간 t 에서의 State 로 배의 6자유도 상황을 설정하고, 이때 취할 수 있는 Action으로 타각과 RPM 을 설정하였다.

시뮬레이션 환경에서의 외부 장애물을 설정하기 위해 격자 형태의 지도를 설정하였다. 외부 장애물은 실험이 진행되는 동안 움직이지 않는다고 가정한다.

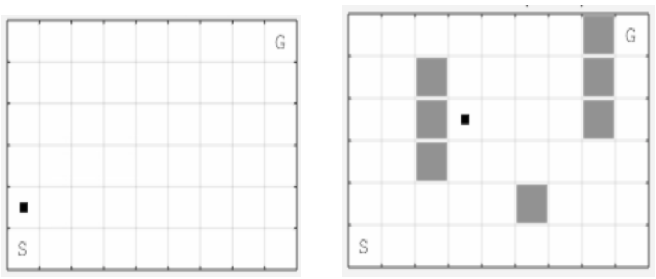


그림 3 가상 시뮬레이션 환경 구축. 왼쪽 그림: 장애물이 없는 경우, 오른쪽 그림: 장애물이 있는 경우

4.2 실험 결과

4.2.1 선박의 6자유도 운동

본 논문에서는 2차원인 x와 y축 평면에 대한 선박의 움직임을 표현하는 실험을 하였다. 그림 4는 선박이 외부 환경 요인이 없는 경우 직진 운동을 할 때에 선박의 움직임을 나타낸다. 그림 5는 그림 4와 동일한 외부 환경 요인에 선박 운항 시, 가장 중요한 요소인 RPM과 타각에 대한 움직임을 나타내었다. 타각과 RPM에 따라 선박 운항 시 선박의 회전도를 알 수 있다. 타각과 RPM 출력 값에 따라 선박 운항 시 선박 6자유도 운동으로 인하여 지그재그로 변화하는 선박의 움직임을 그림 6에서 확인할 수 있다.

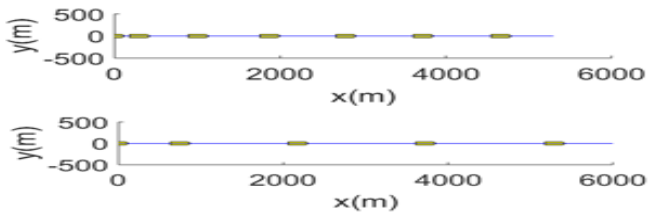


그림 4 선박 운항 속도에 따른 직선운동

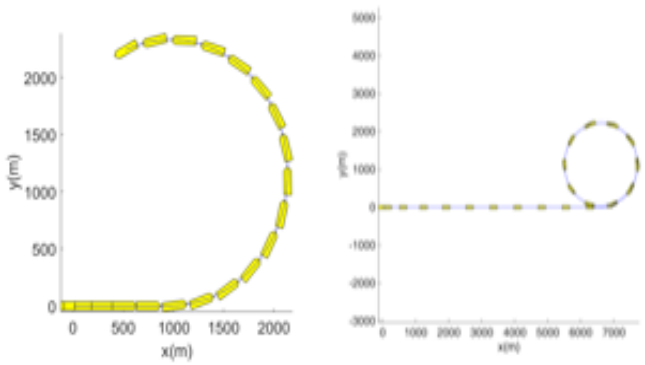


그림 5 선박 운항 RPM에 따른 회전운동

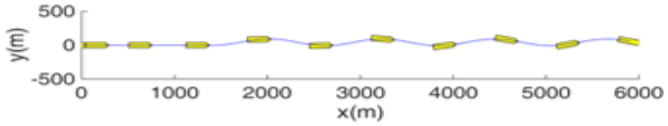


그림 6 선박의 6자유도 운동에 따른 지그재그 운동

4.2.2 최적 경로

그림 7과 8은 강화 학습 기법 중 Q-learning을 활용하여 선박이 출발점에서 도착점까지 이동하는 데에 걸리는 스텝 수에 대한 그래프이다. x축은 출발점에서 도착점까지 도달하는 각각의 에피소드를 의미하며, x축은 한번 운항을 할 때 몇번의 움직임으로 도달했는지를 의미한다. 최적 경로를 운항하기 위하여 본 논문에서는 그림 4와 같이 선박이 직진으로 움직이는 경우에 대한 실험을 하였다. 150번의 에피소드를 통해 학습하였으며, 0.5초 당 1스텝이 움직이도록 설정을 하였다. 그림 7의 최대 스텝 수는 92이고, 최소 스텝 수는 8이며, 그림 8의 최대 스텝 수는 210이고, 최소 스텝 수는 8이다. 그림 7과 8에 대한 스텝 수는 강화 학습 기법의 특성 상 랜덤 한 값이 주어지므로 매 실험마다 다른 스텝 값이 나오게 된다. 그러나, 그림 7과 8의 공통점은 처음 출발점과 도착점까지 경로를 찾는 데에 많은 스텝수가 필요하지만 한번 경로를 찾게 되면 최적화 된 경로를 찾아가므로 스텝의 수가 줄어든다는 것이다. 그림 8은 그림 7과 달리 장애물을 만나면 회피 하기 위해 많은 스텝 수가 발생됨을 알 수 있다. 하지만, 움직이지 않는 장애물을 회피한 뒤 도착점을 찾으면 그림 7과 비슷한 스텝 수로 선박의 움직임을 알 수 있다. 그림 9 는 시나리오 1의 이상적인 최단 경로를 활용하여 시나리오 2의 환경에서 점차적으로 최단 경로를 찾아 나아가는 것을 확인할 수 있다.

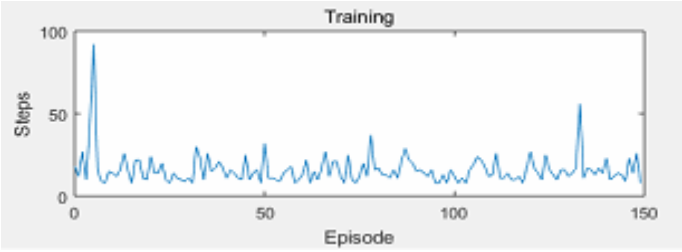


그림 7 시나리오 1. 출발점에서 도착점까지 이동하는 에피소드에 대한 스텝 수

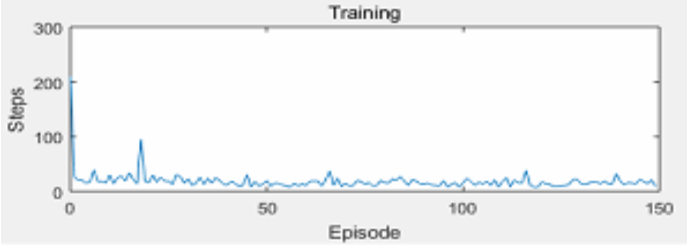


그림 8 시나리오 2. 출발점에서 도착점까지 이동하는 에피소드에 대한 스텝 수

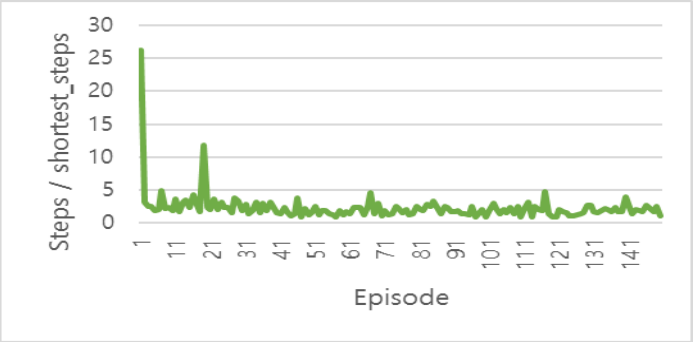


그림 9 이상적인 최단경로를 통한 시나리오 2 스텝 수 변화 추이

5. 결론 및 향후계획

본 논문에서는 선박 자율 주행에서의 최적화 경로 탐색 및 외부 장애물과의 충돌 회피에 대한 실험을 하였다. 선박 운항 시 외부 장애물이 없을 경우에 대한 실험 결과와 외부 장애물이 있을 경우에 대한 실험 시, 학습 초기에는 출발점에서 도착점까지의 경로를 찾는데 오랜 시간 (step)이걸렸지만, 강화 학습 기법을 활용하여 반복적으로 학습을 한 결과 최적의 경로를 학습 하여 빠른 시간 (적은 step수)에 도착점까지 도달하는 것을 보여주었다. 본 연구는 선박에 대한 데이터를 활용하여 자율 선박 운항에 대한 시뮬레이션을 구성하였다. 본 연구를 활용하여 자동차, 비행기 등 다양한 운송 수단에서의 자율 주행에 많은 기여를 할 수 있을 것이라 판단된다.

향후 연구는 외부 환경에 대하여 그림5와 6과 같은 선박의 움직임을 적용한 선박 자율 운항 도출과 움직이지 않는 장애물 외에 선박 간의 충돌 회피 및 해수면 위에 떠있는 부유물 등 움직임이 있는 장애물에 대한 충돌 회피에 대한 연구를 진행할 것이다.

[참고 문헌]

[1] Seong-Gyun Jeong, Jiwon Kim, Sujung Kim, Jaesik Min. “End-to-end Learning of Image based Lane-Change Decision”, 2017 IEEE Intelligent Vehicles Symposium (IV)

[2] 박정홍, 한정욱, 김진환, 손남선, 김선영, “무인수상선의 자율운항을 위한 단안 카메라 기반의 근거리 선박 자동 탐지”, 제어로봇시스템학회 논문지 23(6), 416-423, 2017.6

[3] Kenneth H. Goodrich, Jim Nickolaou, and Mark D. Moore. "Transformational Autonomy and Personal Transportation: Synergies and Differences between Cars and Planes", 16th AIAA Aviation Technology, Integration, and Operations Conference, AIAA AVIATION Forum, (AIAA 2016-3604)

[4] 윤현규,강남선. “정수중을 활주하는 고속선의 6자유도 운동 모델링 및 시뮬레이션”, 대한조선학회논문집 Vol. 53, No. 1, pp. 10-17, February 2016

[5] Lawrence A. Birnbaum, Gregg C. Collins Morgan Kaufmann. “Machine Learning Proceedings 1991: Proceedings of the Eighth International Workshop (ML91)”, 363-367, 2014

[6] Rong Zhu, Dong Sun, Zhaoying Zhou, Dingqu Wang. "A Linear Fusion Algorithm for Attitude Determination Using Low Cost MEMS-Based Sensors", Measurement, Vol. 40, No. 3, pp. 322-328, 2007.4

[7] Dae-Hae Kim, Sang-Min Lee, Gil-Young Kong. “A Study on the Wireless Ship Motion Measurement System Using AHRS”. Journal of Navigation and Port Research Vol.37 No.6 pp.575-580. 2013

[8] 박정홍, 김진환, 손남선. "해양환경에서의 영상기반 물체탐지 및 추적 기법 연구". 2012년도 한국해양과학기술협의회 공동학술대회, 510-513, 2012.5

[9] 박지성, 김진환. “단안 카메라를 이용한 수중 정밀 항법을 위한 모델 기반 포즈 추정”, 로봇학회논문지 제11권 제4호, p.226-234 , 2016.11

[10] Christopher J. C. H. Watkins, Peter Dayan. “Technical Note Q-Learning”, Machine Learning, 8, 279-292, 1992

메모리 상태 모니터링 데몬 소프트웨어 개발

손문규 ○, 안남영, 이규선, 심응보, 유승주
서울대학교, SK 하이닉스
earthtoss@gmail.com, namyoung.ahn@SK.com, kyusun.lee@sk.com,
eungbo.shim@sk.com, sungjoo.yoo@gmail.com

Memory Status Monitoring Daemon Software Development

Mungyu Son○, Namyoung Ahn, Kyusun Lee, Eungbo Shim, Sungjoo Yoo
Seoul National University, SK Hynix

요 약

데이터 센터의 서버환경에서 메모리 에러 관리는 안정적인 서비스를 유지하기 위해서 가장 중요한 주제이다. 하지만 서버환경에서는 많은 메모리 모듈이 장착되고 각각의 메모리 상태 정보를 모두 모니터링하고 분석하기는 매우 힘들다. 본 논문에서는 이를 해결하기 위해 기존의 모니터링 프로그램인 mcelog를 사용하여 우리가 필요로 하는 의미 있는 정보를 수집할 수 있는 데몬을 개발한다. 기존의 mcelog 코드를 수정해 우리가 얻을 정보는 다음과 같다. (1) CPU와 메모리의 기본정보, (2) CPU와 메모리 관련 추가 정보 (3) 에러가 발생한 시간, uptime 시간, 에러 수, MSR 정보 그리고 CPU 온도이다. 실험은 E5-Xeon 서버 시스템으로 진행하였고, 하드웨어 장비 JIG를 이용하여 메모리 에러를 발생시킴으로써 에러 정보를 얻었다. 본 논문에서는 메모리 에러를 발생시키고 그 때 발생한 에러를 수정된 mcelog를 이용해 모니터링하고 의미 있는 정보를 수집하는 것을 목적으로 하지만 추후에 이러한 하드웨어에서 보고 해주는 다양한 상태 정보를 기반으로 메모리 에러의 종류를 분석, 분류할 수 있다. 나아가서는 에러를 예측함으로써 사전에 방지할 수 있을 것으로 기대된다.

1. 서 론

최근 서버 및 고성능 컴퓨팅 시스템에는 많은 데이터를 처리하기 위해 많은 메모리를 사용한다 [1]. 최신 서버에서 이처럼 많은 메모리와 클러스터를 사용하기 때문에 작동 중에 발생하는 메모리 에러의 비율도 증가하고 있다. 서버 시스템에서는 일반적으로 Error Correction Code (ECC) 방법을 사용하여 하드웨어 메모리 에러를 수정한다. 하지만 계속해서 증가하는 메모리 에러 때문에 고객들은 다양한 메모리 상태 정보를 모니터링하고 그 결과를 분석함으로써 예방하기를 원한다. 본 논문에서는 이러한 문제점을 해결하기 위해 기존의 mcelog를 수정하여 의미 있는 에러 정보를 수집할 수 있는 데몬을 개발한다.

2. 본 론

서버 시스템에서 의미 있는 메모리 에러 정보를 수집하기 위해서는 기본적으로 다음과 같은 위치 정보가 필요하다. 하나의 에러가 발생했을 때 발생한 에러가 1) 어떤 CPU 소켓인지, 2) 어떤 메모리 채널인지, 3) 어떤 메모리 DIMM인지, 4) DIMM 내부 어떤 Rank인지, 5) 에러가 발생한 DQ에 해당하는 칩이 16개 중에서 어떤 것인지, 그리고 6) Bank Group, Bank, Row, Column 주소 등 많은 위치 정보와 함께 시간 정보를 수집해야 한다. 또한 에러의 위치뿐만 아니라 에러의 의미 있는 정보를 얻기 위해서는 에러와 관계된 다양한 시스템 자원 및 어플리케이션 소프트웨어를 모니터링 해야 한다. 이 절에서는 우리가 사용한 실험 환경과 개발한 소프트웨어에 대해서 설명할 것이다.

2.1 인텔 E5-Xeon 서버

데이터 센터 및 서버 환경에서는 약 90%이상의 대부분의 시스템이 인텔 E5-Xeon 서버를 사용하기 때문에 우리는 이 시스템에서 실험을 진행하였다 [2]. 인텔 E5-Xeon 서버는 Figure 1과 같이 두 개의 CPU socket으로 구성되어 있으며, 각각 4개의 channel로 구성되어 있다. 또한 각각의 channel은 3개의 DIMM으로 구성되고, 각각의 channel은 address, command 및 data bus를 공유한다. 이와 같이 최신 인텔 서버의 기본적인 구성은 하나의 서버에 최대 12개의 메모리 모듈을 장착할 수 있다. 인텔 E5-Xeon 서버는 내부적으로 Machine Check Architecture (MCA)라는 에러 관리 구조를 가지고 있다. 대부분 정보가 비공개이고 구할 수가 없지만 우리는 기존에 배포된 mcelog 및 다양한 소프트웨어를 바탕으로

에러를 관리할 수 있는 방법에 대해서 많은 방법을 찾았으며, 이 방법을 바탕으로 실험을 진행할 수 있었다.

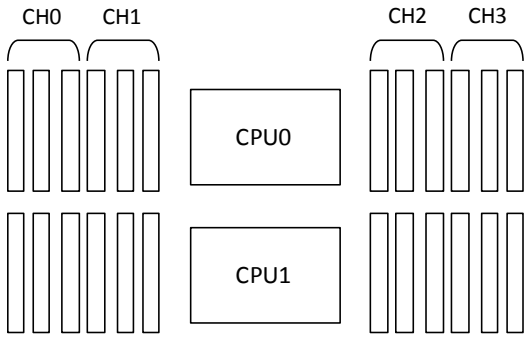


Figure 1 인텔 E5-Xeon 서버

2.2 인텔 E5-Xeon 서버의 에러 분류

인텔 E5-Xeon 서버는 기본적으로 ECC를 Single Device Data Correction (SDDC)를 사용한다 [3]. 여기서 single device는 x4 기준으로 하나의 package 칩을 말하는 것으로 x4로 모듈을 구성 시에 하나의 chip이 완전히 fail이 발생하여도 에러를 수정할 수 있는 메커니즘이다. Figure 2는 인텔 E5-Xeon 서버에서 에러의 분류를 나타낸 것이다 [4]. 그림의 왼쪽의 detected이고 그 밑에 corrected인 경우에만 기본적인 ECC 메커니즘인 SDDC로 에러가 제대로 수정되는 경우이며, 나머지 경우는 모두 감지조차 안되거나 아니면 감지는 되었는데 정정을 못하는 경우를 나타낸 것이다.

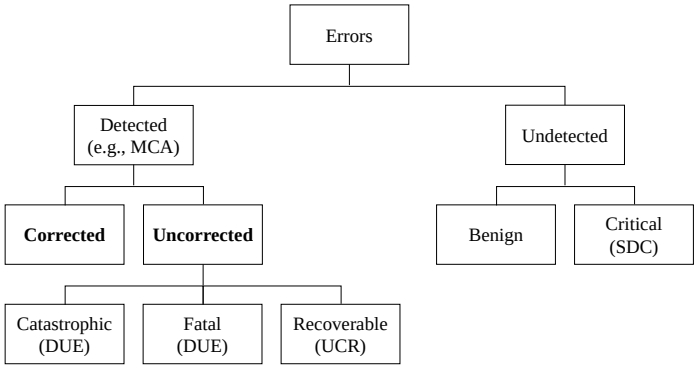


Figure 2 인텔 서버의 에러 분류

실제 데이터센터 및 서버 환경에서는 corrected 에러가 발생하여도 당장은 전혀 문제가 발생하지 않는다. SDDC가 있어 데이터를 수정하면 되기 때문에 서비스에는 전혀 지장이 발생되지 않기 때문이다. 또한 SDDC 메커니즘이 한 개의 package가 모두 fail이 되어도 fail이 발생되지 않기 때문에 확률적으로 uncorrectable 에러가 발생하기 매우 힘들다. 하지만 Figure 2에서와 같이 Silent Data Corruption (SDC) 혹은 Detectable but Uncorrected Error (DUE)가 발생하면 시스템이 정지하거나 재부팅을 하기 때문에 서비스에 영향을 미쳐서 서비스가 중단되거나 늦어질 수 있다.

은행, 금융기관과 같은 서비스의 품질이 매우 중요한 서버환경에서는 서버가 정지하면 서비스에 큰 영향을 끼쳐서 막대한 경제적 손실을 가져올 수 있다. 그렇기 때문에 에러 관리가 매우 중요하다. 이러한 uncorrectable 에러에 의해서 발생하는 서비스 정지는 그 시간의 크기에 따라서 경제적 손실이 더욱 커지므로 에러를 사전에 방지할 수 있어야 한다. 따라서 데이터센터나 서버환경에서는 uncorrectable 에러를 사전에 방지하기 위해서 우선 correctable 에러를 모니터링하여 최대한 uncorrectable 에러가 발생하지 않도록 하는 것이 중요하다.

2.3 mcelog

mcelog의 가장 간단한 역할은 디스크 또는 syslog에 에러를 기록하는 것이다. 기본적으로 로그 파일은 /var/log/mcelog 위치에 생성된다.

인텔 E5-Xeon 서버에서 MCA를 통해서 에러의 정보를 추출하기 위해서는 Model Specified Register (MSR) 정보가 있어야 접근을 할 수 있다. 이 MSR 정보는 인텔 서버 플랫폼마다 서로 다른 레지스터 정보를 가지고 있으며 동작하는 방식이 약간씩 다르다. 하지만 이 정보는 모두 비공개이기 때문에 따로 문서를 구할 수가 없어서 MSR을 통한 MCA의 접근을 할 수가 없었다.

다행히 mcelog라는 리눅스 기본 배포 프로그램에서 우리는 MCA를 접근할 수 있는 방법을 찾았으며, 이 방법을 통해서 메모리 에러 정보를 추출할 수 있었다. mcelog는 MSR을 통해서 MCA를 접근하지 않고 mcelogd라는 새로운 장치를 통해서 MCA를 접근한다. mcelogd는 리눅스의 기본 배포판에서 제공되는 장치로써 메모리 에러가 발생했을 때 이 장치에 데이터가 입력이 되고 mcelog 데몬 프로그램은 이 장치에 소프트웨어 인터럽트를 설정하여 에러가 발생하였음을 확인할 수 있다. 리눅스 멀티플렉싱 기법인 epoll 함수를 사용하여 이 소프트웨어 인터럽트를 구현하였으며, 이러한 방법은 MSR 레지스터를 polling하는 방법 대비 CPU 자원을 사용하지 않아도 되기 때문에 매우 유용한 방법이다. 또한 mcelog 데몬은 page offline기능을 가지고 있어서 이 데몬을 사용하면 사용하지 않을 때와 비교하여 사전에 더 많은 메모리 uncorrectable fail을 방지할 수 있는 기능도 수행한다. mcelog는 리눅스에서 사용하는 메모리 관리를 위한 최소 단위인 page 단위로 correctable error의 개수로 page를 off-line할 수 있다. 따라서 부팅 후에 correctable error가 많이 발생한 page는 새로운 page로 교체되어서 실제 나중에 발생할 수 있는 uncorrectable error를 방지할 수 있다.

2.4 인위적인 에러 발생

메모리 에러는 실제 데이터 센터나 서버환경에서는 아주 많은 서버가 아주 많은 메모리 모듈을 사용하기

때문에 빈번히 발생하지만 실제 몇 개의 서버에서는 에러의 발생을 관찰하기가 매우 어렵다. 우리는 한 대의 서버에서 에러를 모니터링하고 프로그램을 개발하기 때문에 이러한 환경에서 실제 메모리 불량이 발생하기를 기다리면 correctable 에러 하나의 발생을 기다리는데 1년이 걸릴 수도 있다.

Figure 3 하드웨어 JIG

따라서 우리는 Figure 3과 같은 하드웨어 JIG를 사용하여 메모리 에러를 발생시킨다. 이 하드웨어 장비는 메모리 DIMM 아래에 꽂는 장비로 on/off 스위치를 조작함으로써 해당 PIN을 물리적으로 분리시켜 메모리 에러를 발생시킨다. 또한 실제 에러를 발생시키기 위해서는 해당하는 메모리 주소를 접근해야 하기 때문에 리눅스에서 fopenmp로 malloc함수를 사용하여서 현재 가진 메모리 용량의 대부분을 계속해서 접근하도록 프로그램을 만들었다. 이 방법을 사용하여 현재 가지고 있는 CPU 코어를 모두 사용하여 최대 대역폭으로 메모리를 접근하여 최대한 빨리 메모리 불량 현상이 발생할 수 있도록 하였다.

2.5 수정된 mcelog

Figure 4은 우리가 수정한 mcelog 블록 다이어그램을 나타낸다. 의미 있는 정보를 보기 위해 기존의 mcelog 데몬에서 생성하는 로그 정보들은 모두 나오지 않도록 하였다. (1) CPU와 memory 기본정보들은 커널의 CPUinfo, meminfo 파일로부터 정보를 받아왔고, 정보를 모은 뒤 우리가 생성할 로그 파일에 로깅 되도록 하였다. (2) CPU와 메모리 관련 추가 정보는 /proc/으로부터 가져왔다. 마지막으로 (3) 에러가 발생한 시간, uptime 시간, 에러 수, program (workload), MSR 정보 그리고 CPU 온도는 Htop 과 MSR의 DTS 값, offset 값을 읽어 계산하였다.

우리는 메모리 에러와 함께 의미 있는 값을 얻기 위해서 시간, CPU 사용, 발생 위치, 온도, 워크로드 정보를 메모리 에러가 발생할 때마다 로깅하게 하였다. 시스템이 시작하고 얼마나 시간이 흐른 다음에 에러가 발생하는지 확인하기 위해서 current time과 uptime 정보를 추가하였다. 시스템에서 현재 사용하는 system process, user process 그리고 IO waiting에 사용하는 CPU 정보를 불량이 발생될 때마다 로깅하였다. 또한 MSR address와 MSR status를 이용하여 차후에 에러가 발생한 정확한 위치를 찾기 위해서 MSR 레지스터 정보를 로깅하였다. 그리고 온도와 에러의 상관관계를 확인하기 위해서 CPU 온도를 에러와 함께 표시하였다.

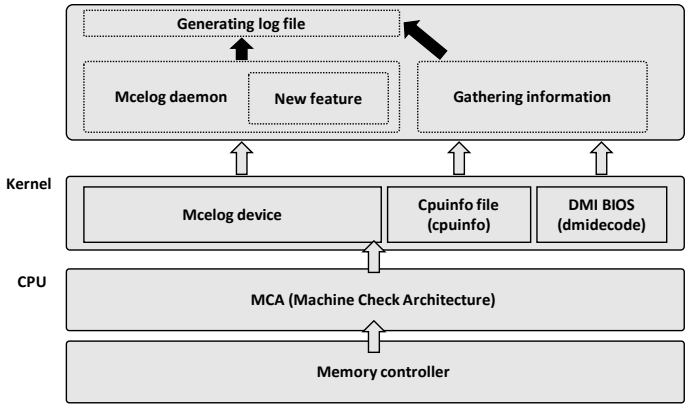


Figure 4 수정된 mcelog 블록다이어그램

3. 실 험

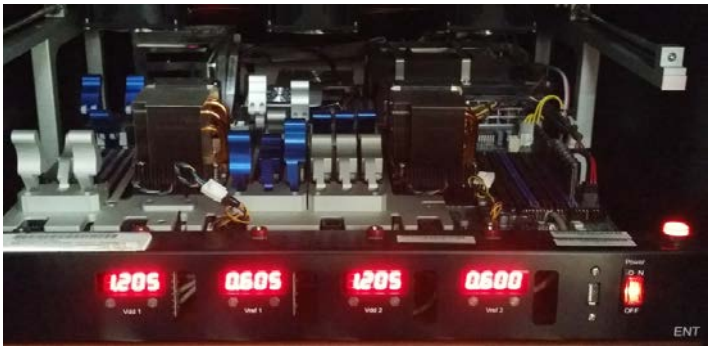


Figure 5 E5-Xeon 서버

우리는 Figure 5 의 E5-Xeon 서버 시스템에서 실험을 진행하였다. 실험은 메모리 모듈을 최소화 하여 불량 발생을 최대한 빨리 확인하기 위해서 2개의 모듈만 사용하였으며, 그 중에서 한 개의 모듈만이 하드웨어 JIG를 사용하여서 불량을 인위적으로 발생시켰다. 우리가 사용한 E5-Xeon 서버의 환경은 다음 Table 1과 같다.

Table 1 서버 성능

ITEM	VALUE
CPU	E5-2698 v4
CPU Core 수	80
CPU Frequency	2.20 GHz
Memory Density	32 GB

```
[cpu]
freq = 1200.439 MHz, cores = 20, cache size = 50 MB
[memory]
total density = 32043 MB
CPU CH DIMM    Manufacturer        Part Number          Serial Number
-----------------------------------------------------------------------------------
 0    0    0 :        Hynix         HMA42GR7AFR4N-TF        26080D39
 1    0    0 :        Hynix         HMA42GR7AFR4N-TF        260868AD
-----------------------------------------------------------------------------------
Start Daemon : Wed Nov 15 15:29:32 2017
Error Count = 1, MSR STATUS = 88000050020000b0, MSR ADDR = 0,
MSR MISC = 900000000002000, CPU socket = 1, CE by normal ECC,
CPU package temp = 39, current time = 4203440, uptime = 1510727372,
CPU for system processes = 0.9, CPU for user processes = 0.2,
CPU for waiting IO = 0.0, mem_total = 32043, mem_free = 213,
mem_used = 31043, mem_buffers = 785, swap_total = 32648,
swap_free = 32597, swap_used = 51, swap_avail = 256, workload = ./a.out
```

Figure 6 로그 결과

Figure 6는 우리가 수정한 mcelog 를 실행시켰을 때 나오는 output log를 나타낸다. Figure 6에서 [cpu] 와 [memory] 는 하나의 서버가 정해지면 바뀌지 않는 정보이기 때문에 항상 출력할 필요 없는 정보이다. 만약 계속 출력될 경우 log의 양이 많아질 뿐만 아니라 추후에 수집한 정보들을 이용해 분석하는 것 또한 어려워지기 때문에 계속해서 출력하기보다는 mcelog 데몬을 실행하였을 때, 한번만 나오면 되도록 설정하였다. 이 값은 데몬이 시작할 때 무조건 표시하기 때문에 만약 uncorrectable 에러에 의해서 mcelog 데몬이 재부팅 되었을 경우에는 이 값이 또 다시 표시가 된다. 이것을 확인하면 uncorrectable 에러가 발생되었다는 것을 알 수 있다. 그리고 cpu 정보 밑의 memory 정보는 현재 CPU 소켓 0번째의 Channel 0번에 DIMM 0번의 자리에 DIMM이 하나 장착 되어 있고 해당하는 메모리 모듈의 제조사 및 part 번호와 serial 번호를 표시하였다. 그리고 CPU 소켓 1에도 동일하게 channel 0번 DIMM 0번 위치에 메모리 모듈이 하나 더 장착되어 있다.

다음은 mcelog에서 표시하는 각각의 로그들을 나타내는 것이다.

(1) Error Count: 에러는 한번에 여러 개가 동시에 발생할 수도 있고 한 개씩 발생할 수도 있다. 이 값은 MCA에서 저장하는 값으로 순간적으로 발생한 에러의 총 수를 표시한다. 만약 에러가 1보다 큰 값인 경우, 해당하는 나머지 로그 값은 모두 마지막 에러 카운트에 대한 정보이다.

(2) MSR STATUS: 이 값은 MSR 레지스터에서 STATUS의 정보이다. MSR 레지스터 정보는 인텔의 CPU 종류마다 모두 다른 address 정보를 가지고 있으나, mcelog에서는 모두 동일하게 쉽게 확인할 수 있다. 하지만 이 값의 정확한 정보는 인텔 내부 문서로 확인이 어려워서 차후 더 많은 연구로 확인할 예정이다.

(3) MSR ADDRESS: MSR STATUS와 동일하게 MSR의 ADDRESS 레지스터 정보 값을 가진다.

(4) MSR MISC: MSR STATUS와 동일하게 MSR의 MISC 레지스터 정보 값을 가진다.

(5) CPU for system processes, CPU for user processes, CPU for waiting IO: 서버의 CPU 자원이 system process, user process, IO 를 처리하기 위해서 사용하는 utilization의 정도를 나타낸 것이다.

(6) CPU socket: 현재 에러가 발생한 CPU 소켓의 위치를 나타낸다. 인텔 CPU의 종류에 따라서 MSR STATUS와 MSR MISC 정보만 가지고 메모리 모듈이 발생한 Channel위치와 DIMM위치까지 알 수 있는 CPU Family가 있으나 최신 CPU Family는 대부분 많은 Core를 사용하기 때문에 이러한 위치 정보를 알 기 위해서는 또 다른 방법이 필요하다.

(7) CE by normal ECC: 에러가 발생한 종류를 나타낸다. Detectable UE의 경우에도 표시가 되나 대부분 normal ECC에 의한 불량이다.

(9) CPU package temp: 서버 CPU의 package temperature를 나타낸다.

(10) current time: 현재 시간

(11) uptime: 부팅 후 시간

4. 결론 및 향후 연구 방향

본 논문에서 우리는 서버 환경에서 다양한 메모리 상태 정보를 모니터링 하고 수집하는 데몬을 개발하였다. 실험은 E5-Xeon 서버 시스템에서 진행하였고, 하드웨어 JIG를 이용하여 메모리 에러를 발생시켰다. 본 논문에서는 메모리 에러를 발생시키고, mcelog를 이용해 에러가 발생하였을 때의 정보들을 수집하는 것만을 목적으로 하였지만 추후에 이러한 하드웨어에서 알려주는 다양한 상태 정보를 기반으로 머신 러닝 등의 방법을 이용하여 메모리 에러의 종류를 분석, 분류하고 예방하는 방법에 대해 연구할 것이다.

5. Acknowledgement

본 논문은 SK 하이닉스(주)의 위탁에 따른 '연구용역'의 결과입니다.

6. 참고 문헌

[1] Kleen A, “Mcelog: Memory error handling in user space,” In Proc. Linux Kongress 2010.

[2] INTEL, "List of Intel Xeon microprocessors," https://en.wikipedia.org/wiki/List_of_Intel_Xeon_microprocessors.

[3] SDDC, "Intel x4 Single Device Data Correction”, http://www.ece.umd.edu/courses/enee759h.S2003/references/29227401.pdf, 2002.

[4] Mahesh Natu et al., “Autonomic foundation for fault diagnosis,” In the Intel Technology Journal, Vol. 16, No. 2, pp. 8-32, 2012.

국방 소프트웨어 자원관리시스템의 소스 코드 품질을 위한 자바 파서 개발을 통한 적용 사례

박보경* 문소영* 서채연* 김영철*○ 김광남** 최영식** 신상훈**

*홍익대학교 소프트웨어공학 연구실, **육군 정보체계관리단

*{park, msy, chyun, bob ○ }@selab.hongik.ac.kr, **kma6471@hanmail.net,

**pmhcyshj@hanmail.net, **godshin2000@naver.com

An Applied Practice through developing Java Parser for Code Quality of the National Defense Software Resource Management System

Bo Kyung Park*, So-Young Moon*, Chae Yun Seo*, R. Young Chul Kim* ○,

Kwang-Nam Kim**, Young-Sik Choi**, Sang-Hoon Shin**

*SELab., Hongik University, **ROK Army

요 약

현재 육군에서는 고품질의 소프트웨어 개발을 위한 품질관리 방법을 적용하는데 어려움을 겪고 있다. 이를 위해, 육군에서는 소프트웨어 품질 개선이 가능한 소프트웨어 가시화 방법을 적용하였다. 하지만 이 방법을 국방 소프트웨어 자원관리 시스템에 적용하였으나 다음과 같은 문제가 발생하였다. 먼저, 1) 기존의 소스 네비게이터는 전자정부프레임워크에서 정확한 분석을 수행하지 못한다. 2) 소스 네비게이터로 분석된 정보들 중 육군에 불필요한 데이터(reference_class, Instance 등)가 다수 존재한다. 본 논문에서는 국방 소프트웨어 자원관리시스템에 적합한 파서 개발 및 적용 방법을 제안한다. 개발된 파서를 적용하여 국방 소프트웨어 자원관리시스템의 가시화를 수행하였다. 개선된 소프트웨어 가시화 방법을 통해, 고품질 소프트웨어 개발 및 유지보수성 향상을 높이고자 한다.

1. 서 론

국방 소프트웨어는 첨단 무기체계의 핵심요소로 운용되고 있으며, 전차, 함정, 전투기 등 주요 무기체계 에서 차지하는 비중이 점차 증가하고 있다. 소프트웨어 중요성이 증가함에 따라, 육군에서는 고품질의 소프트웨어 개발을 위한 품질관리 방법을 적용하고 있다. 하지 만 다음과 같은 문제로 인해 품질관리 적용에 많은 어려움을 겪고 있다. 1) 1인 1건의 개발 추진으로 개발 자 성향에 따라 개발 패턴이 다르기 때문에 복잡도의 일관성이 부족하다. 2) 요구사항이 지속적으로 변경됨에 따라 요구 분석, 설계, 구현의 전 과정이 일치하지 않는다. 3) 개발 후 운용 지원 부서로 개발업무가 이관되기 때문에, 운용 유지 측면에서 다양한 개발 패턴을 유형별로 분석하고 조치하는데 많은 시간이 소요된다. 이러한 문제를 해결하기 위해서는 소프트웨어의 비가시성 문제 해결이 가능한 소프트웨어 가시화 방법이 필요하다. 소프트웨어 가시화는 소프트웨어 개발의 과정과 구조를 파악함으로써 전문적이지 않더라도 고품질을 위한 소프트웨어 개발과 품질 관리가 가능하다[1,2]. 하지만 육군에서는 기존의 소프트웨어 가시화 방법을 적용하였으나, 여러 문제점이 발생하였다. 현재 육군의 개발 환경은 전자정부프레임 워크 기반이다. 하지만 코드 분석 도구인 소스 네비게이터는 전자정부프레임워크에서 정확한 분석을 수행하지 못한다. 그 예로 부분적으로 소스 코드의 구문 파싱이 안 되며, if 문 같은 제어문을 분석하지 못한다. 이러한 문제를 해결하기 위해서, 육군 자원관리 시스템 분석을 위한 파서(Parser)를 개발하였다. 역공학을 통해 소프트웨어 품질 향상이 가능하도록 개발한 자바용 파서를 적용하여 소스코드를 구문 분석하였다. 이 방법을 통해 고품질 소프트웨어 개발 및 유지보수성 향상을 기대할 수 있다.

2. 국방 소프트웨어 자원관리의 소프트웨어 가시화를 위한 파서(Java Parser for Army, JPA) 개발

기존의 소프트웨어 가시화 방법은 소스 네비게이터를 기반으로 소스 코드를 분석한다. 소스 네비게이터는 다양한 언어를 지원하며, 코드 구문을 해석하여 소스 코드를 분석한다. 기존 방법을 국방 소프트웨어 자원 관리 시스템에 적용해 보았으나 다음과 같은 문제점이 발생하였다. 국방 소프트웨어 자원관리 시스템은 전자정부프레임워크 기반이기 때문에 Annotation의 일부 파싱 오

* 본 논문은 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2017R1D1A3B03035421)

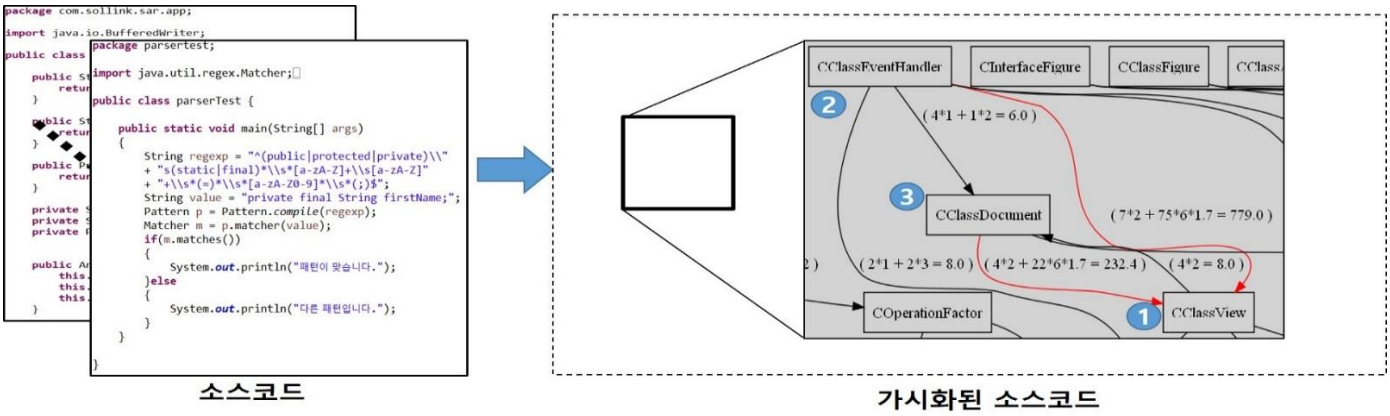


그림 2. 소프트웨어 가시화 결과

류가 발생하였다. 두 번째로 소스 네비게이터로 분석된 정보들은 SQLite에 저장되는데, DB에 축적된 데이터 중 불필요한 reference_class 및 Instance 정보 가 다수 존재한다. 또한 저장된 데이터는 16진수 코드 값 및 # 등의 특수 문자로 저장되기 때문에 데이터를 분석하는데 많은 시간이 소요된다. 따라서 국방 소프트웨어 자원관리 시스템에 적합한 별도의 파서 개발이 필요하다.

2.1 파싱을 위한 정규 표현식

그림 1에서 소스 코드는 정규 표현식을 사용하여 분석한다. 자바에서 정규 표현식을 사용하기 위해서는 Pattern과 Matcher 클래스 객체가 필요하다[3]. Pattern 클래스의 complie() 메소드를 사용하여 정규표현식을 Pattern 객체로 생성한다. 컴파일된 된 정규식 패턴은 Matcher 객체로 생성되며, 입력된 문자열이 정의한 패턴에 일치하는지 여부를 판단하게 된다.

```
public boolean isAnnotation(String
stringLine) {
    String packagePattern = "(@)[A-Z]{1}[a-
     zA-Z]+\\s*(\\()*\\s*[=\"a-zA-Z\\s/._0-9]*
     \\s*(\\))*";
    Pattern p = Pattern.compile(packagePatter
     n);
    Matcher m = p.matcher(stringLine);
    boolean patternExist = m.find();
    if(patternExist)
    {
          System.out.println("패턴이Annotatio
          Pattern Line : "+m.group(0));
    }
    return patternExist;
}
```

그림 1. 정규 표현식의 예제

2.2 비교 결과

표 1. 기존 시스템과 개발된 시스템 비교 결과[2]

항목	기존 시스템	개발된 시스템
산출물 검증	부분 가능	가능
소스코드 가시화	불가	가능
개발일정 관리	수작업	자동화
개발자의 Bad Smell 식별	불가	가능
복잡도 측정	불가	가능
소스코드 품질 개선	미비	가능

표 1은 기존에 구축한 시스템과 소프트웨어 가시화 구축을 통한 비교 결과이다. 기존 시스템에서 부분적으로 가능하거나 불가능했던 항목들이 JPA가 적용된 시스템에서는 산출물 검증, 복잡도 측정 등이 가능하였다.

그림 2는 소프트웨어 가시화 결과이다. 이 그래프는 JPA로 분석된 데이터를 기반으로 출력된 가시화 결과이다. 네모는 클래스를 나타내며, 클래스와 클래스를 연결하는 화살표는 클래스 간의 결합도를 의미한다. 그림 2에서 클래스 간의 높은 결합도를 살펴보면 다음과 같다. ① CClassView와 ② CClassEventHandler 간의 결합도 값은 779이며, ③ CClassDocument와 ① CClassView 간의 결합도는 232.4이다. 이 결합도들은 다른 클래스 간의 결합도 수치보다 높음을 알 수 있다. 따라서 이 2가지 결합도는 많은 코드 수정이 필요하다.

3. 결 론

본 논문은 국방 소프트웨어 자원관리 시스템의 아키텍처 가시화를 위한 파서 개발 및 적용방법을 제안한다. 기존 연구에서 사용하던 소스 네비게이터는 소스코드 파싱 오류로 인해 annotation에서 method 인식 오류 및 제어문 미인식 등의 문제가 발생하였다. JPA 파서는 정규표현식을 이용하여 소스 코드를 분석하기 때문에 소스 네비게이터의 문제점을 보완할 수 있었다. 이 파서를 적용함으로써, 국방 소프트웨어 자원관리 시스템에 필요한 소프트웨어 가시화에 적용하여 올바른 소프트웨어 개발 및 유지보수성 향상을 기대할 수 있다. 또한 개발된 파서는 각 개발 환경에 맞게 변경 가능하기 때문에, Java 외에도 다른 언어에 적용 가능하다.

참고문헌

[1] NIPA SW 공학센터, “SW 개발 품질관리 매뉴얼 (SW Visualization)”, 2013.12.

[2] 박보경, 문소영, 서채연, 김영철, “국방 소프트웨어 자원관리시스템의 소스 코드 품질 향상을 위한 소프트웨어 가시화 적용 구축 사례”, 2017 한국정보과학회 동계학술대회, 2017.

[3] https://docs.oracle.com/javase/tutorial/essential/regex/

비전공자들을 위한 영어 텍스트 데이터 분석 프로그래밍 교육 방법론 (특정 사례를 중심으로)

오하영
아주대학교
hyoh79@gmail.com

Programming Teaching Methodology of Text Data Analysis for Non-Applicants

Hayoung Oh
Ajou University

요 약

소프트웨어 교육이 남녀노소에 관계없이 광범위하게 수행됨에 따라 대학가에서도 전공에 상관없이 교양 필수로 프로그래밍 및 소프트웨어 교육이 한창이다. 본 논문에서는 비전공자 학생들 중에서 영문학과를 대상으로 교육자가 학과 학생들이 관심있어 하는 텍스트 데이터 예제를 초기에 수집하고 활용하여 텍스트 데이터 분석, 프로그래밍 방법론을 교육한 결과 학생들의 습득 능력 및 속도가 생각보다 빠르다는 결과를 보여준다.

키워드: 비전공자를 위한 텍스트 데이터 분석 교육 방법론, 영문학과를 위한 텍스트 분석 교육 방법론, 비전공자를 위한 데이터분석

1. 서 론

MIT 스크래치 및 Naver 엔트리 등의 기초 프로그래밍 플랫폼을 누구나 한번은 들어봤을 것이다. 하지만, 해당 프로그래밍 언어는 레고 블록 맞추기 등으로 작동되고 직접 컴퓨터 명령어 코딩이 아니기 때문에 대부분의 초, 중고 학생들과 같은 초보자에게 적합하다. 따라서 대학가에서는 주로 파이썬, 매틀랩, 데이터분석을 위한 R 및 컴퓨터와 인간 등과 같은 기초과목부터 심화 프로그래밍 방법론의 기초 교양 교육이 한창이다.

소프트웨어 교육 및 프로그래밍 교육은 다양한 각 과의 학생들이 향후 해당 전공 컨텐츠에 이를 융합하여 실제로 상품화 하거나 가시화할 수 있기 때문에 개인 역량차원에서 아주 큰 잠재력 중에 하나로 손꼽힌다. 하지만 타과 전공 학생들은 컴퓨터 자체에도 익숙하지 않을 뿐만 아니라 “인간과 인간”에서 “인간과 컴퓨터”로 왜 일을 시키는 주체와 일을 하는 주체가 변화되고 있는지 그 중요성을 모르기 때문에 교육자는 동기부여 및 소프트웨어 변화의 속도 등을 흥미롭게 전달할 수 있어야 한다.

결과, 본 논문에서는 80명 정도의 비전공 영문학과 학생들을 대상으로 영문학과 학생들이 선호하는 영문 텍스트를 미리 조사하고 해당 데이터를 활용하여 영어 데이터 분석 수업을 수행했을 때 각 학생들의 만족도가 생각보다 높다는 것을 보여준다. 한 학기 동안 영문학과 학생들이 선호하는 영어 데이터를 수집하고 개발된 예제를 활용하여 수업 후, 학생들의 만족도를 조사하여 분석해본 결과 성별, 전공 및 취업 후 활용도 등에 따라서 학생들이 받아들이는 체감 (Experience)이 다양하다는 것을 알 수 있었다.

2. 배경지식 및 제안하는 기법

비전공자 학생들은 크게 인문사회계열, 상경계열, 순수 자연계열 및 간호/의과계열등으로 나뉠 수 있다. 각 과의 학생들은 각 과에서 배우는 내용(Contents)가 다르기 때문에 획일적인 프로그래밍 방법론은 동기부여에 도움이 되지 않는다. 결과, 교육자는 각 과를 수업하기 전에 해당 학생들이 현재 흥미로워하는 전공 주제, 평소에 자주/많이 접하는 컨텐츠 및 연관 키워드를 파악하고 이를 바탕으로 수업 예제를 만들어서 수업하는 것이 매우 중요하다. 이는 학생들의 수업 집중도 및 흥미도에 매우 크게 영향을 주기 때문에 상관관계가 매우 높다고 판단된다 [1][2].

결과, 저자는 영문학과 수업을 시행하기 전에 그림. 1과 같이 영문학과에 맞춰진 영어데이터 분석 기초 교육 방법론을 개발해서 수업을 진행했다. 첫째, 영문학과 커리 큐럼을 파악하고 각 전공에서 공통적으로 활용되는 영미 소설 및 학생들이 주로 관심있게 보는 YouTube 영상, 미드, 뉴욕 타임즈 기사 등에서 영어데이터를 수집하고 분석했다. 결과, 학생들은 영문학과 학생임에도 불구하고 영어를 다각도에서 효율적으로 공부하는 방법론을 YouTube 동영상 시청에 매우 의존하고 있음을 알 수 있었으며, 현지 영어를 공부하기 위해서 “미드”에 매우 많은 시간을 할당하는 것을 알 수 있었다. 따라서 해당 데이터를 미리 크롤링하고 분석하여 수업 예제 및

커리큐럼으로 새로 구성된 교재 개발 [3]을 수행했다. 학생들의 흥미도를 반영해서 개발된 강의자료(교과목개발) 및 교재 [3](“컴퓨터와 R을 활용한 텍스트 데이터 분석 기초-오하영, 박정식 공저, 휴먼사이언스)를 활용한 결과 예상대로 학생들의 강의 만족도는 다 각도에서 매우 높은 편임을 알 수 있었다.

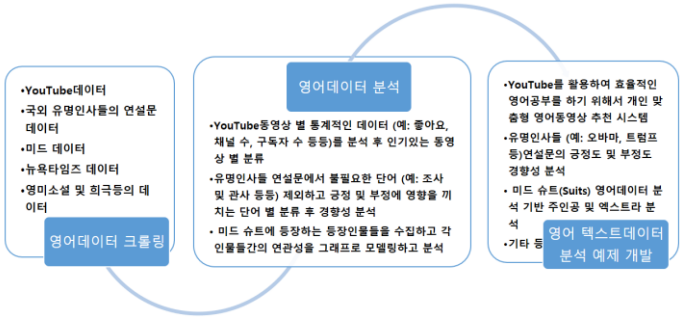


그림. 1. 영문학과에 맞춰진 영어데이터 분석 기초 교육 방법론

3. 성능 평가

앞서 언급했듯이, 영문학과 두 분반 (대략 80명)의 강의 평가 데이터를 모두 수집해서 해당 항목별 데이터 분석을 실행한 결과, 남학생보다 여학생이 많은 영문학과 특성상 여학생들의 만족도가 큰 편으로 나타났다. 기존의 소프트웨어 관련 학과들에서는 대부분의 남학생들이 컴퓨터나 프로그래밍에 익숙해서 성적이 좋은 경향이였다. 하지만, 본 논문 결과에서 증명 했듯이 여학생들의 프로그래밍 습득력 및 흡수도가 생각보다 높기 때문에 이는 향후 타 비전공자들에게 해당 방법론으로 프로그래밍을 교육해도 남학생들은 물론 여학생들에게도 충분히 프로그래밍 및 소프트웨어 교육이 가능하다는 것을 알 수 있다.

그림. 2는 남학생 및 여학생 별 강의에 대한 이해도를, 그림. 3은 본 강의의 주당 공부시간을, 그림. 4는 취업 후 해당과목의 효과를 그림. 5는 강의에 대한 평균 만족도를 보여준다. 앞서 언급했듯이 비율적인 측면에서 상대적으로 여학생이 많은 과목이였기에 여학생들의 만족도가 다각도에서 높은 편임을 알 수 있다. 또한, 남학생과 여학생들을 모두 고려했을 때 평균 값도 대체적으로 높기 때문에 해당 교육 방법론은 간단하고 당연할 수 있지만 매우 중요한 기본이고 비전공자를 가르치려는 교육자가 제일 먼저 꼭 고려해야하는 기법임을 알 수 있다.

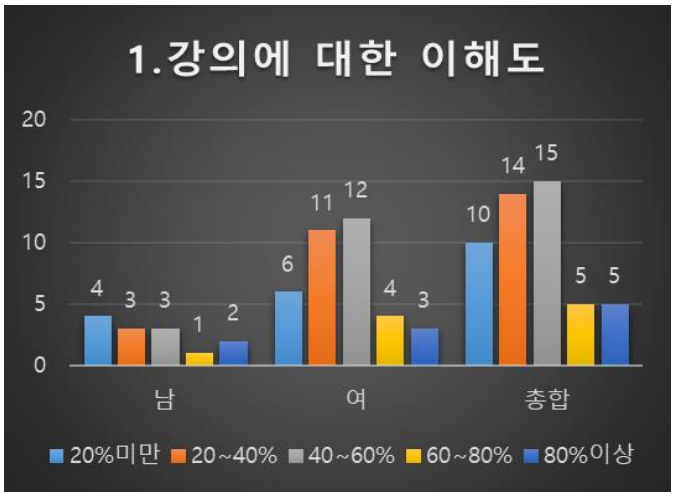


그림. 2. 강의에 대한 이해도

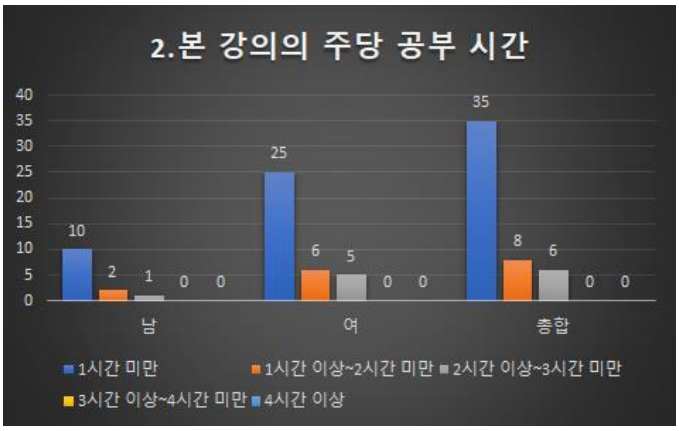


그림. 3. 본 강의 주당 공부 시간

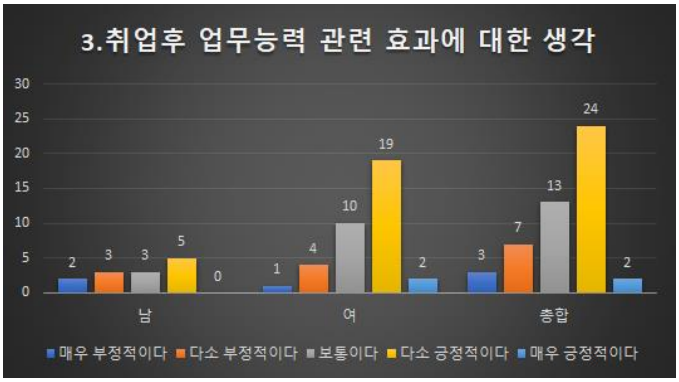


그림. 4. 취업 후 업무능력 관련 효과에 대한 생각

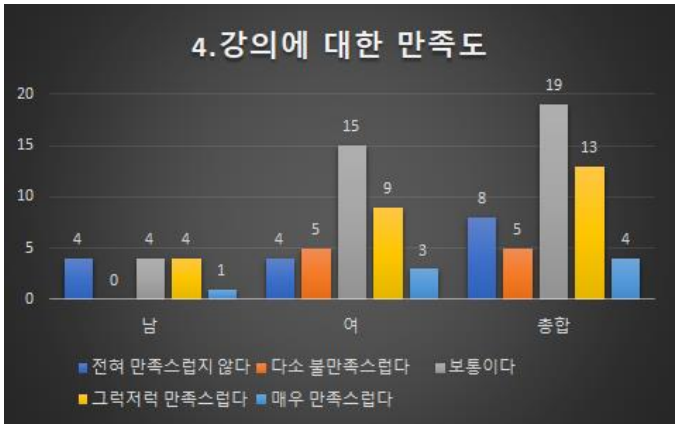


그림. 5. 강의에 대한 만족도

4. 결론

본 논문에서는 비전공자학생들 중에서 영문학과를 대상으로 수강하고자하는 학생들의 관심있는 데이터를 미리 크롤링 및 분석하여 교육하는 방법론을 제시했다. 이를 바탕으로 수업한 결과 예상대로 학생들의 강의 만족도가 높은 편임을 알 수 있었다.

참고문헌

[1] 한옥영, 김재현, “비전공자 소프트웨어 교육을 통한 컴퓨팅 사고력 향상에 대한 연구”, 한국컴퓨터교육학회 학술발표대회논문집 21권1호

[2] 권정인, 김재현, “비전공자를 위한 컴퓨팅사고 기반의 SW교육과정에 관한 연구”, 2016년도 한국인터넷정보학회 춘계학술발표대회 논문집 제17권 제1호, 2016.4, 247-248

[3] 오하영, 박정식, “컴퓨터와 R을 활용한 텍스트 데이터 분석 기초”, 휴먼사이언스, 2017년 10월

동적 데이터 획득 시스템 가시화 프레임워크

임성민[1,O], 김수동[2,*]
숭실대학교 컴퓨터학과[1], 숭실대학교 소프트웨어학부[2]
{hcooch2ch3, sdkim777}@gmail.com

Visualization Framework for Data Acquisition Systems Dynamics

Seong Min Yim[1,O], Soo Dong Kim[2,*]
Department of Computer Science and Engineering, Soongsil University[1],
School of Software, Soongsil University[2]

요 약

산업자동화 시스템은 다수의 센서들을 통하여 운영환경의 정보를 수집하고, 분석하여 필요한 기능을 수행하고 제어한다. 이를 위해 시스템이 센서로부터 데이터를 획득할 때, 데이터의 획득 과다 또는 부족은 시스템이 적절히 작동하는데 문제를 발생시킬 수 있다. 따라서 본 논문은 데이터의 필요에 맞게 데이터 획득 방식을 자율 제어하는 플랫폼의 가시화 계층의 설계를 제안한다.

1. 서 론

산업자동화 시스템은 다수의 센서들을 통하여, 운영환경의 정보를 수집하고[1][2], 이를 분석하여 필요한 기능을 수행하고 제어한다[3]. 예를 들면, 제조 공정 라인 모니터링 시스템의 경우, 제조 공정 라인의 과부하를 확인하기 위해 짧은 주기의 온도 센서를 이용한다. 여러 개의 온도 센서를 통해 제조 공정 라인의 각 모듈에서 발생하는 열이 어느 정도인지를 측정하여 과부하를 판단하고 제어한다.

이런 시스템의 센서들을 짧은 주기(Interval)로 환경 정보를 측정하고, 이를 필요로 하는 어플리케이션에게 전달한다[4]. 이 센서들의 측정 방식, 측정 주기, 일관성이 동적으로 변화한다[5]. 이를 위해서 센서들과 어플리케이션 사이에 데이터 획득 플랫폼 등 장치가 필요하다[6].

이런 플랫폼 개발의 중요한 요구사항 중 하나가, 획득 데이터의 효과적인 가시화(Visualization)이다. 효과적인 가시화의 필요성은 다음과 같다.

- 빠른 주기의 센서 데이터 측정치를 어플리케이션 및 사용자에게 적시(Timely)에 출력하여 인지하게 할 필요성
- 획득 효율성 메트릭 값을 사용자에게 적시에 출력하여 인지하게 할 필요성
- 데이터 획득 방식의 변화를 사용자가 적시에 효과적으로 인지하게 할 필요성
- 사용자가 플랫폼의 조작 및 작동에 대한 변화를 인지하게 할 필요성
- UI의 직관적인 이해를 돕도록 가시화 계층 컴포넌트의 구역의 분리 필요성

최근 연구에서는[2][4] 데이터를 효과적으로 획득하기위해 한가지 전송방식을 유지하면서 데이터 발생 주기만을 조절하는 것과는 달리, 본 연구에서는 데이터 전송방식 세가지를 자율적으로 제어하고 이를 Visualization Layer로 데이터 획득 효율성의 실시간 변화를 사용자에게 보여줌으로써 높은 Usability와 User Experience을 제공하는 것을 목적으로 한다.

2. 센서 데이터의 특징

일반적으로 센서 데이터에는 다음의 3가지 특징이 있다.

첫째로, 센서는 주기적으로 데이터를 발생시킨다. 예를 들어 초음파 센서는 센서 전방의 위치한 물체와의 거리 값을 주기적으로 측정하고, 온도 센서는 센서 주위의 온도를 주기적을 측정한다.

둘째로, 데이터 획득 방식이 한가지 혹은 두 가지 방식으로 제한되어 있다. 대부분의 센서는 Push 방식으로 데이터를 전달하며, Pulling 또는 Notify-N-Fetch 방식으로 데이터를 전달하는 센서도 존재 한다. 이러한 데이터 획득방식을 모두 지원하는 센서는 드물며 보통 센서들은 한, 두가지 방식으로 데이터를 전달한다.

셋째로, 실제 필요한 데이터에 비해 과다한 데이터가

* 교신저자 (Corresponding Author)

센서에서 발생할 수 있다[7]. 예를 들어 약 10초의 주기로 제품을 제조하는 공정 라인의 모니터링 시스템에서 10초보다 작은 주기로 온도를 측정하여 과부하를 판단하도록 하는 것은 비효율적이다. 제품을 제조하는 주기와 비슷한 주기로 온도를 측정하는 것이 효율적이다.

이러한 센서의 특징을 반영하여 주기적으로 발생하는 센서데이터를 적시에 보여주도록 하였다. 필요 이상의 데이터가 센서로부터 전송될 때 데이터 획득 효율성의 저하가 일어나게 된다. 이때 플랫폼은 데이터 획득 방식을 변경하며 이를 사용자가 효과적으로 인식할 수 있도록 하였다.

3. 효과적인 가시화 계층 (Visualization Layer) 설계

가시화 대상 정보와 이 정보를 효과적으로 표현하는 방법에 관해 설명한다.

3.1. 데이터 소스의 가시화

데이터 소스의 측정 변수

- Eff.Threshold(%)

  획득 효율성 자율변경을 위한 최소 임계치를 나타낸다. 이 값이 클수록 빈번하게 획득방식이 변경된다.

- SRC.Interval(millisecond)

  데이터 소스가 데이터를 측정하는 주기를 나타낸다.

- Interval.Range(%)

  데이터 소스가 데이터를 측정하는 주기의 상하 변동 폭을 나타낸다. SRC.Interval이 1000ms이고 Interval Range가 30% 일 경우, 데이터 소스의 측정값이 최소 700ms에서 최대 1300ms사이의 랜덤 간격으로 발생한다.

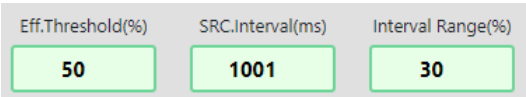


그림 1. 데이터 소스 측정 변수 가시화 컴포넌트

데이터 소스의 정보

데이터 소스의 측정 변수 이외에 데이터 소스의 정보는 아래의 3가지가 있다. **Data Source Index**는 데이터 소스와 싱크가 플랫폼 내부에서 몇 번째 순서인지를 나타내며. **Data Source Name**은 데이터 소스의 이름이며 Data **Acquisition Method**는 데이터 소스의 획득 방식을 나타낸다.

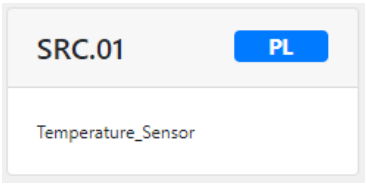


그림 2. 데이터 소스 정보 가시화 컴포넌트

3.2. 데이터 싱크의 가시화

데이터 싱크의 획득 변수

- Pull Interval(ms)

  데이터 싱크의 획득방식이 Pulling일 때, Pulling의 주기를 나타낸다.

- Pull Int.Range(%)

  데이터 싱크의 획득방식이 Pulling일 때, Pulling의 주기의 상하 변동 폭을 나타낸다. Pull.Interval이 1000ms이고 Pull Int.Range가 30% 일 경우, 데이터 소스의 측정값이 최소 700ms에서 최대 1300ms사이의 랜덤 간격으로 발생한다.

- Push Util Rate(%)

  데이터 싱크의 획득방식이 Pushing 일 때, Pushing된 데이터를 실제로 활용하는 비율을 나타낸다. Push Util Rate이 50%일 경우, 전달된 데이터의 절반만 활용한다.

- NF Fetch Rate(%)

  데이터 싱크의 획득방식이 Notify-n-Fetch 일 때, 데이터 싱크가 소스로부터 실제 값을 가져오는 비율을 나타낸다. NF Fetch Rate이 50% 일 경우, 발생한 데이터 중 절반만 싱크가 가져온다.

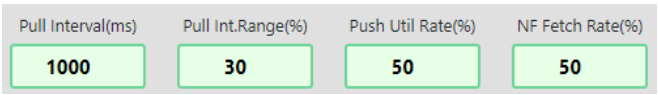


그림 3. 데이터 싱크 획득 변수 가시화 컴포넌트

데이터 싱크의 정보

데이터 소스와 마찬가지로 싱크도 가시화 해야 할 3가지 정보가 있다. **Data Sink Index**은 데이터 싱크가 플랫폼 내부에서 몇 번째 순서인지를 나타낸다. **Data Sink Name**은 데이터 싱크의 이름을 나타낸다. **Data Acquisition Method**은 데이터 싱크의 데이터 전송 방식을 나타낸다.

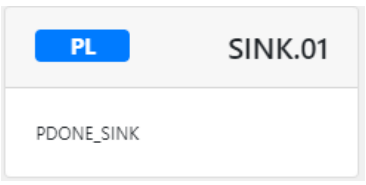


그림 4. 데이터 싱크 정보 가시화 컴포넌트

3.3. 데이터 소스에서 발생된 데이터의 가시화

데이터 소스에서 발생된 데이터는 설정된 주기에 맞게 전송되며 이를 아래의 그래프로 가시화하였다. 센서에서 발생한 데이터가 웹소켓을 통해 전달되어 실시간으로 그래프에 반영되며, 데이터의 주기가 길고 짧음에 관계없이 전달되는 즉시 그래프를 갱신하여 정확하게 반영되도록 하였다.

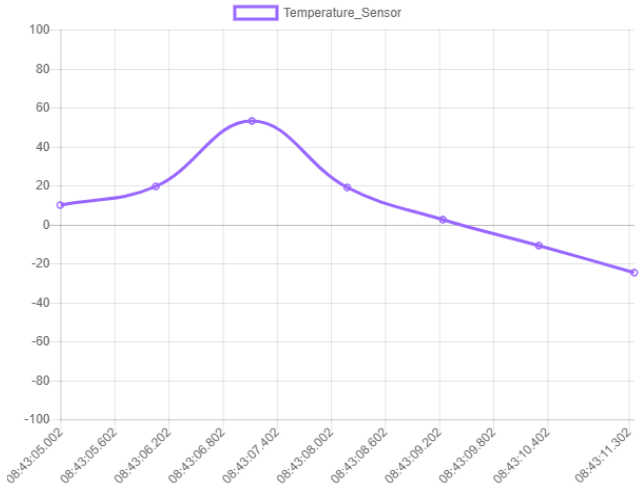


그림 5. 데이터 소스 발생된 데이터 가시화 그래프 컴포넌트

3.4. 데이터 획득 효율성 메트릭 값의 가시화

획득 효율성 메트릭 값

- DAE(%)

  DAE(데이터 획득 효율성, Data Acquisition Efficiency)은 데이터 싱크가 얼마나 효율적으로 데이터를 획득하는 지를 나타내는 지표이다.

- DAR(%)

  DAR(데이터 획득률, Data Acquisition Rate)은 소스에서 발생된 데이터가 싱크에서 획득되는 비율을 나타낸다.

- ATE(%)

  ATE(획득 시간 효율성, Acquisition Time Efficiency)은 소스에서 발생된 데이터가 데이터 싱크에서 확보하는데 소요되는 시간의 효율성을 나타낸다.

- ARE(%)

  ARE(획득 자원 효율성, Acquisition Resource Efficiency)은 소스에서 발생된 데이터가 싱크가 확보하는데 소요되는 자원의 효율성을 나타낸다.

위의 주요 메트릭 값 4가지 이외에 텍스트로 보여줄 메트릭 값은 다음과 같다. **RMR(%)**은 Pulling에서 데이터의 중복 획득율을 나타낸다. **DUR(%)**은 Pushing에서 데이터 활용 비율을 나타낸다. **NAR(%)** Notify-n-Fetch에서 통지에 대한 데이터 확보 비율을 나타낸다.

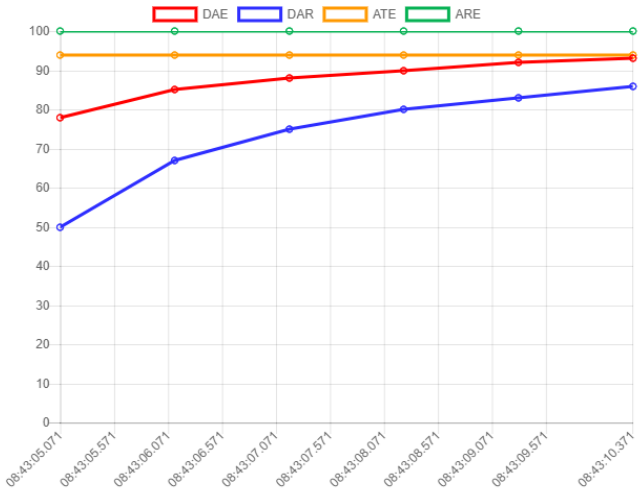


그림 6. 획득 효율성 메트릭 값의 가시화 그래프 컴포넌트

그림 7. 획득 효율성 메트릭 값의 가시화 텍스트 컴포넌트

3.5. 콘솔의 가시화

콘솔이란 화면의 중앙 하단에 위치한 검은색 박스형태의 UI 컴포넌트로서 실시간으로 플랫폼에서의 여러 변화들을 문자열로 출력해주는 기능을 한다. 사용자는 이를 통해 플랫폼 내부의 변화들을 곧바로 인지할 수 있고, 스크롤을 올려서 이전의 변화들을 다시 확인 할 수 있다.

콘솔에서 보여주는 정보들은 사용자에게 중요하기 때문에 이를 효과적으로 확인할 수 있도록 콘솔의 위치와 콘솔의 색을 정하였다. 콘솔의 위치를 중앙 하단에 위치시킴으로써 중앙에 위치한 그래프의 변화를 확인하면서 동시에 플랫폼 내부에서 일어나는 변화들을 중앙 하단의 콘솔에서 확인 할 수 있다. 이러한 콘솔의 배치를 통해 한 눈에 중요 정보들을 실시간으로 확인할 수 있다는 장점이 있다. 또한 UI에서 보여주는 다른 정보들과는 달리, 콘솔의 글씨와 배경 색깔을 반전시킴으로써(검은색 바탕에 흰색 글씨) 플랫폼 내부의 변화들이 발생 했을 때 더 눈에 띄게 하는 효과가 있다.

콘솔에서 보여주는 플랫폼 내부의 변화에 대한 정보는 여러 종류가 있다. : 데이터 획득 방식의 변화, 데이터 획득 메트릭 값의 변화, 데이터 소스의 측정 변수 값의 변화, 데이터 싱크의 획득 변수 값의 변화, 플랫폼의 작동 상태(Run, Stop, Pause). 예를 들어, 사용자가 데이터 소스의 측정변수인 SRC.Interval 을 1000에서 100으로 변경했을 경우, 변경 시간, 변경된 소스의 이름, 그리고 값의 변화를 아래의 그림과 같이 출력해준다. 발생 시점을 문자열의 처음에 위치시켜서

사용자가 변화의 시점을 인지할 수 있게 하였다. 또한 시간 뒤에 변화의 주체가 어디인지를 표시하고, 화살표 뒤에 변화의 내용에 대해 표시하였다.

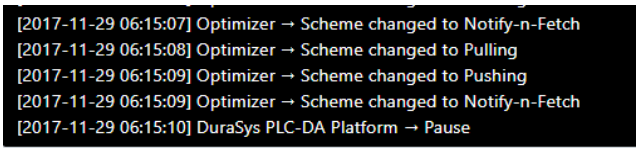


그림 8. 콘솔 가시화 컴포넌트

4. 콘솔의 정보 전달 최적화 설계

데이터 획득 효율성 메트릭 값을 실시간으로 사용자에게 보여주기 위하여, 웹소켓을 이용하여 메트릭 값을 실시간으로 받고 이를 콘솔에 나타내도록 설계하였다.

5. 사례 연구: 웹 기반 획득 데이터의 가시화 구현

가시화 계층 설계를 바탕으로 웹 기반의 획득 데이터 가시화를 구현하였다. 데이터 소스와 싱크에 대한 정보를 화면의 측면에 배치하였다. 그리고 데이터 소스 데이터를 실시간으로 보여주는 그래프와 데이터 획득 효율성을 나타내는 그래프를 중앙에 위치하게 하였다. 그래프 아랫부분에 플랫폼의 작동 시간과 가장 최근의 데이터 획득 효율성을 표시해주는 컴포넌트를 위치시켰다. 그리고 콘솔을 중앙 하단에 위치시켰다.

데이터의 변화 추이가 중요한 정보인 센서 데이터와 데이터 획득 효율성을 그래프로 보여줌으로써 사용자가 실시간으로 데이터 변화 추이를 인지하기 쉽게 하였고, 중요도가 높은 정보들은 중앙에 오도록 위치시키고 상대적으로 덜 중요한 정보들은 중앙에서 멀어지도록 배치시켰다. 그리고 소스에 관련된 정보, 싱크에 관련된 정보를 관련되지 않은 정보와 따로 분리하여 시스템 사용방법을 파악하기 쉽게 하였다.

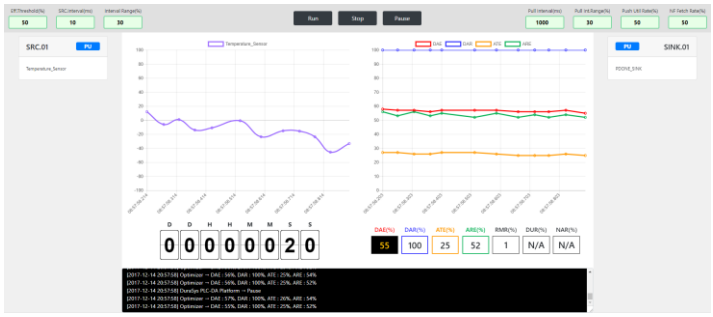


그림 9. 웹 기반 획득 데이터의 가시화 구현 화면

6. 결론

SoC(Separation of Concern)의 원리를 이용하여 정보의 관련도에 따라 사용자에게 보여주는 정보의 배치를 결정하였다. 즉 서로 관련도가 낮은 정보들은 서로 분리하고 관련도가 높은 정보들을 가깝게 위치시켰다. 또한 주요한 정보는 중앙에 배치시키고, 센서 데이터와 데이터 획득 효율성 같이 실시간 변화 추이가 중요한 정보는 그래프로 표현하고 이외의 정보들은 텍스트로 보여주는 방식을 채택하였다. 이를 웹으로 구현하였으며, 웹소켓을 이용하여 실시간으로 소스 데이터와 획득 효율성을 서버로부터 받아 올 수 있도록 설계하였다. 이를 통해 본 플랫폼의 획득 데이터의 효과적인 가시화와 실시간 데이터의 출력을 가능하게 함으로써 본 플랫폼에 대한 User Experience를 향상시키고 사용자에게 높은 Usablity를 제공할 수 있었다.

7. Acknowledgement

이 논문은 2017년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행한 연구임(NRF-2015R1A2A2A01004078).

8. 참고문헌

[1] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, ‘‘Data mining with big data,’’ IEEE Trans. Knowl. Data Eng., vol. 26, no. 1, pp. 97–107, Jan. 2013.

[2] H. Harb and A. Makhoul, "Energy Efficient Sensor Data Collection Approach for Industrial Process Monitoring," in IEEE Transactions on Industrial Informatics, vol. PP, no. 99, pp. 1-1.

[3] J. Yan, Y. Meng, L. Lu and L. Li, "Industrial Big Data in an Industry 4.0 Environment: Challenges, Schemes, and Applications for Predictive Maintenance," in IEEE Access, vol. 5, pp. 23484-23491, 2017.

[4] T. Yu, X. Wang and A. Shami, "Recursive Principal Component Analysis-Based Data Outlier Detection and Sensor Data Aggregation in IoT Systems," in IEEE Internet of Things Journal, vol. 4, no. 6, pp. 2207-2216, Dec. 2017.

[5] H. Sándor, B. Genge and Z. Szántó, "Sensor data validation and abnormal behavior detection in the Internet of Things," 2017 16th RoEduNet Conference: Networking in Education and Research (RoEduNet), Tg. Mures, Romania, 2017, pp. 1-5.

[6] R. Yordanov, R. Miletiev, P. Kapanakov and E. Lontchev, "Design of a portable system for sensor data acquisition and transmission," 2017 XXVI International Scientific Conference Electronics (ET), Sozopol, Bulgaria, 2017, pp. 1-3.

[7] A. Celesti, A. Galletta, L. Carnevale, M. Fazio, A. Ĺay-Ekuakille and M. Villari, "An IoT Cloud System for Traffic Monitoring and Vehicular Accidents Prevention Based on Mobile Sensor Data Processing," in IEEE Sensors Journal, vol. PP, no. 99, pp. 1-1.

다변량 대량 데이터 내의 소량의 이상 데이터 원인 규명 및 학습/추론 방법

이제희 ○, 이상형*, 김동일
한국생산기술연구원
rjchart@kitech.re.kr, zelog@kitech.re.kr, dikim01@kitech.re.kr

Identifying the Cause of a Small Amount of Abnormal Data in Multivariate Large Data and Its Uses for Learning/Reasoning

Jaehee Rhee○, Sang Hyoung Lee*, and Dong-Il Kim
Korea Institute of Industrial Technology
* Corresponding Author

요 약

본 논문에서는 다변량(Multivariate)의 대량 데이터에 포함된 소량의 이상 데이터의 불량을 야기시키는 인자를 규명하고, 이를 학습/추론하는 방법을 제안하고자 한다. 여기서, 정상 데이터와 이상 데이터는 단변량 비교 시 유사 구간에 서로 혼재되어 분포되어 있다. 본 연구에서는 이를 해결하기 위한 다음의 두 가지 확률 분석 기법들을 제안한다: 1. 확률 구간 내 정상/이상 데이터 비율 분석, 2. 단변량(Univariate) 인자 우도 분석. 이 방법들을 검증하기 위해, 실제 자동차 부품 생성 공정에서 취득한 데이터를 활용하여, 기존의 의사 결정 트리 모델링 기법과 비교한다.

1. 서 론

최근 다양한 제조 분야에서 제품의 질과 공정의 효율성을 향상시키기 위한 스마트화가 진행되고 있다. 공정 프로세스를 파악하기 위해 다양한 종류의 센서를 부착하여 대량의 데이터를 수집하지만 이러한 대량의 데이터로부터 공정 프로세스의 이상여부를 판별하는 것은 여전히 쉬운 일이 아니다. 그 이유는 다음과 같다: 1. 기업은 공정 프로세스를 가능한 범위에서 안정화시켜 놓기 때문에 불량을 야기시키는 이상 데이터가 극히 드물게 발생한다. 2. 이상 데이터는 단변량 비교 시 정상 데이터와 유사 구간 내에서 혼재되어 있다. 다시 말해, 정상 데이터와 이상 데이터를 단변량 비교 시 동일한 값을 가질 수 있다. 본 논문은 이러한 다변량의 대량 데이터에서 소량의 이상 데이터의 원인을 규명하기 위한 두 가지 접근 방법을 제안한다. 이 두 가지 접근 방법은 기존의 의사 결정 트리 모델링 기법과 그 성능을 비교 분석한다. [1][2][3]

2. 본 론

1장에서 언급한 것처럼 우리가 다루고자 하는 다변량의 대량 데이터는 정상 데이터와 이상 데이터의 단변량 비교 시 유사 구간에서 혼재되어 분포한다. 혼재하는 데이터는 정확하게 어떤 변수를 통해 정상 데이터와 이상 데이터를 나눌 수 없기 때문에 변수의 특정 범위에서 이상 데이터의 비율이 높아지는 구간이 있는지를 변수별로 파악할 수 밖에 없다. 그러므로, 우리는 먼저 정상 데이터와 이상 데이터를 포함하는 전체 데이터의 유사 구간을 구분하기 위해 확률적으로 구간을 분할한다. 그리고 분할된 구간 내에서 정상 데이터와 이상 데이터의 비율을 비교 분석한다. 모든 변수와 비교하여 가장 특정 구간에 이상 데이터의 비율이 많은 데이터는 이상 데이터의 원인으로 유추할 수 있다. 그림 1은 확률 구간을 나누고, 확률 구간 내에서 정상/이상 데이터의 비율을 분석하는 알고리즘을 보여준다.

1. 다변량 대량 데이터의 각각의 인자(변수)에 대해서 확률 구간을 결정하기 위해 개별적으로 K-평균 클러스터링 (K-means clustering)[4]을 수행한다.

2. 각 인자에 따라 최적 클러스터의 수가 다를 수 있기 때문에 클러스터의 수를 변화시켜가면서 베이지안 정보 기준 (Bayesian Information Criterion; BIC)를 통해 클러스터의 수를 결정한다.

3. 각 클러스터에 속한 정상 데이터와 이상 데이터의 비율을 계산한다.

$$\alpha = \frac{\text{정상 데이터}}{\text{정상 데이터+ 이상 데이터}},\ \beta = \frac{\text{이상 데이터}}{\text{정상 데이터+ 이상 데이터}}$$

4. 각 클러스터마다 β가 α보다 일정 이상 큰 인자만을 불량 원인 인자로 선정한다.

$$\alpha < \sigma\beta$$

5. 선정된 인자와 그것들의 이상 데이터만을 이용하여 가우시안 혼합 모델[5][6]을 생성한다.

그림 1. 확률 구간 생성 및 확률 구간 내 정상/이상 데이터

비율 분석을 통한 이상 데이터 원인 규명 알고리즘

그림 1에서처럼 전체 데이터의 값을 K개의 구간으로 구분하기 위해 다변량 대량 데이터의 각각의 인자에 대해서 개별적으로 K-평균 클러스터링을 수행한다. 이 때, K의 개수는 베이지안 정보 기준을 통해 결정한다. 각 인자는 K-평균 클러스터에 의해 군집화 된 개수만큼의 확률 구간을 갖는다. 다음으로 각 인자의 클러스터에 속한 정상 데이터와 이상 데이터의 비율을 계산한다. 여기서, 이상 데이터의 비율 β가 정상 데이터의 비율 α보다 일정 이상 클 경우 이 인자를 이상 데이터 유발 인자로 간주한다. 이상 데이터 유발 인자를 분류하기 위한 σ값은 데이터에 의존적이다. 그렇기 때문에 σ값을 특정하기보다 각 인자의 구간들을 비교하여 이상 데이터의 비율이 특정 구간에서 가장 높은 경우가 가장 이상 데이터와 연관성이 높다고 할 수 있기에 비율이 높은 인자들을 선정한다. 마지막으로 선정된 인자와 그것의 이상 데이터만을 이용하여 가우시안 혼합 모델을 모델링한다. 이는 추후 이상 데이터를 분류하기 위한 모델로 사용될 수 있다.

다음으로 이상 데이터의 유발 인자를 선정하기 위해 그림 2와 같이 단변량 우도 분석을 수행한다.

1. 각 인자(변수)별로 정상 데이터와 이상 데이터를 분류한다.
2. 각 인자별로 이상 데이터를 위한 가우시안 혼합 모델을 생성한다.
3. 가우시안 혼합 모델을 생성하는 과정에서 가우시안의 개수는 BIC에 의해 결정한다.
4. 이상 데이터를 위한 가우시안 혼합 모델에 정상 데이터와 이상 데이터와 정상 데이터를 넣고 로그-우도(Log-likelihood)를 계산한다.
5. 이상 데이터와 정상 데이터의 로그-우도의 차이가 큰 변수를 이상 데이터 유발 인자로 선정한다.
6. 선택된 인자만을 이용하여 비정상 데이터를 위한 다변량 가우시안 혼합 모델을 모델링한다.

그림 2 단변량 인자의 로그-우도 분석을 통한 이상 데이터 원인 규명 알고리즘

그림 2에서처럼 단변량 우도 분석을 통한 이상 데이터 원인 규명을 위해 먼저 각 인자별로 정상 데이터와 이상 데이터를 분류한다. 각 인자별로 이상 데이터만을 이용하여 가우시안 혼합 모델을 생성한다. 이 때, 가우시안 혼합 모델의 가우시안의 개수는 각 인자마다 차이가 있을 수 있기 때문에 BIC를 비교하여 적합한 개수를 결정한다. 만들어진 각 인자의 가우시안 혼합 모델은 만약 인자가 이상 데이터와 연관이 없다면, 이상 데이터와 정상 데이터의 분포가 큰 차이가 없을 것이다. 만약 둘의 데이터 분포에 차이가 있다면 그 인자는 이상 데이터와 연관이 있다고 볼 수 있다. 이를 이용하여 만들어진 가우시안 혼합 모델과 데이터간의 로그 우도의 차이를 비교하여 그 차이의 평균이 가장 큰 인자를 찾아 마찬가지로 가우시안 혼합 모델을 모델링한다.

3. 실 험

제안하는 알고리즘을 검증하기 위해 본 논문에서는 실제 자동차 생산 공정에서 취득한 데이터를 활용하였다. 날씨, 온도, 습도 등의 12,929개의 인자(변수)를 가지고 있는 총 14,847개의 데이터가 사용되었다. 12,929개의 인자는 각각 생산 공정에서 필요로 하는 데이터를 공장 측에서 뽑아준 데이터로 수많은 공정이 존재하기에 만개가 넘는 인자가 존재한다. 전체 데이터 중 총 3종류의 이상 데이터가 각각 10개, 9개, 6개 존재한다. 이상 데이터의 비율은 각각 0.1% 미만의 비율을 보인다.

표 1. 14,847개 데이터의 변수 정보

	Variable Name
1	SPI_COUNT
2	SPI_RESULT_GOOD
3	SPI_RESULT_E.Exessive
4	SPI_RESULT_E.Insuffi
...	
12927	IS_SUNNY
12928	IS_RAIN
12929	IS_CLOUD

모든 알고리즘에서 사용되는 데이터는 먼저 전처리 과정을 수행한다. 전처리 과정은 노이즈 제거와 정규화(Normalization)작업으로 구성된다. 센서로부터 받는 데이터들은 노이즈를 포함하고 있기 때문에 그 노이즈들을 그대로 학습에 사용하면 잘못된 데이터를 학습할 수 있기 때문에 노이즈를 제거하는 과정이 필요하다. 본 연구는 Median Filter를 통해 기존 데이터를 순화된 데이터로 바꾸고, 순화된 데이터를 기존 데이터와 비교하여 그 차이가 심한 값을 노이즈로 구분하여 제거한다. 또한 모든 데이터들을 동일한 기준으로 비교하기 위해 정규화 작업을 수행한다. 정규화 작업은 각 데이터의 최대값을 1, 최소값을 0으로 바꾸고 모든 데이터를 그 기준에 맞게 0~1 사이의 값으로 변경하는 작업이다.

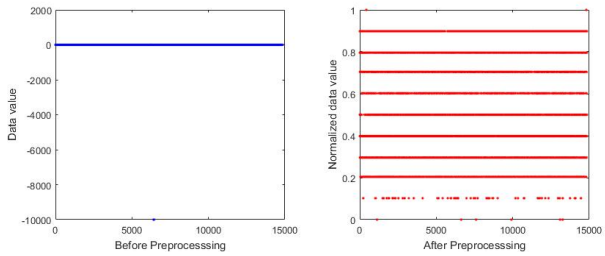


그림 3 노이즈가 존재하는 데이터의 노이즈를 제거하고 정규화한 데이터를 비교

그림 3은 전처리 전과 후의 데이터를 비교하였으며 이전에는 데이터에 큰 노이즈들이 존재하였지만 노이즈를 제거하고 정규화 작업을 거쳐서 받아들인 값의 분포를 제대로 그래프로 나타난다.

본 연구는 학습을 위해 80%의 데이터인 11,877개의 데이터를 사용하였고, 그 중에 이상 데이터를 종류 별로 각각 8개, 7개, 4개 포함시켰다. 그리고 학습된 결과를 테스트하기 위해서 20%의 데이터인 2970개의 데이터로 테스트하여 결과를 분석하였다. 실험 결과는 3개의 이상 데이터 중에 첫번째 이상 데이터(#556)에 대한 결과만을 정리하고 나머지는 표 2에 정리하였다.

확률 구간 내 정상/이상 데이터 비율 분석의 학습을 통해서 구해진 이상 데이터를 야기하는 인자는 ICT_1-68_VALUE (#11524) 외 9개의 변수들이다. 그 과정 내에 적용된 클러스터링의 수, K값과 α값, β값은 각 인자마다 다른데, 대체로 K값은 2~4 사이의 값이 되었으며, β값은 5~8 x 10^{-3} 의 값들을 가지고 α값은 1 - 5~8 x 10^{-3} 의 값을 가진다. 구해진 10개의 인자를 통해 만든 가우시안 혼합 모델로 테스트 데이터를 분석한 결과 99%의 정확도를 보였으며 모든 분석 오류는 False-Negative 데이터이다. True-Negative는 10개 중 10개 모두 제대로 분류한다.

단변량 인자 로그-우도 분석은 이상 데이터의 가우시안 혼합 모델을 위해 1개의 혼합 모델을 사용하였다. 그리고 그 모델과 전체 데이터의 학습을 통해서 구해진 이상 데이터를 야기하는 인자는 정상/이상 데이터 비율 분석 역시 ICT_168_VALUE (#11524) 외 9개의 변수들이다. 가장 연관성이 높은 변수인 ICT_168_VALUE는 두 개의 분석이 동일하지만 그 외 변수들은 다른 값을 가지고 있다. 구해진 10개의 인자를 통해 만든 가우시안 혼합 모델 모델로 테스트 데이터를 분석한 결과 99.3%의 정확도를 보였으며 모든 분석 오류는 False-Negative 데이터이다. True-Negative는 10개 중 10개 모두 제대로 분류한다.

표 2. 제안된 알고리즘과 의사 결정 트리의 정확도 비교

확률 구간 내 정상/이상 데이터 비율 분석			
이상데이터 종류	#556	#487	#187
선택된 인자	#11524, #11526 외 8개	#12904, #11163 외 8개	#11524, #11526 외 8개
클러스터링(K) 개수	2~4개	6~7개	2~4개
β값	5~8 x 10^{-3}	5~8 x 10^{-4}	3~5 x 10^{-3}
Accuracy	99%	65.2%	99.3%
Precision	100%	100%	100%
True-Negative	10/10 (100%)	9/9 (100%)	6/6 (100%)
단변량 인자 로그-우도 분석			
이상데이터 종류	#556	#487	#187
선택된 인자	#11524, #11526 외 8개	#11921, #11920 외 8개	#11524, #11526 외 8개
Accuracy	99.3%	90.2%	99.3%
Precision	100%	100%	100%
True-Negative	10/10 (100%)	9/9 (100%)	6/6 (100%)
CART 의사 결정 트리 모델링 분석			
이상데이터 종류	#556	#487	#187
선택된 인자	#3030, #11526, #62외 1개	#5464, #11749외 7개	#10154, #11184 외 4개
Accuracy	97.4%	92.2%	91%
Precision	99%	99%	99%
True-Negative	8/10 (80%)	3/9 (33%)	4/6 (83.3%)

표 2의 데이터는 제안한 두 개의 알고리즘과 의사결정 트리를 학습시켜 비교한 결과이다. 여기서 확률 구간 내 정상/이상 데이터 비율 분석 방식과 단변량 인자 우도 분석 방식은 의사결정 트리와 비교하여 다소 정확도는 떨어진다. 하지만 True-Negative Ratio의 정확도는 의사결정 트리 모델링에 비해 훨씬 좋은 결과를 보인다. #487의 데이터에서 CART 의사결정 트리를 사용한 경우 전체 데이터에 대한 정확도는 92.2%로 단변량 인자 로그-우도 분석보다 조금 더 정확하지만, True-Negative Ratio 비율은 33%로 거의 이상 데이터를 분류하지 못한다. 이것은 9개의 이상 데이터 중 3개만을 이상 데이터로 분류한다는 의미다. 이는 실제 데이터 중 이상 데이터를 제대로 구분해내지 못할 뿐 아니라 이상 데이터를 야기하는 요소도 제대로 구하지 못한다는 의미이다. 그에 비해, 본 연구에서 제공하는 두 방식은 정확도는 조금 떨어지지만 True Negative 데이터의 정확도가 높다. 이는 알고리즘이 지정한 범위의 데이터 내에 이상 데이터의 분포가 다른 분포에 비해 월등히 높아, 선택된 인자들이 이상 데이터와 연관성이 매우 높다는 것을 보여준다.

이는 의사 결정 트리의 원인 분석 방식은 인자별로 엔트로피를 비교하여 데이터의 이상 데이터와 정상 데이터가 해당 인자로 인해 정확히 나누는 방식이기 때문에 데이터의 이상 데이터와 정상 데이터가 특정 구간에서 혼재하는 경우에는 바르게 구분할 수 없기 때문에 나타나는 결과이다. 본 연구는 이런 혼재 데이터의 구분에 특화시켜서 True-Positive는 정확히 구분 못하더라도 이상 데이터에 중점을 두어 True-Negative를 확실히 찾아내어 이상 데이터의 원인을 찾는 것에 중점을 두었다.

4. 결 론

본 연구가 제안한 방법은 다변량의 데이터 내에 소량의 혼재된 이상 데이터의 원인을 찾는 것에 있어서, 기존에 사용되던 의사결정 트리의 원인 인자를 구하는 방법보다 데이터와의 연관성을 잘 찾아낼 수 있음을 보인다. 또한 본 연구에서 실제 분석 결과를 비교하여 볼 때, 이상 데이터가 극히 적은 데이터에서는 분석된 데이터의 Accuracy만을 목표로 학습하는 것만으로는 원인을 제대로 파악할 수 없다는 것을 알 수 있다. 이상 데이터가 적은 특이성을 가진 데이터에서는 그에 맞는 분석이 필요한데, True-Negative Ratio는 이상 데이터와의 연관성을 보기에 적합하다는 것을 알 수 있다.

5. 향후 계획

본 연구는 다변량의 소량 데이터를 통한 데이터의 분석은 그에 맞는 데이터 분석이 중요하다는 것을 확인하였다. 그러므로 이후 다변량의 소량 데이터에 높은 성능을 내는 LASSO Regression를 통해 분석을 해볼 예정이며, 많은 데이터가 더 주어진다면 딥러닝의 기술을 적용하여 추가적인 연구를 해볼 계획이다.

감사의 글

본 논문은 기관주요사업 중 글로벌진출지원 사업인 “출하 후 품질문제 대응을 위한 제조정보 추적관리 및 가시화 서비스 시스템 (PKM16J80)”의 지원으로 수행한 연구과제입니다.

참고문헌

[1] Van Tung Tran, Machine Fault Diagnosis and Condition Prognosis using Adaptive Neuro-Fuzzy Inference System and Classification and Regression Trees, 한국전자통신학회, 2011 (기계 결함진단 및 예지를 위한 ANFIS와 CART)

[2] Wang, Y. Y., Li, J., Feature-selection ability of the decision-tree algorithm and the impact of feature-selection/extraction on decision-tree results based on hyperspectral data, International journal of remote sensing, 2008

[3] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and regression trees, Chapman & Hall, 1984

[4] Jonghwan Heo, Byeongchae Lim, Feature Selection and k-means for Musical Preference Analysis. 한국정보과학회, 2015

[5] Sun-Kuk Moon, Tack-Sung Choi, Feature Selection for Multi-Class Genre Classification using Gaussian Mixture Model, The Journal of the KICS, 2007

[6] Le, Sy Tai, Singing voice detection in audio music using Gaussian Mixture Model, 2012

군집 비행을 위한 무인비행기 네비게이터 시스템 설계

이성희°, 김덕엽, 윤보람, 이우진
경북대학교 컴퓨터학부, 소프트웨어기술연구소
lee3229910@gmail.com, ejrduq77@naver.com, yunstar1105@naver.com,
woojin@knu.ac.kr

Design of UAV Navigator System for Formation Flight

Sunghee Lee°, Deok Yeop Kim, Boram Yun, Woo Jin Lee
School of CSE & SWRC, Kyungpook National University

요 약

군집 비행에는 세 가지 주요 이슈점이 있다. 그 이슈들 중 하나는 무인 항공기 간의 통신이다. 무인비행기 간의 통신은 군집 비행의 정확성과 정밀도를 결정하는 중요한 요소이기 때문이다. 기존 무인비행기 간의 통신은 주로 리더 무인비행기가 브로드캐스트하여 추종자 무인비행기들을 제어하고 리더 무인비행기는 지상 제어 시스템에서 제어하는 방식으로 수행되었다. 그러나 이 방식은 근본적으로 리더 무인비행기와 지상 제어 시스템 간의 통신 지연이 발생할 경우 통신이 지연되는 동안 제어가 수행되지 않아 추종자 무인비행기가 다른 무인비행기와 충돌 할 위험이 있다. 따라서 본 논문에서는 이러한 문제를 해결하기 위한 UNS (UAV Navigator System)를 제안한다.

1. 서론

최근 무인비행기 관련 기술들이 활발히 개발되면서 군집 비행의 규모가 점점 커지고 군집 비행을 위한 제어도 복잡해지고 있다. 이러한 군집 비행의 많은 문제들 중에 세 가지의 주요 이슈가 있다. 첫 번째는 군집에서의 무인비행기들의 정확한 위치 파악을 위한 위치 인식 이슈이고 두 번째는 군집 비행 중 필요한 정보들을 주고받기 위한 무인비행기들 간의 통신 이슈이며 마지막은 충돌이나 사고 없이 군집 비행을 정상적으로 수행하기 위한 무인비행기의 비행 제어 이슈다. 이들 중 특히 무인비행기 간의 통신은 다른 요소들에 의해 지연 문제가 발생할 수 때문에 군집 비행의 정확성과 정밀도를 결정하는 중요한 요소다.

기존의 무인비행기 간의 통신 방식은 주로 리더 무인비행기가 브로드캐스트하여 추종자 무인비행기들을 제어하고 리더 무인비행기는 지상제어시스템에서 제어하는 방식으로 수행되었다. 이 통신 방식은 크게 두 부분으로 나뉜다. 하나는 지상제어시스템으로부터 받은 제어 명령을 리더 무인비행기가 추종자 무인비행기들로 브로드캐스트하는 비행 제어 부분이다. 다른 하나는 무인비행기들이 서로 충돌하도록 하지 않기 위해 지속적으로 브로드캐스트되는 무인비행기들의 위치 정보를 수집하는 부분이다.

이러한 기존의 통신 방식은 근본적으로 리더 무인비행기와 지상제어시스템 간의 통신 지연이 발생할 경우 오직 리더 무인비행기와만 통신하는 추종자 무인비행기들은 서로의 위치 정보는 알 수 없기 때문에 다른 무인비행기와 충돌 할 위험이 있다[1-4]. 따라서 우리는 비행 중에 지상제어시스템과 무인비행기 간의 통신 실패나 지연으로 인한 비상사태에 대응할 수 있는 시스템인 UNS를 제안한다.

2. 관련연구

2.1 중앙 집중식 제어

중앙 집중식 제어는 군집 비행에 일반적으로 사용되는 제어 방법이다. 군집 비행을 유지하기 위해 필요한 제어 명령은 그림 1(a)와 같이 리더 무인비행기를 기반으로 하여 계산되며, 계산된 제어 명령은 지상제어시스템 또는 리더 무인비행기에 의해 수행된다. 따라서 군집 비행을 위해서는 각 무인비행기들의 위치 및 자세 정보가 지속적으로 필요하기 때문에 지상제어시스템과의 통신이 필요하다. 그런데 중앙 집중식 제어 방법에는 몇 가지 문제가 있다[1-6]. 가장 위험한 문제는 지상제어시스템과 리더 무인기 간의 통신이 끊어지거나 지연되는 것이다. 통신 두절이나 지연이 발생하면 제어가 실시간으로 이루어지지 않아 위험한 사고로 이어질 가능성이 높다.

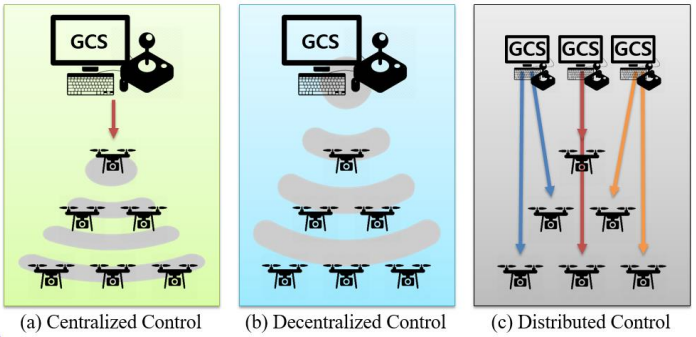


그림 1. 군집비행 시 무인비행기들 간 통신 제어 방법

2.2 비중앙 집중식 제어와 분산 제어

비중앙 집중식 제어는 그림 1(b)와 같이 무인비행기에 일정 수준의 의사 결정권을 부여하는 방법이다[5-6]. 이 방법은 일반적으로 GPS 정보를 사용하여 군집 비행을 수행 할 수 있는 야외에서 사용된다. 그러나 무인비행기에 일정 수준의 의사 결정권을 부여하기 때문에 각 무인비행기에는 구현하기 어려운 복잡한 알고리즘이 필요하다.

분산 제어는 그림 1(c)와 같이 중앙 집중식 제어 및 비중앙 집중식 제어 문제를 극복하기 위해 고안된 방법이다. 이 접근 방법에서는 하나의 지상제어시스템이 모든 무인비행기를 관리하지 않고 여러 지상제어시스템 또는 임무제어시스템에 제어 부담을 분산시킨다. 그러나 지상제어시스템이나 임무제어시스템 간의 정보 공유가 필요하기 때문에 네트워크의 규모와 깊이가 증가하고 통신 지연도 발생 할 수 있다[1].

3. 무인비행기 네비게이터 시스템의 구조

UNS는 군집 비행을 위해서 다섯 가지 구성 요소를 가지고 있다. 구성 요소들은 긴급사태 매니저, 위치 인식기, 제어 명령기, 편대 가이드 및 충돌 회피기이다. 긴급사태 매니저는 지상제어시스템과 UNS 간의 연결이 끊어지거나 UNS가 리더 무인 비행기와 연결되지 않은 경우 RTL(Return To Launch) 명령을 사용하여 모든 무인 비행기를 안전하게 복귀시킬 수 있는 무엇보다도 중요한 구성 요소이다. 동시에 충돌 회피기 및 제어 명령기는 모든 무인 비행기가 군집 비행을 유지하면서 충돌이나 사고 없이 복귀지점으로 복귀하도록 할 수 있다. 위치 인식기 및 편대 가이드는 기존 방법에서의 리더 무인기의 역할을 대체한다.

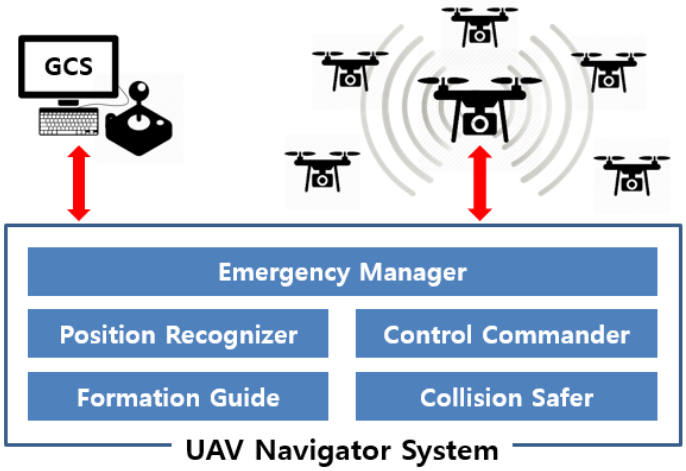


그림 2. 무인항공기 네비게이터 시스템의 구조

그림 2에서 볼 수 있듯이 UNS는 임무제어시스템 또는 중간 지상제어시스템으로 볼 수 있지만 UNS는 하늘에서 비행하는 무인 비행기와 동일한 네트워크에서 실시간으로 통신하여 장애나 문제 발생 시 신속하게 대응할 수 있기 때문에 통신 문제를 고려해야하는 임무제어시스템이나 지상제어시스템과는 근본적으로 다르다.

4. 실험결과

우리는 지상제어시스템으로 qgroundcontrol[7] v3.2.3와 MAVLINK[9] v2.0 통신 프로토콜을 사용하는 무인비행기 PIXHAWK[8] (PX4 FMU V2)를 이용하여 실험하였고 PIXHAWK에 탑재된 센서와 GPS외에 추가 센서를 추가하지 않았다. PIXHAWK가 무인 비행기의 리더라고 가정하였다.

먼저 무인 비행기와 지상제어시스템 사이의 1분 동안 텔레메트리 통신 지연을 측정하였다. 그 다음에 동일 시간 동안 57,600 BAUDRATE의 UART 통신을 하는 무인 비행기와 UNS 사이의 통신 지연을 측정하였다. 1분 동안 무인 비행기는 약 1,850 개의 MAVLINK 메시지를 보냈다. 표 1은 메시지 중에서 상위 5개 메시지에 대한 정보를 나타내며 표 2는 무인 비행기에서 받는 작업에 대한 MAVLINK 메시지의 지연을 보여준다. 결론적으로 약 20~30 바이트 크기의 무인 비행기 메시지의 약 70%가 무인 비행기의 제어를 위한 정보이므로 우리의 사이클을 이용한 UNS를 적용하면 지연을 줄이고 군집 비행에 대해 보다 민감한 UAV 제어를 가능하게 한다.

표 1. 무인비행기로부터 UNS로 전달되는 MAVLINK 메시지 상위 5개

ID	Message name	Length	Proportion	Count
30	attitude	28 bytes	49.2%	912
74	vfr_hud	18 bytes	9.8%	181
22	param_value	25 bytes	6.0%	110
83	attitude_target	32 bytes	4.9%	91
0	heartbeat	9 bytes	3.9%	72

표 2. 무인비행기에서 대상시스템으로 전달되는 메시지 정보

Target	Cycle of receiver task	About MAVLINK messages		
		Transfer time	Total length	Length of per a cycle
UNS	1us	880us	61,211 bytes	1 byte
	100us	1,002 us	65,220 bytes	1 byte
	1,000us	5,926 us	65,785 bytes	6 bytes
	10,000us	32,653 us	64,872 bytes	38 bytes
	our cycle	20,405 us	68,831 bytes	23 bytes
qground control	default	38,966 us	49,947 bytes	62 bytes

5. 결론

본 논문에서는 군집 비행에서 지상제어시스템과 무인비행기간의 통신 지연 문제를 해결하기 위해 UNS를 제안한다. 기존의 방법들은 통신 지연 또는 통신 실패로 인해 예측할 수 없는 상황이 발생하면 신속하게 대처할 수 없다. 그러나 UNS를 사용하면 이러한 문제에 유연하게 대처할 수 있다. 또한 군집비행에서 UNS는 지상제어시스템과 리더 무인비행기의 운영 부담을 줄여 보다 능동적인 비행을 수행할 수 있다.

※ 본 논문은 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행하였으며(No. NRF-2017R1D1A3B04035880) 또한 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보・컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No. 2017M3C4A7066010).

참고문헌

[1] Y. Ding, C. Wei and S. Bao, “Decentralized formation control for multiple UAVs based on leader-following consensus with time-varying delays,” *Proc. Chinese Automation Congress (CAC)*, pp. 426-431, Nov. 2013.
[2] W. Yuan, Q. Y. Chen, Z. X Hou and Y. Li, “Multi-UAVs formation flight control based on leader-follower pattern,” *Proc. Chinese Control Conference (CCC)*, pp. 1276-1281, Jul. 2017.
[3] D. A. Mercado, R. Castro and R. Lozano, “Quadrotors flight formation control using a leader-follower approach,” *Proc. European Control Conference (ECC)*, pp. 3858-3863, Jul. 2013.
[4] L. Pollini, F. Giulietti and M. Innocenti, “Robustness to communication failure within formation flight,” *Proc. American Control Conference*, Vol. 4, pp. 2860-2866, May. 2002.
[5] B. H. Lim, J. W. Kim, S. W. Ha and Y. H. Moon, “Development of software platform for monitoring of multiple small UAVs,” *IEEE/AIAA Digital Avionics Systems Conference (DASC)*, pp. 1-5, Sep. 2016.
[6] A. T. Angonese and P. F. F. Rosa, “Ground Control Station for Multiple UAVs Flight Simulation,” *Latin American Robotics Symposium and Competition (LARS/LARC)*, pp. 136-141, Oct. 2013.
[7] qgroundcontrol. http://qgroundcontrol.com
[8] PIXHAWK. https://pixhawk.org
[9] MAVLINK. http://qgroundcontrol.org/mavlink/

인공지능 SW 오작동 유형 분류·분석 및 개발 시 고려사항 연구

김상기 ○, 정종하, 강지성, 강건희, 권하은, 김정현, 김주리, 이상민, 공준익, 정성룡
한국정보통신기술협회 소프트웨어시험인증연구소
{sangki.kim, jhjung, kang.jiseong, lustiness_hee, khe0402,
kimjh, kim.juli, minuri33, joonik, srjeong}@tta.or.kr

A Study on SW Defect Analysis of Artificial Intelligence, Considerations in SW Development

Sangki Kim○, Jongha Jung, Jiseong Kang, Geon-Hee Kang, Haeun Kwon,
Junghyun Kim, Juli Kim, Sangmin Lee, Joonik Kong, Seongryong Jung
Institute of Software Testing & Certification,
Telecommunications Technology Association

요 약

본 연구에서는 인공지능 SW에서 발생하는 오작동에 대한 유형 분류 및 분석을 통하여 심각한 사고 및 오류를 예방하기 위해 필요한 SW 안전 기능에 대해 소개한다. 그리고 인공지능 SW개발 시 고려해야할 개발 요소를 학습데이터의 활용 관점에서 개발 단계와 서비스 단계에서 구분하여 이를 적용하기 위한 인공지능 SW 검증 모델을 제시한다. 본 연구에서 제시한 다양한 소프트웨어 공학 기술이 적용된 검증 모델을 통하여 보다 안전한 지능정보사회 구현을 할 수 있다.

1. 연구 배경

지능정보기술의 발전과 함께 실생활에서 인공지능 SW 활용이 증대됨에 따라 이로 인한 발생할 수 있는 위험 또는 안전사고 예방을 위한 정책적인 노력이 최근 강하게 요구되고 있다[1]. 또한 정부에서 추진하고 있는 지능정보사회 중장기 종합대책으로 사회 분야의 인공지능SW 오작동과 관련된 인공지능 SW의 안전성 확보를 위한 인공지능 SW 역기능 방지에 선제적 대응을 위한 과제를 추진 중에 있다[2]. 인공지능 SW가 탑재된 자율주행차 및 지능형 로봇, 챗봇과 같이 산업별로 인공지능 SW가 적용되는 영역은 실생활과 밀접하게 관련되어 있으며, 인공지능 SW 스스로 객관적이지 않은 데이터를 통해 학습하는 순간 발생 가능한 차별성, 편향성, 불공정성, 비도덕성에 대한 사회적 문제도 발생할 가능성이 큰 분야이다.

이에 따라 본 연구에서는 분야 별 Case 연구를 통해 인공지능SW로부터 발생 가능한 오작동을 도출 및 분석해보고, 위험 및 안전사고의 예방 차원에서 필요한 안전 기능과 개발 시 고려해야 할 요소를 제안하고자 한다.

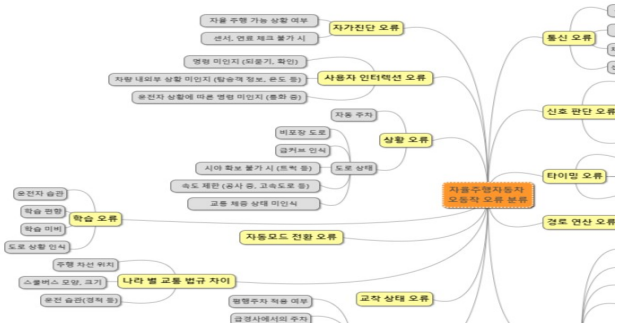
2. 관련 연구

구글 및 오픈 AI 등의 단체에서는 안전한 AI 개발을 위한 5가지 화두를 제시하였다[3]. 또한 일본 AI 네트워크 사회 추진단에서는 AI 개발자가 지켜야 할 개발 원칙을 수립하고 보고서에 수록하였다[4]. 미국에서 열린 AI 컨퍼런스에서는 AI 윤리와 가치, 장기적인 문제에 대한 논의도 이루어졌다[5].

SW 안전과 관련해서는 기능 안전성 표준(IEC61508, ISO26262 등)이 있다[6]. 정보통신산업진흥원에서는 안전성 관련 소프트웨어 제품 및 소프트웨어를 포함하는 시스템을 개발하는 기업들을 위한 다양한 실무 지침을 만들어서 배포하고 있다[7].

3. 인공지능 SW 오작동 분석 및 필요 안전 기능 도출

인공지능 SW는 일반성, 방대성 등 지식의 특성뿐만 아니라 일반SW와 달리 추론 기능 등의 특성을 가진다[8]. 이러한 특성을 바탕으로 실생활에 가장 활발히 적용되고 있는 분야인 자율주행차와 지능형로봇에 발생 가능한 오작동 사례를 도출해보았다.



<그림1>자율주행차의 오작동 분류

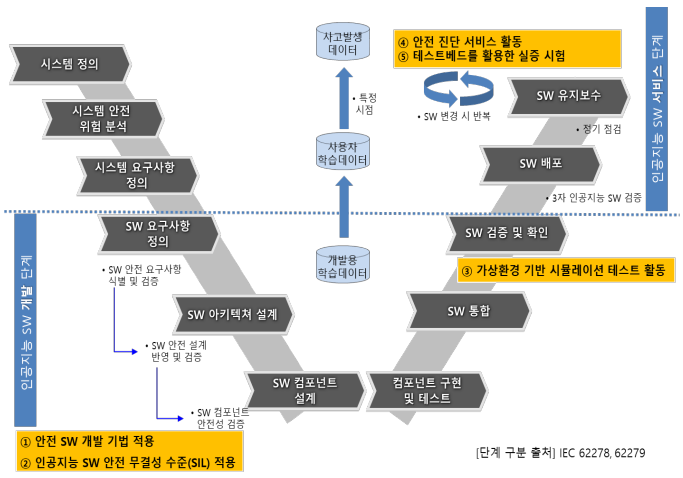
이와 같이 다양한 사례로부터 공통적으로 정의할 수 있는 인공지능 SW에 대한 오작동 유형을 6가지로 정의하고, 필요한 안전 기능을 매핑해보았다.

<표2> 인공지능 SW 오작동 유형

No	오작동 유형	정의	필요 안전기능
1	학습 부족 오류	인공지능 SW가 동작하기 위한 학습 데이터가 충분하지 않아서 발생하는 오류	강제종료 한계설정
2	학습 편향 오류	인공지능 SW가 동작은 하지만 부적절한 데이터로 학습이 되어 발생하는 오류	강제종료 한계설정
3	이미지 인식 오류	인공지능 SW가 동작은 하지만 이미지 인식 알고리즘이 자체가 최적화되지 않아서 발생하는 인식 오류	모드전환 경고알림 접근기록
4	상황 판단 오류	인공지능 SW가 동작 시에 발생하는 다양한 요소로 인해 상황 판단을 부적절하게 하여 발생하는 오류	Fail Safe 예외상황보고
5	조작 알고리즘 오류	인공지능 SW가 동작하는 중 조작 알고리즘 자체가 최적화되지 않아서 발생하는 인식 오류	강제종료 Fail Safe
6	주변 상황 오류	인공지능 SW와 상호 작용하는 다른 외부 요인에 따라 인공지능 SW의 동작을 방해하여 나타나는 오류	예외상황보고 상황모니터링 한계설정

4. 인공지능 SW 개발 시 고려 요소

인공지능 SW 오동작을 예방하기 위한 안전 기능의 효과적인 적용을 위해 인공지능 SW 개발 시에 적용해야 인공지능 SW 검증 모델을 제안한다. 이 모델에서는 인공지능 SW 개발 시 가장 중요하게 고려해야 될 요소인 학습 데이터를 활용하는 관점에서 크게 인공지능 SW 개발 단계와 인공지능 SW 서비스 단계별로 구분하며 각 단계별로 어떤 소프트웨어 공학 기법을 적용해야 하는 지를 정의하였다.

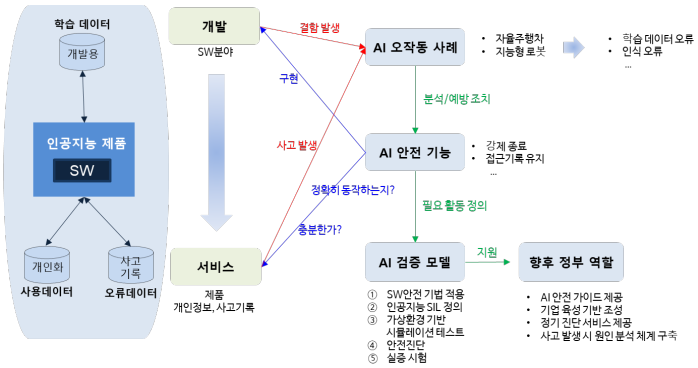


<그림3> 인공지능 SW 검증 모델

첫째, 기능 안전성에서 요구하고 있는 안전 관리 프로세스를 적용한다. 이때 인공지능 SW 특성을 고려해야하고, 인공지능 제품의 활용 분야(도메인)에 따라 적용되는 수준이 차별화되도록하는 것이 중요하다. 둘째, 인공지능 SW 안전 무결성 수준(SIL)을 적용한다. ISO26262의 ASIL 등급 기준 선정 시 사용되는 요소인 영향도, 발생빈도, 제어 가능성을 각 도메인 별로 판단하는 기준을 수립하고, 이를 레벨 별로 구분하여 필요한 안전 기능을 구현할 수 있도록 한다. 셋째, 가상환경 기반 시뮬레이션 테스트 활동을 실시한다. 가상 시나리오 구축 및 학습데이터의 생성 및 반복, 평가 등을 시뮬레이션을 통해 수천 가지의 위험을 파악하고 안전 기능이 제대로 동작하는지 확인할 수 있어야 한다. 넷째, 정기적인 안전 진단 서비스 활동을 실시한다. 안전 진단 서비스는 크게 안전 기능을 충분하게 갖추고 있는 지 판단하는 서비스와 안전 기능이 정확히 동작하는 지 기능을 검증하는 서비스를 해야 한다. 다섯째, 테스트 베드 구축을 통한 실증 시험을 실시한다. 인공지능 SW가 HW와 결합하여 실제 환경과 유사한 곳에서 오동작이 없는지를 검출하는 시험을 실시하여 안정적인 서비스가 가능한지를 확인하여야 한다.

5. 결 론

본 연구에서 인공지능 SW의 오작동으로부터 이를 예방하기 위한 안전 기능의 도출하고, 이러한 안전 기능을 효과적으로 구현하고 검증하기 위한 인공지능 SW 개발 시 적용해 볼 수 있는 검증 모델을 제안하였다.



<그림4> 안전한 인공지능 SW를 위한 검증 체계(안)

하지만 인공지능 SW는 여러 산업에 다양하게 적용될 수 있는 만큼 도메인 별로 특화된 안전 기능과 테스트베드 구축이 반드시 필요한 분야이다. 따라서, 향후 정부에서는 이런 기반 조성을 위해 노력해야 할 것이며, SW로 구현될 안전한 지능정보사회를 위해 다양한 소프트웨어 공학 기법 적용 및 연구를 지속적으로 해야 할 것으로 생각된다.

참고문헌

[1] 국회입법조사처, 2017 국정감사 정책자료 (소프트웨어 안전관리 정책 추진 현황), 2017, 8
[2] 관계부처합동, 지능정보사회 중장기 종합대책, 2016, 12
[3] 구글, 오픈AI, 스탠포드대학, 버클리대학 공동 연구, Concrete Problems in AI Safety, 2016, 7
[4] 일본 총무성, AI 네트워크 사회 추진 회의(AI 개발 원칙), 일본 정보통신정책연구소(IICP) 보고서 2017
[5] Future of Life Institute, Asilomar AI Principles, 2017, 1
[6] ISO 26262-6: Road vehicles-Functional safety-Part 6: Product development at the software level,2011
[7] 정보통신산업진흥원, SW신뢰안정성확보를 위한 공통가이드, 2016.11
[8] 석왕헌, 이광희(ETRI), 인공지능 기술과 산업의 가능성, 2015.10.30

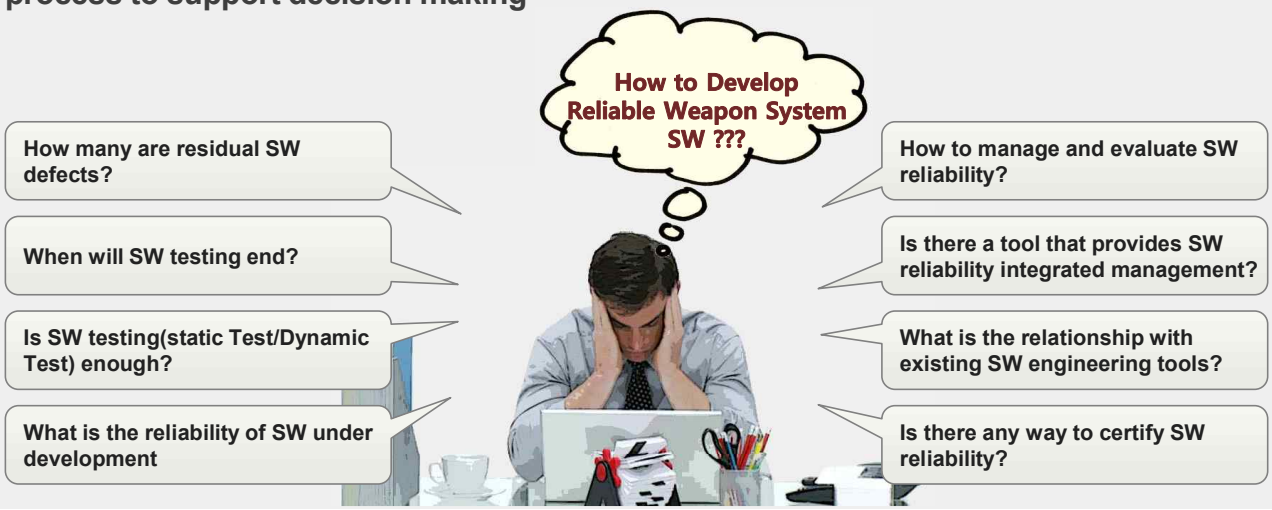
V-Diagram 기반 SW Reliability Engineer의 업무 분석과 효과적인 도구 사용 방안

서달미, 손근태, 서형오, 오정섭*
*㈜엔에스이 융합솔루션개발그룹

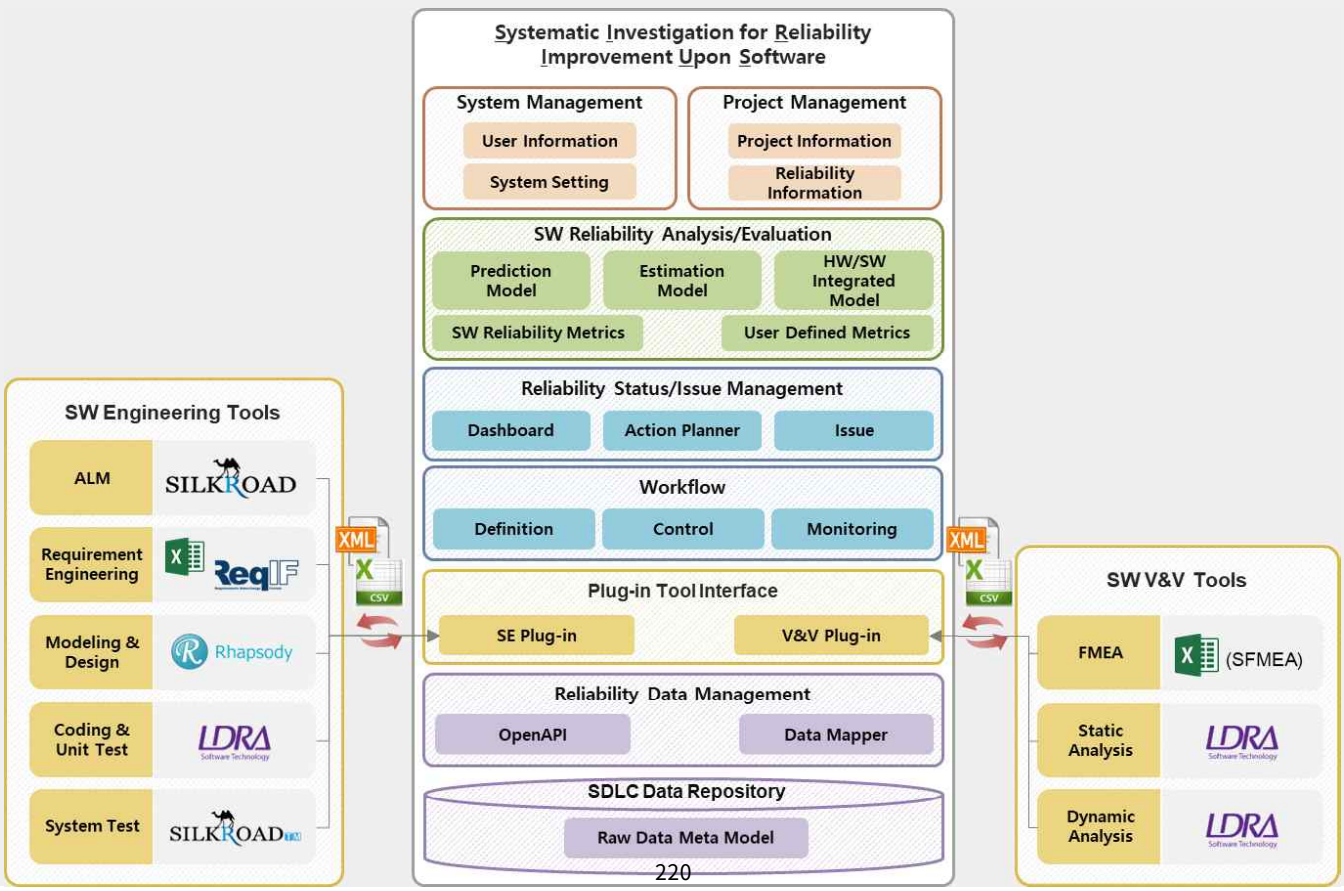
개요

SIRIUS : Development Background

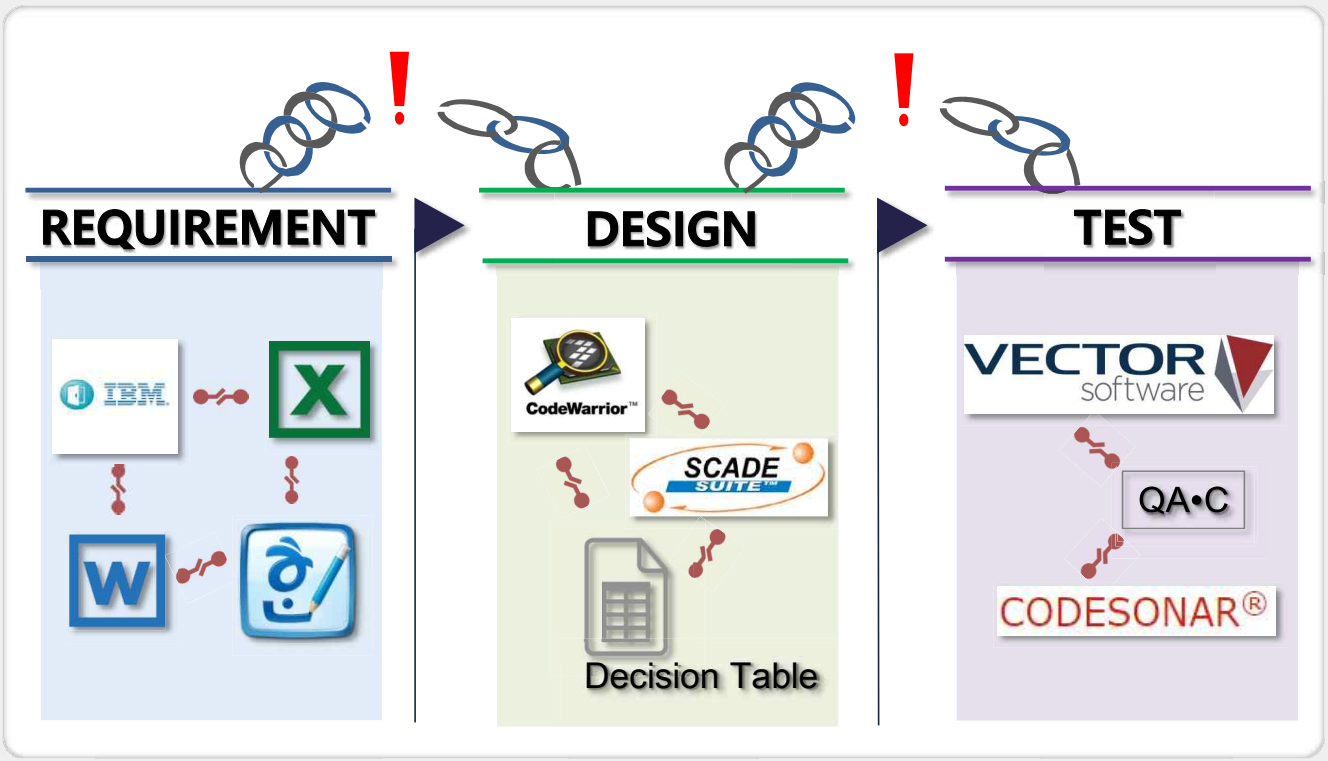
- **Enlargement and complexity of weapon system force demands for establishment of SW reliability process**
  - Improvement countermeasures for overcoming limitation of source code-based SW reliability evaluation are required.
  - Difficult to understand number of remaining failures, level of SW quality and test end point.
- **Limitations of putting unlimited SW testing efforts due to limited test period & budget**
- **Needs for quantitative evaluation of SW reliability throughout the entire development process to support decision making**



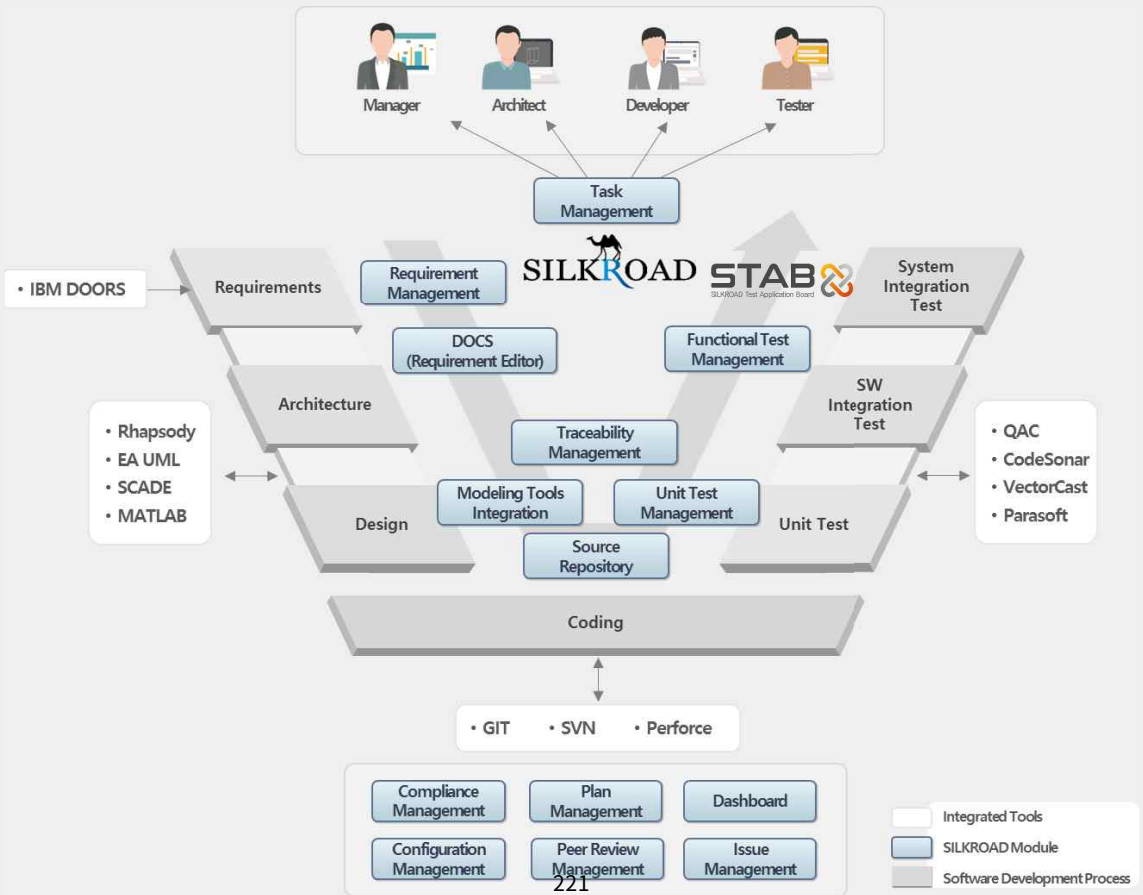
Composition of SIRIUS



SILKROAD : Development Background



SILKROAD : Composition

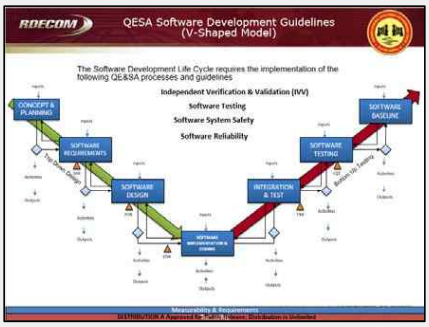


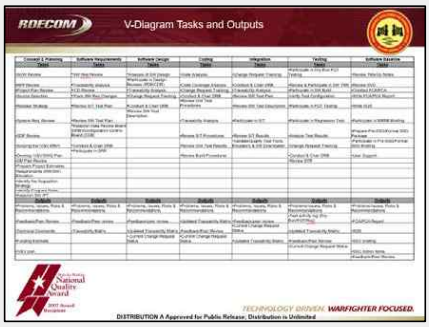
QE&SA Software Development Guidelines

- Software(SW) Quality, Reliability and Safety assures that the software is safe, reliable and performs are intended.

- The purpose of software quality is to build quality into the software by performing analyses and evaluations of the requirements, design, code, and perform verification and validation of processes and products to mitigate risks.

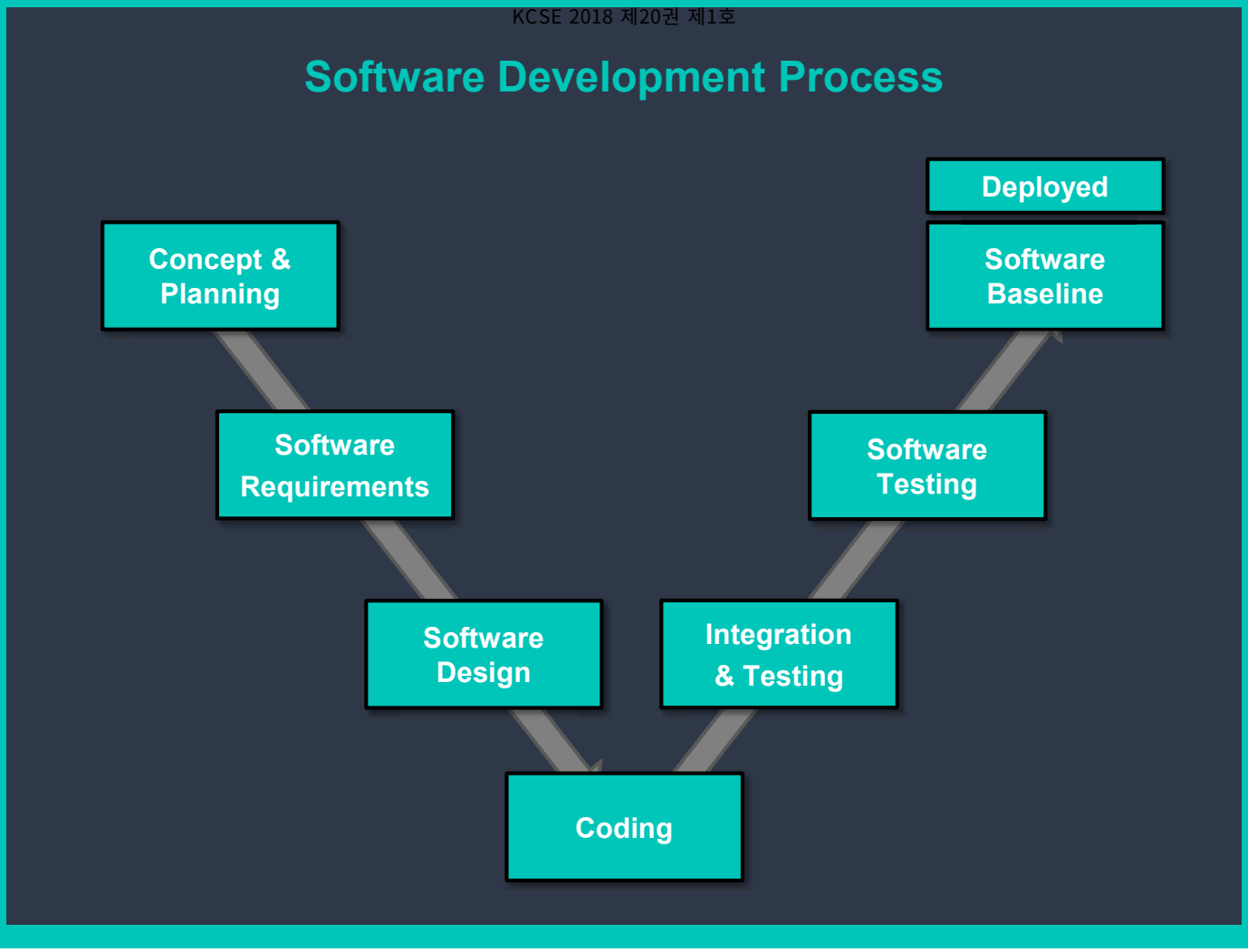
- Key objective is to eliminate defect escapes during SW development.

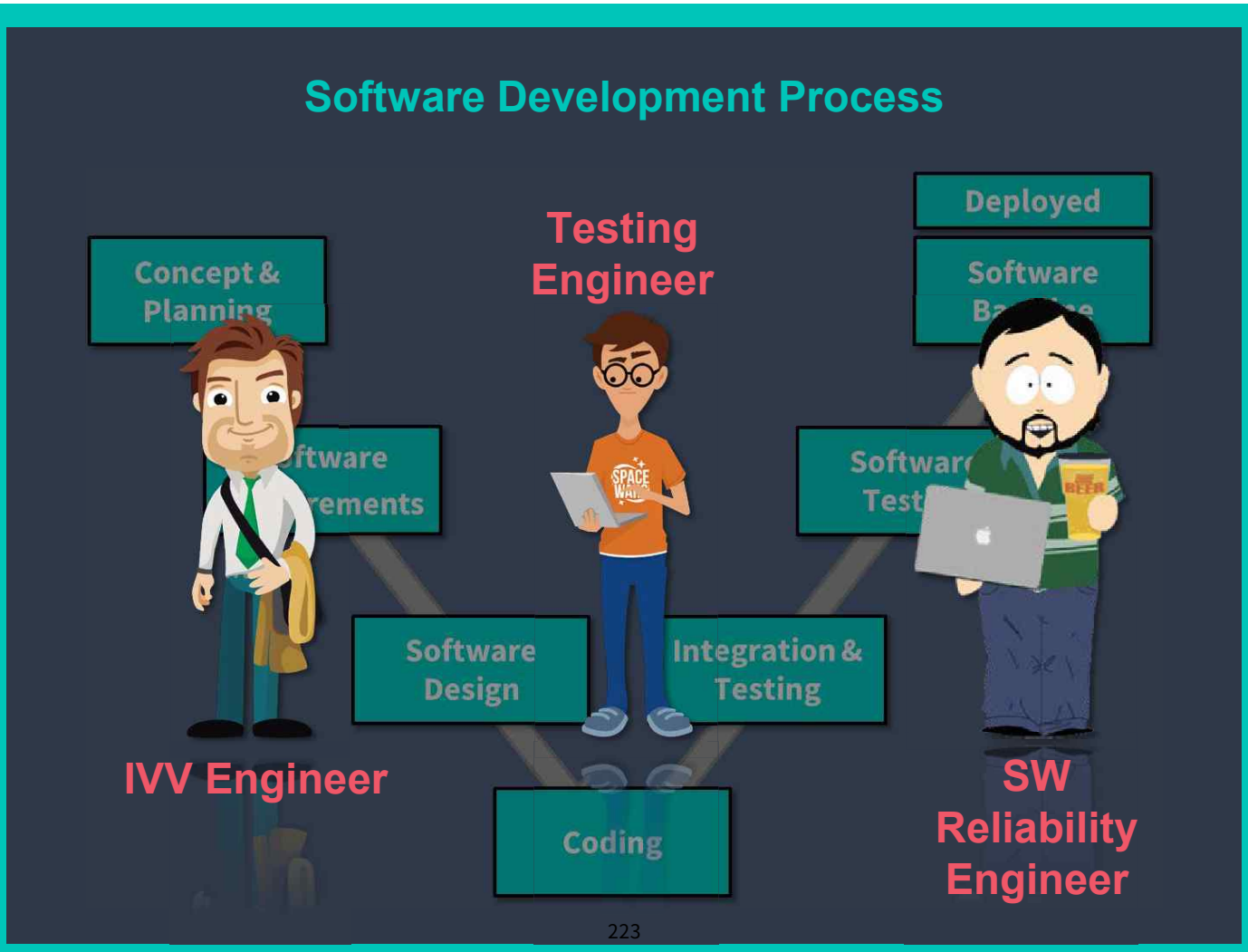




*ARDEC, "QESA Software Development Guidelines"

V-Diagram 기반
SW Reliability Engineer의 업무분석


Software Development Process
Concept & Planning
Software Requirements
Software Design
Coding
Integration & Testing
Software Testing
Software Baseline
Deployed


Software Development Process
Concept & Planning
Testing Engineer
Deployed
Software Design
Integration & Testing
Coding
IVV Engineer
SW Reliability Engineer

Roles and Responsibilities

Hello!

I am SW Reliability Engineer.

I plan, monitor and executes SW Reliability analysis/activities in order to mitigate defects leading to System abort and Essential Function Failures as documented in the Failure Definition/Scoring Criteria (FD/SC).

My main deliverables are: Reliability Statement and so on.

V-Diagram Tasks and Outputs for SW Reliability Engineer

Concept & Planning	Software Requirements	Software Design	Coding	Integration & Testing	Software Testing	Software Baseline
Tasks	**Tasks**	**Tasks**	**Tasks**	**Tasks**	**Tasks**	**Tasks**
• Review SOW	• Review Trace Matrix	• SW FMECA	• Review Code Testing	• Review Test Results	• Review Test Results	• Input to SW reliability Recommendation?
• Project Estimate	• Review	• SW FTA	• Mission Critical Test Procedures are Identified and Tagged	• PCR Review	• PCR Review	• Input to Software Quality Statement
• Input to Software System Reliability Plan	• Review DCM	• Review DCM	• Verify Mission Critical Code is Tagged	• SWR MCF Maturity Assessment	• SWR MCF Maturity Assessment	• Participate in SSG
• Establish SW Reliability Working Group	• Requirements for Mission Critical Tagging	• Verify SW Critical Components are Testable	• Review DCM			
• Preliminary Hazard Analysis			• Update SDFs			
Outputs	**Outputs**	**Outputs**	**Outputs**	**Outputs**	**Outputs**	**Outputs**
• Feedback to PMs	• Feedback/Peer review	• Feedback/Peer review	• Track SW Change Requests	• Feedback/Peer review	• Provide recommendation to conduct FQT	• FCA/PCA Report
• Technical Comments	• Traceability Matrix	• Update Traceability Matrix	• Updated Traceability Matrix	• Update Traceability Matrix	• Test activity log (FQT/Reg)	• SQS
• Funding Estimate	• Req. phase Checklist	• Design phase Checklist	• Feedback/Peer Review	• Phase Checklist	• Incident data (FQT/Reg)	• SSG briefing
• Concept & Planning Phase Checklist	• Draft SW Reliability		• SW Implementation Phase Checklist		• Phase checklist	• SSG Action Items
	• Draft FDSC				• Update Traceability Matrix	
					• SW Test Report	
					• SWR Metrics Analysis Report	

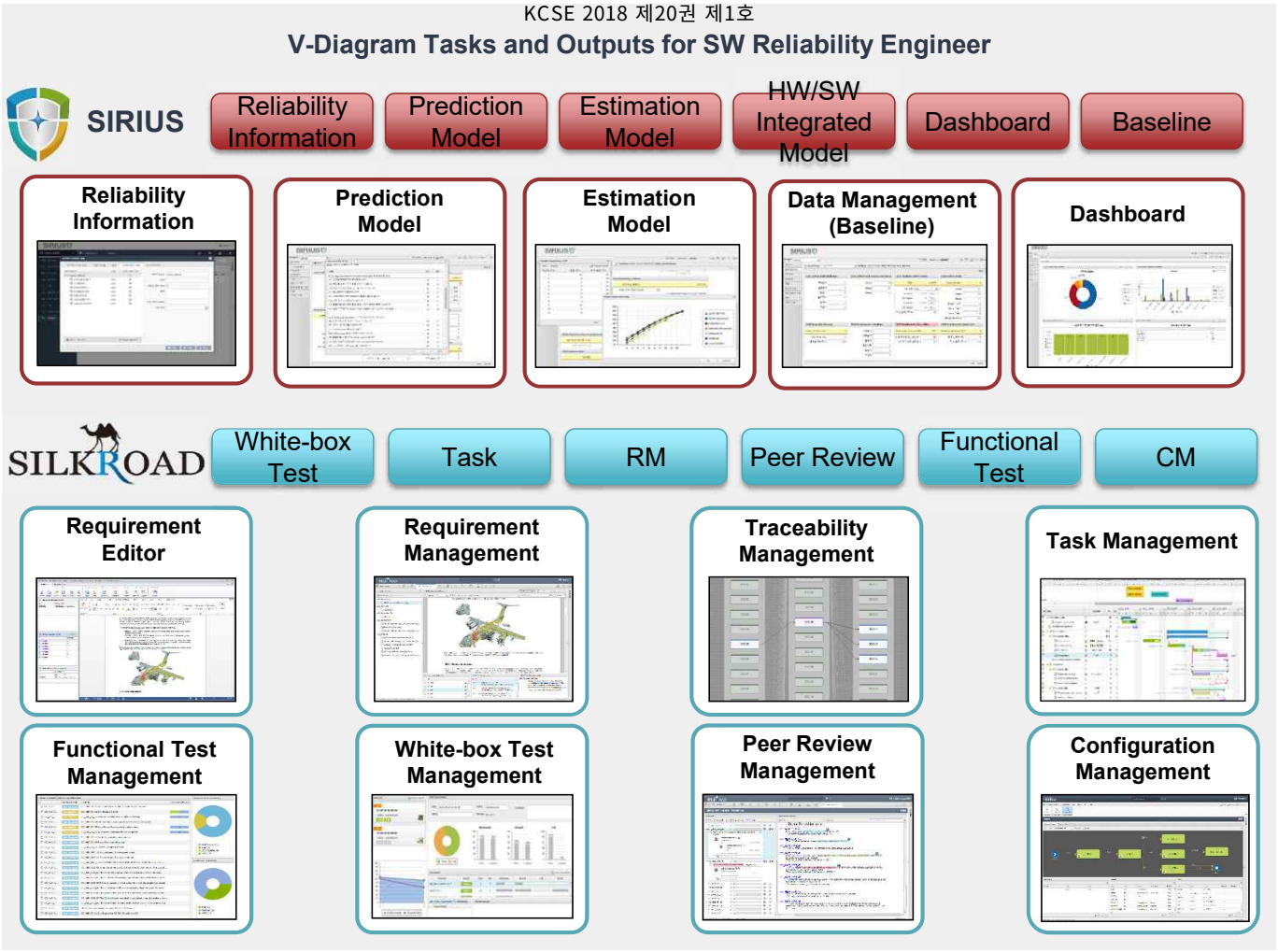
*ARDEC, "QESA Software Development Guidelines"

V-Diagram Tasks and Outputs for SW Reliability Engineer

Concept & Planning	Software Requirements	Software Design	Coding	Integration & Testing	Software Testing	Software Baseline
Tasks	**Tasks**	**Tasks**	**Tasks**	**Tasks**	**Tasks**	**Tasks**
Peer Review	Traceability Matrix	SW FMECA	Review Code Testing	Review Test Results	Review Test Results	Reliability Recommend.
Project Estimates	Peer Review	SW FTA	Test Proc. Tagging	PCR Review	PCR Review	• Input to Software Quality Statement
• Input to Software System Reliability Plan	• Review DCM	• Review DCM	• Verify Mission Critical Code is Tagged	SWR MCF Assessment	SWR MCF Assessment	• Participate in SSG
• Establish SW Reliability Working Group	Requirement Tagging	Verify Testable	• Review DCM			
Prelim. Hazard Analysis			• Update SDFs			
Outputs	**Outputs**	**Outputs**	**Outputs**	**Outputs**	**Outputs**	**Outputs**
Review Feedback	Review Feedback	Review Feedback	Review Feedback	Review Feedback	STR	FCA/PCA Report
Test Funding Estimate	Traceability Matrix	Traceability Matrix	Traceability Matrix	Traceability Matrix	SWR Metric Analysis	SQS
• Concept & Planning Phase Checklist	Checklist	Checklist	Checklist	Checklist	Checklist	SSG Action items
Checklist	SW Reliability		Track SW CR		Traceability Matrix	
	FD/SC				• SWR Metrics Analysis Report	

V-Diagram Tasks and Outputs for SW Reliability Engineer

Concept & Planning	Software Requirements	Software Design	Coding	Integration & Testing	Software Testing	Software Baseline
Prelim. Hazard Analysis	Requirement Tagging	SW FMECA	Test Proc. Tagging	SWR MCF Assessment	SWR MCF Assessment	Reliability Recommend.
Project Estimates	Traceability Matrix	SW FTA	Review Code Testing	Review Test Results	Review Test Results	
Peer Review	Peer Review	Verify Testable		PCR Review	PCR Review	
Reliability Information	Prediction Model	Prediction Model	Prediction Model	Estimation Model	Estimation Model	Dashboard
Task	RM	RM	Functional Test	Functional Test	Functional Test	Baseline
Peer Review	Peer Review		White-box Test	CM	CM	
Review Feedback	Review Feedback	Review Feedback	Review Feedback	Review Feedback	STR	FCA/PCA Report
Test Funding Estimate	Traceability Matrix	Traceability Matrix	Traceability Matrix	Traceability Matrix	SWR Metric Analysis	SQS
Checklist	Checklist	Checklist	Checklist	Checklist	Checklist	SSG Action items
	SW Reliability		Track SW CR		Traceability Matrix	
	FD/SC					



V-Diagram 기반 SW Reliability Engineer의 효과적인 도구 사용 방안

Concept & Planning

PURPOSE

- Identify the acquisition strategy in terms of cost, schedule, resources, scope of the project, and expected quality levels.
- Identify the major activities and software products for the project.

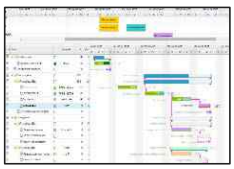
SWQ Roles, Responsibilities & Major Activities

- SOW/RFP Review
- Project Plan Review
- Prepare project estimates

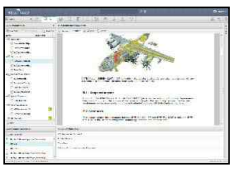
Reliability Information

Prediction Model

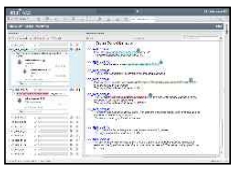
Task Management

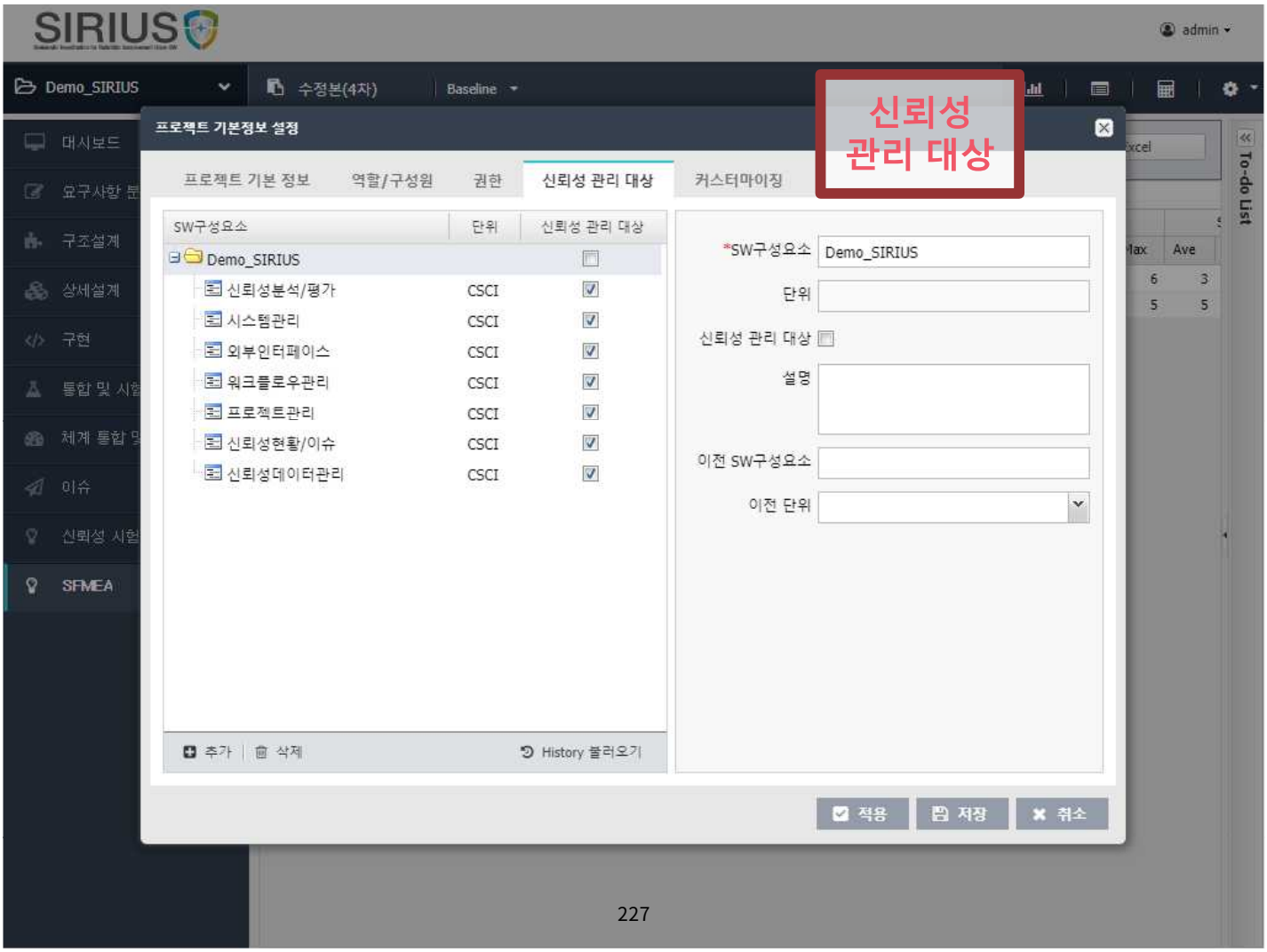


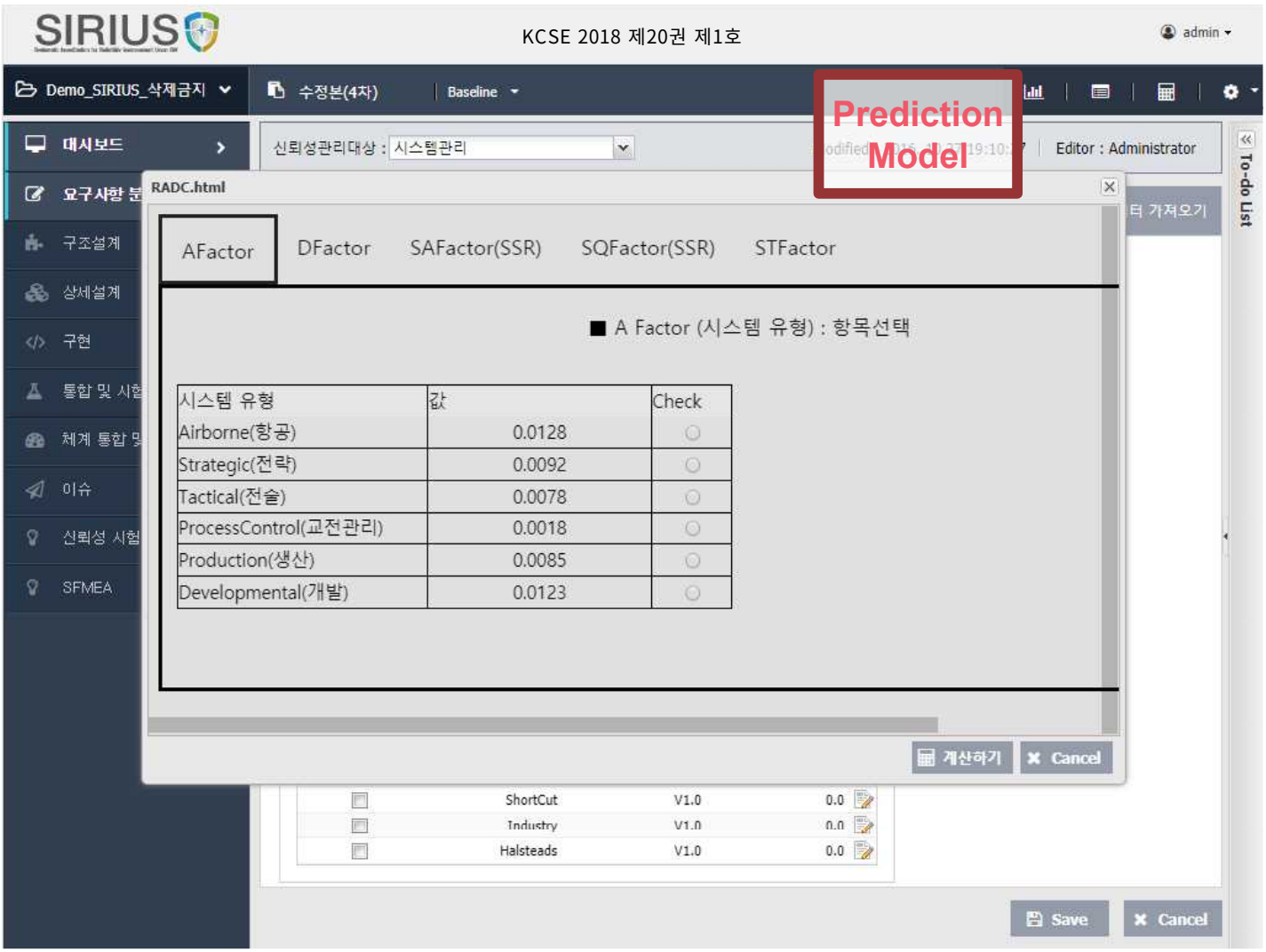
Plan Management

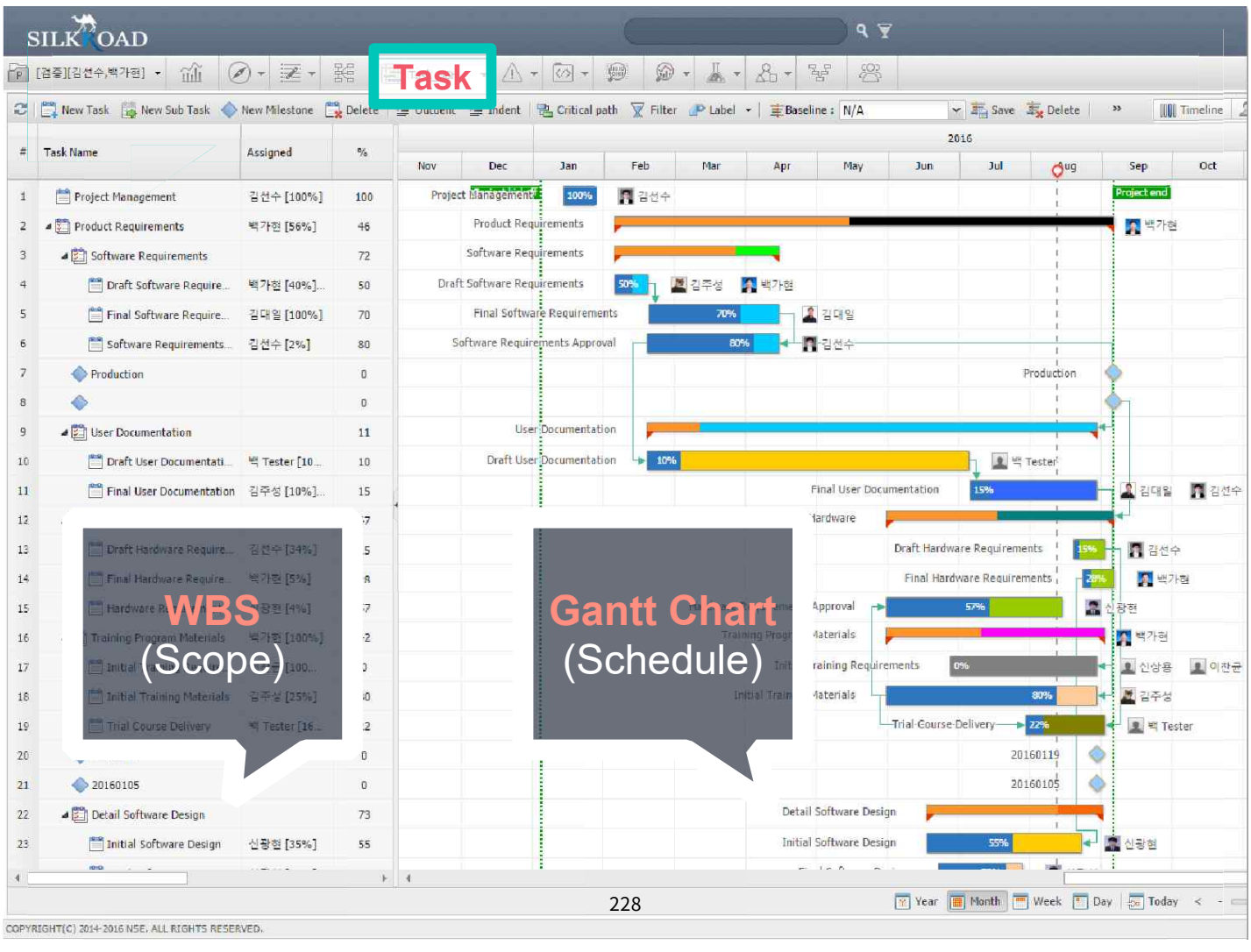


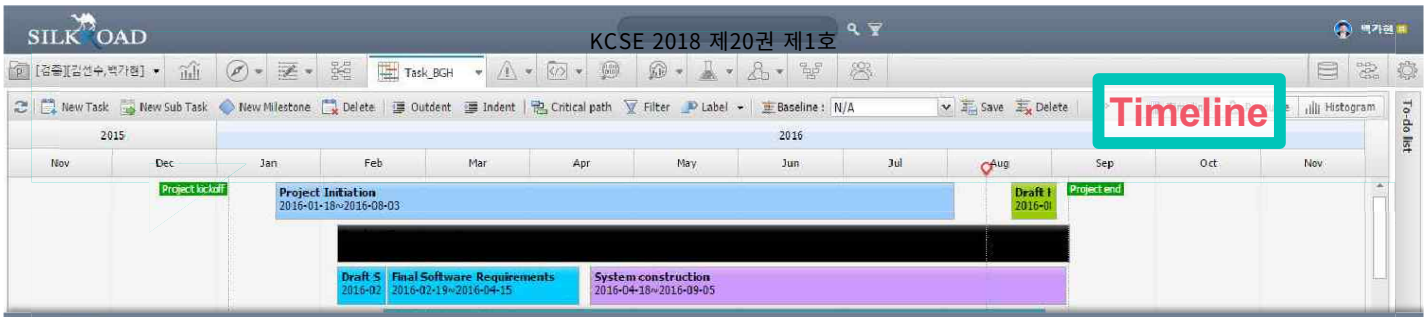
Peer Review Management

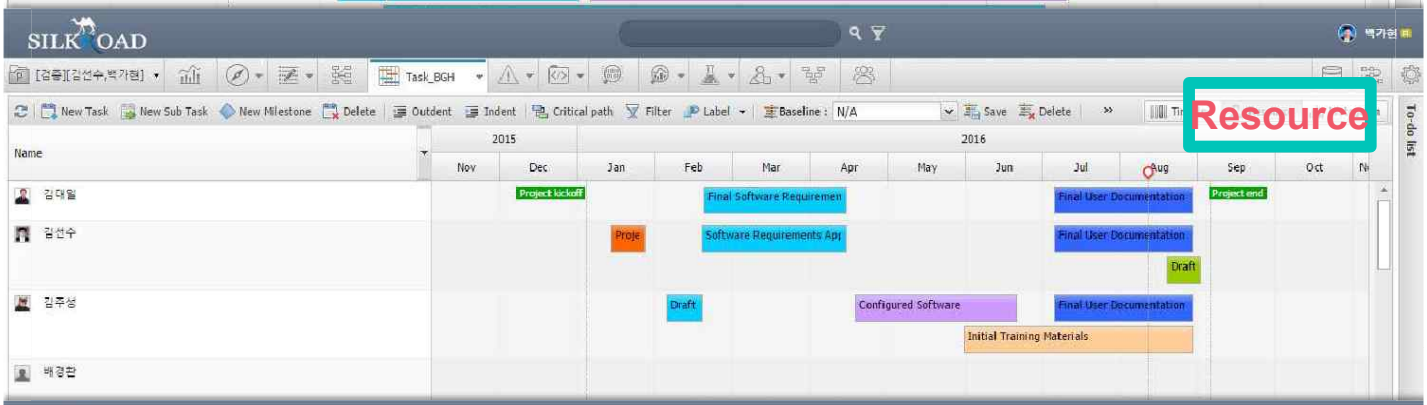


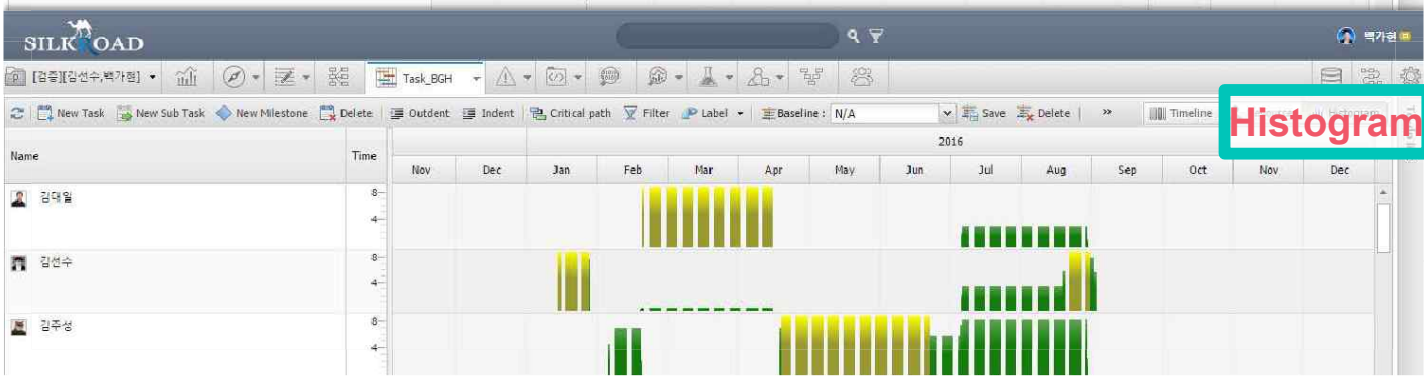


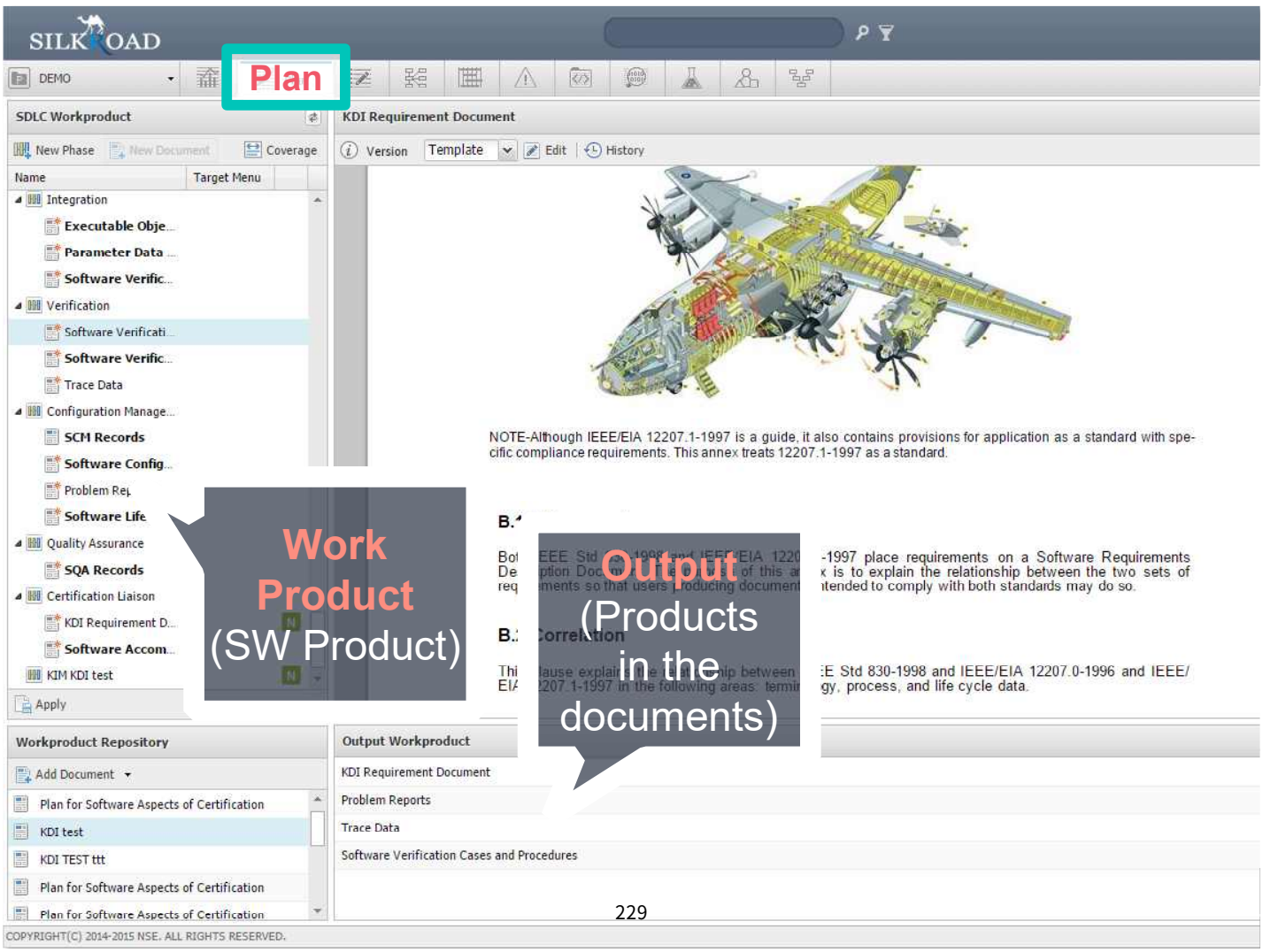

SIRIUS
admin
Demo_SIRIUS_삭제금지
수정본(4차)
Baseline
Prediction Model
대시보드
요구사항 분
구조설계
상세설계
구현
통합 및 시험
체계 통합 및
이슈
신뢰성 시험
SFMEA
신뢰성관리대상 : 시스템관리
Editor : Administrator
To-do List
RADC.html
AFactor
DFactor
SAFactor(SSR)
SQFactor(SSR)
STFactor
■ A Factor (시스템 유형) : 항목선택
시스템 유형 | 값 | Check
Airborne(항공) | 0.0128
Strategic(전략) | 0.0092
Tactical(전술) | 0.0078
ProcessControl(교전관리) | 0.0018
Production(생산) | 0.0085
Developmental(개발) | 0.0123
계산하기
Cancel
ShortCut V1.0 0.0
Industry V1.0 0.0
Halsteads V1.0 0.0
Save
Cancel


SILKROAD
Task
New Task
New Sub Task
New Milestone
Delete
Critical path
Filter
Label
Baseline :
Save
Timeline
Task Name
Assigned
2016
Project Management
Product Requirements
Software Requirements
Draft Software Requirements
Final Software Requirements
Software Requirements Approval
Production
User Documentation
Draft User Documentation
Final User Documentation
Draft Hardware Requirements
Final Hardware Requirements
Trial Course Delivery
Detail Software Design
Initial Software Design
WBS
(Scope)
Gantt Chart
(Schedule)
Year
Month
Week
Day
Today
COPYRIGHT(C) 2014-2016 NSE, ALL RIGHTS RESERVED.

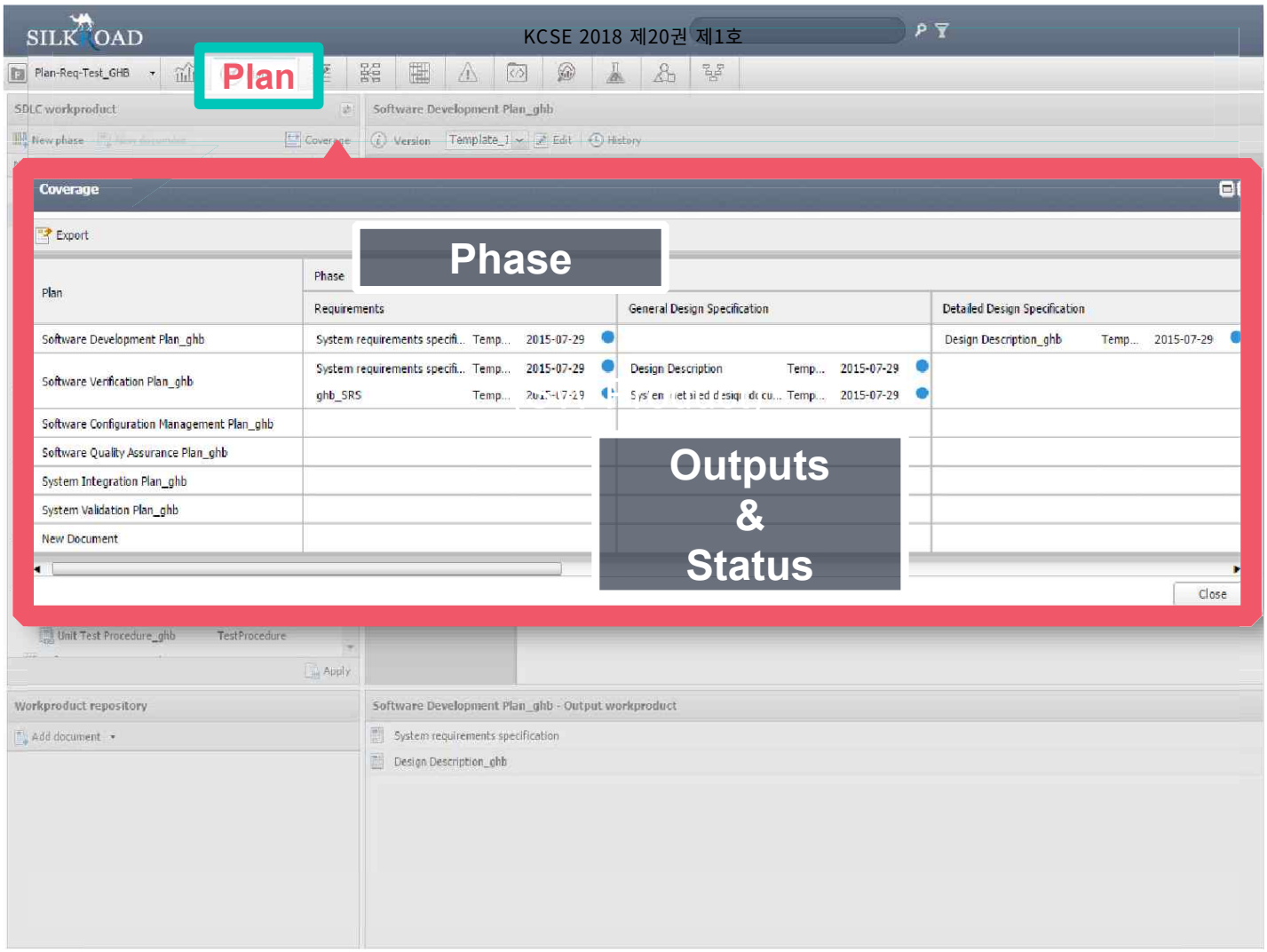


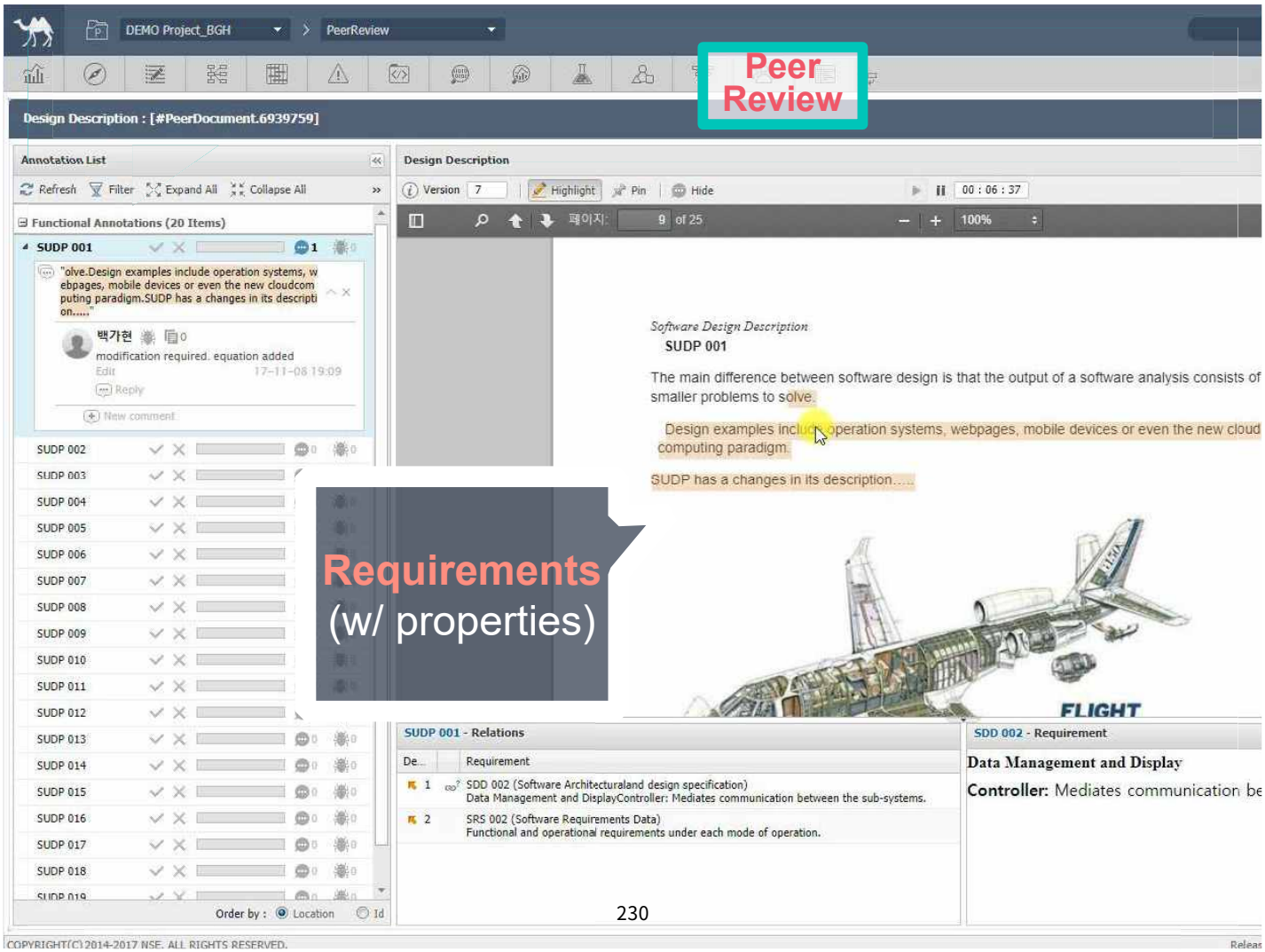






COPYRIGHT(C) 2014-2015 NSE. ALL RIGHTS RESERVED.





COPYRIGHT(C)2014-2017 NSE. ALL RIGHTS RESERVED.

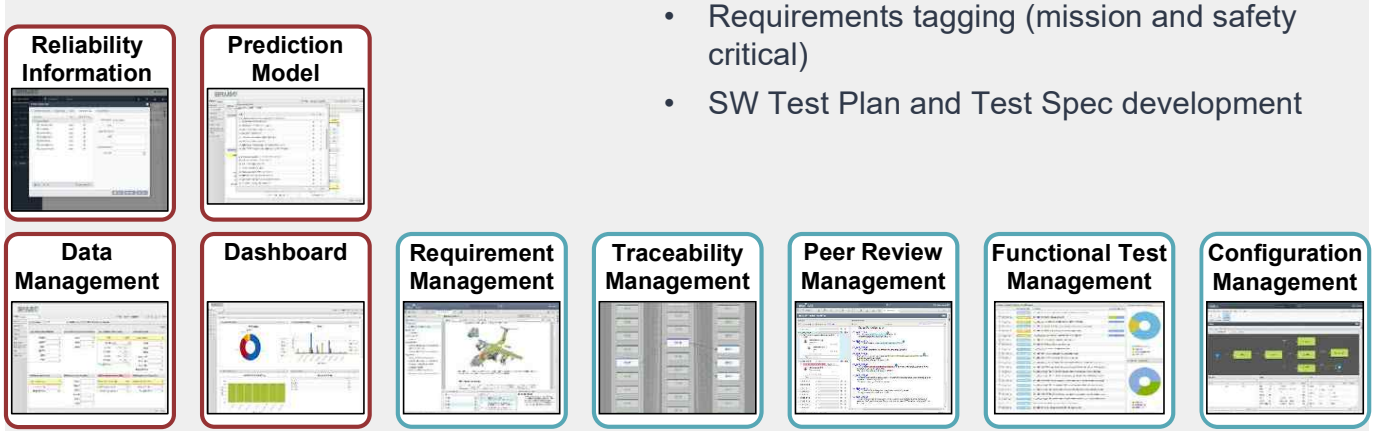
Software Requirements

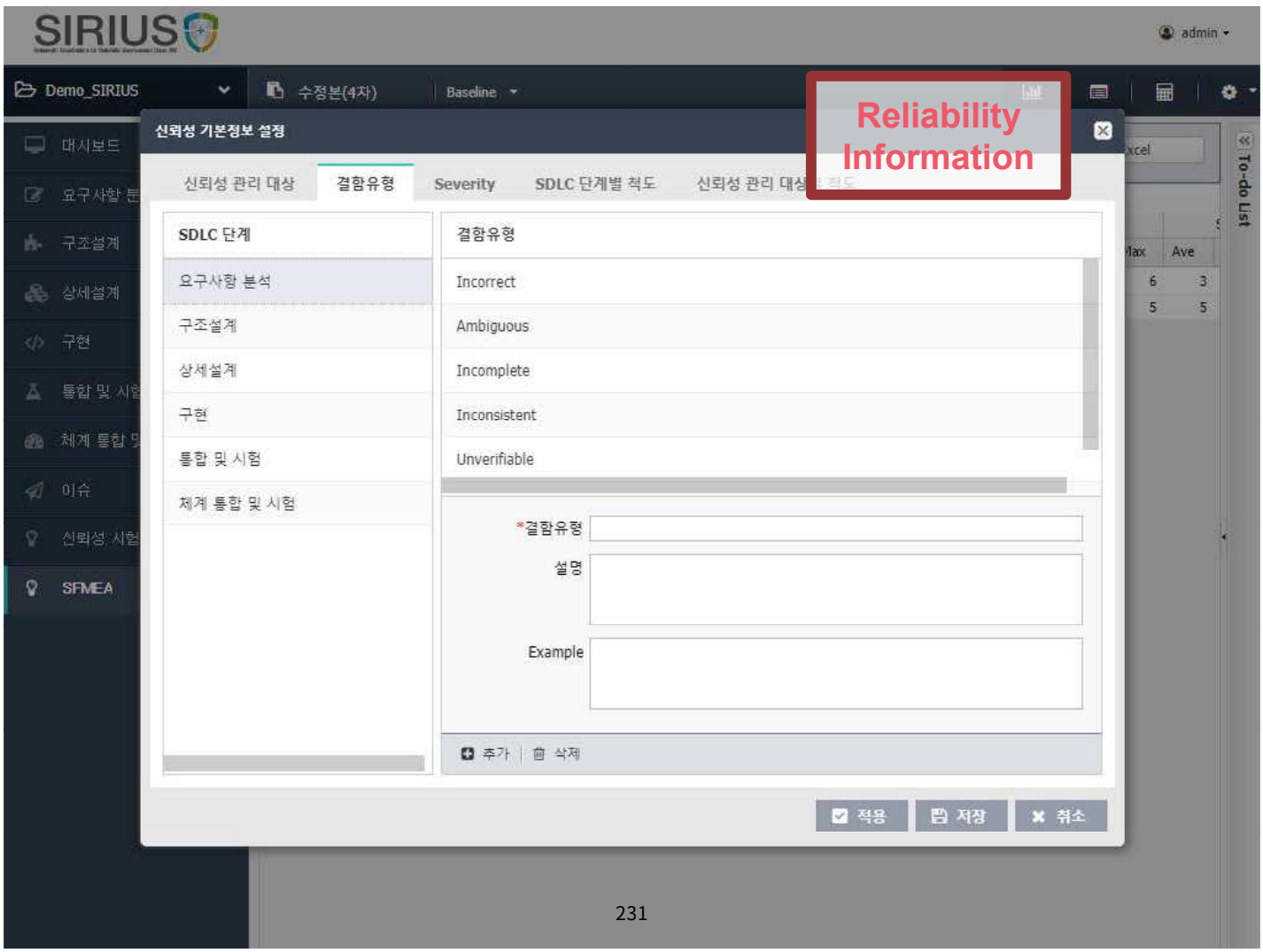
PURPOSE

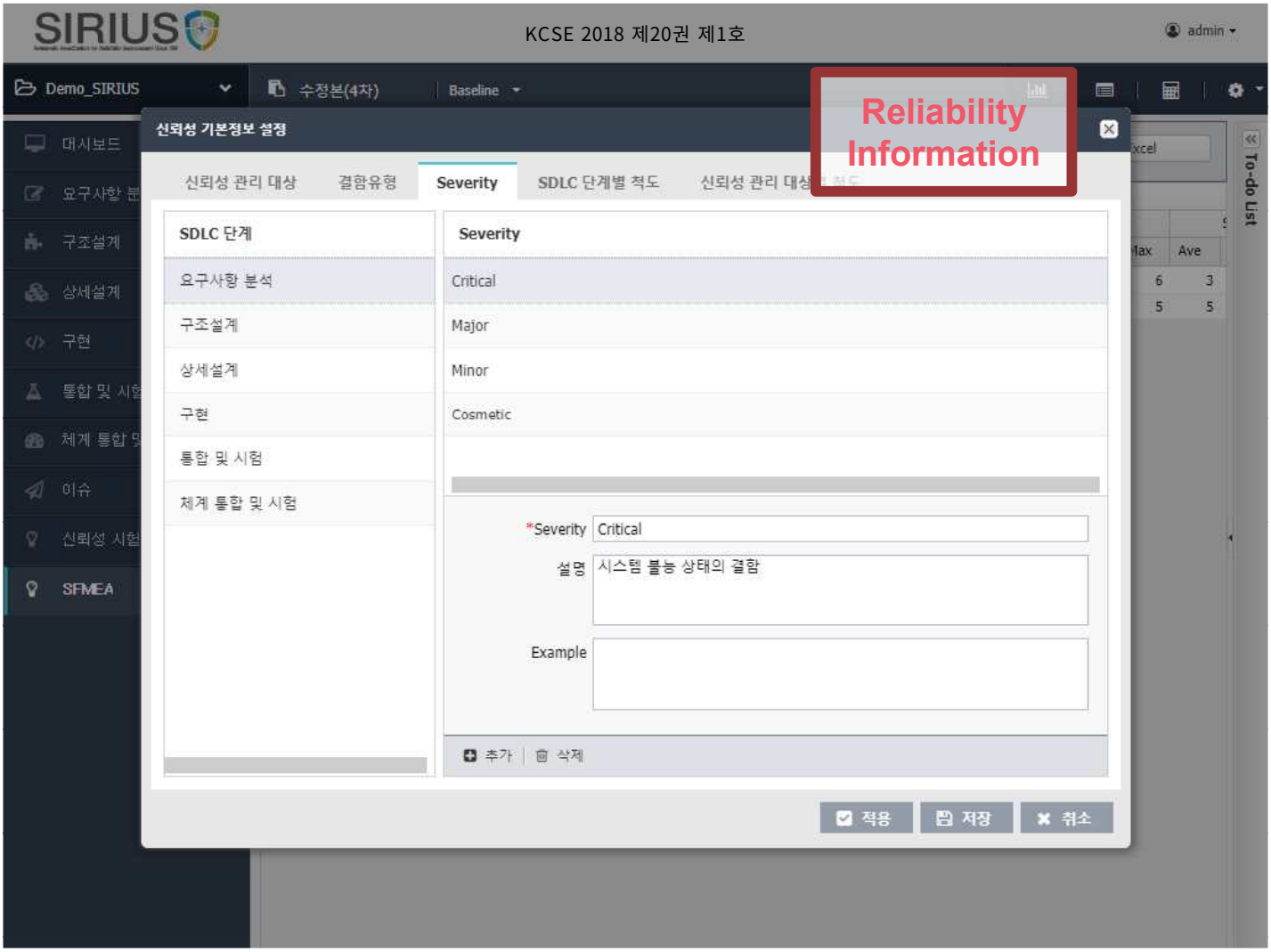
- Defines the software user and performance requirements in terms of quality, safety, testability and reliability.
- Define what is within the scope of the project.

SWQ Roles, Responsibilities & Major Activities

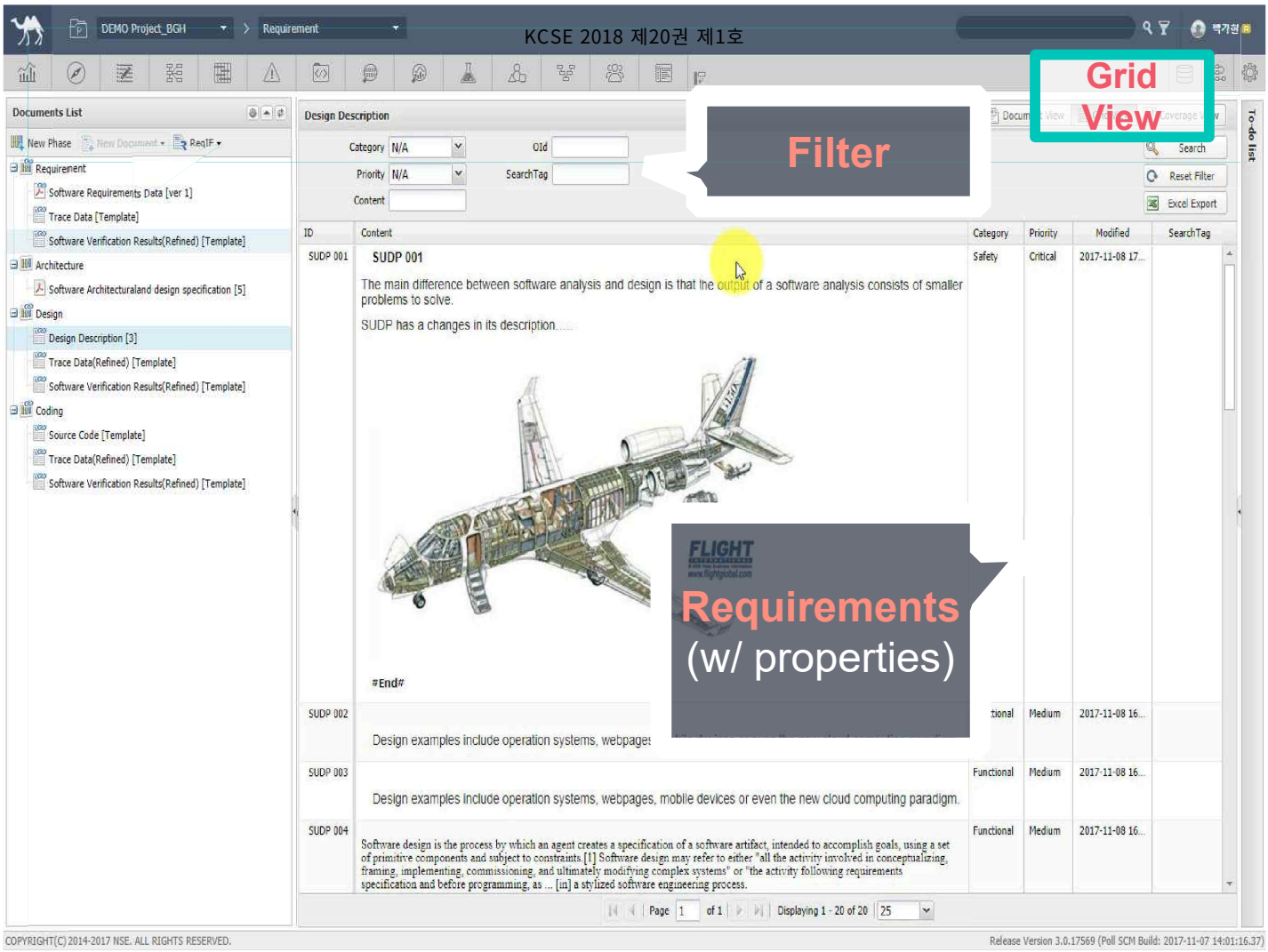
- Draft SW Reliability
- Draft FDSC
- SW Requirements review
- Traceability analysis
- Begin to track SW change requests – DRB
- Requirements tagging (mission and safety critical)
- SW Test Plan and Test Spec development

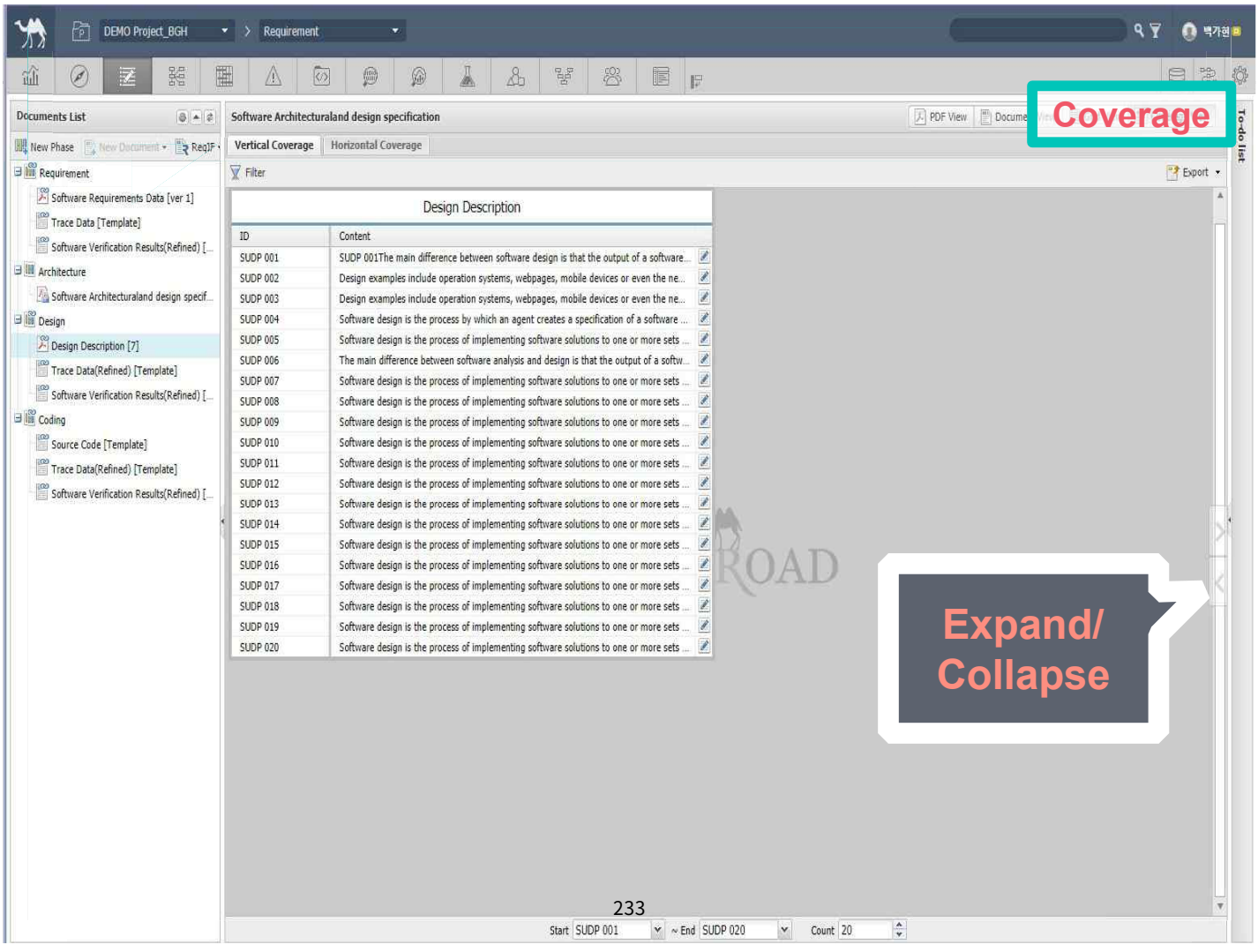


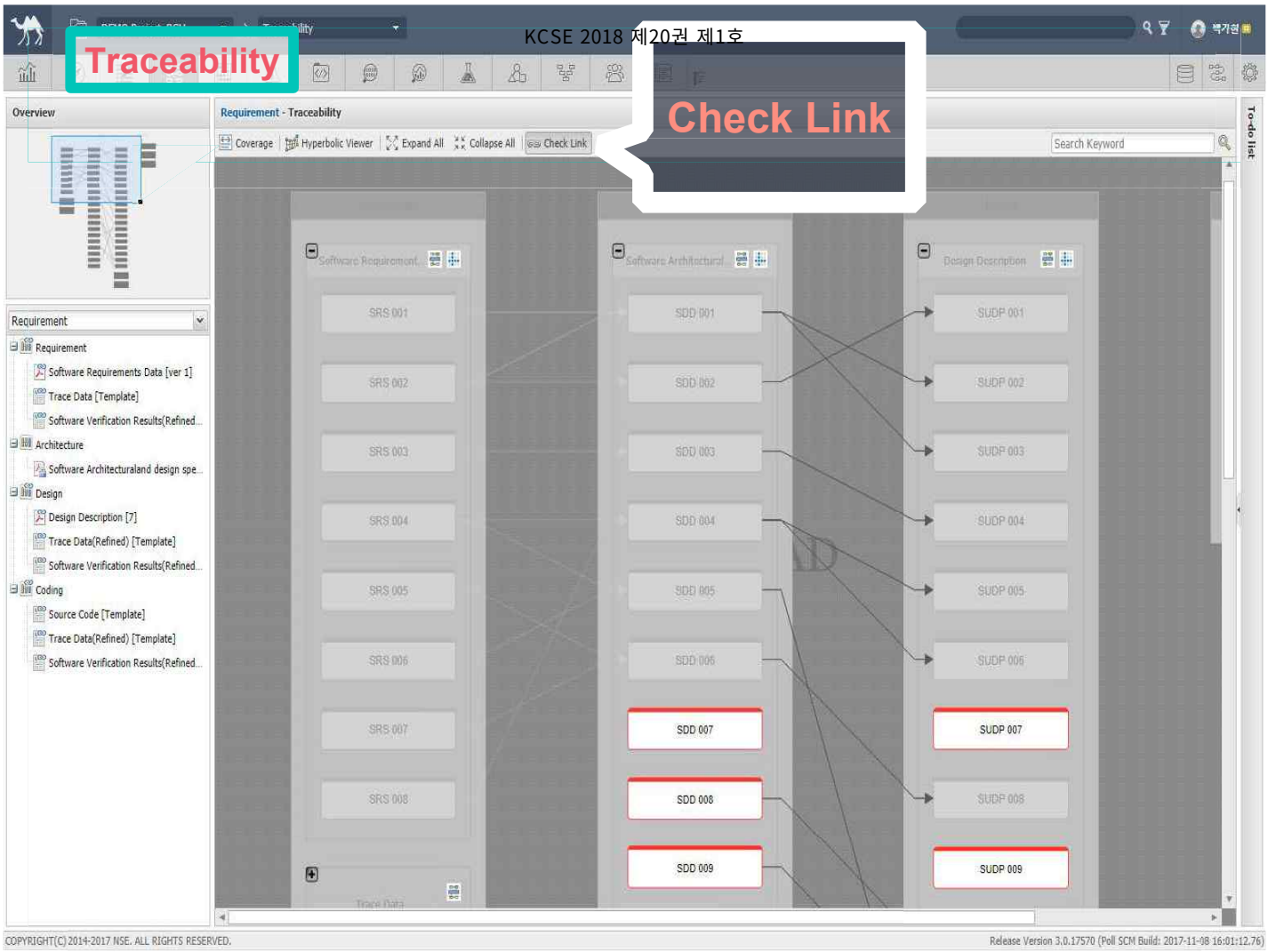


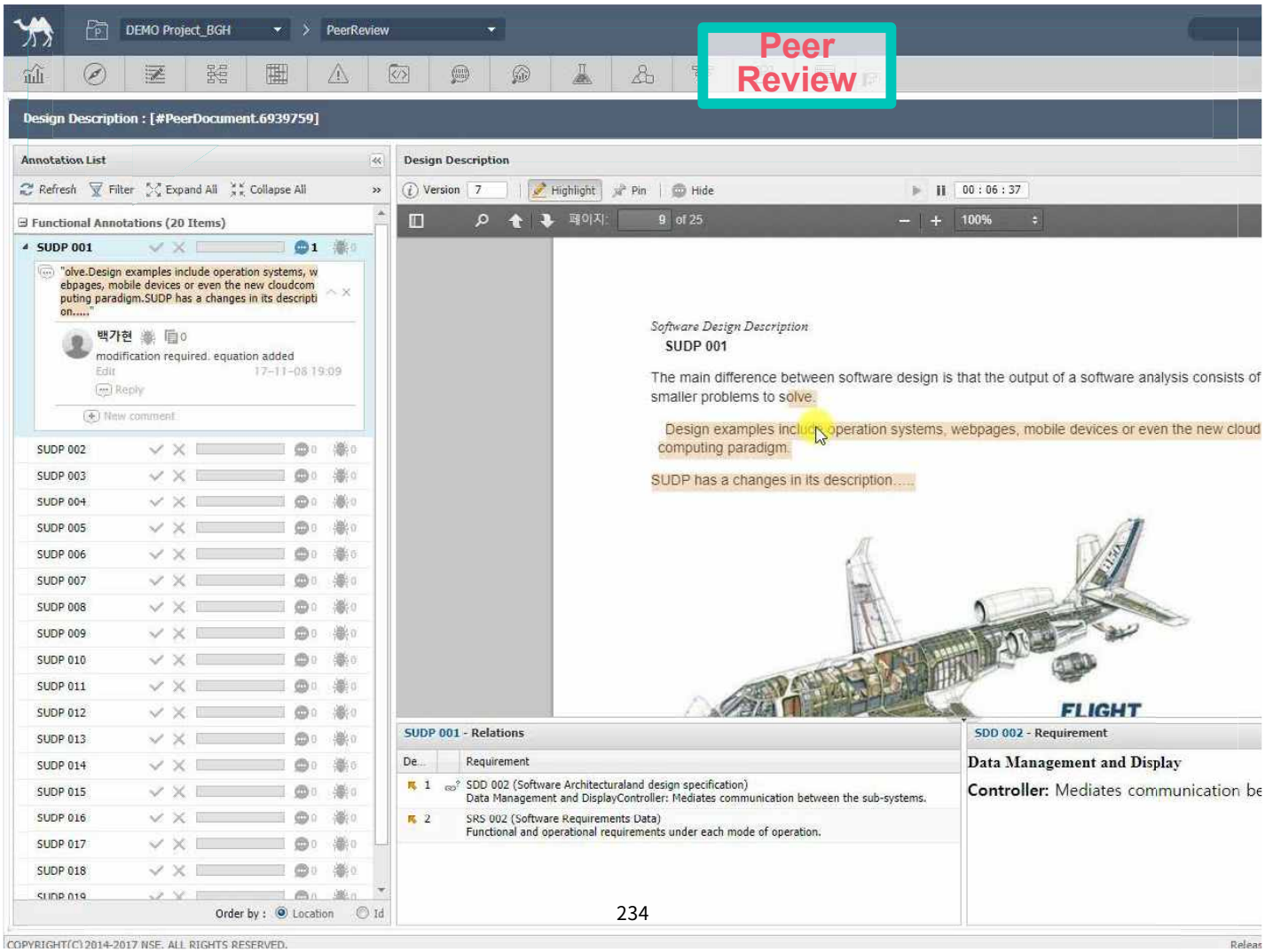

SIRIUS
admin
Demo_SIRIUS
수정본(4차)
Baseline
Reliability Information
신뢰성 기본정보 설정
신뢰성 관리 대상
결함유형
Severity
SDLC 단계별 척도
SDLC 단계
Severity
요구사항 분석
구조설계
상세설계
구현
통합 및 시험
체계 통합 및 시험
Critical
Major
Minor
Cosmetic
*Severity
Critical
설명
시스템 불능 상태의 결함
Example
추가
삭제
적용
저장
취소
대시보드
요구사항 분석
구조설계
상세설계
구현
통합 및 시험
체계 통합 및 시험
이슈
신뢰성 시험
SFMEA
To-do List

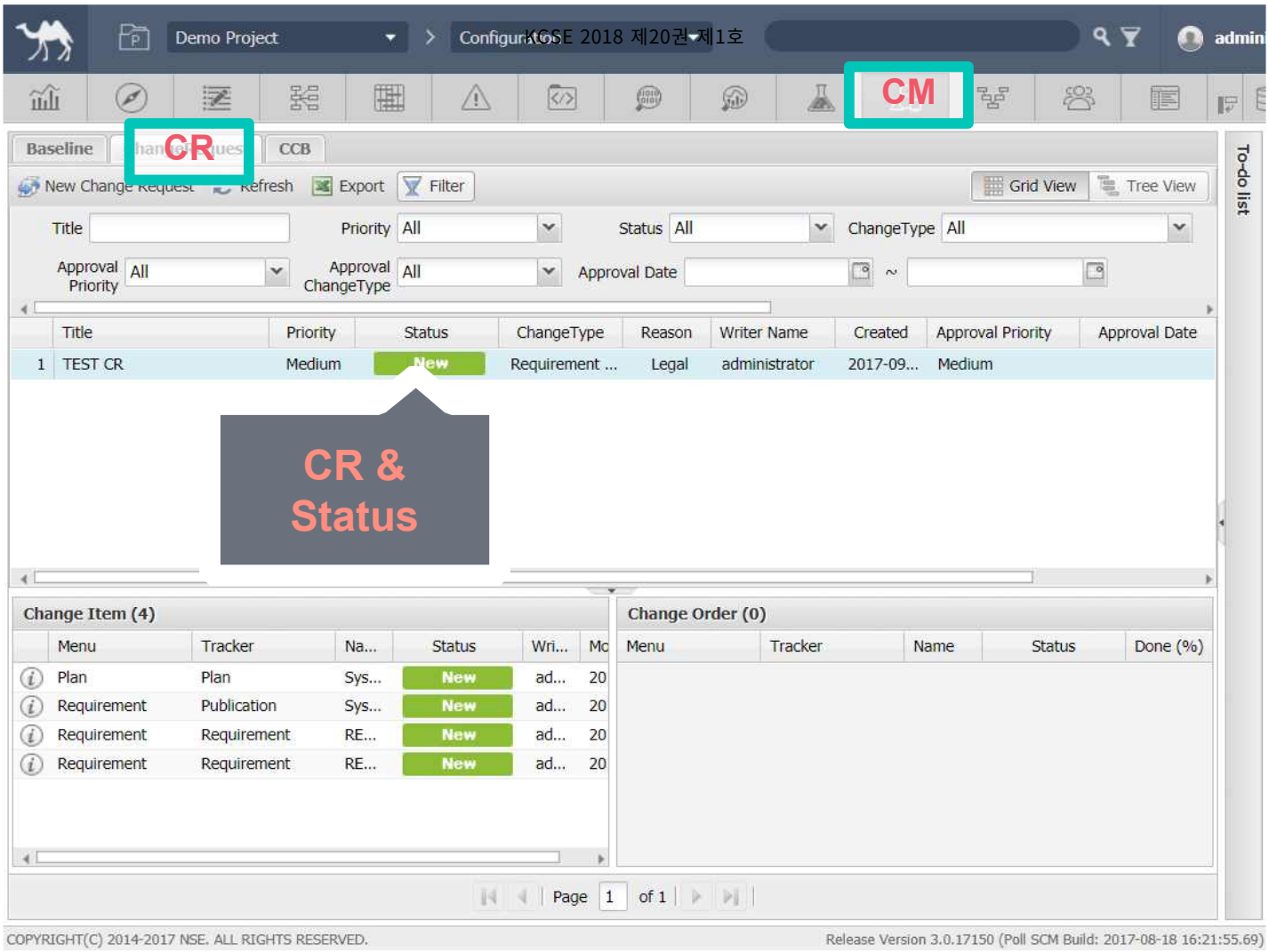
SIRIUS
admin
Demo_SIRIUS
수정본(4차)
Baseline
Dashboard
대시보드
공통 대시보드
개인 대시보드
위젯 추가
레이아웃 정렬
요구사항 분석
구조설계
상세설계
구현
통합 및 시험
체계 통합 및 시험
이슈
신뢰성 시험
SFMEA
요구사항 분석
SM1 Defect/Fault Distribution
Stack/Group
신뢰성관리대상:
전체
Sort by
35
30
25
20
15
10
5
0
초안
수정본(1차)
수정본(2차)
수정본(3차)
수정본(4차)
Incorrect
Ambiguous
Incomplete
Inconsistent
Unverifiable
Untraceable
To-do List

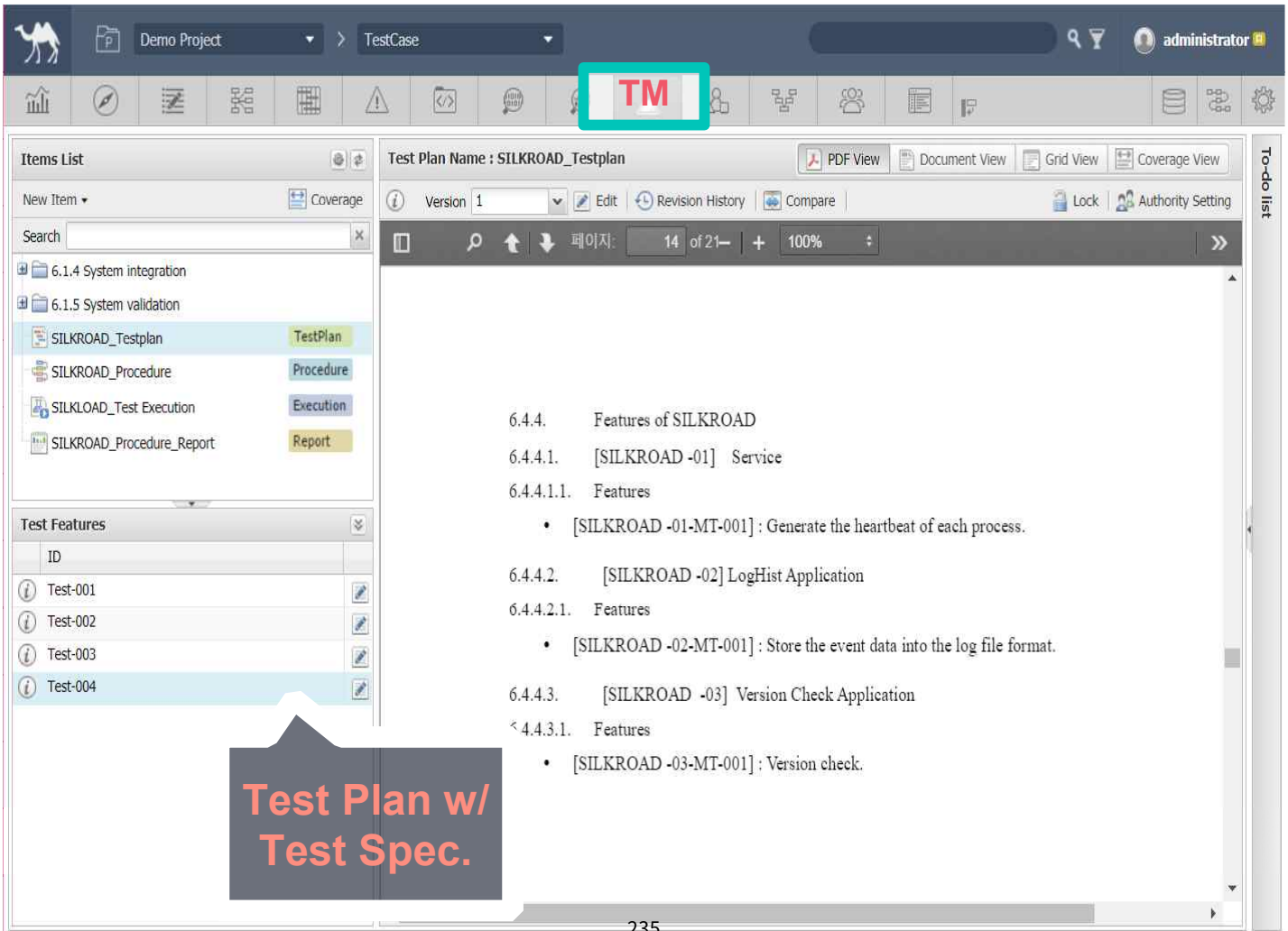

Grid View
Filter
Requirements
(w/ properties)
Documents List
Design Description
Requirement
Architecture
Design
Coding
Software Requirements Data [ver 1]
Trace Data [Template]
Software Verification Results(Refined) [Template]
Software Architecturaland design specification [5]
Design Description [3]
Trace Data(Refined) [Template]
Source Code [Template]
Category
Priority
Content
Old
SearchTag
Search
Reset Filter
Excel Export
ID
Modified
SUDP 001
SUDP 001
The main difference between software analysis and design is that the output of a software analysis consists of smaller problems to solve.
SUDP has a changes in its description.....
#End#
SUDP 002
Design examples include operation systems, webpages
SUDP 003
Design examples include operation systems, webpages, mobile devices or even the new cloud computing paradigm.
SUDP 004
Software design is the process by which an agent creates a specification of a software artifact, intended to accomplish goals, using a set of primitive components and subject to constraints.[1] Software design may refer to either "all the activity involved in conceptualizing, framing, implementing, commissioning, and ultimately modifying complex systems" or "the activity following requirements specification and before programming, as ... [in] a stylized software engineering process.
Safety
Functional
Critical
Medium
2017-11-08 17...
2017-11-08 16...
Page 1 of 1
Displaying 1 - 20 of 20
25
COPYRIGHT(C) 2014-2017 NSE. ALL RIGHTS RESERVED.
Release Version 3.0.17569 (Poll SCM Build: 2017-11-07 14:01:16.37)


Coverage
Expand/
Collapse
Software Architecturaland design specification
Vertical Coverage
Horizontal Coverage
Filter
Export
Design Description
ID
Content
SUDP 001 SUDP 001The main difference between software design is that the output of a software...
SUDP 002 Design examples include operation systems, webpages, mobile devices or even the ne...
SUDP 003 Design examples include operation systems, webpages, mobile devices or even the ne...
SUDP 004 Software design is the process by which an agent creates a specification of a software ...
SUDP 005 Software design is the process of implementing software solutions to one or more sets ...
SUDP 006 The main difference between software analysis and design is that the output of a softw...
SUDP 007 Software design is the process of implementing software solutions to one or more sets ...
SUDP 008 Software design is the process of implementing software solutions to one or more sets ...
SUDP 009 Software design is the process of implementing software solutions to one or more sets ...
SUDP 010 Software design is the process of implementing software solutions to one or more sets ...
SUDP 011 Software design is the process of implementing software solutions to one or more sets ...
SUDP 012 Software design is the process of implementing software solutions to one or more sets ...
SUDP 013 Software design is the process of implementing software solutions to one or more sets ...
SUDP 014 Software design is the process of implementing software solutions to one or more sets ...
SUDP 015 Software design is the process of implementing software solutions to one or more sets ...
SUDP 016 Software design is the process of implementing software solutions to one or more sets ...
SUDP 017 Software design is the process of implementing software solutions to one or more sets ...
SUDP 018 Software design is the process of implementing software solutions to one or more sets ...
SUDP 019 Software design is the process of implementing software solutions to one or more sets ...
SUDP 020 Software design is the process of implementing software solutions to one or more sets ...
Start SUDP 001
~ End SUDP 020
Count 20


Traceability
Check Link
Overview
Requirement - Traceability
Coverage
Hyperbolic Viewer
Expand All
Collapse All
Check Link
Search Keyword
To-do list
Requirement
Requirement
Software Requirements Data [ver 1]
Trace Data [Template]
Software Verification Results(Refined...
Architecture
Software Architecturaland design spe...
Design
Design Description [7]
Trace Data(Refined) [Template]
Software Verification Results(Refined...
Coding
Source Code [Template]
Trace Data(Refined) [Template]
Software Verification Results(Refined...
Software Requirement...
Software Architectural...
Design Description
SRS 001
SRS 002
SRS 003
SRS 004
SRS 005
SRS 006
SRS 007
SRS 008
SDD 001
SDD 002
SDD 003
SDD 004
SDD 005
SDD 006
SDD 007
SDD 008
SDD 009
SUDP 001
SUDP 002
SUDP 003
SUDP 004
SUDP 005
SUDP 006
SUDP 007
SUDP 008
SUDP 009
Trace Data
COPYRIGHT(C) 2014-2017 NSE. ALL RIGHTS RESERVED.
Release Version 3.0.17570 (Poll SCM Build: 2017-11-08 16:01:12.76)


DEMO Project_BGH
PeerReview
Peer Review
Design Description : [#PeerDocument.6939759]
Annotation List
Refresh
Filter
Expand All
Collapse All
Functional Annotations (20 Items)
SUDP 001
"olve.Design examples include operation systems, webpages, mobile devices or even the new cloudcom puting paradigm.SUDP has a changes in its descripti on....."
백가현
modification required. equation added
Edit
17-11-08 19:09
Reply
New comment
SUDP 002
SUDP 003
SUDP 004
SUDP 005
SUDP 006
SUDP 007
SUDP 008
SUDP 009
SUDP 010
SUDP 011
SUDP 012
SUDP 013
SUDP 014
SUDP 015
SUDP 016
SUDP 017
SUDP 018
SUDP 019
Order by : Location Id
Design Description
Version 7
Highlight
Pin
Hide
00 : 06 : 37
9 of 25
100%
Software Design Description
SUDP 001
The main difference between software design is that the output of a software analysis consists of smaller problems to solve.
Design examples include operation systems, webpages, mobile devices or even the new cloud computing paradigm.
SUDP has a changes in its description.....
FLIGHT
SUDP 001 - Relations
De...
Requirement
1
SDD 002 (Software Architecturaland design specification)
Data Management and DisplayController: Mediates communication between the sub-systems.
2
SRS 002 (Software Requirements Data)
Functional and operational requirements under each mode of operation.
SDD 002 - Requirement
Data Management and Display
Controller: Mediates communication be
COPYRIGHT(C) 2014-2017 NSE. ALL RIGHTS RESERVED.


Demo Project
CM
CR
Baseline
CCB
New Change Request
Refresh
Export
Filter
Grid View
Tree View
To-do list
Title
Priority
Status
ChangeType
Approval Priority
Approval ChangeType
Approval Date
Title
Priority
Status
ChangeType
Reason
Writer Name
Created
Approval Priority
Approval Date
1 TEST CR Medium New Requirement ... Legal administrator 2017-09... Medium
CR & Status
Change Item (4)
Menu
Tracker
Na...
Status
Wri...
Mc
Plan Plan Sys... New ad... 20
Requirement Publication Sys... New ad... 20
Requirement Requirement RE... New ad... 20
Requirement Requirement RE... New ad... 20
Change Order (0)
Menu
Tracker
Name
Status
Done (%)
Page 1 of 1
COPYRIGHT(C) 2014-2017 NSE. ALL RIGHTS RESERVED.
Release Version 3.0.17150 (Poll SCM Build: 2017-08-18 16:21:55.69)


Demo Project
TestCase
administrator
TM
Items List
New Item
Coverage
Search
6.1.4 System integration
6.1.5 System validation
SILKROAD_Testplan TestPlan
SILKROAD_Procedure Procedure
SILKLOAD_Test Execution Execution
SILKROAD_Procedure_Report Report
Test Features
ID
Test-001
Test-002
Test-003
Test-004
Test Plan Name : SILKROAD_Testplan
PDF View
Document View
Grid View
Coverage View
Version 1
Edit
Revision History
Compare
Lock
Authority Setting
페이지: 14 of 21
100%
6.4.4. Features of SILKROAD
6.4.4.1. [SILKROAD -01] Service
6.4.4.1.1. Features
• [SILKROAD -01-MT-001] : Generate the heartbeat of each process.
6.4.4.2. [SILKROAD -02] LogHist Application
6.4.4.2.1. Features
• [SILKROAD -02-MT-001] : Store the event data into the log file format.
6.4.4.3. [SILKROAD -03] Version Check Application
6.4.4.3.1. Features
• [SILKROAD -03-MT-001] : Version check.
Test Plan w/ Test Spec.
To-do list
COPYRIGHT(C) 2014-2017 NSE. ALL RIGHTS RESERVED.
Release Version 3.0.17150 (Poll SCM Build: 2017-08-18 16:21:55.69)

Software Design

PURPOSE

- Decompose the software requirements into major functional areas and describe how these components interact with each other.
- Establishes the basis for software implementation and coding effort.

SWQ Roles, Responsibilities & Major Activities

- Peer Review(PR) SW Design
- Ensure Traceability
- Start Test procedure development (IVV PR)
- Verify SW safety/mission critical components are testable
- SW FMECA/FTA

Prediction Model

SFMEA

Data Management

Dashboard

Requirement Management

Traceability Management

Peer Review Management

Functional Test Management

SIRIUS

admin

SFMEA

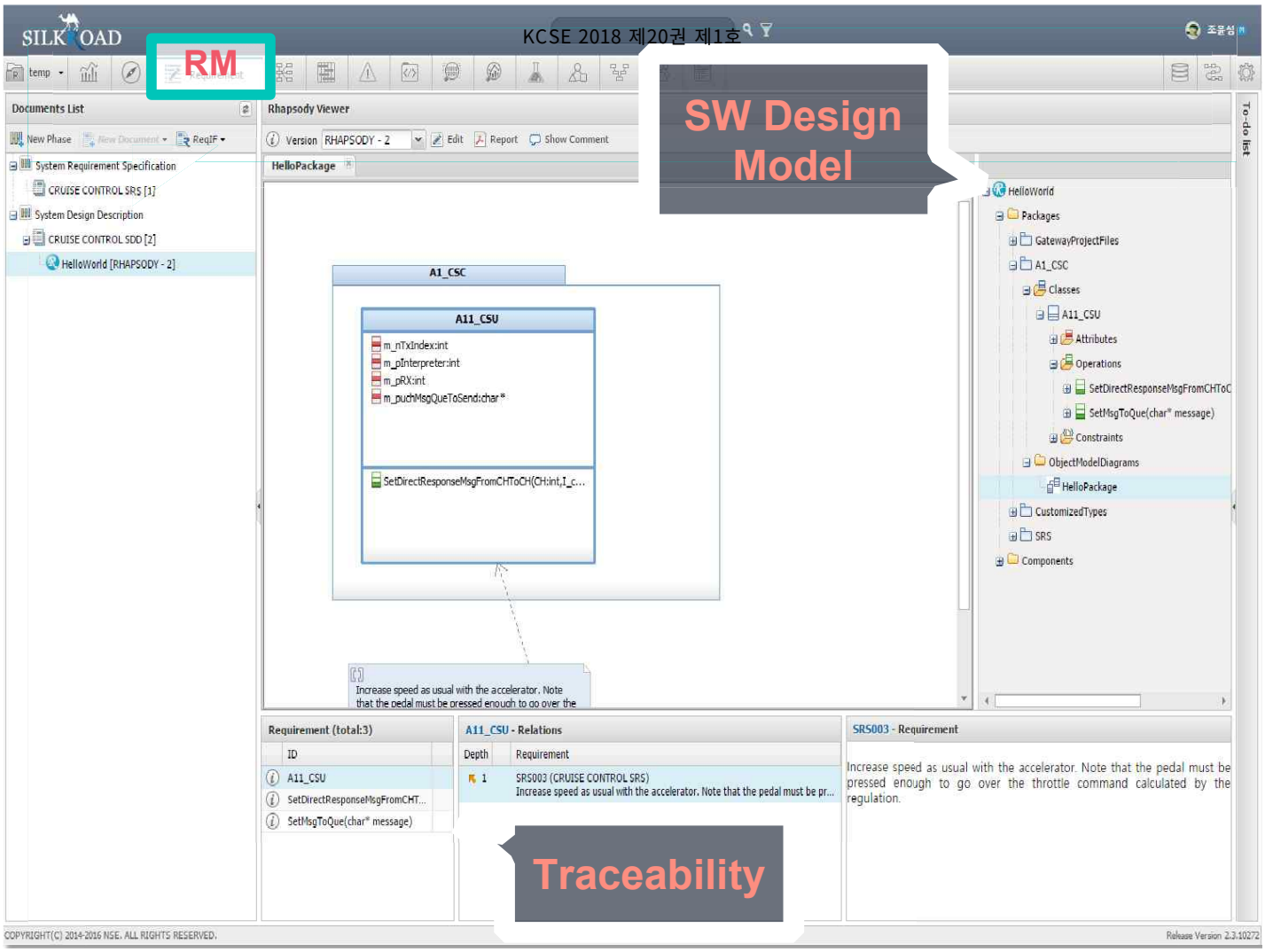
Demo_SIRIUS | 수정본(4차) | Baseline

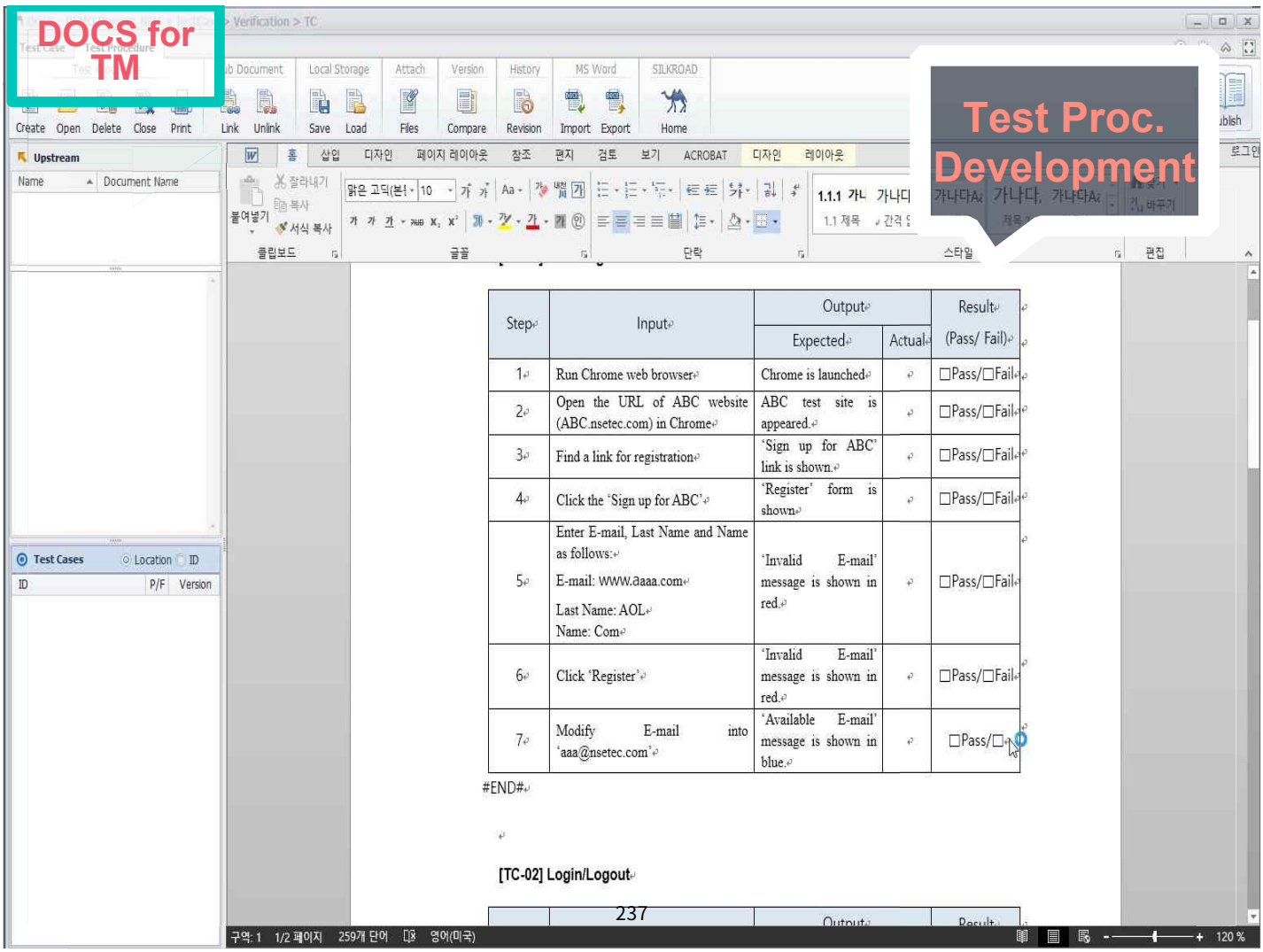
- 대시보드
- 요구사항 분석
- 구조설계
- 상세설계
- 구현
- 통합 및 시험
- 체계 통합 및 시험
- 이슈
- 신뢰성 시험
- SFMEA

Import Excel

CSCI Name	Functional									Interface						
	RPN			Likehood			Severity			RPN			Likehood			
	Ave	Min	Max	Ave	Min	Max	Ave	Min	Max	Ave	Min	Max	Ave	Min	Max	Ave
ACSCI	7	4	12	3	2	4	2	2	3	8	0	18	3	0	6	3
BCSCI	2	0	6	3	0	2	4	1	3	9	8	15	6	2	5	5

To-do List





Step	Input	Output		Result (Pass/ Fail)
		Expected	Actual	
1	Run Chrome web browser	Chrome is launched		□Pass/□Fail
2	Open the URL of ABC website (ABC.nsetec.com) in Chrome	ABC test site is appeared.		□Pass/□Fail
3	Find a link for registration	'Sign up for ABC' link is shown.		□Pass/□Fail
4	Click the 'Sign up for ABC'	'Register' form is shown		□Pass/□Fail
5	Enter E-mail, Last Name and Name as follows: E-mail: WWW.daaa.com Last Name: AOL Name: Com	'Invalid E-mail' message is shown in red.		□Pass/□Fail
6	Click 'Register'	'Invalid E-mail' message is shown in red.		□Pass/□Fail
7	Modify E-mail into 'aaa@nsetec.com'	'Available E-mail' message is shown in blue.		□Pass/□

#END#

[TC-02] Login/Logout

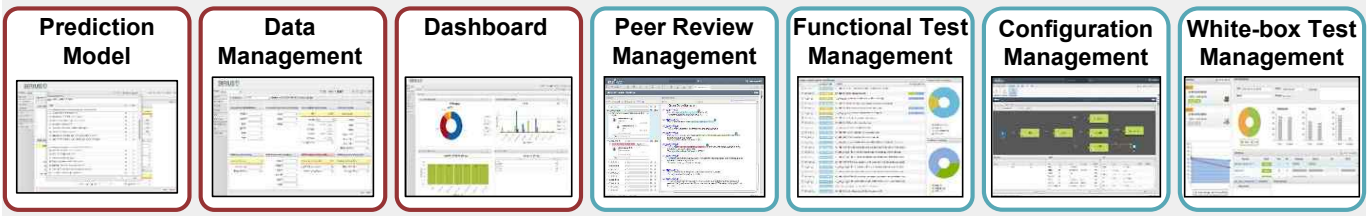
Software Implementation & Coding

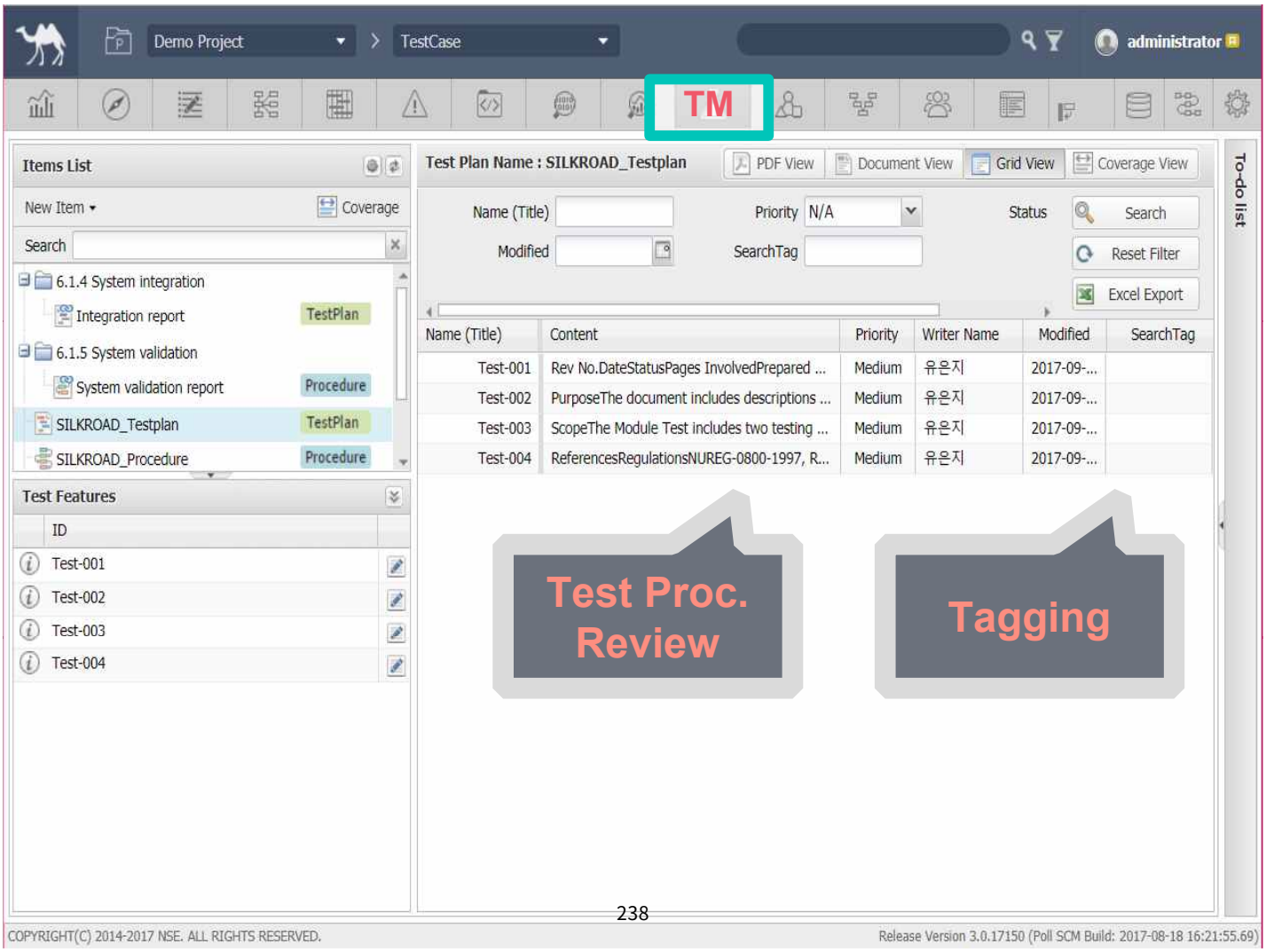
PURPOSE

- Implements the software design.
- Code is written at the lowest unit level and tested to remove defects. For example, code languages include, Ada, C, C++, Java, PLC, etc.

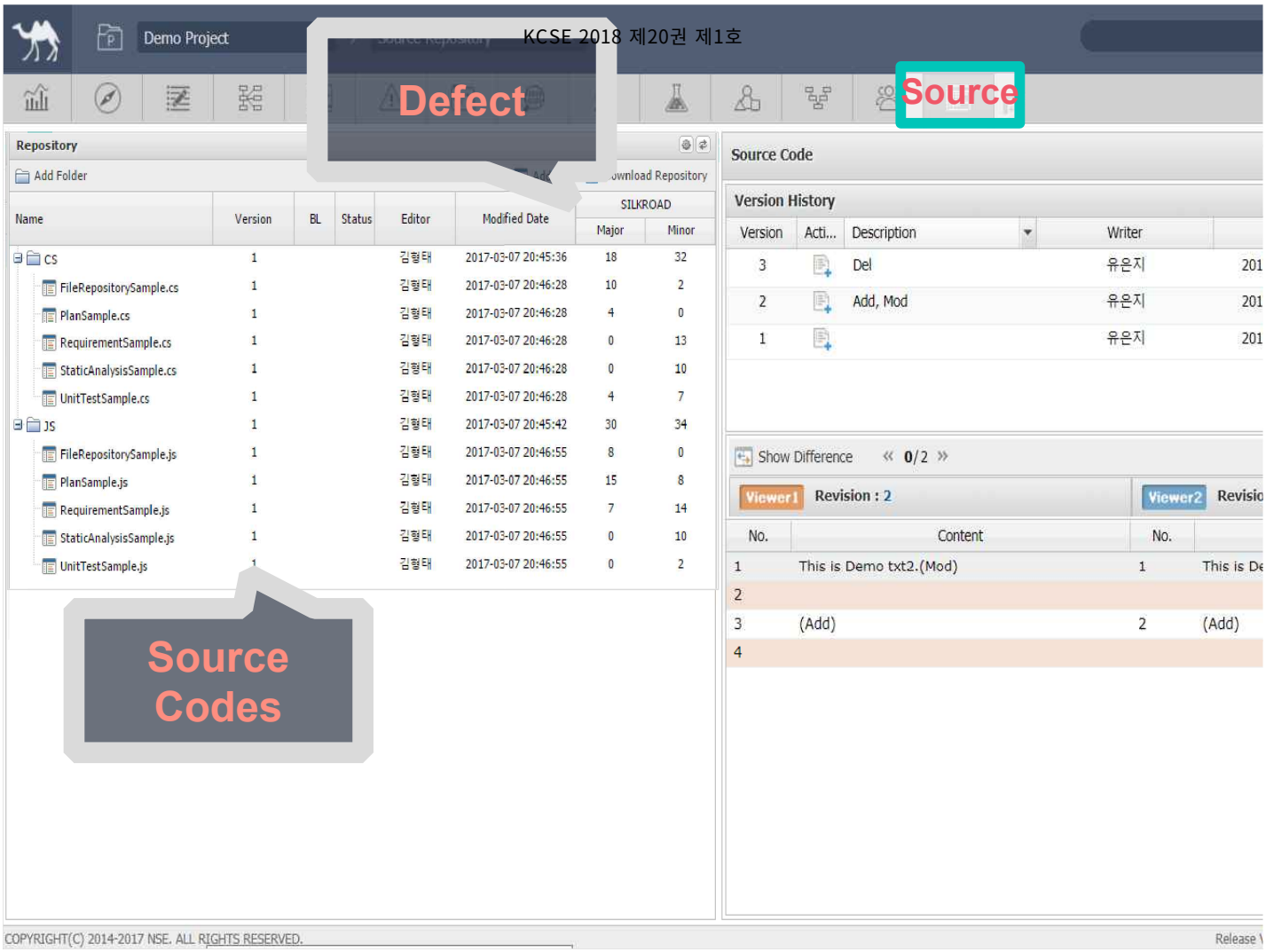
SWQ Roles, Responsibilities & Major Activities

- Safety/mission critical test procedures tagged
- Verify Safety/mission critical code is tagged
- Finalize Test procedures (IVV PR)
- PCR tracking
- Review unit test results





COPYRIGHT(C) 2014-2017 NSE. ALL RIGHTS RESERVED.



Integration & Test

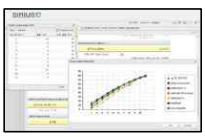
PURPOSE

- The software is assembled/build-up from the lowest unit level into the overall software system.
- Interfaces are checked among design components and defects are removed prior to acceptance testing.

SWQ Roles, Responsibilities & Major Activities

- SWR MCF Maturity Assessment
- Conduct Engineering evaluation (EE)
- PCR tracking - DRB
- Validate test tools

Estimation Model



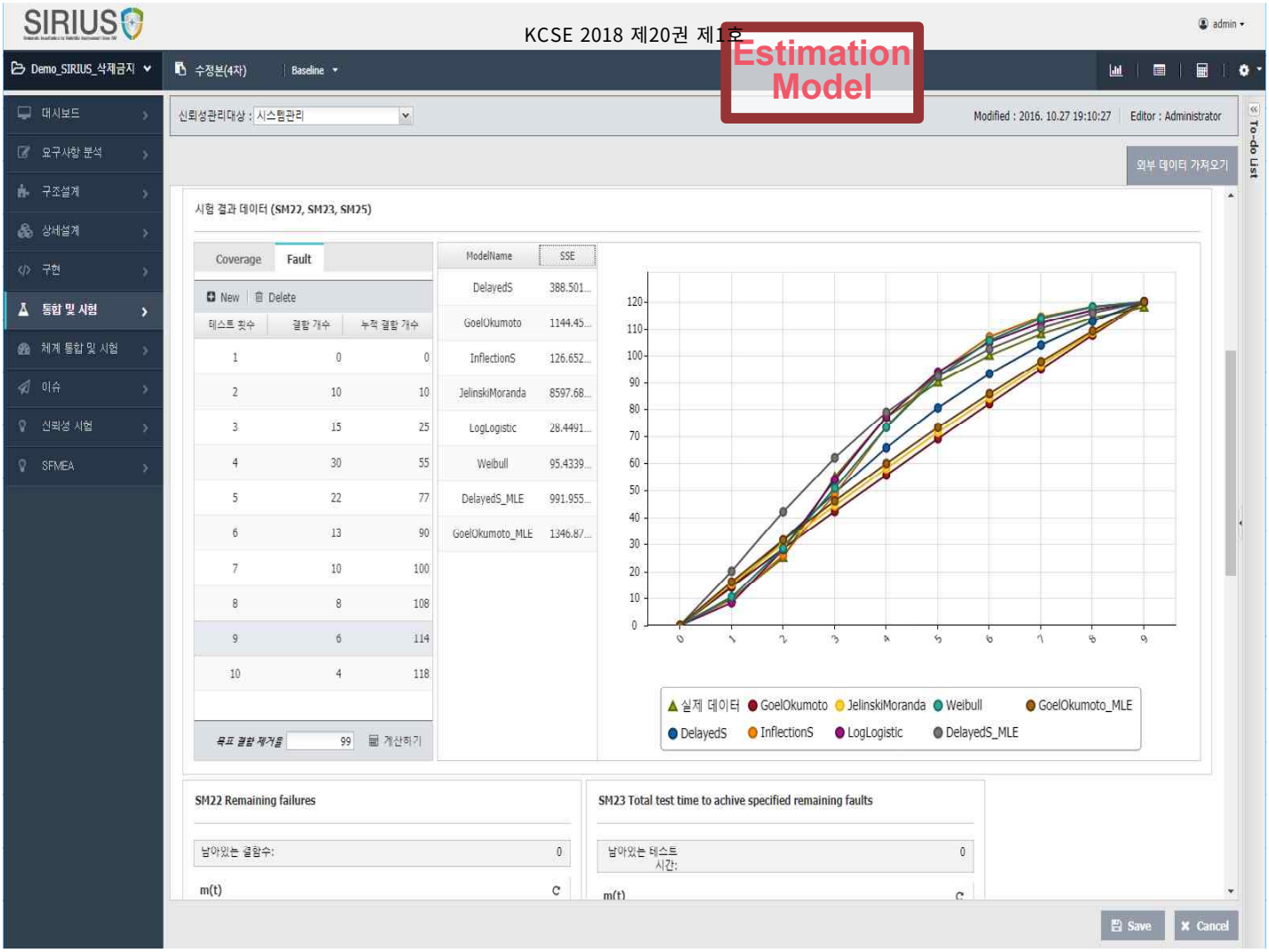
Data Management

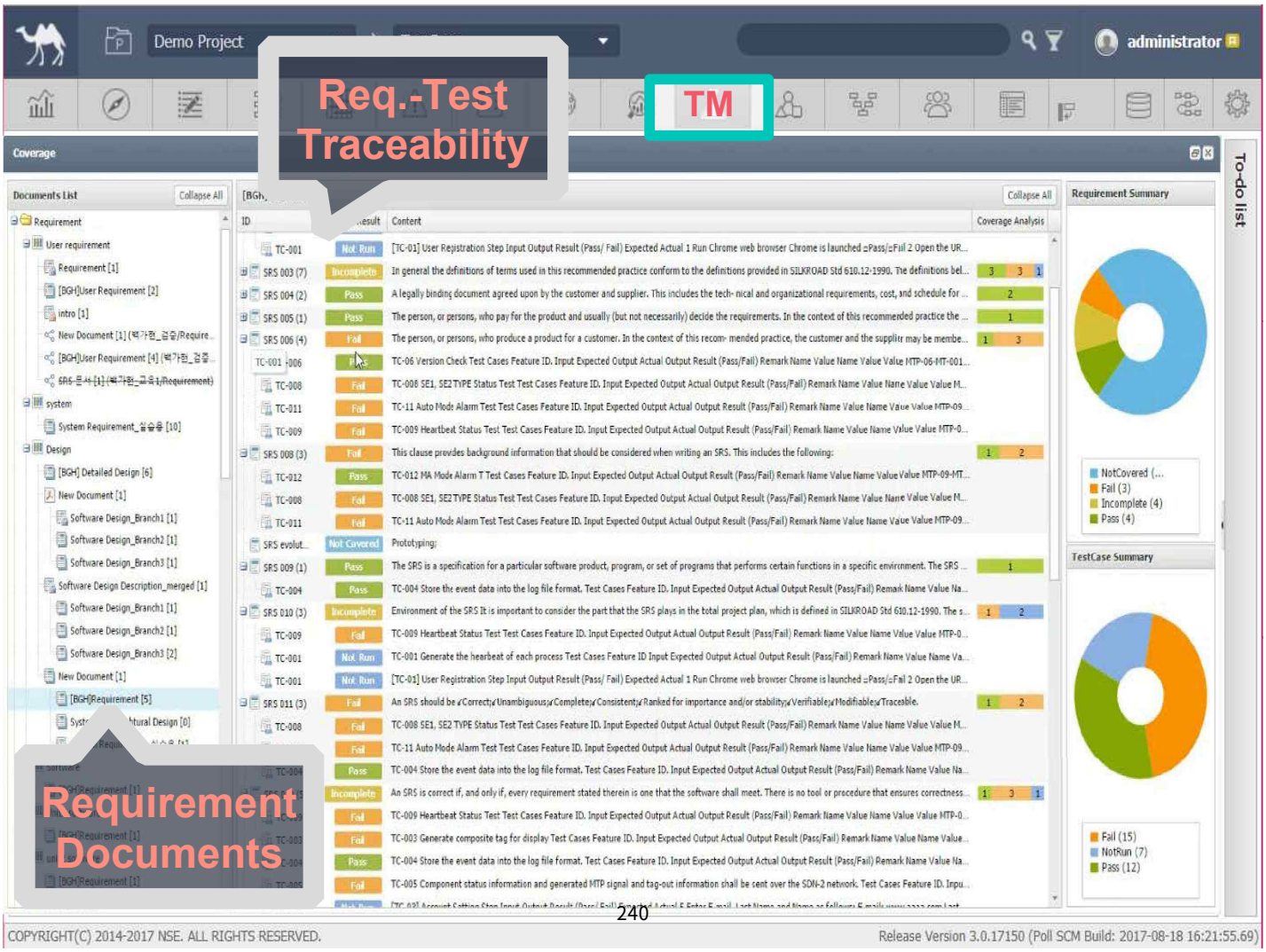
Dashboard

Functional Test Management

Configuration Management

White-box Test Management





COPYRIGHT(C) 2014-2017 NSE. ALL RIGHTS RESERVED.

Software Testing

PURPOSE

- Tests the final software against its technical and performance requirements on the production hardware.

SWQ Roles, Responsibilities & Major Activities

- SWR MCF Maturity Assessment
- SWR Metrics Analysis
- Conduct Test Readiness Review (TRR)
- Conduct Dry Run, FQT & regressions – IVV witness
- PCR tracking - DRB
- Develop Test Report (IVV PR)

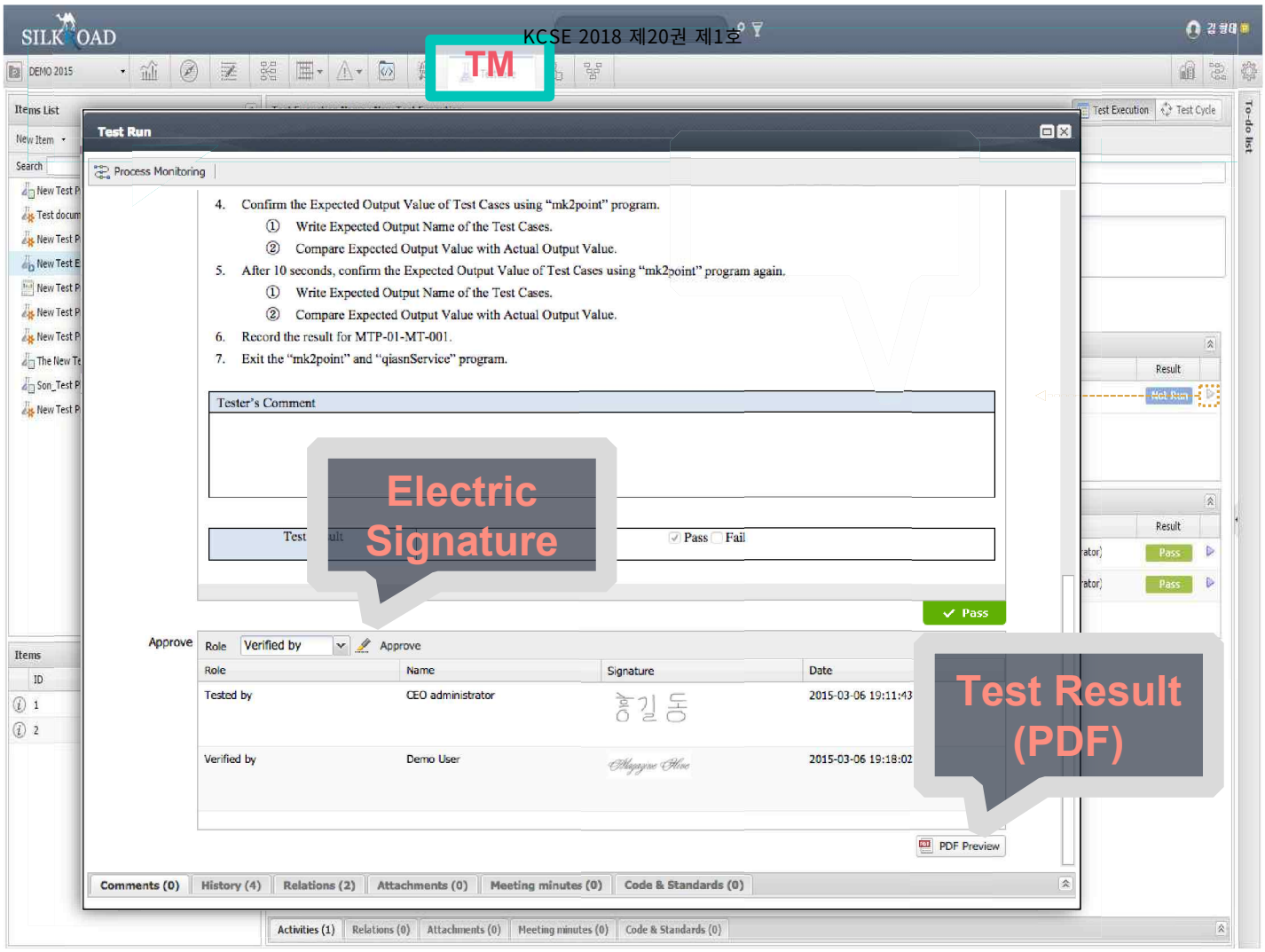
Estimation Model

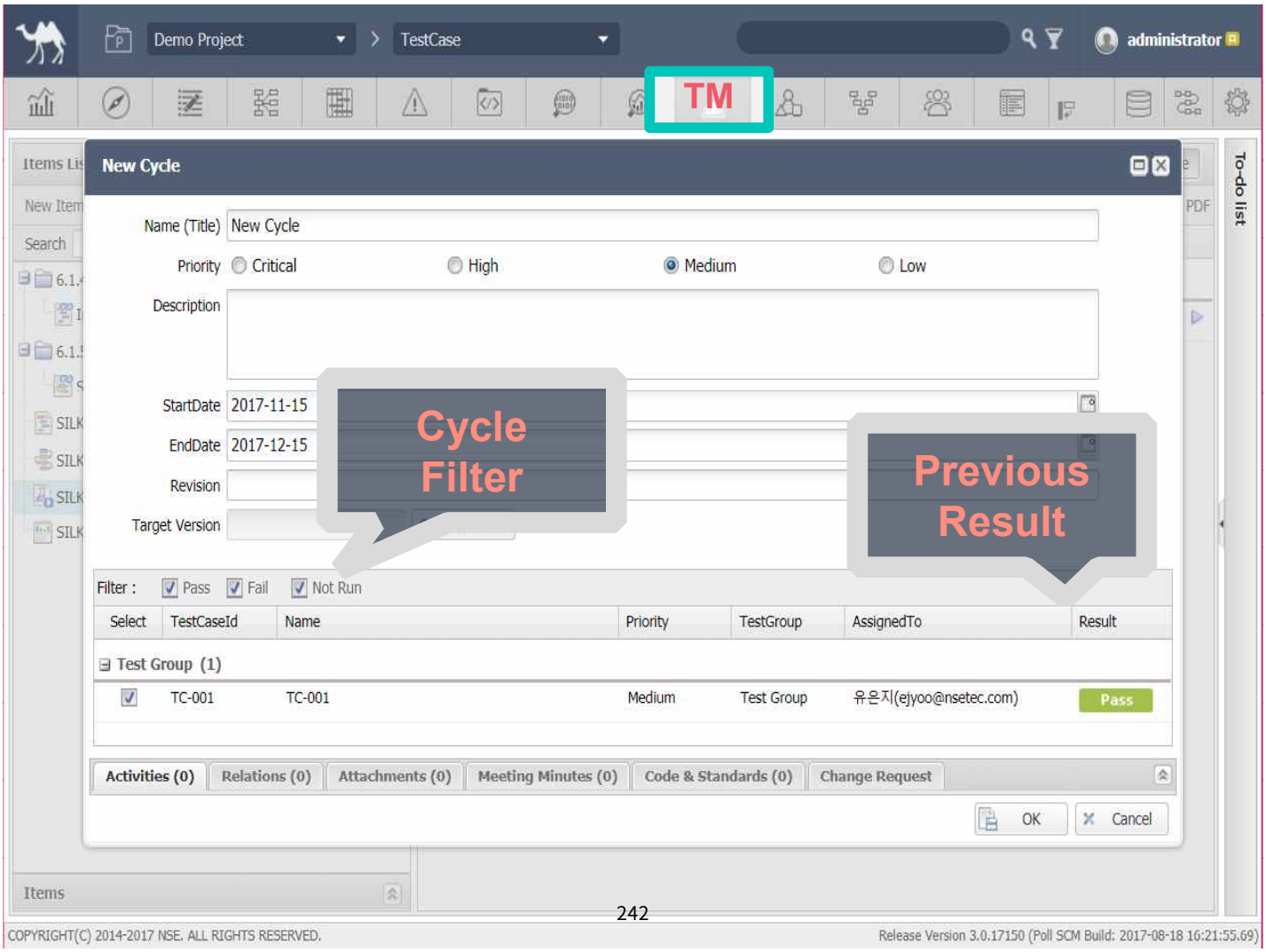
Data Management (Baseline)

Dashboard

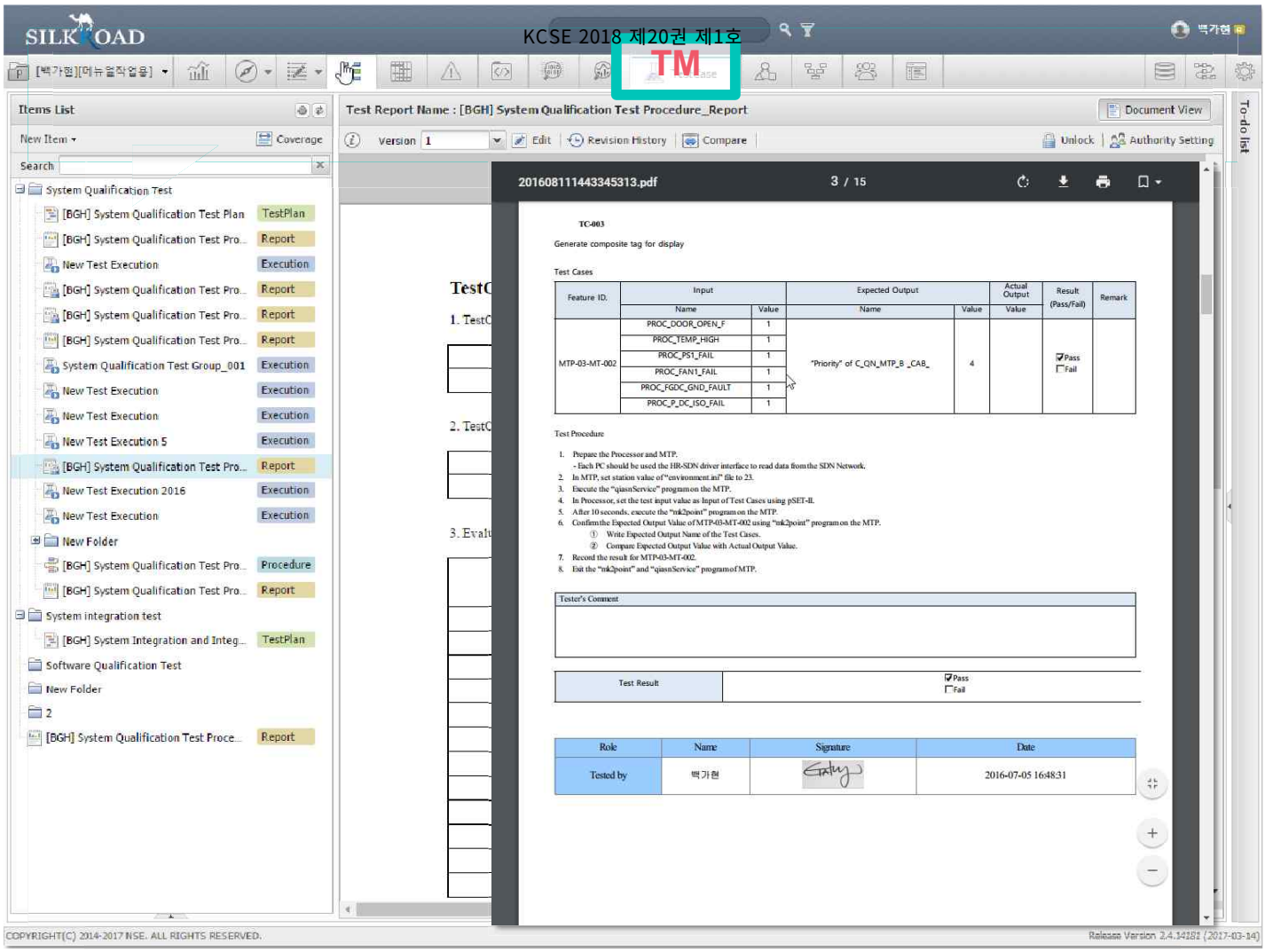
Functional Test Management

Configuration Management





COPYRIGHT(C) 2014-2017 NSE. ALL RIGHTS RESERVED.
Release Version 3.0.17150 (Poll SCM Build: 2017-08-18 16:21:55.69)



Software Baseline

PURPOSE

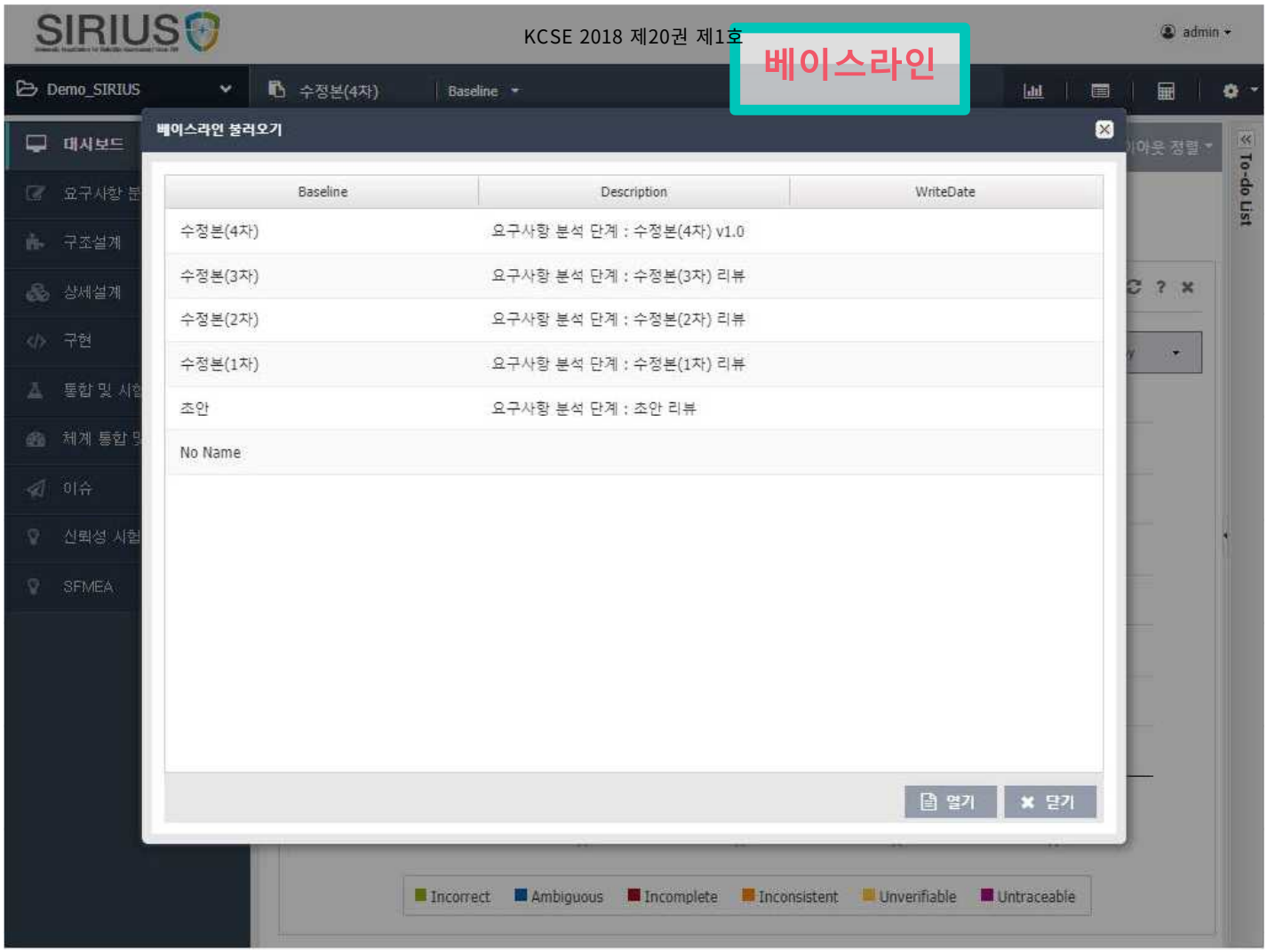
- The software is baselined via a formal release of the software version description and put under configuration control.
- Provides an overall risk assessment of software safety, suitability, supportability and readiness to be released to the field.

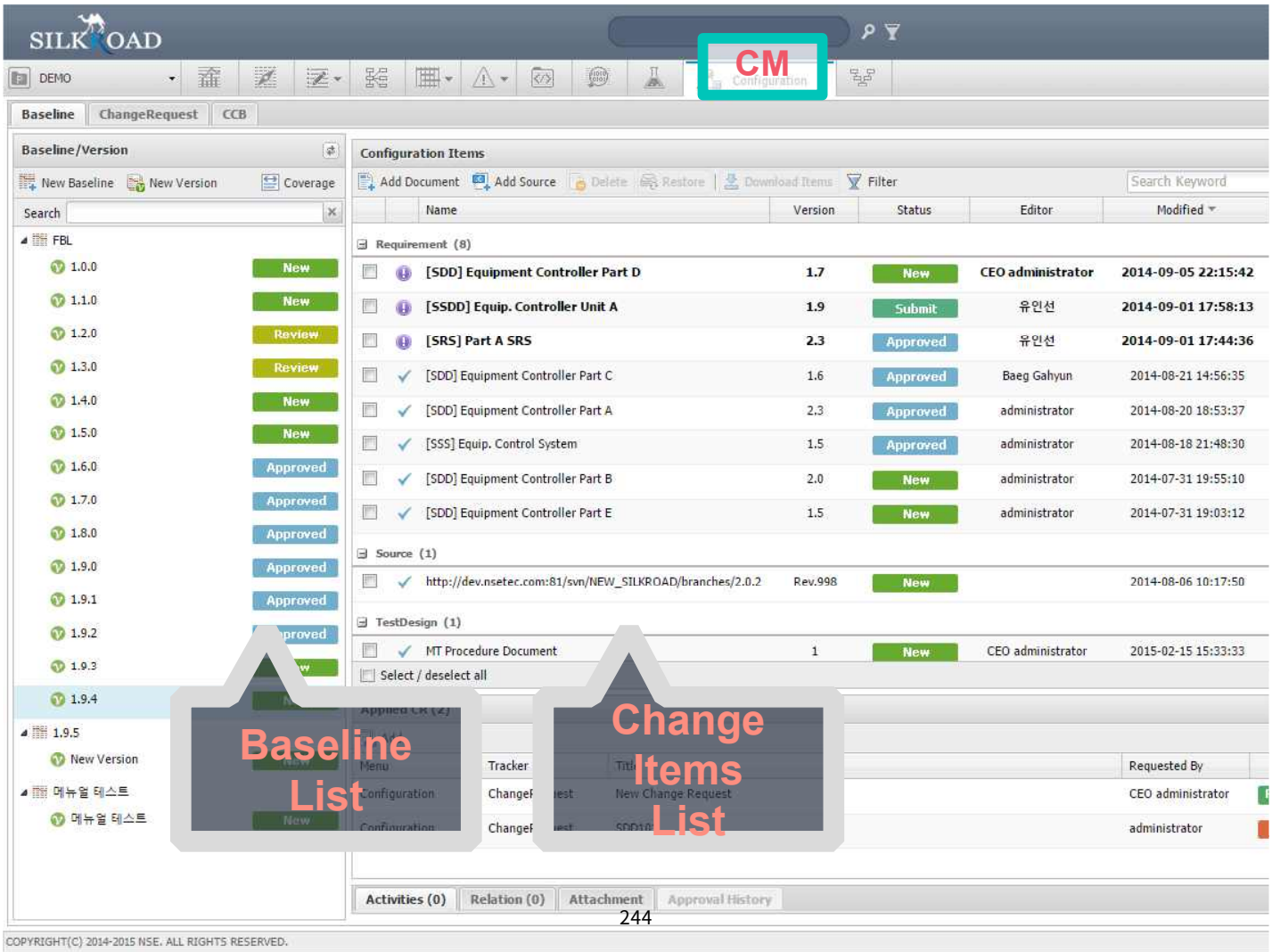
SWQ Roles, Responsibilities & Major Activities

- Chair the SW Subgroup (SSG)
- Conduct Functional Configuration Audit / Physical Configuration Audit (FCA/PCA)
- Develop SW Qualification Statement (SQS)

Data Management

Configuration Management



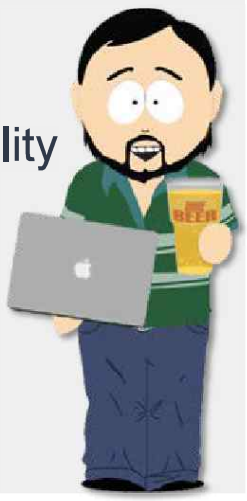


COPYRIGHT(C) 2014-2015 NSE. ALL RIGHTS RESERVED.

결론

결론

- ARDEC (미 육군 무기연구, 개발 및 엔지니어링 센터) QE&SA SW Development Guidelines의 SW Reliability Engineer 업무 분석
- SW요구사항 분석/평가, 시스템 및 프로세스의 검증 수행을 통한 위험 완화 및 SW 신뢰성 강화
- 도구를 통한 SW개발 전(全) 단계의 SW Reliability Engineer 업무 효율 증대



*ARDEC : U.S. ARMY Armament Research, Development and Engineering Center

Thanks!

Any questions?

Code X-Ray:
LG Electronics' code quality inspection platform

Milhan KIM[O], Jewhi Ryu, Heewon Choi
Software Center, LG Electronics
{milhan.kim, jay.ryu, heewon.choi}@lge.com

Abstract

In LG Electronics, we developed novel static analysis platform called Code X-Ray to improve quality of source codes. Code X-Ray is successfully adopted by many projects in the enterprise. While many COTS and OSS analyzers are available in the market and provide rich features, we decided to build another one to meet our own needs. In this paper, we describe why and how we designed Code X-Ray and how we use it as a platform rather than a static analysis tool.

1. Introduction

Code defect is one of the most frequent and impacting types of the software defect. And we also found that code defect is highly effective and cheap type of software defect in terms of handling. In a survey of independent research institution, it is known that among top 3 origins of low impact software defect, more than 50% of defects are from code defect [Capers, Hiller]. Such trend is also valid for high impact defects (>30%) and massive-issue-resulting defects (>40%). On the other hands, code defect has highest DRE (defect removal efficiency).

$$\text{DRE} = \frac{Defects\ removed\ during\ a\ development\ phase}{Defects\ latent\ in\ the\ product\ at\ that\ phase}(\%)$$

Equation 1. Definition of *DRE*

And it also has one of the cheapest removal cost per defect.

To handle such code defect, static analysis and code inspection, reaching 95% in terms of *DRE* in best case and 60% in average, is known to be best performing method among 7 different category of methods: static analysis & code inspection, detailed logic review, system test, detailed function design review, integration test, external beta test, unit test.

While using a static analyzer to handle code defect seems promising method to improve software quality in high cost-effective fashion, we failed to find tools that meet our criteria.

2. Why we ended up building our own static analyzer

In current software market, there is a plethora of COTS (commercial off-the-shelf software) and OSS (open source software) static analysis tools. None of them, however, met our needs as far as we know.

First of all, we wanted tools with high usability to increase usage by developers. We wanted well-polished UI and UX so our developers use static analyzer painlessly. In the current market, many of the tools usually require specific environment settings and comprehensive understanding of static analyzer. Some tools do not force such overhead, but they require noticeable compromise in terms of accuracy. And importantly, we wanted our tool to be integrated into developers' workflow not disrupting it, which is one of the many reasons why developers are not using analyzers. [Johnson]

Second, we wanted our thousands of developer to perform static analysis in high frequency, every code save for example, so the discovery of code defects are not delayed to later than the coding phase. Some very powerful COTS supports incremental and laptop analysis these days, but maintaining such infrastructure comes at a high cost in both computing and monetary aspect.

Third, while conventional tools are very powerful in finding wrong codes, we couldn't search a tool which can find a not-a-clean code. We wanted to track the quality of codes in terms of

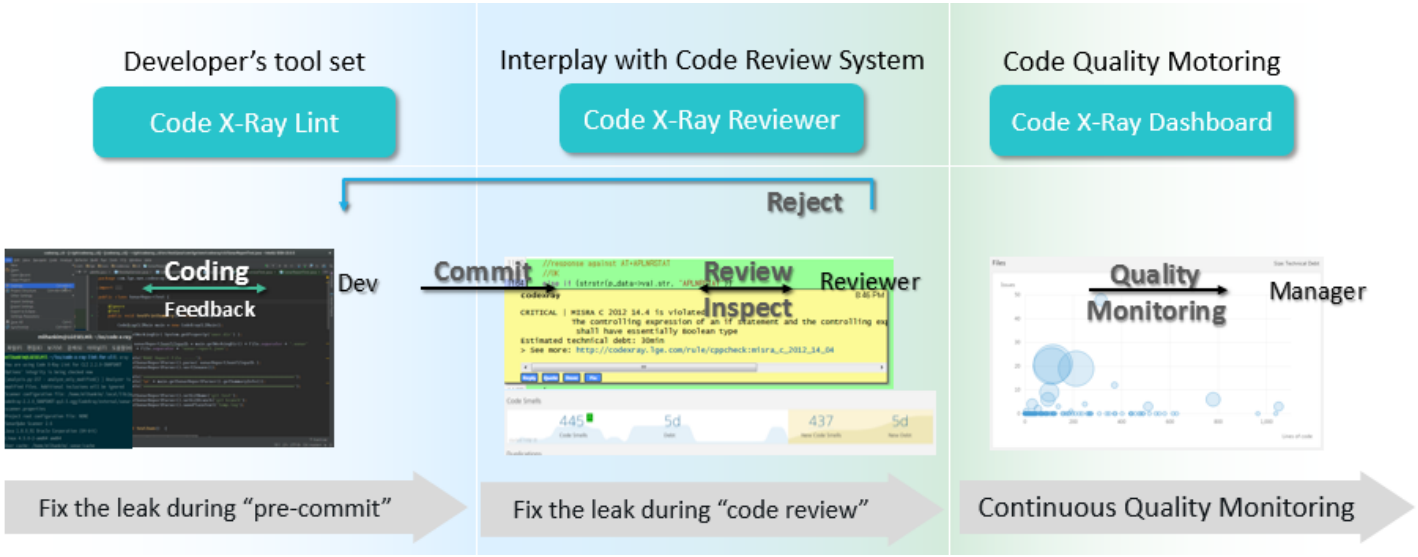


Figure 1. Code X-Ray Workflow

technical debt, but even top COTS solutions do not provide such functionality.

So we designed and built our own static analysis toolset called Code X-Ray.

3. How we use Code X-Ray as a platform

We designed Code X-Ray to encompass as many as possible aspects of developer's workflow. Code X-Ray continuously tracks code quality from IDE, terminal, code review to every *git* snapshot. Also, Code X-Ray analysis can be done distributed fashion. In other words, analysis can be done in dedicated analysis server, developer's laptop, or in CI server.

Also to provide static analysis as a platform, we maximized convenience by requiring no configuration. Code X-Ray stores all necessary information to analyze source code at the central server, and distributes and synchronizes them across every tool in each coding stage.

To provide static analysis as a platform, we service 3 different interfaces:

- Code X-Ray Lint: Developer tool set. Used in code writing stage.
- Code X-Ray Reviewer: Code review system (such as *Gerrit*, *BitBucket* or *GitLab*) integration. Used in code review stage.
- Code X-Ray Dashboard: Code quality tracker. Used in maintenance stage or by managers.

Code X-Ray Lint is a set of developer tools which includes CLI/GUI standalone application, and IDE plugins. And Code X-Ray Lint also work as a succinct *git* client. Since Code X-Ray Lint requires no configuration, users can simply perform incremental analysis on laptop by typing

$ xray

or clicking "Analyze" button. Also, developers can submit their code by a simple operation to the code review system and open pull request. Code X-Ray Lint is designed to emit result within several seconds (incremental analysis).

Figure 2. Code X-Ray Lint (GUI Application)

Code X-Ray Reviewer is a tool for developers and reviewers. It interplays with *Gerrit*, *BitBucket* or *GitLab* code review system. It automatically analyzes every new review, and present new issues as an inline comment in code review system. Code X-Ray Reviewer not only automates simple code review process but also bolsters code review itself since human reviewers can focus on real problem, not trivial errors and mistakes.

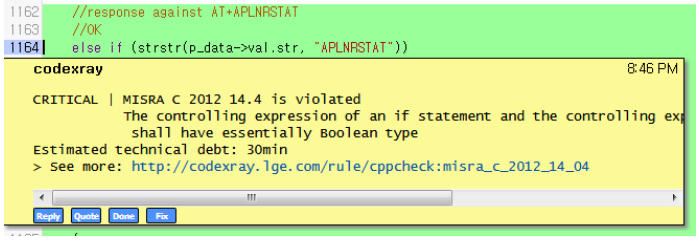


Figure 3. Inline comment of Code X-Ray Reviewer

Code X-Ray Dashboard is a web interface for quality inspection of source code. We provide comprehensive metrics such as technical debt to understand current (or former) status of code calculation. And these dashboards are updated at every revision of major branches.

To meet our non-functional requirements such as "within seconds/minutes analysis", we are exploiting flexible computing power. While overall systems are based on hundreds-of-core scale *OpenStack* infrastructure, we scale-in/out to the thousands to meet rush hour loads.

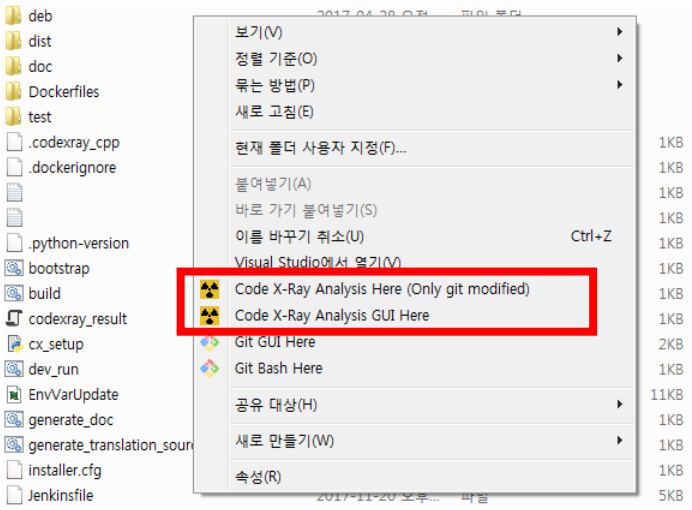


Figure 4. Shell integration

Code X-Ray is currently supporting following environments:

- Languages
  - *C, C++, MISRA C 2012, MISRA C++ 2008*
  - *Java, C#*
  - *JavaScript, Python, php*
- Integrated development environment
  - *Eclipse*
  - *IntelliJ IDEA* and its family (such as *Android Studio*, *PyCharm*, *WebStrom*)
- Operating System
  - *Windows*
  - *Linux*
  - *MacOS*

4. Limitation and compromises

In designing Code X-Ray, we needed to make a compromise between accuracy and run-time performance. Rather than developing a betwixt and between product, we decided to make a fast and accessible product. We chose run-time performance over accuracy and precision over recall.

Many COTS analyzers are providing a very high recall and thorough analysis to find code defect which can cause serious problems such as system crash. But such accuracy comes with cost called false positives. We minimized false positives by the help of numerous feedback from our big user pool. Even some false positives in corner case required us to remove checkers.

Powerful COTS analyzers are still being used as a software quality inspector in LG Electronics. At the same time, Code X-Ray is working as a code quality inspector. By helping developers not making mistakes in coding and technical debts, we witnessed several cases which Code X-Ray helps to relieve time spent in handling COTS analyzer issues.

5. Conclusion and further work

Code X-Ray is currently performing more than 400 analysis per hour 24/7 inside LG Electronics. We analyze every footprint of almost all production-level codes of LG Electronics. We keep providing analysis service within seconds/minutes fashion, as we advocate in the earlier. Projects of various sizes are analyzed in a timely manner every seconds. In our company, Code X-Ray works as a crucial part of software development infrastructure. We have witnessed numerous cases which Code X-Ray actually saves developer's time by helping not to make mistakes in the early stage of coding practice.

In Software Center of LG Electronics, we are managing and servicing most part of SCM (source code management) systems. We are currently planning to make Code X-Ray as a default feature of our SCM infrastructure so all repositories exploit Code X-Ray services. By doing so, we expect we would achieve quality management of source code across enterprise.

With Code X-Ray, we can gather a wide range of dataset to improve developers work. For example, we found some meaningful correlations between some metrics. Since Code X-Ray generates a very rich set of analysis data, we work on many

MSR (Mining Software Repositories) research with the dataset. We are currently working on calculating useful information such as quality of commit or efficacy of coding rule. We are also planning to build a visualization and measurement platform to bolster our empirical software engineering research in near future.

References

[Capers] Capers Jones, Software Quality in 2012: A survey of the state of the art, 2012

[Hiller] Hiller, Martin, Arshad Jhumka, and Neeraj Suri. “An approach for anlysing the propagation of data errors in software.”, 2001

[Johnson] Johnson, Bryant, et al. “Why don’t software developers use static analysis tools to find bugs?.”, 2013 35th International Conference on Software Engineering. IEEE, 2013.

ISO 26262 Test Method의 적용 방안과 효과

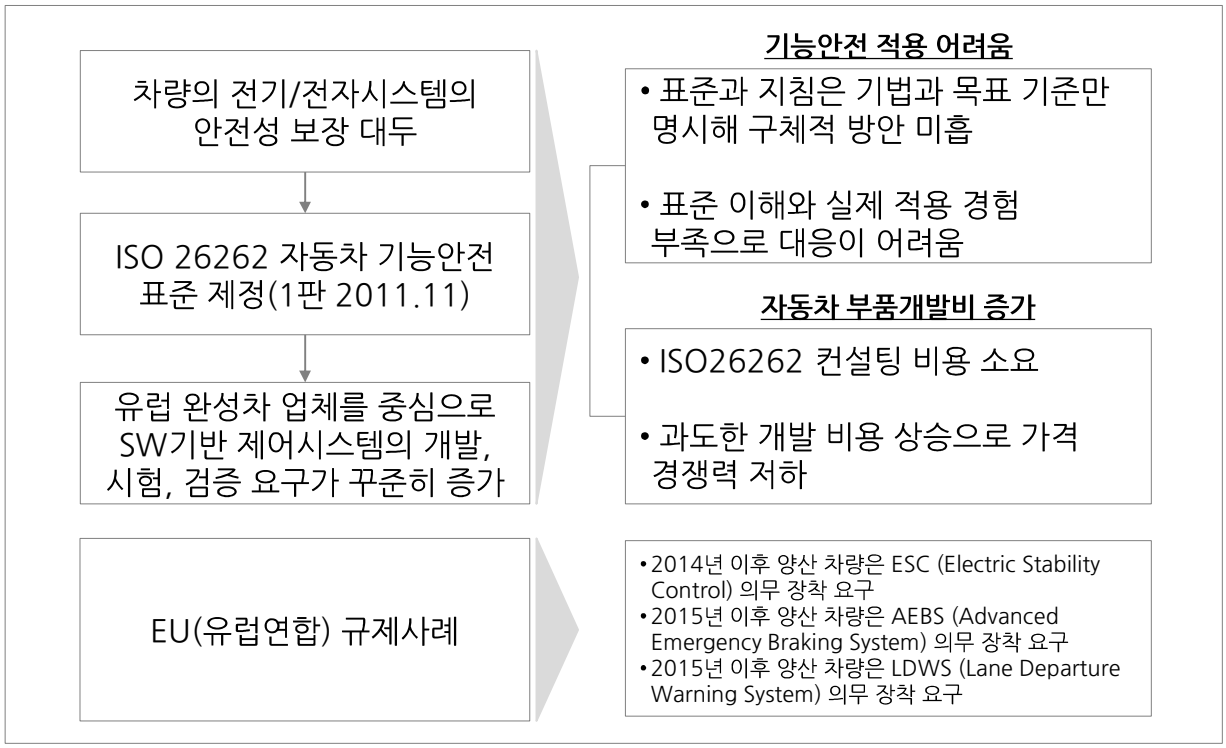
박찬혁,전수남,김태열(정보통신산업진흥원)
Stuart Reid,김선준(STA테스팅컨설팅)

2018 한국 소프트웨어공학 학술대회(KCSE 2018)
2018.01.29~01.31

I. 배경과 필요성
II. Test Method 이슈분석
III. Test Method 적용범위
IV. Test Method 적용방안
V. Test Method 가이드 도출
VI. 실무 적용과 효과 분석
VII. 결론
VIII. 향후 관련 연구

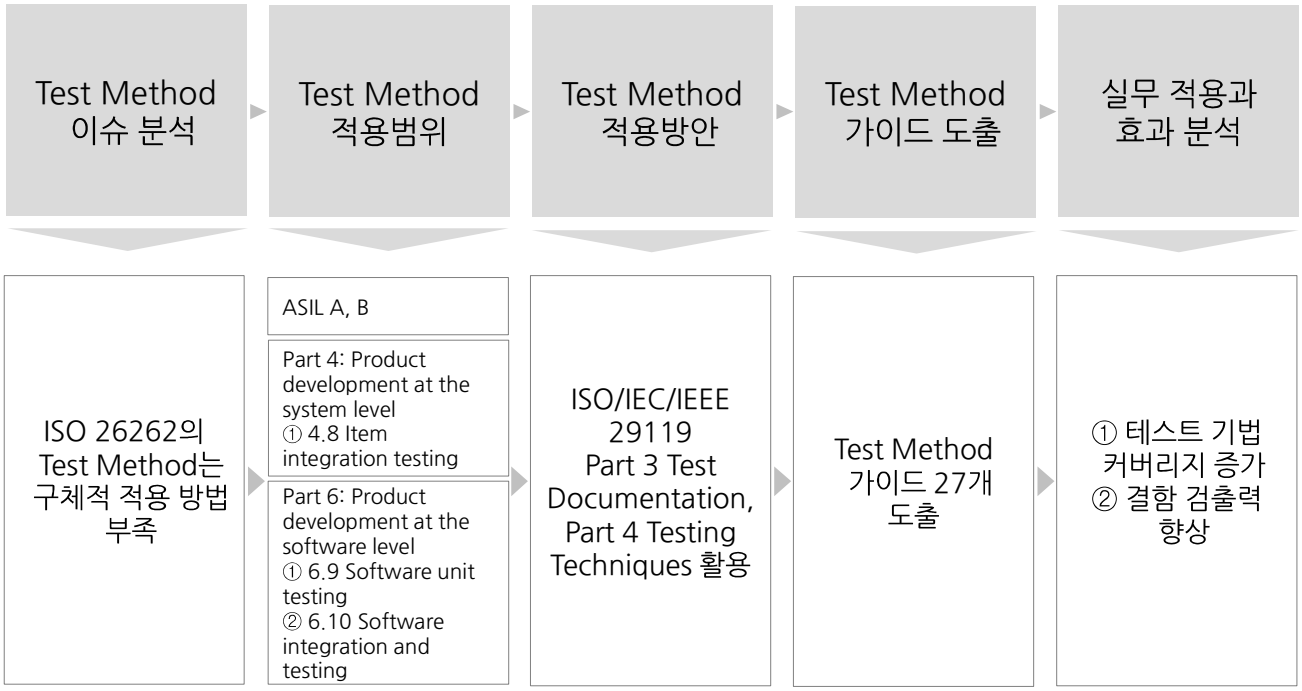
배경과 필요성

ISO 26262 자동차기능안전표준의 테스트 메소드(Test Method)의 구체적 적용 방안이 미흡해 이를 해결하고자 테스트 메소드 가이드를 만들고 실무에 적용해 효과를 확인함



요약

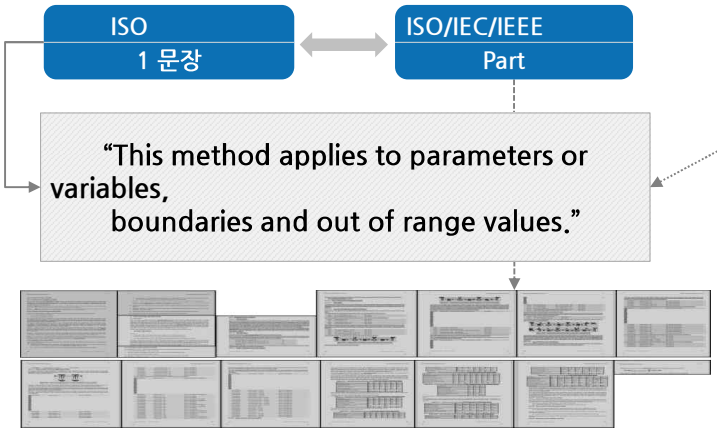
ISO 26262 테스트 메소드의 이슈를 분석하고 이슈 해결 범위를 선정함. 테스트 메소드 도출 방안을 ISO 29119을 활용해 테스트 메소드 가이드를 작성함. 현장 적용해 테스트 메소드의 효과를 확인함



Test Method 이슈분석

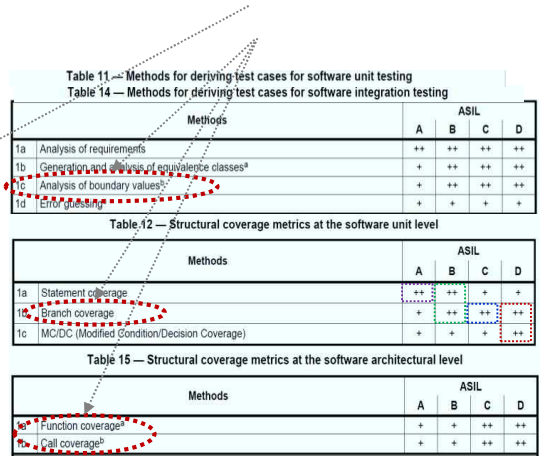
ISO 26262 Test Method는 구체적 적용 방법이 부족해 실무에서 사용이 어려워 활용 가이드가 필요함

예) 경계값분석)



<ISO 29119-4 Boundary Value Analysis(경계값 분석)>

Method의 구체적 적용방법 부족



ISO 26262, 4.8 Item integration testing -> '4.8 Table 4 Methods for deriving test cases for integration testing'-> '경계값 분석(Analysis of boundary values)' 기법을 한 문장으로 풀이함.

실무에서 사용이 어려워 활용 가이드가 필요

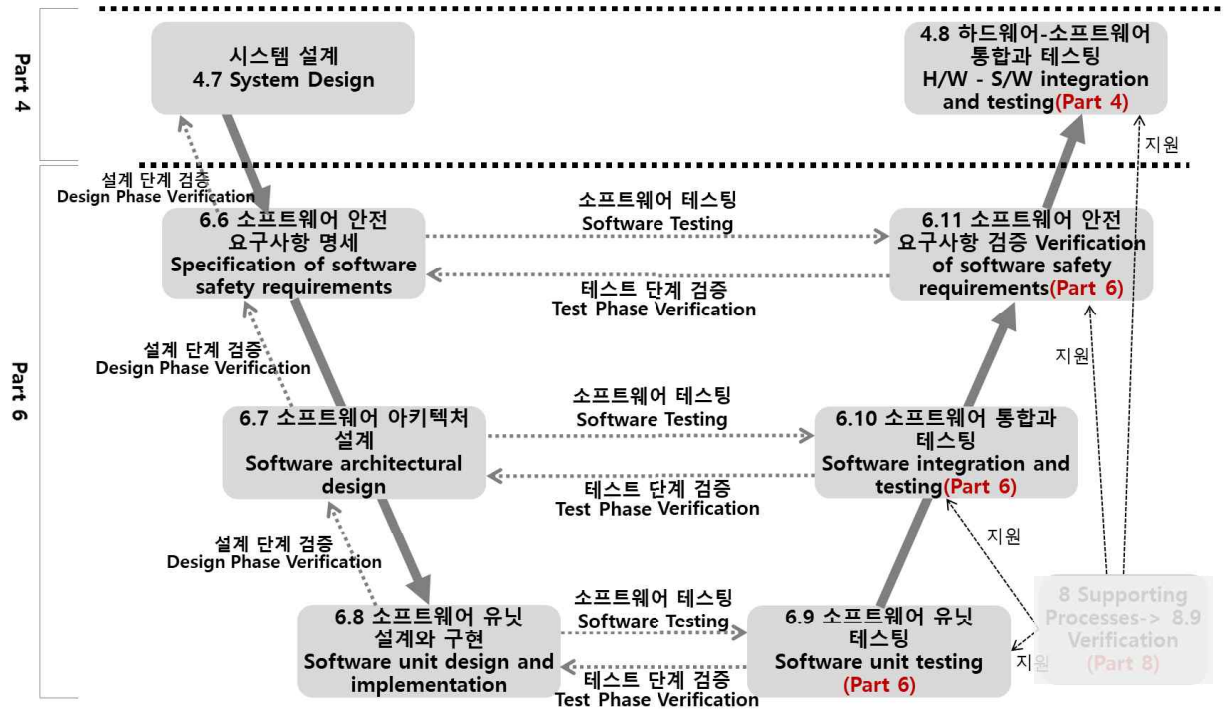
Test 적용범위(1)

ISO	장(Chapter)	표(Table)
PART - Product development at the software 소프트웨어 레벨 제품개발	9. Software unit testing 소프트웨어 단위 테스팅	Methods for software unit testing (Table 10, p.20) 소프트웨어 유닛 테스팅 메소드
		Methods for deriving test cases for software unit testing (Table 11, p.20) 소프트웨어 유닛 테스팅 테스트 케이스 도출 메소드
		Structural coverage metrics at the software unit level (Table 12, p.21) 소프트웨어 유닛 레벨, 구조적 커버리지 메트릭
	10. Software integration and testing 소프트웨어 통합과 테스팅	Methods for software integration testing (Table 13, p.23) 소프트웨어 통합 테스팅 메소드
		Methods for deriving test cases for software integration testing (Table 10, p.24) 소프트웨어 통합 테스팅 테스트 케이스 도출 메소드
		Structural coverage metrics at the software architectural level (Table 12, p.24) 소프트웨어 아키텍처 레벨, 구조적 커버리지 메트릭
	11. Verification of software safety requirements (System Testing) 소프트웨어 안전 요구사항 검증	Test environments for conducting the software safety requirements verification (26262-6, Table 16, p.26) 소프트웨어 안전 요구사항 검증 테스트 환경
PART - Product development system 시스템 레벨 제품 개발	8. Hardware-Software integration and testing 하드웨어-소프트웨어 통합과 테스팅	Methods for deriving test cases for (item) integration testing (Table 4, p.18) (아이템) 통합 테스팅용 테스트 케이스 도출 메소드
		Correct implementation of technical safety requirements at the hardware-software level (Table 5, p.19) 하드웨어-소프트웨어 레벨, 기술 안전 요구사항의 올바른 구현
		Correct functional performance, accuracy and timing of safety mechanisms at the hardware-software level (Table 6, p.19) 하드웨어-소프트웨어 레벨, 안전 매커니즘의 올바른 기능적 성능, 정확성, 타이밍
		Consistency and correct implementation of external and internal interfaces at the hardware-software level (Table 7, p.20) 하드웨어-소프트웨어 레벨, 외부와 내부 인터페이스의 일관성과 올바른 구현
		Effectiveness of a safety mechanism's diagnostics coverage at the hardware-software level (Table 8, p.20) 하드웨어-소프트웨어 레벨, 안전 매커니즘의 진단 커버리지의 효과성
		Level of robustness at the hardware-software level (Table 9, p.20) 하드웨어-소프트웨어 레벨, 강간성 레벨
PART - Supporting processes 지원 프로세스	9. Verification* 검증	9.4.1 Verification Planning 검증 계획
		9.4.2 Verification Specification 검증 명세
		9.4.3 Verification execution and evaluation 검증 실행과 평가

* 가이드에 수록한 테스트 메소드 목록은 별첨 참조

Test 적용범위(2)

V 모델 관점에서 테스트 메소드 적용 범위임. Part 8은 본 발표 범위에서 제외함



<ISO 26262 Part 6 소프트웨어 개발 모델>

Test 적용방안(1)

ISO 26262의 Part 4, Part 6 테스트 메소드를 ①메소드와 ②테스트 기법으로 분류하고 ASIL 등급에 따라 테스트 메소드 설명 수준을 달리함

테스트 메소드 (Test Method) 27개	2개로 분리	**메소드 (Method) 12**	메소드 상세설명(Full Explanation Methods) 6
			(메소드) 간략 설명(Brief Explanations) 6
		테스트 기법 (Test 15	기법 상세설명(사례-Examples 포함) 7
			(기법) 간략 설명(Brief Explanations) 8

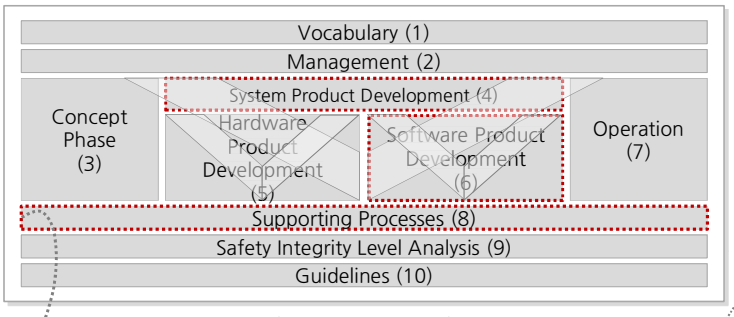
간략설명(Brief Explanations)	ASIL 등급이 A, B이고 권장 수준이 ++(highly recommended)가 아니어서 중요성이 떨어지는 메소드나 설계 기법은 A4 용지 반 페이지 정도로 간략히 설명함.
기법상세설명(Full Explanation Techniques, 사례 포함)	ASIL 등급이 A, B이면서 권장 수준이 ++(highly recommended)여서 중요하기 때문에 설계 기법 사례(Example)를 제시해 상세히 설명함. 설계 기법이 테스트 레벨마다 중복되더라도 레벨마다 설명하지 않고 한 번만 설명함. * 사례(Examples): 테스트 설계 기법을 적용해 테스트 케이스 도출 사례를 상세히 제시함.
메소드상세설명(Full Explanation Methods)	ASIL 등급이 A, B이면서 권장 수준이 ++(highly recommended)여서 중요하기 때문에 메소드를 자세히 설명함. 메소드가 테스트 레벨마다 중복되더라도 레벨마다 설명하지 않고 한 번만 설명함.

* 가이드에 수록한 테스트 메소드 목록은 별첨 참조

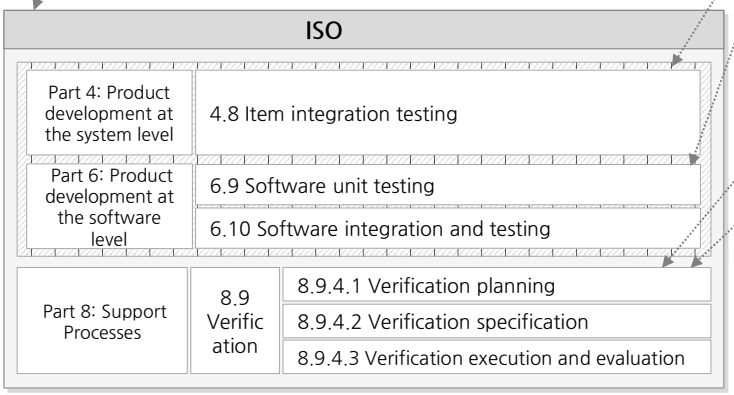
Test 적용방안(2)

테스트 메소드 가이드는 ISO/IEC/IEEE 29119 Part 3, 4를 참조해 작성함. ISO/IEC/IEEE 29119의 미지원 영역은 다른 표준이나 지식체계로 커버함

ISO 구조

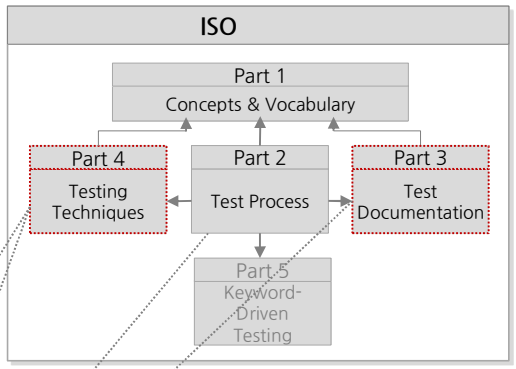


<ISO 26262 Overview>

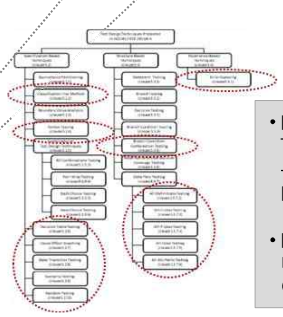


<ISO 26262 Part 4, 6, 8의 SW테스팅 관련 범위>

ISO/IEC/IEEE 구조



<ISO/IEC/IEEE 29119 Overview>



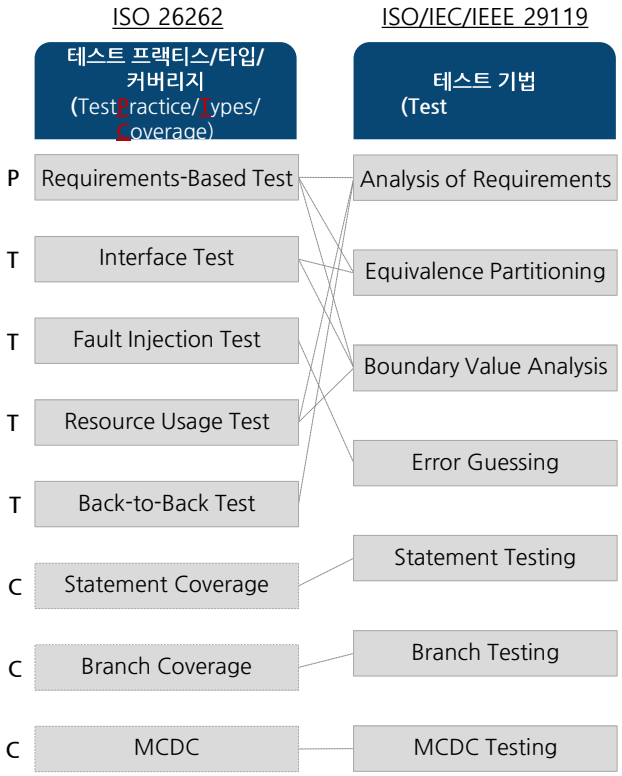
- ISO/IEC/IEEE 29119의 Test Techniques을 적용해 ISO 26262 Test Method 가이드 작성
- ISO/IEC/IEEE 29119의 미지원 영역은 다른 표준이나 지식체계로 커버함

<ISO/IEC/IEEE 29119 Test Techniques>

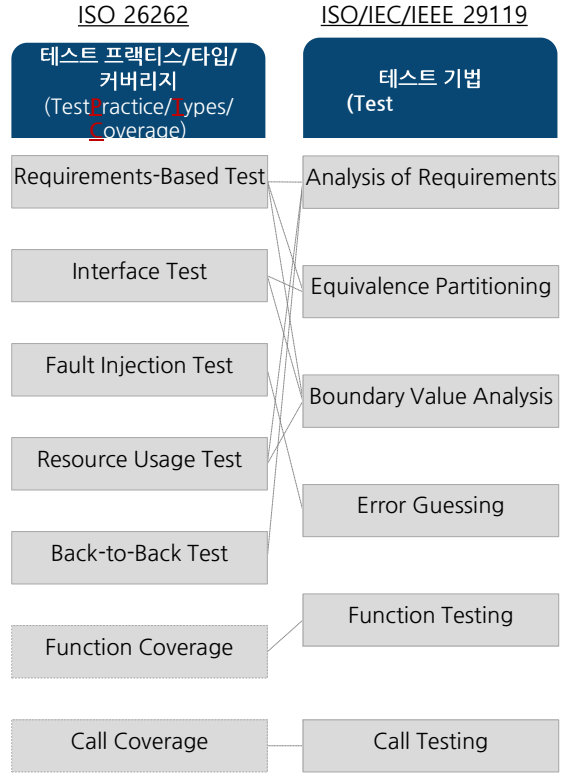
Test 적용방안(3)

ISO 26262 Part 4, 6 테스트 메소드와 ISO/IEC/IEEE 29119 Part 4 테스트 기법과 매핑 관계임

Software Unit Mapping



Software Integration Testing Mapping



Test 가이드 도출(1)

: 인터페이스 테스트(1)

1. **메소드 이름(Method** Interface Test

2. **정의(Definition)** 인터페이스 테스트는 두 가지가 있다. ① 소프트웨어 단위 내 입력 인터페이스 자체와 ②소프트웨어 단위(Unit)가 다른 단위와 입력/출력 인터페이스 명세대로 잘 동작하는지 테스트한다. ①은 단위 테스팅에서 인터페이스 테스트 ②는 통합 테스팅에서 인터페이스 테스트로 나눌 수 있다.

3. **ISO 영역**
Part 6 - 9. Software unit testing, 10. Software integration testing
Part 4 - 8. Hardware-Software integration and testing

4. **ASIL 등급**
Part 6 - 9. Software unit testing, 10. Software integration testing

A	B	C	D
++	++	++	++

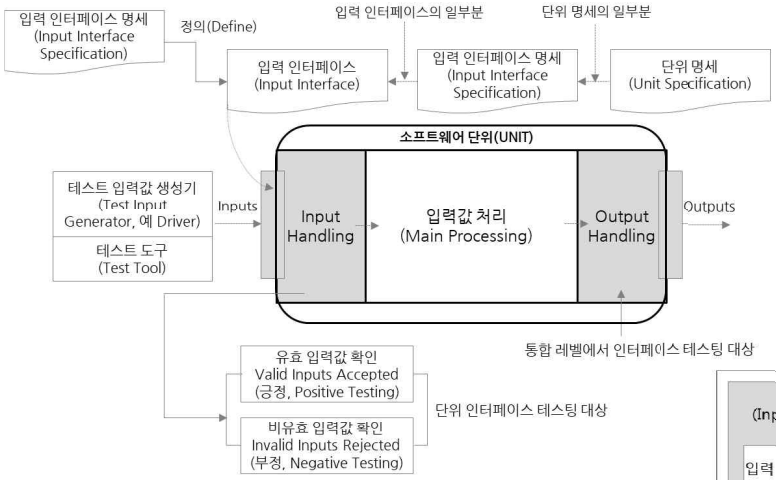
Part 4 - 8. Hardware-Software integration and testing

A	B	C	D
++	++	++	++

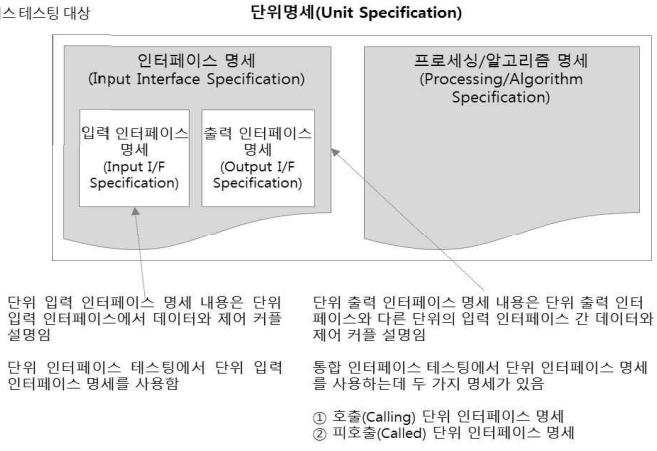
5. **설명(Explanation)**
인터페이스 테스트는 ①소프트웨어 단위(Unit)인 함수(예, 구조체 멤버 변수, 라이브러리 함수 호출)나 모듈이 함수나 모듈 내에서 입력 처리나 ②다른 함수나 모듈의 인터페이스와 입력/출력 동작을 올바르게 수행하는지 테스트한다. 자동차 속도 감지기의 감지 모듈(A)은 감지한 속도를 A의 출력 명세에 따라 모듈(B)로 넘길 때 모듈(B) 는 모듈(B)의 입력 명세에 따라 동작(입력 받고)하고, B 가 받은 데이터는 출력 명세에 따라 모듈 A나 다른 모듈로 출력되는지 확인한다. 인터페이스 테스트는 단위테스팅, 통합테스팅에서 가능하다. 아래 그림처럼 인터페이스 단위 테스팅은 모듈 내의 입력 처리 여부에 관심이 있다. 인터페이스 테스트에 필요한 데이터는 입력값 생성기((예, 데이터를 생성하는 도구나 드라이버(Driver) 역할을 하는 모듈)에서 생성한다. 생성된 데이터는 입력 인터페이스 명세에 따라 입력 처리(Input handling)되는지 확인한다. 이 때 테스트 출력값을 확인하고자 테스트 출력값 확인기로 출력 인터페이스 명세대로 출력값이 맞는지 확인한다. 입력 에러 처리도 제대로 하는지 함께 살펴본다.

Test 가이드 도출(2)

: 인터페이스 테스트(2)



<인터페이스 단위 테스팅 모델>



<단위 인터페이스 명세>

Test 가이드 도출(3)

: 동등분할(1)

1. **기법 이름(Technique** Generation and analysis of equivalence classes

2. **ISO 영역**
Part 6 - 9. Software unit testing, 10. Software integration testing
Part 4 - 8. Hardware-Software integration and testing

3. **ASIL 등급**
Part 6 - 9. Software unit testing, 10. Software integration testing

A	B	C	D
+	+	++	++

Part 4 - 8. Hardware-Software integration and testing

A	B	C	D
+	++	++	++

4. **설명(Explanation)**
소프트웨어나 시스템의 입력과 출력값의 영역을 유한 개의 상호 독립적인 분할 영역(=집합, 클래스 class, 도메인)으로 구분한 후 각 분할 영역에서 대표값을 선택해 테스트 케이스를 작성한다. 같은 영역에 있는 입력값은 내부에서 같은 방식으로 처리하기 때문에 어떤 입력값을 선택해도 결과값(기대결과)이 같다는 원리를 이용한다. 입력값 분할 영역은 유효 영역, 비유효(입력되면 안되는 값)….
일반적으로 동등분할을 적용할 수 있는 분야는 아래와 같다.

- 입력값(Inputs), 출력값(Outputs), 내부값(Internal values)
- 시간 관련 값(Time-related values, 이벤트 이전과 이후)
- 통합 테스팅에서 모듈간 인터페이스 파라미터(Interface parameters)

동등분할 적용 방법은 아래와 같다.
- 시스템 정보(예, 요구사항정의서)를 분석해 입력과 출력 영역을 비슷한 특징을 가진 클래스(Equivalence Class)로 분할
- 분할한 클래스에서 각 클래스를 대표하는 테스트 케이스 도출
- 모든 유효 영역을 커버할 때까지 테스트 케이스를 생성
- 모든 비유효 영역을 커버할 때까지 테스트 케이스를 생성

Test 가이드 도출(5)

: 동등분할(2)

5. 테스트 케이스 도출(Example)

(1) 테스트 컨디션

속도는 정수(0, 양수, 음수)로 한정한다. 비유효 출력값이 반드시 가능한 건 아니다(예, 스펙에서 부저의 간헐적인 울림은 유효 출력값이 아님).
그러나 테스팅 관점에서 시스템이 비유효한 출력값을 생성하는지 시도해 볼 만하다. 테스트 컨디션은 ①~⑨까지 모두 9개이다.

구분	입력값 Inputs	출력값 Outputs
유효값 Valid	① EP1 : -30 ≤ speed ≤ 220 ② EP2 : button press	③ EP3 : light on ④ EP4 : light off ⑤ EP5 : buzzer on ⑥ EP6 : buzzer off
비유효값 Invalid	⑦ EP7 : speed < -30 ⑧ EP8 : speed > 220	⑨ EP9 : intermittent buzzer*

테스트 컨디션

(2) 테스트 커버리지 아이템과 테스트 케이스

+++은 테스트 케이스 3개 도출이 이상적이라는 뜻이며 테스트 커버리지 아이템 3개를 처리하는 것이다. 또 ##은 결함 주입 기법 적용이 필요하면 비유효 입력값(속도)으로 테스트 케이스 2개가 필요하다는 뜻이다.

equivalence partitions = coverage items			Test	
			preconditions/ test inputs	expected results / postconditions
-30 ≤ speed ≤ 220	+++	EP1	active speed = 28 button pressed	active light on buzzer off
button press	+++	EP2	active speed = 28 button pressed	active light on buzzer off
speed < -30	##	EP3	active speed = -60 button pressed	active
speed > 220	##	EP4	active speed = 366 button pressed	active
light on	+++	EP5	active speed = 28 button pressed	active light on buzzer off
light off		EP6	active buzzer off speed = 44 button pressed	active light off
buzzer on		EP7	active buzzer off speed = 52	active buzzer on
buzzer off		EP8	active buzzer on speed = 47	active buzzer off
intermittent buzzer		EP9	active buzzer off speed = 50*	active buzzer off

테스트 케이스

Test 가이드 도출(6)

: 페어와이즈 테스팅

1. 설명(Explanation)

페어와이즈(Pairwise) 조합 테스팅은 커버해야 할 범위에 비해 상대적으로 적은 량의 테스트 세트(Set)를 구성해 결함을 찾고 테스트에 대한 자신감(Confidence)을 얻는 방법 중에 하나이다.

페어와이즈는 결함 대부분이 2개 요소의 상호작용(Interactions of two factors)에 기인함에 착안해 2개 요소의 모든 조합을 다룬다. 즉 페어와이즈 조합이란 테스트에 필요한 각 값들이 다른 파라미터(Parameters)의 값(Value)과 최소한 한번씩은 조합한다. 제한된 자원으로 테스트 대상의 설정, 기능, 이벤트 등의 조합을 모두 테스트하는 것은 현실적으로 경제적으로나 불가능하다.

반면 테스트지 않거나 일부 조합을 의도적으로 빠뜨리는 건 그만큼의 위험을 동반하므로 조합 테스팅은 매우 중요한 의미가 있다. 페어와이즈 조합 테스팅으로 모든 조합을 고려해 테스팅하더라도 모든 결함을 발견하는 건 아니다.

그러나 페어와이즈 조합 기법을 사용해 테스팅한 결과에 '결함이 없음' 까지는 보장한다. 그리고 경험적으로 의미 있고 결함을 발견할 가능성이 높은 조합을 추가해 조합을 늘리면 조합 테스팅의 효과성을 더 높일 수 있다. 물론 늘어난 조합을 포함해 도출한 테스트 케이스가 보장하는 범위는 페어와이즈 조합 테스팅 기법이 보장하는 범위까지이다.

2. 시스템 파라미터와 값

Conditions	Road	Braking	Speed
dry	straight	heavy	low
wet	curved	light	medium
snow		no braking	high
ice			

3. 페어링(Pairing) 상세

PAIRING

No	var1	var2	value1	value2	appearances	cases
1	Conditions	Braking	dry	heavy	1	1
2	Conditions	Braking	dry	light	1	2
3	Conditions	Braking	dry	no braking	1	9
4	Conditions	Braking	wet	heavy	1	3
5	Conditions	Braking	wet	light	1	4
6	Conditions	Braking	wet	no braking	1	10
7	Conditions	Braking	snow	heavy	1	5
8	Conditions	Braking	snow	light	2	6, 13
9	Conditions	Braking	snow	no braking	1	11
10	Conditions	Braking	ice	heavy	1	12
11	Conditions	Braking	ice	light	1	14
12	Conditions	Braking	ice	no braking	2	7, 8
13	Conditions	Speed	dry	low	1	1
14	Conditions	Speed	dry	medium	1	2
15	Conditions	Speed	dry	high	1	9
16	Conditions	Speed	wet	low	1	4
17	Conditions	Speed	wet	medium	1	3
18	Conditions	Speed	wet	high	1	10
19	Conditions	Speed	snow	low	1	11
20	Conditions	Speed	snow	medium	1	13
21	Conditions	Speed	snow	high	2	5, 6
22	Conditions	Speed	ice	low	2	7, 14
23	Conditions	Speed	ice	medium	1	8
24	Conditions	Speed	ice	high	1	12
25	Conditions	Road	dry	straight	2	1, 9
26	Conditions	Road	dry	curved	1	2
27	Conditions	Road	wet	straight	1	3
28	Conditions	Road	wet	curved	2	4, 10
29	Conditions	Road	snow	straight	2	6, 13
30	Conditions	Road	snow	curved	2	5, 11
31	Conditions	Road	ice	straight	2	7, 14
32	Conditions	Road	ice	curved	2	8, 12
33	Braking	Speed	heavy	low	1	1
34	Braking	Speed	heavy	medium	1	3
35	Braking	Speed	heavy	high	2	5, 12
36	Braking	Speed	light	low	2	4, 14
37	Braking	Speed	light	medium	2	2, 13
38	Braking	Speed	light	high	1	6
39	Braking	Speed	no braking	low	2	7, 11
40	Braking	Speed	no braking	medium	1	8
41	Braking	Speed	no braking	high	2	9, 10
42	Braking	Road	heavy	straight	2	1, 3
43	Braking	Road	heavy	curved	2	5, 12
44	Braking	Road	light	straight	3	6, 13, 14
45	Braking	Road	light	curved	2	2, 4
46	Braking	Road	no braking	straight	2	7, 9
47	Braking	Road	no braking	curved	3	8, 10, 11
48	Speed	Road	low	straight	3	1, 7, 14
49	Speed	Road	low	curved	2	4, 11
50	Speed	Road	medium	straight	2	3, 13
51	Speed	Road	medium	curved	2	2, 8
52	Speed	Road	high	straight	2	6, 9
53	Speed	Road	high	curved	3	5, 10, 12

4. 최종 테스트 케이스

No.	테스트 케이스			
1	dry	Straight	heavy	low
2	dry	curved	light	medium
3	wet	straight	heavy	medium
4	wet	curved	light	low
5	snow	curved	heavy	high
6	snow	straight	light	high
7	ice	straight	no braking	low
8	ice	curved	no braking	medium
9	dry	~straight	no braking	high
10	wet	~curved	no braking	high
11	snow	~curved	no braking	low
12	ice	~curved	heavy,	high
13	snow	~straight	~light	medium
14	ice	~straight	light	~low

<페어와이즈, 테스트 케이스 도출 절차>

실무 적용과 효과 분석(1)

자동차 부품 제조사와 '가이드 적용-> 가이드 개선' 과정을 반복해 실무 활용도를 높이고 가이드의 적정성을 검증함

적용 방법

<Test Method 가이드>

교육/훈련

<QA담당자>

<개발자>

회의

적용 절차

2017. 09 | 2017. 10 | 2017. 11

- 테스팅 역량 Quick 진단
- Test Method 교육
- Test Method 일부 선적용
- 가이드에 따라 Test Method 적용
- 산출물 검토와 수정
- 가이드 테일러링 I (기보유) 가이드 업데이트

기업 요구사항 반영
이해도 향상
가독성 향상

개선 피드백

<가이드 중간산출물>

기업 요구사항 반영
이해도 향상
가독성 향상

개선 피드백

<가이드 중간산출물>

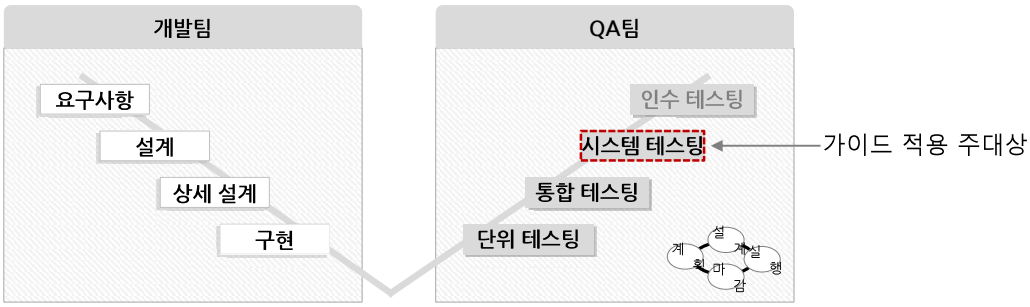
기업 요구사항 반영
이해도 향상
가독성 향상

개선 피드백

<가이드 중간산출물>

실무 적용과 효과 분석(2)

시범 적용 회사는 개발팀과 독립적인 QA팀이 있고 개발 단계별 산출물 품질이 좋음. 가이드 적용 주 대상은 시스템 테스팅 단계임



구분	개발팀	QA팀	비고
영역	개발 단계	단위~시스템테스트	개발 산출물 품질 좋음
업무	• 정적분석 • 단위테스트	SW 기능시험	비기능과 HW 관련 시험은 별도팀이 담당
		• 단위 : 함수 단위 • 통합 : 단위 수준의 통합 • 시스템 : 기능시험	확인, 리그레션 테스트 실행
도구	정적분석도구	단위테스트 도구사용	-

<개발팀과 QA팀 역할>

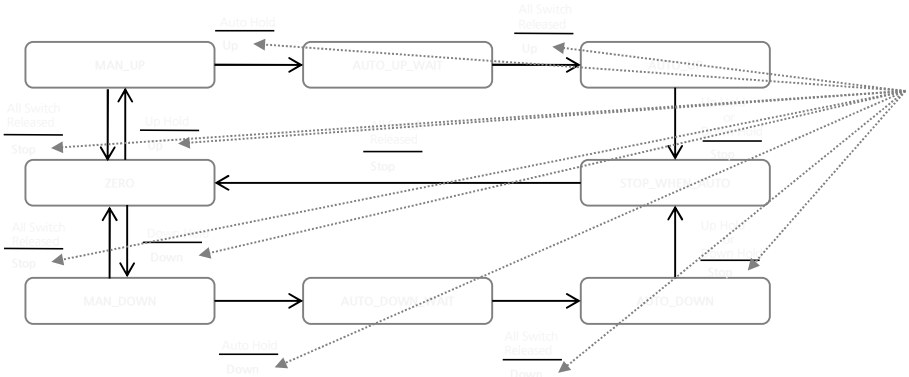
실무 적용과 효과 분석(3)

테스트 메소드 11개를 적용해 테스트 커버리지 향상, 새로운 테스트 기법 적용, 개념 정립 등 효과를 확인함

No.	주 사용중 메소드	가이드한 메소드	핵심 가이드(기존 대비 보완점)
1	Requirements-based test	Requirements-based test	개념 정립
2	Analysis of requirements	State Transition testing	• Action 추가 • 상태전이테이블 추가 • 전이트리 추가 • 1 Switch 추가 • 비유효한(Invalid) 케이스 추가
3		Decision Table Testing	결정테이블 커버리지 추가
4		Pairwise Testing	페어와이즈 커버리지 추가
5	Generation and analysis of equivalence classes	Generation and analysis of equivalence classes	동등분할 커버리지 강화(출력영역 추가)
6	Analysis of boundary values	Analysis of boundary values	경계값분석 커버리지 강화(3 개념 추가)
7	Error Guessing	Error Guessing	에러추정 정확한 개념과 접근법 추가
8	Analysis of field experiences	Analysis of field experiences	탐색적 테스팅 절차 소개로 정확한 이해
9	Statement coverage	Statement coverage	도구 사용 시 구문 커버리지 원리 상세 이해
10	Branch coverage	Branch coverage	도구 사용 시 분기 커버리지 원리 상세 이해
11	-	MC/DC	• 미사용중이나 MC/DC 개념 정립 • 도구의 정확한 사용 • 블랙박스 테스트에 응용 가능

실무 적용과 효과 분석(4)

상태전이 테스팅은 시범적용 업체가 사용치 않은 '액션', '상태전이 테이블', '1 스위치 전이트리' 등을 추가해 기존에 발견하지 못한 결함을 발견함



Action(액션) 추가

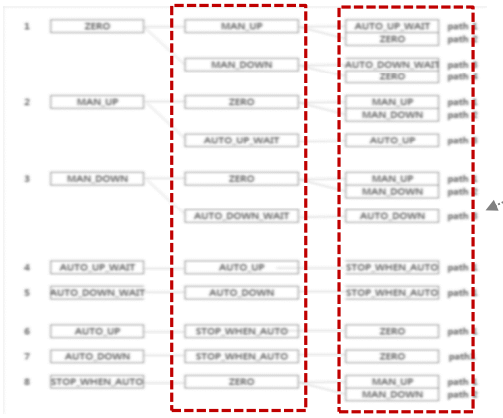
〈상태전이도〉

상태전이 테이블 추가

	ZERO	MAN_UP	AUTO_UP_WAIT	AUTO_UP	MAN_DOWN	AUTO_DOWN_WAIT	AUTO_DOWN	STOP_WHEN_AUTO
All	ZERO	ZERO	AUTO_UP	AUTO_UP	ZERO	AUTO_DOWN	AUTO_DOWN	ZERO
Up Hold	MAN_UP	MAN_UP	AUTO_UP_WAIT	STOP_WHEN_AUTO			STOP_WHEN_AUTO	STOP_WHEN_AUTO
Down Hold	MAN_DOWN			STOP_WHEN_AUTO	MAN_DOWN	AUTO_DOWN_WAIT	STOP_WHEN_AUTO	STOP_WHEN_AUTO
Up Hold & Hold		AUTO_UP_WAIT	AUTO_UP_WAIT	STOP_WHEN_AUTO			STOP_WHEN_AUTO	STOP_WHEN_AUTO
Down Hold & Hold				STOP_WHEN_AUTO	AUTO_DOWN_WAIT	AUTO_DOWN_WAIT	STOP_WHEN_AUTO	STOP_WHEN_AUTO

〈상태전이 테이블〉

실무 적용과 효과 분석(5)



〈전이트리(0 Switch)〉 〈전이트리(1 Switch)〉

•전이트리(1 Switch 커버리지) 추가

•비유효(Invalid) 테스트케이스 추가
•결함 검출

#	사전조건	시작상태	이벤트	기대결과	테스트 결과
STT1(IV)	-	ZERO	Up Hold & Auto Hold	Manual Up	PASS
STT2(IV)	-	ZERO	Down Hold & Auto Hold	Manual Down	PASS
STT3(IV)	-	MAN_UP	Down Hold	모터 정지	FAIL
STT4(IV)	-	MAN_UP	Down Hold & Auto Hold	모터 정지	FAIL
STT5(IV)	-	AUTO_UP_WAIT	Down Hold	모터 정지	PASS
STT6(IV)	-	AUTO_UP_WAIT	Down Hold & Auto Hold	모터 정지	PASS
STT7(IV)	-	MAN_DOWN	Up Hold	모터 정지	FAIL
STT8(IV)	-	MAN_DOWN	Up Hold	모터 정지	FAIL
STT9(IV)	-	AUTO_DOWN_WAIT	Up Hold & Auto Hold	모터 정지	PASS
STT10(IV)	-	AUTO_DOWN_WAIT	Up Hold & Auto Hold	모터 정지	PASS

〈비유효(Invalid) 테스트케이스〉

결론

1	테스트 메소드 가이드를 적용해 테스트 기법 커버리지 향상
2	테스트 기법 커버리지 증가로 결함 검출력 향상 예상
3	ISO26262 대응 체계를 구축한다면 테스트 메소드 분야는 충분한 근거를 제시함

향후 관련 연구

1	미적용한 테스트 메소드 추가 적용해 활용 사례 연구
2	ISO 26262 Part 8 Support process - 8.9 Verification 활용 사례 연구
3	철도, 의료 등 도메인 별 테스트 메소드 가이드 적용과 효과 분석

Test 목록(1)

가이드에 수록한 ISO 26262 Part 4, Part 6 Test Method

ISO			ISO		Interpreted Techniques	Test Technique	ASIL				Description(B,
							A	B	C	D	
PART - Product development software	9. Software unit testing	Methods for software unit testing (Table 10, p.20)	1a	Requirements-based test			++	++	++	++	Full Method
			1b	Interface test			++	++	++	++	Full Method
			1c	Fault injection test			+	+	+	++	Brief
			1d	Resource usage test			+	+	+	++	Brief
			1e	Back-to-back comparison test between model and code, if applicable			+	+	++	++	Brief
		Methods for deriving test cases for software unit testing (Table 11, p.20)	1a	Analysis of requirements	State Transition testing	Yes	++	++	++	++	Brief
					Decision Table Testing	Yes	++	++	++	++	Full Technique
					Pairwise Testing	Yes	++	++	++	++	Brief
			1b	Generation and analysis of equivalence classes		Yes	+	++	++	++	Brief
			1c	Analysis of boundary values		Yes	+	++	++	++	Full Technique
			1d	Error guessing		Yes	+	+	+	+	Brief
		Structural coverage metrics at the software unit level (Table 12, p.21)	1a	Statement coverage		Yes	++	++	+	+	Full Technique
			1b	Branch coverage		Yes	+	++	++	++	Full Technique
			1c	MC/DC (Modified Condition/Decision Coverage)		Yes	+	+	+	++	Brief
	10. Software integration and testing	Methods for software integration testing (Table 13, p.23)	1a	Requirements-based test			++	++	++	++	Brief
			1b	Interface test			++	++	++	++	Brief
			1c	Fault injection test			+	+	++	++	Brief
			1d	Resource usage test			+	+	+	++	Brief
			1e	Back-to-back comparison test between model and code, if applicable			+	+	++	++	Brief
		Methods for deriving test cases for software integration testing (Table 10, p.24)	1a	Analysis of requirements	State Transition testing	Yes	++	++	++	++	Brief
					Decision Table Testing	Yes	++	++	++	++	Brief
					Pairwise Testing	Yes	++	++	++	++	Brief
			1b	Generation and analysis of equivalence classes		Yes	+	++	++	++	Full Technique
			1c	Analysis of boundary values		Yes	+	++	++	++	Brief
			1d	Error guessing		Yes	+	+	+	+	Brief
		Structural coverage metrics at the software architectural level (Table 12, p.24)	1a	Function coverage		Yes	+	+	++	++	Brief
			1b	Call coverage		Yes	+	+	++	++	Brief

Test 목록(2)

ISO			ISO		Interpreted Techniques	Test Technique	ASIL				Description(B,
							A	B	C	D	
PART - Product development the level	11. Verification of software safety requirements (System Testing)	Test environments for conducting the software safety requirements verification (26262-6, Table 16, p.26)	1a	Hardware-in-the-loop			+	+	++	++	Brief
			1b	Electronic control unit network environments			++	++	++	++	Full Method
			1c	Vehicles*	범위 제외(실차 테스트에 해당)						
PART - Product development the	4.Item integration testing	Methods for deriving test cases for (item) integration testing (Table 4, p.18)	1a	Analysis of requirements	State Transition testing	Yes	++	++	++	++	Full Technique
					Decision Table Testing	Yes	++	++	++	++	Brief
					Pairwise Testing	Yes	++	++	++	++	Full Technique
			1b	Analysis of external and internal interfaces		Yes	+	++	++	++	Full Technique
			1c	Generation and analysis of equivalence classes for hardware-software integration		Yes	+	+	++	++	Brief
			1d	Analysis of boundary values		Yes	+	+	++	++	Brief
			1e	Error guessing		Yes	+	+	++	++	Brief
			1f	Analysis of functional dependencies		Yes	+	+	++	++	Brief
			1g	Analysis of common limit conditions, sequences, and sources of dependent failures		Yes	+	+	++	++	Brief
			1h	Analysis of environmental conditions and operational use cases		Yes	+	++	++	++	Full Technique
			1i	Analysis of field experiences		Yes	+	++	++	++	Full Technique
		Correct implementation of technical safety requirements at the hardware-software level (Table 5, p.19)	1a	Requirements-based test			++	++	++	++	Brief
			1b	Fault injection test			+	++	++	++	Full Method
			1c	Back-to-back test			+	+	++	++	Brief
		Correct functional performance, accuracy and timing of safety mechanisms at the hardware-software level (Table 6, p.19)	1a	Back-to-back test			+	+	++	++	Brief
			1b	Performance test			+	+	++	++	Brief
		Consistency and correct implementation of external and internal interfaces at the hardware-software level (Table 7, p.20)	1a	Test of external interfaces			+	++	++	++	Full Method
			1b	Test of internal interfaces			+	++	++	++	Full Method
			1c	Interface consistency check			+	++	++	++	Full Method

Test 목록(3)

ISO			ISO		Interpreted Techniques	Test Technique	ASIL				Description(B,
							A	B	C	D	
PART - Product development the	4.Item integration testing	Effectiveness of a safety mechanism's diagnostics coverage at the hardware-software level (Table 8, p.20)	1a	Fault injection test			+	+	++	++	Brief
			1b	Error guessing			+	+	++	++	Brief
		Level of robustness at the hardware-software level (Table 9, p.20)	1a	Resource usage test			+	+	+	++	Brief
			1b	Stress test			+	+	+	++	Brief
PART - Supporting processes	9. Verification	9.4.1	-	Verification Planning			++	++	++	++	Full Method
		9.4.2	-	Verification Specification			++	++	++	++	Full Method
		9.4.3	-	Verification execution and evaluation			++	++	++	++	Full Method

참고 문헌

1. Eckard Bringmann, Andreas Kramer, “Model based Testing of Automotive Systems”, IEEE, 2008 International Conference on Software Testing, Verification, and Validation, 2008
2. ISO/IEC/IEEE 24765 Systems and software engineering — Vocabulary
3. Hardware- in- the- Loop Test Environment for Interconnected ECUs for Passenger Cars and Commercial Vehicles, Seoul 2000 FISITA World Automotive Congress F2000A113 June 12-15, 2000, Seoul, Korea
4. Safe Automotive Software Architecture (SAFE), 10039- SAFE- WP- 6- SAFED611.pdf

SW 신뢰성 관리를 위한 테스팅 도구로부터 데이터 추출 방법

손현승 O1 박상건 1 정인용 1 류인수 1 장주수 1 김태현 2 박삼준 2
㈜모아소프트 신뢰성기술연구소 1
{hson, sgpark, iyjung, isryu, jsjang}@moasoftware.co.kr1
국방과학연구소 2
{tae_hyoun, samjoon}@add.re.kr2

A Method of Extracting Data from Testing Tools for Software Reliability Management

Hyun Seung Son O1 Sang Gun Park1 Inyong Jung1 In Su Ryu1 Jusu Jang1 Tae Hyoun Kim2 Samjoon Park2
Reliability Technology Institute, Moasoft1
Agency for Defense Development2

요 약

최근, 국방 무기체계처럼 복잡하고 중요한 시스템 소프트웨어 신뢰성의 품질 중요성은 부각되고 있다. 소프트웨어 신뢰성 측정에서 품질 척도를 확인하는 것으로 시작되지만 자동화된 시스템이 구축되지 않은 상태에서 품질 척도를 측정하는 것은 어렵다. 본 논문에서는 국방과학연구소의 “무기체계 SW 신뢰성 확보 및 검증 기술 연구”에서 개발 중인 SIRIUS(Systematic Investigation for Reliability Improvement Upon SW) 시스템을 소개하고 이 시스템과 상업용 테스트 자동화 도구인 LDRA testbed로부터 데이터를 자동으로 추출하는 방법에 대하여 논하고자 한다. 적용사례로 테스팅 자동화 도구인 LDRA testbed를 SIRIUS 시스템에 플러그인하여 동작하는 사례를 보여준다.

1. 서 론

현대 무기체계는 하드웨어보다 소프트웨어의 비중이 늘어나고 있으며 소프트웨어의 결함은 무기체계 전체에 영향을 준다[1]. 또한 이것은 양산 및 운용시 막대한 비용을 소모 시키고 문제 발생시 막대한 인명 피해를 발생시킬 수도 있다. 그러므로 이제는 무기체계 하드웨어보다 소프트웨어의 신뢰성을 확보하기 위한 방안이 절실히 요구되고 있는 시점이다.

무기체계 소프트웨어의 신뢰성 확보를 위해서는 소프트웨어 개발 라이프 사이클 동안에 각 단계에 신뢰성과 관련된 활동을 포함 시키는 방향으로 진행하고 있다[2]. 제안 요청 및 개발 준비 단계에는 신뢰성 확보활동을 포함해서 작성하도록 하고, 요구사항 분석 단계에서는 신뢰성 요구사항을, 소프트웨어 구현 및 시험 단계에서는 신뢰성 시험절차를, 개발시험평가 단계는 신뢰성 시험을 개발시험평가 항목으로 반영, 규격화 단계에서는 신뢰성시험 결과를 제출, 양산 단계에서는 형상 변경에 따른 신뢰성 시험 결과를 제출하도록 하고 있다.

[1] 본 연구는 국방과학연구소 무기체계 SW 신뢰성 확보 및 검증 기술사업의 지원을 받아 수행하였음.

하지만 이러한 소프트웨어 개발 라이플 사이클 단계에 신뢰성 활동만 수행한다고 했을 때 적절하게 수행되었는지 확인하기 어렵다. 또한 다른 대안인 무기체계 표준 및 인증을 취득 하기는 현실적으로 많은 비용과 노력이 필요하다.

이런 문제를 해결하고자 국방과학연구소는 “무기체계 SW 신뢰성 확보 및 검증 기술” 사업을 2015년도부터 진행하고 있다[3]. 이 사업은 무기체계 개발 전(全) 단계에서 SW 신뢰성의 정량적인 측정 및 평가를 위한 요소기술과 통합관리시스템을 개발하여 무기체계 SW 신뢰성 확보 및 검증 기반환경 구축을 하는 것이다.

본 논문에서는 국방 무기체계 소프트웨어의 신뢰성을 높이기 위해 개발 중인 SIRIUS(Systematic Investigation for Reliability Improvement Upon SW) 시스템을 소개하고 이 시스템과 상업용 테스트 자동화 도구인 LDRA testbed[4]에 대한 연계 방법에 대해서 논하고자 한다. 적용사례로 테스팅 자동화 도구인 LDRA testbed를 SIRIUS 시스템에 플러그인하여 동작하는 사례를 보여준다.

본 논문의 구성은 다음과 같다. 2장에서는 LDRA Tool Suite에 대해서 소개한다. 3장은 테스팅 자동화 도구인 LDRA testbed로부터 데이터 추출 방법에 대해서 설명한다. 4장은 적용사례로 SIRIUS 시스템과 LDRA

testbed를 연계한 사례를 언급한다. 5장에서는 결론 및 향후 연구를 언급한다.

2. 관련 연구

LDRA Tool Suite는 LDRA(Liverpool Data Research Associates)[4]사의 자동화 테스트 도구로, Hennell에 의해 소프트웨어 품질 평가를 수행하기 위해 1975년 Liverpool에 설립되었다. 현재 항공우주, 국방, 원자력, 의료, 자동차 및 제어 분야와 사람의 생명과 관련한 의료산업까지, 다양한 산업분야에서 연구 및 시스템 안정성 테스트에 활발히 쓰이고 있다. LDRA Tool Suite는 소프트웨어 개발 생명 주기 전반에 SW 품질 향상을 위해 구축 가능하다. 또한 LDRA의 소프트웨어 표준 준수, 테스트 및 검증 툴은 효과로 입증된 베스트 산업 사례들을 기반으로 하기 때문에 사용자가 고품질의 안전 및 보안이 중요한 제품을 개발할 수 있도록 도와준다. LDRA Tool Suite은 그림 1과 같이 소프트웨어의 요구사항, 설계 구현, 검증에 이르는 전체 소프트웨어 개발단계별 추적이 가능하며 정적분석을 통해 소프트웨어의 잠재적 결함 발견과 품질 측정을 제공한다[5].

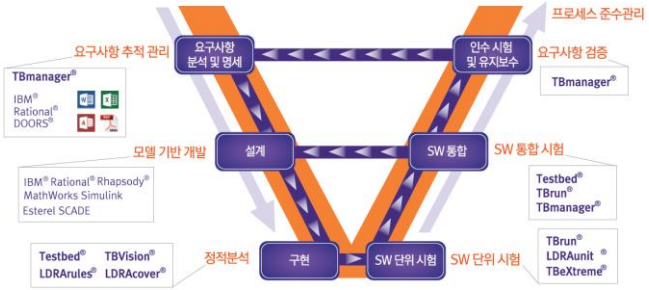


그림 1. LDRA Tool Suite를 이용한 소프트웨어개발

3. 테스팅 자동화 도구로부터 데이터 추출 방법

소프트웨어 신뢰성을 측정하기 위해서는 기본적으로 측정 지표 설정이 되어 있어야 한다. 그리고 테스팅 자동화 도구로부터 해당 지표의 값을 추출해야 한다. 하지만 모든 자동화 도구들이 해당 지표를 가지고 있지 않고 우리가 원하는 지표 값이 아닐 수 있다. 그러므로 하나의 도구만 활용해서 지표를 얻는 것이 아니라 다양한 도구와의 연계도 필요하다.

3.1 소프트웨어 신뢰성 측정 지표 설정

SIRIUS 시스템에서 사용되는 소프트웨어 신뢰성 측정 지표는 표 1과 같다. 지표는 Cyclometric Complexity, Halstead's Model, Class Size, Method Inheritance Factor, Depth of the Inheritance Tree, Total of classes, Statement Code Coverage, Branch Code Coverage, MCDC Code Coverage이 있다. 자세한 설명은 표 1에 있다.

표 1. 소프트웨어 신뢰성 측정을 위한 지표(Metrics)

Metrics name	Type	Comment
Cyclometric Complexity	int	SW의 복잡도 Edges-Nodes+1
Halstead's Model	double	연산자와 피연산자의 정보로 결함수 예측 ((Total Operator + Total Operand) * $\log_2$ *(Unique Operator + Unique Opened))/3000
Class Size	int	클래스의 크기 Methods + Attributes
Method Inheritance Factor(MIF)	double	객체지향 소프트웨어의 상속의 깊이 Total Inheritance Methods /Total Methods
Depth of the Inheritance Tree(DIT)	int	클래스 계층의 노드의 길이 Parent Classes / (Parent Classes + Methods)
Total of Classes	int	시스템 내의 총 클래스 개수 Total Classes
Statement Code Coverage	int	코드라인이 한번 이상 실행 Visited Lines / Total Lines
Branch Code Coverage	int	조건문의 참과 거짓을 모두 충족을 측정 Visited Conditions / Total Conditions
MC/DC Code Coverage	int	다른 조건 변동 없이 자신의 상태가 변경 되었을 때 결과 값에 영향을 미치는 경우를 측정 Visited Conditions / Total Conditions

이 지표는 소프트웨어 신뢰성 예측 및 추정모델을 위한 입력으로 사용하기 위해 선정하였다. 예측모델은 개발하는 소프트웨어의 잠재적인 결함을 예측한다. 적용하는 예측모델은 RADC[6], Halstead[7], CMMI[8], Shortcut[8], Industry[8] 모델이고 소프트웨어 개발 단계 중 요구사항 분석 단계부터 구현단계까지 사용한다. 추정모델은 소프트웨어 단위 시험부터 체계 통합 시험까지의 결함데이터를 바탕으로 목표 신뢰도에 도달 하기까지의 테스트 시간 및 결함률 그리고 남은 결함수를 추정한다. 적용하는 추정모델은 Goel-Okumoto[9], Delayed-S[9], Jelinski-Moranda[9], Inflection-S[9], Weibull[8], Log-logistic[10] 모델이다.

3.2 테스팅 자동화 도구로부터 신뢰성 측정 지표 추출

테스팅 자동화 도구 LDRA testbed는 다양한 종류의 측정 지표를 제공한다. 그러므로 우리는 LDRA에서 제공하는 API를 통해서 측정 지표를 추출하고 이를 SIRIUS에서 사용하는 지표로 변환해 시스템에 데이터를 업로드 한다.

그림 2와 같이 LDRA testbed로부터 데이터를 추출하는 방법은 다음과 같다. 먼저 테스터가 LDRA testbed를 이용해서 소스코드를 테스팅 한다.

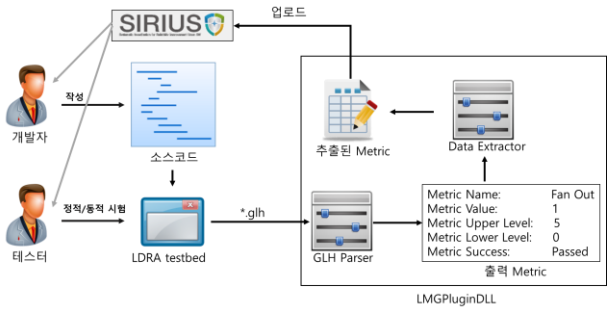


그림 2. LDRA testbed로 부터 데이터 추출 방법

이렇게 테스팅을 수행하고 나면 LDRA는 glh 파일을 생성한다. 생성된 glh 파일을 LDRA가 제공하는 API를 이용해 데이터를 추출하는 응용프로그램을 만든다. LDRA는 측정 지표에 대해 이름, 값, 레벨, 성공 여부를 텍스트로 가져올 수 있도록 제공한다. 그러므로 우리는 이 값을 데이터 처리해서 SIRIUS에서 필요한 측정 지표로 변환해 시스템에 업로드한다.

3.3 데이터 추출 응용 프로그램

LDRA testbed API를 사용하기 위해서는 LDRA가 설치 되어 있어야 하고 라이선스를 가지고 있어야 한다. 그것은 LDRA testbed API는 LDRA Tool Suite의 dll을 공유하기 때문이다. LDRA API를 사용해 데이터 추출 응용 프로그램의 알고리즘은 그림 3과 같다. 지면 관계상 예외처리는 생략한다.

```
char ResultFile[] = "example.glh";
CGLHWrapperClass wi;
IToolSuiteInterface* io = wi.getCreatedIEntityTool()
io->loadResults(ResultFile) // glh file load
CQualityResultHolder nrQuality;   // space for a quality report
char nextSource [MAXSTR] = {0};
io->loadSourceFilesList();
while (io->getNextFile(nextSource)) {
        getFileInformation(fileNode, io, nextSource, FileResults);
        io->loadQualitySettings(nextSource, FileResults);
        while (io->readNextQualityResult(nrQuality)) {
                getQualityMetric (nrQuality);
        }
        // space for a coverage report
        CCoverageResultHolder nrCoverage;
        io->loadCoverageSettings(nextSource, FileResults);
        io->showActiveDynamicMetricsOnly();
         while (io->readNextDynamicResult(nrCoverage)) {
                getCoverageResult (nrCoverage);
         }
}
```

그림 3. 데이터 추출 응용프로그램의 알고리즘

4. 적용 사례

SIRIUS(Systematic Investigation for Reliability Improvement Upon SW)는 무기체계의 대형화/복잡화 및 수출증가로 고수준의 SW 신뢰성 프로세스 정립 요구와 소스코드 시험 기반의 SW 신뢰성 평가 한계 극복을 위한 미래 지향적 개선 대책, 그리고 의사결정 지원을 위해 개발 전(全)단계에서 SW 신뢰성의 정량적 평가 필요에 의해서 국방과학연구소 주관으로 연구 과제가 진행되고 있다. SIRIUS는 잔존 결함 수, 시험 종료 시점 판단 및 SW 품질수준 파악 곤란한 문제, 한정된 기간/시험예산 문제로 무한정의 SW 테스팅 노력 투입 한계를 극복한다. SIRIUS 시스템의 전체 구조도는 그림 4와 같다.

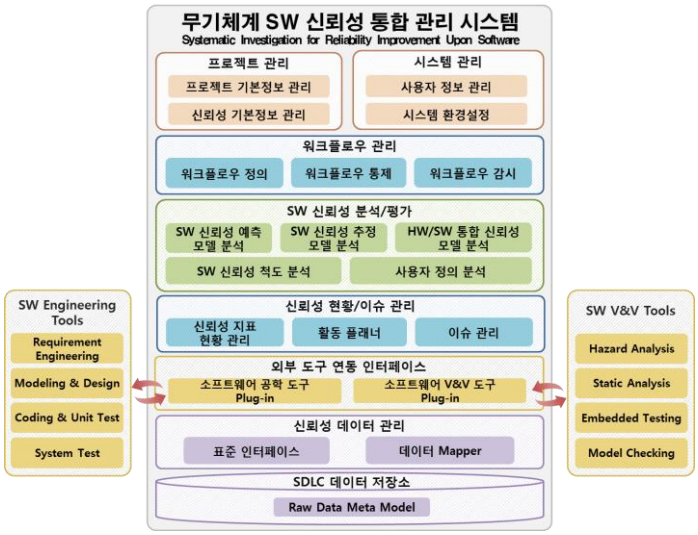


그림 4. 무기체계 SW 신뢰성 통합 관리 시스템[3]

현재 웹사이트[11]를 통해서 그림 5와 같이 접속 가능하고 LDRA testbed가 플러그인되어 사용 가능하다.

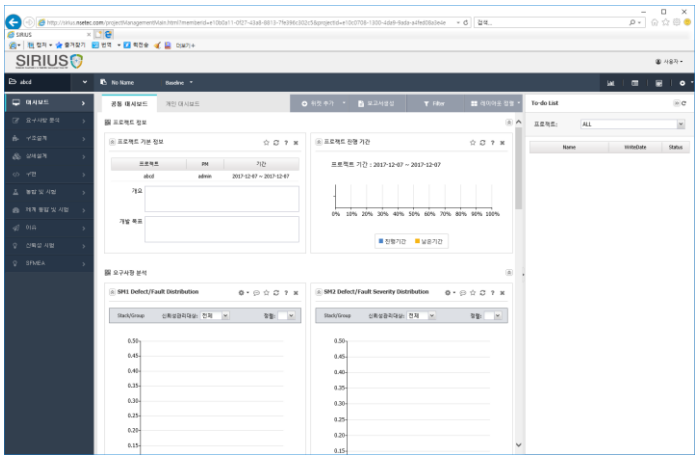


그림 5. 개발중인 SIRIUS 시스템

그림 6은 SIRIUS 시스템에 온라인으로 측정된 지표 데이터를 올릴 수 있는 프로그램이다. 이 프로그램은 LDRA glh 파일을 파일 열기를 하면 제안한 알고리즘을 통해서 SIRIUS에 필요한 지표 데이터를 추출 한다. 그리고 연동하고자 하는 서버 주소, ID, PW를 입력하고 로그인을 누르면 SIRIUS에 접속하게 된다. 마지막 단계로 프로젝트, 대상단계, 신뢰성관리대상을 입력하고 보내기 버튼을 누르면 측정된 지표 데이터를 SIRIUS에 전송한다. 우리는 이 프로그램을 통해서 LDRA testbed로 정적/동적 테스트한 결과를 SIRIUS 시스템과 연동 가능하다.

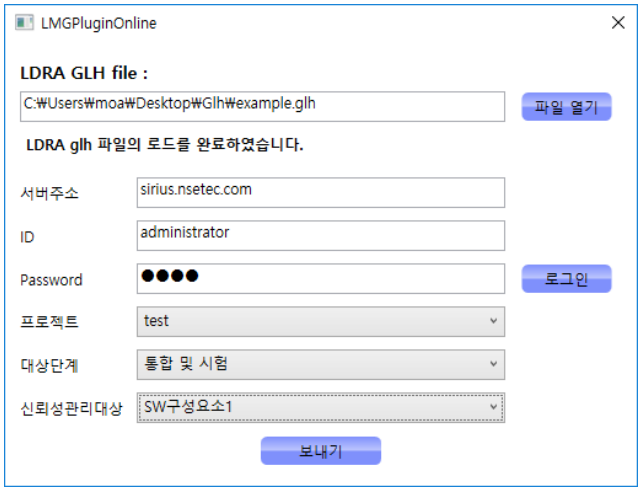


그림 6. LDRA testbed 파일 처리를 위한 SIRIUS 온라인 플러그인

본 논문에서 제안한 프로그램은 데이터 저장과 전송을 위해서 그림 7과 같은 xml 형태를 가지고 있다. 그림 7은 특정 코드를 LDRA testbed로 테스트 한 glh 파일에서 데이터를 추출한 결과이다.

```
<MeticsModel name="Set_Test.glh"
    totalSourceCodeLine="395"
    totalFunctionCount="20"
    totalClassCount="5"
    maxClassSize="5"
    maxMethodInheritanceFactor="77.777778"
    lineLimit200FunctionCount="20"
    ccLimit20FunctionCount="1"
    ccLimit7FunctionCount="19"
    statementCoverage="18"
    branchCoverage="17"
    mcdcCoverage="25"
    maxCyclomaticComplexity="29"
    halsteadsModel="1.037700">
```

그림 7. LDRA testbed를 이용한 Metrics 추출 결과

5. 결론

무기체계 소프트웨어의 신뢰성 확보를 위해서는 소프트웨어 개발 라이프 사이클 동안에 각 단계에 신뢰성과 관련된 활동을 포함 시켜야 한다. 하지만 자동화된 시스템이 구축되지 않은 상태에서 신뢰성과 관련된 품질 척도를 측정하는 것은 어려움이 있다. 이런 문제를 해결하고자 국방과학연구소는 “무기체계 SW 신뢰성 확보 및 검증 기술” 사업을 2015년도부터 진행하고 있다. 본 논문에서는 국방 무기체계 소프트웨어의 신뢰성을 높이기 위해 개발 중인 SIRIUS(Systematic Investigation for Reliability Improvement Upon SW) 시스템에 대해서 소개하였고 이 시스템과 상업용 테스트 자동화 도구인 LDRA testbed에 대한 연동 방법에 대해서 논했고 테스팅 자동화 도구인 LDRA testbed를 SIRIUS 시스템에 플러그인하여 동작하는 사례를 보여주었다. 향후 연구로는 다양한 정적/동적 분석도구를 연동 가능하도록 적용하는 연구를 수행하고자 한다.

참고문헌

[1] Michael R. Lyu, Handbook of Software Reliability Engineering, IEEE COMPUTER SOCIETY PRESS, McGraw-Hill, 2005.
[2] 한국정보통신기술협회, 무기체계 내장형 (embedded) 소프트웨어 신뢰성 확보방안 연구, 연구보고서, 방위사업청, 2012.
[3] 서달미, 손근태, 서형오, 오정섭, 김태현, 박삼준, 무기체계 SW 신뢰성 관리를 위한 프로세스 연구, 2017 한국 소프트웨어공학 학술대회(KCSE 2017), Vol. 19, No. 1, 2017. 02. 08 - 10.
[4] LDRA, http://ldra.com/
[5] 모아소프트, http://moasoft.tistory.com/33
[6] James A. McCall, William Randell, Janet Dunham, Linda Lauterbach, “Software Reliability, Measurement, and Testing Software Reliability and Test Integration”, SCIENCE APPLICATIONS INTERNATIONAL CORPSAN DIEGO CA, 1992.
[7] Maurice H. Halstead, “Elements of Software Science”, Elsevier Science, 1977.
[8] IEEE std. 1633.
[9] Hoang Pham, “System Software Reliability”, Springer Series in Reliability Engineering, 2006.
[10] S. Gokhale, K. Trivedi, “Log-logistic software reliability growth model”, Proceedings of Third IEEE International High-Assurance Systems Engineering Symposium, 1998.
[11] http://sirius.nsetec.com/

SW중소기업을 위한 경량 개발 방법론
(리드미, LeDeMe, Lean Development Methodology)
개발 및 실무 적용에 대한 연구

송원민,전수남,김태열(정보통신산업진흥원),
Stuart Reid,권원일,유희진,강응길(STA테스팅컨설팅),
이재왕(애자일소사이어티)

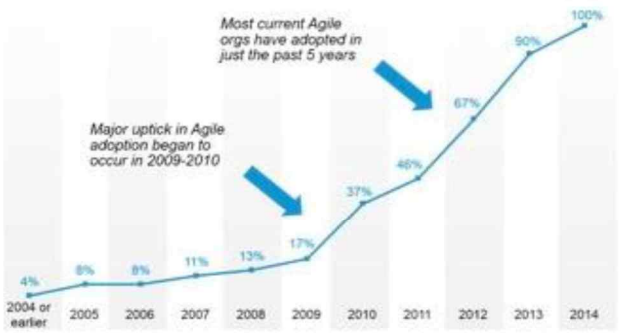
목 차

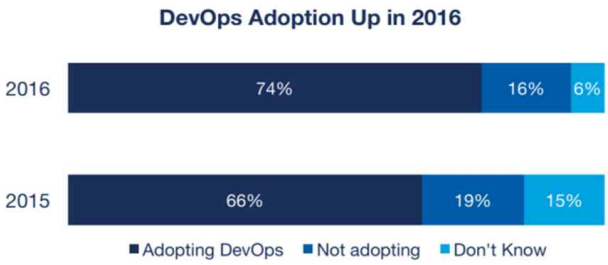
1. 연구 배경 및 필요성
2. 애자일 방법론 개요
3. 전통적 방법론과 애자일 방법론의 비교분석
4. 애자일 적용 한계점 분석
5. 방법론 설문 조사
6. 경량 개발 방법론(리드미) 개요
7. 경량 개발 방법론(리드미) 특징
8. 경량 개발 방법론(리드미) 실무 적용
9. 관련 향후 연구 대상

별첨: 참고 문헌

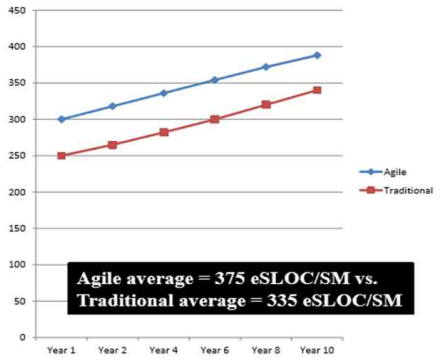
1. 연구 배경 및 필요성 (1/2)

전 세계의 IT기업들은 산출물 배포 가속화, 요구사항 변경 관리 및 생산성 강화, 그리고 품질 향상을 위해 빠르게 애자일(Agile), 데브옵스(DevOps) 등과 같은 경량 개발 방법론을 채택





• 생산성



Agile vs. Traditional Software Productivity Trends

• 프로젝트 성공률

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%

1. 연구 배경 및 필요성 (2/2)

SW 개발 기업(특히, 중소기업)의 경쟁력 강화를 위한 한국 상황에 맞는 경량 개발 방법론이 필요

배경

국내 애자일 및 데브옵스의 느린 전파와 적용 실패

- 국외 SW기업의 애자일 방법론 도입 지속적으로 증가
- SW개발 기업들의 상품 시장 출시 기간 단축 필요 (+ 품질 확보)
- 4차 산업혁명으로 인해 빠른 요구사항 대응 필요 (+ 품질 확보)

SW중소기업을 위한 경량 개발 방법론

필요성

한국 상황에 맞는 경량 개발 방법론 개발 필요

- 국내외 경량 개발 방법론 현황 분석
- 한국형 경량 개발 방법론 개발
- 실무 적용을 통한 케이스 분석 통해 실무 효용성 입증

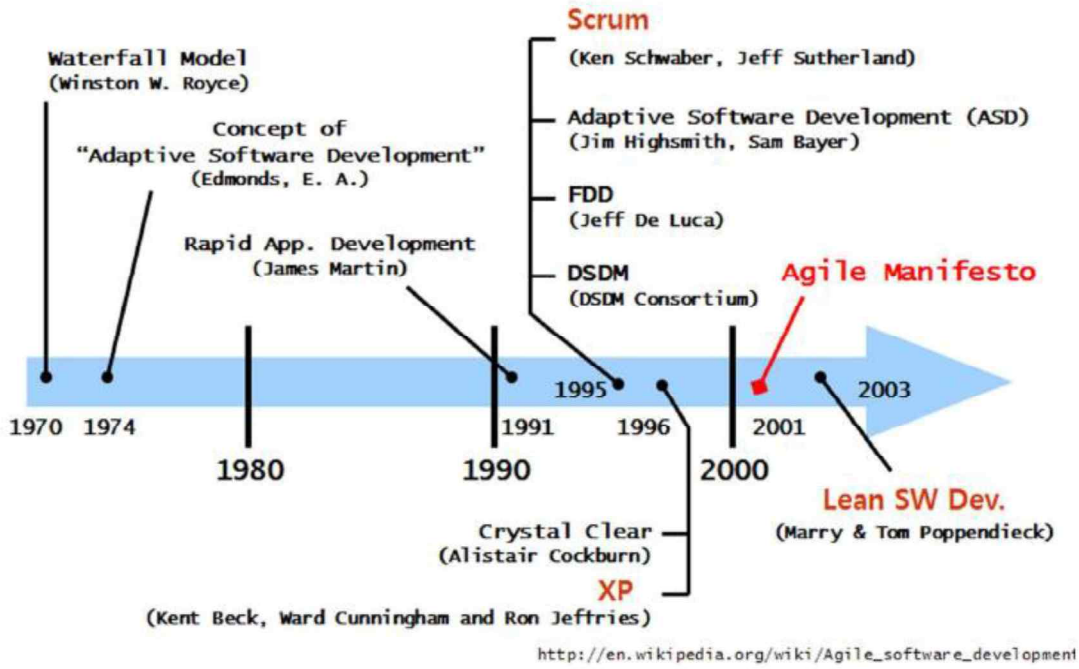
한국 상황

o **국내의 경우 애자일에 대한 관심도는 꾸준히 증가하는 추세지만, 국내 기업의 애자일 정착 사례는 아직 많지 않음**

한국 특유의 수직적 조직문화, 하향식(top-down) 명령하달구조, 수동적 업무 프로세스에 익숙해져 있어 애자일은 도입과 동시에 그 목적성을 잃고 오히려 기존 문화에 동화되기도 하며, 심지어 방법론의 도구 중 하향식 관리에 유리한 점만 몇 가지 추출돼 도입 주체의 입맛에 맞게 개조되기도 함

2. 애자일 방법론 개요 (1/2)

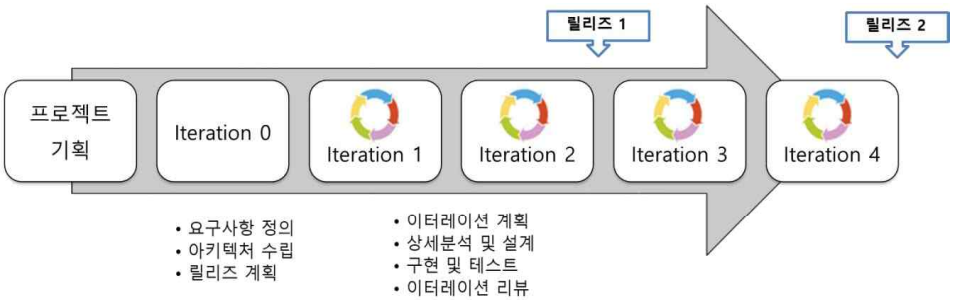
기존의 무겁고 전통적인 방법론에서 탈피하여 가벼운 방법론을 지향하며 등장. 애자일 선언문을 만든 후 '애자일'이라고 불려짐



2. 애자일 방법론 개요 (2/2)

애자일은 초기 요구사항의 불확실성을 가정하고 프로젝트 초기에 개략적인 릴리즈 계획만 세우고 주기적으로 이터레이션 계획을 수립하면서 점진적/반복적 개발을 추구

일반적인 애자일 개발 형태



애자일 선언문

우리는 SW를 개발하는 더 나은 방법을 찾는 과정에서 다음과 같이 생각하게 되었다.

1. 프로세스와 도구보다는 개인과 **개인 간의 상호 작용**에 더 큰 가치를 둔다.
2. 포괄적 문서화보다는 **동작하는 소프트웨어**에 더 큰 가치를 둔다.
3. 계약 협상보다는 **고객과의 협력**에 더 큰 가치를 둔다.
4. 계획을 따르기보다는 **변화에 대응**하는 것에 더 큰 가치를 둔다.

3. 전통적 방법론과 애자일 방법론의 비교분석 (1/2)

전통적 개발 방식과 애자일이 추구하는 목표와 관리 방법의 **차이**에 대한 분석

구 분	전통적 개발	애자일 개발
적용환경	요구사항이 비교적 명확하고 리스크가 낮은 경우	요구사항이 불확실성과 리스크가 높은 경우
통제방식	**Process –driven**	**People –driven**
프로세스 모델	정의된 (Defined) 프로세스	경험적(Empirical) 프로세스
계획수립	초기에 최대한 요구사항을 확정하고 상세 계획 수립	개략적인 범위와 일정을 수립하고 주기적으로 상세 계획 수립
개발 모델	폭포수 중심	점진적/반복적 중심
사용자 참여	**사용자 참여가 제한적**	**사용자의 지속적인 참여**
요구변경	**변경을 가급적 억제**	**변경을 주기적으로 포용**
프로젝트 목표	계획 준수	고객 가치 증대
리더십	명령과 통제, **수직적 관계**	코칭과 퍼실리테이션, **수평적 관계**
팀원 역할	개인 책임으로 일하며 한정된 역할 수행 (I자형 인재)	팀 책임으로 일하면 포괄적인 역할 수행(T자형 인재)
조직	기능(Functional) 팀	기능복합(Cross-functional) 팀
팀 평가	상위관리자 평가, 상대평가	다면 평가, 절대평가

3. 전통적 방법론과 애자일 방법론의 비교분석 (2/2)

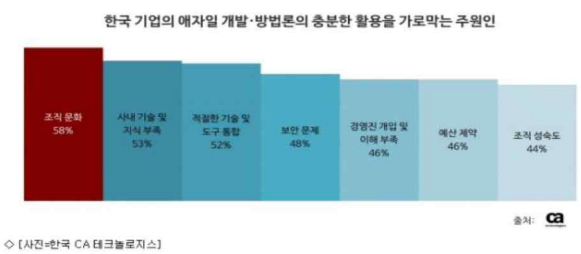
전통적 개발 방식과 애자일이 추구하는 목표와 관리 방법의 **장단점**에 대한 분석

구 분	장점	단점
전통적 개발 방법론	• **체계적인 소프트웨어 개발**에 필요한 분석/설계/구현/테스트 프로세스 및 기법, 템플릿 등을 제공 • **대규모 프로젝트** 수행에 필요한 관리 요소 등을 제공	• **과도한 개발 산출물**이 발생할 수 있음 • 초기 계획 준수에 따라 불필요한 기능 개발이나 일정, 품질 저하가 발생할 수 있음 • **분석단계 이후의 요구변화 수용이 어려움** • 사람들간의 소통과 협력을 위한 활동이 빈약함
애자일 개발 방법론	• **요구변화를 주기적으로 포용** • 개발 과정의 낭비요소를 줄일 수 있는 효과적인 지침을 제공 • 개발 구성원들간의 **소통과 협력을 원활**하게 하는 다양한 활동을 제공 • 수평적인 의사결정 및 팀원들의 사기를 높이기 위한 방법 및 리더십을 제공	• 기존 활동과는 전혀 다른 **애자일 활동을 익히기가 어려움** • 애자일이 지향하는 개발 문화와 자기 조직화된 팀을 만들기가 어려움 • 사용자가 지속적으로 참여하기 어려운 환경에서는 적절하지 않음 • 체계적인 소프트웨어 개발에 필요한 분석/설계/구현/테스트 프로세스 및 설계 기법 등을 제공하지 않음 • **대규모 프로젝트 수행에 필요한 관리 요소 등을 제공하지 않음**

4. 애자일 적용 한계점 분석

2016년 전세계 비즈니스 및 IT 경영진 1,770명을 대상으로 조사되었던 자료와 연구 과정 중 진행한 설문 조사에 따르면 애자일 방법론의 활용을 가로막는 주요 원인은 아래와 같음

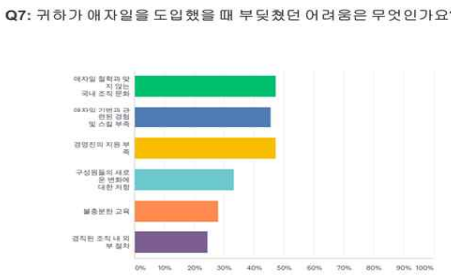
CA 보고서*



◇ [사진=한국 CA 테크놀로지스]

- **애자일과 맞지 않는 조직 문화(58%)**
- **애자일에 대한 사내 기술 및 지식 부족(53%)**
- **적절한 기술 및 도구 통합의 어려움 (52%)**
- 보안 문제(48%)
- **애자일 시작 방법에 대한 무지(46%)**
- **경영진 참여 및 이해 부족(43%)**

설문조사 (61명 응답)



- **애자일 철학과 맞지 않는 조직 문화(56%)**
- **경영진의 의지 및 지원 부족(51%)**
- **애자일 관련된 경험 및 스킬 부족(44%)**
- **국내 현실을 반영한 적용 가이드 부족(44%)**
- 구성원들의 새로운 변화에 대한 저항(28%)
- 불충분한 교육(30%)

* '한국, 데브옵스·애자일 성숙도 낮다', ZDNet Korea, 2017.3.19.
Agile & DevOps Survey Report : Accelerating Velocity and Customer Value with Agile and DevOps, CA Technologies, 2016

5. 방법론 설문 조사 (1/3)

설문 조사 개요

- 설문 기간: 2017년 8월 11일~9월 15일
- 설문 응답자 수: 61명 (응답 시 다중 선택 가능)
- 주요 설문 내용
  - 모든 애자일 방법론의 국내 도입 현황을 묻는 질문
  - 애자일 방법론 도입 시 겪는 어려움에 대한 질문
  - 애자일 방법론의 실패 및 성공 요인을 묻는 질문
  - 응답자 배경을 묻는 질문
- 설문 문항: 12문항

✓ 귀하가 애자일 방법론을 도입한다면 어떤 목적으로 도입하려고 하십니까?

보기	응답	
제품개발 기간 단축	31.15%	19
프로젝트 및 개발과정의 투명성 향상	42.62%	26
생산성 및 업무효율성 향상	65.57%	40
팀 사기 증대	8.20%	5
비즈니스와 IT 간의 효과적인 소통	26.23%	16
제품품질 향상	49.18%	30
제품 딜리버리 예측력 향상	16.39%	10
총 응답자 수: 61		

5. 방법론 설문 조사 (2/3)

✓ 귀하가 애자일 개발 방법론을 도입 하려고 할 때 겪는 어려움은 무엇인가요?

보기	응답	
국내 현실에 맞는 적용 지침 및 가이드 부족	42.62%	26
애자일과 맞지 않는 조직 문화	50.82%	31
애자일 기법과 관련된 경험 및 스킬 부족	44.26%	27
경영진의 지원 및 의지 부족	47.54%	29
구성원들의 새로운 변화에 대한 저항	26.23%	16
불충분한 교육	29.51%	18
경직된 조직 내 외부 절차	21.31%	13
총 응답자 수: 61		

✓ 국내 현실을 반영한 애자일 적용 가이드가 만들어진다면 어떤 내용이 들어가는 것이 좋은지요?

보기	응답	
수직적인 국내 조직 문화 반영	24.59%	15
애자일 활용 도구에 대한 상세 가이드	55.74%	34
애자일 적용 시 주로 발생하는 이슈에 대한 해결책	55.74%	34
애자일과 전통적 개발 방식과의 적절한 조화 활용 방법	47.54%	29
애자일 개발 방법과 사용 도구간의 원활한 연계 방법	32.79%	20
총 응답자 수: 61		

5. 방법론 설문 조사 (3/3)

✓ 귀하가 애자일을 적용하고 있다면 애자일의 어떤 점이 도움이 되었나요?

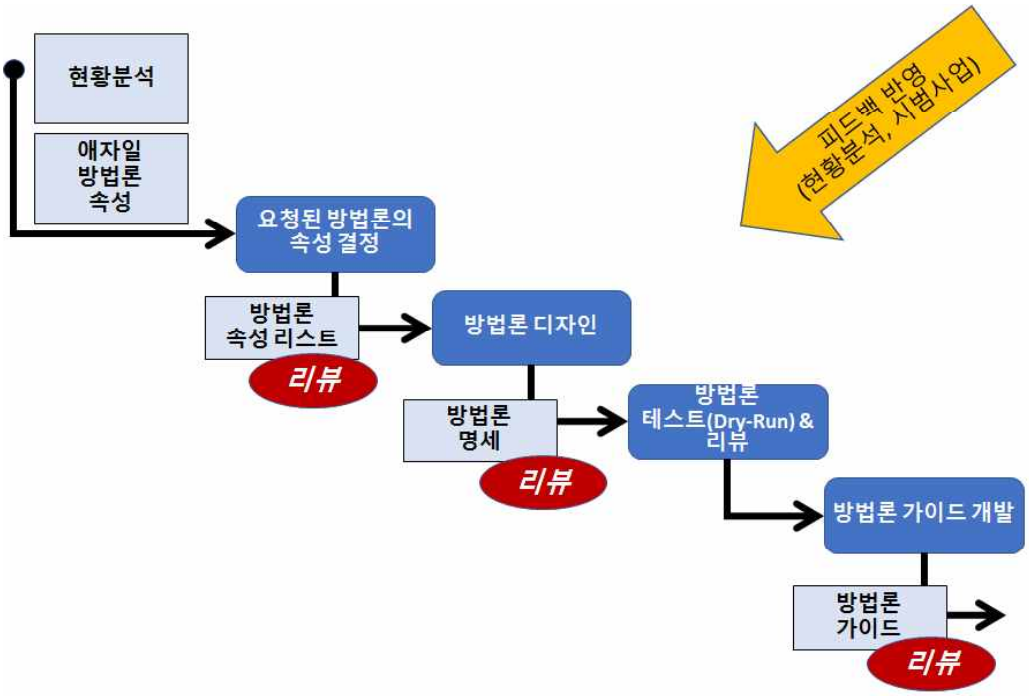
보기	응답	
변화하는 요구사항에 대한 조절 능력	37.70%	23
프로젝트 및 개발 과정의 투명성	27.87%	17
팀의 생산성 및 업무효율성 향상	52.46%	32
제품개발 기간 단축	22.95%	14
비즈니스와 IT 간의 효과적인 소통	32.79%	20
제품 품질 향상	32.79%	20
프로젝트 예측력 향상	21.31%	13
프로젝트 리스크 감소	21.31%	13
총 응답자 수: 61		

✓ 귀하는 애자일을 도입할 때 성공요인이 무엇이라고 생각하나요?

보기	응답	
경영진의 확고한 지원	47.37%	27
경험과 역량 있는 애자일 코치	49.12%	28
충분한 교육	43.86%	25
상세한 애자일 적용 지침 및 가이드	28.07%	16
유연한 조직 문화	45.61%	26
총 응답자 수: 57		

6. 경량 개발 방법론(리드미) 개요 (1/6)

방법론 **개발 과정**은 현황 분석과 애자일 방법론 속성을 분석해 경량 개발 방법론에 포함될 속성을 결정하고 실무 적용을 통해 피드백을 반영함으로 보완하고 개선



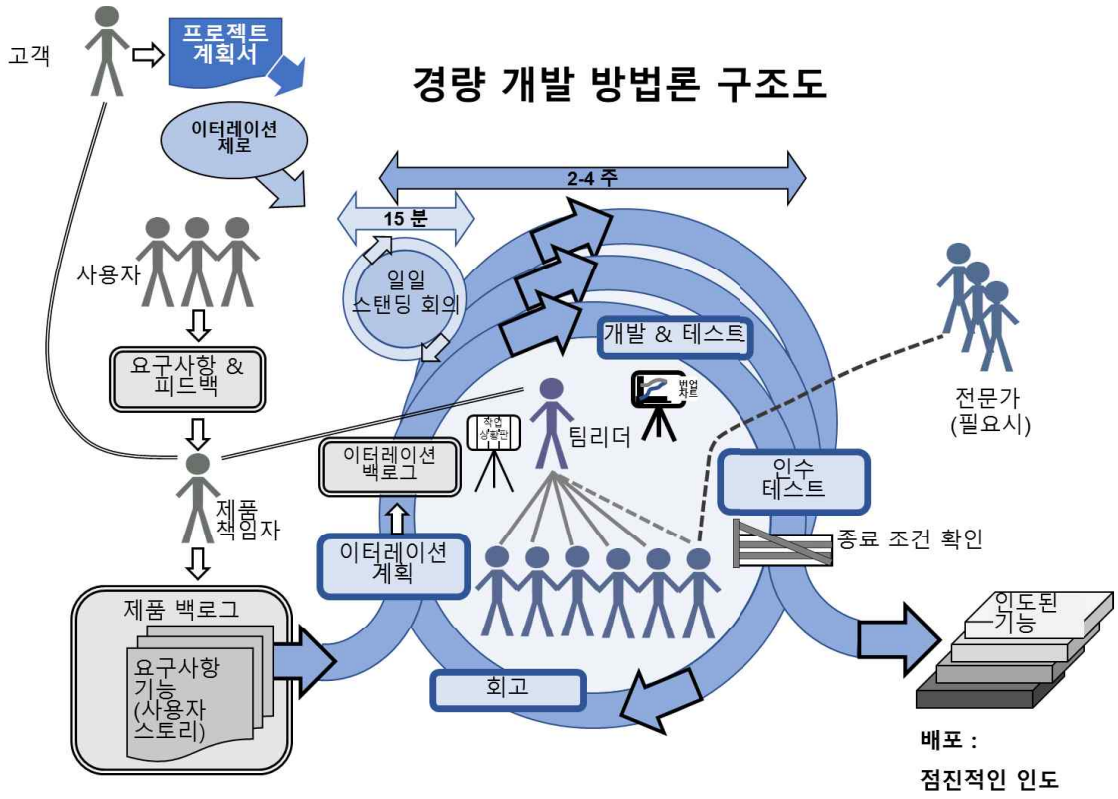
6. 경량 개발 방법론(리드미) 개요 (2/6)

해외 전문가 2명과 국내 전문가 4명이 **71개의 애자일 속성**에 관해 가이드에 포함 되야 할 속성을 1점에서 5점 사이의 점수를 주어서 평균 3.0 이상 된 속성을 우선적으로 가이드에 포함시킴

번호	속성	이사벨	스튜어트	심우곤	조승빈	이재왕	김창준	평균
1	Iterative Development	5	5	5	5	5	5	5.0
2	Timeboxed iterations	4	4	4	2	4	4	3.7
3	Defined Iteration Length	4	4	4	2	4	4	3.7
4	Incremental Development	4	5	5	5	5	5	4.8
5	Feature-Driven Development	4	4	4	3	4	5	4.0
6	Teams–Size-6±3	5	4	4	4	5	3	4.2
7	Teams – Cross-Functional	4	4	4	4	4	4	4.0
8	Teams–Specialists	4	4	2	3	5	3	3.5
9	Teams – Self-Organizing	3	2	2	3	3	3	2.7
10	Teams – Collective Ownership	4	4	4	3	4	3	3.7
11	Teams – Sign up	4	2	2	2	3	3	2.7
12	Teams – team leader	4	4	4	5	4	3	4.0
13	Roles – scrum master	3	2	2	3	2	2	2.3
14	Teams – Responsible for Process	4	4	3	2	3	4	3.3
15	Teams – co-located	4	4	4	4	4	4	4.0
16	Team – team room	4	4	3	2	4	2	3.2
17	Roles – product owner	5	5	5	4	4	3	4.3
18	Product Owner – Continuous Interaction	5	4	5	4	4	5	4.5
19	Product Owner – Product Backlog	5	5	5	5	3	3	4.3
20	Prioritized Product Backlog	4	3	4	4	2	4	3.5

6. 경량 개발 방법론(리드미) 개요 (3/6)

팀 리더가 주도해 팀을 이끌며 이터레이션 백로그가 완료조건이 충족되는지를 확인하기 위해 인수 테스트를 수행하며 필요 시 전문가 그룹이 참여함



6. 경량 개발 방법론(리드미) 개요 (4/6)

방법론 가이드는 크게 역할, 활동, 산출물 작성 방법, 힌트&팁, 사례 등으로 구성됨

역할	활동	산출물	힌트& 팁	사례	기타
고객	프로젝트 계획 수립	프로젝트 계획서	48의 힌트& 팁	47개의 사례	• 적용 방법 • 산출물 템플릿 • 용어 • 탐색기 기능 • 리스크 및 이슈 관리
사용자	이터레이션 제로	제품 백로그			
제품 책임자	제품 백로그 도출 및 관리	릴리즈 계획			
팀 리더	릴리즈 계획	릴리즈 번-업 차트			
팀 멤버	이터레이션 계획	이터레이션 백로그			
전문가(필요 시)	일일 스탠드 업 미팅	사용자 스토리			
	개발 및 테스팅	인수 기준			
	인수 테스팅	태스크 보드			
	회고				
	배포 및 유지보수				

6. 경량 개발 방법론(리드미) 개요 (5/6)

경량 개발 방법론 적용 시 도움이 될 수 있는 **힌트와 팁을 48개 제공함**

10 힌트와 팁

1 힌트와 팁은 어떤 모양인가? 15
2 왜 소프트웨어를 인도(deliver)하기 위해 반복적 점증적 방법론을 사용하나? 16
4 제품 책임자가 너무 바빠서 지속적 상호작용(continuous interaction)을 하기 어려운 경우에는 어떻게 해야 하나? 27
5 팀 리더 및 기술적 전문 지식 31
6 단일 또는 다중 프로젝트 참여 32
7 하나 이상의 프로젝트에 대한 작업 36
8 전문가가 팀 내에서 작업하는 방법 40
9 사용자와 접촉할 수 없는 경우 43
10 바쁜 고객 - 이터레이션 제로 45
11 타임박싱(Timeboxing) 이터레이션 제로 56
12 시스템 제품군 56
13 스토리 카드를 사용하는 이유 59
14 릴리즈 계획은 상위 레벨이다. 62
15 제품 책임자는 항상 보다 많은 것을 원한다! 66
16 이터레이션 백로그 변경 사항 66
17 일일 스탠드업 미팅 - 진행 시간을 짧게 하기 69
18 회고(Retrospectives) 72
19 지속 가능한 페이스(sustainable pace) 72
20 권장되는 첫 단계 개발 프랙티스 72
21 프로젝트 요구 사항에 맞도록 테스팅 프랙티스 개선하기 75
22 권장되는 첫 단계 스토리 테스팅 프랙티스 75
23 전문적 테스트에는 추가적인 도구가 필요할 수 있다. 76
24 팀들마다 제각기 다른 인수 테스팅 프랙티스가 있다. 80
25 권장되는 첫 인수 테스팅 프랙티스 80
26 회고에서 무엇을 기여할지 생각나지 않나요? 87
27 개발 및 유지보수 작업? 91
28 자원 계획 (Planning resources) 93
29 첫 번째 제품 백로그 94
30 완료 정의 변경 97
31 이터레이션 기간 결정하기 98
32 아직 전념하지 않고 있음: 스토리 포인트 평가만 하기 100
33 스토리 포인트 크기에 대해 합의하기 101
34 정기적인 백로그 미팅 104
35 제품 책임자와 바쁜 사용자 107
36 이터레이션 백로그가 많거나 적지 않게 하기 위해 협상하기 112
37 어떤 순서로 사용자 스토리들을 작업 현황판에 게시할 것인가? 114
38 코딩 우선이란 무엇인가? 124
39 회귀 테스트 126
40 지속적 통합이란 무엇인가? 131
41 지속적 전달은 무엇인가? 133
42 지속적 배포란 무엇인가? 134
43 왜 리팩토링은 최고의 해결책이 아닌가? 135
44 왜 짝 프로그래밍인가? 135
45 왜 짝 테스트인가? 136
46 왜 인수 기준이 형식적이어야 하는가? 169
47 인수 기준에 대해 질문하기 169
48 페르소나 사용하기 177

6. 경량 개발 방법론(리드미) 개요 (6/6)

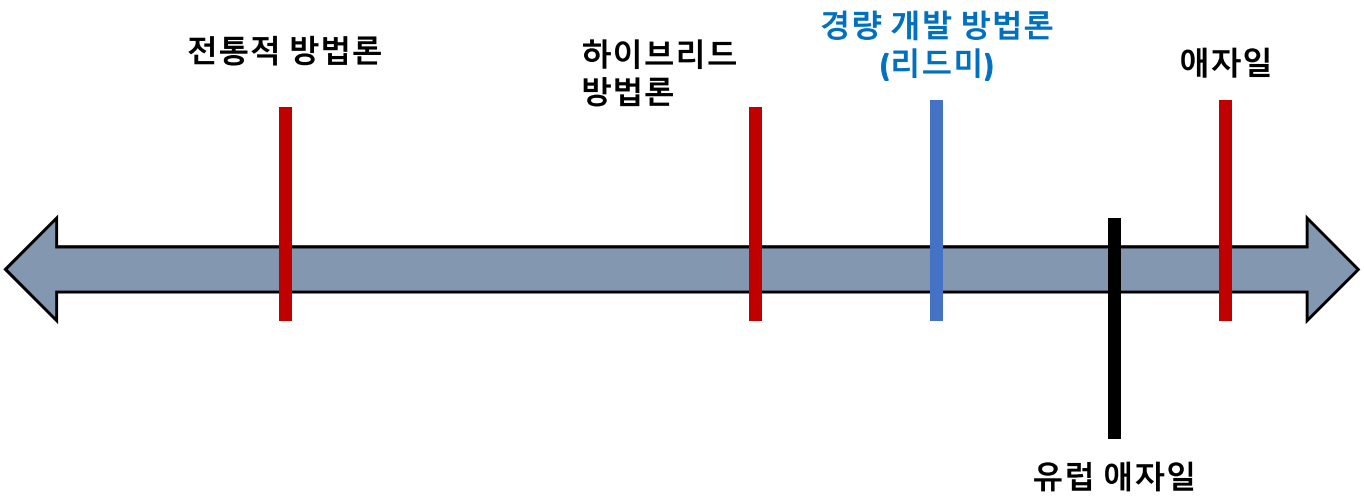
경량 개발 방법론 적용 시 도움이 될 수 있는 **예제를 47개 제공함**

11 예제 목록

구성원이 3 명인 팀과 10 명인 팀의 비교 24
제품 책임자 역할을 대체하는 방법 – 팀 리더가 제품 책임자 역할을 함 27
제품 책임자 역할을 대체하는 방법 28
팀 리더가 기술적 설계의 리더인 경우 32
팀 리더가 기술적 설계의 리더가 아닌 경우 32
팀 리더가 2 개의 팀을 운영하는 경우 33
희소한 자원인 한 팀원에 대한 운영 37
몇 차례이 이터레이션을 위해 팀과 단기간 공동 작업하는 전문가 40
이터레이션 동안 팀의 비정기적 작업하는 전문가 40
상용 소프트웨어를 제작하는 경우의 사용자 대표 43
사내 소프트웨어를 제작하면서 사용자와 직접 대화할 수 있는 경우 43
이터레이션 제로, 바쁜 고객 및 프로젝트 파악 46
프로그램 내의 프로젝트: 오키드밀라이즈 52
전문적 테스팅에 특수한 환경이 필요할 수 있다. 77
너무 바빠서 인수 테스트를 할 수 없는 사용자 81
회고 미팅 88
개발 및 유지보수 작업 91
완료 정의 (Definition of Done) 95
이터레이션 기간의 결정 98
요구 사항에 대한 긴급함의 정도 정하기 103
정기적인 백로그 관리 105
릴리즈 계획의 이유 106
제품 책임자와의 의사소통 108
이터레이션 목표 예시 109
완료 정의 변경 109
이터레이션 기간은 얼마나 되는가? 110
속도는 얼마인가? 110
이터레이션 계획 협의 112
이터레이션 작업 현황판 115
추가적 자원의 필요성 평가 115
일정관리 후 이터레이션 작업 현황판 116
일일 스탠드 업 미팅 정시로 유지하기 118
일일 스탠드 업 미팅 후 장애물 처리하기 119
진척회의를 대체하는 일일 스탠드 업 미팅 119
액션-결과-오브젝트 형식의 기능 137
플립차트에 게시된 목록 상의 '마이트래블(MyTravel)' 프로젝트 목표들 143
'타이거 토이즈'(TigerToys) 프로젝트 계획서(포스터) 144
'오키드밀라이트' 프로젝트 계획서(마인드맵) 145
완료된 스토리 카드 앞면 164
완료된 스토리 카드 뒷면 164
소프트웨어 기능이 아닌 사용자 스토리 166
인수 기준 예시 168
인수 기준 협의 170
인수 기준 충족하기 170
이터레이션 진행중 작업 현황판 173
오키드밀라이트 페르소나 예시– 철순 177
오키드밀라이트 페르소나 예시 - 광 178

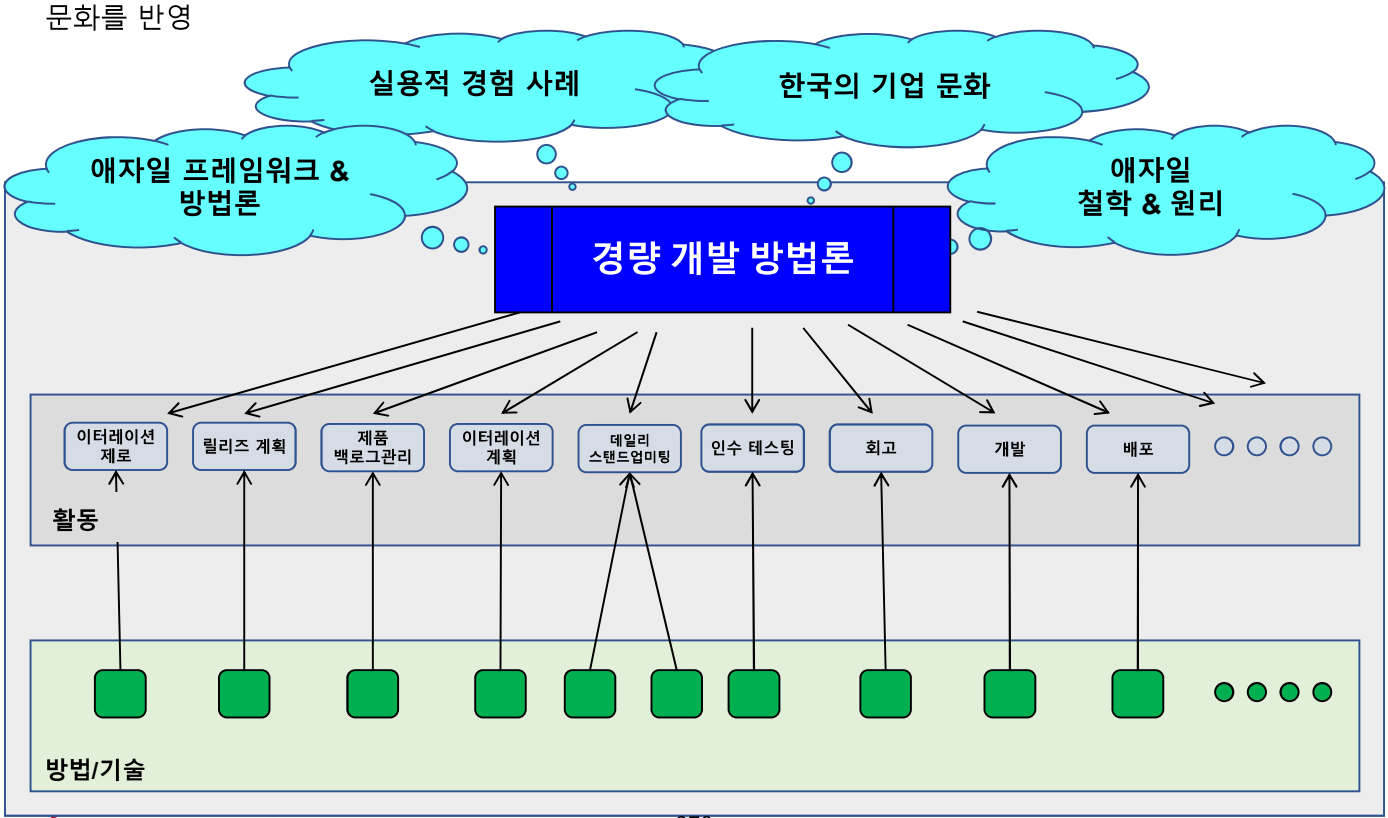
7. 경량 개발 방법론(리드미) 특징 (1/4)

경량 개발 방법론의 **포지션**은 하이브리드 방법론과 순수한 애자일 방법론의 중간으로 애자일 원리를 바탕으로 한국의 수직적 문화를 반영해 개발함



7. 경량 개발 방법론(리드미) 특징 (2/4)

방법론 적용 초기에 **활동을 정의**해 방법 및 기술이 지정돼 있고, **실용적 경험 사례**가 풍부하며 한국의 기업 문화를 반영



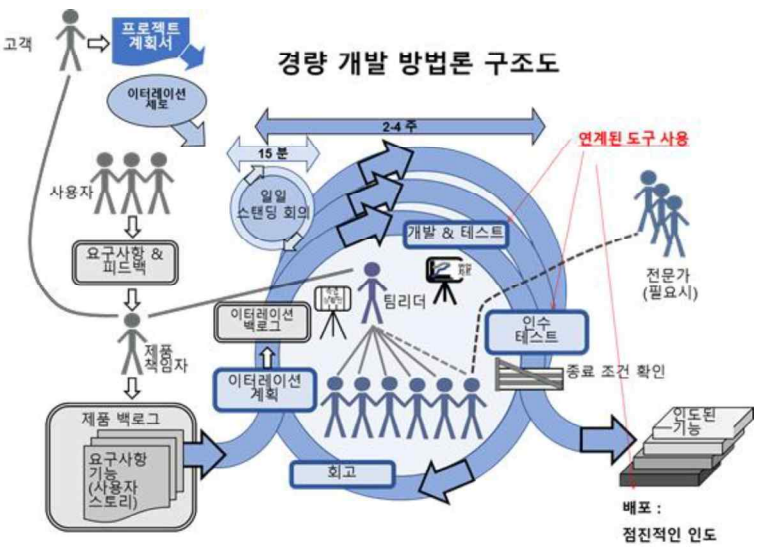
7. 경량 개발 방법론(리드미) 특징 (3/4)

국내 중소 기업의 특징을 분석해 기존 애자일 방법론 대비 국내 중소 기업의 상황을 반영함

기존 애자일	중소 기업의 특징	경량 개발 방법론(리드미)
스크럼의 경우 스크럼마스터의 역할이 필요함	프로세스 관련 전문적인 인력 채용이 어렵다. (ex 스크럼마스터)	스크럼 마스터 역할이 없음. 팀 리더의 역할은 제품 책임자, 팀원과 협력해 의사소통을 하며 팀원들을 코치함
수평적 문화를 강조함	대부분 수직적 문화임	수직적 문화 인정
운영에 대한 가이드 부족	개발과 운영을 동시에 하는 경우가 많다.	운영에 대한 가이드 제시
가이드가 추상적이어서 구체적으로 적용하려면 학습에 상당한 시간이 걸림	프로세스를 어떻게, 무엇부터 적용해야 할 지를 모른다.	가이드 적용 시 구체적인 활동을 제시해 상대적으로 쉽게 적용할 수 있음
가이드 부족	프로세스 적용 시에도 어려움에 부딪히면 어떻게 해야 할 지를 잘 모르기 때문에 중도에 포기하는 경우가 많다.	적용 시 도움을 줄 수 있는 다양한 상황의 사례, 힌트&팁 제공
교육 및 학습에 고비용임	품질, 프로세스에 대한 인식 부족 및 교육 기회 부족	교육 및 학습에 저비용임

7. 경량 개발 방법론(리드미) 특징 (4/4)

방법론과 도구를 연계했으며 지속적 통합과 지속적 전달 및 배포 구축을 위한 도구 가이드 제공

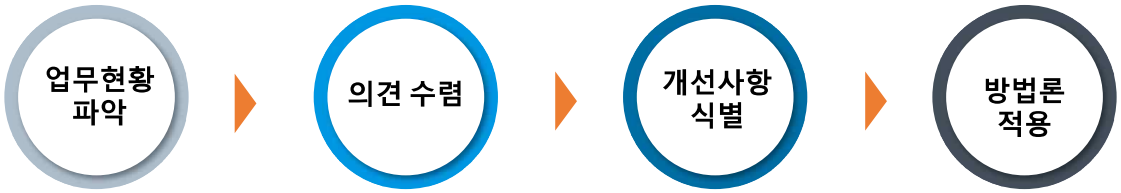


기술 활동	연계된 도구
지속적 통합	Jenkins, SVN, Git, Maven, Ant, Findbugs,
자동화된 빌드& 테스트	Maven, Ant, Junit, Selenium, Findbugs
자동화된 단위&회귀 테스트	Junit, Selenium, Findbugs
지속적 전달 및 배포	Jenkins, Nexus
인수 테스트 주도개발	Fitness

8. 경량 개발 방법론(리드미) 실무 적용 (1/4)

실무 프로젝트 적용 절차는 대상 기업의 업무현황 파악→ 의견 수렴→개선사항 식별→ 방법론 적용의 절차대로 적용

방법론 실무 적용 절차



- 현장 사용자 요구를 바탕으로 영업과 경영진이 새로운 서비스나 기능 개선을 요청하면 기획자는 경영진과 밀접하게 소통하면서 기획서를 작성
- 완료된 기획서는 레드마인에 일감으로 등록
- 완성된 기획서를 바탕으로 디자인 작업이 먼저 수행되며 디자인이 필요 없는 작업은 개발팀에서 업무를 수행. 업무할당은 팀장 주도로 디자인 및 안드로이드, IOS, 웹 개발 등을 맡은 각각의 담당자에게 분배

- 부서간(기획자, 디자인, 개발팀)에 소통이 부족하며 변경 사항에 대한 공유가 잘 이루어지지 않음으로 인해 갈등이 발생됨
- 개발자들은 팀장의 지시를 받아서 자기 일만 수행하다 보니 자기 업무 외에는 다른 업무에 대하여 관심이 없음
- 개발자들간에 기술적인 공유가 거의 일어나지 않음
- 개발자들에게 동시에 여러 가지 일을 요청하고 있어 멀티태스킹으로 인한 일정준수가 어려움

- 업무 우선순위 미팅 및 제품책임자의 역할 정립
- 일일 스탠드업 미팅을 통한 파트간 소통 강화
- 작업 진행 상황판(Task Board) 구축을 통한 업무진행 상황 공유 및 협력
- 주기적인 이터레이션 계획을 통한 업무 계획
- 주기적인 회고 수행을 통한 프로세스 개선
- 서로가 배려하고 존중하는 문화 만들기

- 경량 개발 방법론 교육
- 업무 흐름에 대한 시각화 및 완료 정의. 전체 업무 진행상황의 가시성을 높이기 위하여 모든 직원들이 볼 수 있게끔 기획팀과 개발팀의 작업 진행 상황판을 벽면에 구축
- 업무 우선순위 미팅
- 이터레이션 계획
- 일일 스탠드업 미팅
- 회고
- 도구 활용 - 지속적 통합 도구인 Jenkins를 설치해 구축. 빌드 도구인 gradle을 추가적으로 가이드

8. 경량 개발 방법론(리드미) 실무 적용 (2/4)

W사 사례로 방법론 적용은 교육, 업무 흐름에 대한 시각화 및 완료 정의, 이터레이션 계획, 일일 스탠드업 미팅, 회고, 도구 활용 등을 수행

W사 조직 개요

- 체육학원 (체력측정 및 출결 알림 시스템 등) 및 일반 학원(출결, 수납, 셔틀버스 위치관제 등) 관리 프로그램, 식생활 개선 프로그램 등을 개발하는 전문 솔루션업체
- 전체 직원은 20여명 정도로 영업, 기획, 개발팀 등으로 구성
- **개발팀은** 웹개발자 2명, 앱 개발자 4명(안드로이드, iOS), 디자이너 1명, 테스터 1명으로 **총 8명으로 구성**
- 기획자 2명은 개발팀과 밀접하게 일하고 있으며 전체 기획과 개발팀을 이끌고 있는 팀장이 1명 있음

적용 기간 (약 2개월)
17년 9월 하순~11월 하순

<교육 진행>

<컨설팅 수행>

적용 효과

- 자기가 해야 할 일을 2주 단위로 **자발적으로 세울 수가 있어서 좋았다.**
- 구성원들 중에 대부분은 일일 스탠드업 미팅을 긍정적으로 생각하게 되었다. 서로가 **업무를 공유하고 필요 시 언제라도 도움을 요청할 수가 있어서** 좋게 생각했다. 다만 시간이 약간 길어지는 경향이 있어서 그것에 대한 불만이 존재했다.
- 업무 진행 상황판은 **전체 업무를 가시적으로 볼 수 있고 무엇에 병목이 걸려있는지 바로 알 수가 있어서 좋았다.**
- 부서간의 소통이 늘어남으로써 기존에 존재했던 갈등이 많이 줄어들었다.

8. 경량 개발 방법론(리드미) 실무 적용 (3/4)

SW사 사례

S사 조직 개요

- POS, 금전등록기, 미니프린터, 카드단말기 등을 개발 및 생산하는 업체
- 내부 조직은 연구소에 시스템 개발, 소프트웨어 개발, 기구개발, QA 팀으로 구성
- 경량방법론을 적용한 조직은 소프트웨어 개발팀이며 3개 파트 (금전등록기, 안드로이드 POS, 카드단말기)로 나누어져 있는데 안드로이드 POS 파트에 적용됨
- 적용된 팀 구성은 POS 파트리더1명과 개발자 4명, QA팀의 테스터 2명으로 **7명으로 구성**
- 미국에 협력 업체 1인이 있음

적용 기간 (약 2개월)
17년 9월 하순~11월 하순

<교육 진행>

<컨설팅 수행>

적용 효과

- 작업 진행상황판을 통하여 팀간, 담당자간의 업무 진행 현황을 **한눈에 알아 볼 수 있는 것이 좋았다.**
- 업무의 유형이 구분되어 있어 **우선적으로 해야 할 업무의 판단이 쉬워짐**
- 매일 진행되는 스탠드 미팅을 통하여 **담당 업무의 의견 전달이 명확해짐**
- 과거보다 **QA와 협력이 잘되어** 이슈를 빨리 해결하는 것 같다.
- 일일 미팅을 통하여 자잘한 회의들이 많이 없어졌다.
- Findbugs을 도입해 Null Exception과 같은 **잠재된 치명적 결함을 사전에 검출 할 수 있었다.**

8. 경량 개발 방법론(리드미) 실무 적용 (4/4)

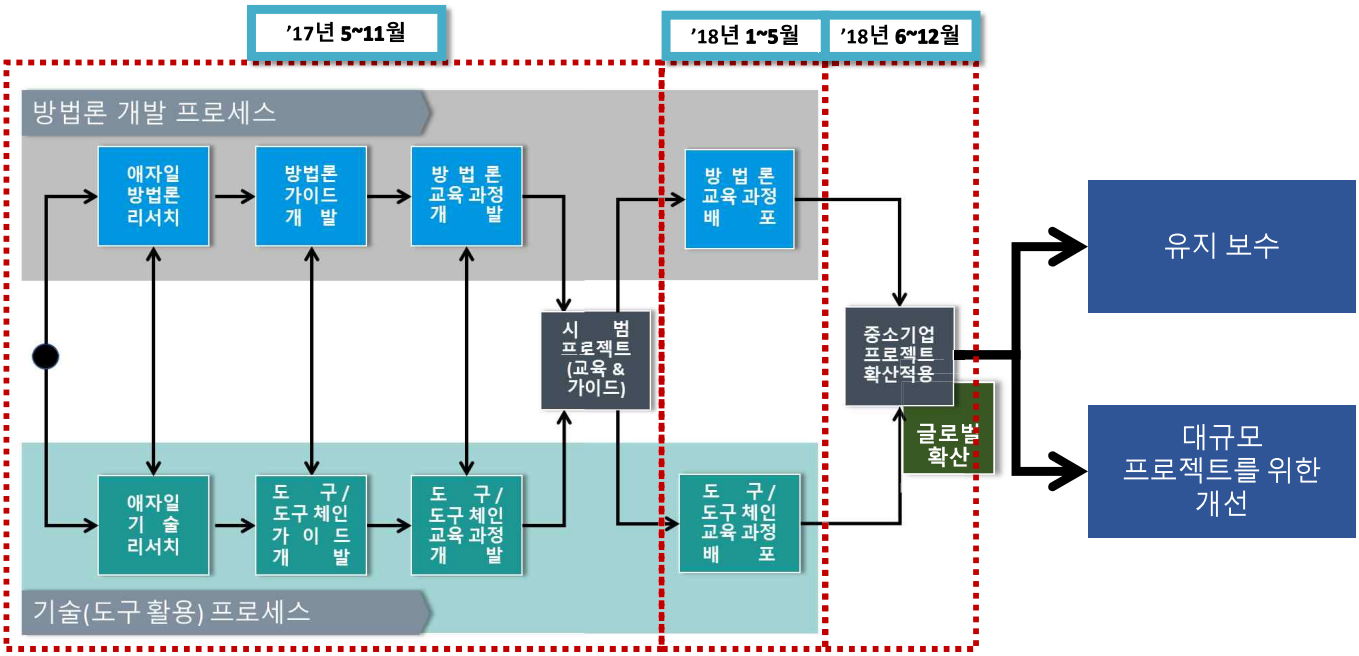
실무 적용 후 적용 교훈

- 경량 개발 방법론에는 다양한 활동이 있으나 한꺼번에 적용하는 것은 바람직하지 않으며 해당 조직에 **가치**가 있는 것을 **중심**으로 **점진적**으로 **도입**하는 것이 바람직함
- W사, S사 모두 도구 활용보다는 경량 개발 방법론 적용에 관심이 많았으며 짧은 실무 프로젝트 기간에 방법론 적용과 도구 활용 적용을 모두 하기에는 기간이 부족함. **최소 4개월에서 6개월** 정도의 기간이 필요함
- 팀원들이 개인 플레이에서 팀 플레이로 바로 변화되기는 어려우므로 이것을 변화시키기 위해서는 팀 리더가 팀원들에게 팀플레이를 하도록 지속적으로 독려하는 활동이 필요함
- 회고 및 이터레이션 계획 등을 주기적으로 수행하기 위해서는 주당 0.5일은 회의를 위한 시간으로 할애하는 것이 좋음. 이런 미팅이 끝나고 남는 시간은 팀원들이 하고 싶은 일을 할 수 있도록 자율성을 주는 것이 바람직함
- W사 - 평상시에 팀원들은 기존 솔루션에 대한 문의 및 장애 응대, 결함 해결 등을 병행하고 있었기에 업무 할당은 가용공수의 70% 정도만 할당하고 30%는 여유시간을 줌
  - 사용자 스토리에 대한 포인트 추정을 처음 시도해 어려워하여 평균 투입공수로 추정
- S사 - 미국 협력업체에서 발생되는 이슈 해결이 주된 업무여서 매일 아침 업무의 우선순위를 결정하고 이를 빠르게 해결하는 쪽에 적용의 초점을 맞춤

9. 관련 향후 연구 대상

글로벌 확산 – 한국이 글로벌 리더십 견인
- 국제표준 반영 ISO/IEC, JTC1, SC7 애자일(Agile) 스터디 그룹과 함께 국제 표준화 작업 진행

중소기업에 확산할 수 있도록 세미나 교육 등을 수행하고 유지 보수함



참고 문헌

1. '한국, 데브옵스·애자일 성숙도 낮다', ZDNet Korea, 2017.3.19
2. 메리 포펜딕·톰 포펜딕 지음, '린 소프트웨어 개발의 적용', 엄위상 외 2인 옮김, 위키북스, 2007, pp.86-87
3. 데이비드 J 앤더슨외 1인 지음, ' 에센셜 칸반 요약 가이드', 조승빈 외 2인 옮김, 2016
4. 이재왕 지음, '애자일 & 스크럼 프로젝트 관리', 길벗출판사, 2016
5. Sridhar Nerur, Radhakanta Mahapatra, George Mangalapa : Challenges of migrating to agile methodologies, Commun. Of ACM 48, 5(May 2005)
6. Daniel J. Fernandez & John D. Fernandez : Agile Project Management —Agilism versus Traditional Approaches, Journal of Computer Information Systems Winter 2008-2009
7. What is Hybrid Agile, Anyway?, https://www.agilealliance.org/what-is-hybrid-agile-anyway/
8. The State of Scrum report, 2017 edition, Scrum Alliance
9. Darrell K. Rigby, Jeff Sutherland, Hirotaka Takeuchi : Embracing Agile, May 2016 issue (pp.40–48, 50) of Harvard Business Review.
10. Vishnu Vinekar, Craig W. Slinkman, and Sridhar Nerur : Can agile and traditional systems development approaches coexist?, www.ism-journal.com, 2006
11. 에릭 리스 지음, '린 스타트업' , 이창수외 1인 옮김, 인사이트, 2012
12. 효과적인 데브옵스(DevOps) 적용을 위한 10가지 실행방안, 정보통신산업진흥원 동향브리핑, 제 110호, 2014
13. When-Why-How to use the Hybrid model, https://intland.com/blog/agile/when-why-how-to-use-the-hybrid-model/
14. 11th annual State of Agile Report, Versionone, 2017
15. Agile & DevOps Survey Report : Accelerating Velocity and Customer Value with Agile and DevOps, CA Technologies, 2016
16. SK㈜C&C, 클라우드 전환 코칭 서비스 확대, ZDNet Korea, 2017.8.8
17. 카카오의 일하는 방식, www.kakaocorp.com
18. IT프로젝트 애자일 개발방식 확산, 디지털타임스, 2010.10.18
19. 블로그 : '애자일 적용을 올바르게 하기 위해서는'
    http://blog.naver.com/clippe/30166572173
20. 블로그 : '애자일. 한국에서 실패하는 이유' http://blog.naver.com/PostView.nhn?blogId=cloud2519&logNo=220744431349
21. MATTHEW BADGLEY : Top 10 Tips for Measuring Agile Success, https://blog.versionone.com/top-10-tips-agile-success/
22. 속도와 혁신의 마법, 애자일, 전사·전업무 확산 가이드, IDG Deep Dive
23. 마이크 콘 : '경험과 사례로 풀어낸 성공하는 애자일', 인사이트, 2012, PP 486-509
24. Tsun Chow, Dac-Buu Cao, A survey study of critical success factors in agile software projects, The Journal of Systems and Software 81 (2008) 961–971

감사합니다.

29

TPC-DI 기반의 EAI 솔루션 주요기능 테스트 사례

한국정보통신기술협회 강건희, 권하은, 한종혁, 김상기

2018. 1.

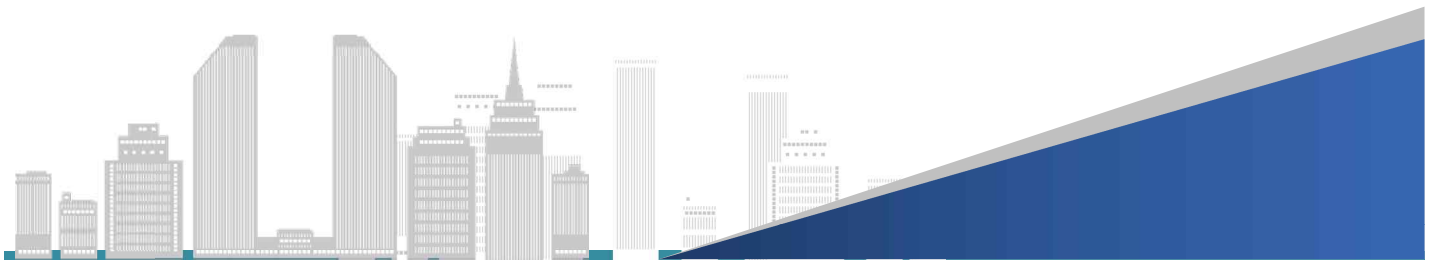
TTA 한국정보통신기술협회 Telecommunications Technology Association

목차

- 연구 배경
- 관련 연구
- 본 론
  - 주요기능 테스트
  - 테스트 시나리오 및 데이터 설계
  - 테스트 환경 구축
  - 주요기능 테스트 수행
- 결 론

Ⅰ 개요

- EAI 정의 및 목적
- EAI 솔루션 분류
- EAI 솔루션 평가의 현상 및 문제점



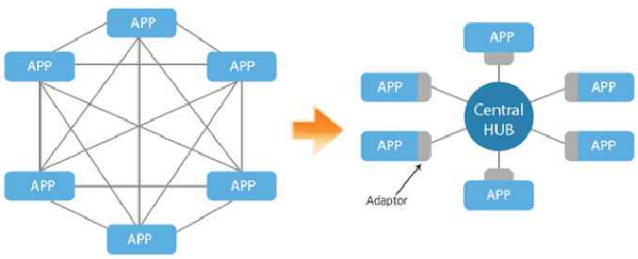
EAI 솔루션 정의 및 목적

EAI 정의

기업 응용 프로그램 통합(Enterprise Application Integration)의 약자이며, 일반적으로 국내 IT시장에서는 기업 내 응용 프로그램의 구조적 통합을 의미함

EAI 기대효과

- 여러 시스템의 정보가 일관되게 유지되는 것을 보증(정보의 통합)
- 응용 프로그램 간 비즈니스 프로세스를 연결(프로세스 통합)
- 어떤 응용 프로그램이 다른 밴더의 응용 프로그램으로 변경되더라도, 비즈니스 규칙을 다시 구현할 필요가 없음(벤더에 대한 독립)
- 기존 분산 시스템에서의 데이터 포맷, 프로세스 지연 등과 같은 문제점 해결
- 1:1 통합 방식의 과도한 비용과 통합 지연에 대한 문제점 해결



[Point-to-Point 연계] [EAI 연계]

EAI 솔루션 분류

EAI 적용 방법

구분	특징
Point-to-Point (포인트 투 포인트 방식)	▪ 기초적인 어플리케이션 통합 방법으로 1:1 통합 방법 ▪ 복잡한 인터페이스, 일괄전송처리방식으로 인한 데이터 흐름의 지체 현상 및 과다한 유지비용 발생
Hub & Spoke (허브 & 스포크 방식)	▪ 어플리케이션 간 미들웨어(Hub, 허브)를 통해 통신 ▪ 중앙집중 방식으로 단일접점인 허브 시스템을 통해 데이터를 전송 ▪ 허브 내 여러 큐(Queue)를 통해 데이터를 입력 및 출력 ▪ 데이터 전송이 보장되며, 유지/보수 비용 절감 가능
Messasing Bus (메시징 버스 방식)	▪ 어플리케이션 간 미들웨어(Bus, 버스)를 통해 통신 ▪ 병목 현상 및 단일 장애점의 발생 가능성을 최소화 ▪ 어댑터가 각 시스템과 버스를 연결하는 구조 ▪ 높은 확장성을 가지며, 대용량 데이터 처리 가능
Hybrid (하이브리드 방식)	▪ Hub & Spoke 방식과 Messasing Bus 방식의 혼합형 ▪ 통합 대상 시스템을 여러 그룹으로 분할하고, 그룹 내부에서는 Hub & Spoke 방식을 사용하고, 그룹 간에는 Messasing Bus 방식을 사용
Peer-to-Peer (피어 투 피어 방식)	▪ 미들웨어를 사용하지 않고, 각 어플리케이션의 피어서버(Peer Server)를 Point-to-Point 방식으로 연결 ▪ Hub & Spoke 방식과 Messasing Bus 방식에 비해 저렴한 비용으로 통합 가능

EAI 솔루션 평가의 현상 및 문제점

다양한 EAI 솔루션의 등장

데이터 통합 및 이동의 중요성에 따라 고객 요구에 맞는 다양한 기능을 개발한 EAI 솔루션이 시장에 출시됨.

표준화된 EAI 솔루션 평가 방법의 필요성

- 데이터 획득(통합, 이동 등) 시 다양한 시나리오에 대한 표준화된 평가 방법의 부재
- 복잡하고 다양한 경우의 수를 대비한 테스트 데이터 준비의 한계
- 테스트 환경 제약 조건(서버, OS 등)에 따른 환경 구축의 어려움

→ 방안 연구 → **기업의 EAI 솔루션 선택**
- 의사 결정 시 활용
- 기능/성능 비교

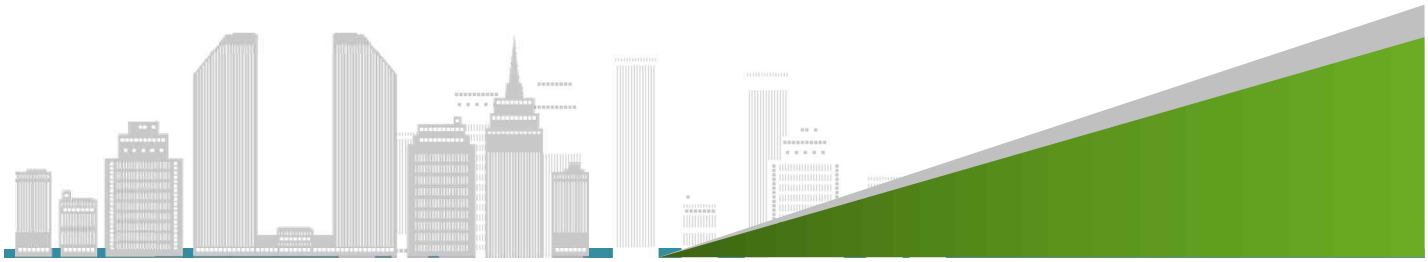
정부기관의 SW분리발주 증가

- 여러 발주기관의 동일 SW 분리발주
- 동일 SW의 지속적인 기능 시험 반복
- 반복시험으로 인한 SW업체의 인력(인건비) 및 시험 수수료 부담

→ 방안 연구 → **주요기능 테스트 필요**
- 기능시험 결과 재사용
- 업체 부담 경감

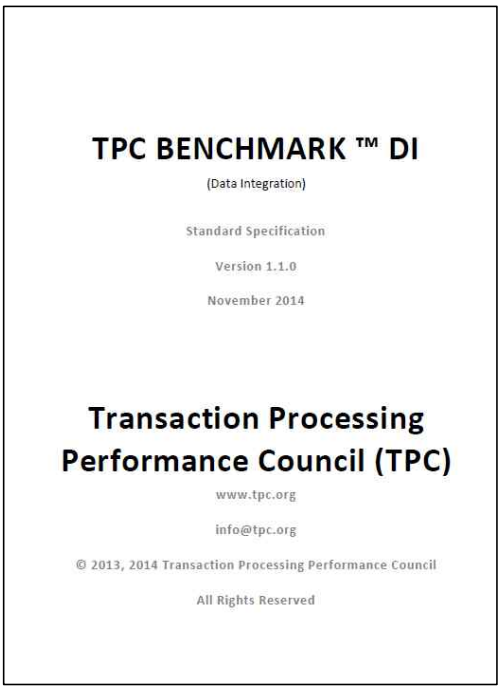
Ⅱ 관련 연구

- TPC-DI
- 해외 연구 논문
- 주요기능 테스트



TPC-DI

TPC-DI 개요

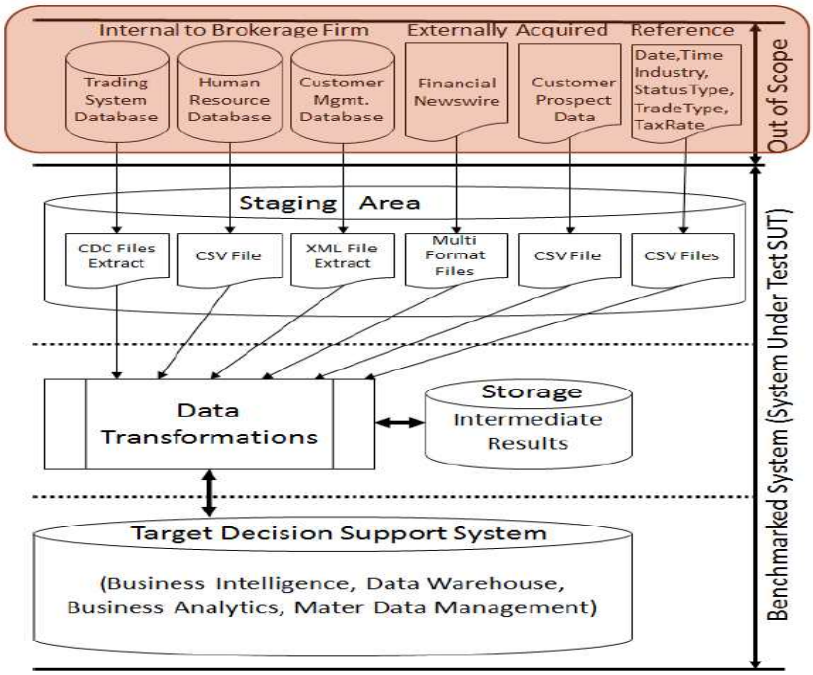

TPC BENCHMARK ™ DI
(Data Integration)
Standard Specification
Version 1.1.0
November 2014
Transaction Processing Performance Council (TPC)
www.tpc.org
info@tpc.org
© 2013, 2014 Transaction Processing Performance Council
All Rights Reserved

- 데이터 이동 및 통합하는 도구의 성능 테스트 활용
- 다양한 데이터 통합(DI) 도구의 비교를 위한 표준 방법 제공
- TPC-DI-Tool을 통하여 필요한 크기만큼의 데이터웨어하우스(DW) 데이터를 준비할 수 있음
- 다음과 같은 특징이 있음
  - 대량 데이터의 조작 및 로딩
  - Error 오류 검사
  - 데이터 유형 변환
  - 집계 연산
  - 데이터 업데이트
- http://www.tpc.org/tpcdi/default.asp

* TTA 표준용어 기준 번역

TPC-DI

TPC-DI 범위



➔ TPC-DI-Tool을 통해 생성
> java –jar DIGen.jar

➔ 생성된 파일을 타겟시스템에 적재하기 위한 명세를 제공함

8

TPC-DI 데이터 모델 선정

TPC-DI 소스 스키마

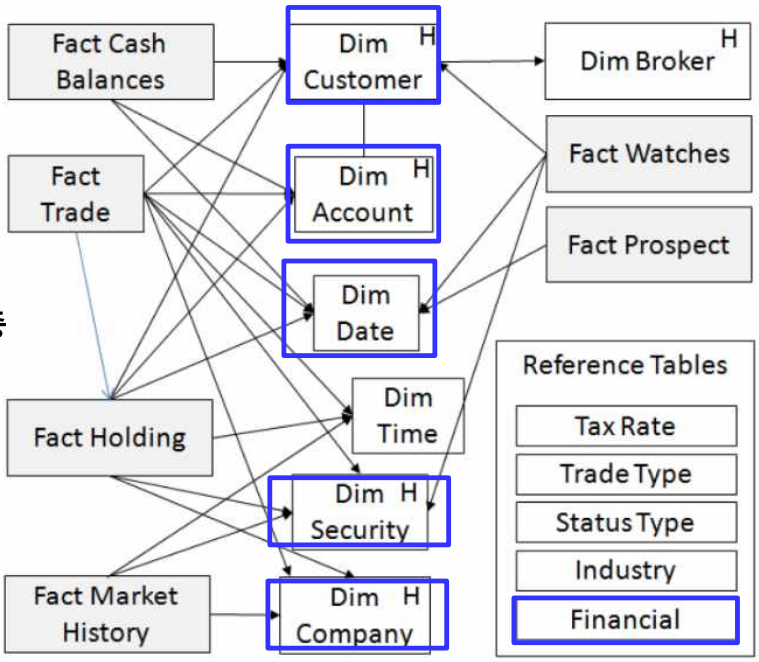
- **18개 다른 테이블**
- **다양한 종류의 포맷**
  - **CSV (Comma Separated)**
  - **CDC (Change Data Capture)**
  - **Multi Record**
  - **DEL (Pipe Delimited)**
  - **XML**
- **Historical Load와 Incremental Load 구분**
- **테스트 시 전체 데이터를 사용하지는 않음**

Source Table	Format	H	I
Account.txt	CDC		✓
CashTransaction.txt	DEL/CDC	✓	✓
Customer.txt	CDC		✓
CustomerMgmt.xml	XML	✓	
DailyMarket.txt	DEL	✓	✓
Date.txt	DEL	✓	
Time.txt	DEL	✓	
FINWIRE	Multi-record	✓	
HoldingHistory.txt	DEL	✓	✓
HR.csv	CSV	✓	
Industry.txt	DEL	✓	
Prospect.csv	CSV	✓	✓
StatusType.txt	DEL	✓	
TaxRate.xt	DEL	✓	
TradeHistory.txt	DEL	✓	
Trade.txt	DEL/CDC	✓	✓
TradeType.txt	DEL	✓	
WatchItem.txt	DEL/CDC	✓	✓

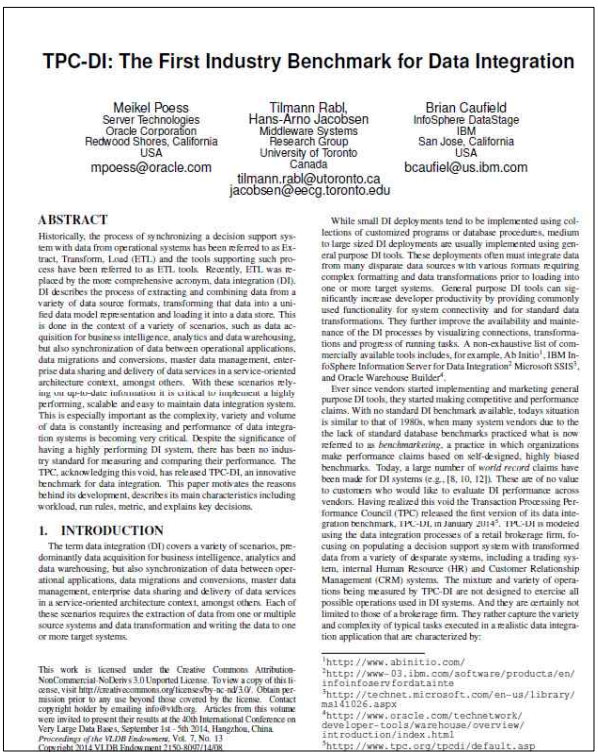
TPC-DI 데이터 모델 선정

TPC-DI 타겟 스키마

- **6개 Fact 테이블**
- **7개의 Dimension 테이블**
- **5개의 Reference 테이블**
- **테스트 수행 시에는 소스 스키마 중 선택된 일부 파일로 생성 가능한 일부 테이블만을 대상으로 함.**



해외 연구 논문



TPC-DI: The First Industry Benchmark for Data Integration

Meikel Poess
Server Technologies
Oracle Corporation
Redwood Shores, California
USA
mpoess@oracle.com

Tilmann Rabl, Hans-Arno Jacobsen
Middleware Systems Research Group
University of Toronto
Canada
tilmann.rabl@utoronto.ca
jacobsen@eecg.toronto.edu

Brian Caufield
InfoSphere DataStage
IBM
San Jose, California
USA
bcaufiel@us.ibm.com

ABSTRACT

Historically, the process of synchronizing a decision support system with data from operational systems has been referred to as Extract, Transform, Load (ETL) and the tools supporting such process have been referred to as ETL tools. Recently, ETL was replaced by the more comprehensive acronym, data integration (DI). DI describes the process of extracting and combining data from a variety of data source formats, transforming that data into a unified data model representation and loading it into a data store. This is done in the context of a variety of scenarios, such as data acquisition for business intelligence, analytics and data warehousing, but also synchronization of data between operational applications, data migrations and conversions, master data management, enterprise data sharing and delivery of data services in a service-oriented architecture context, amongst others. With these scenarios relying on up-to-date information it is critical to implement a highly performing, scalable and easy to maintain data integration system. This is especially important as the complexity, variety and volume of data is constantly increasing and performance of data integration systems is becoming very critical. Despite the significance of having a highly performing DI system, there has been no industry standard for measuring and comparing their performance. The TPC, acknowledging this void, has released TPC-DI, an innovative benchmark for data integration. This paper motivates the reasons behind its development, describes its main characteristics including workload, run rules, metric, and explains key decisions.

1. INTRODUCTION

The term data integration (DI) covers a variety of scenarios, predominantly data acquisition for business intelligence, analytics and data warehousing, but also synchronization of data between operational applications, data migrations and conversions, master data management, enterprise data sharing and delivery of data services in a service-oriented architecture context, amongst others. Each of these scenarios requires the extraction of data from one or multiple source systems and data transformation and writing the data to one or more target systems.

While small DI deployments tend to be implemented using collections of customized programs or database procedures, medium to large sized DI deployments are usually implemented using general purpose DI tools. These deployments often must integrate data from many disparate data sources with various formats requiring complex formatting and data transformations prior to loading into one or more target systems. General purpose DI tools can significantly increase developer productivity by providing commonly used functionality for system connectivity and for standard data transformations. They further improve the availability and maintenance of the DI processes by visualizing connections, transformations and progress of running tasks. A non-exhaustive list of commercially available tools includes, for example, Ab Initio[1], IBM InfoSphere Information Server for Data Integration[2] Microsoft SSIS[3], and Oracle Warehouse Builder[4].

Ever since vendors started implementing and marketing general purpose DI tools, they started making competitive and performance claims. With no standard DI benchmark available, todays situation is similar to that of 1980s, when many system vendors due to the the lack of standard database benchmarks practiced what is now referred to as *benchmarketing*, a practice in which organizations make performance claims based on self-designed, highly biased benchmarks. Today, a large number of *world record* claims have been made for DI systems (e.g., [8, 10, 12]). These are of no value to customers who would like to evaluate DI performance across vendors. Having realized this void the Transaction Processing Performance Council (TPC) released the first version of its data integration benchmark, TPC-DI, in January 2014[5]. TPC-DI is modeled using the data integration processes of a retail brokerage firm, focusing on populating a decision support system with transformed data from a variety of desparate systems, including a trading system, internal Human Resource (HR) and Customer Relationship Management (CRM) systems. The mixture and variety of operations being measured by TPC-DI are not designed to exercise all possible operations used in DI systems. And they are certainly not limited to those of a brokerage firm. They rather capture the variety and complexity of typical tasks executed in a realistic data integration application that are characterized by:

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 40th International Conference on Very Large Data Bases, September 1st - 5th 2014, Hangzhou, China.
Proceedings of the VLDB Endowment, Vol. 7, No. 13
Copyright 2014 VLDB Endowment 2150-8097/14/08.

[1] http://www.abinitio.com/
[2] http://www-03.ibm.com/software/products/en/infoinfoserverfordatain
[3] http://technet.microsoft.com/en-us/library/ms141026.aspx
[4] http://www.oracle.com/technetwork/developer-tools/warehouse/overview/introduction/index.html
[5] http://www.tpc.org/tpcdi/default.asp

Benchmarking ETL Workflows

Alkis Simitsis[1], Panos Vassiliadis[2], Umeshwar Dayal[1], Anastasios Karagiannis[2], Vasiliki Tziovara[2]

[1] HP Labs, Palo Alto, CA, USA.
{alkis, Umeshwar.Dayal}@hp.com
[2] University of Ioannina, Dept. of Computer Science, Ioannina, Hellas
{pvassil, ktasos, vickit}@cs.uoi.gr

Abstract. Extraction–Transform–Load (ETL) processes comprise complex data workflows, which are responsible for the maintenance of a Data Warehouse. A plethora of ETL tools is currently available constituting a multi-million dollar market. Each ETL tool uses its own technique for the design and implementation of an ETL workflow, making the task of assessing ETL tools extremely difficult. In this paper, we identify common characteristics of ETL workflows in an effort of proposing a unified evaluation method for ETL. We also identify the main points of interest in designing, implementing, and maintaining ETL workflows. Finally, we propose a principled organization of test suites based on the TPC-H schema for the problem of experimenting with ETL workflows.

Keywords: Data Warehouses, ETL, benchmark.

1 Introduction

Data warehousing is a technology that enables decision-making and data analysis in large organizations. Several products are available in the market and for their evaluation, the TPC-H benchmark has been proposed as a decision support benchmark [16]. TPC-H focuses on OLAP (On-Line Analytical Processing) queries and it mainly deals with the data warehouse site. Another version termed TPC-DS has been around for the last few years, but this version is still in a draft form [11, 15]. TPC-DS considers a broader picture than TPC-H including the whole flow from the sources to the target data warehouse. However, it partially covers the data warehouse maintenance part, considering only simple mechanisms for inserting and deleting tuples.

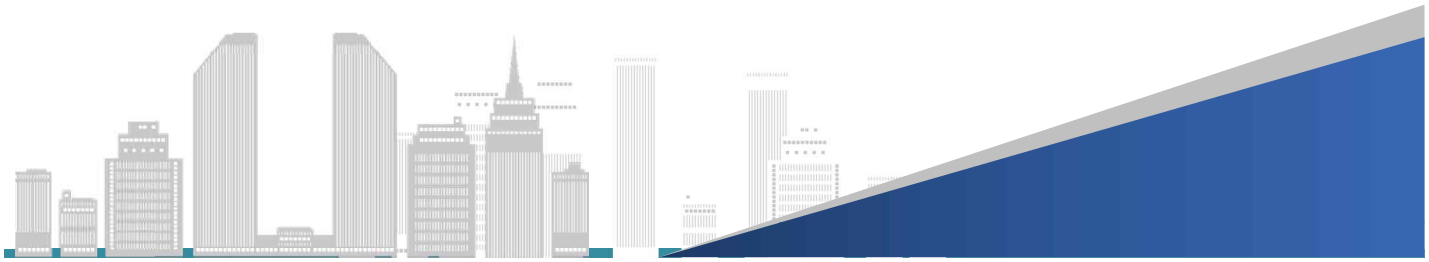
To populate a data warehouse with up-to-date records extracted from operational sources, special tools are employed, called *Extraction – Transform – Load* (ETL) tools, which organize the steps of the whole process as a workflow. To give a general idea of the functionality of these workflows we mention their most prominent tasks, which include: (a) the *identification* of relevant information at the source side; (b) the *extraction* of this information; (c) the *transportation* of this information to the Data Staging Area (DSA), where most of the transformation usually take place; (d) the *transformation*, (i.e., customization and integration) of the information coming from multiple sources into a common format; (e) the *cleansing* of the resulting data set, on the basis of database and business rules; and (f) the *propagation* and loading of the data to the data warehouse and the refreshment of data marts.

TPCDI: The First Industry Benchmark for Data Integration,
Meikel Poess Server Technologies Oracle Corporation
Redwood Shores, California USA

Benchmarking ETL Workflows
Alkis Simitsis, Panos Vassiliadis, Umeshwar Dayal

III 본 론

- 주요기능 테스트
- 테스트 시나리오 및 데이터 설계
- 테스트 환경 구축
- 주요기능 테스트 수행

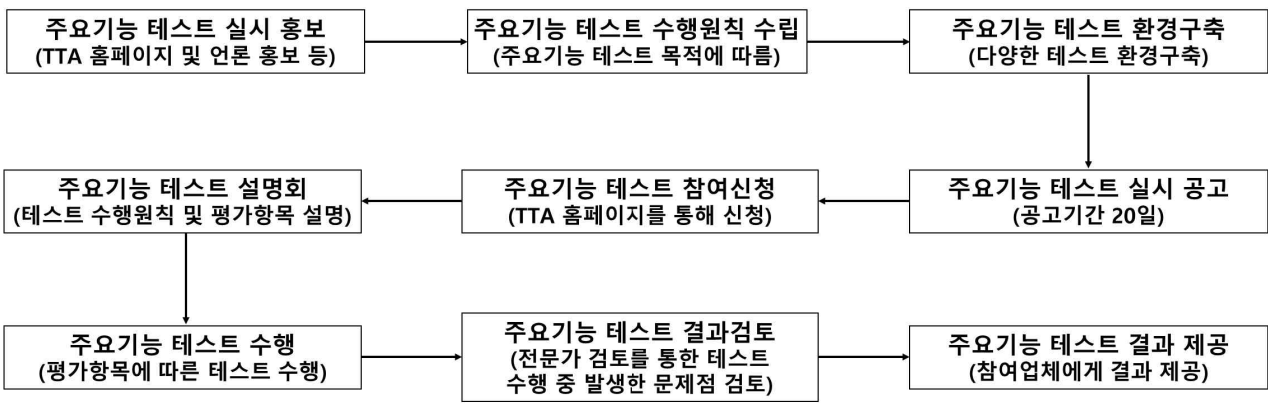


주요기능 테스트

주요기능 테스트 정의

여러 발주기관의 공통 요구사항(핵심기능)을 바탕으로 표준평가항목을 개발하여 평가하고 결과를 재활용하도록 하는 테스트

주요기능 테스트 절차



테스트 시나리오 및 데이터 설계

시나리오 설계

1) EAI 솔루션의 주요 기능 평가 항목

- ✓ 메시지 연계
- ✓ 송수신 방법
- ✓ File배치 처리
- ✓ DB배치 처리
- ✓ 암호화
- ✓ 고 가용성
- ✓ 데이터 변환
- ✓ 메시지처리확인
- ✓ 시스템 운영
- ✓ 서비스 상태 모니터링
- ✓ 연계메시지 모니터링
- ✓ 연계시스템 모니터링

➔ 위 평가 항목이 포함될 수 있도록 **TPC-DI**의 데이터 모델 선정하고, 기능 평가항목이 포함된 성능 시나리오를 개발함.

테스트 시나리오 및 데이터 설계

데이터 설계

1) 데이터 파일 결정

- 입력 데이터 타입을 **TXT, XML, Multi Record** 파일로 한정

2) 데이터 사이즈 결정

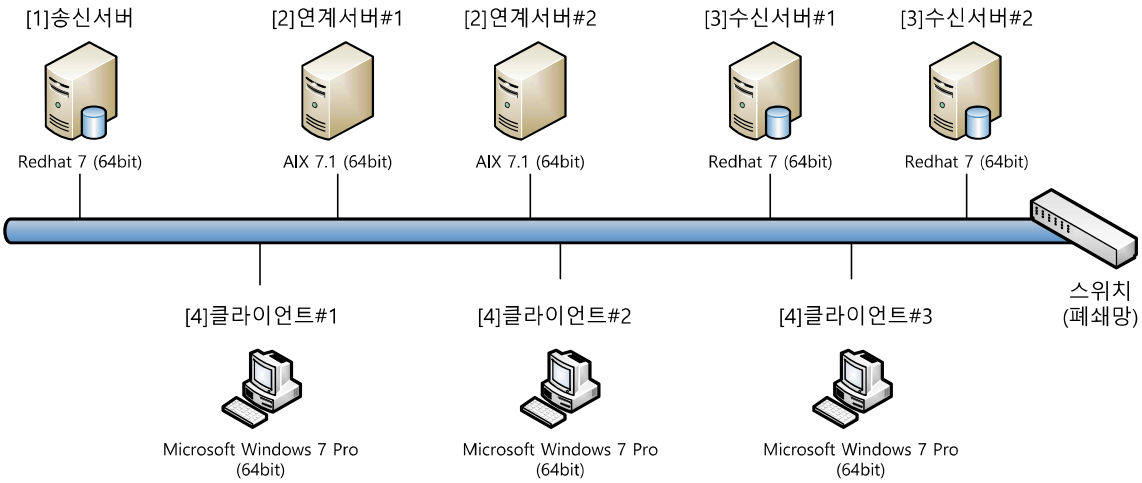
파일 명	Size in Bytes	Number of rows
Date.txt	3,420 KB	25,933 건
CustomerMgmt.xml	9,150 KB	14,572 건
FINWIREXXXX	10,800,333KB	48,914,170 건

➔ 빠른기능 수행 여부를 측정하기 위한 테스트 데이터 사이즈 선정

: TPC-DI Scale Factor를 3으로 세팅

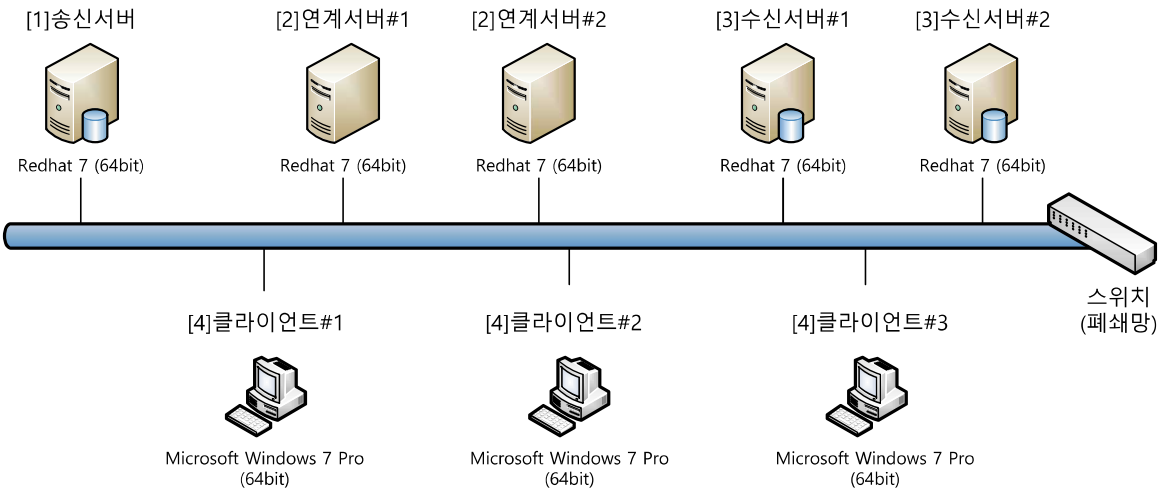
테스트 환경 구축

테스트 환경 (Unix)



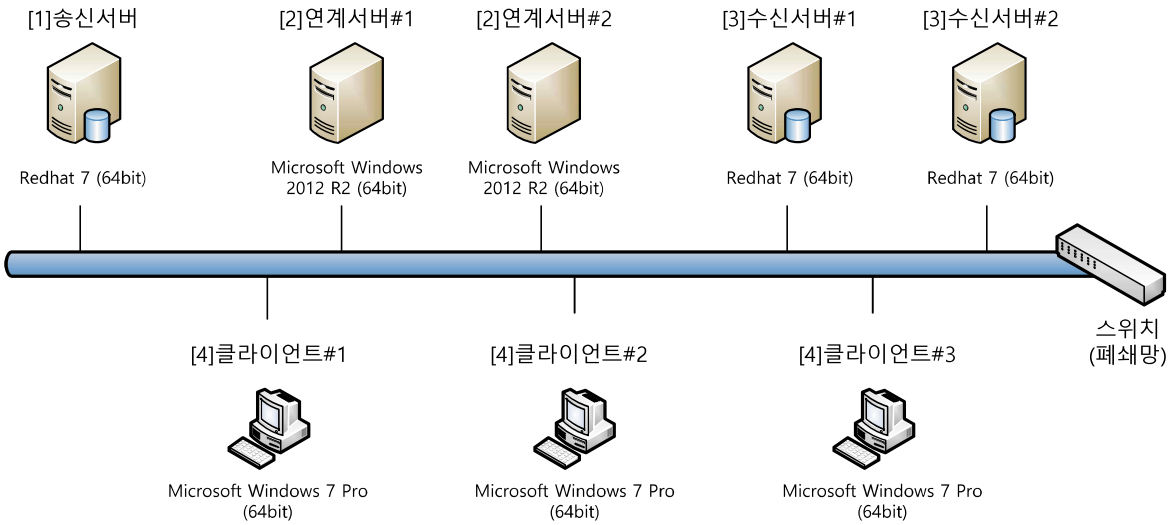
테스트 환경 구축

테스트 환경 (Linux)



테스트 환경 구축

테스트 환경 (Windows)



주요기능 테스트 수행

'데이터 변환 기능' 평가 시나리오

1. 송신서버의 **XML** 파일(CustomerMgmt.xml)에서 **ActionType** 속성 값이 "New"인 **Action/Customer/Name, Action/Customer/Address** 정보를 수신서버의 **customer** 테이블에 **Insert** 한다. → **1:1**

2. 송신서버의 **XML** 파일(CustomerMgmt.xml)에서 **ActionType** 속성 값이 "New"인 **Action/Customer/Name, Action/Customer/Address** 정보를 각각 수신서버의 **name** 테이블과 **address** 테이블에 **Insert** 한다. → **1:N**

3. **2**번 절차에서 생성된 **name** 테이블과 **address** 테이블 정보를 송신서버로 복제하고, 복제된 두 테이블 정보를 토대로 수신서버에 하나의 **XML** 파일을 생성한다. → **N:1**

4. 송신서버의 **name** 테이블과 **XML** 파일을 조인(**Join**)하고, **XML** 파일에서 **ActionType** 속성 값이 "UPDCUST"인 **Action/Customer/Address** 정보를 수신서버의 **address** 테이블에 **Update** 하고, 동시에 **address2** 테이블에 **Insert** 한다. → **M:N**

주요기능 테스트 수행(UNIX)

테스트 결과

평가항목	A업체	B업체	C업체	D업체	E업체	F업체	G업체
메시지 연계	P	P	P	P	P	P	P*
송수신 방법	P	P	P	P	P	P	P
File배치 처리	P	P	P	P	P	P	P
DB배치 처리	P	P	P	P	P	P	P
암호화	P	P	P	P	P	P	P
고 가용성	P	P	P*	P*	P	P*	P*
데이터 변환	P	P	P	P	P	P	P
메시지처리확인	P	P	P	P	P	P	P
시스템 운영	P	P	P	P	P	P	P
서비스 상태 모니터링	P	P	P	P	P	P*	P*
연계메시지 모니터링	P	P	P*	P	P	P*	P
연계시스템 모니터링	P	P	P	P	P	P*	P*

P : 합격 (모든 기능이 동작), **P*** : 부분 합격 (일부 기능 동작이 미흡)

* 본 테스트 결과는 실제 테스트 결과 중 발생된 수행 결과를 일부 분석하여
논문의 활용이 필요한 부분에 대하여만 발췌하여 정리한 결과로써 실제 테스트 결과가 아님.
또한 위 평가 항목도 일부이거나 상위 레벨로 추상화하여 정리한 결과로 실제 평가 항목은 이보다 상세함.

주요기능 테스트 수행(Linux)

테스트 결과

평가항목	A업체	B업체	C업체	D업체	E업체	F업체	G업체
메시지 연계	P	P	P	P	P	P	P*
송수신 방법	P	P	P	P	P	P	P*
File배치 처리	P	P	P	P	P	P	P
DB배치 처리	P	P	P	P	P	P	P
암호화	P	P	P	P	P	P	P
고 가용성	P	P	P*	P*	P	P*	P*
데이터 변환	P	P	P	P	P	P	P
메시지처리확인	P	P	P	P	P	P	P
시스템 운영	P	P	P	P	P	P	P
서비스 상태 모니터링	P	P	P	P*	P	P*	P*
연계메시지 모니터링	P	P	P*	P	P	P*	P
연계시스템 모니터링	P	P	P	P*	P	P*	P*

P : 합격 (모든 기능이 동작), **P*** : 부분 합격 (일부 기능 동작이 미흡)

* 본 테스트 결과는 실제 테스트 결과 중 발생된 수행 결과를 일부 분석하여
논문의 활용이 필요한 부분에 대하여만 발췌하여 정리한 결과로써 실제 테스트 결과가 아님.
또한 위 평가 항목도 일부이거나 상위 레벨로 추상화하여 정리한 결과로 실제 평가 항목은 이보다 상세함.

주요기능 테스트 수행(Windows)

테스트 결과

평가항목	A업체	B업체	C업체	D업체	E업체	F업체	G업체
메시지 연계	P	P	P	P	P	P	P*
송수신 방법	P	P	P	P	P	P	P
File배치 처리	P	P	P	P	P	P	P
DB배치 처리	P	P	P	P	P	P	P
암호화	P	P	P	P	P	P	P
고 가용성	P	P	P*	P*	P	P*	P*
데이터 변환	P	P	P	P	P	P	P
메시지처리확인	P	P	P	P	P	P	P
시스템 운영	P	P	P	P	P	P	P
서비스 상태 모니터링	P	P	P	P	P	P*	P*
연계메시지 모니터링	P	P	P*	P	P	P*	P
연계시스템 모니터링	P	P	P	P	P	P*	P*

P : 합격 (모든 기능이 동작), **P*** : 부분 합격 (일부 기능 동작이 미흡)

* 본 테스트 결과는 실제 테스트 결과 중 발생된 수행 결과를 일부 분석하여
논문의 활용이 필요한 부분에 대하여만 발췌하여 정리한 결과로써 실제 테스트 결과가 아님.
또한 위 평가 항목도 일부이거나 상위 레벨로 추상화하여 정리한 결과로 실제 평가 항목은 이보다 상세함.

Ⅳ 결론

- 요 약
- 향후 계획

요 약

EAI는 기업 응용 프로그램 통합(Enterprise Application Integration)의 약자이며, 일반적으로 국내 IT시장에서는 기업 내 응용 프로그램의 구조적 통합을 의미함

본 논문에서는 다양한 EAI솔루션 등장에 따라 해당 솔루션의 품질을 비교/평가하기 위한 표준화된 기준이 필요하여, 이에 대한 연구를 통해 TPC-DI에서 정의된 데이터 모델 중 일부를 선택하고, 7개의 제품에 적용하여 비교/평가를 한 사례를 제시 하였음.

EAI솔루션의 주요기능을 다양한 환경(AIX, LINUX, WINDOWS)에서 테스트하고 그 결과를 재활용함으로, SW분리발주를 위한 테스트 시 기능의 불필요한 반복 시험을 지양하고, 업체의 시험 수수료 및 인력의 낭비를 막을 것으로 예상됨.

***** 본 연구는 **EAI** 솔루션에 대한 테스트 수행 시 자체 연구하였던 내용을 바탕으로 작성하였으며, 한국정보통신기술협회**(TTA)** 기업의 의견이 아닌 개인의 의견으로 기술한 내용임.

향후 계획

2016년 가트너 조사 Report

Figure 1. Magic Quadrant for Data Integration Tools

Source: Gartner (August 2016)

➔ 가상화, 클라우드 서비스 데이터에 대한 통합 지원으로 솔루션이 버전 업되고 있음.
(디지털 / IoT 데이터 통합, iPaaS, 셀프 서비스 데이터 준비, 빅 데이터, 데이터 거버넌스 및 데이터 보안 등)

2016. Magic Quadrant for Data Integration Tools

https://www.gartner.com/doc/reprints?id=1-3CUJXZO&ct=160727&st=sb

➔ 개선된 EAI 솔루션의 기능에 따른 평가 기준 업데이트가 지속적으로 필요함.

TTA 한국정보통신기술협회
Telecommunications Technology Association

2018 한국 소프트웨어공학 학술대회(KCSE 2018)

무기체계 소프트웨어 IV&V 현황 및 사례 연구

국방과학연구소 공용기술센터 SW신뢰성기술실
김태현 임종일 최민관 박다운 백옥현 박삼준

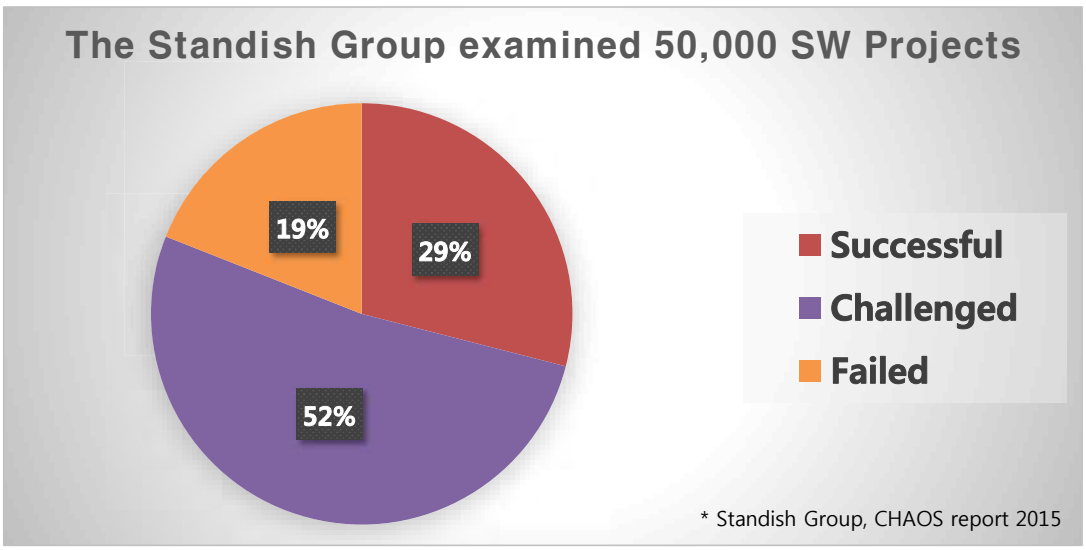
목차

1. SW IV&V 개요
2. 국외 SW IV&V 현황
3. 국내 무기체계 SW IV&V 현황
4. 국내 무기체계 SW IV&V 수행 사례
5. 결론

1. SW IV&V 개요

소프트웨어 프로젝트의 성공과 실패

- **2015년 Standish Group의 소프트웨어 프로젝트 사례 조사**
  - 과반수 이상의 SW 프로젝트는 예산 초과, 프로젝트 지연, 일부 요구사항 미 충족 등의 문제에 직면



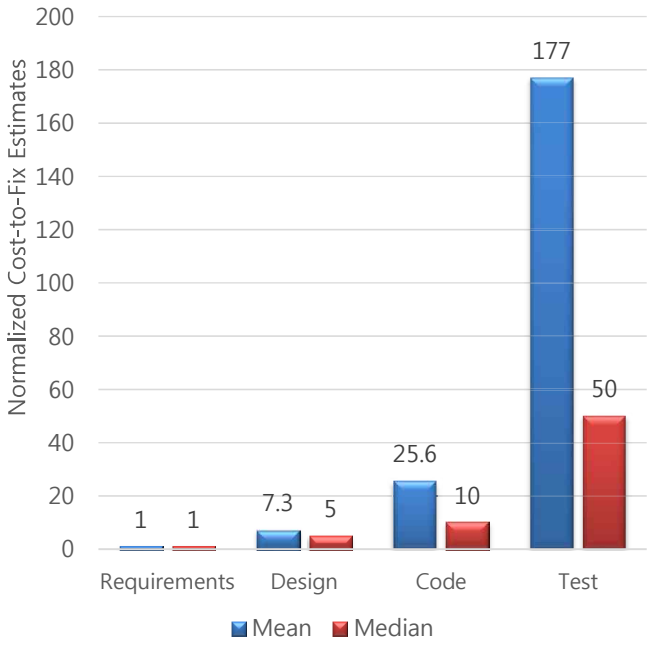
"Successful"	"Challenged"	"Failed"
In Budget On Time Meets Requirements	Over budget Delay project Missing of Capabilities	Project Fail

* Standish Group, CHAOS report 2015, https://blog.standishgroup.com/post/50

소프트웨어 결함 수정 비용

- **개발 단계 별 소프트웨어 결함 수정 비용 조사**
  - 개발 초기 단계보다 이 후 단계에서의 결함 수정 시 요구되는 비용 증가

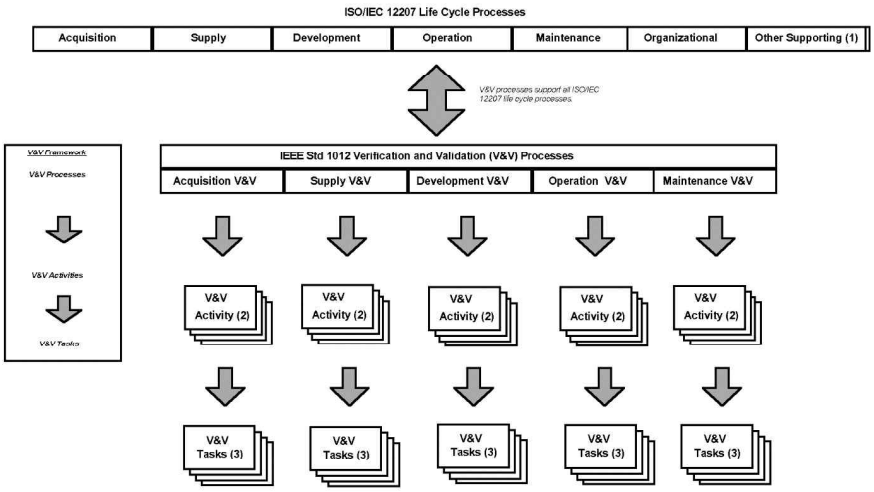
Source	Normalized Cost-to-Fix Estimates			
	Requirements	Design	Code	Test
[Boehm, 1981]	1	5	10	50
[Hoffman, 2001]	1	3	5	37
[Cigital, 2003]	1	3	7	51
[Rothman, 2000]		5	33	75
[Rothman, 2000] Case B			10	40
[Rothman, 2000] Case C			10	40
[Rothman, 2002]	1	20	45	250
[Pavlina, 2003]	1	10	100	1000
[McGibbon, 2003]		5		50
Mean	1	7.3	25.6	177
Median	1	5	10	50



* J. M. Stecklein, "Error cost escalation through the project life cycle" in 14th Annual International Symposium of INCOSE, June 2004.

Verification & Validation (V&V) 정의

- **V&V(Verification & Validation)**
  - 『"V&V comprehensively analyzes and tests software to determine that it performs its intended functions correctly, to ensure that it performs no unintended functions, and to measure its quality and reliability."』
    * Herington, D., E., Nichols, P., A. and Lipp, R., D., Software Verification Using Branch Analysis, Hewlett-Packard Journal, vol. 38, no. 6, pp. 13 - 23, June 1987.
    - Verification: Am I building the product right?
    - Validation: Am I building the right product?
  - V&V 수행 목적 및 효과
    - 초기 결함 발견
    - 잠재적 결함 발견
    - 개발 비용 감소
    - 계획된 일정 수행
    - 개발 성과물 품질 향상



*IEEE Std 1012, IEEE Standard for Software Verification and Validation

국제 표준에서의 V&V 동향 1

- **ISO 26262** (ISO 26262, Road vehicles – Functional safety)
  - 자동차 분야의 소프트웨어 기능 안전 국제 표준
  - Safety 등급에 따라 독립적인 사람/부서/기관의 검증 활동 요구

Table. Required confirmation measures, including the required level of independency

Confirmation measures	Degree of independency[a] applies to ASIL A	B	C	D	Scope
Confirmation review of the hazard analysis and risk assessment of the item (see ISO 26262-3:2011, Clauses 5 and 7, and, if applicable, ISO 26262-8:2011, Clause 5) Independence with regard to the developers of the item, project management and the authors of the work product	I3	I3	I3	I3	The scope of this review shall include the correctness of the determined ASILs and quality management (QM) ratings of the identified hazardous events for the item, and a review of the safety goals
Confirmation review of the safety plan (see 6.5.1) Independence with regard to the developers of the item, project management and the authors of the work product	—	I1	I2	I3	Applies to the highest ASIL among the safety goals of the item
Confirmation review of the item integration and testing plan (see ISO 26262-4) Independence with regard to the developers of the item, project management and the authors of the work product	I0	I1	I2	I2	Applies to the highest ASIL among the safety goals of the item
Confirmation review of the validation plan (see ISO 26262-4) Independence with regard to the developers of the item, project management and the authors of the work product	I0	I1	I2	I2	Applies to the highest ASIL among the safety goals of the item
Confirmation review of the safety analyses (see ISO 26262-9:2011, Clause 8) Independence with regard to the developers of the item, project management and the authors of the work products	I1	I1	I2	I3	Applies to the highest ASIL among the safety goals of the item
Confirmation review of the software tool criteria evaluation report and the software tool qualification report[b] (see ISO 26262-8:2011, Clause 11) Independence with regard to the persons performing the qualification of the software tool	—	I0	I1	I1	Applies to the highest ASIL of the requirements that can be violated by the use of the tool
Confirmation review of the proven in use arguments (analysis, data and credit), of the candidates (see ISO 26262-8:2011, Clause 14) Independence with regard to the author of the argument	I0	I1	I2	I3	Applies to the ASIL of the safety goal or requirement related to the considered behaviour, or function, of the candidate
Confirmation review of the completeness of the safety case (see 6.5.3) Independence with regard to the authors of the safety case	I0	I1	I2	I3	Applies to the highest ASIL among the safety goals of the item
Functional safety audit in accordance with 6.4.8 Independence with regard to the developers of the item and project management	—	I0	I2	I3	Applies to the highest ASIL among the safety goals of the item
Functional safety assessment in accordance with 6.4.9 Independence with regard to the developers of the item and project management	—	I0	I2	I3	Applies to the highest ASIL among the safety goals of the item

[a] The notations are defined as follows:
- —: no requirement and no recommendation for or against regarding this confirmation measure;
- I0: the confirmation measure should be performed; however, if the confirmation measure is performed, it shall be performed by a different person;
- I1: the confirmation measure shall be performed, by a different person;
- I2: the confirmation measure shall be performed, by a person from a different team, i.e. not reporting to the same direct superior;
- I3: the confirmation measure shall be performed, by a person from a different department or organization, i.e. independent from the department responsible for the considered work product(s) regarding management, resources and release authority.

[b] A software tool development is outside the item's safety lifecycle whereas the qualification of such a tool is an activity of the safety lifecycle.

I1 : the confirmation measure shall be performed, by a different person

I2 : the confirmation measure shall be performed, by a person from a different team

I3 : the confirmation measure shall be performed, by a person from a different department or organization

국제 표준에서의 V&V 동향 2

- **DO-178C** (DO-178C, Software Considerations in Airborne Systems and Equipment Certification)
  - 항공분야의 소프트웨어 기능 안전 국제 표준
  - Software verification 활동 시 등급에 따라 독립적 검증 수행 요구

6.2 Overview of Software Verification Process Activities

Verification independence is achieved when the verification activity is performed by a person(s) other than the developer of the item being verified. A tool may be use to achieve equivalence to the human verification activity. For independence, the person who created a set of low-level requirements-based test cases should not be the same person who developed the associated source code from those low-level requirements

Table A-3 Verification of Outputs of Software Requirements Process

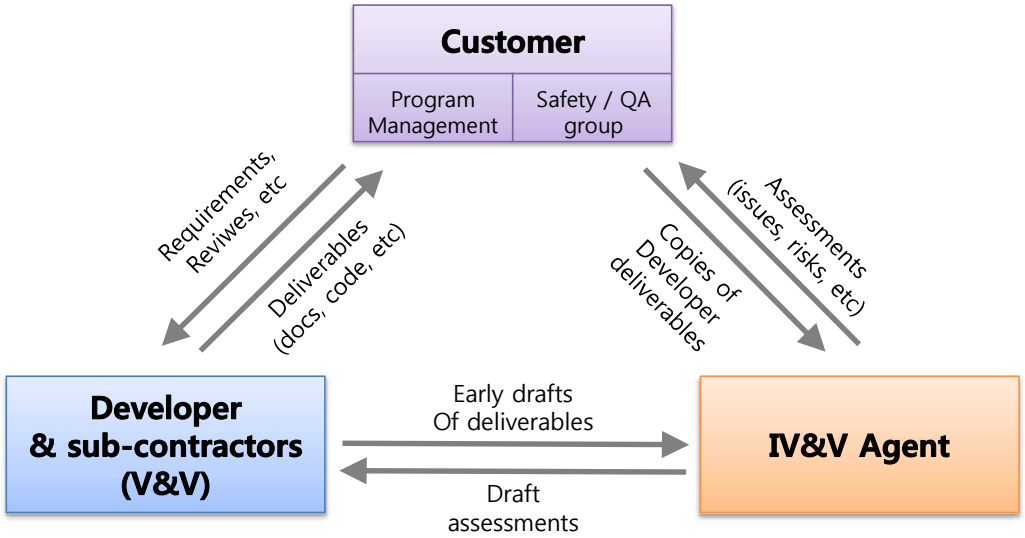
	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	High-level requirements comply with system requirements.	6.3.1.a	6.3.1	●	●	O	O	Software Verification Results	11.14	②	②	②	②
2	High-level requirements are accurate and consistent.	6.3.1.b	6.3.1	●	●	O	O	Software Verification Results	11.14	②	②	②	②
3	High-level requirements are compatible with target computer.	6.3.1.c	6.3.1	O	O			Software Verification Results	11.14	②	②		
4	High-level requirements are verifiable.	6.3.1.d	6.3.1	O	O	O		Software Verification Results	11.14	②	②	②	
5	High-level requirements conform to standards.	6.3.1.e	6.3.1	O	O	O		Software Verification Results	11.14	②	②	②	
6	High-level requirements are traceable to system requirements.	6.3.1.f	6.3.1	O	O	O	O	Software Verification Results	11.14	②	②	②	②
7	Algorithms are accurate.	6.3.1.g	6.3.1	●	●	O		Software Verification Results	11.14	②	②	②	

● : The objective should be satisfied with independence.

O : the objective should be satisfied.

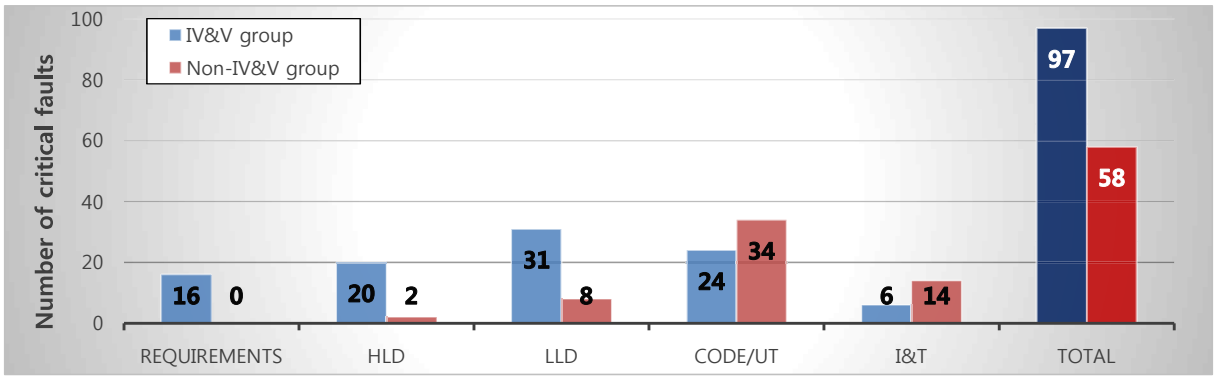
IV&V 정의

- **IV&V(Independent Verification & Validation)**
  - 『"Independent Verification and Validation (IV&V) is the activity of verifying and validating software as conducted by an organization that is technically, managerially, and financially independent from the development organization."』 Lewis, Robert O. "Independent verification and validation: A life cycle engineering process for quality software." Vol. 11. JohnWiley & Sons,1992.



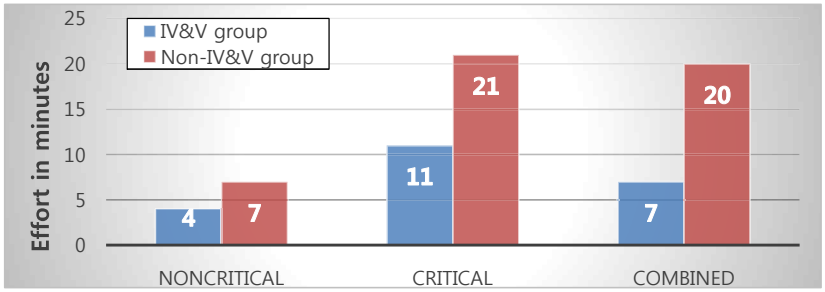
IV&V 수행 vs. IV&V 미수행 결과 비교

- **IV&V 수행 시와 IV&V 미수행 시 결함 발견 및 공수 추이 비교**
  - NASA Langley Research Center와 Virginia Tech 공동 연구
  - IV&V 수행 조직과 미수행 조직을 별도로 구성하여 연구 수행



IV&V와 Non-IV&V 적용 시 각 개발단계에서 발견된 결함 수

※ HLD(High-Level Design)
LLD(Low-Level Design)
UT(Unit Test)
I&T(Integration and Testing)



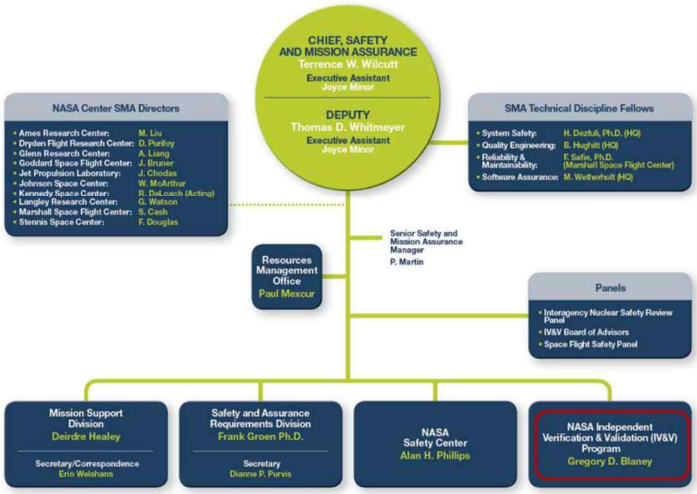
개발 프로세스 동안 결함(defaults)들을 제거하기 위해 필요한 평균 effort 값

* James D. Arthur, "Evaluating the Effectiveness of Independent Verification and Validation"

2. 국외 SW IV&V 조직 현황

국외 SW IV&V 조직 현황 1

- **NASA OSMA (Office of Safety & Mission Assurance)**
  - NASA의 모든 활동에 대한 성공 향상 및 안전 보증을 위한 조직
  - 소프트웨어의 독립적 검증을 위한 별도의 팀(IV&V Program) 존재
    - NASA의 안전/임무 보증에 대한 전략, 정책 및 표준 구축
    - 안전, 신뢰성, 유지보수성 및 품질 보증 활동 증진
    - 독립적인 안전/임무 보증 평가 수행 및 프로세스 검증
    - 중대한 안전 결정 사항에 대한 분석 및 권고사항 제시



* https://www.nasa.gov/sites/default/files/atoms/files/organizationchart_nov2015.pdf

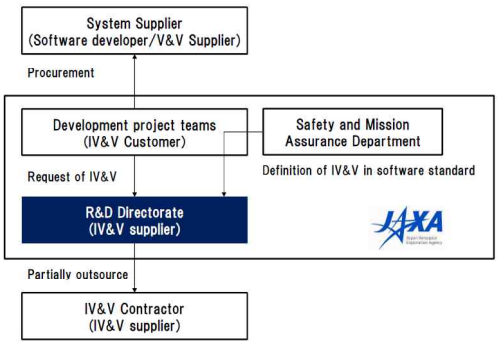
국외 SW IV&V 조직 현황 2

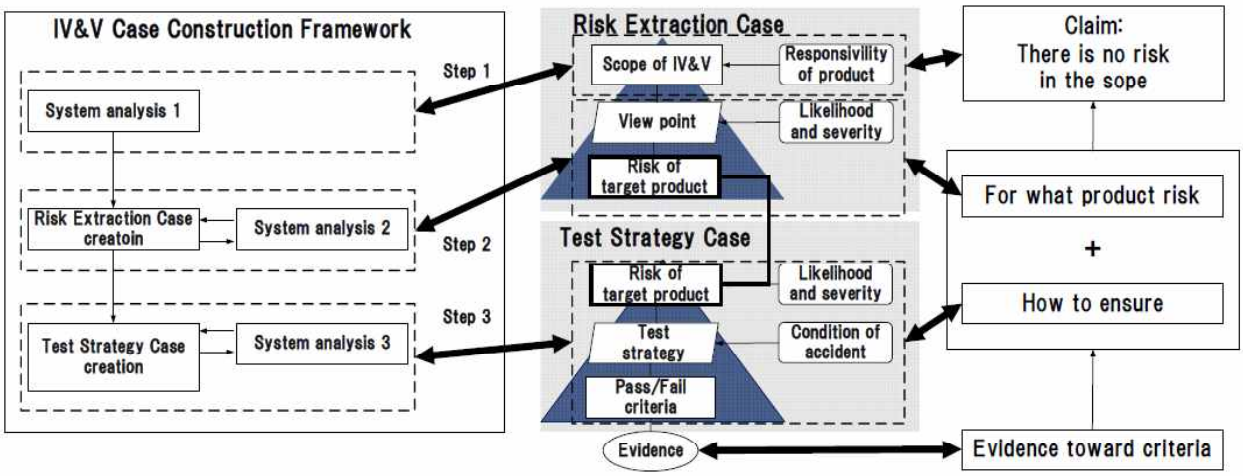
- **ARDEC QE&SA (Quality Engineering & System Assurance)**
  - 소프트웨어 생명 주기 품질 및 시스템 보증 활동 수행
  - 통합 생명주기 품질 및 시스템 보증 활동 수행 및 관리
  - 소프트웨어 품질/신뢰성 안전성 관련 필수 기술 역량 제공
  - 품질 측면에서의 소프트웨어 평가 및 연구 수행

* 출처: http://www.ardec.army.mil/organizations/esic/

국외 SW IV&V 조직 현황 3

- **JAXA JEDI(JAXA's Engineering Digital Innovation) Center IV&V팀**
  - 위성 및 미사일체계에 탑재되는 소프트웨어의 독립적 검증 및 개발 프로세스 개선을 위한 조직
  - Risk analysis 기반 독자적 IV&V Framework 개발을 통한 체계적 활동 수행

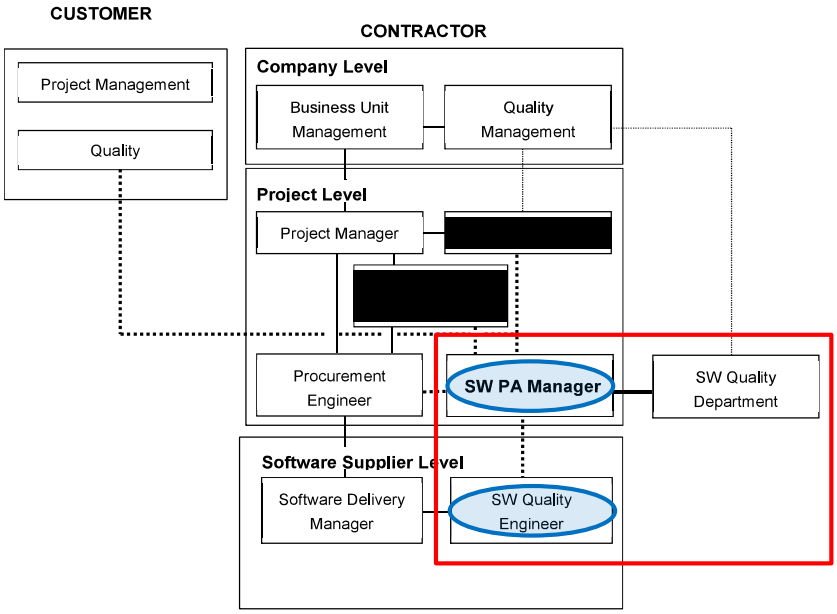




* IV&V Case: Empirical Study of Software Independent Verification and Validation based on Safety Case. Kazuki Kakimoto, Kensuke Sasaki, Hiroki Umeda and Yasushi Ueda, ISSRE 2017

국외 SW IV&V 조직 현황 4

- **Airbus의 SW Quality 조직**
  - SW 품질 보증(Quality Assurance)과 제품 보증(Product Assurance)의 역할을 독립적으로 수행하는 팀을 보유 및 운영
  - Project level에서 **Product Assurance manager**를 선정 및 운영하며 SW 개발 단계에서 품질 보증을 위한 **Quality Assurance Engineer** 운영



3. 무기체계 SW IV&V 현황

무기체계 SW IV&V 현황 (조직)

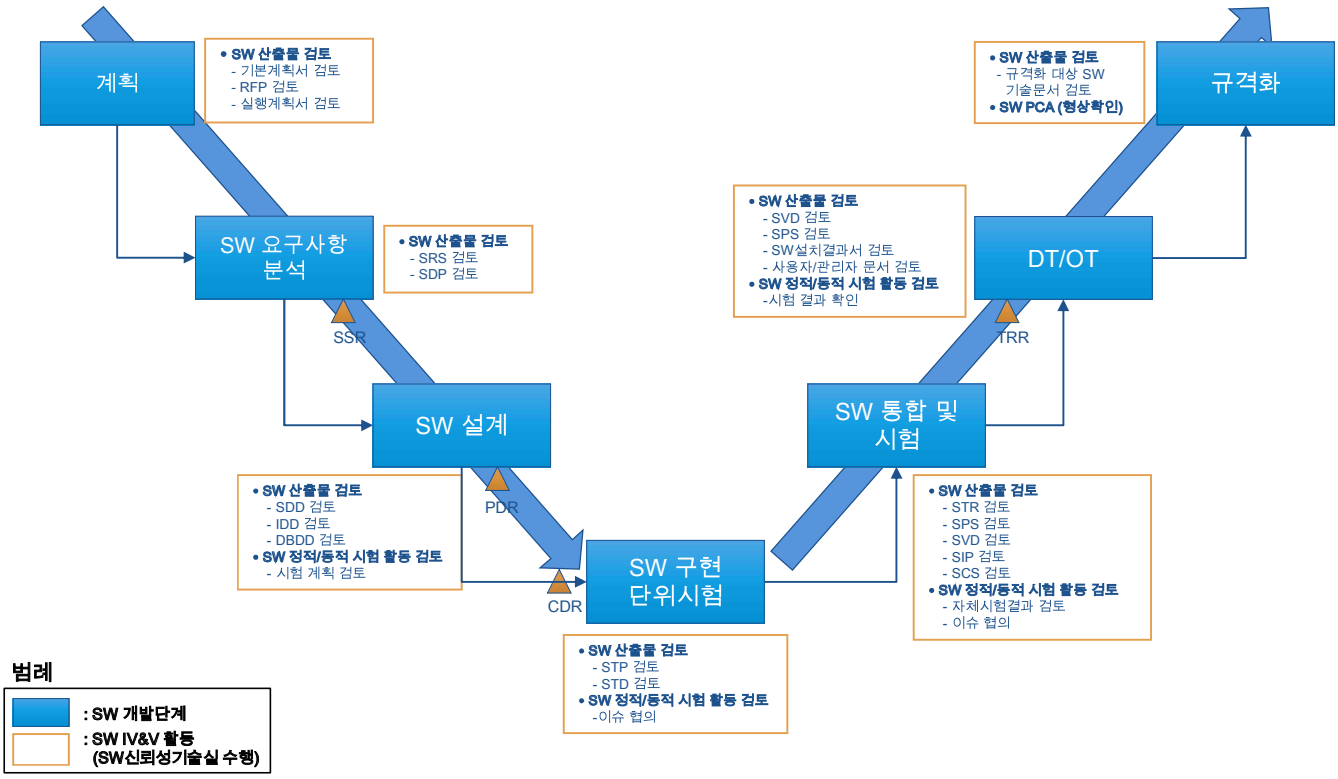
- **무기체계 SW IV&V 조직 현황 (기술지원기관)**

- **기술지원기관 업무 분장**

사업 주관 \ 획득 단계	개발 단계	양산 단계
국과연 주관	국과연 공용기술센터 SW신뢰성기술실	국방기술품질원 SW/IT 융합실
업체 주관	방산기술지원센터 SW기술팀	

무기체계 SW IV&V 현황 (프로세스 및 활동)

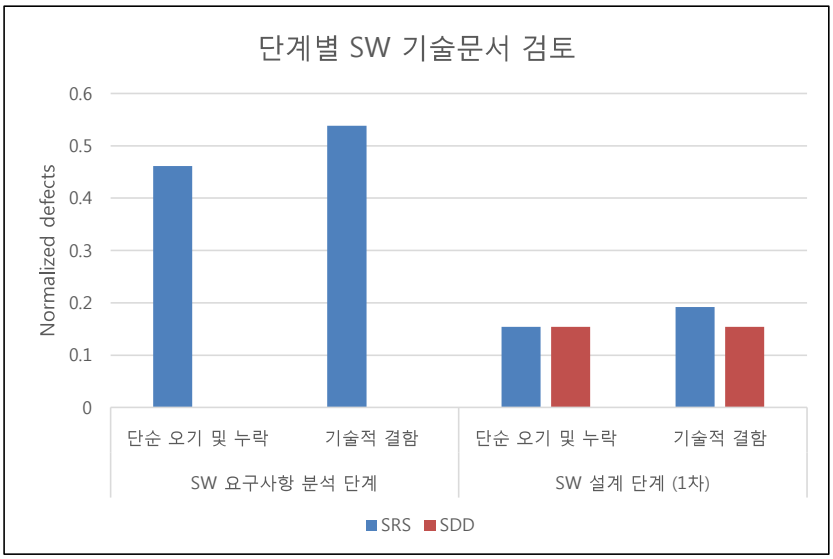
- **IV&V를 적용한 국내 무기체계 소프트웨어 개발 프로세스 및 활동**



4. 무기체계 SW IV&V 수행 사례

단계별 산출물 및 규격화 검토

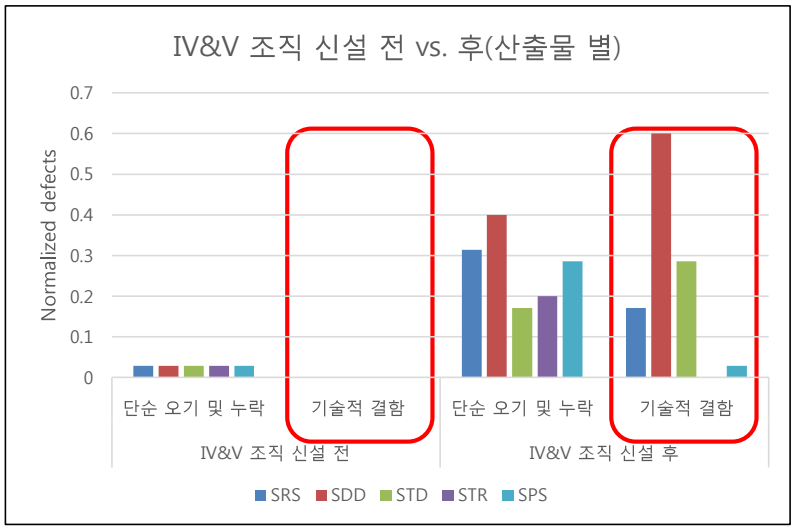
- **OO 사업 단계별 소프트웨어 기술문서 검토 수행 (진행 중)**
  - 규모 및 환경: C 언어, Small Embedded SW
  - 활동 내역
    - SW 요구사항 분석 단계 SRS 검토 수행
    - SW 설계 단계 SRS, SDD 검토 (1차) 수행



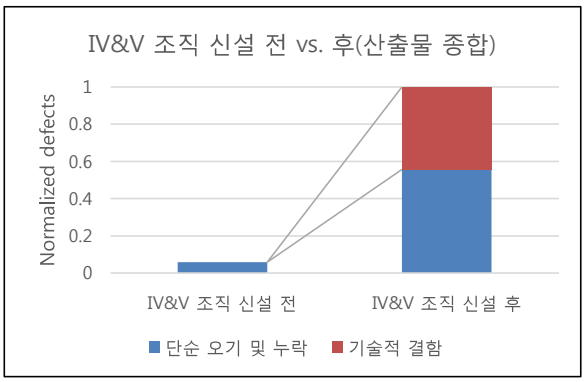
* 보안 상 최대 결함 수(문서 단위)를 기준으로 백분위 환산

단계별 산출물 및 규격화 검토

- **IV&V 조직 신설 전/후 규격화 산출물 검토 비교**
  - 비교 대상 사업: 환경 및 규모가 유사한 2개 소프트웨어 선정
  - 비교 대상 산출물 : SRS, SDD, STD, STR, SPS
  - 활동 내역
    - 규격화 단계 필수 산출물 5종 기술문서 검토



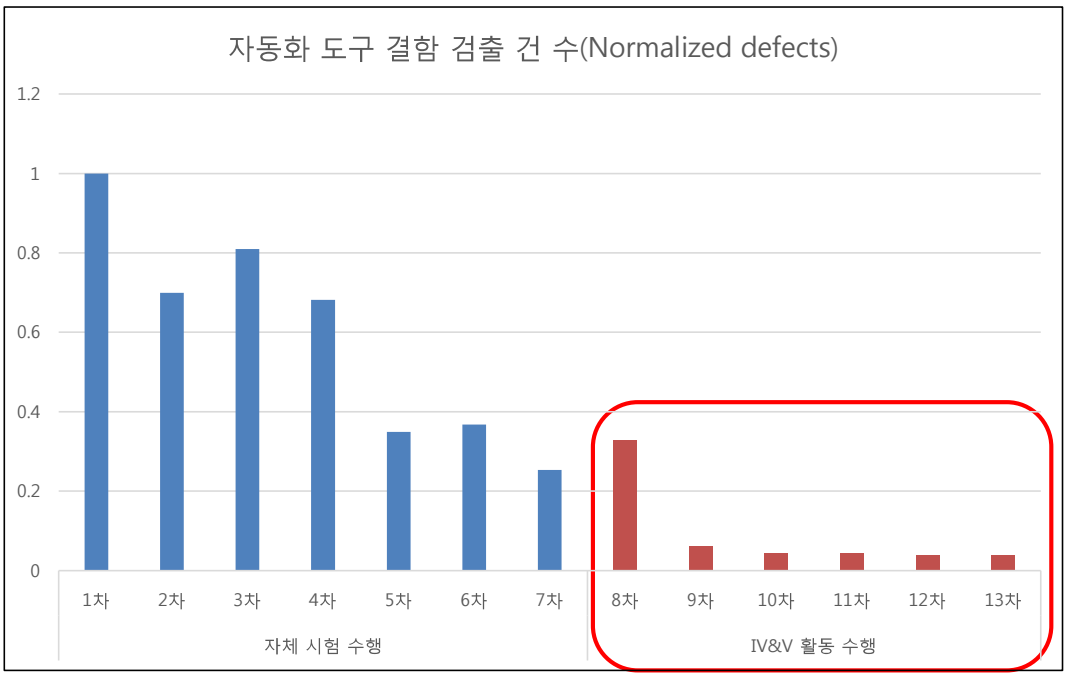
- 보안 상 최대 결함 수(문서 단위)를 기준으로 백분위 환산



- 보안 상 최대 결함 수(IV&V 조직 신설 후)를 기준으로 백분위 환산

소프트웨어 정적/동적 시험 활동 검토

- **XX 사업 소프트웨어 정적 시험 활동 검토**
  - SW 개발 및 통합/시험 기간 동안 SW 정적시험 IV&V 활동 수행
    - 도구 검출 결함 건에 대한 기술 지원 및 시험 결과 확인



- 보안 상 최대 결함 수를 기준으로 백분위 환산

- **OOO 사업 소프트웨어 형상확인**
  - **검토 중점사항**
    - ① SPS와 원시파일의 일치 여부
    - ② 목록 누락 여부(원시파일, 정적/동적 라이브러리 파일, 리소스파일, 프로젝트 파일 등)
    - ③ SPS내 실행파일 생성 및 수정 절차의 적절성
    - ④ 라이브러리의 획득 및 생성 절차 기술 여부
    - ⑤ 실행파일 생성/수정 절차에 따라 정확한 실행파일의 생성 여부
    - ⑥ SPS, SCS, 도면의 주기에 기술된 실행파일이 정확히 일치하는지 확인
  - **주요 검토결과**

①	②	③	④	⑤	⑥	기타	계*
5	**16**	**25**	**2**	**0**	**3**	**11**	**62**

※ '공통사항' 및 '모든 관련항목'에 관한 검토의견은 1건으로 Counting 함

5. 결론

결론 및 발전 방향

- **결론**
  - **소프트웨어 품질 확보를 위한 IV&V 활동의 국제적 요구 증대**
    - 선진국의 소프트웨어 관련 기관 내 체계적인 IV&V 조직 운영
    - 소프트웨어 기능 안전 국제 표준에서 소프트웨어 IV&V 요구
  - **IV&V 활동을 통한 국내 무기체계 소프트웨어 품질 향상 기여**
    - **제3자** 관점에서 V&V 활동에 참여함으로써 초기 결함 및 잠재적 결함 식별
    - 개발 전(全)순기 동안 지속적 IV&V 활동 수행
- **발전 방향**
  - **제한된 자원(인력, 시간 등)으로 효율적인 IV&V 수행을 위한 독자적 IV&V Framework 구축 필요**
    - 무기체계 군(유도, 항공, 지상, 해양 등) 특성을 고려한 독립검증 프로세스 정립
    - 정형화된 검증기법 도입 및 과학적 검증도구 활용 체계화
  - **개발자와의 협업 강화를 통한 무기체계 연구개발 전(全)단계 IV&V 내제화**
    - 개발 단계별 적시적 검토 및 피드백
    - 최적 품질확보를 위한 실질적이고 기술적인 상호간 협업

참고문헌

- Standish Group, CHAOS report 2015, https://blog.standishgroup.com/post/50
- J. M. Stecklein, “Error cost escalation through the project life cycle” in 14th Annual International Symposium of INCOSE, June 2004.
- Herington, D., E., Nichols, P., A. and Lipp, R., D., Software Verification Using Branch Analysis, Hewlett-Packard Journal, vol. 38, no. 6, pp. 13 - 23, June 1987.
- IEEE Std 1012, IEEE Standard for Software Verification and Validation
- DO-178C, Software Considerations in Airborne Systems and Equipment Certification
- ISO 26262, Road vehicles – Functional safety
- Lewis, Robert O. ” Independent verification and validation: A life cycle engineering process for quality software.” Vol. 11. JohnWiley & Sons,1992.
- James D. Arthur, “Evaluating the Effectiveness of Independent Verification and Validation”
- https://www.nasa.gov/sites/default/files/atoms/files/organizationchart_nov2015.pdf
- http://www.ardec.army.mil/organizations/esic/
- IV&V Case: Empirical Study of Software Independent Verification and Validation based on Safety Case. Kazuki Kakimoto, Kensuke Sasaki, Hiroki Umeda and Yasushi Ueda, ISSRE 2017
- S. M. Easterbrook, “The Role of Independent V&V in Upstream Software Development Processes,” Journal of Integrated Design and Process Science, vol. 2, pp. 37-46, 1998.
- B. W. Boehm, “Verifying and validating software requirements and design specifications,” IEEE Sofiware, vol. 1, no. 1, pp. 75-88, Jan. 1984.
- ARTHUR, J., “Evaluating the effectiveness of independent verification and validation” IEEE Computer, 32, 10, October 1999, pp. 79-83.

Q&A

무기체계 SW PCA 활동 및 수행 사례

Weapon System Software PCA Activities and Case Study

최민관, 임종일, 국승학, 노성규, 이태호

국방과학연구소 SW신뢰성기술실

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

순 서

I. SW PCA 배경 및 필요성

II. SW PCA 정의

III. SW PCA 수행 사례

IV. 결론 및 향후 발전 방향

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

1. SW PCA 배경 및 필요성

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

배경 및 필요성(1/6)

- **무기체계 SW 개발 프로세스**

	준비	개 발								시험평가 및 인도			
내장형 SW 개발 공정	개발준비	체계요구사항분석	체계구조설계	SW요구사항분석	SW구조설계	SW상세설계	SW구현	SW통합 및 시험	체계통합 및 시험	개발시험평가	운용시험평가	SW설치/규격화	인도
주요 검증 활동		▲체계요구사항검토회의 (SRR)	▲체계기능검토회의 (SFR)	▲SW요구사항검토회의 (SSR)	▲기본설계검토회의 (PDR)	▲상세설계검토회의 (CDR)			▲시험준비검토회의 (TRR)				
SW 기술 문서*		SDP		SRS	SDD IDD DBDD		STP STD	STR SPS SVD	FIG SCS			규격화 대상 기술문서	

* SDP: SW개발계획서, SRS: SW요구사항명세서, SDD: SW설계기술서, IDD: I/F설계기술서, DBDD: DB설계기술서, STP: SW시험계획서, STD: SW시험절차서, STR: SW시험결과보고서, FIG: 펌웨어설치절차서, SVD: SW버전기술서, SPS: SW산출물명세서, SCS: SW목록명세서

배경 및 필요성(2/6)

국방표준화란?

- 표준화(Standardization)란 군수품의 조달·관리 및 유지를 경제적·효율적으로 수행하기 위하여 표준을 설정하여, 이를 활용하는 조직적 행위와 기술적 요구사항을 결정하는 품목지정, 규격제정, 형상관리 등의 지정에 관한 제반활동

표준화 업무지침 제352호, 용어정의

국방표준화 필요성

- 군수품의 다양성 감소로 총수명주기비용 절감 및 획득기간 단축, 군수품 상호운용성·호환성·공통성 증진을 통하여 저비용·고효율의 국방자원 운영체제 구축
- 사업관리, 품질보증, 원가산정, 계약관리 군수자원관리의 기준 제공

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

배경 및 필요성(3/6)

국방규격이란?

- 군수품의 조달을 위하여 제품 및 용역의 기술적인 요구사항과 요구필요조건의 일치성 여부를 판단하기 위한 기술문서로 **국방규격서, 도면, 부품/BOM*, 품질보증요구서(QAR), S/W기술문서** 등의 기술자료로 구성

* BOM(Bill Of Material) : 특정 제품이 어떤 부품들로 구성되는가에 대한 데이터로 최종 제품과 부품관계를 계층적으로 표현

국방표준업무 실무핸드북

국방규격의 역할/중요성

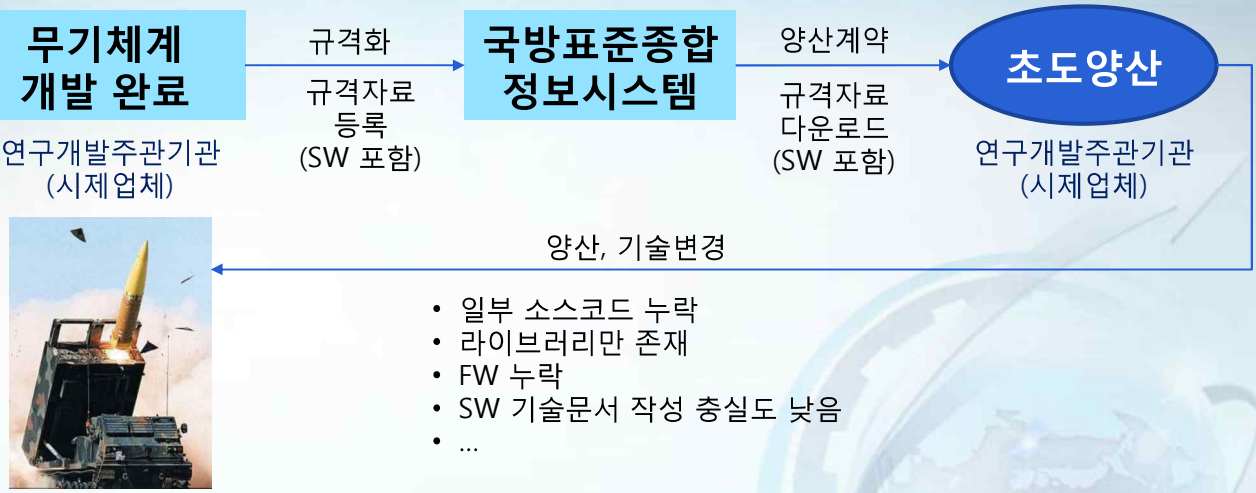
(사업관리적 측면)
- 연구개발, 설계 결과를 정리하여 제정된 유형의 **최종 연구개발 산출물**
- 연구개발 사업(체계개발) 종료의 **행정적 기준**

(활용 및 기능 측면)
- 생산 라인, 공정·치공구 설계 및 **품질보증활동 기준**
- 하자·결함 발생 시 설계/제작 결함 원인 규명의 근거 제공
- 소요 원자재·구매부품 산정 등 생산소요 자원 판단 기준
- BOM 등 물리적 **원가의 산정 기준 및 사업관리를 위한 기준**
- 교범 작성, 목록화 및 장비운영을 위한 수리부속 구매 시 계약 기준

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

배경 및 필요성(4/6)

- **SW 규격화 자료 충실도 미흡**



- 일부 소스코드 누락
- 라이브러리만 존재
- FW 누락
- SW 기술문서 작성 충실도 낮음
- …

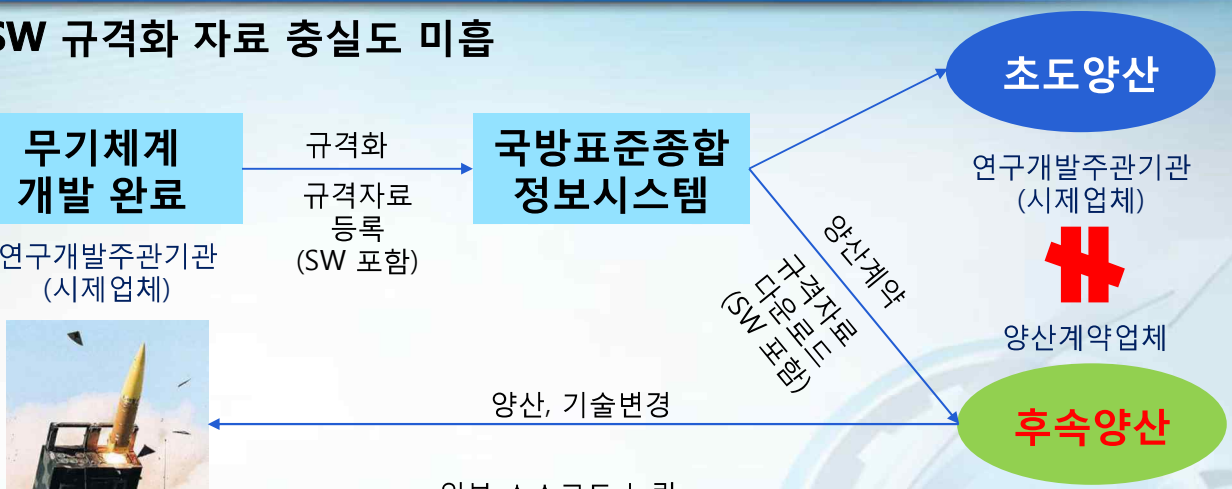
(만약)
- 개발자(담당) 퇴직
- 협력업체 폐업
- 업체 자체 버전(형상) 관리체계 미작동
- …

➔ **과다한 기술변경 소요 발생**
➔ **수정 혹은 빌드를 위한 자원 소요 증가**

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

배경 및 필요성(5/6)

- **SW 규격화 자료 충실도 미흡**



- 일부 소스코드 누락
- 라이브러리만 존재
- FW 누락
- SW 기술문서 작성 충실도 낮음
- …

(만약 개발업체와 양산업체가 다를 경우)
- …

➔ **수정 혹은 빌드 불가**

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

배경 및 필요성(6/6)

- SW 규격화 자료 충실도 미흡
- SW PCA 업무 제도 인식 미흡
- SW PCA 수행 필요성 공유 필요
- 규격화 일정 계획 수립시 SW PCA 수행 일정 계획 미반영
- SW PCA 수행 세부 절차 및 가이드라인 미비

SW PCA 수행 세부 절차 안내 및 규격화 일정 계획 반영으로 규격화 품질 향상

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

2. SW PCA 정의

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

형상확인

- 형상식별서 : 모든 형상관리 품목에 대하여 품목의 기능적, 물리적 기준 특성을 식별하기 위한 규격서, 도면 등의 기술문서이며, 형상통제 및 현황유지를 위한 기본자료로 활용 (제58조 1항)
- 기능적 형상확인(FCA, Functional Configuration Audit) : 형상식별서에서 요구되는 성능 발휘 여부를 확인하는 활동 (제81조)
- 물리적 형상확인(PCA, Physical Configuration Audit) : 국방규격의 제품기준에 맞게 완전하고 정확하게 제조되었는지 확인 점검하는 활동 (제81조)

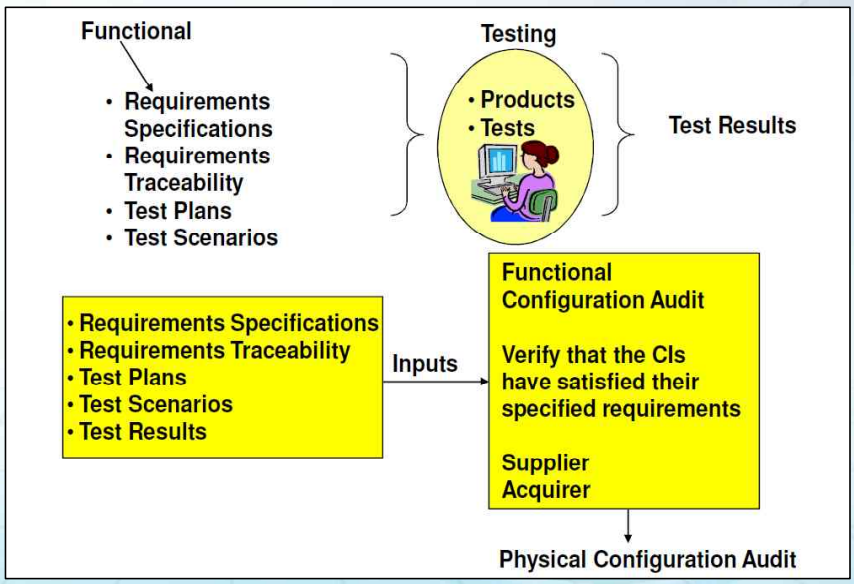
표준화 업무지침

- **형상확인은 제품이 형상식별서와 합치되는지 여부를 점검하는 활동**
- **형상관리 책임기관이 기능적 형상확인(FCA)과 물리적 형상확인(PCA)으로 구분하여 수행하고 필요시 관련기관의 지원을 받아 수행**
- **기능적 형상확인(FCA)은 물리적 형상확인(PCA)에 우선하여 실시**

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

기능적 형상확인(FCA)

- **FCA**(Functional Configuration Audit)
  - A formal examination of test results of the as-built functional configuration of Configuration Item(CI)s, prior to acceptance, to verify that the CIs have satisfied their specified requirements
  - 수행 : 공급자
  - 참석 : 수요자 및 공급자
    - Management
    - System Engineering
    - HW/SW Engineering
    - Test Engineering
    - QA and CM
    - Contracts

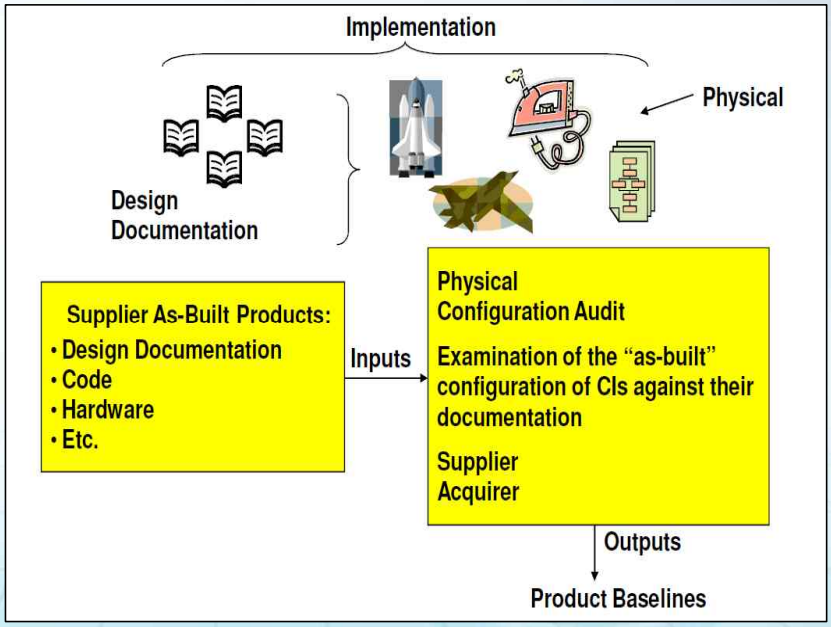


※ Ref : MITRE, 2010, by AI Florence

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

물리적 형상확인(PCA)

- **PCA**(Physical Configuration Audit)
  - A formal examination of the as-built physical configuration of CI products against their design documentation
  - **establishes the Product Baseline**
  - 수행 : 공급자
  - 참석 : 수요자 및 공급자
    - Management
    - System Engineering
    - HW/SW Engineering
    - Test Engineering
    - QA and CM
    - Contracts



※ Ref : MITRE, 2010, by AI Florence

관련 법령/지침(1/3)

- **국방전력발전업무훈령 (국방부 훈령 제2040호)**
  - **제186조(국방규격 제정범위)**

    국방규격은 연구개발을 위한 시제품, 소프트웨어, 기술협력 생산품목, 국외구매 품목, 완성장비 수리부속품 등으로 구분하여 그 제정범위를 정할 수 있으며 세부적인 사항은 방사청 규정(방위사업관리규정 및 표준화업무지침)에 따른다.
  - **제197조(형상확인)**

    ① 형상확인은 제품이 형상식별서와 합치되는지 여부를 점검하는 활동으로 형상관리 책임기관이 기능적 형상확인과 물리적 형상확인으로 구분하여 수행하고, 필요시 관련기관의 지원을 받아 수행한다.

- **방위사업관리규정 (방사청 훈령 제379호)**
  - **제123조(체계개발 수행)**

    ⑦ 통합사업관리팀장은 품질보증업무, 규격화 및 목록화 업무를 효율적으로 수행하기 위하여 체계개발 단계부터 기품원 및 계획지원부에 품질보증활동 및 규격화·목록화 활동 지원을 요청하여야 한다. 이 경우 기품원 및 계획지원부는 지원하여야 한다.

    ⑨ 사업관리본부장은 소요군·기품원·방산기술센터 및 관련기관과 협조하여 체계개발단계의 각종 기술적 문제에 대한 검토를 수행하며, 국방규격 제정안에 대한 검토를 체계개발 초기부터 수행한다.

관련 법령/지침(2/3)

- **표준화 업무지침 (방사청 예규 제352호)**
  - **제14조(국방규격 제정 범위)**
    - ① 연구개발을 위하여 시제품을 제적 중에 있는 군수품은 시제품 및 정비단계별 수리부속품의 기술자료를 포함하여 국방규격을 제정하여야 한다.
    - ② 소프트웨어의 국방규격화 대상은 소프트웨어 기술문서와 소스코드를 말하며, 모든 기술자료에 대한 규격화가 비경제적이라고 판단될 때는 국방규격 작성관리기관의 검토를 거쳐 획득기획국장과 협의하여 규격화 범위를 조정할 수 있다.

- **무기체계 SW 개발지원에 관한 규정 (방사청 훈령 377호)**
  - **제10조(무기체계 소프트웨어 규격화)**
    - ① 센터장은 통합사업관리팀장으로부터 기술지원을 요청받은 경우, 규격화 단계에서 소프트웨어의 최신 컴퓨터 파일(소스코드, 라이브러리, 실행파일, 데이터베이스 등) 및 관련 기술자료가 규격화되도록 기술지원을 실시하여야 한다.
    - ④ 기품원장은 방위사업관리규정 제123조 및 제602조에 따라 소프트웨어 규격화 자료에 대한 기술검토를 수행한다.

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

관련 법령/지침(3/3)

- **무기체계 SW 개발 및 관리 매뉴얼 (방사청 매뉴얼 제2017-8호)**

[시험평가, 규격화 및 인도]

가. 사업관리부서

9) 개발된 소프트웨어의 규격화 및 목록화가 필요한 경우 관련기관과 협조하여 수행

10) 기술지원기관에게 소프트웨어 규격화 자료 검토 의뢰 및 필요시 소프트웨어 규격화준비검토회의 의뢰

나. 연구개발주관기관

7) 개발된 소프트웨어의 규격화 및 목록화 필요시 관련 자료를「방위사업관리규정」,「국방규격의 서식 및 작성에 관한 지침」,「표준화 업무 지침」등 관련 규정·지침에 따라 작성하여 사업관리부서 및 국방표준종합정보시스템에 제출

마. 기술지원기관

4) 소프트웨어 규격화 대상 기술문서가「국방규격의 서식 및 작성에 관한 지침」및 본 매뉴얼의 산출물 작성기준을 준수하여 충실히 작성 되었는지 검토

5) **모든 규격화대상 소프트웨어 산출물의 최종버전이 규격화되도록 검토**

[형상관리]

라. 형상확인

나) 연구개발주관기관은 형상확인 계획에 의거 자체 형상확인을 수행 후 결과보고서를 종합하여 사업관리부서에 형상확인을 요청한다.

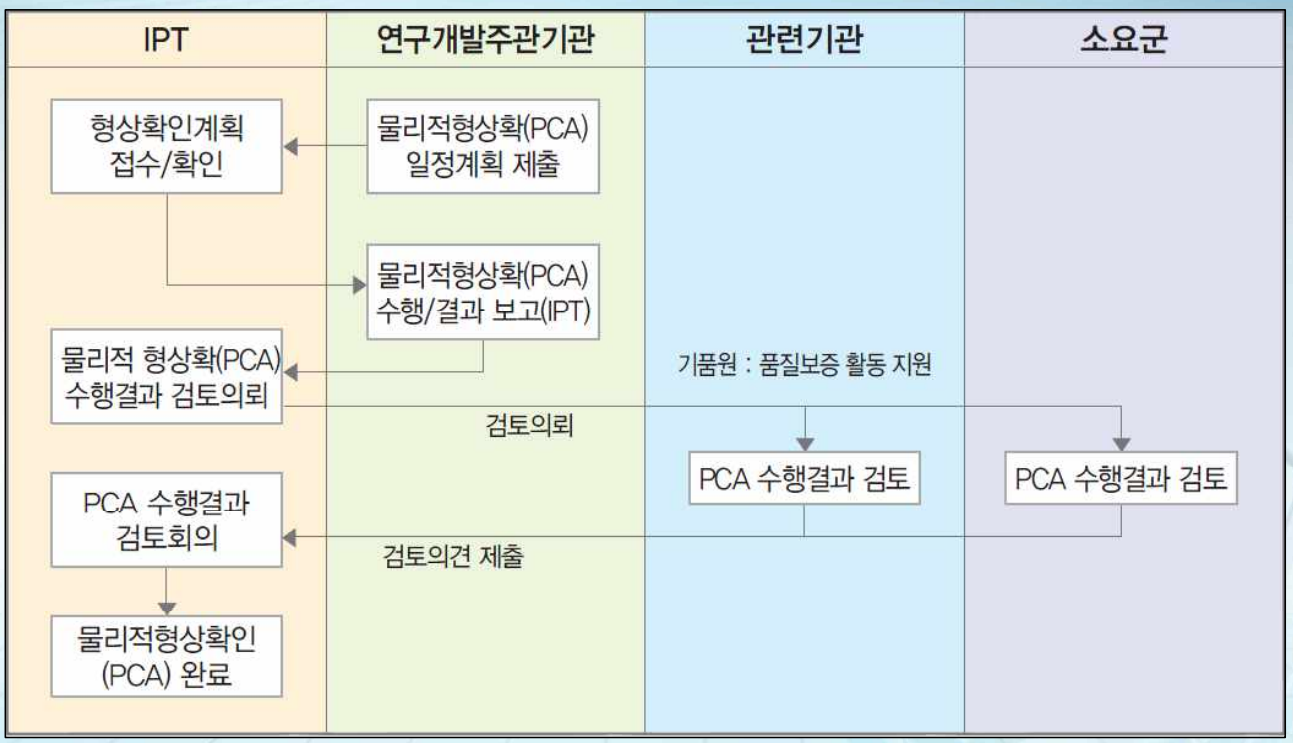
라) 사업관리부서는 연구개발주관기관의 형상확인 수행 결과 보고서를 검토하여 형상 확인을 수행하고, 필요시 관련기관의 지원을 받아 수행한다.

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

PCA 수행시기

※ Ref : SE기반 기술검토회의 가이드북, 2017, 방위사업청

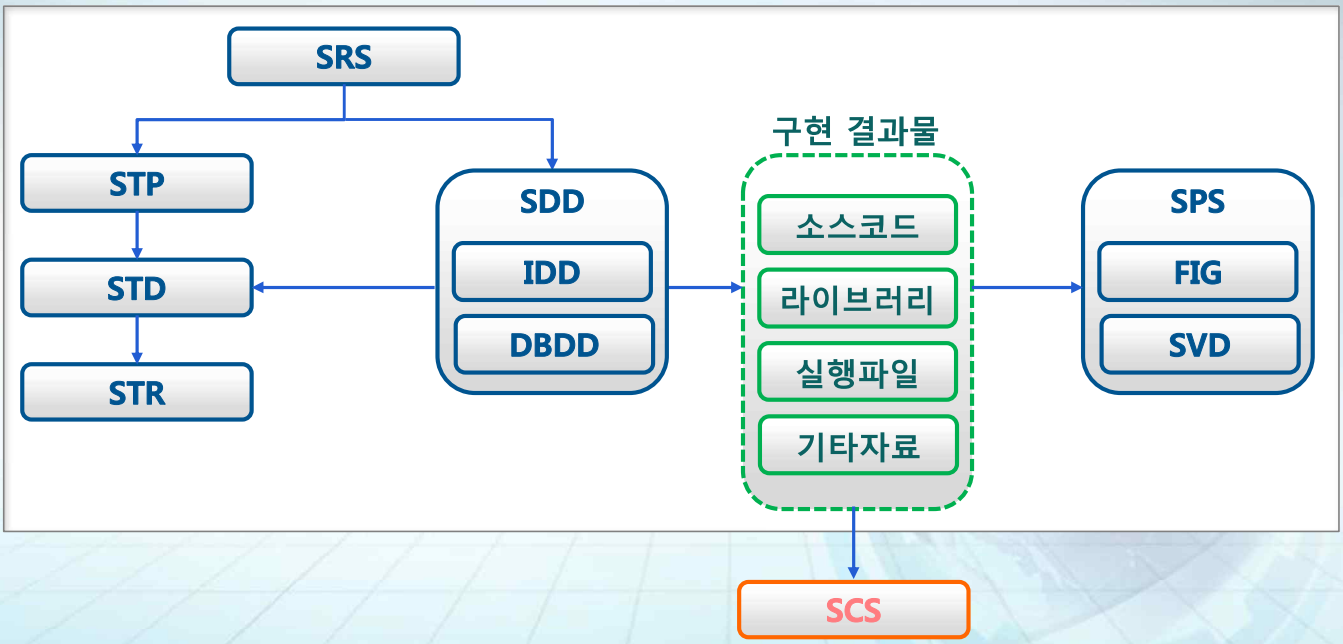
PCA 업무절차



※ Ref : SE기반 기술검토회의 가이드북, 2017, 방위사업청

SW 기술문서 검토(규격화 자료 검토 1단계)

- **주요 SW 기술문서 검토**
  - 설계 내용 및 문서 간 추적성, 일관성에 대해 검토



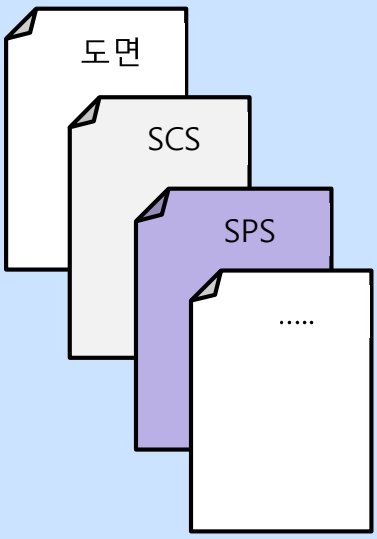
* SRS: SW요구사항명세서, SDD: SW설계기술서, IDD: I/F설계기술서, DBDD: DB설계기술서, STP: SW시험계획서, STD: SW시험절차서, STR: SW시험결과보고서, SPS: SW산출물명세서, FIG: 펌웨어설치절차서, SVD: SW버전기술서, SCS: SW목록명세서

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

SW PCA(규격화 자료 검토 2단계)

- **소프트웨어 물리적 형상확인**
  - 국방규격의 제품기준에 맞게 완전하고 정확하게 소프트웨어가 제조되었는지 확인 점검하는 활동

SW 형상식별서



SW 제품

Software
- 소스코드
- 실행파일
- 라이브러리
- ...

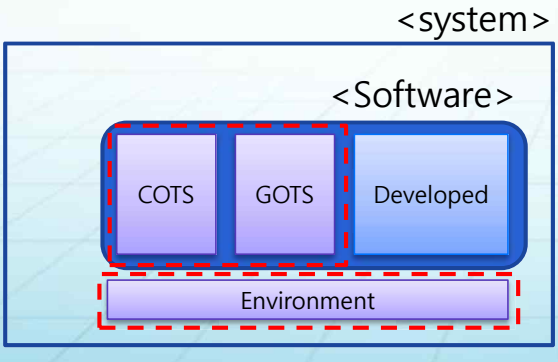
국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

3. SW PCA 수행 사례

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

SW PCA 검토 중점사항

- ☐ SPS와 원시파일의 일치 여부
- ☐ 목록 누락 여부
  (원시 파일, 정적/동적 라이브러리 파일, 리소스 파일, 프로젝트 파일 등)
- ☐ SPS내 실행파일 생성 및 수정 절차의 적절성
- ☐ 라이브러리의 획득 및 생성 절차 기술 여부
- ☐ 실행파일 생성/수정 절차에 따라 정확한 실행파일의 생성 여부
- ☐ SPS, SCS, 도면의 주기에 기술된 실행파일이 정확히 일치하는지 확인



국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

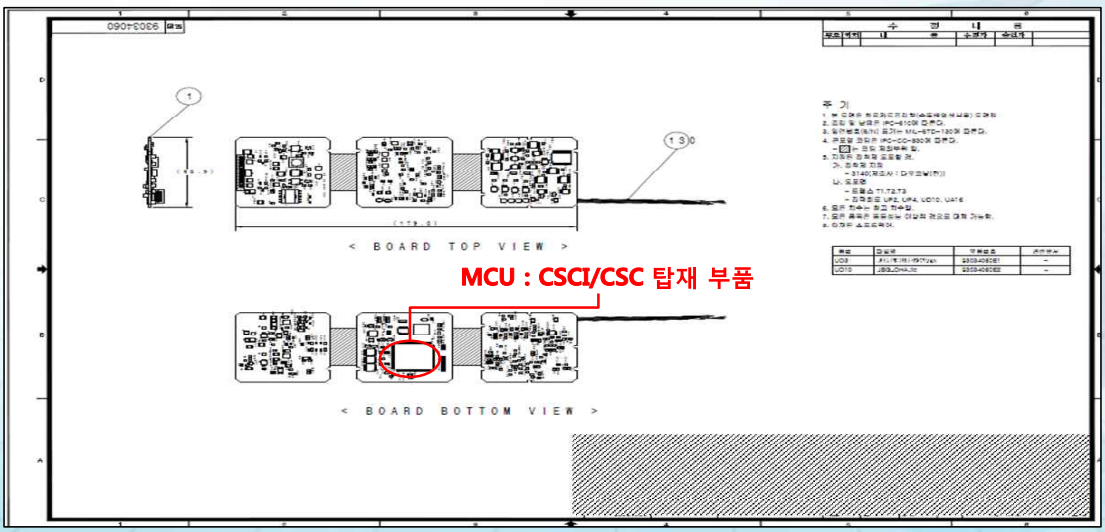
SW PCA 세부 업무 절차

준비 단계	• 기술문서 준비(SCS, SPS, 도면) • SW 개발환경 및 개발 완료 SW • 개발 SW 운용 실장비 • SW의 정상 기능수행 확인용 장비(필요시) • SW PCA 지원 유틸리티 ✓ Check-sum 추출 프로그램 ✓ 한글/워드, 엑셀 등
현장 확인	• 실장비 및 도면 확인 • 문서(SCS, SPS) 검토 • SW 파일 목록(개발 환경 및 탐색기) 및 Check-sum 일치 확인 • SPS에 기술된 실행파일 생성/수정 및 탑재 절차 수행 확인 • SW의 정상 기능수행 확인
결과보완 및 규격화 추진	• 수정/보완 사항 조치 및 피드백 • 규격화를 위한 후속 업무 추진

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

SW PCA 세부 업무(1/5)

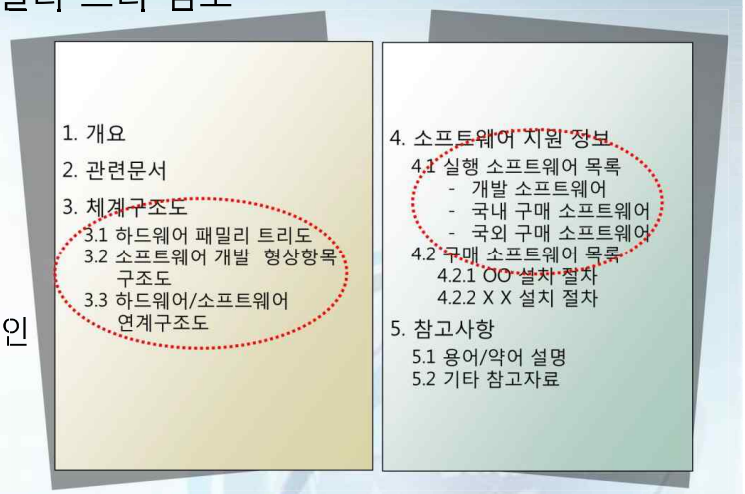
- **실장비, 도면 확인**
  - 실장비, 도면 확인 후 누락 SW 존재 여부 확인
    - 실장비, 도면 확인 후 SW 탑재 부품 확인
    - 탑재 부품별 실제 탑재되는 SW 확인
      - ✓ SPS, SCS 문서 내 탑재 장비 목록
    - 누락 SW 식별 시
      - ✓ SW 기술자료 보완 및 정적/동적 시험 재 실시 필요



국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

SW PCA 세부 업무(2/5)

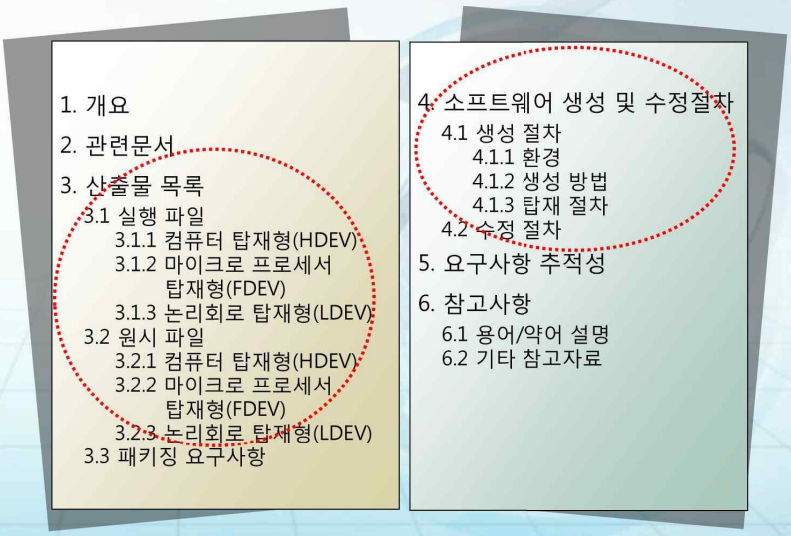
- **SCS 문서 검토**
  - SW PCA에 해당되는 하드웨어 패밀리 트리 검토
  - SW 형상항목 확인
    - 물리적 산출물 목록
    - 납품 목록과 일치 여부 확인
  - HW/SW 연계구조도 확인
    - 하위 HW와 SW 탑재 간의 관계 확인
  - 실행 SW 파일 목록 확인
    - 실행파일이란 HW의 동작을 제어하기 위해 탑재되는 프로그램을 의미
  - 구매 SW 목록 확인
    ※ 구매 SW 목록에는 Compiler, IDE(Integrated Development Environment)는 불포함



국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

SW PCA 세부 업무(3/5)

- **SPS 문서 검토**
  - SPS 문서의 실행파일 목록과 일치 여부 확인
  - 실행파일을 생성하기 위한 원시파일 확인
    - 실제 프로젝트 내의 소스코드 목록과 일치 여부
    - 솔루션/프로젝트 파일(프로젝트의 속성 및 경로 등을 정의한 파일) 포함
  - 환경 작성 확인
    - 소스코드 개발 환경
    - 컴파일/빌드 환경
    - 플래시 메모리 퓨징 환경
    - 컴퓨터 하드웨어 자원
      - Build를 위한 컴퓨터
      - 탑재를 위한 연결선 등



국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

SW PCA 세부 업무(4/5)

- **SW 파일 목록(개발 환경 및 탐색기) 및 Check-sum 일치 확인**
  - SPS와 원시파일의 Check-sum, Size, 생성일 등 확인을 통해 최종 파일이 기술되었는지 확인
    - Check-sum 추출 프로그램, 추출 알고리즘 및 버전 명시

- **SPS에 기술된 실행파일 생성/수정 및 탑재 절차 수행 확인**
  - 실행파일 생성 및 수정 절차의 적절성 확인
  - SPS에 기술된 실행파일 생성, 수정 절차가 상세하게 기술 및 재현될 수 있는지 확인

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

SW PCA 세부 업무(5/5)

- **SW의 정상 기능수행 확인**
  - 실행파일 생성/수정 절차에 따라 정확한 실행파일의 생성 여부 확인
  - 생성된 실행파일을 실 장비에 탑재해 동작 여부 확인
    - 실장비, 점검장비 시뮬레이터, HILS(Hardware-in-the-Loop Simulation) 등 활용

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

SW PCA 결과 보완 및 후속조치

- **수정/보완 사항 조치 및 피드백**

SW PCA 활동 시 제시된 검토사항 수정/보완 → 수정/보완 결과 검토 (SW PCA 참여자) → 베이스라인 설정

- **규격화를 위한 후속 업무 추진**

규격실무토의 → 표준종합정보시스템 (KDSIS) 등록

※ KDSIS : 군수품 획득, 운영유지 전(全) 단계의 표준화 업무를 지원하는 규격 및 목록관리, 부품단종 관리, BOM(Bill Of Material)관리로 구성된 표준화 정보시스템

SW PCA 수행사례(1/4)

- **대상사업 : OOO 체계개발 사업**
- **검토 중점사항**
  - ① SPS와 원시파일의 일치 여부
  - ② 목록 누락 여부(원시파일, 정적/동적 라이브러리 파일, 리소스파일, 프로젝트 파일 등)
  - ③ SPS내 실행파일 생성 및 수정 절차의 적절성
  - ④ 라이브러리의 획득 및 생성 절차 기술 여부
  - ⑤ 실행파일 생성/수정 절차에 따라 정확한 실행파일의 생성 여부
  - ⑥ SPS, SCS, 도면의 주기에 기술된 실행파일이 정확히 일치하는지 확인

- **중점사항 검토결과 : OOO체계 규격화 자료 검토 결과**

①	②	③	④	⑤	⑥	기타	계*
5	16	25	2	0	3	11	62

※ '공통사항' 및 '모든 관련항목'에 관한 검토의견은 1건으로 Counting 함

SW PCA 수행사례(2/4)

검토 1. SPS와 원시파일 일치 여부

- 원시파일의 Checksum, Size, 생성일 등 확인을 통해 최종 파일이 기술되었는지 확인

검토 2. 목록 누락 여부

- 원시파일, 실행파일 생성 절차 확인 과정에서 필요한 파일 및 라이브러리 등이 누락되었는지 확인
  - 원시파일 누락 사례 : project 파일, makefile, configuration 파일, image 파일, script 파일 등
    ※ **일반적으로 개발자들은 원시파일을 source code에 한정하여 작성**
  - 실행파일 누락 사례 : library 파일, script 파일, image 파일, ini 파일 등
    ※ **일반적으로 개발자들은 실행파일을 exe 파일에 한정하여 작성**
  - 기타파일 누락 사례 : SW 동작에 필요한 별도 프로그램 등
    ※ **지도변환을 위해 별도로 개발한 유틸리티 프로그램 등**

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

SW PCA 수행사례(3/4)

검토 3. 실행파일 생성 및 수정 절차의 적절성

- SPS에 기술된 실행파일 생성, 수정 절차가 상세하게 기술/재현될 수 있는지 확인
  - 실행파일 생성 절차 중 사전조건 누락 : 생성을 위해 필요한 설정, resource 등의 확인 절차가 누락
  - 실행파일 생성 후 확인 사항 누락 : 실행파일 생성 절차의 결과로 생성되는 파일을 확인 하는 절차가 누락
  - 탑재절차와의 연계 미흡 : 생성절차의 결과 실행파일과 탑재절차에 명시된 내용이 다른 경우
    ※ **실행파일 생성 절차에는 exe 파일을 생성, 탑재절차에는 InstallShield, NSIS를 통해 패키징된 설치파일을 이용하는 경우 등**

검토 4. 라이브러리 획득 및 생성 절차 기술여부

- 개발 SW 내 라이브러리의 생성 절차, 상용 라이브러리 등의 획득 절차가 정확히 기술되었는지 확인
  - 개발 SW의 라이브러리 누락 : 타 부체계에서 개발한 라이브러리를 이용할 경우 획득 및 확인 절차가 누락됨
  - 상용 라이브러리 등 획득/설치절차 누락 : DDS(Data Distribution Service), Boost 등 실행파일 생성을 위해 사용되는 라이브러리의 획득, 설치 절차가 누락됨.

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

SW PCA 수행사례(4/4)

검토 5. 실행파일 생성/수정 절차에 따라 정확한 실행파일의 생성 여부

- 생성된 실행파일을 실 장비에 탑재해 동작 여부를 확인 및 일부 기능 시험 확인

검토 6. SPS, SCS, 도면의 주기에 기술된 실행파일이 정확히 일치하는지 확인

- SCS, 도면에 생성된 실행파일의 정보가 정확히 기술되었는지 확인
- (사례)
  - 도면 주기란에 실행파일 목록 누락 : exe 파일만 명시하고, 기타 실행에 필요한 파일(library, image, resource)이 누락

주기 x. 탑재된 소프트웨어

품번	파일명	SW 부품번호	관련문서
A	B	C	D

✓ A항 : 파일이 탑재되는 부품의 부품목록란의 품번 기록
✓ B항 : 소프트웨어의 파일명(소프트웨어 산출물 명세서상의 파일명) 기록 (예: control.bin, bitetest.hex, ..)
✓ C항 : 소프트웨어의 부품번호 기록(소프트웨어 산출물 명세서상의 소프트웨어 부품번호)
(예: 40002488E001, 40002488E002)
✓ D항 : 소프트웨어 산출물 명세서의 번호 및 관련 항목번호 등 특기사항 기록

  - SPS, SCS의 부품번호 불일치 : '국방규격의 서식 및 작성에 관한 지침'에 정의된 SW 부품번호 부여 방법에 따르지 않고 임의로 작성하는 경우

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

4. 결론 및 향후 발전 방향

국방과학연구소 AGENCY FOR DEFENSE DEVELOPMENT

결론 및 향후 발전 방향

무기체계 SW PCA 활동 소개

- 수행 시기 및 절차
- 검토 중점사항
- 세부 업무 절차
- 수행 사례

"SW PCA" 규격화 품질 향상에 큰 기여

향후 발전 방향

- **SW PCA** 업무 및 수행 필요성 인식제고
- 규격화 일정 계획 수립 시 **SW PCA** 수행 일정 반영
- **SW PCA** 세부절차 및 가이드라인 개선

References

- 국방부, "국방전력발전업무훈령," 2017년 6월.
- 방위사업청, "표준화 업무지침," 2017년 1월.
- 방위사업청, "국방표준업무 실무핸드북," 2017년 10월.
- 방위사업청, "방위사업관리규정," 2016년 10월.
- 방위사업청, "무기체계 소프트웨어 개발 지원에 관한 규정," 2016년 10월.
- 방위사업청, "무기체계 소프트웨어 개발 및 관리 매뉴얼," 2017년 8월.
- 방위사업청, "SE기반 기술검토회의 가이드북," 2017년 6월.
- 방위사업청, "국방규격의 서식 및 작성에 관한 지침," 2017년 1월.
- 국방과학연구소, "무기체계 SW 개발관리 아카데미 교육자료," 2017년 9월.
- Al Florence, "Configuration Management," in MITRE, 2010년. https://ndiastorage.blob.core.usgovcloudapi.net/ndia/2010/CMMI/TuesdayTrack2_10963Florence.pdf

소프트웨어 제품 라인 가변부 구성 자동 관리 시스템 개발

김태호, 강성원, 이지현*
한국과학기술원, 전북대학교*
evan.kim@kaist.ac.kr, sungwon.kang@kaist.ac.kr, jihyun30@jbnu.ac.kr*

Development of the Variable Part Auto Configuration and Management System for Software Product Line

Taeho Kim, Sungwon Kang, Jihyun Lee*
KAIST, Chonbuk National University*

요 약

소프트웨어 제품 라인 공학에서 어플리케이션은 모든 제품군이 공통으로 갖는 공통부와 제품 별로 달라지는 부분인 가변부로 구성된다. 본 연구에서는 A기업의 어플리케이션 가변부 개발 현황의 문제점을 파악하고 이를 해결하는 시스템을 제시하고 구현한다.

1. 서 론

현대사회는 소비자의 요구가 급변하고 이에 따라 소비 트렌드 역시 급변하고 있다. 소프트웨어 시장 환경도 비슷한 용도의 다양한 소프트웨어가 시장에 빠르게 출시되고, 소비 트렌드의 변화 주기도 짧아지는 추세이다.

치열한 시장 경쟁에서 출시되는 소프트웨어가 우위를 점하기 위해서는 제품의 품질 경쟁력을 기반으로 시장의 선 점유가 필요하다. 이를 달성하기 위해 검증된 플랫폼을 기반으로 다양한 제품군을 동시에 개발하는 능력이 요구된다. 산업군에서는 이런 시장 상황에 대처하기 위해 소프트웨어 제품 라인 공학 (SPLE, Software Product Line Engineering)을 활용하여 어플리케이션을 개발하고 있다. SPLE에서는 검증된 플랫폼을 재사용하여 다양하고 유사한 소프트웨어 제품을 동시 개발, 개발 기간 단축을 통해 빠른 시장 출시를 가능하게 한다[1, 3].

플랫폼은 SPLE의 첫 단계인 도메인 공학을 통해 제품군의 공통성과 가변성을 식별하여 개발된 산출물이다. 어플리케이션 공학은 두 번째 단계로 플랫폼을 실체화하고 제품에 따라 달라지는 요소를 결합하여 제품을 만드는 단계이다. 이를 통해 어플리케이션이 개발되며 이는 모든 제품군이 공통으로 사용하는 공통부와 제품 내에서 변경되는 영역인 가변부로 구성된다.

본 연구에서는 A기업[1]의 기존 어플리케이션 가변부 개발 방법을 검토한 뒤, 현재 개발 방법의 문제를 분석하고 이를 해결한 가변부 구성 자동 관리 시스템(VACMS: Variable Part Auto Configuration and Management System)을 개발한다.

2. 기존 어플리케이션 개발 방법

본 논문에서 어플리케이션이란 제품 라인 공학에서 정의한 개념인 공통부와 가변부를 각각의 파티션으로 구성한 형태의 소프트웨어를 의미한다. 어플리케이션은 제품군에 따라 동일하게 사용하는 공통부와 변경되는 가변부가 별도의 파티션으로 구성된다. 이에 대한 특징으로 가변부 파티션 변경 시에 다른 제품군으로 전환이 가능하다.

2.1. 어플리케이션의 가변부

SPLE에서의 가변부는 플랫폼을 실체화한 후 어플리케이션 개발 단계에서 달라지는 소프트웨어 전 영역을 의미한다. 본 논문에서 정의하는 어플리케이션의 가변부는 플랫폼 영역에서의 변경사항은 포함하지 않는다. 플랫폼의 변경사항을 제외한 나머지 변경사항으로 어플리케이션 가변부를 정의한다.

2.2. 어플리케이션 가변부 구성 항목

[1] 업종: 전기전자, 주요품목: 이동통신단말기 및 가전제품, 종업원수: 35,000명

어플리케이션의 가변부의 주요 구성 항목은 제품군의 특징을 규정하기 위한 파일들로 구성된다. 구성 파일 중 대부분이 기능별 세부 피처를 정의한 설정 파일이다. 이 파일을 구성하는 항목의 설정에 따라 제품이 제공하는 기능의 범위와 동작 시나리오가 결정된다.

다음은 A기업에서 안드로이드 OS를 기반으로 개발한 어플리케이션의 가변부 구성 항목이다:

① 시스템 프로퍼티 파일: 윈도우 OS의 레지스트리와 같이 시스템 동작에 영향을 주는 설정 파일이다.
② 피처 설정 파일: 기능별 세부 피처를 정의하고, 시스템이 지원하는 기능 범위와 동작하는 시나리오에 영향을 주는 설정 파일이다.
③ APK (Android Package Kits) 오버레이 파일: 리소스만 존재하는 형태의 APK로 원본 APK가 이 리소스의 사용을 우선한다.
④ 기타: 음원 및 동영상 등의 컨텐츠 파일이다.

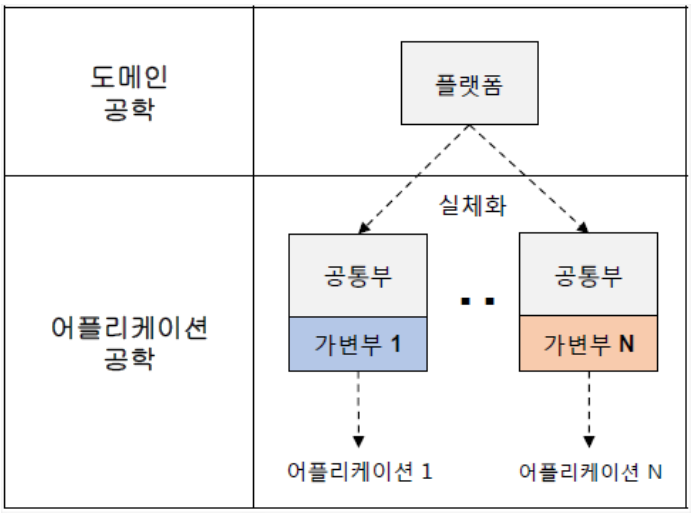


그림 1. 기존 어플리케이션 개발 방법

3. 기존 어플리케이션 개발 방법의 문제점 및 개선 방향

본 절에서는 A기업의 현재 어플리케이션 가변부 개발 프로세스에서의 문제점 도출을 통해 프로세스 개선 안과 이에 대한 기대효과를 기술한다.

3.1. 문제점

2.6.절과 같이 어플리케이션 가변부를 개발하는 과정에서 다음과 같은 문제점이 발생한다:

첫째, 가변부 구성 파일의 설정 변경 시 항목 간 의존성이 존재한다. 그러나 유관부서 간 업무 범위로 인해 설정이 의존적 항목이 누락되거나 잘못된 설정이 입력되어 문제가 발생한다.

둘째, 어플리케이션의 공통부 변경으로 인해 가변부를 구성 파일의 변경(추가/삭제)이 발생하게 된다. 이와 관련된 개발 문서가 유관부서에 공유되나 담당자의 실수(미숙지/오설정)로 인해 잘못된 설정이 입력되거나 혹은 누락된다. 위와 같은 문제점이 발생할 경우, 소프트웨어를 재 발행해야 한다. 이는 추가적인 개발 리소스가 요구되며 개발 인원 추가 투입, 개발 기간 지연을 유발한다.

셋째, 어플리케이션의 가변부 구성 진행 현황을 파악하기 어렵다. 제품군 별 가변부 구성 현황은 프로젝트 관리자가 확인한다. 모든 제품 개발자가 구성 완료 시에 공식적인 어플리케이션 가변부 파티션을 발행하게 된다. 구성 현황을 확인하기 위해 모든 제품 개발자 연락을 취한다. 이는 반복적이고 비생산적인 과정이라 할 수 있다.

넷째, 제품 개발자가 어플리케이션 가변부를 변경할 경우, 가변부 전체 항목에서 담당하고 있는 항목만을 선별해야 한다.

어플리케이션 가변부는 파티션으로 생성되고, 파티션은 파일의 집합, 파일은 항목들의 집합이라 할 수 있다. 담당자가 가변부 항목을 변경하기 위해 전체 가변부 항목 집합에서 찾고자 하는 항목을 검색해야 한다. 어플리케이션이 N개 존재할 경우, N번의 검색 시간이 소요된다.

4. 어플리케이션 가변부 구성 자동 관리 시스템 개발

본 절에서는 3.1절에서 기술한 기존 어플리케이션 가변부 개발 방법의 문제점을 해결하기 위한 시스템인 VACMS를 상세히 기술한다. VACMS는 어플리케이션 가변부 개발 프로세스를 지원할 시스템으로서 어플리케이션 가변부를 구성하고, 자동으로 가변부 파일을 생성 및 관리를 지원한다. VACMS는 다음과 같은 하부 시스템들로 구성된다:

① 가변부 정보 입력 시스템: 사용자가 가변부 구성에 필요한 정보를 입력하기 위한 하부시스템
② 가변부 정보 처리 시스템: DBMS 로부터 필요한 정보를 읽어 가변부 생성 시스템이 요구하는 데이터형으로 변환해 주는 하부시스템
③ 가변부 생성 시스템: 사용자로부터 입력된 정보를 기반으로 어플리케이션 가변부 파일을 생성하는 하부시스템
④ 가변부 생성 스크립트 관리 시스템: 가변부 파일 생성에 필요한 스크립트 파일을 관리하는 하부시스템
⑤ 사용자 입력 정보 DBMS (DataBase Management System): 가변부 생성에 필요한 사용자 입력 정보의 관리용 데이터베이스 관리 하부시스템

VACMS의 주요 개발 단계는 다음과 같다:

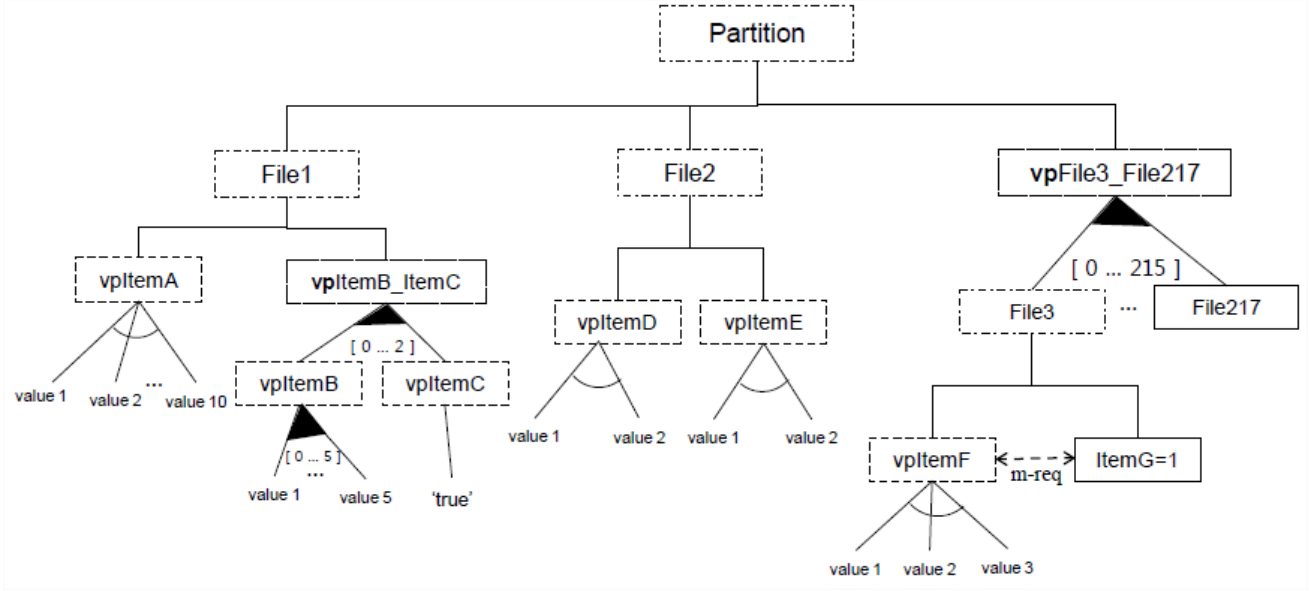


그림 2. CFM으로 표현한 어플리케이션 가변부의 메타 모델

4.1. VACMS 요구사항

플랫폼 개발자, 제품 개발자, 프로젝트 관리자와의 인터뷰를 통해 시스템을 통한 어플리케이션 가변부 개발을 할경우 필요한 시스템 요구사항을 취합한다. 취합한 요구사항들을 기능 및 비기능 요구사항으로 명세화한다.

4.2. 어플리케이션 가변부 자산 재공학

어플리케이션 가변부 자산 재공학(Re-engineering)의 목적은 다음과 같다: 첫째, 어플리케이션 가변부를 구성하는 항목에서 재사용이 가능한 자산과 아닌 자산을 분류한다. 둘째, 재사용이 가능한 항목과 그 이외 항목을 관리할 수 있는 시스템 기능을 도출하고, VACMS 설계에 반영한다.

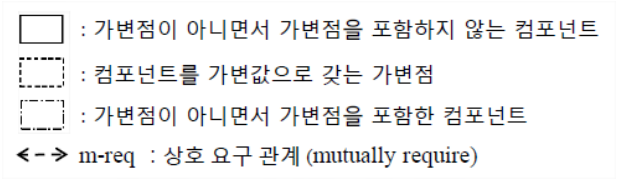
어플리케이션 가변부 자산 재공학은 아래와 같이 다섯 단계로 진행된다:

첫째, 현재 어플리케이션 가변부 구성 항목을 작성한다.

둘째, 어플리케이션 가변부 항목을 다음과 같은 자산으로 세분화한다. 공통자산과 공유자산은 재사용이 가능한 핵심자산 (Core asset)으로 정의한다[3]:

① 공통자산 (Common asset): 항목의 설정 값이 다를 수 있지만 모든 제품들에서 재사용되는 자산
② 공유자산 (Shared asset): 항목의 설정 값이 다를 수 있지만 일부 제품군에서 재사용되는 자산
③ 제품특정 부분 (Product specific part): 한 제품이 가지고 있지만 다른 제품에 공유되지 않는 부분

셋째, 어플리케이션 가변부에 대한 가변성 모델링[2]을 진행한다. 가변성 모델링이란, SPLE에서 아키텍처의 가변성을 표현하기 위한 방법이다. 그림 2는 가변성 모델링 방식 중 CFM (Composite Feature Model)을 사용하여 어플리케이션 가변부를 모델링한 메타 모델이다. CFM은 SPLE에서 가변성을 피처 수준에서 도형으로 표현하는 모델링 방식이다. 다음은 컴포넌트에 대한 CFM 표기법이다[6]:



넷째, CFM에서 파일, 항목의 가변성을 확인한다:

① 어플리케이션 가변부 파티션 구성 파일 중, 가변점이 아닌 파일을 1차로 분류한다.
② 1차로 분류한 파일의 구성 항목 중, 가변점의 가변값이 컴포넌트를 가변값으로 갖는 가변점이 아닌 항목을 2차로 분류한다.

다섯째, 가변부 항목을 핵심 자산과 제품특정 부분으로 분류한다[3].

4.3. VACMS 구현

VACMS의 아키텍처는 그림 3과 같다. VACMS의 주요 기능과 특징은 다음과 같다:

첫째, 주요 시스템 메뉴는 핵심자산과 제품특정 부분을 관리하는 핵심자산 관리 컴포넌트와 제품특정 부분 관리 컴포넌트이다. 핵심자산 및 제품특정 부분 관리 컴포넌트는 그림 4와 같이 각 모델 개발자 별로 확인할 수 있는 뷰어 (Viewer)를 제공하여 담당 항목의 확인 시간을 최소화한다.

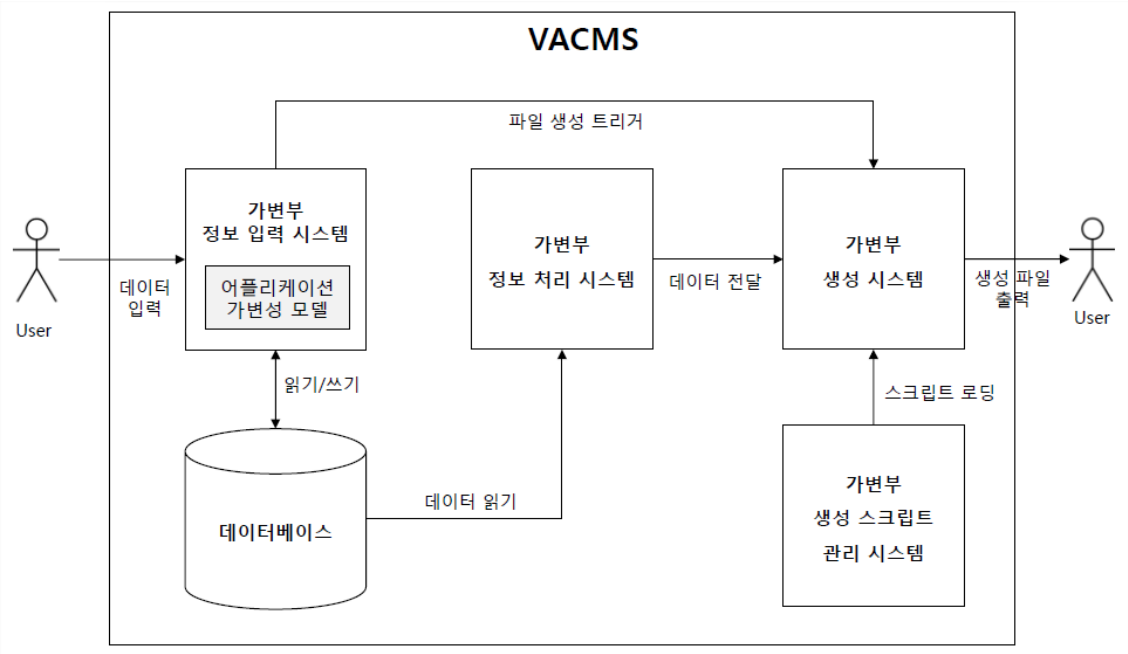


그림 3. VACMS 시스템 아키텍처

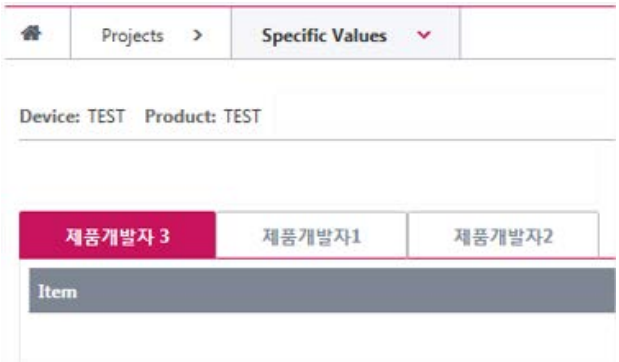


그림 4. 제품특정 부분 관리 메뉴의 제품 개발자 별 뷰어

둘째, PL 이 가변부 구성 진행 현황을 바로 파악할 수 있는 어플리케이션 가변부 모니터링 컴포넌트를 구현하며, 이 때 각 제품 담당자 지정 및 구성 현황을 쉽게 파악하도록 한다.

셋째, 가변부 자산을 등록할 수 있는 등록 관리 컴포넌트를 구현한다. 등록한 항목의 세부 정보에서 항목 간 종속성을 설정할 수 있도록 구현한다. 또한 항목 간 종속성을 설정할 수 있는 메뉴를 구현한다. 이는 제품 개발자가 어플리케이션 가변부 구성 시, 종속적인 항목의 누락을 방지한다.

5. 결론

본 논문에서는 A기업의 소프트웨어 제품 라인 공학을 활용한 어플리케이션의 가변부 개발 현황을 분석하여 어플리케이션 가변부 개발 프로세스의 문제점을 도출하고 이를 개선하기 위한 개발 방안을 제시한 후, 개선된 개발을 가능하게 하는 시스템인 VACMS를 구현하였다.

참고문헌

[1] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns* (The SEI Series in Software Engineering), Addison-Wesley Professional, 2001.

[2] S. D. Kim, Soo Ho Chang, "*A variability modeling method for adaptable services in service-oriented computing*," In Proc. of the 11th Int. Conf. on Software Product Lines, SPLC, 2007.

[3] 강성원, 체계적인 소프트웨어 제품라인 개발, 홍릉과학출판사, 2017 년 6 월.

[4] Klaus Pohl, Günter Böckle, Frank J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*, Springer, 2005.

[5] Kwan Woo Lee, *"Managing and Modeling Variability of UML Based FORM Architectures through Feature-Architecture Mapping,"* KIPS Journal D, 2012.

[6] 강성원, “CFM: 합성적 피처모델”, CS-TR-2017-414, KAIST 전산학부 Technical Report, 2017.

금속재료 피로시험의 최적설계 소프트웨어의 클린코드 적용 사례

정인용 O1 박상건 1 손현승 1 김광진 2

㈜모아소프트 신뢰성기술연구소 1

㈜ 알에스피 2

{iyjung, sgpark, hson}@moasoftware.co.kr1

kjkim@rsp.co.kr2

A Case Study of Clean Code for Optimal Design Software of Metal Material Fatigue Test

Jung O1 Sang Gun Park1 Hyun Seung Son1 Kwang-Jin Kim2

Reliability Technology Institute, Moasoft1

RSP2

요 약

피로시험 최적설계 소프트웨어는 금속재료의 피로시험에서 최적의 시험결과가 나오도록 시험시 시험하중을 결정하고 시험시간을 예측할 수 있도록 해 시험결과를 자동으로 분석 가능하다. 본 논문에서는 이 피로시험 최적설계 소프트웨어의 성능향상을 위해서 클린코드를 적용하고 사례를 소개한다. 클린코드의 3가지 기법 객체 정규화, 반복문 최적화, Exception 예외처리 성능 향상의 적용으로 피로시험 최적설계 소프트웨어의 개발에 있어서 성능 향상을 이룰 수가 있다.

1. 서 론

대부분의 기계는 운동 상태에 있고 그 사용기간이 경과하면 재료의 강도가 저하된다. 하지만 그 저하되는 속도는 사람이 인지하기에 매우 느린 경우가 많고, 파괴 시점 자체를 예측하기가 어렵다. 그리고 외형상의 큰 변화가 없어 어느 순간에 돌발적으로 파괴가 일어나 큰 사고가 일어나기도 한다.

피로시험은 실용 기계부품이나 구조물 등의 재료를 대상으로 예상치 못한 파괴를 미연에 방지하고 부품의 수명이나 교체시기를 예측하는 것이다[1-3]. 피로시험의 목적은 기계 제품의 안전을 도모하고 물적 인적 피해를 방지키 위하여 수행된다.

본 논문에서는 금속재료 피로시험의 최적설계를 도와주는 소프트웨어개발에 클린코드 적용사례를 보여주고자 한다. 클린코드는 애자일 소프트웨어의 혁명적인 패러다임으로 로버트 마틴에 의해 오브젝트 멘토(Object Mentor)의 동료들과 함께 개발과정에서 힘을 모아 만든 기법이다[4]. 클린코드에서는 아주 많은 코드 들에서 그 코드의 무엇이 옳은지, 그른지 생각하며 프로그램 전문가로서 자신이 지니는 가치를 돌아보게 한다.

본 논문의 구성은 다음과 같다. 2장에서는 피로시험에 대한 요구사항 소개한다. 3장은 클린코드에 대해서 설명한다. 4장은 피로시험의 최적설계 소프트웨어의 클린코드의 적용사례를 언급한다. 5장에서는 결론 및 향후 연구를 언급한다.

† 본 연구는 과학기술정보통신부 및 연구성과실용화진흥원 2017년 연구개발서비스혁신역량강화사업의 지원을 받아 수행하였음(2017K000118)

2. 피로시험 소프트웨어의 요구사항

2.1 클린코드

프로그래밍이란 컴퓨터가 해석하고 실행할 수 있도록 사용자의 요구사항을 구체적으로 명시하고 소스코드를 통해 구현화 시키는 작업이다. 보기 쉽고 최적화된 상태로 소스코드를 작성할 수 있는지, 또한 나쁜 코드(Bad Code) 여부에 따라 개발자의 실력과 자질이 결정된다고 해도 과언이 아니다.

나쁜 코드는 왜 작성하게 되는 것일까? 나쁜 코드가 작성되는 이유는 다양하겠지만 대부분의 개발자는 자신이 작성한 프로그램이 잘 돌아가는 순간 안심하게 된다. 비록 그 코드가 지저분하고 마음에 들지 않더라도 일단은 잘 동작하는 코드는 개발자를 방심하게 만들기 때문이다. 나쁜 코드를 방치하면 또 다른 나쁜 코드가 발생할 가능성이 높아진다. 따라서 적절하게 조치하지 않은 나쁜 코드는 이후에 더 큰 버그를 발생시킬 수 있고, 전체 시스템의 개발 속도를 크게 떨어뜨리기도 한다.

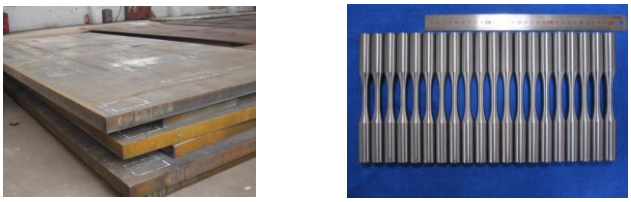
클린코드는 이러한 나쁜 코드를 식별하고 코드를 읽기 쉽게 만들어, 사람이 코드를 이해하는 데 들이는 시간을 최소화 하는 방식이다. 나쁜 코드를 클린코드로

변경하기 위해 디자인 패턴, 리팩토링 등 다양한 기법들을 알고 있어야 한다. 소프트웨어를 보다 쉽게 이해하고, 적은 비용으로 수정할 수 있도록 내부 구조를 변경하는 일련의 과정으로 볼 수 있다. 즉 기존에 작성된 소스를 리팩토링 하여 겉으로 보이는 동작의 변화 없이 소프트웨어를 수정하는 개념으로 볼 수 있는 것이다.

2.2 피로시험 소프트웨어의 요구사항

피로시험에 있어서는 특히 시험편의 선정에 주의해야한다. 시편은 원소재의 임의의 부분에서 채취되어야 하고, 신뢰성 있는 통계분석을 위해서는 다음과 같이 최소의 개수가 요구된다. 예비시험 또는 부품개발을 위한 실험 최소 6-12개, 설계자료를 위한 실험 최소 12-24개이다. 선정된 시험편들에 대한 피로시험의 수행과정은 다음과 같다.

① 금속재료의 채취 및 가공



(가) 금속재료 (나) 시험의 위한 가공

그림 1 금속 재료 및 시험을 위해 가공된 예

② 피로 실험 설계(시험일정 및 데이터 준비)

③ 피로 실험 수행

그림 2 피로시험의 수행[5]

④ 결과 분석

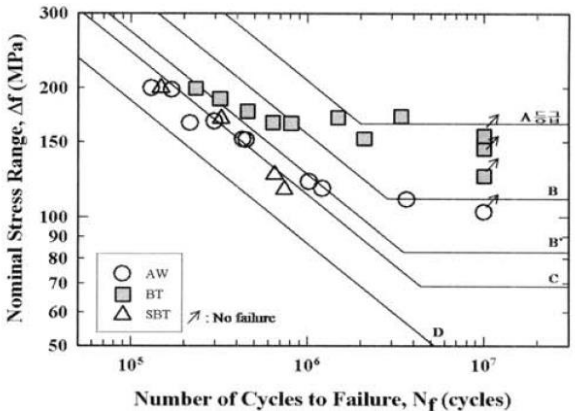


그림 3 S-N 커브를 통한 결과 분석[5]

금속재로 피로시험의 최적설계 소프트웨어는 앞의 피로시험 과정에 있어서 피로시험 설계를 도와주는 일을 한다. 이 소프트웨어는 금속재료의 피로시험에서 최적의 시험결과가 나오도록 시험시 시험하중을 결정한다. 또한 시험시간을 예측할 수 있도록 하고 시험결과를 자동으로 분석하여 인위적인 개입을 차단하도록 하는 요구사항을 가지고 있다.

3. 클린코드 적용방법

3.1 객체 정규화

객체의 정규화란 많은 객체의 사본을 몇 개의 객체로 대체하는 기법이다. 가능하면 많은 객체를 한 개(혹은 몇 개)의 객체로 대체하는 것이 좋다. 다음은 static 변수를 이용한 객체 정규화의 예시이다.

그림 4와 같이 NormalCs 객체가 한 개 필요하다면 static 변수를 사용하는 것이 좋다. 생성자를 private로 정의하고 클래스 내부에 변수를 유지하는 방법을 사용할 수 있다. 클래스와 그 안의 모든 메소드와 각 변수를 static으로 정의하는 방법으로 정규화 효과를 볼 수 있다. 또한 정규화한 객체는 각 객체마다 '=='연산자를 사용하여 비교할 수 있으므로 성능을 향상시킬 수 있다. '=='연산자는 간단한 포인터 비교이기 때문에 equals를 사용하는 것보다 속도가 빠르다.

```
public class NormalCs
{
    public static NormalCs SINGLETON = new NormalCs(10);
    Vector[] pool;
    Boolean[] inUse;

    private NormalCs(int itialPoolSize){ }
}
```

그림 4 객체 정규화 예시코드

3.2 반복문 최적화

매번 실행하는 필요 없는 코드는 루프에서 최소화 시키면 성능을 향상시킬 수 있다. 또한 같은 코드를 실행하더라도 메소드 호출은 속도가 더 느리기 때문에 루프에서 메소드를 계속 반복해서 호출하면 프로그램이 느려진다. 그러므로 메소드 호출은 루프에서 가능한 피하는 것이 좋다. 뿐만 아니라, 배열은 루프 밖에서 임시 변수에 값을 부여하는 것이 루프에서 반복하여 배열을 이용하는 것보다 성능 면에서 좋다. 가상 머신은 배열에서 범위를 확인하기 때문에 배열을 이용하는 것은 임시 변수를 이용하는 것보다 성능 면에서 떨어지기 때문이다.

그림 5에서 (2)는 (1)을 임시 변수를 이용하여 최적화한다. (2)와 같이 배열을 루프 밖에서 이용한 것이 (1)의 루프 안에서 이용한 것보다 두배 더 빠르다.

```
/* (1) */
int[] arr = new int[10];

for (int i = 0; i < 1000; i++)
{
    arr[0] += 10;
}

/* (2) */
int count = arr[0];

for (int i = 0; i < 1000; i++)
{
    count += 10;
}
arr[0] = count;
```

그림 5 반복문 최적화

3.3 Exception 예외처리 성능 향상

try-catch는 일반적으로 예외 상황이 발생하지 않을 경우에는 성능 저하가 없으나 몇몇 가상머신에서는 약간의 성능 저하가 있을 수 있다. 그림 6과 같이 전체 루프에 try-catch를 1회 실험한 결과 같은 코드를 루프에서 반복할 때마다 try-catch를 하는 것과, 전체에서 try-catch를 한번 하는 것과 비교하여 여러 번 반복하는 경우 수행속도가 약 10% 증가하는 것을 확인할 수 있었다.

```
try
{
    Integer i = (Interger)o;
    return false;
}
catch (Exception e)
{
    return true;
}
```

그림 6 try-catch예외처리

그러나 사실 상 예외상황을 생성하는데 걸리는 시간은 몇 백 줄의 프로그램이 수행하는데 걸리는 시간과 동일할 정도여서 catch부분이 실행되면 성능이 상당히 떨어진다. 이것은 대부분 예외 상황이 발생했을 때, 스택 정보를 얻는 시간 때문에 생기는 이유이다.

instanceof를 사용하는 것이 ClassCastException이 발생하는 Try-catch를 이용한 코드의 경우보다 훨씬 빠르게 처리한다. 객체 타입을 꼭 확인해야 하고 또한 그것의 예외처리에 시간이 많이 걸린 것을 확인했다면, 그림 7과 같이 캐스팅을 하는 대신에 instanceof를 사용하는 것이 좋다. 캐스팅은 성능면에 문제가 있고 소스 코드도 복잡하기 때문에 사용하지 않는 것이 좋다.

```
if( o instanceof Integer)
    return false;
else
    return true;
```

그림 7 instanceof 예외처리

4. 적용사례

금속재료 피로시험의 최적설계 소프트웨어의 화면 구성은 그림 8과 같다.

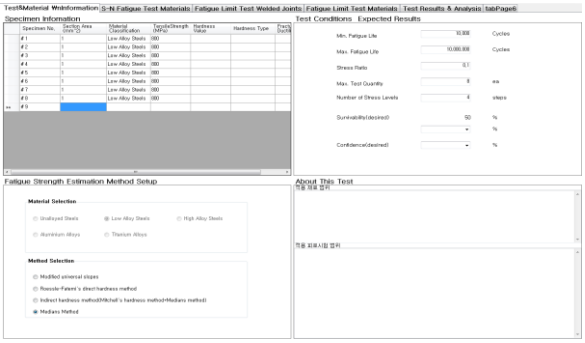


그림 8 금속재료 피로시험 소프트웨어 화면 구성

각 다른 업무를 수행하는 탭 5개와 각 탭마다 해당 업무의 각 기능을 맡고있는 레이아웃 화면으로 구성되어 있다. 각 레이아웃은 하나의 Panel컴포넌트 클래스이다. 각 탭의 레이아웃끼리 데이터 상호작용 할 때, 레이아웃의 정보에 접근하기 위해서는 각 레이아웃 클래스의 객체를 필요로 할 때마다 항시 생성해 주어야 했다.

그림 9의 코드는 하나의 탭에 각 레이아웃을 객체를 정규화 형식으로 접근하는 방식으로 구현한 것이다. 해당 탭의 각 레이아웃 클래스들을 모두 객체화 하고 해당 탭을 static 변수를 이용하여, 따로 해당 탭의 객체를 더 이상 생성하지 않고 바로 접근할 수 있도록 한다.

```
public partial class Tab1_UserControl : UserControl
{
    public Tab1_Panel1 tab1_panel1 = new Tab1_Panel1();
    public Tab1_Panel2 tab1_panel2 = new Tab1_Panel2();
    public Tab1_Panel3 tab1_panel3 = new Tab1_Panel3();
    public Tab1_Panel4 tab1_panel4 = new Tab1_Panel4();

    static Tab1_UserControl tab1_UserControl= null;

    public Tab1_UserControl()
    {
        InitializeComponent();

        tab1_UserControl = this;


    }
    public static Tab1_UserControl getTab1_Control()
    {
        return tab1_UserControl;
    }
```

그림 9 객체 정규화 적용 코드

피로시험 연산을 위해서는 그림 10과 같이 각 수식에 알맞은 데이터가 수시로 들어가져야 한다. 각 데이터가 들어가질 때마다 반복문을 통해 연산을 반복하게 되며, 당연히 데이터 수가 많아질수록 연산횟수도 비례하게 된다. 금속재료 피로시험 소프트웨어에서는 연산을 수행하는 함수들을 따로 클래스화 하여, 함수 호출을

통해 필요한 연산을 할 수 있도록 한다.

```
/* 4번 식 */
public double Tab1_RoessleFatemisHardnessMethod(double E, double Nf, double HB)
{
    double TDE = ((4.25 * HB + 225) / E) * Math.Pow((2 * Nf), -0.09) + ((0.32 * Math.Pow(HB, 2) - 4
    return TDE;
}

/* 5번 식 */
public double Tab1_ModifiedMitChellMethod(double sigmaU, double E, double Nf, double 입실론)
{
    double TDE = ((sigmaU + 335) / E) * Math.Pow((2 * Nf), -(1 / 6) * Math.Log((sigmaU + 335) / (0.
    return TDE;
}

/* 6번 식 */
public double Tab1_MediansMethodSteel(double sigmaU, double E, double Nf)
{
    double TDE = 1.5 * (sigmaU / E) * Math.Pow((2 * Nf), -0.09) + 0.45 * Math.Pow(2 * Nf, -0.59);
    return TDE;
}

/* 7번 식 */
public double Tab1_MediansMethodAluminium(double sigmaU, double E, double Nf)
{
    double TDE = 1.9 * (sigmaU / E) * Math.Pow((2 * Nf), -0.11) + 0.28 * Math.Pow(2 * Nf, -0.66);
    return TDE;
}
```

그림 10 피로시험 알고리즘 코드

그림 11은 데이터 그리드 뷰 안에 있는 행 개수만큼 각 데이터를 가져와서 연산을 수행하는 코드이다. 루프문이 실행될 때마다 연산 클래스 객체를 생성하여 함수를 호출하는 방식이다. 데이터 개수가 500개가 넘어가니 모두 수행하는 데 걸리는 시간이 1000(n/s)가 걸렸다.

```
for (int i = 0; i < dataGridView1.Rows.Count; i++)
{
    Tab1_Calculate calculte = new Tab1_Calculate();

    double TDE = calculte.Tab1_ModifiedUniversalSlopes(SigmaU, E, N1, ePsilon);
}
```

그림 11 연산 알고리즘 사용 코드(클린코드 적용 이전)

그림 12는 연산 클래스를 루프문 밖에서 객체화 하여, 루프문 안에서 객체의 함수호출만 하도록 하였다. 위 코드를 이용하여 데이터 개수 500개를 처리하니 500(n/s)가 걸린 것을 확인하였다. 같은 기능을 하는 두개의 코드에 처리시간이 다른 것을 확인할 수 있었다.

```
Tab1_Calculate calculte = new Tab1_Calculate();
for (int i = 0; i < dataGridView1.Rows.Count; i++)
{
    double TDE = calculte.Tab1_ModifiedUniversalSlopes(SigmaU, E, N1, ePsilon);
}
```

그림 12 연산 알고리즘 사용 코드(클린코드 적용 이후)

금속재료 피로시험 소프트웨어 내에서 dataGridView 내의 데이터를 가져올 때, 그림 13과 같이 데이터 형식이 무조건 Object로 반환하게 된다. 그래서 String이나 int형 변환을 이용하여 데이터를 활용할 수 있으나, 부득이하게 문자열인 데이터를 int형으로 변환할 수도 있다.

Specimen Infomation

	Specimen No.	Section Area (mm^2)	Material Classification	TensileStrength (MPa)	Hardness Value	Hardness Type	Fractu Ductili
	# 1	1	Low Alloy Steels	800			
	# 2	1	Low Alloy Steels	800			
	# 3	1	Low Alloy Steels	800			
	# 4	1	Low Alloy Steels	800			
	# 5	1	Low Alloy Steels	800			
	# 6	1	Low Alloy Steels	800			
	# 7	1	Low Alloy Steels	800			
▸▸	# 8						

그림 13 DataGridView 사용자 입력

또한, 그림 14와 같이 TextBox 안의 통계형식의 숫자를 데이터를 가져와 연산을 시도할 때도 많다. 그러나 TextBox 내의 데이터는 String형 데이터를 반환하게 된다. 만약 'Max. Fatigue Life' 항목의 데이터를 가져오게 될 시, '100,000,000'라는 문자열 데이터를 가져오게 되고, 연산을 위해 형 변환을 하게 될 시, 오류를 발생하여 미리 작성한 예외처리 코드에서 캐스팅을 하게 될 것이다. 그러나 try-catch문을 통하여 ClassCastException를 이용하면 예외처리에 시간이 많이 걸리게 된다.

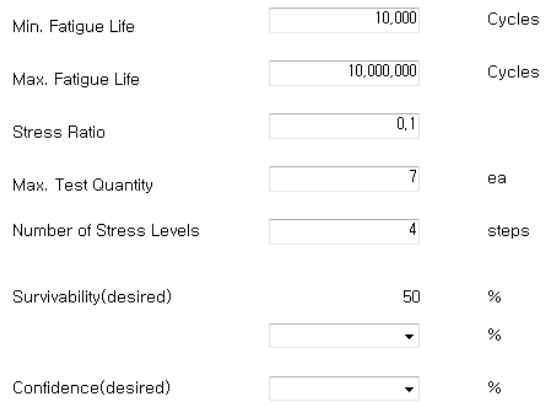


그림 14 TextBox 사용자 입력

금속재료 피로시험 소프트웨어 내에서 대량의 데이터를 하나씩 불러와 해당 데이터를 가지고 연산을 수행하여 S-N Curve 그래프 정보를 생성하게 된다. 해당 시스템에서 데이터를 한번 불러올 시 약 500개 정도의 데이터를 가져오게 되는데, 이러한 데이터를 바로 연산처리를 하게 되면, 오류가 발생하는 데이터가 적지않게 나타나며, try-catch문을 이용하여 캐스팅을 하게 될 시 오류 발생 개수만큼 예외처리에 시간이 걸리게 된다. 그러나 위와 같이 미리 객체타입을 확인하고 그것에 맞게 instanceof를 사용으로 예외처리를 작성해 여러 개의 예외처리 작업을 수행하는데 걸리는 시간을 최소화 시킬 수 있었다.

```
if(ch instanceof AreaSection){
    AreaSection Section = (AreaSection) ch;
    System.OutOfMemoryException.println(AreaSection.getMessage());
}else if (ch instanceof Fatigue){
    Fatigue fat = (Fatigue) ch;
    System.OutOfMemoryException.println(Fatigue.getMessage());
}else if(ch instanceof Avatar){
    Avatar av = (Avatar) ch;
    System.OutOfMemoryException.println(av.getMessage());
```

그림 15 instanceof 적용

5. 결론

본 논문에서는 금속재료 피로시험의 최적설계를 도와주는 소프트웨어개발에 클린코드 적용사례를 보여주었다. 이를 위해 클린코드의 3가지 기법인 객체 정규화, 반복문 최적화, Exception 예외처리 성능 향상을 적용하였다. 그 결과 피로시험 최적설계 소프트웨어의 개발에 있어서 성능 향상을 이룰 수가 있었다.

향후연구로 클린코드의 적용범위를 늘려 피로시험 소프트웨어 전체 코드에 반영하고자 한다. 또한 고품질 소프트웨어를 위한 다양한 방법을 적용하고자 한다.

관련연구

[1] Yung-Li Lee, Jwo Pan, Richard Hathaway, Mark Barkey, “Fatigue Testing and Analysis”, ELSEVIER, 2004.

[2] 장동일, “구조공학에서의 파괴와 피로”, 동명사, 2000.

[3]일본강구조협회, “강구조물의 피로설계지침·동해설”, 과학기술, 2000.

[4] 로버트 C. 마틴, “Clean Code”, 인사이트, 2008.

[5] 김인태, 정영수, 김광진, 이동욱, “블라스트 표면처리가 하중비전달형 십자필렛 용접이음의 피로거동에 미치는 영향”, 한국강구조학회 논문집, 제21권 1호, 55-62, 2009.

AOP를 적용한 Fault Manager 컴포넌트 설계

[1] 오성수, [1] 김지성, [2] 김수동
[1](주) 효성중공업, [2] 숭실대학교 소프트웨어학부
[1]Sungsoo_oh@hyosung.com, [1]intelligent@hyosung.com, [2]sdkim777@gmail.com

Design of Fault Manager Component Applying Aspect Oriented Programming

[1]Sung Soo Oh, [1]Ji Sung Kim, [2]Soo Dong Kim
[1]Power & Industrial Systems R&DB Center of Hyosung Co., [2]Soongsil University

요 약

4차 산업혁명이 핵심 키워드로 대두되면서 산업의 플랫폼화가 진행되면서 전력분야에서도 전력망을 제어하여 효율적인 에너지 관리를 위한 플랫폼에 대한 연구 및 개발이 활발하게 이루어지고 있다. 4차 산업혁명으로 소프트웨어의 중요성이 커지면서 소프트웨어의 규모와 복잡도 또한 증가하고 있다. 그에 따라 기존 객체 지향 설계의 단점이 드러나고 있다. 이에 소프트웨어의 품질을 높이기 위해 관점 지향 설계(AOSD, Aspect Oriented Software Design) 기법을 적용한 Power Grid 플랫폼 소프트웨어의 설계 사례에 대해 소개한다.

1. 서론

근래 4차 산업혁명이 핵심 키워드로 대두되면서 산업의 플랫폼화가 진행되고 있다. 이에 전력분야에서도 전력망(Power Grid)을 제어하여 효율적인 에너지 관리를 위한 플랫폼에 대한 연구 및 개발이 활발하게 진행되고 있다.

4차 산업혁명으로 소프트웨어의 중요성이 커지면서 소프트웨어의 규모와 복잡도 또한 증가하고 있다. 이는 소프트웨어의 완전한 객체지향 설계를 어렵게 한다. 즉, 하나의 기능을 하나의 모듈로 캡슐화하기 어려워져서 기능들이 여러 모듈에 분산(scattering)되어 존재한다는 것을 의미한다. 분산된 코드는 의존성이 매우 강해서 소프트웨어의 유지보수를 어렵게 하고 모듈의 재사용성을 떨어트린다는 문제점이 발생할 수 있다.

본 논문에서는 이러한 단점을 보완하고 소프트웨어의 품질을 높이기 위해 프로그램의 여러 곳에서 사용되는 기능들을 모아서 처리하는 관점 지향 설계 기법(AOSD, Aspect Oriented Software Design)을 적용하여 Power Grid 플랫폼의 소프트웨어 설계했다.

본 논문의 구성은 다음과 같다. 2장에서는 소프트웨어 설계에 사용되는 대표적인 설계 기법의 특징과 장점 및 단점을 비교, 분석하고 3장에서는 Power Grid 플랫폼 시스템에 포함된 기능 컴포넌트에 대한 분석을 진행한다. 4장에서는 관점 지향 설계 기법을 적용하여 식별한 Power Grid 플랫폼 시스템의 부가 기능 중에 시스템에서 발생하는 다양한 고장을 관리하는 컴포넌트인 Fault Manager를 설계한 사례에 대해 기술한다.

2. 소프트웨어 설계 기법

소프트웨어 설계 기법은 매우 다양한 방법이 있지만 크게 기능 중심으로 설계하는 구조적(Structured) 설계 방법과 데이터를 중심으로 설계하는 객체 지향(Object Oriented) 설계 방법으로 구분할 수 있다. 표 1은 소프트웨어 설계에 사용되는 대표적인 방법인 기능 지향 설계(FOSD, Function Oriented Software Design), 객체 지향 설계(OOSD, Object Oriented Software Design), 관점 지향 설계 기법(AOSD, Aspect Oriented Software Design)의 특징과 장점 및 단점을 정리한 것이다.

표 1. 소프트웨어 설계 기법 특징과 장점 및 단점

설계 기법		설명
기능 지향	특징	· 시스템을 주요 기능 단위로 분리하여 설계 · 하향식 접근, 분할 정복식 접근
	장점	· 개발이 쉽고 개발기간이 짧음
	단점	· 기능 위주의 설계로 변화에 취약함 · 유지 보수 및 관리가 어려움
객체 지향	특징	· 체인 클래스(객체) 기반 설계 · 상향식 접근, 객체간 상호 관계 정의
	장점	· 객체의 독립성으로 재사용성이 좋음 · 유지 보수가 용이
	단점	· 핵심 기능과 부가 기능의 분리가 힘들어 중복, 복잡한 코드가 될 수 있음
관점 지향	특징	· 객체 지향 설계의 특징을 포함 · 관점 분리(핵심, 부가 기능) 설계
	장점	· 객체 지향의 장점을 부각, 단점 보완 · 중복코드, 복잡한 코드 문제 해결
	단점	· 설계 및 개발이 복잡하다.

그림 1은 표 1에서 비교, 분석한 기능 지향 설계, 객체 지향 설계, 관점 지향 설계 기법을 적용한 시스템 구조를 보여준다.

기능 지향 설계는 시스템을 주요 기능 단위로 분해하여 설계하고 기능 간의 상호작용으로 동작한다. 객체 지향 설계는 객체들을 클래스로 구현하고 클래스를 구체화한 인스턴트 간의 상호작용으로 시스템이 동작한다. 관점 지향 설계는 객체 지향 설계에서 각 모듈에 분산되어 있는 부가 기능(Cross-Cutting Concern)을 분리하여 설계하고 핵심 기능(Core Concern)을 수행하는 동안 필요한 지점에서 부가 기능을 핵심 기능에 적용하는 방식으로 동작한다.

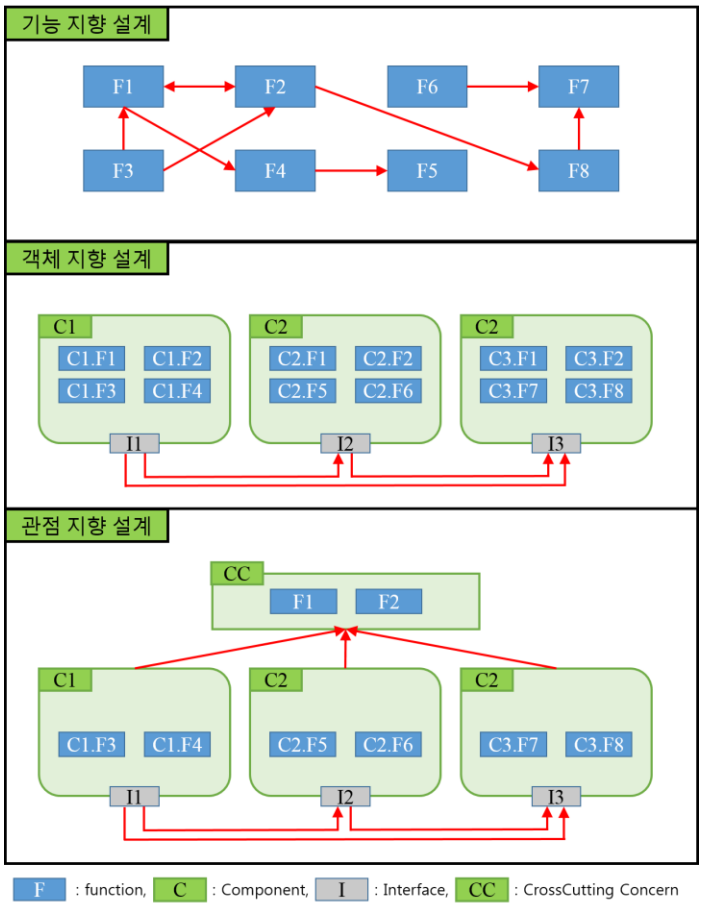


그림 1. 소프트웨어 설계 기법 별 시스템 구조

3. Power Grid 플랫폼 시스템의 기능 분석

Power Grid 플랫폼의 소프트웨어 설계에 관점 지향 설계 기법을 적용하기 위해 시스템에 포함된 기능에 대한 분석을 진행했다. 기능 분석을 위해 시스템 구성요소를 파악하고 구성요소 별로 필요한 기능을 식별했다. 그 다음 각 기능의 핵심 기능에서 사용하는 부가 기능을 식별하는 순으로 진행했다.

Power Grid 플랫폼 시스템의 구성요소는 어플리케이션, 디바이스, 플랫폼, 사용자가 있으며 시스템에 포함된 기능은 크게 프로파일 관리, 고장 관리, 세션 관리, 데이터 수집 관리, 데이터 분석 관리, 플랫폼 제어 관리, 어플리케이션 제어 관리, 디바이스 제어 관리, 사용자 계정 관리의 9개 컴포넌트로 분류할 수 있다.

플랫폼 제어 관리, 어플리케이션 제어 관리, 디바이스 제어 관리, 데이터 수집 관리, 데이터 분석 관리, 사용자 계정 관리 컴포넌트는 Power Grid 플랫폼 시스템에서 특정 작업을 수행하는 모듈로 핵심 기능으로 분류할 수 있고 프로파일 관리, 고장 관리, 세션 관리는 핵심 기능을 수행하는데 필요한 부가적인 기능으로 분류할 수 있다. 표 2는 Power Grid 플랫폼 시스템의 기능 컴포넌트와 각 컴포넌트에서 수행하는 기능에 대한 설명이다.

표 2. Power Grid 플랫폼 시스템의 기능 컴포넌트

컴포넌트	설명
프로파일 관리	· 시스템의 구성요소에 대한 다양한 설정 값과 데이터에 대한 프로파일 관리하는 기능
세션 관리	· 시스템에서 발생하는 모든 세션을 관리하는 기능
고장 관리	· 시스템에서 발생하는 고장을 탐지, 분석, 처리하고 사용자에게 알림을 보내는 기능
데이터 수집 관리	· 플랫폼에 연결된 디바이스에서 발생한 데이터를 취득하고 관리하는 기능
데이터 분석 관리	· 디바이스에서 취득한 데이터를 분석하고 필요에 따라 저장한 데이터를 가공하는 기능
플랫폼 제어 관리	· 플랫폼을 제어하는 기능
어플리케이션 제어 관리	· 어플리케이션 절차에 따라 제어하는 기능
디바이스 제어 관리	· 설치된 디바이스를 제어하는 기능
사용자 계정 관리	· 소프트웨어를 사용하는 사용자의 계정을 생성, 삭제하고 계정에 따라 권한을 관리하는 기능

4. 관점 지향 설계 기법을 적용한 Fault Manager 컴포넌트 설계

이 장에서는 관점 지향 설계 기법을 적용하여 Aspect 컴포넌트로 설계한 컴포넌트 중에 Power Grid 플랫폼 시스템에서 발생하는 다양한 고장을 관리하는 Fault Manager 컴포넌트를 사례로 설계 과정을 설명한다.

Power Grid 플랫폼 시스템의 Fault Manager 컴포넌트를 설계하는 과정에서 고려한 사항은 다음 세 가지가 있다. 첫 번째로 Fault Manager 컴포넌트에서 수행하는 기능을 분류하여 Aspect 컴포넌트로 정의했다. 두 번째로 템플릿 메소드 패턴을 적용하여 기능별로 중복되는 부분을 상위 클래스에 일반화하고 세부 구현은 하위 클래스에서 하도록 설계하였다. 세 번째로 객체 지향 설계 원칙 중에 확장엔 열려 있고 변화에는 닫혀 있는 원칙인 개방 폐쇄의 원칙(OCP, Open Closed Principle)을 고려하여 설계했다.

아래 그림은 위의 세 가지를 고려하여 설계한 Fault Manager 컴포넌트의 클래스 다이어그램을 보여준다. Fault Manager 컴포넌트는 고장을 분석하고 복구하며, 사용자에게 알림을 보내고 알림 메시지에 대한 사용자의 수신을 확인하는 하위 기능을 갖는다.

그림 2는 QualityManager 기능의 클래스 다이어그램이다. QualityManager 기능은 시스템에서 발생하는 고장을 탐지하고 사용자에게 알림 메시지를 보낸다. 시스템에서 발생하는 고장은 다양한 종류가 있기 때문에 QualityManager는 추상(Abstract) 클래스로 정의하고 세부 기능을 하위 구체(Concrete) 클래스에 정의했다. 상위 클래스의 notify 메소드는 고장을 탐지하여 사용자에게 전송하는 템플릿 메소드로 하위 클래스에 상속되지 않는다.

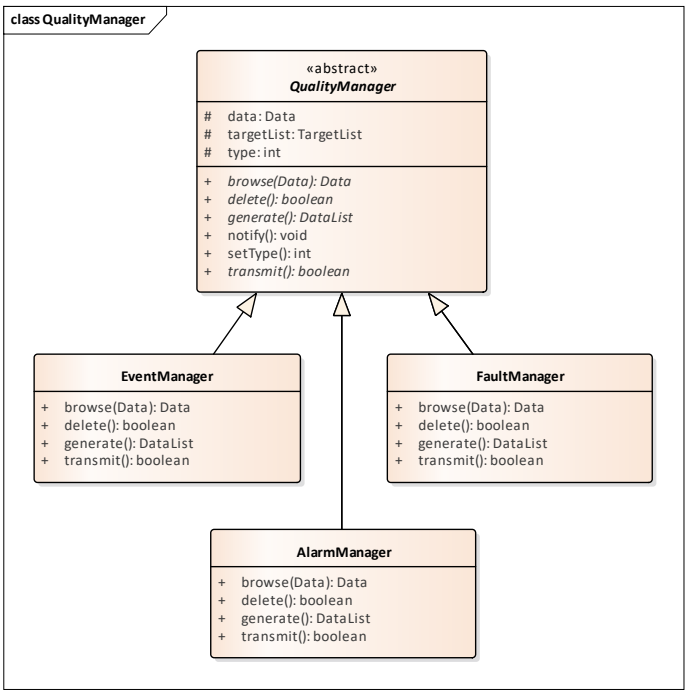


그림 2. QualityManager 클래스의 클래스 다이어그램

그림 3은 FailureRecovery 기능의 클래스 다이어그램이다. FailureRecovery는 Power Grid 플랫폼 시스템에서 발생한 고장의 원인을 분석하고 고장 때문에 발생한 피해를 복구하는 기능이다. FailureRecovery 클래스 또한 QualityManager 클래스와 같은 방법으로 정의했다. 상위 클래스의 recoverFailure 메소드가 원인 분석, 복구 방법 선택, 손상 데이터 복구 기능을 통해 고장을 복구하는 기능의 템플릿 메소드가 되고 하위 클래스에 상속되지 않는다.

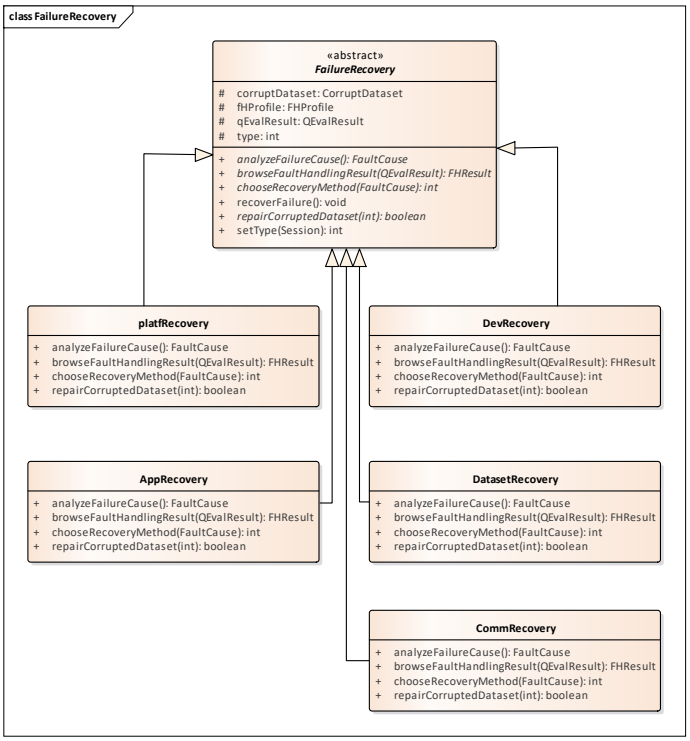


그림 3. FailureRecovery 클래스의 클래스 다이어그램

그림 4는 ACKHandler 기능의 클래스 다이어그램이다.

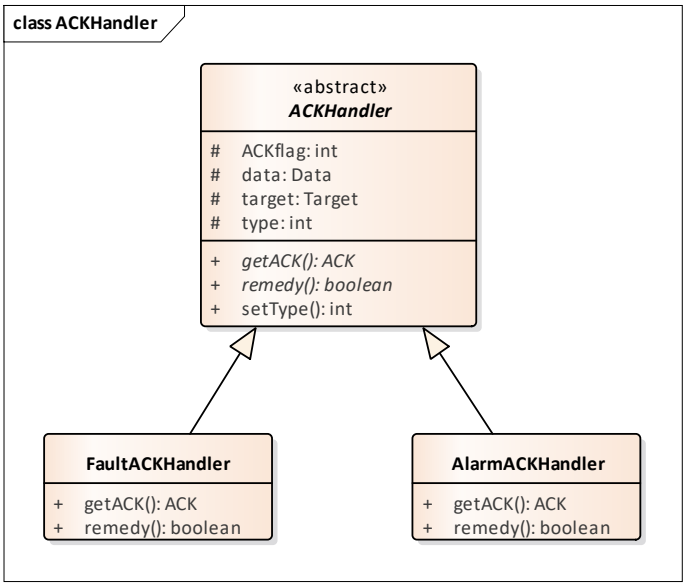


그림 4. ACKHandler 클래스의 클래스 다이어그램

ACKHandler는 사용자에게 보낸 알림 메시지에 대한 사용자의 수신을 확인하는 기능이다. Power Grid 플랫폼

시스템에서 고장이 발생하면 사용자에게 고장에 대한 알림과 정보가 전송된다. 사용자에게 전송된 알림 메시지에 대한 수신 확인과 고장의 처리 여부를 확인하는 기능을 제공한다.

Aspect으로 설계한 Fault Manager 컴포넌트에 포함된 기능은 각각 핵심 기능에 적용(Weaving)되는 시기가 다르다. Power Grid 플랫폼 시스템에서 고장이 발생하면 QualityManager 기능이 적용되며 QualityManager의 수행 결과로 사용자에게 알림이 전송된다. 그 후에 시스템에 발생한 고장의 원인을 분석하여 손상된 데이터를 복구하는 FailureRecovery와 사용자에게 보낸 알림에 대한 수신 확인과 고장 처리를 확인하는 ACKHandler가 사용된다. Fault Manager 컴포넌트에 포함된 기능이 적용되기 전에 setType 메소드를 사용하여 어디서 호출한 것인지 현재 사용자가 갖는 권한은 무엇인지 설정하고 해당 기능을 수행한다.

5. 결 론

관점 지향 설계 기법은 시스템의 핵심 기능에서 부가 기능을 분리하여 설계하는 방법으로 소프트웨어의 규모와 복잡도가 증가하면서 발생하는 객체 지향 설계의 단점을 보완할 수 있다. 이는 소프트웨어의 품질을 높이고 유지 보수 비용을 줄이는 등의 다양한 장점이 있다.

본 논문에서는 소프트웨어 설계에 사용되는 대표적인 설계 기법의 특징과 구조, 장점 및 단점에 대해 분석하고, 그 중에 관점 지향 소프트웨어 설계 기법을 적용하여 Power Grid 플랫폼 소프트웨어를 설계했다. 또한 Aspect으로 설계한 Power Grid 플랫폼 소프트웨어의 고장 관리 컴포넌트를 예로 들어 설계 과정 및 적용 과정에 대해 기술했다.

이와 같이 개발하려는 소프트웨어의 특징과 요구 사항을 정확히 파악하여 그에 맞는 설계를 한다면, 고품질의 소프트웨어를 개발할 수 있을 것으로 예상된다.

참 고 문 헌

[1] Rajib Mall, “Fundamentals of Software Engineering (4th Edition).”, PHI Learning Pvt. Ltd., 223-275, 335-372, 2014.

[2] Navasa, Amparo, et al., “Aspect Oriented Software Architecture: a Structural Perspective.”, Workshop on Early Aspects, AOSD.Vol.2, 2002.

[3] Zhao, Liping, Elizabeth Kendall, “Role modelling for component design.”, System Sciences, 2000. Proceeding of the 33rd Annual Hawaii International Conference on IEEE, 2000.

맥락 데이터 기반 모바일 사용자 인증을 위한 통계적 접근 방안

남상진*, 강승현*, 김순태*, 고덕윤**, 유용환**

A Statistical Approach for Mobile User Authentication based on Context Data

Sang-Jin Nam*, Seung-Hyun Kang*, Sun-Tae Kim*, Deok-Yoon Ko**, Young-Hwan Yoo**

요 약

본 연구에서는 모바일 환경에서의 사용자 인증을 위해 사용자의 상태도들 간의 동치 여부를 검증하고자 한다. 통계적 수치의 집계 및 동치 판정은 사용자가 별로도 제어하는 일 없이 자동으로 진행되며 만약 두 개의 상태도가 동치라고 판정된다면, 요청한 사용자 인증을 허가한다. 이러한 인증 과정은 기존 인증 장치와 비교하여 사용자가 편리하게 인증을 요청할 수 있으며 인증에 실패하는 경우에 대해서만 다른 인증 방법을 제시할 수 있기 때문에 사용성이 증가한다.

1. 서 론

모바일 기기는 현대 사회인에게 필수적인 기기가 되었으며, 이 중에서 스마트폰은 인터넷, 쇼핑, 게임, 앱 등으로 인하여 사용자들에게 가장 널리 사용되는 모바일 기기가 되었다[4,5]. 또한 스마트폰의 대중화와 함께 여러 앱의 사용자 인증에 대한 요구도 증가하기 시작하였다[3]. 하지만, 현재 사용되는 앱의 사용자 인증 방법은 사용자의 기억력에 의존하고 있거나, 일반적인 앱에 적용하는 것에 어려움이 있고 복잡한 인증 절차, 또는 별도의 장치가 필요하다는 데에 그 한계가 있다[1,2,6]. 본 연구에서는 사용자의 맥락 데이터를 활용하여 통계적으로 모바일 사용자의 인증을 수행하는 방안을 제안한다. 사용자의 맥락 데이터는 사용자의 모바일에 축적되는 과거 데이터를 의미하며 이는 CDR(Call Detail Record)로 불리는 통화 기록과 위치 기록 등을 일컫는다. 본 연구에서는 이 맥락 데이터 중 CDR과 위치 기록을 사용하여 인증 여부를 판정하였으며 이를 검증하기 위한 시스템을 구축하여 약 77%의 정확도를 달성하였다.

2. 관련 연구

Elaine Shi와 Markus Jakobsson은 SMS, GPS, Browser Usage 등 여러 문맥 데이터를 기반으로 한 사용자 인증 기법을 다뤘다. 이 연구에서는 데이터를 수집할 때마다 기존 사용자의 데이터와 유사한지 조건부확률을 계산하고 통합하여 인증여부를 확인한다[7,8]. Eiji Hayashi의 연구에서는 문맥 데이터를 active factor와 passive factor로 구분했다. passive factor의 경우는 사용자의 조작없이 수집되는 데이터로 위치 데이터가 이에 해당하고, active factor는 사용자가 기기를 사용하여 발생하는 데이터이다. 이렇게 구분된 데이터에 대해서 각 factor별로 확률적 검증을 한다[10]. 앞선 두 연구에서는 각 문맥 데이터 종류에 대한 조건부확률을 곱하여 얻은 인증 점수를 제시한다. 본 연구에서는 데이터에 대한 분산 및 자유도를 고려하며 기존 방법보다 검정력이 더 높은 T-test[9]를 기반으로 얻은 인증점수를 제시한다.

3. 맥락 데이터 기반 모바일 사용자 인증의 통계적 접근 방안

본 접근 방법은 두 단계로 이루어져 있다. 맥락 데이터 집계 단계에서는 사용자의 맥락 데이터를 수집하고 이를 집계하며, 사용자 인증 단계는 사용자가 앱을 통하여 인증은 요청하면, 사용자의 최근 기록(Request Context Data)과 과거 기록(Base Context Data)을 통계적으로 검증한다. 맥락 데이터는 사용자가 평소에 스마트폰을 소지하고 다니면서 수집되는 기기의 데이터로 본 연구에서는 통화 기록과 위치 기록을 맥락 데이터 수집의 대상으로 한다. 맥락 데이터는 현 시점을 기준으로 과거의 년, 월, 일 시 기준으로 각 요일 단위로 집계하여 이를 기초 맥락 데이터(Base Context Data)로 관리한다. 위치(Location) 데이터는 위치별 각 빈도를 집계한 것이고 CDR 데이터는 통화 대상별 빈도를 집계한 것이며 각 데이터는 1시간 단위로 갱신된다. 기초 맥락 데이터는 다음과 같이 정리할 수 있다. 이 식에서 CDR은 특정인(x)의 특정 1시간(t)의 통화 빈도를 의미하며, Loc는 특정 장소(z)의 특정 1시간(t)의 머무르는 시간을 측정한 빈도를 의미한다.

$$\begin{aligned}\text{ContextDataBase} = \{(&\text{CDR}_x, \text{Loc}_z)_t \mid \\ &x \in \text{All Incoming/Outgoing Contacts}, \\ &z \in \text{All Zones}, \\ &t \in \{1h, 2h..24h, 2d, .. 365d\}\}\end{aligned}$$

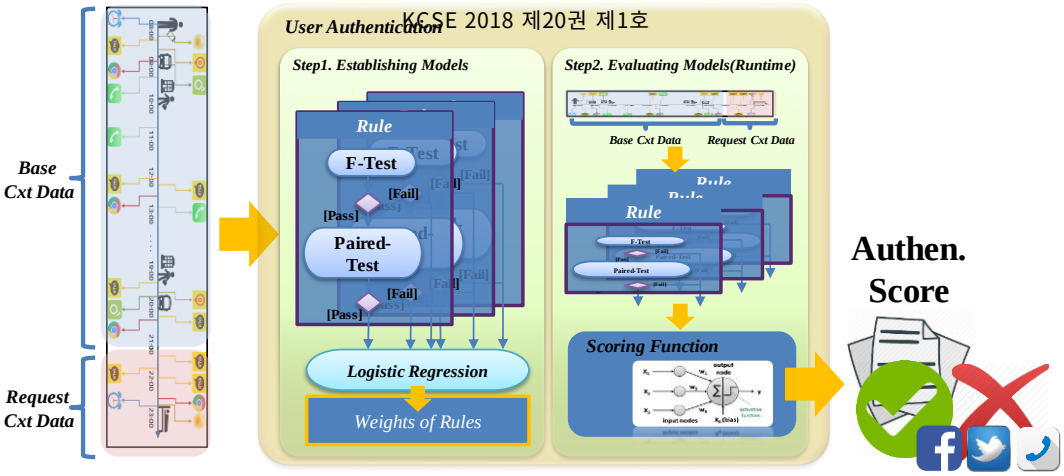


그림 1. 통계적 사용자 인증 방법

모델 수립 과정의 입력은 트레이닝 데이터를 기초 맥락 데이터와 요청 맥락 데이터로 분리한 데이터이며 이것을 이용하여 가설검정을 기반한 검증 규칙을 수행한다. 각 규칙은 서로 다른 기간, 종류의 비교 데이터를 가지며, 이는 모두 맥락 데이터 베이스에서 추출이 된다. 기초 맥락 데이터는 요청 맥락 데이터를 제외한 과거의 데이터이다. 각 규칙은 가장 먼저 등분산성을 F-test[9]로 검정한다. 검정이 기각되는 경우는 두 데이터의 분산이 다르다는 의미이므로, 각 해당 규칙은 바로 0(false)이 반환된다. 분산이 다른 것은 두 데이터를 통계적으로 유의한 비교를 하기 어렵고, 이것을 통해 두 데이터가 다른 모양을 가지고 있다고 판단할 수 있다고 가정한다. 만약 F-test가 통과하면, Paired T-test를 수행하여 통계적으로 유의미하게 차이가 있는지를 확인한다. 규칙은 최종적으로 Boolean값을 출력한다. 각 규칙의 결과는 사용자 인증에 대해 점수로 반환하기 위해 로지스틱 회귀(Logistic Regression)[11]를 활용하여 최종적으로 각 규칙의 적정 가중치를 산출한다. 평가 과정은 그림 1의 Step 2에서와같이 사용자가 앱을 통하여 인증을 요청하는 경우에 실행되며, 별도의 입력 없이 맥락 데이터베이스의 기초 맥락 데이터와 요청 맥락 데이터를 기반으로 수행된다. 1단계의 가중치를 기반으로 다음 함수를 통하여 그 점수를 계산하게 된다.

$$Authen.Score = Sigmoid(\sum_{i=0}^{n} w_i Rule_i)$$

Sigmoid는 x=0을 중심으로 출력(y)이 0~1 사이의 값이며, 본 연구의 적절한 정확도를 제공하기에 차용되었다.

4. 검증

본 연구에서 CDR 데이터와 위치 데이터는 모두 MIT Reality Mining[12]의 데이터를 사용하였다. 이 중에서 85명의 데이터를 사용하였으며 도난 상태의 기기를 표현하는 데이터를 필요에 의해 타인의 데이터를 이어붙인 85개의 데이터를 생성하였다.

표1. 사용 데이터

	Valid Data	Invalid Data
Training Set	50	50
Evaluation Set	50(35+15)	50(35+15)
Total	100(85)	100(85)

실험 과정의 첫 번째로 CDR 데이터와 위치 데이터에 대해서 각 규칙은 위치별 방문 횟수, 대상별 통화 횟수만을 사용하기 때문에 시간별 통화 데이터를 (시간-통화대상), (시간-위치)로 정리하였다. 두 번째로 기초 맥락 데이터베이스를 만들기 위해 사용자 별로 한 시간 단위의 위치, 통화 대상의 빈도를 계산하여 저장한다. 세 번째로 기초 문맥 데이터 베이스에 저장된 데이터를 대상으로 각 규칙을 실행한다. 결과적으로 총 170명 분량의 데이터 셋이 만들어진다. 네 번째로 전 단계에서 얻은 데이터들에 대해서 유효한 사용자 50명과 무효한(도난 데이터를 갖은) 사용자 50명의 데이터를 훈련에 사용하였고 결과적으로 각 규칙에 대한 가중치를 얻을 수 있었다. 마지막으로 구해진 가중치와 각 규칙의 결과를 사용하여 인증 점수를 구한다. 인증 점수를 평가하기 위해서 인증 점수에 대한 임의의 Threshold=0.90을 가정하여 평가하였다.

표2. Threshold = 0.90

	Positive	Negative	
True	23	27	50
False	7	43	50
	30	70	

Precision	0.766667
Recall	0.348458
Accuracy	0.5
F-measure	0.479167

5. 결론

본 연구에서는 F-test 와 Paired T-test 를 통한 사용자 인증 방법에 대해서 제안하였다. 100 개의 문맥 데이터를 사용하여 사용자 인증을 수행하여 인증 여부를 분석한 결과 정확성이 평균 0.7667 이 나오는 것을 확인할 수 있었다. 따라서 문맥 데이터 기반 사용자 인증 방법이 기존의 사용자가 아닌 다른 사람이 인증을 요청했을 경우 두 사람을 잘 구분할 수 있다고 판단한다.

Reference

[1] Herley C. 2009. So long, and no thanks for the externalities: the rational rejection of security advice by users. In *Proc. Of SACMAT*

[2] Hulsebosch J. R., Salden H. A., Bargh S. M., Ebben P. W. G, and Reitsma J. 2005. Context sensitive access control. In *Proc. Of SACMAT*

[3] C. Nachenberg, 2011 "A window into mobile device security: Examining the security approaches employ. *SSR*

[4] L. Whitney. 2010 Smartphones to dominate PCs in Gartner forecast. *CNET Business Tech News*

[5] L. Rainie and J. Anderson. 2008. The Future of the Internet III. *Pew Internet Project*

[6] Feng Zhang, Aron Kondoro, Sead Muftic. 2012. Location-based Authentication and Authorization Using SmartPhone. *TrustCom*

[7] Elaine Shi, Yuan N., Markus J. and Richard C. 2010. Implicit Authentication through Learning User Behavior. *ISC*

[8] Markus J., Elaine Shi, Philippe G., Richard C. 2009. Implicit Authentication for Mobile Devices. *USENIX*

[9] 안재형. *R을 이용한 누구나 하는 통계분석. 한나래 아카데미*. 서울. 2011

[10] Eiji H., Saivik Das, Shahriyar A., Jason Hong and Ian Oakley. 2013. CASA:Context-Aware Scalable Authentication. *SOUP*

[11] Gary King, Langche Zeng. 2017. Logistic Regression in Rare Events Data. *Political Analysis.*

[12] Mit Human Dynamics Lab. *"Mit Reality Mining Dataset - Reality Commons"*. http://realitycommons.media.mit.edu/realitymining.html. 2017-12-22.

딥러닝 모델 기반의 버그 담당자 자동 배정 시스템 성능 측정

박해성 ○ 김수빈 이찬근 채병훈
중앙대학교 컴퓨터공학부
hea9549@gmail.com, lauren026@naver.com, cglee@cau.ac.kr, cbh1203@naver.com

Measuring Performance of Automatic Bug Triagers on Deep Learning Models

Hae-Sung Park○, Su-Bin Kim, Chan-Gun Lee, Byung-Hoon Chae
chool of Computer Science and Engineering, Chung-Ang University

요 약

버그 담당자 자동 배정 시스템은 버그 리포트를 자연어처리와 기계학습 알고리즘을 이용한 분석을 통해 적절한 개발자를 예측하는 시스템이다. 이 시스템은 초기 버그 해결을 위한 할당 시간을 줄여 효율적인 버그 처리를 돕는다. 본 논문은 버그 담당자 자동 배정 시스템의 자연어처리 단계에서 워드임베딩 기법의 하나인 Word2Vec의 다양한 딥러닝 기법 적용에 따른 버그 자동 배정 시스템의 성능향상에 관해 연구한다. 단순히 문서 내에서 쓰이는 단어의 빈도수만을 고려하는 기존의 자연어 처리 기법과는 달리 워드임베딩 기법은 단어의 분산 이론에 의한 벡터화를 제공하여 더욱 우수한 분류 성능을 보인다. CNN과 RNN을 이용한 딥러닝 모델과의 조합을 통해서 시스템의 정확도를 측정한 후 실험 결과를 비교 분석한다..

1. 서 론

오픈소스 소프트웨어 프로젝트는 불특정 다수의 개발자가 참여 하므로 버그 저장소의 사용을 통해서 프로젝트의 질을 높일 수 있다. 이런 대규모의 프로젝트에서는 하루에 수백 건 이상의 버그가 보고될 수 있으며 이를 관리하는 것이 소프트웨어의 성능과 품질을 유지하기 위해 필수적으로 요구된다.

버그 저장소에서 보고된 버그를 해결하기 위해 적합한 담당자를 찾는 과정을 버그 선별(bug triage)이라고 한다. 그런데 수십 명 이상의 개발자가 참여하는 오픈소스 프로젝트에서는 보고된 버그를 담당할 가장 적합한 개발자를 찾는 작업이 전문성을 요구하는 어려운 일이다. 따라서 하루에 수백 건 가까이 쏟아지는 버그 리포트를 사람이 분석하는 것은 많은 인적, 시간적 비용이 필요하게 된다.

기존의 버그 담당자 자동 배정 시스템 연구에서는 비교적 단순한 단어 빈도수에 기반을 둔 자연어 처리 기법들을 이용했다. 이 분야의 최근 연구들은 신경망 기반의 워드임베딩(word embedding) 기법과 같은 보다 고도화된 임베딩 기법을 활발히 적용하기 시작하고 있다.

딥 러닝은 이미지 처리, 음성 인식 등 여러 분야에서 기존의 기계학습 기법들보다 좋은 성능을 보이며, 자연어 문서 분류에서도 다양한 시도가 진행 중이다. 버그 담당자 자동 배정 관련 연구에서 딥러닝을 이용한 사례가 있으며[1], 본 연구에서는 시퀀스 정보를 이용한 RNN층과의 조합에 따른 성능 변화를 확인한다.

따라서 본 연구에서 우리는 Word2Vec기법과 CNN(Convolutional Neural Network) 및 RNN(Recurrent Neural Networks)을 이용하여 학습 모델을 구성하고 다양한 적용 방안에 따른 버그 담당자 배정 시스템의 성능 향상을 연구한다.

본 논문의 실험과정을 간단히 설명하면, 먼저 Word2Vec을 이용한 워드 임베딩을 통해 버그 리포트의 모든 단어를 각각 벡터화한다. 단어 벡터들을 이용하여 CNN 및 RNN의 버그 담당자 분류기 모델을 구성한다. 버그 담당자 분류기 모델을 이용하여 버그 리포트의 적절한 담당자를 분류한 뒤 실제 Assigned 된 버그 리포트 결과와 비교하여 그 정확도를 분석한다.

본 논문의 구성은 다음과 같다. 2절에서는 버그 담당자 배정 문제에 관해 연구에서 쓰인 주요 기술을 소개하고, 3절에서는 데이터와 텍스트의 전처리 및 분류기 모델 구성 방법과 같은 실험 세팅과 실험목표를 소개한다. 4절에서는 본 연구의 평가 방법과 실험 결과를 다루고, 5절에서는 본 연구의 결론과 향후 연구 방향을 제시한다.

2. 배경지식

2.1 버그 추적 시스템

버그 추적 시스템은 소프트웨어의 개발 과정에서 일어나는 버그들을 관리해주는 응용 소프트웨어이다. 성공적인 소프트웨어 프로젝트를 위해서 많은 오픈소스

프로젝트는 물론, 기업 내부의 프로젝트에서도 사용된다. 개발자들은 버그 추적 시스템을 이용하여 소프트웨어 개발 중 발생하는 버그를 정해진 형식으로 보고한다. 버그 리포트에는 보고자의 이름, 버그에 대한 설명, 버그를 발생시키는 환경, 버그의 심각도 등을 입력해야 한다. 개발 관리자는 보고된 버그들의 상태를 추적하여 소프트웨어 개발 프로젝트를 관리할 수 있다[2].

버그 추적 시스템은 소프트웨어 개발 관리를 위해서 버그의 생애 주기를 제공한다. 누군가가 버그 추적 시스템을 이용하여 버그를 보고하면, 버그 저장소를 OPEN[2] 하였다고 한다. 저장소에 보고된 버그들은 개발자에게 할당되어 ASSIGNED 상태가 된다. 또한, 버그가 관리되고 있다면 RESOLVED 되었다고 한다. RESOLVED 는 다음의 상태변화를 통하여 이루어진다. 해당 버그를 해결하여 코드에 변화가 있다면 FIXED, 이전의 버그들과 중복된다면 DUPLICATE, 실제로 버그로 판정될 수 없거나 해결할 수 없는 버그라면 INVALID 상태가 되어 RESOLVED 된다. 이후에 버그가 테스트 단계에서 통과하면 VERIFIED 상태가 되지만 통과하지 못한다면 REOPENED 상태가 되어 다시 개발자들에게 할당된다. 테스트 단계가 끝나면 버그는 CLOSED 상태가 된다[2, 3].

2.2 워드 임베딩

워드 임베딩은 분산 이론(Distributional Hypothesis)에 따라 이루어지기 때문에 단어 간의 관계 속에서 각 단어의 출현 빈도를 고려하여 단어마다 유의미한 벡터를 부여한다. 즉, 단어의 의미를 수치로 나타내는 것을 의미한다. Mikolov 는 워드 임베딩의 학습 방법으로 Word2Vec 모델을 제안하였다. Word2Vec 은 기존의 방법과 비교하여 학습 속도와 성능을 비약적으로 끌어올렸다. Word2Vec 모델은 주변 단어로부터 중심단어를 예측하는 CBOW(Continuous Bag Of Words) 방식과 중심 단어로부터 주변 단어를 예측하는 Skip-gram 방식이 있다[3]. 본 연구에서는 Skip-gram 을 사용하였다.

2.3 CNN

CNN(Convolutional Neural Networks)은 인공 신경망의 한 종류로서 다음 그림 1 은 일반적인 CNN 의 구조이다.

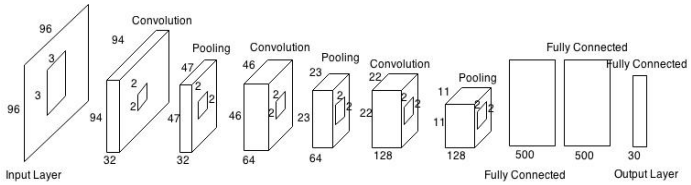


그림 1 CNN의 일반적인 구조

그림 1 과 같이 CNN 은 Convolution 층(Convolutional Layer), 풀링층(Pooling Layer), 그리고 FC 층(Fully-Connected Layer)의 핵심 구조로 이루어져 있다. Convolution 층은 필터를 이용하여 Convolution 연산을 수행한다. 이 과정을 통해 의미있는 특징들을 추출한다. Convolution 과정으로 의미있는 맵을 얻은 후, 풀링 층에서 풀링 과정을 거쳐 필터와 stride 의 갯수에 따라 데이터의 크기가 결정된다. 풀링 과정에서는 필터 내에서 가장 큰 값을 추출하는 Max Pooling 이 주로 사용된다. FC 층은 마지막에 적용되며 Convolution 층과 풀링 층에서 나온 특징을 이용하여 분류를 한다.

CNN 이 텍스트 분류(Text Classification)에서 뛰어나다고 알려져있다[5]. 더 나아가 Yoav Goldberg 는 CNN 이 문서 분류(Document Classification)에서 효과적이라고 하였다[6]. 입력 순서 내에서, 문장의 지역 정보를 보존함으로써 단어/표현의 등장순서를 학습에 반영할 수 있기 때문이다. 이에 본 연구에서는 CNN 을 이용하였다.

2.4 RNN

CNN과 달리, RNN(Recurrent Neural Network)은 입출력이 모두 시퀀스인 인공 신경망이다. 따라서 자연어를 처리하는데 쉬우며 그 범위가 점차 확대되고 있다. 다음 그림2는 RNN의 기본적인 구조를 나타낸다.

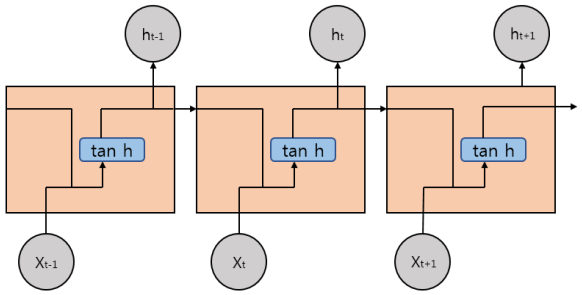


그림 2 RNN의 기본 구조

데이터의 시퀀스가 길어질수록 vanishing/exploding gradient 로 인해 장시간에 걸친 데이터 의존도가 있는 문제를 풀기가 어려워졌다.

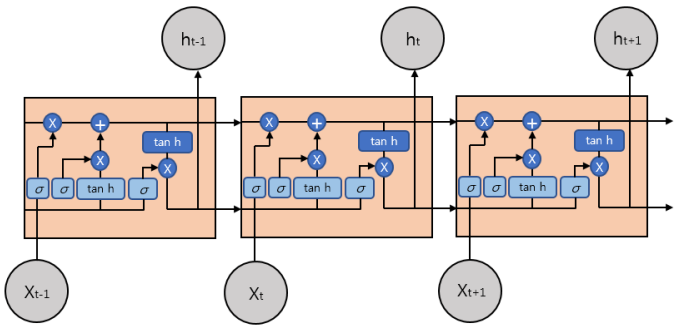


그림 3 LSTM 구조

LSTM 은 RNN 의 장기 의존성 문제를 보완하기 위해 고안되었다. 그림 3 은 LSTM 의 구조이다. LSTM 은 장기간 메모리 역할을 하는 cell state 와 3 가지 게이트(Input Gate, Output Gate, Forget Gate)로 구성된다. 게이트를 통하여 이전의 정보가 현재의 정보에 미치는 영향 등을 조절할 수 있어 장기 의존성 문제를 해결할 수 있었고, 현재 많은 분야에서 활용되고 있다. LSTM 은 문서 분류에서도 우수한 성능을 보인다[7]. 이에 본 연구에서는 LSTM 을 이용하였다.

3. 실험준비

3.1 실험목표

본 논문에서는 다음과 같은 내용을 확인하기위해 실험을 준비하고 진행하였다.

- 기존 분류기법(SVM,DT,NB)보다 딥러닝을 적용 한 것이 성능이 우수한가?
- 딥러닝의 모델 CNN 과 RNN 중 어떤 모델이 성능이 더 우수한가?
- CNN 과 RNN 을 같이 사용하였을 때 더 좋은 성능을 보이는가?
- CNN 및 RNN 의 파라미터 설정에 따라 성능이 어떻게 변하는가?

3.2 데이터 및 전처리

이 실험에서 사용한 데이터는 오픈소스 프로젝트인 Eclipse JDT 의 버그 리포트를 수집하여서 사용하였다. 수집한 데이터는 2013 년 2 월부터 2015 년 2 월의 기간의 JDT 프로젝트에서 해결된 상태 (FIXED)의 리포트들이다. 이슈 관리 시스템인 버그질라에서 수집된 리포트는 1,452 개이며 해결된 상태가 아니거나 담당자 정보가 없는 리포트는 학습에 사용할 수 없으므로 수집하지 않았다. 또한, 실험을 통해 10 개 미만의 버그를 해결한 개발자와 해당 보고서를 제외하는 것이 효과적인 방법이란 것을 알아냈다. 따라서 1,452 개 리포트에서 1,340 개의 리포트를 선정하였으며, 담당자는 16 명을 선정하였다.

워드 임베딩 모델의 학습 데이터는 2013 년 10 월부터 2015 년 6 월까지의 모든 이클립스 프로젝트 약 5 만 개의 리포트를 사용하였다.

리포트 내에서 데이터로 쓰인 부분은 리포트 설명 부분과 담당자 부분이다. 담당자에 특정 개발자가 명시되어 있으면, 그 개발자를 해당 리포트의 담당자로 하였다. 반면 담당자에 특정 개발자가 명시되어 있지 않을 경우, 버그 리포트 상태변화 기록에서 리포트의 상태를 FIXED 로 바꾼 개발자를 해당 리포트의 담당자로 하였다.

워드임베딩의 성능을 높이기 위해서 각 단어의 희소성을 완화하였다. 워드 임베딩 전 단계에서 특수문자와 불용어(stop words)를 모두 제거 하였다. 그리고 첨부된 소스코드의 카멜 케이스 내의 분리 가능한 단어를 독립적인 단어로 처리했다. 마지막으로 같은 단어에 대해 대소문자 구분을 피하고자 모든 단어를 소문자화 하였다

3.3 CNN 모델 구성

하나의 버그 리포트에 포함된 단어들의 벡터들은 Convolution 필터와 Max-Pooling층을 걸친 뒤 FC층과 SoftMax를 이용하여 결과가 나오게 된다. 결괏값은 각 개발자와 대응하게 되는데, 따라서 결괏값에서 가장 큰 값을 보이는 개발자가 모델의 추천 결과이다

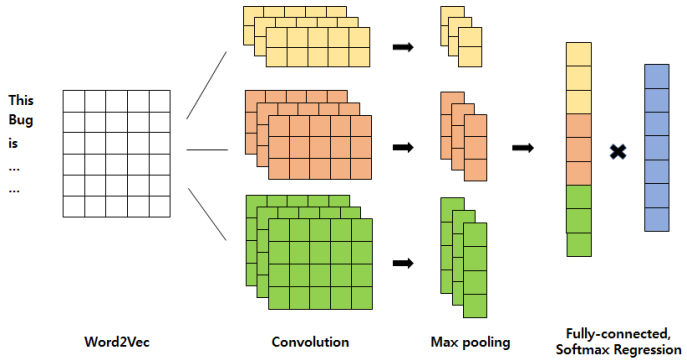


그림 4 CNN 기반 학습 모델 구조

그림 4 는 본 연구에서 사용된 CNN 모델의 구성이다. Word2Vec 을 이용하여 버그 리포트 내의 단어들을 벡터화시킨 후에 리포트 중 가장 단어 수가 많은 리포트(M 개)를 기준으로 단어 수가 적은 리포트들은 0 벡터를 추가한다. 이 과정을 통해 모든 리포트는 (단어벡터 차원 수, M)인 행렬로 변환된다. 변환된 데이터는 Convolution 층에서 필터를 학습하는 데 사용하며, 필터의 크기는 (단어벡터 차원 수, s) 로 학습한다. 본 연구에서 s 는 각각 3, 4, 5 로 수행하였고, 같은 크기의 필터를 512 개씩 사용하였다. 따라서 이 과정을 지나면 (1, M-s+1) 크기의 벡터가 512 개가 생성되는데, 이를 max-pooling 을 이용하여 (1, 1) 크기의 벡터를 512 개를 얻을 수 있다. s 를 각각 3, 4, 5 로 수행하였으므로 512 개의 벡터가 3 쌍 생성되고 이 벡터들을 병합하였다. 그 후 softmax regression 을 이용해 각각 개발자가 해당 리포트에 적합할 확률을 얻은 후 가장 높은 확률의 개발자를 추천 개발자로 선정하였다.

3.4 RNN 모델 구성

하나의 버그 리포트에 포함된 단어들의 벡터들을 모두 RNN 모델에 넣을 경우, 순서길이가 너무 길어져 성능이 떨어지는 것을 실험을 통해 확인하였다. 입력값의 크기를 줄이기 위해 단어 벡터들을 일정 크기로 합하여 차원을 축소하였다.

축소된 행렬을 RNN 층에 차례대로 입력하여 RNN 층을 거친 뒤 FC 층과 SoftMax 를 이용하여

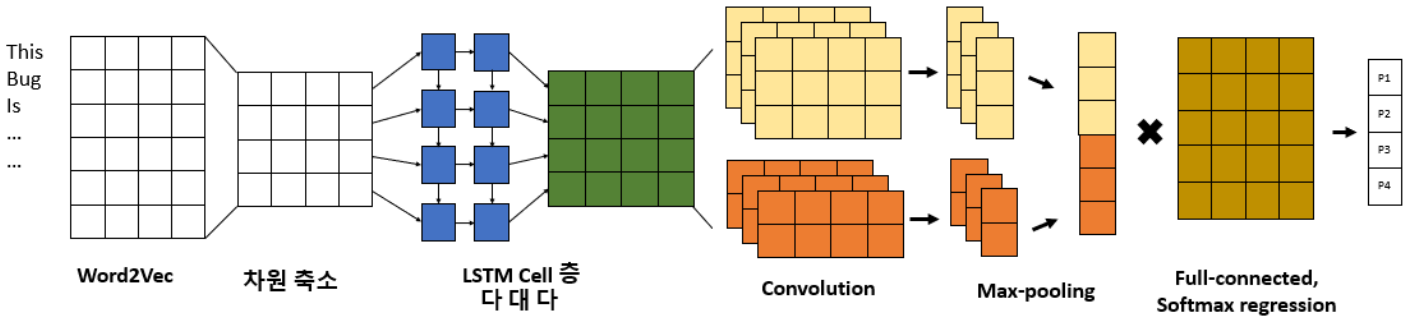


그림 5 RNN, CNN 합 학습 모델 구조

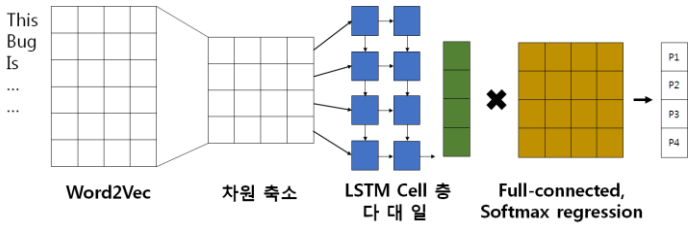


그림 6 RNN 기반 학습 모델 구조

결과가 나오게 된다. 결괏값은 각 개발자와 대응하게 되는데, 따라서 결괏값에서 가장 큰 값을 보이는 개발자가 모델의 추천 결과이다.

그림 6 은 본 연구에서 사용된 RNN 모델의 구성이다. Word2Vec 을 이용하여 버그 리포트 내의 단어들을 벡터화시킨 후에 리포트 중 가장 단어 수가 많은 리포트(M 개)를 기준으로 단어 수가 적은 리포트들은 0 벡터를 추가한다. 이 과정을 통해 모든 리포트는 (단어벡터 차원 수, M)인 행렬로 변환된다. 변환된 행렬을 차원축소를 위해 본 연구에서는 벡터 합, 벡터 붙이기 방법을 이용하였다. 축소한 벡터를 LSTM 셀에 입력하여 다 대 일 층을 구성하여 마지막 셀의 출력을 이용해 FC 층과 SoftMax 를 통해 해당 버그 리포트의 개발자를 분류하게 하였다.

3.5 RNN , CNN 합 모델 구성

그림5는 RNN과 CNN의 합 모델의 구성이다. RNN + CNN 모델은 RNN층의 순차 입력의 이점을 이용해 나온 결과를 CNN 층에 이용해 보고자 실험 항목에 추가하였다. 위 3.2와 3.3절에서 나온 모델을 붙인 형태이며, RNN층에서 CNN 층으로 입력은 다 대 일 모델이 아닌 다 대 다 모델로 변경하여 입력하였다.

4. 실험 결과 및 분석

4.1 평가방법

성능을 평가하기 위해 정확도(Accuracy) 방법을 이용한다. 정확도는 평가 데이터에서 정답을 맞힌 것의 비율이며 수식은 다음과 같다.

$$\text{정확도} = \frac{\text{정답 수}}{\text{테스트 데이터 수}}$$

CNN, RNN, RNN+CNN 모델의 성능 비교를 위해 기계학습 기반의 버그 담당자 자동 배정 연구에서 주로 사용되고 있는[8, 9] SVM(Support Vector Machine), NB(Naive Bayes), DT(Decision Tree) 알고리즘과 비교하여 실험을 진행하였다.

4.2 실험결과

표 1 버그 담당자 자동 배정 성능 측정 결과

기계학습 알고리즘	정확도
RNN	36.5%
RNN+CNN	40.6%
CNN	45.9%
SVM	31.2%
NB	20.1%
DT	21.9%

표 1 은 기존 기계학습 모델과 이 논문에서 제시하는 기계학습 모델을 이용하여 성능을 측정한 결과이다. 실험결과 RNN, RNN+CNN, CNN 모델이 기존 기계학습에서 가장 성능이 좋은 SVM 대비 각각 5.3%, 9.4% 14.7%의 정확도 향상을 보였다. CNN 모델을 이용한 것이 가장 좋은 성능을 보이는 것을 확인할 수 있다. 따라서 3.1 절 실험 목표에 “기존 분류기법(SVM, DT, NB)보다 딥러닝을 적용 한 것이 성능이 우수한가?” 에 대해서 딥러닝을 적용 한 것이 성능이 더 우수하다 할 수 있다. “딥러닝의 모델 CNN 과 RNN 중 어떤 모델이 성능이 더 우수한가?”에 대해서는 CNN 모델이 RNN 모델보다 성능이 더 우수하다 할 수 있다. 또한 “CNN 과 RNN 을 같이 사용하였을 때 더 좋은 성능을 보이는가?” 에 대해서는 RNN 과 CNN 의 모델 조합은 RNN 단독모델보다는 더 좋은 성능을 보이나 CNN 의 성능에는 못 미치는 것으로 확인된다.

4.3 실험분석

이 절에서는 3.1 절 실험목표 “CNN 및 RNN 의 파라미터 설정에 따라 성능이 어떻게 변하는가?”에 대한 실험 분석내용이다.

4.3.1 단어 벡터의 차원 및 CNN 필터 개수에 따른 CNN 모델 성능 비교

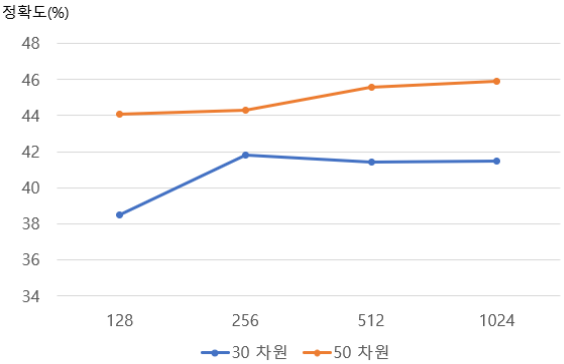


그림 7 단어 벡터 차원, 필터 개수에 따른 성능 변화

CNN 모델은 Convolution 층에서 각 필터의 개수가 분류 성능에 영향을 줄 수 있다. 또한, 단어를 벡터로 나타내는 과정에서 변환되는 벡터의 길이가 긴 경우 더 많은 특성을 반영할 수 있다. 따라서 필터 개수와 단어벡터의 차원 수를 조절함에 따른 성능을 비교한 결과로 정확도를 측정하였다. 그림7의 X축은 Convolution 필터의 개수를 나타내고 세로축은 정확도를 나타낸다. 각 선은 30차원과 50차원을 나타낸다. 실험 결과 필터의 개수가 증가함에 따라 성능도 높아지는 것을 확인할 수 있는데, 512개보다 많으면 더는 향상되지 않는 것을 확인할 수 있다. 또한, 단어의 차원이 클수록 더 많은 특징을 담아 성능이 향상되는 것을 볼 수 있는데 50차원 이상의 경우 큰 향상이 없는 것으로 보인다.

4.3.2 차원 축소 방법 및 층 개수에 따른 RNN 모델 성능 비교

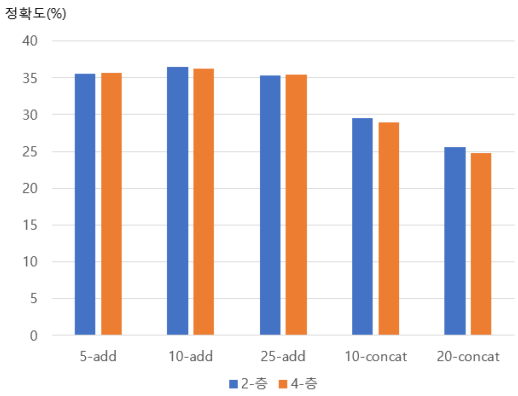


그림 8 차원 축소 방법 및 층 개수에 따른 성능 변화

그림 8은 RNN층의 시퀀스 길이를 조절하기 위해 이용한 차원 축소 방법과 RNN 층의 개수에 따른 성능을 비교한 것이다. Add 접미사가 붙은 항목은 일정 개수 단어 벡터를 합산해서 차원을 축소한 것이고, Concat 접미사가 붙은 항목은 일정 개수의 단어를 이어 붙여 차원 축소를 한 것이다. 차원 축소 방법으로는 합산 방법이 이어 붙이는 방법보다 성능이 더 좋은 것을 볼 수 있으며, 층의 개수에 따라 큰 성능의 차이는 없는 것을 볼 수 있다.

5. 결론 및 향후연구

버그 담당자 자동 배정과 관련된 연구는 소프트웨어 프로젝트 크기가 점점 커짐과 소프트웨어 아키텍처의 중요성이 주목받으면서 중요성이 높아지고 있다. 본 연구에서는 딥러닝 기법을 버그 담당자 자동 배정 시스템에 적용한 결과, 약 5.3%~14.7% 정도 기존의 기법들에 비해서 나은 성능을 보이는 것을 확인하였다. RNN 의 경우 기존 기법들에 비해 나은 성능을 보이나, CNN 을 적용 시킨 결과가 더 좋은 성능을 보이는 것을 확인할 수 있다.

후속 연구에서는 버그 리포트에서 단순 리포트 내용뿐만 아니라 제목, 리포트 항목 등 다양한 요인을 추가하여 버그 담당자 자동 배정 시스템의 성능 향상을 목표로 한다. 또한, 담당자 뿐만 아닌 버그 리포트의 예상 해결 기간 및 해결 담당자 후보를 추천하는 등 다양한 결과를 낼 수 있는 연구를 목표로 한다.

감사의말

본 연구는 한국연구재단 기초연구사업(과제번호 NRF-2017R1E1A1A01075803, NRF-2017R1D1A1B03029552)의 지원을 받았습니다.

참 고 문 헌

[1] 이선로, 딥 러닝을 이용한 버그 담당자 자동 배정 연구,한국소프트웨어공학학술대회,19 권,1 호,168 쪽,2017

[2] J. Jan'ak. “Issue tracking systems”. Master's thesis, Masaryk University, Brno, The Czech Republic, 2009.

[3] J. Anvik, L. Hiew, and G.C. Murphy. “Who should Fix This Bug?” Proc. of 28th Int'l Conf. Software Eng., pp. 361-370, 2006.

[4] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado & J. Dean. “Distributed representations of words and phrases and their compositionality” Proc. of Advances in neural information processing systems, pp.3111-3119, 2013.

[5] Yoon Kim. “Convolutional Neural Networks for Sentence Classification”
arXiv:1408.5882. 2014
[6] Yoav Goldberg. “A Primer on Neural Network Models for Natural Language Processing”
arXiv:1510.00726. 2015
[7] Duyu Tang, Bing Qin, Ting Liu. “Document Modeling with Gated Recurrent Neural Network for Sentiment Classification”. Harbin Institute of Technology, pp.1422-1432, 2015
[8] John Anvik, Lyndon Hiew and Gail C. Murphy, “Who should fix this bug?”, ICSE '06 Proceedings of the 28th international conference on Software engineering, pp. 361-370, 200
[9] Davor Cubranic, Gail C. Murphy, “Automatic bug triage using text categorization”, SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering, pp. 92-97, 2004

척도를 이용한 소프트웨어의 진화 단계 파악

조 희태°[1], 이 선아[1, 2]
항공우주 및 소프트웨어공학전공, 경상대학교[1]
정보과학과, 경상대학교[2]

cht3205@gnu.ac.kr[1], saleese@gnu.ac.kr[2]

Identifying of The Software Evolution Phase Using Metrics

Heetae Cho°[1], Seonah Lee[1, 2]
Department of Aerospace and Software Engineering, Gyeongsang National University[1]
Department of Informatics, Gyeongsang National University

요약

소프트웨어는 지속적으로 진화하는 반면 기존의 소프트웨어 척도는 특정한 시점에서 소프트웨어의 상태를 파악하기 위해서 사용된다. 그러나 우리는 소프트웨어의 진화 상태를 표현할 수 있는 척도가 무엇인지 파악하기 위해 척도 적용 실험을 수행한다. 이를 위해, 깃허브(GitHub)에서 선정한 소프트웨어 프로젝트 개정이력을 사용한다. 그 중 어떤 척도가 소프트웨어의 진화 단계를 파악하는데 유용한지 실험하기 위해, 프로젝트의 커밋의 빈도, 코드의 라인수, 회전복잡도를 기간 별 수치로 나타내어 비교한다. 또한 종료된 프로젝트와 진행 중인 프로젝트에 소프트웨어의 진화 관점에서 동일한 척도를 적용했을 때 결과의 차이가 존재하는지 논의한다.

1. 서 론

소프트웨어는 개발을 완료하고 운용을 하는 동안에도 사용자의 요구사항, 버그 등의 이유로 지속적인 유지보수이 필요하다. 이러한 유지보수에 의해 소프트웨어가 변화하는 과정을 소프트웨어의 진화라고 한다. 연구[1]는 소프트웨어 진화의 5가지의 법칙을 제시한다. 첫째, 진화는 시스템의 재생산 비용이 유지보수에 드는 비용을 초과할 때까지 지속된다. 둘째, 진화하는 소프트웨어의 복잡도는 지속적으로 증가한다. 셋째, 진화 과정과 피드백의 경로 등의 요소가 결합되어 통계적으로 규칙적인 결과가 산출된다. 네 번째, 소프트웨어의 유지보수가 발생하는 동안 전역 활성 비율이 통계적으로 일정해야 한다. 다섯 번째, 소프트웨어의 유지보수가 발생하는 동안 릴리즈 컨텐츠가 통계적으로 일정해야 한다.

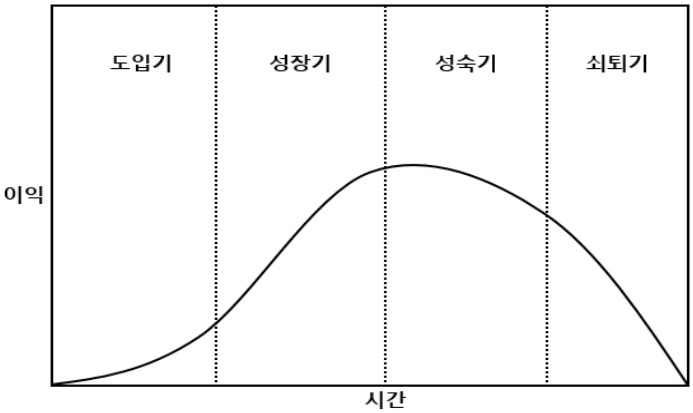
진화하는 소프트웨어는 하나의 제품으로 간주할 수 있으며, 제품 생명 주기론을 적용할 수 있다. 제품 생명 주기론은 모든 제품이 하나의 생물처럼 생명 주기를 가지고 있다는 이론이다. 제품 생명 주기론에서는 제품의 생명 주기를 총 네 가지 단계로 나누었고, 각 단계 마다 고유의 특징을 명시하였다. 먼저 도입기는 제품이 시장에 처음 등장하여 인기를 끌기까지의 시기다. 둘째, 성장기는 제품이 인기를 얻어 이익이 증가하는 시기다. 셋째 성숙기는 제품의 인기가 감소하면서 이익도 감소하는 시기다. 마지막 쇠퇴기는 시장에서 제품의 인기가 없어지며 수요가 거의 없는 시기다.[2]

우리는 소프트웨어의 진화 단계를 파악하기 위한 기준으로, 연구[1]에서 제시한 소프트웨어 진화의 법칙과 연구[2]에서 나타난 제품생명주기의 특징이 적용될 수 있는지, 그리고 이 두가지의 성질과 척도를 이용하여 소프트웨어 진화의 단계를 파악할 수 있는지에 관심을 두고있다.

따라서 우리는 본 논문에서 소프트웨어의 특성을 측정할 수 있는 척도를 활용하여 이러한 관심에 대한 대답을 얻을 수 있는지 시도해 보기로 하였다. 그 일환으로 소프트웨어 개정 이력에서 측정할 수 있는 커밋의 빈도, 코드의 라인 수, 회전복잡도를 소프트웨어가 진화하는 동안의 변화를 측정한다. 또한, 종료된 프로젝트와 지속되고 있는 프로젝트간의 척도 측정 결과를 비교하여 차이점이 있는지 논의한다. 본 연구의 결과와 향후 추가적인 척도를 적용한 연구를 진행하여, 소프트웨어 진화의 단계를 파악하고 각 단계에 부합하는 조치방법의 연구에 공헌 하기를 기대한다.

본 논문의 구성은 다음과 같다. 2절에서는 척도 측정에 대한 가설을 기술한다. 3절에서는 척도에 대해 기술한다. 4절에서는 프로젝트 선정과 척도 측정 방식을 설명한다. 5절에서는 측정된 결과를 보여주고 이러한 결과에 기반하여 가설에 대해 논의한다. 또한 두 개의 서로 다른 프로젝트의 결과를 비교한다. 6절에서는 관련연구에 대하여 간단히 소개한다. 7절에서는 결론을 정리한다.

2. 가 설



[그림 1] 제품수명주기론의 생명주기와 특징

우리는 소프트웨어 진화의 법칙과 제품생명주기론을 기반으로 아래와 같은 세가지 가설을 세웠다.

H1: 커밋의 빈도는 제품생명주기 그래프 [그림1]과 유사한 형태를 포함하고 있을것이다.

H2: 소프트웨어 전체 코드의 라인수는 지속적으로 증가할 것이다.

H3: 소프트웨어 전체 회전복잡도는 증가하거나, 일정하게 유지될것이다.

먼저 H1은 진화하는 소프트웨어를 제품으로 간주했을 때 소프트웨어는 연구[2]의 생명주기 특징을 가진다는 가설이다. [그림1]은 제품의 시간에 따른 이익의 변화에 대한 그래프인데, 이익의 변화는 소비자의 제품에 대한 관심에 비례한다. 커밋의 빈도는 사용자의 프로젝트에 대한 피드백, 즉 관심에 비례한다. 따라서 [그림1]과 커밋의 빈도 그래프는 동류의 그래프이다. 그러므로 커밋의 빈도는 [그림1]과 유사한 형태를 포함하고 있을 것으로 가정한다. 예를 들어 도입기에는 소수의 개발자들이 커밋을 진행하기 때문에 횟수가 많지 않다가, 성장기에는 사용자의 피드백을 반영하기 위해 많은 개발자가 커밋을 진행하기 때문에 횟수가 늘어날 수 있다. 성숙기에서는 지속적인 사용자의 요구를 만족시키기 위하여 지속적인 커밋의 횟수를 유지하다가, 쇠퇴기에는 더 이상 관심이 없는 상태가 되어 커밋의 횟수가 줄어들 것으로 가정하였다.

다음으로, 가설 H2는 연구[1]에서 제시한 진화의 법칙중 두번째인 지속적인 복잡도의 증가와 관련된 가설이다. 코드의 라인수는 가장 쉽게 측정가능한 소프트웨어 복잡도의 척도이며, 지속적인 코드 추가가 발생한다는 측면에서 코드 라인 수는 지속적으로 증가한다고 가정하였다.

가설 H3 또한 연구[1]의 진화하는 소프트웨어는 복잡도가 지속적으로 증가한다는 법칙에 맞춘 가설이다. H3는 회전복잡도인데 증가한다고만 가정하기에는 리팩토링 활동이나 구조화 활동 등을 통해 일정하게 유지될 수도 있지 않겠는가라는 측면에서 증가하거나 일정하게 유지된다고 가정하였다.

3. 척 도

우리는 소프트웨어 진화의 일부를 나타내는 소프트웨어 특성을 측정하기 위한 척도로 커밋의 빈도, 코드의 라인수, 회전복잡도를 활용한다.

3.1 커밋의 빈도

하나의 프로젝트에는 적게는 수십개에서 많게는 수천개, 수만개의 커밋이 존재할 수 있다. 커밋은 코드의 수정뿐만 아니라 여러 목적을 가질 수 있으며 또한 여러 정보를 가지고 있다. 이러한 커밋은 소프트웨어에 대한 관심이 높아서 사용율이 많아져 피드백이 활발할 때 빈번하게 발생할 것으로 예상한다. 따라서 일정한 주기를 정하여 한 주기동안 몇번의 커밋이 발생하였는가를 체크해 보기로 하였다. 본 논문에서는 소프트웨어 진화를 위하여 2차원으로 표기를 하기 위하여 다음과 같이 X축과 Y축을 적용하여 커밋의 빈도 척도를 측정한다.

- X축: 커밋이 발생한 매월
- Y축: 각 달에 발생한 커밋 횟수

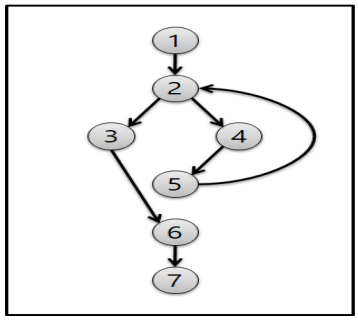
3.2 코드의 라인수

코드의 라인수는 소프트웨어의 크기를 측정하는 방법으로 복잡도를 측정하기 쉽다는 장점이 있는 반면, 프로그래머의 실력에 따라 다를 수 있으며, 복잡도의 단일 척도로는 불충분하다는 단점이 존재한다. 코드의 라인수를 측정하는 방법은 파일의 모든 라인을 세는 방법, 실행되는 라인만을 세는 방법, 주석 라인만을 세는 방법등 연구 목적과 필요에 따라 다양한 방법으로 측정할될 수 있다. 본 논문에서는 다음과 같이 X축과 Y축을 적용하여 코드 라인 수 척도를 측정한다.

- X축: 커밋이 발생한 매월
- Y축: 각 달의 마지막에 발생한 커밋을 기준으로 프로젝트의 모든 소스 파일의 실행 라인 수

3.3 회전복잡도

회전복잡도는 코드에서 나타나는 제어의 복잡성을 기준으로 복잡도를 산출하는 방법이다. 회전복잡도는 [그림2]과 같은 제어흐름 그래프를 화살표의 수에 노드의 수를 빼고 2를 더하여 산출한다. 회전복잡도는 결과가 수치적으로 나오기 때문에 이해하기 편하고,



[그림 2] 제어흐름 그래프

계산 또한 간편하기 때문에 유용하게 쓰인다. [그림2]를 예로 들면 7-7+2 이므로 복잡도는 2이다. 이것은 즉, 해당 프로그램이 종료되는 2가지 경로가 존재한다는 의미이다. 본 논문에서는 다음과 같이 X축과 Y축을 적용하여 회전복잡도 척도를 측정한다.

- X축: 커밋이 발생한 매월
- Y축: 각 달의 마지막에 발생한 커밋을 기준으로 프로젝트의 모든 소스 파일의 회전 복잡도의 평균

4. 프로젝트 선정과 척도 측정

우리는 깃허브 싸이트에서 두개의 프로젝트를 선정하였다. 그 선정 기준, 그리고 선정된 프로젝트로부터 척도를 어떻게 측정했는지 설명한다.

4.1 프로젝트 선정

우리는 깃허브에서 jake2 프로젝트와 JFreeChart 프로젝트를 선정하였다. 먼저 두 프로젝트 공통적인 선정기준은 Java언어로 프로그래밍된 프로젝트라는 점이다. 다음으로 개별 선정 기준은 jake2 프로젝트의 경우 커밋이 지난 몇 년간 이루어 지지 않은 종료된 프로젝트 이면서 커밋의 수가 500을 넘는 프로젝트이다. JFreeChart 프로젝트의 선정 기준은 지속적인 유지보수가 발생하고 있는 프로젝트이다.

4.2 척도 측정 방법

우리는 각 프로젝트의 개정이력으로 부터 커밋의 빈도, 코드의 라인수, 회전복잡도를 측정하였다. 커밋의 빈도는 Eclipse의 jgit을, 나머지 두 척도는 Eclipse JDT의 AST를 사용해 측정하여, 커밋 ID, 커밋된 시간, 파일이름과 함께 CSV파일로 출력하였다. 그런다음 파이썬의 Pandas모듈을 사용하여 커밋된 시간으로 정렬한뒤 커밋이 발생한 달의 커밋수를 계산하고, 복잡도의 평균을 계산하고 코드의 라인수를 합산하였다.

5. 가설과 척도의 비교

두 프로젝트의 척도의 측정 결과는 [그림3] ~ [그림8]과 같다. 이 결과를 기반으로 우리가 설정한 가설을 논의하고 선정된 프로젝트의 결과를 비교한다.

5.1 H1에 대한 결과 분석

H1는 커밋의 빈도가 [그림1]과 유사한 형태를 포함하고 있을것으로 가정한다. 이를 각 프로젝트 별로 그려본 결과가 [그림 3]과[그림 6]이다.

먼저 jake2프로젝트의 커밋의 빈도 그래프인 [그림3]을 보면 영에서 증가를 시작하는 도입기의 형태는 없지만, 이후의 시기인 상승하는 형태의 성장기, 빈도가 유지되는 형태의 성숙기, 영으로 내려가는 형태의 쇠퇴기가 나타남을 볼 수 있다. 즉, 성장기에서 쇠퇴기까지의 그래프와 유사하다고 볼 수 있다.

다음으로 JFreeChart 프로젝트의 커밋의 빈도 그래프인 [그림6]에서도 영에서 증가를 시작하는 도입기가 나타나지는 않지만 jake2프로젝트보다 단기간에 성장기, 성숙기와 쇠퇴기를 볼 수 있다. 하지만, 쇠퇴기 이후 다시 성장기를 맞이하는 모습을 볼 수 있다. 이것은 지속적인 갱신등의 결과로 인기가 다시 올라가는 경우이다. 따라서 측정 결과는 기대했던 H1의 가설에 근접함을 찾을 수 있었다.

5.2 H2에 대한 결과 분석

H2는 소프트웨어가 진화할수록 전체 코드의 라인수가 길어진다고 가정한다. 하지만, 전체 코드의 라인수를 측정한 결과 [그림4]와 [그림7]처럼 두 프로젝트 모두 시간에 따라 10000~20000줄 정도의 전체 코드 라인수가 줄어든 것을 발견하였다. 따라서 H2의 가설과 측정 결과는 상반되는 것을 볼 수 있었다.

5.3 H3에 대한 결과 분석

H3는 소프트웨어가 진화할수록 회전복잡도가 증가하거나 유지된다고 가정한다. 이에 따라, 회전복잡도를 측정한 결과 [그림5]와 [그림8]처럼 두 프로젝트 모두 소프트웨어 진화에 맞추어 회전복잡도가 적절하게 유지된 것을 발견하였다. 따라서 H3의 가설과 측정 결과는 근접함을 찾을 수 있었다. 연구 [1]과는 달리 복잡도는 증가하지 않고 유지되는 추세를 보임을 이해할 수 있었다.

6. 관련 연구

소프트웨어의 척도를 이용하는 여러 연구들이 있었다. 먼저 소프트웨어 복잡도의 가시화에 관한 연구이다.[3] 클래스를 메소드 단위로 잘라 그에 대한 회전복잡도를 3단계로 나누어 색깔로 구별한 바형태로 시각화하였다. 이에 기반하여 개선할 클래스를 명시하고자 하였다..

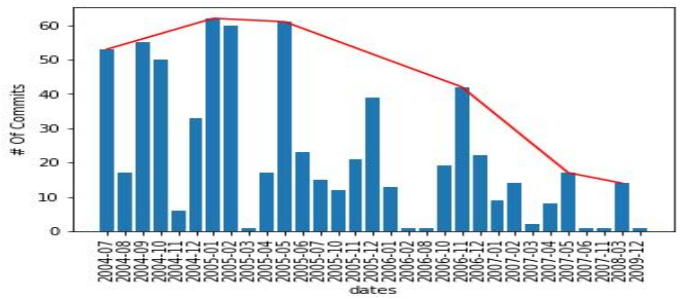
다음으로 소프트웨어의 진화에 따른 소프트웨어 척도의 변화, 척도와 에러율의 관계 그리고 척도 사이의 관련성을 파악하는 연구이다.[4] 이 논문은 척도사이의 관련성 파악에 초점을 두었다.

마지막으로 세가지 오픈소스 프로젝트의 진화에 따른 7가지 척도의 변화를 파악하는 연구이다.[5] 하지만 세가지 프로젝트의 릴리즈 버전을 대상으로 측정하였고 소프트웨어 진화의 단계와 연관짓지 않았다.

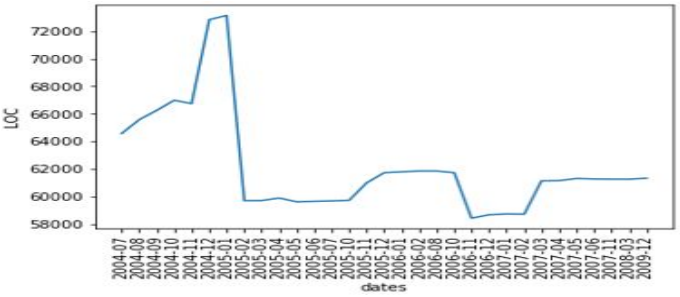
기존 논문은 소프트웨어 척도를 사용한다는 점은 본 논문과 유사하지만, 소프트웨어의 진화와 연관지은 논문은 찾기가 힘들었다. 본 논문은 개정이력을 바탕으로, 제시한 척도들이 제품수명수기 그래프와 과거 소프트웨어 진화의 법칙에 일치하는지에 초점을 두고 있기 때문에 기존과 다르다.

7. 결 론

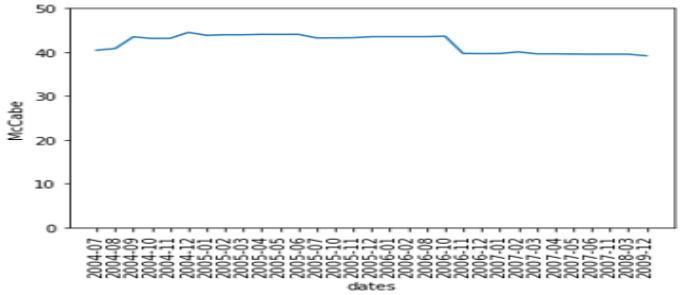
우리는 프로젝트의 진화 상태를 파악하기 위해 소프트웨어 프로젝트의 개정이력을 이용하여 소프트웨어 진화



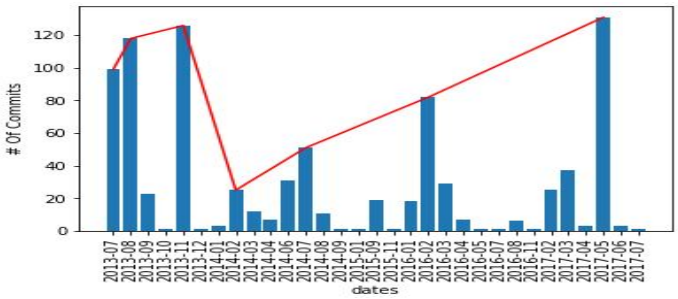
[그림 3] jake2 프로젝트 커밋의 빈도



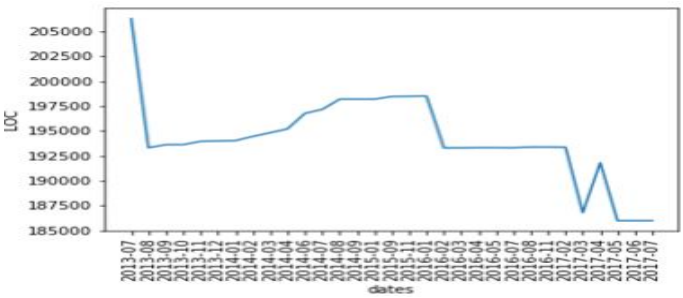
[그림 4] jake2 프로젝트 코드의 라인수



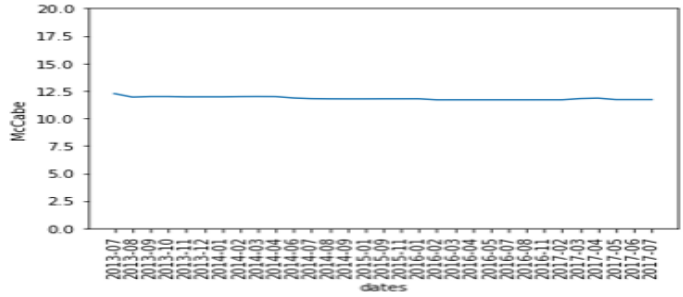
[그림 5] jake2 프로젝트 회전복잡도



[그림 6] JFreeChart 프로젝트 커밋의 빈도



[그림 7] JFreeChart 프로젝트 코드의 라인수



[그림 8] JFreeChart 프로젝트 회전복잡도

를 커밋의 빈도, 코드의 라인수, 회전복잡도를 계산하였다. 이를 위해 세가지의 가설을 세우고 척도를 측정하여 비교한 결과, 첫번째와 세번째 가설에 대한 척도의 측정 결과는 우리가 예상한 결과와 근접하게 측정되었지만, 두번째 가설에 대한 측정결과는 우리가 예상하던 결과와 다른 결과가 나타났다. 또한 두 프로젝트를 비교한 결과는 세가지 척도 모두 유사한 형태를 나타냄을 발견하였다. 그리고 종료된 프로젝트와 지속되고 있는 프로젝트의 척도 측정 결과를 비교한 결과 두 프로젝트는 첫번째 가설에서 차이를 보였다. 종료된 프로젝트는 쇠퇴기와 유사하게 커밋 횟수가 줄어드는 부분이 발견되었지만, 지속되는 프로젝트에서는 발견되지 않았다.

본 논문의 연구 결과로 부터 커밋의 횟수가 소프트웨어 진화의 생명주기 상태와 연관이 있을 것으로 추정할 수 있었다. 그러나 코드의 라인 수는 연관성이 적음을 보았으며, 복잡도는 유지되는 점으로 보아 두 척도는 소프트웨어 진화와 직접적으로 관련되지 않은 척도라는 사실을 배울 수 있었다. 이러한 결과에 기반하여 소프트웨어 진화의 단계를 가시화하는 연구와 소프트웨어 진화에 연관되는 다른 척도를 발견하고자 한다. 또한 이러한 척도를 다양한 프로젝트에 적용하여 소프트웨어의 진화하는 상태와 변화 패턴을 찾음으로서, 현재 소프트웨어의 생명주기 단계를 진단하고 그 단계에서 필요한 조치를 취하는 방법을 연구하고자 한다.

8. Reference

[1] MeirM.Lehman,“Programs, Life Cycles, and Laws ofSoftware Evolution”,1980, Proceedings of the IEEE, Vol.68, pp.1060-1076

[2]William E. Cox, Jr.,“Product Life Cycles as Marketing Models”,1967,The Journal of Business, Vol.40,pp.375-384

[3] Dileram Bansal,Ajit Saxena,Gajendra Singh, “Improvement and Implementation of Software Quality by Using Software Metrics”,2014,International Journal of Scientific Engineering and Technology, Vol.No.3,pp.738-742

[4]Yahya Tashtoush,Mohammed Al-Maolegi,Bassam Arkok, “The Correlation among Software Complexity Metrics with Case Study”,2014,International Journal of Advanced Computer Research

[5]Nicholas Drouin,Mourad Badri,Fadel Touré,“Analyzing Software Quality Evolution using Metrics: An Empirical Study on Open Source Software”,2013,JOURNAL OF SOFTWARE, VOL.8,NO.10,

블록체인 기반 문서 구성요소 단위 무결성 지원 시스템

장지현, 노지우, 홍준기, 송성한, 김순태

전북대학교 소프트웨어공학과

wkdwlgus901@jbnu.ac.kr, wldn306@jbnu.ac.kr, rlwns012@jbnu.ac.kr,
201314106@jbnu.ac.kr,stkim@jbnu.ac.kr
전라북도 전주시 덕진구 백제대로 567

A Block-chain based System for Document Integrity Support by using Document Unit.

Jihyun Jang, Jiwoo Noh, Joongi Hong, Seoung han Song, Suntae Kim

Dept. of Software Engineering, Chonbuk National University

567 Baekje-daero, deokjin-gu, Jeonju-si, Jeollabuk-do 54896, Republic of Korea

요 약

문서관리 시스템은 중앙 집중화된 저장소를 기반으로 다양한 형태의 문서와 자료를 통합하여 관리하는 시스템이다. 기존의 문서관리 시스템은 문서 전체를 저장의 단위로 사용하기 때문에 문서에 대한 공동 작성 자들 간의 수정 이후 특정 구성요소에 문제 발생 시 해당 문서 내 문제 발생 부분을 발견하는데 비용과 시간을 많이 소모하고 문제 발생 구성요소에 대한 최종 작성자가 불분명하고 제 3의 작성자가 수정자의 명의를 악의적으로 변경이 가능하여 기록된 정보의 유효성이 의심되는 무결성 문제가 발생된다. 본 논문에서는 이러한 문제를 해결하기 위해, 기존의 문서관리 시스템의 저장단위보다 좀 더 세분된 문서 내의 구성요소를 저장단위로 하여 구성요소의 최종 작성자를 찾는데 드는 시간과 비용을 줄이고 Private block chain 기술을 이용하여 문서와 문서 내의 수정정보의 무결성을 증명하는 관리 시스템을 제안한다.

1. 서 론

문서관리 시스템이란 중앙 집중화된 저장소를 기반으로 하여 조직의 여러 사용자가 공동으로 작업을 수행할 수 있도록 저작도구로써의 역할과 형상관리기능을 지원해주는 시스템이다[1]. 문서관리 시스템에서 중요한 특징 중 하나는 문서의 무결성을 보호하는 것으로 문서의 무결성이란 권한이 없는 악의적인 제 3의 작성자에 의한 문서의 변경, 추가, 또는 삭제 되지 않는 상태를 의미한다[2]. 문서의 무결성이 지켜지지 않는다면 공동으로 작업을 수행하는 문서관리 시스템 안에 왜곡된 문서가 사용되고 공유되게 되어 어떤 문서가 왜곡되지 않는 문서인지 확인하기 어렵게 되어 혼란을 초래할 수 있다. 문서관리 시스템에서 생성된 전자 문서의 경우 종이 문서와는 달리 간단한 조작만을 통해서 도 쉽게 내용이 위변조될 수 있고 이용과 보존의 과정에서도 내용이 왜곡되거나 삭제되는 등 문서의 무결성이 쉽게 위험에 노출된다[3]. 문서 무결성 문제를 개선하기 위한 기존의 문서 무결성을 보호하는 연구로는 Public Key와 Secrete Key를 이용하여 인증하고 암호화 하여 무결성을 보호하는 Digital signature[5]를 통한 접근방법과 문서의 데이터를 이차원 바코드[4]에 담아 문서의 이동 또는 보관 중에 발생할 수 있는 문서의 왜곡의 여부를 발견하기 위한 접근방법 등이 있다.

문서의 무결성을 보호하기 위해 본 연구에서는 블록체인을 기반으로 하여 문서 전체를 저장단위로 하는 기존의 시스템과 달리 문서의 구성요소를 저장단위로 하고 기존의 문서와 비교하여 수정된 문서의 구성요소, 작성자(수정자)의 정보, 문서의 정보, 수정된 시간 등을 블록에 담아 블록체인에 추가하여 기록함으로써 제 3자의 왜곡을 방지하고 문서의 무결성을 보호할 수 있도록 하는 문서 구성단위 무결성 지원 시스템을 제안하고자 한다.

2. 관련 연구

전자문서의 무결성을 보장하기 위해서는 다음과 같이 크게 두 가지 종류의 접근방법이 존재한다. 첫 번째 방법으로는 Digital Signature가 있다. Digital signature기술은 가장 넓게 퍼져있는 무결성과 인증을 증명하는 방법 중에 하나이고, digital signature 기술은 공개키(PK), 개인키(SK)를 사용하는 기술이다. Digital signature 방법은 문서를 먼저 hash화 하고 작성자의 개인키로 암호화된 hash값이 생성되어 문서에 digital sign이 되면 이를 공개 키와 함께 전송한다. sign이 된 문서와 공개 키를 받은 수신자는 발신자의 공개 키로 디지털 서명을 해독하여 문서의 유효성을 확인하고 해독 후 나타난 문서의 hash 값을 비교하여 문서의 무결성을 확인한다[5].

하지만, 이 접근 방법은 문서의 물리적 형식을 기반으로 문서의 유효성을 검사하기 때문에 이 문서의 신원(원본 문서의 진위여부)이 확인된 경우에만 신빙성이 입증될 수 있습니다. 이러한 문제점은 블록체인을 사용하여 문서의 초기 형태를 기록해 놓는다면 블록체인은 한번 기록된 정보는 수정할 수 없는 특징을 가지고 있기 때문에 해결될 수 있다[6].

두 번째, 이차원 바코드를 사용하여 문서의 무결성을 증명하는 접근 방법이 있다. QR코드를 사용한 방법에서는 암호화 알고리즘을 사용하여 문서의 무결성을 증명하였는데, 먼저 문서 내에 있는 모든 데이터를 AES암호화 알고리즘을 통하여 암호화한다. 이후에 이를 이차원 바코드(QR코드)를 사용하여 문서에 삽입한다. 이 QR코드는 추후에 다시 AES알고리즘을 통하여 복호화 되고, 원본 문서와의 내용 비교를 통하여 문서의 무결성을 확인한다[4].

하지만, 이 방법은 문서의 무결성을 확인하는 단계에서 종료되기 때문에 문제가 발생되어 무결성이 훼손된 부분을 찾기 위해서는 문서 전체를 비교 대조해야 하고 수정된 부분 중 어떤 요소가 악의적으로 왜곡되었는지 확인하기 어려운 문제점이 있다. 이러한 문제점을 개선하기 위해서는 문서의 저장단위를 구성요소단위로 개정하여 작성자의 정보와 함께 블록체인에 기록한다면 권한이 있는 작성자가 수정한 요소와 악의적인 사용자가 왜곡한 문서의 구성요소를 구별할 수 있게 되어 해결할 수 있다.

3. 본 론

본 장에서는 블록체인을 기반으로 한 구성요소단위 문서 무결성 지원 시스템을 제안한다. 본 시스템은 아래의 그림1과 같이 4단계로 구성된다. 첫 번째 단계는 세부 저장단위 추출로 수정된 문서의 세부 문서 구성요소 단위를 추출한다. 두 번째 단계는 새로운 블록의 생성으로 추출된 구성요소, 작성자 및 수정자, 수정 시간 등을 블록화 한다. 세 번째 단계는 분산 합의 단계로 생성된 블록을 P2P를 사용하여 다른 사용자들과 공유하고 검증한다. 마지막 단계는 블록체인에 블록을 추가하는 단계로 검증된 블록을 블록체인과 연결한다. 다음 각 세부장에서 상세한 내용을 설명한다.

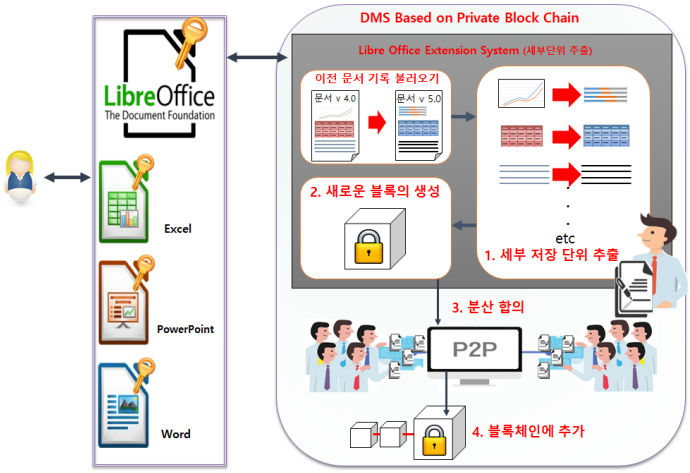


그림1. 문서 구성요소단위 무결성 지원 시스템

3.1 세부 저장단위 추출

세부 저장단위란 문서가 저장될 때 저장되는 단위를 세분화 한 것을 의미한다. 이 단계에서는 문서를 구성요소단위로 세분화하여 추출한다. 추출은 문서 무결성 지원 시스템에 의해 자동으로 진행이 되고 문서의 구성요소는 아래의 그림 2과 같이 문단, 표, 차트, 이미지 등으로 나누어 추출된다. 각각의 추출된 구성요소는 독립적으로 관리 되며 TXs(Transaction)단위로 취급한다.

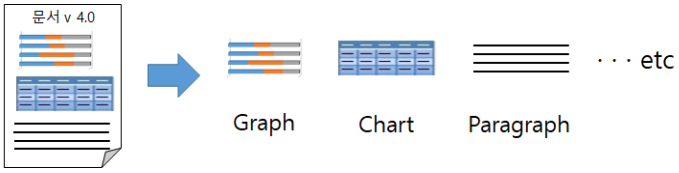


그림2. 문서의 세부 저장단위

구성요소단위 추출 단계에서 문서 무결성 지원 시스템은 기존의 저장되어있던 문서 구성요소 TXs들과 현재의 수정된 TXs을 비교 분석하여 구성요소 단위에서의 변경된 사항을 추출하고 추출된 구성요소는 블록화 단계에서 블록에 포함된다. 위와 같이 세분화되어 추출된 구성요소는 블록에 저장될 때 수정자의 정보와 같이 저장이 되어 문서의 특정 구성요소에 문제발생시 모든 문서를 비교 대조해야 하는 문서 전체를 저장단위로 하는 다른 시스템과 달리 최초 문제발생시점의 수정자를 찾는데 시간과 비용을 줄일 수 있다. 또한 중복적인 왜곡이 일어나더라도 각각의 Tx들이 독립적으로 관리 되기 때문에 문제 발생

부분의 구성요소와 왜곡전의 데이터를 발견하는데 시간과 비용을 단축시킬 수 있다.

3.2 새로운 블록의 생성

이번 단계에서는 블록체인에서 하나의 저장단위인 블록을 생성하는 단계로 블록화 단계라고도 정의할 수 있다. 블록화 단계에서 블록을 이루는 요소로는 아래의 그림 3과 같이 이전에 언급되었던 수정되었다고 인정된 TXs와 TXs의 Hash 값, 수정시간, 수정자의 정보, 이전 블록의 Hash point, 문서 고유의 ID 등이 포함된다. 각각의 요소들은 다른 요소들에게 영향을 미치지 않고 독립적으로 관리되어 블록에서 TXs로 사용된다. 이중 수정자의 정보는 블록이 생성되는 과정에서 하나의 Tx로 사용되어 블록체인에 기록되기 때문에 차후에 악의적인 사용자에 의해 왜곡될 위험에서 벗어날 수 있다.

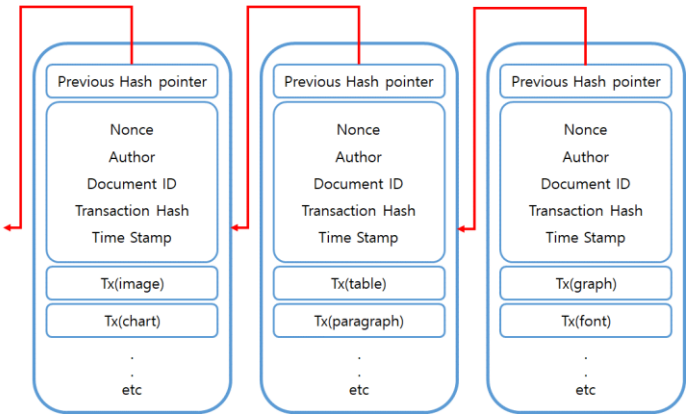


그림3. 블록화

블록을 생성할 때 각각의 Tx들을 담은 블록을 SK(secrete Key)로 암호화 하여 생성한다. 위 그림3는 블록을 도식화 한 그림으로 각각의 블록이 이전 블록의 hash pointer을 가지고 있음을 나타낸다. 이 hash pointer들은 다음 과정인 3.3 분산 합의 과정에서 pointer들이 가리키는 블록이 유효한지 판단하고 판단을 기반으로 하여 유효성과 무결성이 의심되는 블록은 더 이상 쌓이지 않고 폐기됨으로써 왜곡된 문서의 확산을 방지하고 이로 인해 문서의 무결성을 보호할 수 있다.

3.3 분산합의

이 단계에서는 블록화 과정으로 생성된 새로운 블록들을 기반으로 아래의 그림 3과 같이 수정된 문서를 DMS에 업로드하고 수정된 Txs을 블록에 담아 SK로 암호화 하여 다른 사용자들에게 전달하면 다른 사용들이 DMS로부터 다운로드 한 수정된 문서와 전달받은 암호화된 블록을 PK(Public Key)로 복호화 하여 문서를 비교하여 전달받은 블록에 담긴 문서의 Tx의 유효성과 다운로드 받은 문서의 무결성을 검사하는 분산합의 과정을 통해 새롭게 생성된 블록과 DMS에 저장된 문서의 유효성을 검증하고 이 결과를 보장한다.

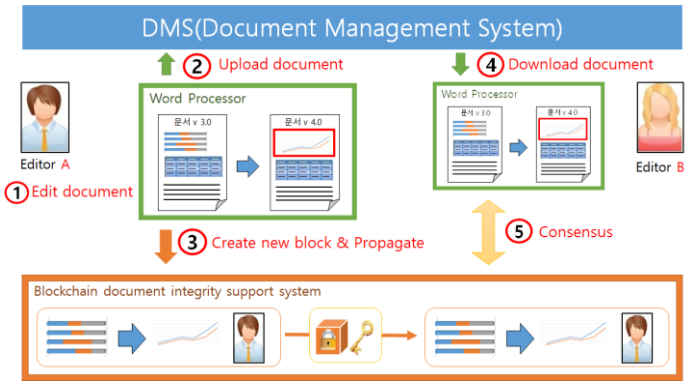


그림4. 분산 합의 과정

분산합의 과정은 크게 4가지 단계로 진행이 되는데 아래의 그림 4.2는 먼저 수신자가 블록을 수신 받았다고 가정하고 단계를 설명한다. 첫 번째 단계에서는 수신된 블록이 유효한 블록인지 위의 그림 4와 같은 과정을 통해 판단하고 유효하지 않다면 수신 받은 블록을 제거하고 유효하다고 판단되면 다음단계로 넘어간다. 두 번째 단계에서는 생성된 블록이 가진 이전블록에 대한 hash pointer가 가리키는 블록이 현재 소유중인 블록체인의 마지막 블록을 가리키고 있는지 확인한다. Hash pointer가 가장 마지막 블록을 가리키고 있다면 수신 받은 블록을 블록체인에 추가하고 가리키고 있지 않다면 다음 단계로 넘어간다. 세 번째 단계에서는 마지막 블록에 기록된 timestamp와 수신 받은 블록에 기록된 timestamp를 비교하여 마지막 블록에 기록된 timestamp가 수신 받은 블록의 timestamp보다 작다면 마지막 블록이 수신 받은 블록보다 먼저 생성되었으므로 수신 받은 블록을 거부하고 수신 받은 블록의 timestamp가 더 작다면 수신 받은 블록이 더 먼저 생성된 블록임으로 다음단계로 넘어간다. 마지막 네 번째 단계에서는 마지막 블록의 생성자가 자신인지 확인하고 자신이라면 블록을 재 생성하여 다시 분산합의 과정을 거치도록 하고 아니라면 마지막 블록을 invalid한 블록으로 표시한다.

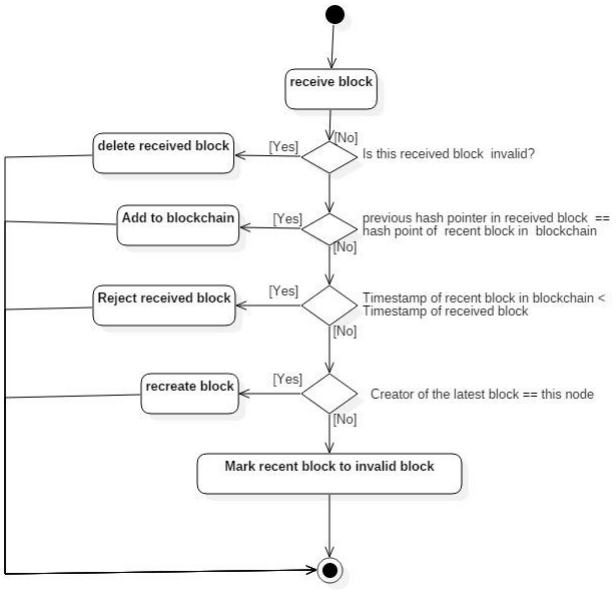


그림4.2. 분산 합의 과정 알고리즘

기존의 중앙 집중형 합의에서는 모든 권한이 집중된 하나의 중앙 데이터베이스가 존재하여 거래의 유효성을 입증하였지만 분산 합의에서는 탈 중앙화된 가상 네트워크에 권한을 이전시켜 분산된 네트워크의 각각의 노드(사용자)들의 신뢰를 기반으로 거래를 입증하는 방법을 사용한다. 이러한 분산 합의 과정을 통해 분산된 네트워크의 노드 중 하나의 노드에서 생성된 블록의 트랜잭션 내역을 같은 블록체인장부를 공유하고 있는 다른 노드들에게 공유하여 과반수 이상이 생성된 블록의 유효성을 검증하고 유효한 블록으로 간주되어 블록체인에 추가하게 된다[6].

3.4 블록체인에 블록 추가

이 단계에서는 위의 3.1 세부 저장단위 추출, 3.2 블록화 그리고 3.3 분산합의 단계를 거쳐 생성되고 다수의 사용자가 유효성과 신뢰성을 검증한 블록은 유효한 블록으로 취급된다. 이 블록은 모든 노드(사용자)들이 동일하게 소유하고 있는 블록체인에 추가된다. 위의 방식으로 블록체인에 추가된 블록의 기록(leger)은 모든 노드들이 동일한 기록을 가지고 있기 때문에 하나의 노드가 임의로 수정한다 하여도 다음 블록이 생성되고 다시 분산합의 과정을 거칠 때 노드들이 서로 가지고 있는 leger들의 블록의 유효성을 확인 할 때 걸러지게 됨으로 권한이 없는 제 3의 악의적인 사용자가 왜곡한 문서는 블록체인에 기록을 남기지 못한다. 설명과 같이 블록체인을 사용하는 네트워크의 모든 노드들이 동일한 기록을 가진다는 특성은 본 연구의 목적인 문서 무결성을 지원하는 가장 중요한 특성이다.

4. 결 론

본 연구에서는 기존의 문서 무결성을 위한 연구중 하나인 digital signature방법에서 문제점으로 여겨졌던 문서의 신원 확인 문제를 블록체인의 특징인 모든 노드들이 동일한 leger를 가지고 블록의 생성 마다 비교하고 유효성을 검사함에 따라 초기 원본문서의 왜곡 또는 수정의 기록유실을 방지하여 초기 문서의 신원확인 문제를 해결하였다. 또 다른 연구방법인 이차원 바코드를 사용하는 접근방법에서 개선되어야 할 사항이었던 문서 내에서 왜곡된 문제 부분을 탐색하는데 시간과 비용이 많이 소모되는 문제를 문서의 저장단위를 세분화 하여 블록으로 만들고 이를 블록체인에 추가하여 문서를 일일이 비교하지 않아도 가장 최근의 블록에 저장된 구성요소 Tx의 비교를 통해 문서의 문제부분을 발견할 수 있도록 개선방안을 제안했다. 또한 수정자(작성자)의 정보를 블록에 추가하여 블록체인에 기록하는 방법으로 제 3의 악의적인 사용자가 기존에 저장되었던 수정자의 정보를 왜곡할 위험성도 배제하였다.

References

[1]Rivera, A. (2017, September 08). Document Management Systems: A Buyer's Guide. Retrieved December 27, 2017, from http://www.businessnewsdaily.com/8026-choosing-a-document-management-system.html

[2] Maintaining Document Integrity. (n.d.). Retrieved December 27, 2017, from http://www.ironmountain.com/resources/general-articles/m/maintaining-document-integrity

[3]김용. (2008). 온라인 환경에서의 전자문서 안전배포 및 이용을 위한 인증방법 설계 및 구현. 한국정보관리학회 정보관리학회지. 25, 75-98.

[4]류기훈, 임선영, 한희일. (2007). 이차원바코드를 이용한 공문서 위·변조 방지 확인 시스템 제안. 한국지능정보시스템학회 학술대회논문집, 444-448.

[5] F Boudrez, "digital signatures and electronic records", *Arch Sci*, vol. 7, no. 2, pp. 179-193, June 2007.

[6] M. Mainelli, M. Smith, *et al.*Sharing ledgers for sharing economies: an exploration of mutual distributed ledgers (aka blockchain technology) J Finan Perspect, 3 (3) (2015), pp. 38-69

ThingML의 IoT 시스템 설계 적용 사례 연구

김가영 ○, 안지흔, 최은만
동국대학교 컴퓨터공학과
rkdud2507@naver.com, {dkswlgms05, emchoi}@dongguk.edu

A Case Study on Application of ThingML to IoT System Design

Ga Young Kim○, Ji Heun Ahn, Eun Man Choi
Dept. of Computer Engineering, Dongguk University

요 약

이 연구는 다양한 플랫폼과 네트워크 노드로 구성되는 IoT 시스템의 구현하기 위하여 ThingML이라는 모델을 도입하여 이질성을 극복하고 점증적인 소프트웨어 공학 기술의 적용성을 탐구하는 것이다. IoT 시스템은 제한적인 성능을 가진 작은 디바이스로 구성된다. 따라서 여러 가지 다른 수준, 객체지향 프로그래밍 뿐만 아니라 C, nesC의 낮은 수준의 프로그래밍도 필요하다. 이 연구의 목표는 ThingML 프레임워크가 IoT 시스템을 개발하는데 얼마나 효과적으로 적용될 수 있는지 사례 구현을 통하여 확인하는 것이다. 자전거 도난 경보 IoT 시스템을 ThingML로 모델링하고 이를 바탕으로 코드로 구현하였다.

Keywords: Internet of Things, ThingUI,UML, Software Design, Case Study

1. 서 론

사물 인터넷(IoT: Internet of Things)은 인터넷 기술과 소프트웨어 기술이 융합하여 진화하여야 할 다음 스텝이다. IoT의 기초는 이질적인 분산 시스템에 제공되는 서비스이다. 이질적 분산 시스템은 다양한 노드와 네트워크로 구성된 인프라에서 수행되어야 하며 서비스의 구현은 비동기적으로 통신하는 독립된 서비스로 구성된다. 클라우드에 모든 것을 연결하는 단순한 방법이 가능하지만 자원의 효율성, 실시간, 강인성, 확장성 측면에서 한계가 있다.

IoT를 제한 없이 구현하려면 인프라에 분산된 서비스가 구현되어야 한다[1,2]. 즉 이질적인 네트워크 컴포넌트 위에 어떻게 손쉽게 다양한 서비스를 올릴 수 있고 그것들이 연동될 수 있는가가 관건이다[4,5]. IoT는 센서 등의 네트워크 컴포넌트에 비즈니스 가치를 얹은 서비스이므로 개발 과정에도 다양한 추상 수준에 대한 분석이 필요하며 따라서 이질적인 전문가의 협업이 필요하다.

다양한 이질적인 네트워크 컴포넌트들을 IoT의 동일한 소프트웨어 층에 숨기려면 ThingML과 같은 높은 추상적 설계 표현이 필요하고 이를 기반으로 네트워크 요소들을 구현하면 모듈화, 재사용, 지속적 통합, 검증 등 품질을 획기적으로 개선할 수 있다.

이 논문에서는 ThingML이라는 모델을 도입하여 이질성을 극복하고 점증적인 소프트웨어 공학 기술의 적용성을 탐구한다. 먼저 ThingML에 의한 모델링 방법을 소개하고 IoT에 적용한 설계 및 구현 사례를 소개한다. 사례 연구에서 구현한 결과를 기술하고 그 효과를 정리하였다.

2. IoT 소프트웨어의 설계와 구현

2.1 IoT 시스템의 프레임워크

IoT에 대한 초기 연구는 주로 통신 및 상호 운용성에 중점을 두었다. 최근 연구들은 인프라에 분산된 서비스들을 통합하기 위한 다양한 접근을 시도하고 있다.

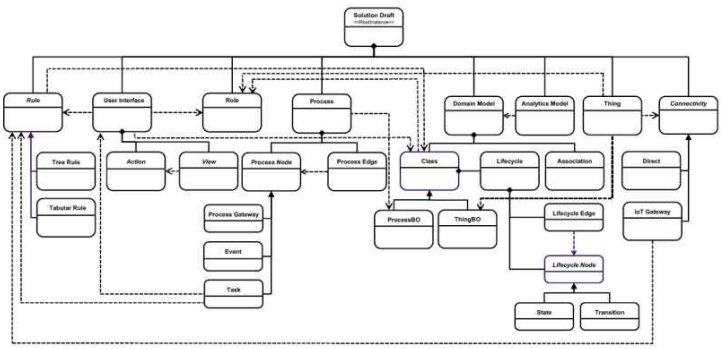
IoT 시스템의 설계와 개발에 대한 현재 접근 방식은 사물(thing)에 사람, 장소, 물체 등 다양한 물리적인 물체가 포함된다. 간단한 RFID태그 또는 Bluetooth 비콘, 환경 또는 모션센서를 모든 종류의 사물에 부착하면 위치 및 상태를 추적하고 디지털 정보를 전송할 수 있다. 이러한 다양한 사물을 원격 제어 가능한 디지털 액츄에이터로 통합할 수 있다.

IoT 시스템에 부가 서비스가 가능하도록 하기 위해 서로 다른 것을 함께 연결하고, IoT 시스템과 그 기능에 접근하는 방법을 제공하는 소프트웨어 인프라, 즉 IoT 미들웨어가 필요하다. 이렇게 연결하기 위해서 고려해야 할 문제점들이 몇 가지 있는데, 첫번째는 상호 운용성이다. 이질적인 객체 간의 상호작용에는 사물식별 외에도 통신 프로토콜 및 데이터의 표현 체계가 필요하다. 이 문제에 대한 연구는 이미 초기의 IoT 연구에서 다루고 있다. 두 번째로는 의미론이다. 사물의 통합을 가능하게 하기 위해서는 개념에 대한 공통된 의미가 정의되어야 한다. 세 번째 문제는 사물과 사물, 사물과 인간 관계를 파악하여 확립하고, 활동의 조율이 필요하다는 점이다[6]. 마지막으로 문맥 인식과 적응이 필요한데, 시스템 구성 요소의 특성상 서비스가 어느 시점에, 언제까지 제공 가능할 지 알 수 없다. 이를 위해서는 상황에 따라 동적으로 적응할 수 있는 발견 및 조정 메커니즘이 필요하다[3,5].

IoT 시스템이라는 용어는 일반적으로 IoT 장치 세트와, 네트워킹 및 상호 작용을 관리하는 미들웨어 인프라를 뜻한다. 특정 소프트웨어를 IoT시스템 위에 논리적으로 배치하여 시스템 활동을 조율하고, 특정 서비스 및 범용 프로그램 혹은 프로그램 집합을 제공할 수 있다. 특정 서비스를 제공한다는 것은 이해 관계자와 사용자가 사물에 액세스하여 이용하고 감지 또는 작동기능을 지시할 수 있음을 뜻한다.

2.2 IoT 설계 방법론

IoT 프로젝트는 여러 도메인으로 분할되는 경향이 있어 성공적으로 수행하기 어렵다. 하드웨어를 위한 임베디드 IT 프로젝트, 통신 측면을 위한 텔레콤 프로젝트, 백엔드 및 통합을 위한 전통적인 엔터프라이즈 IT 프로젝트 등의 다양한 도메인들을 서로 연결하고, 모든 이해 관계자간에 공통적인 의미 이해를 도출하기 위해 비즈니스 프로세스를 핵심 결과로 사용하는 IoT 설계 분석 방법을 이용한다.



[그림 1] IoT 설계의 메타 모델

설계 클래스는 규칙, 사용자 인터페이스, 역할, 프로세스, 분석 모델을 포함하는 도메인 모델, 사물과 연결성 개념들을 그룹화한다[7].

규칙은 일반적으로 IoT 설계에서 필요한 상태가 없는 논리를 설명한다. 규칙은 입력 값과 출력 값의 설명을 위해 도메인 모델의 클래스를 참조해야 한다. 사용자 인터페이스는 액션과 뷰를 그룹화한다. 역할은 사용자로 매핑될 액세스 권한이 있는 이해 관계자들을 나타낸다. 프로세스는 IoT 프로젝트의 중심이다.

메타 모델에서 중요한 요소는 프로세스 게이트웨이, 이벤트 및 작업을 나타내는 것이다. 작업은 사용자 인터페이스 혹은 규칙을 참조하여 상태가 없는 논리의 실행을 의미한다. 도메인 모델은 시스템 내부의 클래스 및 연관을 나타낸다. 이는 프로세스 내부의 트랜지션 데이터 또는 사물을 나타내는 데이터로 사용된다. 웹 로그 분석 모델은 도메인 모델의 요소를 참조하여 IoT 솔루션의 머신러닝 혹은 딥러닝 알고리즘의 목표와 계산을 의미한다.

결국 IoT 시스템 개발에 필요한 설계 방법은 네트워크 요소 중심의 트랜젝션 모델보다 서비스 층을 잘 추상화 하여 객체 관계 중심 서비스 모델을 이용하는 것이 적합하다. 이런 의미에서 객체 중심 모델인 UML을 사물 중심 모델링 방법으로 최적화 한 ThingML 모델링은 IoT 시스템 모델링 방법에 적합하다. 즉 비즈니스 층의 여러 프로세스와 상태 변화 모델을 추상화 하고 이를 기초로 IoT의 디바이스와 네트워크 자산들의 일반성, 연동성, 확장성을 확보하는 구현을 얻어낼 수 있다.

3. ThingML을 이용한 모델링

3.1 IoT 시스템 모델링

IoT 애플리케이션은 네트워크 노드 처리에 대한 균등 분배 문제, 처리 노드 사이에 사용되는 프로토콜의 이질성이라는 문제점을 극복하는데 어려움을 겪고 있다. 이것을 해결하기 위하여 ThingML은 서비스 개발자와 플랫폼 전문가 간의 협력을 추진하여 가치가 높은 IoT 기반 서비스를 생산할 수 있도록 하였다.

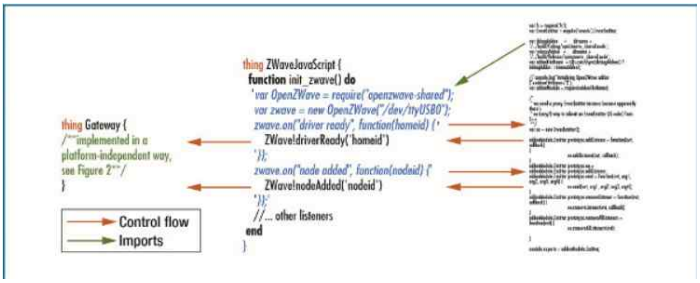
ThingML은 여러 가지 프로그래밍 언어를 지원하는 편집기를 가지고 있는 IoT 소프트웨어 모델링 도구이다. 기본 IoT 플랫폼의 복잡함을 관리하고 플랫폼에 제약이 없어 가치가 높은 IoT 기반 서비스를 생산한다. ThingML은 모델링 언어, 도구 집합 및 방법론으로 구성된다. ThingML은 UML에 맞춘 소프트웨어 모델링 구조와 플랫폼 독립적인 액션 언어를 IoT 애플리케이션을 대상으로 제공한다. 또한 편집기와 UML을 기초로 한 모델 및 다양한 프로그래밍 언어를 지원하는 다중 플랫폼 코드를 생성하는 프레임워크[8]를 포함하고 있다. 여기에 추가하여 개발 프로세스 및 도구를 문서화 하기 편하다.

ThingML 접근법의 목표는 이질적 플랫폼 및 IoT 장치를 추상화 하여 원하는 IoT 시스템의 아키텍처에 모델링 할 수 있게 하는 것이다. 일반적인 아키텍처가 정의되면 상태 차트와 액션 언어를 사용하여 플랫폼 독립적인 방식으로 컴포넌트의 비즈니스 로직을 명세한다. 상태 모델은 일반적으로 IoT 구성 요소의 포트에 수신되는 메시지에 해당되는 이벤트에 반응한다. 액션언어를 사용하면 개발자는 구성요소에서 가드, 액션, 변수 및 기능을 지정하고 상태 머신이 구성 요소의 포트에서 메시지를 보내고 받을 수 있다.

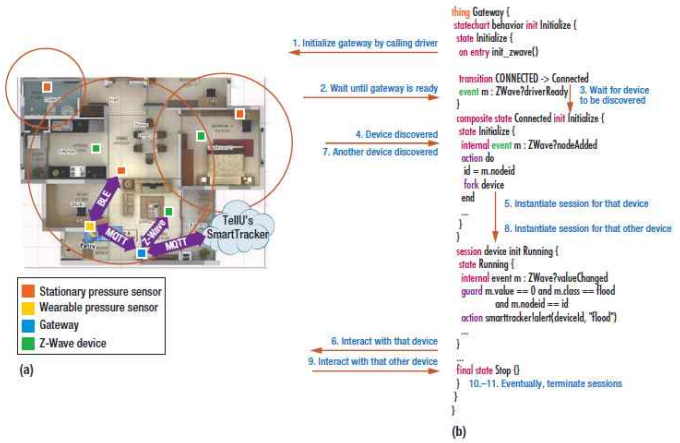
3.2 이질적 플랫폼에 대한 코드 생성

ThingML 접근법은 IoT 개발을 용이하게 하기 위해 ThingML 코드를 대상 코드와 혼합하여 기존 라이브러리를 래핑하는 메커니즘을 포함하고 있다.

플랫폼 독립적인 사양은 개발자가 구체적인 플랫폼을 선택하게 되면 일부 구성 요소를 효율적으로 구현해야 한다. ThingML은 유연성을 제공하기 위해 목표 언어의 액션과 모델 액션을 임의로 혼합한다. 이 메커니즘을 이용하면 쉽게 변수를 공유하고 호출 및 콜백을 양방향으로 구현할 수 있다.



[그림 2] ThingML의 코드화



[그림 3] ThingML 코드와 IoT 구성요소의 매핑

[그림 2]와 [그림 3]은 구성요소의 구현을 보여준다. 파란색 코드는 Z-Wave 드라이버를 초기화하고 콜백을 등록하는 일반 JavaScript 코드이다. 콜백이 호출 될 때 ThingML 이벤트를 발생시키기 위해 이러한 콜백에 ThingML 이벤트를 내보내는 코드를 포함 되어있는 것을 [그림 3]의 코드에서 확인할 수 있다. 이런 방식으로 게이트웨이는 플랫폼 독립적인 상태로 유지되며 이러한 ThingML 이벤트에 단순히 반응한다.

ThingML 접근법을 성공적으로 사용하려면 대상 플렛폼뿐만 아니라 프로젝트의 코딩 스타일, 라이브러리, API 및 레거시 구성요소에 맞게 코드 생성기를 조정해야 한다. ThingML 코드 생성 프레임 워크는 개발자가 도메인 및 프로젝트의 필요에 맞게 코드 생성기를 적용하고 확장할 수 있다.

현재 프레임워크는 추상 프레임워크와 3개 언어(C/C++, Java 및 JavaScript) 및 여러 라이브러리 및 개방형 플랫폼을 위한 미리 준비된 코드생성기 세트로 구성된다. IoT 애플리케이션에서 ThingML 구성요소는 서로 다른 코드 생성기가 사용되는 이질적 플랫폼에 분산되어 있다. 특정 코드 생성기 플러그인을 사용하면 이질적 플랫폼의 구성요소 간에 통신 및 메시지 교환을 생성할 수 있다.

코드 생성기를 사용, 확장 및 정의할 때 테스트 프레임워크를 사용하면 코드 생성기를 자동으로 진단하여 모두 ThingML 구문에 대해 동일한 실행 의미를 제공하는지 확인할 수 있다. 복잡한 사용자 정의에서 프레임워크는 플러그인을 사용하여 여러 플랫폼에서 구성 요소 간의 메시지 교환을 처리 할 수 있다.

4. 사례 연구

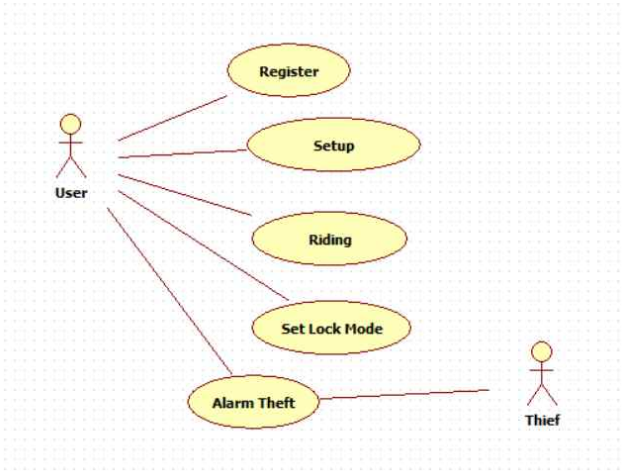
본 논문에서는 센서 투명성 및 서비스 모델이 가능한 소프트웨어 공학기술을 적용하여 IoT 시스템을 구현하는 사례 연구를 하였다. IoT 응용 도메인 중 이벤트와 상태가 그리 복잡하기 않고 명확한 자전거 도난 경보 시스템을 선택하여 적용하여 보았다. ThingML 적용 후 얻는 결과와 효과를 확인하고 평가하였다.

4.1 자전거 도난 경보 시스템의 설계

자전거 도난 경보 시스템은 만약 사용자의 자전거가 타인에 의하여 도난 당하려 할 때, 사용자의 스마트폰 애플리케이션을 통해 이 사실을 알려주어 자전거가 도난 당하는 것을 방지하는 시스템이다. 또한 본 시스템은 사용자가 자전거 주행 시 현재 위치를 확인하는 기능도 갖고 있다.

본 연구의 설계는 다음과 같은 순서로 진행하였다. 사용사례 다이어그램과 명세서 작성, 이를 토대로 상태 다이어그램을 작성하여 기능 모델링 하였다.

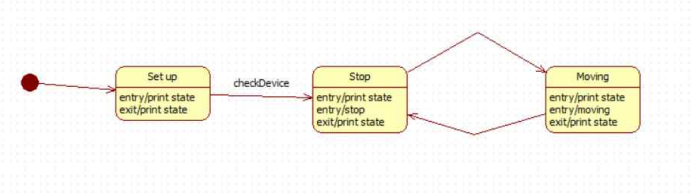
IoT 시스템의 프로세스와 역할을 명확하게 정의하기 위해 사용사례 다이어그램과 명세서를 작성하였다. 자전거 도난 경보 시스템은 자전거에 센서를 부착하고 사용자가 앱을 통하여 확인하는 방식이다. 사용사례 다이어그램에는 자전거 주인(User)과 도둑(Thief) 두 가지 액터가 존재한다. 사용사례는 앱에 등록하는 Register, 앱에 자전거의 센서를 등록하는 Setup, 자전거를 타고 있는 Riding, 잠금 모드를 적용하는 Set Lock Mode, 도둑이 자전거를 훔쳤을 경우 발생하는 Alarm Theft가 있다. 액터와 사용사례의 관계는 아래 그림과 같다.



[그림 4] 사용사례 다이어그램

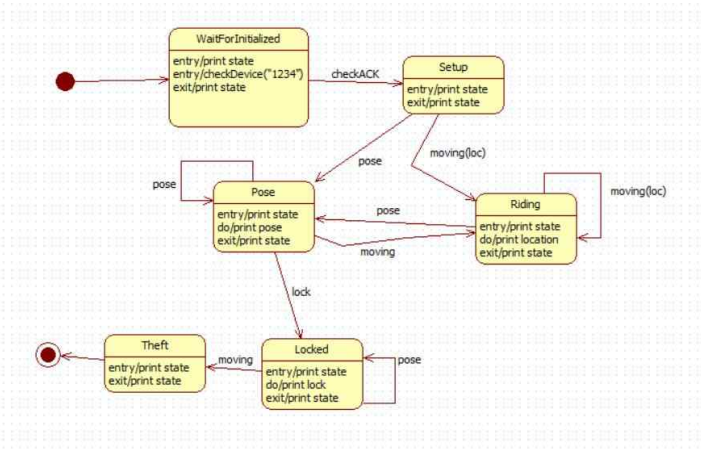
위 사용사례 다이어그램을 토대로 확장시켜 상태

다이어그램을 작성하였다. 앱과 센서는 서로 다른 상태를 갖고 있고 서로 통신하는 관계이기 때문에 앱과 센서 각각에 대해 두 가지 다른 상태 다이어그램이 필요하다.



[그림 5] 센서 상태 다이어그램

센서 상태 다이어그램에서 센서의 상태는 앱과 연결화는 Setup상태, 정지된 상태인 Stop, 자전거가 움직이는 상태인 Moving 상태이다. Stop상태와 Moving상태로 전환 시 entry에서 앱에게 이를 알리는 stop과 moving은 앱에게 상태를 알리는 액션을 수행한다.



[그림 6] 애플리케이션 상태 다이어그램

애플리케이션의 상태 다이어그램에서의 상태는 스마트폰 애플리케이션의 메뉴로 대칭된다. 애플리케이션에서의 상태 변경은 사용자의 선택에 따라서 변하지만, 본 상태 다이어그램에서는 임의로 트랜지션 이벤트를 지정하였다. 'Wait For Initialized' 상태는 센서와의 연결을 위한 상태이다. 'Setup'은 앱에 센서 장치를 등록하는 상태이다. 'Pose'는 자전거가 정지되었을 때의 상태이고 'Riding'은 자전거가 움직이고 있는 상태를 나타낸다.

애플리케이션에서 이 두 상태를 다르게 나타낸 이유는 자전거가 움직이는 동안 애플리케이션은 위치정보를 출력하기 때문이다. 위치 정보는 'moving(loc)' 이벤트의 액션으로 수행된다. 'Locked'은 잠금 상태이며 센서에서 자전거가 움직인 것이 확인되면 도난 당한 'Theft' 상태로 변환한 다음 종료한다.

4.2 자전거 도난 경보 시스템의 구현

위에서 설계한 기능 및 상태 모델링 작업을 기반으로 자전거 도난 경보 시스템을 ThingML을 이용하여 구현하였다.

```
thing Sensor includes bicycleMsgs
{
    required port bicycle{
            sends checkACK, pose, moving, lock
            ...
        }
        statechart behavior init Init{
            ...
            state Moving{
                on entry do
                    print "on state : Moving\n"
                    bicycle!moving("100")
                end
                on exit print "exit state : Moving\n"
                ...
            }
            ...
        }
}
```

[그림 7] 센서 주요 코드

[그림 7]은 구현한 ThingML코드 중 센서에 관한 코드 중 일부이다. 센서는 thing이 되며 port를 통해서 앱과 통신한다. 구현한 상태 중 'Moving'은 state가 시작하고 끝날 때 이를 알리는 내용을 출력하며, state가 시작될 때 앱에게 Moving("100")이라는 메시지를 보낸다. 이때 100은 현재위치를 나타내는 임의의 숫자이다.

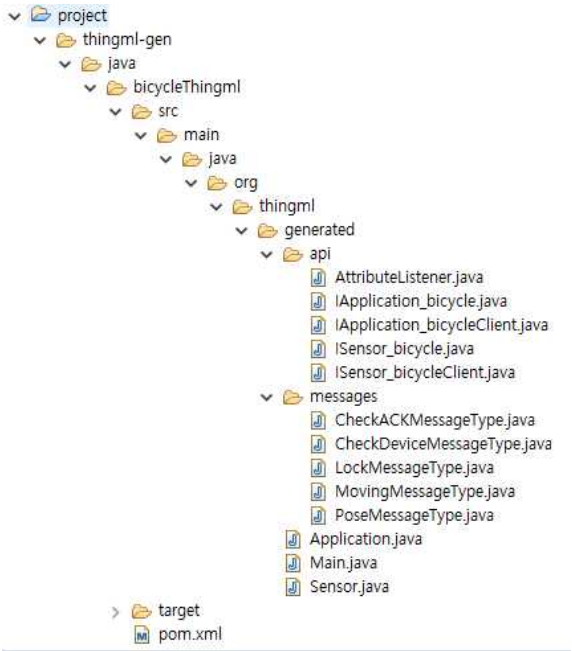
```
thing Application includes bicycleMsgs
{
    provided port bicycle {
        sends checkDevice
        receives checkACK, pose, moving, lock
    }

    statechart behavior init Init {
        ...
        state Locked {
            on entry print "on state : Locked\n"
            on exit print "exit state : Locked\n"

            transition -> Theft
                    event m : bicycle?moving
                    action do
                        print "Theft location : " + m.loc + "\n"
                    end
            ...
        }
        ...
    }
}
```

[그림 8] 애플리케이션의 주요 코드

[그림 8]은 앱에 관한 ThingML 코드 중 일부이다. 앱은 Thing이 되며 port를 통해 센서와 통신을 한다. 구현한 state들 중 Locked 상태는 잠금 상태인 상태이며 state가 시작하고 끝날 때 이를 알리는 출력을 한다. transition은 다른 state로 전환하는 것으로 이때 트리거 이벤트는 moving으로 이는 센서가 보낸 메시지이다. 따라서 만약 센서가 움직이는 상태일 경우 이는 도난 당한 상태이므로 Theft 상태로 전환하고 이때 위치를 출력한다.

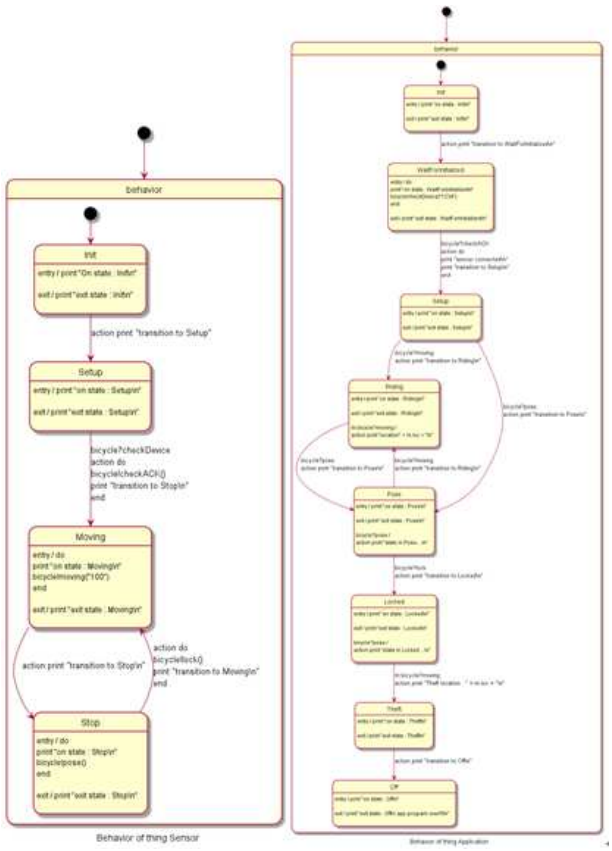
3장에서 기술한 ThingML코드의 다중 플랫폼 코드 생성 프레임워크를 확인하기 위해 구현한

ThingML코드로 java코드와 UML 모델을 생성하였다.



[그림 9] 작성한 ThingML코드 컴파일로 생성된 Java 코드

ThingML 코드와 동일한 의미를 가지는 java 코드들이 생성된다. 이것을 java 컴파일러로 컴파일하면 실행 가능한 코드가 생성된다. 특히 메시지의 경우, 코드 생성시 직렬화와 역직렬화가 이루어짐을 알 수 있다.



[그림10] ThingML 코드로 역생성한 상태 다이어그램

ThingML의 컴파일 옵션 중 UML로 매핑이 가능하기 때문에, UML 모델링을 통해 구조를 쉽게 파악하고 ThingML 코드, 즉 구현한 IoT의 설계와 분석을 할 수 있음을 확인하였다.

Java와 UML로의 컴파일을 통해, 다양한 플랫폼으로의 컴파일이 가능한 ThingML 코드의 특성을 확인할 수 있었다.

5. 평가 및 결 론

ThingML로 IoT 시스템을 모델링하고 구현한 결과 개발 과정에 다양한 추상성 수준을 확보할 수 있었다. 최고 추상인 사용 사례 수준, 네트워크 노드의 연동을 나타낸 상태 모델 수준, 센서에 구현될 코드 수준을 제공하여 다양한 전문가가 IoT를 구현하기 위하여 협력할 수 있다. 또한 추상 수준에 따라 IoT 설계 과정을 독립적으로 개발, 운영할 수 있고 각 결과물이 매끄럽게 잘 매핑 되는 장점을 사례 연구 결과로 발견하였다.

ThingML을 이용하여 코딩한 후 사용한 도구가 생성한 Java 코드의 품질은 더 많은 개선이 필요한 것으로 판단된다. 모델과 코드의 매핑에는 문제가 없으나 코드의 재사용성과 모듈화라는 측면의 단점이 발견되었다.

참고 문헌

[1] A. Taivalsaari, N. Technologies. “Roadmap to the Programmable World: Software Challenges in the IoT Era”, IEEE Software, Volume: 34(1), pp.72-80, 2017.

[2] D. Slama, F. Puhlmann, J. Morrish, R. Bhatnagar, Enterprise IoT, Oreilly, 2016.

[3] F. Zambonelli, “Key Abstractions for IoT-Oriented Software Engineering”, IEEE Software Volume 34(1): pp.38-45, 2017.

[4] B. Morin, N. Harrand, F. Fleurey. “Model-Based Software Engineering to Tame the IoT Jungle”, IEEE Software Volume 34(1): pp.30-36, 2017.

[5] 1. M.A. Razzaque et al., “Middleware for Internet of Things: A Survey,” IEEE Internet of Things J., vol. 3, no. 1, pp. 70-95, 2016.

[6] L. Atzori, D. Carboni, and A. Iera, “Smart Things in the Social Loop: Paradigms, Technologies, and Potentials,” Ad Hoc Networks, pp. 121-132, 2014.

[7] Frank Puhlmann, Dirk SlamaAn, “IGNITE: IoT Solution Methodology”, http://enterprise-iot.org/

[8] N. Harrand et al., “ThingML: A Language and Code Generation Framework for Heterogeneous Targets,” Proc. ACM/IEEE 19th Int’l Conf. Model-Driven Eng. Languages and Systems (MODELS 16), pp. 125-135. 2016.

Ontology를 이용한 지식 기반 지능적 강의 및 진로 추천 시스템 구축

최준영O , 이석원
아주대학교 소프트웨어학과
wnsdud0213@ajou.ac.kr , leesw@ajou.ac.kr

Construction of Intelligent Knowledge-based Lecture and Career Recommendation System Using Ontology

Jun-Young ChoiO , Seok-Won Lee
Ajou University Dept. of Software

요 약

기존의 정보 추천 시스템들을 분석하여 한계점을 분석한 결과, 학생들이 자신의 역량을 정량적으로 파악하기 어렵다는 점을 파악하였다. 본 연구에서는 이러한 문제를 해결하기 위하여 학생들의 수강 내역, 어학 점수 등의 학생의 다양한 정보를 이용해서 학생의 역량을 정량적으로 분석한 후, 학생들에게 가장 적합한 직무를 추천해 주는 시스템을 제안하고자 한다. 이를 통해서 학생들이 저학년 때부터 자기 주도적으로 자신의 경력 및 학습 계획을 설계할 수 있도록 보조하고자 한다.

1. 서 론

4차 산업 혁명의 도래와 함께 기업에서의 채용 직무도 세부화 및 다양화 되고 있으며, 해당 직무에 전문적인 인재를 위주로 선발하고 있다. 대학에서는 이러한 추세에 맞추어 학과별로 체계화된 커리큘럼을 제공하고 있지만, 학생들은 진로 결정과 선택을 하는데 있어서 실질적인 도움이 되지 않는다고 판단하고 있다.

이러한 원인 중 하나는, 학생들이 자신이 가지고 있는 현재 전공 역량에 대한 객관적인 수치에 대한 인지의 부재와 함께 자신이 원하는 직무에 대한 필요 역량을 어디서, 어떻게 얻어야 할 지 모르기 때문이다.

이러한 문제점을 해결하고자 학교에서는 학생들에게 학교 내 사회진출센터 내 인력과의 상담이나 진로 상담 검사 등 학생의 진로 설계를 위한 다양한 프로그램을 제공하고 있다.

그러나 이러한 프로그램들의 경우, 세부 전공 역량을 다루기 보다는 개인의 성격 분석 등에 그치는 한계점이 존재하여, 이를 통해 학생들이 구체적으로 “자신이 어느 직업의 어느 직무에 적합한가?” 등의 구체적인 답안을 얻기는 힘들다는 한계점이 존재한다.

공학 인증 상담의 경우에는 전공 과목을 기반으로 관련 분야에 대한 상담을 받을 수 있으나, 지도 교수의 분야와 다른 분야의 경우 구체적인 정보를 얻기가 힘들다는 한계점이 존재한다.

이러한 문제점 때문에 학생들은 주로 졸업한 선배들을 통해서 주로 직업 관련 정보를 얻게 되고 진로를 선택하는 경향이 높다. 이러한 경우, 본인의 역량과 능력을 면밀히 파악하지 않은 채 다른 사람의 정보에 수동적으로 의존할 수 밖에 없다. 그렇기 때문에 진로를 선택하게 되더라도 자신의 역량을 충분히 발휘하지 못할 수 있다는 잠재적인 문제점이 존재한다.

본 연구에서는 상기한 문제점들을 해결하기 위해서 학사 정보 및 기업 채용 공고를 분석해 OWL 언어로 기술된 지식 기반 지능적 강의 및 진로 추천 시스템을 제안하고자 한다.[1]

본 시스템을 통해, 학생들은 자신의 교과목 수강 데이터를 바탕으로 자신의 객관적인 역량과 졸업까지 필요한 교과목을 확인할 수 있다.

이를 기반으로 학생들은 자신의 보유 역량 및 직무 적합도 등을 표나 그래프 등을 통해서 쉽게 파악할 수 있으며, 이를 통해 학생 개개인이 졸업까지 자기 주도적인 학습계획을 설계 할 수 있도록 도움을 받을 수 있다.

학습 커리큘럼 이외에도, 학생들은 본 시스템을 통해서 자신이 원하는 직무, 기업에서 요구하는 다양한 전공 역량을 갖출 수 있는 다양한 활동(공모전, 해커톤, 관련 자격증, 어학 능력) 등을 시스템 상에서 추천받을 수 있기 때문에 교과 과목 이외의 전공 역량을 향상하는데 도움을 받을 수 있다. 추가적으로 이미 졸업한 선배들의 데이터를 통해 학생들은 자신만의 학습, 진로 계획을 수립하는데 지침으로 삼을 수 있다.

2. 관련 연구

2.1. 대만

대만의 경우, Chaoyang 대학교에서 Ontology 기반의 수강 추천 시스템을 개발하였다. 높은 미취업률과 함께 최초 구직활동에서 취업을 성공하는 학생들이 매우 적기 때문에 이를 해결하기 위한 대책으로 제안되었다. 학생들은 자신의 수강 내역과 함께 학교 교육과정을 종합한 데이터를 이용하여 자신이 이후 수강 시 도움이 될 것 같은 과목들을 추천 받을 수 있다. 그러나 본 시스템을 통해 취업이나 진로 설정 등의 구체적 학생 진로 설계를 하기에는 구체적인 역량 수치 등을 제시하지 못한다는 한계점이 존재하고 있다.

2.2. 미국

미국 MIT Electrical and Computer Science 학과에서는 학생이 학과에서 요구하는 졸업 요건을 만족하는 것을 돕기 위해서 교육과정과 목표를 문서화하여 그래프를 통해 보여주는 EECS Curricular Goal Map을 개발하였다.[3]

현재 Web 기반의 서비스로 제공되고 있으며, 학생들은 학과 교육과정과 교과목에 대해서 검토해 볼 수 있다. 또한, 성취도를 평가하는 기준들을 제시하기 때문에 목표 달성을 위한 내용들을 준비하는데 도움을 받을 수 있다. 하지만 학생 개인 수강 내역, 기타 역량 정보 등을 통해 사용자 개인이 어떤 직무에 적합한 지에 대해서는 알기 힘들다는 한계점이 존재한다.

3. 시스템 설계

3.1. Ontology Class Hierarchy

본 시스템의 Ontology 계층 구조는 다음 그림 1과 같다. 강의, 직업, 스펙, 직무 역량 등의 Class 하에 해당 속성들이 포함되며 리소스 간 Property 설정을 통해 선수과목, 후수과목 등의 다양한 관계에 대해서 인간이 사고하는 것처럼 고차원적인 질문에 대한 답을 도출할 수 있도록 하였다.

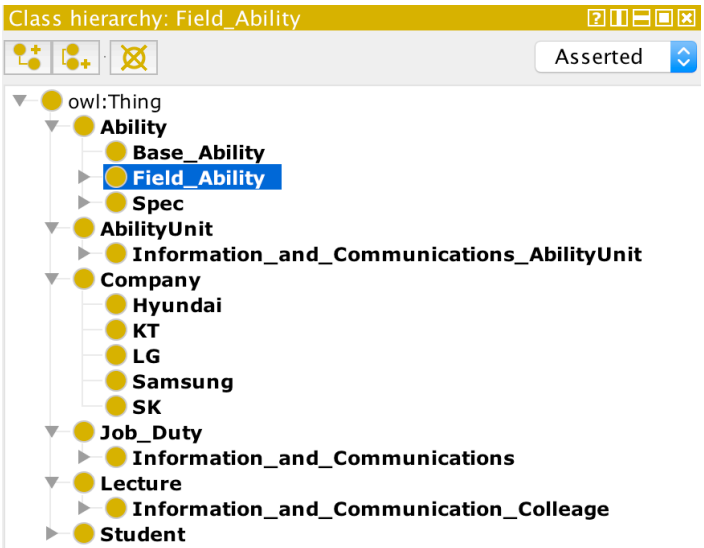


그림 1 Ontology Class Hierarchy

3.2. 전체 시스템 구조

전체 시스템 구조는 다음 그림 2와 같은 구조를 가지고 있다. 사용자는 Web Application을 통해서 자신이 원하는 질문을 SPARQL Query로 Server에 송신하게 된다. 이후, Server는 mongoDB 기반의 외부 데이터 베이스인 mLab에 저장되어 있는 사용자 개인 데이터와 SPARQL Query를 비교하여 사용자에게 적합한 데이터를 다시 Web Application으로 보내주게 된다.[2]

본 시스템은 서버와 클라이언트 그리고 사용자의 개인 데이터를 저장하고 있는 외부 데이터 베이스로 구성되어 있다. Server 구축을 위해 Node.JS, MongoDB, ExpressJS를 이용하였으며, Web 구축은 html/css, Bootstrap, Javascript를 활용하였다. 또한, 사용자 개인 데이터를 저장하기 위하여 MongoDB 기반의 Cloud인 mLab을 적용하였으며, Ontology OWL 파일을 저장하기 위해서 Jena Fuseki Server를 이용하였다.

Web Application의 경우 반응형 웹 구조를 적용하여 구현하여 날로 증가하는 다양한 모바일 디바이스, 태블릿 등 다양한 실행 환경에서도 원활한 시스템 이용을 할 수 있게 할 수 있게 하였다.

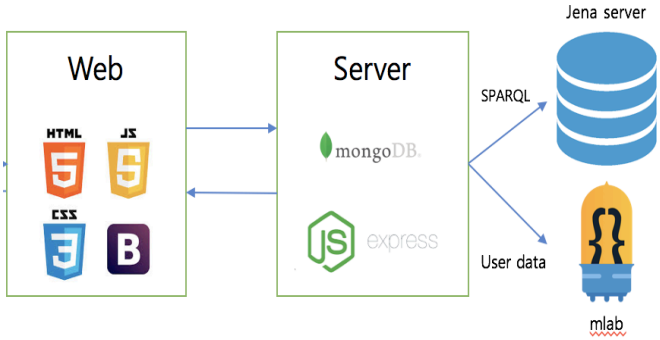


그림 2 System Architecture

3.3. 시스템 구현

본 논문의 Web Application의 메뉴 구성은 과목 정보, 직무 역량, 기업 채용 직무, 개인 역량 확인 페이지로 구성되어 있다.

개인 역량 확인 페이지의 경우 사용자들이 가입 시 작성한 수강 이력 데이터를 기반으로, 전체 전공 과목 대비 자신이 가진 역량을 Ontology를 통해 정량적으로 측정해 수치화한다.

사용자들은 그림 3과 같이 자신이 현재 어느 직무에 가장 적합한 지에 대해서 알 수 있다. 추가적으로 졸업까지 필요한 학점, 앞으로 들어야 할 전공 과목 등을 도표, 그래프 등을 통해 쉽게 인지할 수 있다.

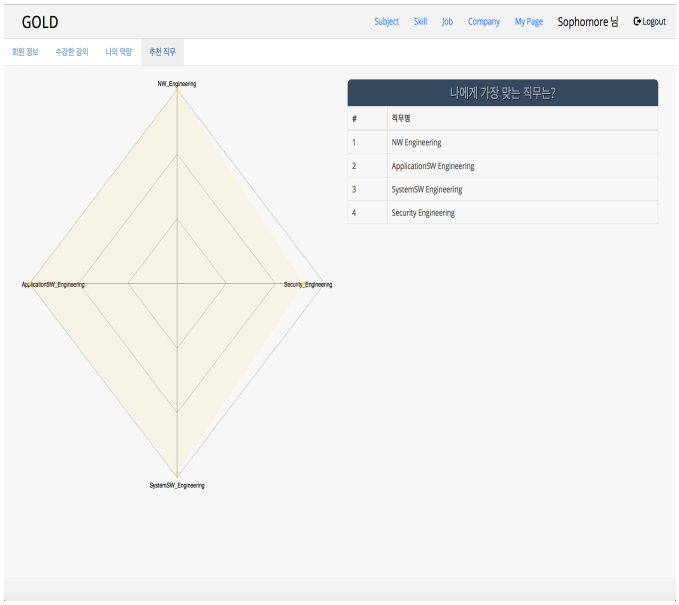


그림 3 개인 역량 별 직무 추천

과목 정보 페이지의 경우 다음 그림 4와 같이 화면 우측에 학생들이 가장 관심있어 하는 질문들을 버튼 형식으로 제공하여 자신이 관심이 있는 과목들을 통해 얻을 수 있는 세부 역량 항목 등을 확인할 수 있으며, 이외에도 관련 역량을 보유한 사람을 채용하는 기업 정보 등의 다양한 정보를 확인할 수 있다.

전체 과목들은 Directed Forced Graph를 활용하여 학과 커리큘럼과 동일하게 각 과목 별 연관 관계를 보여주고 있다. 학생 개인이 수강한 과목과 수강하지 않은 과목을 다르게 표현하여 다음 수강 과목을 선택하는데 도움을 받을 수 있게 구현하였다.

이 페이지를 통해 학생들은 개인 역량 확인 페이지를 통해 얻은 정보 등을 활용하여, 자신에게 최적화된 수강 계획을 짜는데 도움을 얻을 수 있다.

그림 4 개인 별 전공 과목 진행 확인

기업 채용 직무 페이지의 경우 아래 그림 5와 같은 화면 구성을 보이고 있다. 본 시스템에서는 삼성, SK, LG, 현대, KT 등 주요 대기업에서 IT 관련 업무를 채용하는 계열사들의 채용 정보를 기반으로 구성하였다. 같은 기업 별로 같은 색의 Node를 가지고 있으며, Cluster화 시켜서 사용자들이 기업 별로 식별하기 쉽게 구성하였다.

각 계열사 Node를 클릭하게 되면 해당 계열사에서 채용하는 세부 직무들을 확인 할 수 있으며, 추가적으로 해당 직무의 요구 역량 등을 확인할 수 있다.

예를 들어 사용자가 '현대 자동차'를 선택할 경우 해당 계열사에서 모집하는 'IT 전략', '응용 SW 개발' 등의 다양한 직무 별 역량을 확인할 수 있으며, 해당 역량을 얻기 위한 과목 등을 확인해 해당 역량을 기르기 위한 수강 계획을 세우는데 도움을 받을 수 있다.

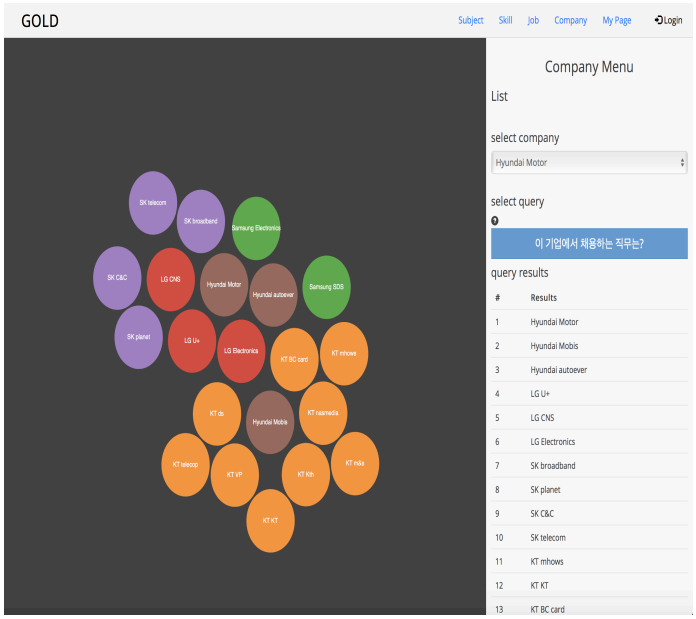


그림 5 기업 별 채용 직무 확인

직무 역량 확인 페이지의 경우 학과 수업 계획서를 기반으로 학과 과목 수강 시 얻을 수 있는 '코드 분석력', '특정 기술 구현 능력' 등의 다양한 직무 역량 항목들을 아래 그림 6과 같이 제시하고 있다.

학생들은 본 기능을 통해서 자신이 지원하고자 하는 기업에서 채용하는 직무의 필수 역량을 얻을 수 있는 과목들을 확인하거나 특정 기술에 대한 지식을 얻기 위해서 어떤 과목을 수강해야 하는 지 쉽게 확인할 수 있다.

전공 역량 이외에도, 기업에서 필수로 요구로 하고 있는 TOEIC, OPIC 등 다양한 어학 역량들을 시스템에 반영하여 학생들이 본 시스템을 통해서 자신의 역량에 맞는 기업들을 찾아 지원하거나, 취업에 필요한 역량을 채우기 위한 자기주도 학습 계획 설계를 할 수 있게 유도하였다.

그림 6 직무 역량 확인

4. 결 론

이미 다양한 국가에서 Ontology를 이용한 수강 지도 시스템이 연구 및 개발되었고 실제 활용되고 있으나, 현실적으로 기업의 채용 공고와는 다소 차이가 존재하기 때문에 학생들이 이를 통해서 실제 자신의 학습 계획을 세우는데 효율적인 도움을 준다고 보기에는 무리가 있다.

본 논문을 작성하기 이전, 현재 학교에서 취업 준비를 하거나 재학중인 학생들을 대상으로 설문조사를 실시한 결과 학생들이 가장 중요하다고 생각한 부분은 자신이 가지고 있는 현재 개인 역량의 객관적인 수치 파악이었다. 그렇기 때문에 현재 기업들의 채용 공고를 분석하여 학생들이 수강한 교과목 이외에도 기업들이 요구하는 다양한 어학, 자격증 어학, 자격증 보유 여부에 따른 학생 개개인의 역량을 수치화 하여 기업의 채용 직무 적합도 등을 확인할 수 있게 하였다. 또한 이를 통해서 학생들이 필요한 역량을 얻을 수 있는 교과목을 추천을 해 줄 수 있는 시스템을 연구하였다. 본 시스템을 이용해 기존 취업한 학생들의 수강 정보를 입력해 본 결과, 회사의 직무와 유사한 역량 보유 수치를 확인해 볼 수 있었다.

하지만 당장은 학생 데이터가 부족하다는 한계점이 존재하기 때문에 다양한 학생들의 데이터를 일반화하여 통계적으로 신뢰도가 높은 참고 자료를 사용자들에게 제공하기는 힘들다는 한계점이 존재한다.

또 다른 한계점으로는 소프트웨어학과만을 대상으로 시스템을 설계하였기 때문에 현재 새롭게 생겨나고 있는 'IT 신사업 기획'등 타 전공과 융합된 직무에 대해서는 사용자들에게 아직까지 단일 전공에 비해 신뢰도 높은 답변을 제시하기가 어렵다는 점과, 마케팅이나 경영 관리 등 소프트웨어학과가 아닌 타 전공에서 얻을 수 있는 역량을 요구하는 직무의 경우 답변을 줄 수 없다는 한계점이 존재하였다.

이러한 한계점을 해결하기 위해서는 향후 학교 학사 시스템과의 연동을 통해 다양한 전공의 학과 커리큘럼, 기업의 채용 직무, 다양한 분야로 취업한 졸업생들의 수강 데이터 및 역량 정보를 수집한 후, 머신 러닝 및 딥 러닝 등 다양한 기법을 통해 데이터 분석 및 활용을 하여 사용자들에게 직무 및 전공 별로 일반화된 지침을 제공할 수 있을 것이다. 또한 학년이나 학기 단위로 세부적인 학습 지침을 추천해 줄 수 있도록 시스템을 확장할 수 있다면 본 시스템을 이용하는 학생들이 자신의 진로를 설계하는데 더 큰 도움을 얻을 수 있을 것이라고 생각한다.

Acknowledgement

본 연구는 과학기술정보통신부 및
정보통신기술진흥센터의 SW 중심대학지원사업의
연구결과로 수행되었음"(20150009080031001)

5. 참고문헌

[1] Deborah, M and Frank, V, H, "OWL Web Ontology Language Overview," W3C Recommendation.

[2] M.S Kang, "A study on constructing a Job competence ontology knowledge Base for self-directed career path recommendation", Ajou University, Suwon, 2016년도 한국통신학회 하계종합학술발표회 논문집 (Vol.60), p553-554, June. 2016.

[3] Ward, S. (2017). *Curricular Goal Map.* [online] Gmap.csail.mit.edu.
Available at:
https://gmap.csail.mit.edu/gmap/public.html
[Accessed 27 Oct. 2017].

Keystroke Dynamics를 기반으로 한 OTP 게임계정 보안 시스템에 대한 이론적 강화방법

안정수 ○, 김상민, 신정우, 정재웅

경희고등학교

bulebed1754@gmail.com, hacuman7@naver.com, nayo1997@naver.com, bestamicus@naver.com

Theoretical way of reinforcing established online gaming security system based on OTP using Keystroke Dynamics

Jung-Soo Ahn○, Sang-Min Kim, Jung-Woo Sin, Jae-Woong Jeong
Kyunghee Highschool

요 약

본 논문에서는 기존 온라인 게임 계정 보안에 있어 높은 안정성으로 인해 많이 사용되는 OTP 보안 시스템의 해킹 시나리오에 대해 살펴보고 분석하여 이를 생체인식 기술 보안 시스템과 결합시켜 보완한 이론적 보안 시스템인 Keystroke Dynamics를 활용하여 OTP 시스템을 보완을 할 수 있다는 점에 대해 서술하고자 한다.

1. 서론

지금까지 게임 시장에서 가장 널리 이용되고 안정성이 높은 보안 방법 중 하나인 것은 OTP(One Time Password) 방식이다. 이 OTP 방식 보안성이 높다는 장점이 있어 많은 인터넷 뱅킹, 게임 등 개인 정보가 중요시 되는 많은 분야에서 이용되는데, 해커들이 이러한 대중성을 이용하여 MITB(Man in the Browser), MITM(Man in the Middle), MITPC(Man in the PC), Key Logging 등 여러 가지 방법을 통해 해킹하는 사례가 점점 증가하는 추세를 보이고 있다. 이러한 해킹은 단순히 개인적 차원의 피해부터 기업에게 손해를 끼치는 등의 사회적 차원의 파장까지 초래할 수 있다는 점에서 해킹공격을 막을 보안 시스템의 필요성이 더 고조된다. 이 논문에서 해커들의 해킹 시나리오에 대해 알아보고 분석함으로써 OTP 보안 방식에 키스트로크 다이나믹스(Keystroke Dynamics)를 기반으로 한 biometrics를 입혀 OTP를 보완하는 이론적 보안 기술을 이 논문에 기술하였다.

2. 관련 연구

2.1 OTP (One Time Password)

OTP란 난수 알고리즘을 이용하여 생성된 일회성 패스워드를 이용하는 사용자 인증 방식이다. 사용자는 접속 시마다 seed값을 시간으로 한 난수 알고리즘을 이용하여 생성된 일회성 비밀번호를 통해 접속하게 된다. 이 시드값은 보통 시간 동기나 해시체인을 사용하여 매번 일정 시간마다 바뀌게 된다. OTP는 현재 게임 계정 보안, 인터넷 뱅킹 보안 등 여러 분야에서 폭넓게 사용되고 있다.

2.2 MITM(Man in the Middle), MITPC(Man in the PC), MITB(Main in the Browser)

MITM는 해커가 사용자가 서버에 정보를 전송하는 과정에서 중간에 개입하여 정보를 탈취하는 해킹 방식을 말한다. MITPC공격은 사용자가 서버에 접속하는 경로를 타깃으로 삼아 공격하여 접속에 필요한 키 값을 탈취하는 공격을 말한다. 이 두 방식의 주 원리로는 사용자의 파일을 변조하여 사용하는 방법으로 탈취하려는 사용자의 정보를 입력하는 페이지에 거짓으로 만든 페이지를 띄워 사용자 입력하면 정보가 자동으로 탈취되는 방식이다. MITB는 브라우저에 악의적인 프로그램을 설치하여 사용자에 대한 정보를

탈취하는 공격을 말한다. 예로 키보드, 마우 로깅, 화면 캡쳐 등 운영체제 기반에서의 사용자가 액세스하는 모든 경로가 공격 대상이 되는 방식이다. [1]

2.3 바이오 매트릭스(Biometrics)

바이오 매트릭스는 생체 인식과 같은 개념으로 통하며 하나 이상의 고유한 신체적, 행동적 형질에 기반하여 사람을 인식하는 방식을 폭넓게 이르는 말이다. 이러한 생체 인식 기술은 크게 2가지 종류로 구분할 수 있는데, 그 종류에는 신체적 특징을 이용한 생체 인식 기술(지문인식, 홍채인식, 심장 박동수 인식, 뇌파 인식 등)과 행동적 특징을 이용한 생체 인식 기술(키 스트로크, 걸음걸이)이 있다. 이 중 지문 인식 기술과 홍채 인식 기술은 이미 스마트폰 사용자 인증과 같은 부분에서 실용화되어 사용되고 있다. 이 논문에서는 생체 인식 보안 기술 중에서도 키 스트로크 기술을 이용한 보안 시스템에 대해 다루고자 한다. 그 이유는 신체적 생체 인식 기술을 사용한다고 하면 지문과 홍채인식 같은 신체 인식 같은 경우에는 한번 해커가 신체 정보를 해킹해서 그 정보를 탈취하여 사용하게 된다면 홍채와 지문과 같은 신체부분은 사람마다 고유한 값으로 불가변성을 지니기 때문에 해킹 당하게 된다면 더 이상 보안 장치로서의 효력을 가지지 못하게 된다. 그래서 이 키스트로크 기술을 사용한다면 이러한 정적인 신체부위를 이용한 biometrics 기술에 비해 상대적으로 해킹 당했을 시에 비밀번호를 입력할 때에 있어서만 적용되는 새로운 Keystroke Dynamics를 생성하는 방안으로 대처가 가능하다. 서버에 등록되어 있는 Keystroke Dynamics를 지속적으로 바꾸게 된다면, 더 이상 서비스를 이용 중인 사용자가 정상적 사용자인지 지속적 모니터링은 어려워지겠지만, 신체 인식에 비해 상대적인 유연성을 가진다고 할 수 있다. 또한 홍채 인식이나 지문 인식은 인식 소프트웨어만 추가적으로 필요할 뿐만 아니라 추가적인 외부 하드웨어 또한 필요로 하게 되는데, 키스트로크 인식 기술은 소프트웨어만을 이용하여 인식이 가능하다는 점에서 더 경제적이고 보다 간편하고 실용적인 기술이라고 할 수 있다. [2]

2.3.1 키 스트로크 인식(Keystroke recognition)

Keystroke 보안의 기본 개념은 키보드에 패스워드를 입력할 때의 한 키를 지속적으로 누르는 시간, 문자를 간의 이동 시간에 의해 발생되는 개인 간의 문자의 시간차를 패턴화하여 해커에게 해킹을 당해 패스워드가 노출 되거나 허가 받지 않은 사용자의 무단접근 하였을 때 암호가 유출되어 일차적인 보안망을 통과하더라도 개인이 가지고 있던 자신만의 keystroke 시간패턴 인식이 다르면 로그인(log-in)이 안되도록 하는 보안이다. [3], [4]

2.4 OTP 보안의 약점과 보완책

본 논문에서는 기존의 OTP 보안 방법을 활용한 온라인 계정 보안 방법에 대한 해킹 시나리오에 대해 살펴보고 이를 생체 인식 기술을 통해 보안한 이론적 보안 방법에 대해 제안할 것이다.
먼저 OTP 생성절차는 다음과 같다. 사용자의 요청에 의해 새로 생성된 OTP값을 사용자가 입력하게 되면 인증 서버에서는 이 값이 서버에서 계산한 값과 동일한지를 확인하게 된다. 동일한 것으로 확인되면, 이후 사용자는 해당 사이트에서 필요로 하는 서비스를 제공받게 된다.
이러한 OTP 보안을 해킹하는 일반적 시나리오에는 피싱 사이트를 개설한 후, MITPC와 MITM를 병행한 공격을 통해 하는 해킹, 피싱 사이트가 아닌 같은 사이트에 가짜 폼을 만들어 사용자를 속이는 MITB 공격을 통한 해킹, HttpSendRequestA/W 후킹을 통한 해킹이 있다. [5]

MITPC & MITM : MITPC 공격을 통해 입력되는 ID, Password를 비롯한 개인정보를 탈취한다. 이 후, 사용자가 선행하여 개설해둔 피싱 사이트에 OTP값 입력할 때마다 MITM공격 수행하여 OTP값 탈취하게 된다면, 사용자의 계정에 접근할 수 있게 된다.
MITB : MITB 공격은 MITM 공격과는 달리 피싱 사이트를 이용하지 않는다. 대신 사용자의 웹 브라우저에 악성 프로그램을 설치하는 작업을 필요로 한다. 이 프로그램을 통해 만들어진 가짜 입력 창에 사용자로 하여금 정보를 입력하게 하여 OTP 코드를 탈취하게 된다.

HttpSendRequestA & HttpSendRequestW Hooking : HTTP, HTTPS, FTP등의 인터넷 프로토콜을 다루는 파일인 wininet.dll 의 API 함수 중 각각 아스키 코드와 유니코드로 빌드된 메세지를 HTTP서버에 전송하는 함수인 HttpSendRequest A/W를 후킹(hooking)하면 HTTP로 통신하는 모든 쿼리를 감청할 수 있게 되는데, 이 때 만들어지는 쿼리는 유저 클라이언트 내부에서 생성된 것이므로 암호화되지 않은 상태이기 때문에 해커가 쿼리 단에서 ID, password, OTP값을 손쉽게 탈취할 수 있게 된다. 이러한 해킹 방법을 이용한 사례로는 중국계 해커들이 넥슨 이용자의 계정을 탈취한 사건이 있다.

이러한 방법들을 통해 OTP 값과 ID, Password 정보를 탈취하더라도 사용자가 먼저 OTP 값을 먼저 입력하게 된다면 해킹은 실패하게 된다. 따라서 위 방법들을 통해 정보를 얻은 후 해커들은 사용자보다 앞서 OTP 값을

입력하기 위해 HttpSend-RequestA/W를 후킹하여 그 안에서 OTP값을 변조한 후 이 값을 서버로 전달되게 한다. 이로 인해 서버에서는 사용자가 입력한 OTP값이 변조되어 전달되므로 사용자 인증이 처리되지 않게 된다. 이를 통해 해커는 OTP 코드의 일반적 유효기간인 60초의 시간을 획득하게 되고 그 시간 동안에 탈취한 OTP값을 입력하여 서버에 접속하게 된다.

해커들은 이러한 방법을 통해 OTP의 맹점을 찾아 사용자의 계정을 해킹하게 된다. 위에서 분석한 해킹 시나리오를 토대로 OTP 보안에 대해 생각해보자면 모두 OTP 값이 해커에게 노출되더라도 부차적인 보안 방법으로 접속 권한을 얻지 못하게 한다면 차단이 가능하게 된다는 생각을 할 수 있다. 이 때 필요한 것이 개인을 단순한 입력 코드가 아닌 다른 수단으로 인식할 수 있는 보안 시스템이다. 이 조건에 맞는 보안 기술로는 생체인식 보안이 있다.

생체인식 보안은 차세대 보안 기술로 각광받고 있는 분야로 이와 관련된 기술 중 가장 널리 확산된 기술로는 위에서 제시한 홍채인식, 지문인식을 통한 보안이 있다. 하지만 이러한 바이오 매트릭스는 이미 널리 확산되어 있는 만큼 다른 분야의 보안 시스템과 접점을 생길 가능성이 있어 특수한 보안 시스템의 필요성에 대해 인식하게 되었다. 게임 보안에 대해 생각을 해보았을 때 게임 환경에서는 키스트로크 과정이 지속적으로 이루어지게 된다. 우리는 이러한 게임환경의 특징을 바탕으로 게임 계정 보안기술에 있어 생체인식 보안 기술 중 키스트로크 다이나믹스(Keystroke Dynamics)를 기반으로 한 바이오 매트릭스의 활용 가능성에 대해 생각해볼 수 있었다.

키스트로크 다이나믹스(Keystroke Dynamics)란 간단히 말해 개인이 문자열을 입력하는 행동적 형질이라고 할 수 있다. 먼저 키스트로크를 이용한다면 네트워크에 접속한 사용자들은 여러 문서작성과 같은 타이핑을 통하여 간단한 키스트로크 다이나믹스 수집 프로그램을 설치하는 것만으로도 사용자의 행동 패턴이 모니터링이 가능하게 된다. 그리하여 키스트로크는 다른 프로그램들보다 보다 간편하게 사용자들이 사용할 수 있으며 입력 처리 속도가 빠르기 때문에 모니터링까지 걸리는 시간이 오래 걸리지 않아 빠르게 작업할 수 있다. 또한 키스트로크 다이나믹스를 사용하게 된다면 입력 언어에 대해 제한이나 특수한 영향을 받지 않는다. 다른 시스템에서는 그 시스템에서 사용하는 문자를 지정해서 사용하지만 키스트로크 기법에서는 키의 압력으로 눌리는 깊이를 측정하기 때문에 입력문자에는 제한이 없다. 그래서 키스트로크 다이나믹스는 개개인의 고유한 행동속성임을 확인할 수 있다. 하지만 이러한 방법은 개인마다 고유한 시스템 속성을 소유하고 있기 때문에 이 키스트로크 다이나믹스 기법을 이용할 때 이 기법의 고유적인 알고리즘에 대해 개인마다 인증하는 측면에서 성능이나 여러 방면에서 차이가 있는 것을 확인할 수 있다. 이러한 측면에서는 향후에 개인마다 최적화된 알고리즘을 만들어내는 방법에 대한 연구가 필요할 것이라고 생각된다. [6]

3. 해킹 공격에 대한 이론적 방어 방법

여기서는 본론에서 다룬 해킹 공격 방법 중 후킹 방법을 통한 해킹 공격을 예시로 들어 방어하는 방안을 제시하고자 한다.
일단 처음 정상적 사용자가 계정을 생성할 때에 있어 패스워드를 등록하기에 앞서 반복적으로 입력을 하도록 하여 이 사용자에 대한 Keystroke 데이터를 수집한다. 모은 자료를 이용하여 이 사용자의 고유한 Keystroke Dynamics를 생성한 후 비밀번호와 함께 Keystroke Dynamics 데이터를 함께 등록한다. [그림1]

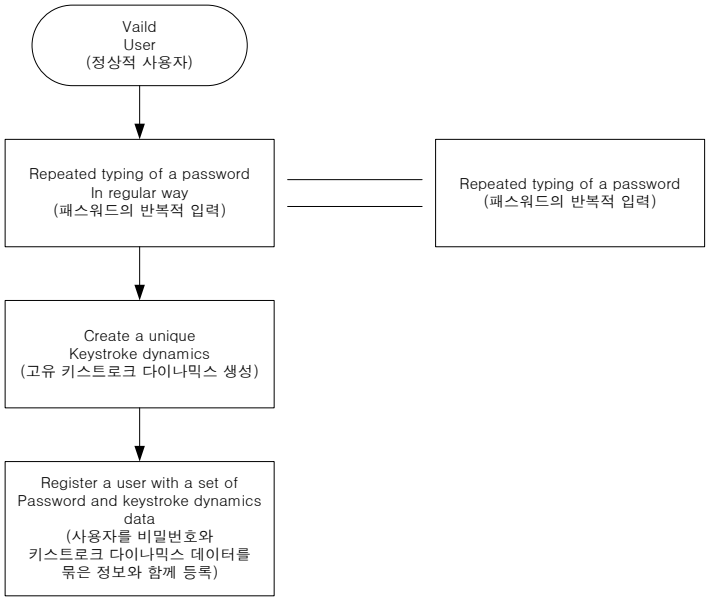


그림1. Step of user registration

이후에 해커가 후킹을 통해 키를 탈취하여 계정에 접속하려는 시도를 하게 된다면, 서버에 구축된 Keystroke Dynamics Classifier에 의해 Keystroke interval이 분석되게 된다. 해커는 사용자의 고유한 Keystroke Dynamics를 모르므로 키 값을 알지라도 계정에 대한 접근이 거부되게 된다. 그 동안 사용자는 코드를 입력하여 로그인에 성공하게 되고 이 때 사용된 OTP는 폐기되게 되어 해커는 해킹에 실패하게 된다. [그림2]

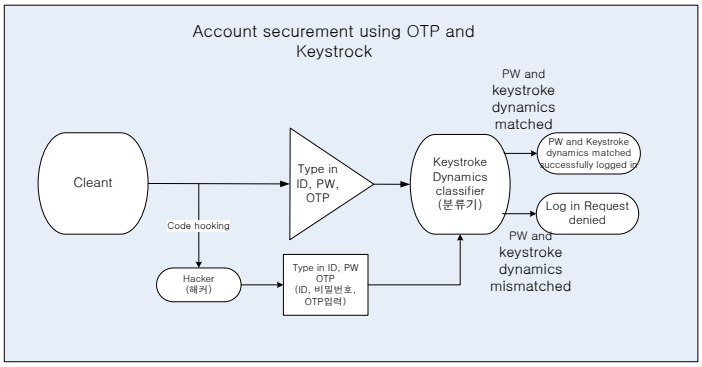


그림2. Security system using keystroke recognition algorithm

기존의 OTP만을 이용한 보안 시스템에 대한 Keystroke Dynamics를 기반으로 한 보안 시스템의 개선점에 대해 서술하자면 해커로 하여금 단순한 코드 탈취만을 통한 해킹의 가능성을 차단할 수 있게 된다. 다시 말해 클라이언트 해킹만을 통해 계정에 접속하는 것을 막을 수 있다는 것이다. 이는 사용자의 입장에서 해킹에 대한 복구 서비스를 받는 데에 있어 효과가 있다는 장점이 있다. 기존의 보안 시스템에서 OTP 코드가 탈취당해 계정이 해킹 당했을 경우에 게임 회사 측에서 계정 도용의 가능성 등을 근거로 하여 사용자에게 책임을 전가하여 해킹에 대한 계정 복구 서비스를 받지 못하는 경우가 많이 존재한다. 하지만 키스트로크 바이오 매트릭스를 적용하게 된다면 계정 도용의 가능성을 배제함으로써 복구 서비스를 받지 못하는 피해자의 수를 줄일 수 있다. 따라서 Keystroke Dynamics를 기반으로 보안 시스템은 계정에 대한 접근 가능성을 줄이고 피해자의 수를 줄인다는 점에서 그 의의가 있다고 할 수 있다.

4. 결론

Keystroke Dynamics를 이용한 키스트로크 인식 보안 기술은 지문 인식이나 홍채 인식과 같은 Biometrics 보안 기술들과 비교하여 비해 해킹 후의 대처에 유연하다는 장점을 지닌다. 또한 부가적 하드웨어를 필요로 하지 않는다는 점에서 이용자 입장에서는 더 경제적이다. 다른 보안 프로그램과 비교하였을 때는 입력 처리 속도가 빠르기 때문에 사용자의 상태를 모니터링하는 데에 있어 많은 시간이 소요되지 않는다는 점과 입력 언어에 대해 제약을 받지 않는다는 점에서 더 실용적이라고 할 수 있다. 하지만 이러한 Keystroke Dynamics 인식 기술은 아직 홍채 인식이나 지문 인식과 같이 보편적으로 상용화되어 있지는 않은 것이 실정이다. 또한 변화 가능한 데이터를 이용한 보안 기술인 만큼 오차범위에 따라 정상적 사용자 또한 계정에 대한 접근 승인을 받지 못하게 되는 상황이 발생될 수도 있다. 따라서 이 기술을 게임 계정 보안에 있어 보편화시키기 위해서는 각각의 고유한 Keystroke Dynamics를 지닌 사용자에 대해 균등한 수준의 보안 서비스를 제공할 수 있도록 하는 알고리즘의 개발이 요구될 것이다. 이러한 알고리즘의 개발에 대한 연구가 선행되어야 한다는 점에서 Keystroke 인식 보안 기술은 현재의 상황에서 바로 적용할 수는 없는 미래적 기술이라고도 할 수 있으나, 앞에서 다룬 Keystroke 인식 기술을 활용하였을 때의 이점들을 생각해본다면 개발할 가치가 충분히 입증되었다고 본다.

또한 추가적으로 키스트로크 보안 기술의 가능성에 대해 생각해본 후 이 보안 기술을 보완할 수 있을 3단계 보안 방안에 대해 생각해보았다.

Ⅰ. ID/PW 입력 시 GPS 위치정보, IP주소를 기반으로 근접한 위치에 있을 때 접근 제어를 수행 (클라이언트의 물리적인 위치와 IP 주소 정보에 기반을 둔 사용자 인증을 통하여 악의적인 접속자의 공격을 사전에 예방함)

Ex) GPS 위치정보는 서울 동대문구 인데 IP주소는 중국 베이징일 경우 1차 접근 거부함

Ⅱ. 등록된 스마트폰 혹은 PC카메라를 이용해 안면인식 및 홍채인식을 활용한 FIDO(Fast Identity Online)시스템을 활용하여 접근 제어를 수행한다. (단, 안면인식 데이터베이스와 홍채 데이터베이스의 철저한 보안이 선행되어야 한다는 것을 전제조건으로 한다.)

Ⅲ. 게임 사에서 일반적으로 널리 쓰이는 OTP 형태인 토큰 형식이 아닌 렌즈 형태의 웨어러블 디바이스로 개발한다. 이용자는 게임 사로부터 구매한 공공 OTP 렌즈를 착용하고 보안코드를 60초 이내에 입력하게 되고 이 보안 시스템은 양방향 인증을 통해 접근 제어 기능을 수행하게 된다.

위 Ⅰ, Ⅱ, Ⅲ 단계에서 제안한 보안 방법을 각각 1차적, 2차적, 3차적 접근 제어를 수행하도록 하여 위 논문의 내용에서 제시되었던 Keystroke Dynamics를 활용한 보안 시스템과 결합시킨다면 보다 높은 단계의 보안이 가능해지며 해킹에 대한 위험을 원천적으로 차단 가능할 것으로 생각된다.

5. 참고 문헌

[1], [5] 강병탁, 김휘강. MITM 공격과 리버스 엔지니어링을 이용한 OTP 적용환경의 취약점 연구. 정보보호학회논문지, 21(6), 83-99. (2011).

[2] 육모세, 김희연, 심혜린. "생체인식기술 기반 개인인증수단에 따른 사용자 인식." 한국콘텐츠학회논문지, 11-19 (2016.11).

[3] 조태훈. 키스트로크 인식을 위한 패턴분류 방법. 한국정보통신학회논문지, 10(5), 956-961. (2006).

[4] 황성섭, 조성준, 박성훈. "키스트로크 다이나믹스 분석을 이용한 모바일 사용자 인증." 한국경영과학회 학술대회논문집, 652-655 (2006.11).

[6] 강필성, 조성준. 자유로운 문자열의 키스트로크 다이나믹스를 활용한 사용자 인증 연구. 산업공학 (IE interfaces), 25(3), 290-299. (2012).

비전문가들에 의한 소프트웨어 설계

고대권˙ 김승섭, 김상훈, 성주혁, André van der Hoek, Ayushi Rastogi, 이석원

아주대학교, 국민대학교, 어바인 캘리포니아 대학교
kingkdg@ajou.ac.kr , kss5662@ajou.ac.kr, jesskim1126@gmail.com,
sym92540@gmail.com, andre@ics.uci.edu, ayushir@uci.edu, leesw@ajou.ac.kr

Software Design by the Ordinary People

Daegweon Ko, Seungsup Kim, Sanghun Kim, Juhyeok Seong, André van der Hoek,
Ayushi Rastogi, Seok-Won Lee
Ajou University, Kookmin University, University of California, Irvine

요 약

누군지 모르는 많은 작업자들이 전통적으로 자유로운 공간에서 수행했던 작업으로부터 정보를 얻는 행위인 크라우드 소싱은 소프트웨어 공학 연구를 수행하는데 중요한 수단으로 여겨지고 있다. 예를 들어, 사람들은 누구나 프로젝트에 참여할 수 있는 깃허브와 같은 웹사이트에서 크라우드 소싱을 쉽게 찾아 볼 수 있다. 하지만 지금까지, 크라우드 소싱은 소프트웨어 개발의 제한된 부분에만 적용되어 왔다. 이 연구에서는 개인이 하나의 설계 문제의 필수적인 면을 정의하는 결정 지점들의 집합을 명시하는 프로세스에 기여할 수 있는 새로운 협력적[4] 소프트웨어 설계[5] 환경을 만들어보고자 했다. 특히 넓은 군중 참여를 통해 결정 지점들의 집합을 생성하는 작업 흐름을 제시한다. 이 작업 흐름의 첫 단계는 군중들이 웹 어플리케이션을 통해 요구사항들로부터 중요한 단어를 추출해내는 것이다. 두 번째 단계는 그 단어들을 사용해서 결정 지점들을 만드는데 두 번째 군중을 이용하는 것이다. 지금까지 첫 단계에 대해 연구가 진행되었으며 중요한 고려사항에 대해 논의하고, 그리고 다음에 진행될 실험을 수행하기 위해 개발한 예비 툴을 제시한다.

1. 서 론

비전문가들의 지식은 많은 분야에서 점점 더 중요해지고 있다. 비전문가는 해결하고자 하는 문제에 대해 전문지식을 가지고 있지 않는 사람부터 전문가의 전문지식과 버금가는 지식을 가지는 사람을 포함하며, 그 문제에 대해 전문적으로 연구하지 않는 사람들을 말한다. 각 비전문가들은 전문가들에 의해 수행된 원래의 작업과 다르게 분리된 문제들의 부분을 해결한다. 많은 비전문가들이 분리된 문제들을 해결했을 때, 그들은 전문가들로부터 오지 못할 수도 있는 다양하고 서로 다른 아이디어들을 주었다[2,4,6]. 위키피디아는 수 백만명의 개인의 다양하고 전문적인 정보를 가지는 아이디어들로 만들어졌다. 깃허브 또한 창의적이고 다양한 아이디어들을 얻을 수 있도록 많은 개발자들이 오픈 소스 프로젝트에 참여하고 있다. Topcoder는 소프트웨어 개발에서 크라우드 소싱을 관리하는 동시에 각 개발자를 경쟁하게하는 웹 사이트이다. 이 회사는 크라우드 소싱 센터 (crowdsourcing center)의 역할을 수행하여 회사가 사용자 정의 디자인 또는 개발 작업을 Topcoder 사용자에게 할당 할 수 있다. 스택 오버플로우에서는, 사람들이 많은 다른 사람들의 문제를 해결하는데 기여하고 제안된 해결책들은 그들의 문제들을 푸는데 다양성과 적합성을 가진다. 위의 모든 예들은 크라우드 소싱으로, 개인들 또는 회사들이 다양한 아이디어들과 스킬들을 얻기 위해 많은 사람들의 기여를 이용하는 것이다[2,5,6]. 그리고 크라우드 소싱을 통해 많은 사람들로부터 다양한 아이디어를 얻을 수 있을 뿐만 아니라 시간, 비용의 측면으로도 더 많은 효과를 얻을 수 있다. 한 명의 전문가가 높은 비용을 받고 문제 해결을 위해 많은 시간을 들이는 것과 달리 많은 비전문가들이 개개인의 지식을 기여했을 때 그 지식들이 모여 한 명의 전문가가 문제를 해결하는 것보다 더 빠르고 낮은 비용으로 문제를 해결할 수 있다. 이렇게 소프트웨어 설계[5]에서 효과적으로 구현의 기초를 구성 할뿐만

아니라 소프트웨어를 재사용하고 유지 보수하기 쉽게 만드는 데 중요한 포인트로서 소프트웨어 설계에서 크라우드 소싱을 사용하려는 많은 시도가 있었다. 하지만 많은 비전문가들이 전통적인 작업 흐름을 따를 때 소프트웨어 설계에서 크라우드 소싱을 사용하기 힘들었다. 왜냐하면 전통적인 작업 흐름은 너무 어려워서 비전문가들이 그들에게 할당된 일을 모두 마칠 수 없었기 때문이다.

이에 따라, Ander Van der Hoek는 크라우드 소싱을 소프트웨어 설계에 적용하기 위해 세 가지 단계를 제시한다.

첫 째, 요구사항에 따라, 작업자들은 하나의 설계 문제의 필수적인 면들을 정의한 결정 지점들을 만든다. 둘 째, 여러 결정 지점들에 기반하여 각 결정 지점을 해결하기 위한 대안 해결책들을 제안한다. 마지막으로, 각 결정 지점에서 하나의 해결책을 선택하는 것이다.

Consuelo Lopez는 세 가지 단계 중 두 번째 단계에 대한 연구를 군중들을 통해 진행했다. 즉, 비전문가 집단을 통해 이미 정의된 결정 지점들에 대한 대안 해결책들을 제시하도록 하였다. 이 연구는 크라우드 소싱 플랫폼인 아마존 메카니컬 터크(Amazon Mechanical Turk) 에서 수 천명의 참여자와 함께 진행되었는데, 각각의 참여자들이 비교적 적은 시간과 노력을 들여 다양한 대안 집합(Alternative set)을 생성했으며, 그 중 충분히 만족할만한 부분 집합이 생성됨을 확인했다[8].

앞선 선행연구와 달리 본 연구에서는 결정 지점들을 만드는 첫 단계에 초점을 두었다. 군중들이 결정 지점들을 더 쉽게 제안할 수 있도록 하는 Keyramid(Keywording Pyramid)라는 소프트웨어 설계 작업 흐름을 개발했다. Keyramid의 키워딩 시스템은 요구사항들에서 고려해야하는 중요한 키워드를 추출하는 것이다.

2. 본 론

하나의 요구사항을 위해 결정 지점들을 생성하기 위해 작은 그룹들을 타겟으로 하여 연구를 진행했다. 요구사항들로부터 곧바로 소프트웨어 설계의 결정 지점을 얻는 것이 어려웠기 때문에 , 크라우드 소싱을 이용하여 결정 지점을 만들기 전에 키워드를 먼저 추출하기로 결정했다. 키워드는 여러 솔루션이 존재할 수 있는 소프트웨어 설계 지점을 설정하는 데 도움이 되는 표시기로 사용된다.

2.1. 사전 실험

컴퓨터 관련 전공의 아주대, 서울여대, 국민대, 한양대, 서강대 총 28명의 작은 그룹의 사람들로부터 키워드들의 추출을 진행했다. 두 페이지 분량의 교통 시뮬레이터의 요구사항을 나눠주고, 키워드를 추출하는 목적과 방법을 설명했다. 28명을 두 개의 그룹으로 나누고 계획한 두 가지 방법을 제시했다.

첫 번째 방법은 전반적인 요구사항을 읽고 난 뒤 요구사항의 특정 단어를 하이라이트 하는 것이다. 이 방법은 요구사항의 상호의존성을 고려하고 사용자들이 모든 요구사항을 읽어야 했기 때문에 수준 높은 키워드 집합을 얻을 수 있었다. 하지만 사람들이 그들이 중요하다고 생각하는 모든 단어를 하이라이트 했을 때 너무 많은 데이터를 얻는다는 문제점이 있었다. 그래서 비슷한 의미의 단어들을 통합하고 작업자들의 단어의 하이라이팅 비율에 따라 단어들의 랭킹을 제공하고 싶었다[7].

두 번째 방법은 전체 요구사항을 나누어 작업자들에게 제시하고, 중요한 단어를 추출하는 것이다. 이 방법은 고려해야할 요구사항의 양이 작았기 때문에 작업하기 쉬울 것이라고 예상되었으나 사람들이 전반적인 요구사항을 이해하기 힘들어했다. 부분적인 요구사항을 제공하는 것은 상호의존성의 부족과 필수적인 요구사항을 생략하는 결과를 초래했다. 각 그룹의 결과들이 수집하고 분석했을 때, 필수적인 요구사항 정보의 생략을 방지하고 상호의존성을 고려하는 것이 중요하다고 결정되었다.

Highlighting a Phrase	
Keyword	**Number**
sensors that detect whether any cars are present in a given line	3
not be complex	3
traffic signal timing	3
determining the amount of time	3
traffic signal timing schemes	3
intersection's lights'	3
behavior of the traffic lights	2
simulate	2
visually	2
updated	2
roads	2
traffic flows	2
user	2
crashes should not be allowed	2
each intersection	2
elegance and clarity	2
traffic flow simulation program	2
given intersection's traffic lights	2
changing the timing	2
roads of varying length to be placed	2
create a visual map	2
at least six intersections	2
conveyed visually	2
the current state	2

그림 1 사전 실험 결과 – phrase 하이라이팅

Highlighting Only a Word	
Keyword	**Number**
traffic	37
timing	25
intersection	18
simulate(tion)	17
visually	15
signal	13
light	13
map	12
sensors	12
density	11
flow	11
abstract	11
pattern	10
easy	9
left-hand	8
minimized	7
change	7
real-time	6
behavior	6
levels	6
road	6
elegance	5
six	5
length	4

그림 2 사전 실험 결과 – 한 단어 하이라이팅

Freely Suggestion	Number
traffic lights	9
traffic signal timing / +schemes	6
map	5
different arrangements of intersections / + to be created	5
intersections	4
sensors	4
traffic levels	4
real-time manner	3
not be complex	3
change the traffic density	3
traffic density	3
simulate	3
roads of varying length to be placed	3
current state of the intersections'	3
4-way	2
visually	2
updated	2
traffic flow simulation program	2
Combinations of individual signals	2
4-way	2
intersection's lights' behavior	2
crashes should not be allowed	2
traffic flows	2
user	2
the amount of time	2

그림 3 사전 실험 결과 자유로운 하이라이팅

2.2. 웹 어플리케이션을 통한 실험

본 연구에서는 실제 크라우드 소싱이 행해지고 있는 Amazon Mechanical Turk에서 작업하는 웹 에플리케이션 환경에서 계속해서 실험하기로 결정했다. 단위 테스트의 결과들을 기반으로 본 연구에서는 decision point들의 생성을 위해 키워딩 작업이 반드시 선행되어야 한다고 결론지었다. 그래서 이 어플레케이션은 작업자들이 단어들을 쉽게 하이라이트하게 도와주고 다른 작업자들과 동기적으로 작업할 수 있게 도와주는 툴이다. 사용자들은 제외어(stop words)들을 제외한 모든 단어들 중에서 그것이 중요한지 중요하지 않은지 선택할 수 있다. 제외어들은 회색 처리되어 있으며 클릭할 수 없다. 제외어가 아닌 단어들에 대해 한 번 클릭하는 것은 파란색으로 중요한 단어임을 알려주고, 그리고 더블 클릭은 주황색으로 중요하지 않은 단어를 알려준다. 문서에서 같은 줄기를 가지는 모든 단어들은 nltk를 사용한 자연어 처리 과정의 결과로서 그들 중 하나가 클릭 될 때 다같이 하이라이트 되도록 하였다.

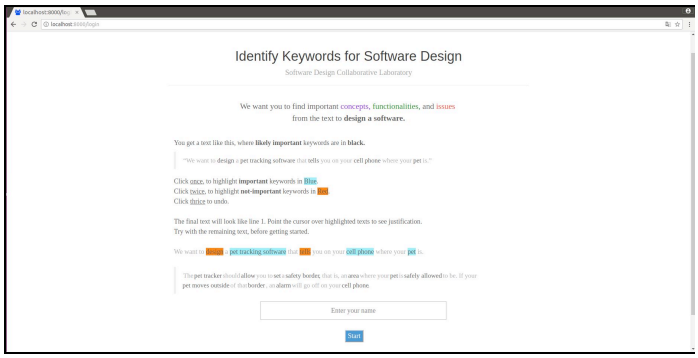


그림 4 웹 어플리케이션 메인 페이지

그림 5 웹 어플리케이션 활동 페이지

하나의 부분 요구사항이 작업자들에게 주어졌고 작업자들의 작업은 다른 작업자들의 작업과 실시간으로 동기화 되었다. 작업자들은 제안된 요구사항 내의 단어들을 '중요' 또는 '중요하지 않음' 으로 무제한으로 분류할 수 있다.

작업자들은 주어진 텍스트의 모두를 하이라이트할 필요가 없다. 주어진 요구사항의 모든 단어들을 분류한 후, 사용자는 추가적인 작업들을 할당 받을 지를 선택할 수 있다. 각 작업자의 단어 하이라이팅 결과는 실시간으로 서버에 전송되며, 서버는 그 단어의 줄기가 몇 번 하이라이트 되었는지 수를 샌다. 이것은 동기적으로 다른 작업자들의 작업에 영향을 준다. 같은 줄기의 단어들이 10명의 작업자들에 의해

하이라이트되면, 그 단어는 모든 작업자의 작업에서 회색 처리되고 클릭할 수 없게 된다. 작업자들의 작업의 결과로서 모든 단어들이 회색 처리 될 때, 문서를 키워딩 하는 작업이 종료 된다.

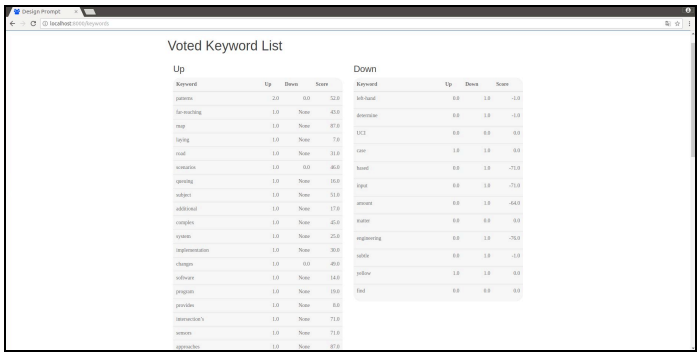


그림 6 웹 어플리케이션의 결과 페이지

이 어플리케이션은 실험, 문서의 자연어 처리 및 빈번한 강조 표시 요청의 효율적인 처리를 위해 설계된 간단한 프로토 타입이다. 웹 어플리케이션과 그 논리는 빠른 개발 속도와 풍부한 자연어 처리 라이브러리를 가지는 Python과 Django 프레임워크들을 사용하여 개발되었다. 그리고 작업자들의 데이터와 자세한 요구 특이사항을 유지하기 위해 데이터베이스로서 postgresql을 사용했다. 또한 빈번한 요청들과 작업자들의 각 하이라이팅을 정렬하고 카운팅하는 것을 제어하기 위해 in-memory 자료구조로서 Redis를 사용했다. 개발된 어플리케이션은 Docker를 통해 컨테이너화 되었으며 모든 프레임워크들은 docker-compose를 통해 패키지화 되었다. 이 실험에서 웹 서버는 Amazon Web Service (AWS)의 Ubuntu 인스턴스를 통해 호스팅되지만 Docker를 사용하여 외부 네트워크에서 액세스 할 수 있는 모든 컴퓨터에 쉽게 설치하고 호스팅 할 수 있다.

3. 결 론

연구의 결과로 결정 지점들을 생성하는데 필요한 핵심 키워드들을 얻을 수 있었다. 비전문가들을 통해 결정 지점들을 생성하는데 여러 방법이 있지만, 일련의 실험을 통해 실험 참가자들로부터 얻어낸 키워드 리스트들을 사용하는 것이 효과적이라고 증명하였다. 이러한 이유로, 결정 지점들을 생성하는데 키워딩 작업 흐름을 채택했다. 요구사항들로부터 키워드들을 추출하는 과정에서 다음과 같은 두 가지 방법으로 실험을 진행했다.

첫 번째는 자유롭게 단어를 생성하는 방법, 두 번째는 하이라이팅 방법이다. 실험 참가자들이 자유롭게 요구사항들로부터 키워드들을 제시한다면 다양한 키워드들의 집합을 얻을 것이라고 예상했다[2]. 하지만 실험에 참가하는 사람에 따라 그 결과가 매우 다르고, 그 결과를 통합하는 과정이 매우 힘들었다[7]. 추가적으로, 키워드 생성 작업 흐름으로부터 자유롭게 추출한 키워드 집합에 필수적으로 요구사항에서 다뤄야 하는 키워드가 빠지는 문제와 각 단어들 간의 상호의존성이 고려되지 않는 문제가 있었다. 하이라이팅 하는 방법을 사용한 실험에서는 실험 참가자들이 자유로운 키워드 생성을 했을 때 결정 지점의 생성이 더 단순하고 쉽다고 느꼈기 때문에 더 적은 시간을 소요했다.

그리고 또한 하이라이팅 방법에서 하나의 구를 선택할 지 혹은 하나의 단어를 선택할 지의 결정 요소가 있었다. 구를 선택하는 방법을 고려했을 때, 구의 범위를 결정하는데 어려움과 실험 참가자들이 선택한 구들의 의미를 결정하는데 어려움이 있었다. 따라서, 오직 하나의 단어를 하이라이팅하는 키워드 추출 방법을 선택하게 되었다.

4. 토의 및 향후 연구

키워딩을 하는 단계는 결정 지점들을 생성하기 위한 작업 흐름의 일부분으로서 작업자들이 그들이 전문가가 아니거나 정확히 결정 지점이 무엇인지 모르더라도 기여할 수 있도록 하였다. 본 연구에서 사전 실험 대상이 소프트웨어 설계 경험이 매우 적은 학부생들임에도 불구하고 의미 있는 결과를 생산한 것을 볼 수 있었다. 이는 키워딩이 작업 흐름을 세분화하여 각 작업을 보다 쉽게 만들고 작업자들의 참여를 유도할 수 있음을 의미한다. 따라서 키워딩은 크라우드 소싱을 통한 소프트웨어 설계 작업 흐름의 일부로 매우 유용하다고 결론 지을 수 있다.

키워딩을 위해 개발한 웹 어플리케이션은 작업자들이 쉽게 하이라이트하게 하고 작업자들을 통제[3]할 수 있도록 편리함을 느낄 수 있도록 하는 깔끔한 인터페이스 디자인을 가지는 최첨단 도구이기도 하다. 그리고 소켓 통신을 사용해서 다른 작업자들과의 실시간 반응을 제공하여, 작업자들이 쓸모 없는 일을 하지 않도록 하였다. 작업자들의 하이라이팅 결과는 지연 없이 업데이트 되었고, 간편 후처리(easy-to-post-processing) 자료구조인 Redis에 저장되었으며, 연구들로부터 설계된 작업 흐름을 지원하기 위한 모든 필수 요소를 갖추었다. 이제 문제는 실제 작업자들에게 얼마나 잘 적용되는지에 대한 것이다.

이 문제에 대답하기 위해 이 어플리케이션을 실제 크라우드 소싱 플랫폼인 Amazon Mechanical Turk와 통합하는 작업을 하고 있다. 플랫폼 위에서 실제 작업자들이 어떻게 반응할 지와 얼마나 의미 있는 키워드들이 생성될지를 확인할 실험을 진행할 것이다.

이 실험은 결정 지점들을 만드는 중요한 부분으로 설계된 키워딩 방법이 도움이 되는지 아닌지를 알 수 있게 하는 아주 중요한 이정표가 될 것 이므로 이 단계를 따라 작업을 계속할 수 있다. 그리고 작업자들의 작업을 장려할 수 있도록 얼마나 많은 일에 대해 얼마를 지불 할 것인가와 같은 지불 정책도 논의되어야 할 것이다.

5. Acknowledgment

항상 에너지 넘치고 긍정적으로 격려하여 연구를 활발하게 할 수 있도록 해준 UCI의 Andre Van der hoek 교수님과, Ph.D Ayushi Rostagi에게 감사를 드립니다. 또한 연구에 집중할 수 있도록 편하게 지낼 수 있게 해준 Said M. Shokair에게도 감사드립니다. 마지막으로, UCI로 가기 전부터 지금까지 학생들의 건강과 프로젝트의 진행 여부를 관리해주시고 직접 찾아오셔서 물심양면으로 지원을 아끼지 않으셨던 아주대학교의 이석원 교수님에게 감사드립니다.

"본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 SW중심대학지원사업의 연구결과로 수행되었음"(20150009080031001)

6. 참고 문헌

[1] André van der Hoek, Marian Petre. Software Designers in Action: A Human-Centric Look at. CRC Press. 2013.

[2] T. D. LaToza and A. van der Hoek, “Crowdsourcing in software engineering: Models, motivations, and challenges,” *IEEE Software*, vol. 33, no. 1, pp. 74–80, 2016.

[3] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut, “Crowdforge: Crowdsourcing complex work,” in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011, pp. 43–52.

[4] A.Kulkarni, M.Can, and B.Hartmann, “Collaboratively crowdsourcing workflows with turkomatic,” in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, 2012, pp. 1003–1012.

[5] W. S. Lasecki, J. Kim, N. Rafter, O. Sen, J. P. Bigham, and M. S. Bernstein, “Apparition: Crowdsourced user interfaces that come to life as you sketch them,” in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 2015, pp. 1925– 1934.

[6] T. D. LaToza, M. Chen, L. Jiang, M. Zhao, and A. Van Der Hoek, “Borrowing from the crowd: A study of recombination in software design competitions,” in *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 2015, pp. 551–562.

[7] T. D. LaToza, W. B. Towne, C. M. Adriano, and A. Van Der Hoek, “Microtask programming: Building software with a crowd,” in *Proceedings of the 27th annual ACM symposium on User interface software and technology*, 2014, pp. 43–54.

[8] C. López, E. R.Q. Weidema, S. Nayebaziz, F. Spanghero, A. van der Hoek, “An Experiment in Designing Software: Microtasking Design Alternatives with a Crowd”, 2015.

최초의 민간 소프트웨어 품질 인증, ‘SW인증’ 파헤치기!

2018년 1월
STA테스팅컨설팅 김관호 수석

목 차

- SW 인증 개요
- SW 인증 취득 기준 및 절차
- SW 인증을 위한 “테스트 사례”
- SW 인증의 미래

- **SW 인증 개요**
  - SW 인증 취득 기준 및 절차
  - SW 인증을 위한 “테스트 사례”
  - SW 인증의 미래

SW 인증이란?

- **SW 인증이란,**
  - SW 제품을 **ISO/IEC/IEEE 29119 SW 테스팅 표준**에 따라 테스트하고
  - **인증 기준**에 준하는 SW 제품에
  - “**SW인증 컨소시엄**”이 인증을 부여하는 제도

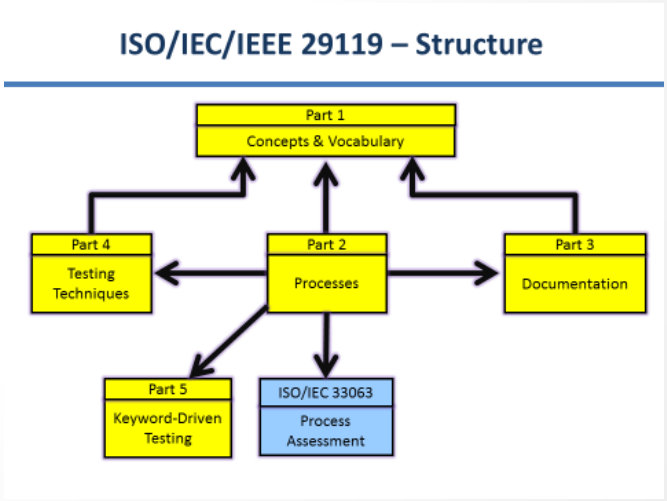
SW 인증이란?

- SW 인증이란,

  – SW 제품을 **ISO/IEC/IEEE 29119 SW 테스팅 표준**에 따라 테스트하고

  – **인증 기준**에 준하는 SW 제품에

  – "**SW인증 컨소시엄**"이 인증을 부여하는 제도



4

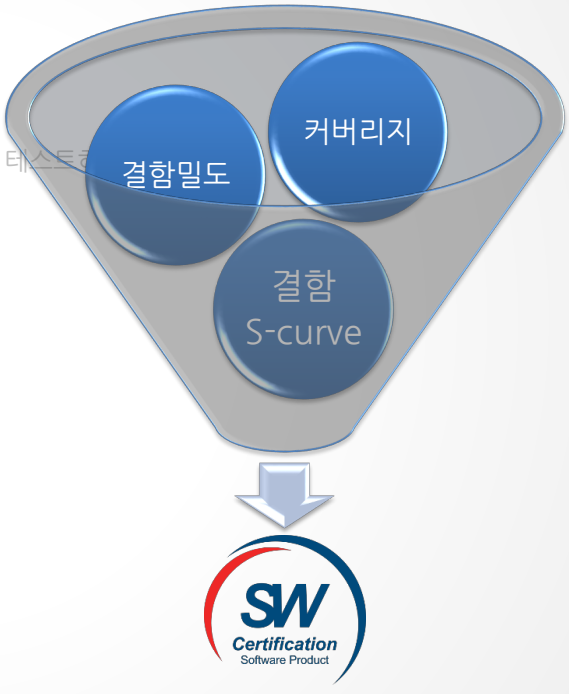
SW 인증이란?

- SW 인증이란,

  – SW 제품을 **ISO/IEC/IEEE 29119 SW 테스팅 표준**에 따라 테스트하

  – **인증 기준**에 준하는 SW 제품에

  – "**SW인증 컨소시엄**"이 인증을 부여하는 제도



5

SW 인증이란?

- SW 인증이란,
  - SW 제품을 ISO/IEC/IEEE 29119 SW 테스팅 표준에
  - 인증 기준에 준하는 SW 제품에

– "SW인증 컨소시엄"이 인증을 부여하는 제도

SW 인증의 특징

구 분	내용	비고
대상	SW 전분야	
주관 기관	SW 인증 컨소시엄 - 전자신문, KOSA, STA테스팅컨설팅	
평가 기관	STA테스팅컨설팅 외 다수 테스팅 기업과 1개 지역 특화연구소(KOLAS 인정 기관)	
참조 표준	ISO/IEC/IEEE 29119 SW 테스팅 국제 표준 (IT를 포함한 국방,의료,항공,자동차,기능안전성 전 분야에서 참조하는 SW 테스팅 표준)	
테스트 방법	리스크 기반 테스트 전략을 기반으로 SW 제품의 의미 있고 중요한 부분을 더 강도 높게 테스트 (세계적으로 가장 효과/효율적인 테스팅 방법)	
인증 기준	커버리지, 결함 밀도(리스크와 심각도를 고려한 종합적인 관점), 결함 S-curve	
인증 등급	**Criticality** High, Medium, Low	도메인 및 제품 특성 등으로 분류
테스트 결과물	테스트 계획 문서, 결함리포트, 테스트 결과보고서 (결과 내용은 기업이 실무 테스팅 활동에 활용될 수 있는 내용임) 테스트 관리 시스템 제공 및 활용(OTM) - 설계문서, 실행결과 공유	
기타 특징	국제 테스트 표준 최초 제안자(ISO/IEC/IEEE 29119 의장, Stuart Reid 박사)가 참여 실무적 활용성 높음	

SW 인증의 장점 및 혜택

엄격한 **인증 기준** 충족과
잘 정리된 평가 결과서로
SW 제품의 높은 품질 향상

인증서

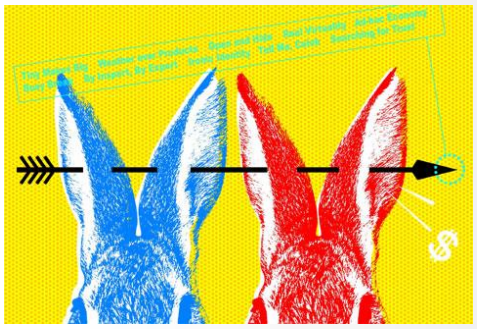
품질
평가
결과

제안 입찰 시 경쟁 차별화
포인트 역할 및
기업의 홍보/마케팅 효과

SW Certification Software Product

Quality

+α



테스트
절차

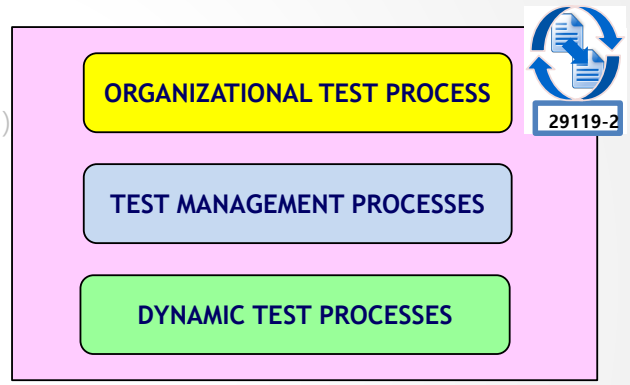
테스팅 활동 전 과정에 대한 노하우 전수로
테스트 방법론 자산 획득의 기회

테스트 계획 및 전략	테스트 분석 및 설계	테스트 환경구축	테스트 실행	테스트 종료

- SW 인증 개요
- **SW 인증 취득 기준 및 절차**
- SW 인증을 위한 “테스트 사례”
- SW 인증의 미래

SW 인증 기준

기본적으로 ISO/IEC/IEE 29119 표준 테스트 프로세스에 근거해 테스트를 수행하며



- 결함 밀도(리스크와 결함 심각도를 고려한 종합적 관점)
- 결함 검출 추이,
- (제품의 모든 기능 셋에 대해)커버리지 충족 여부를

인증 기준으로 함

SW 인증 기준

기본적으로 ISO/IEC/IEE 29119 표준 테스트 프로세스에 근거해 테스트를 수행하며

- **결함 밀도(리스크와 결함 심각도를 고려한 종합적 관점),**
- 결함 검출 추이,
- (제품의 모든 기능 셋에 대해)커버리지 충족 여부

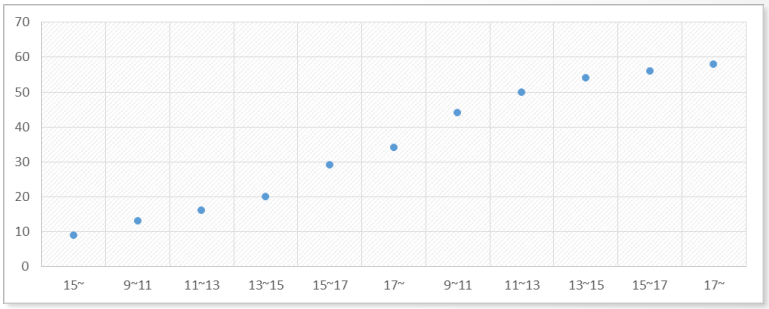
		리스크 레벨		
		Low	Medium	High
결함 심각도	Critical	0	0	0
	Major	2	2	0
	Minor	3	3	2
	Note	12	9	6

인증 기준으로 함

SW 인증 기준

기본적으로 ISO/IEC/IEE 29119 표준 테스트 프로세스에 근거해 테스트를 수행하며

- 결함 밀도(리스크와 결함 심각도를 고려한 종합적 관점),
- **결함 검출 추이(S-curve),**



- (제품의 모든 기능 셋에 대해)커버리지 충족 여부를

인증 기준으로 함

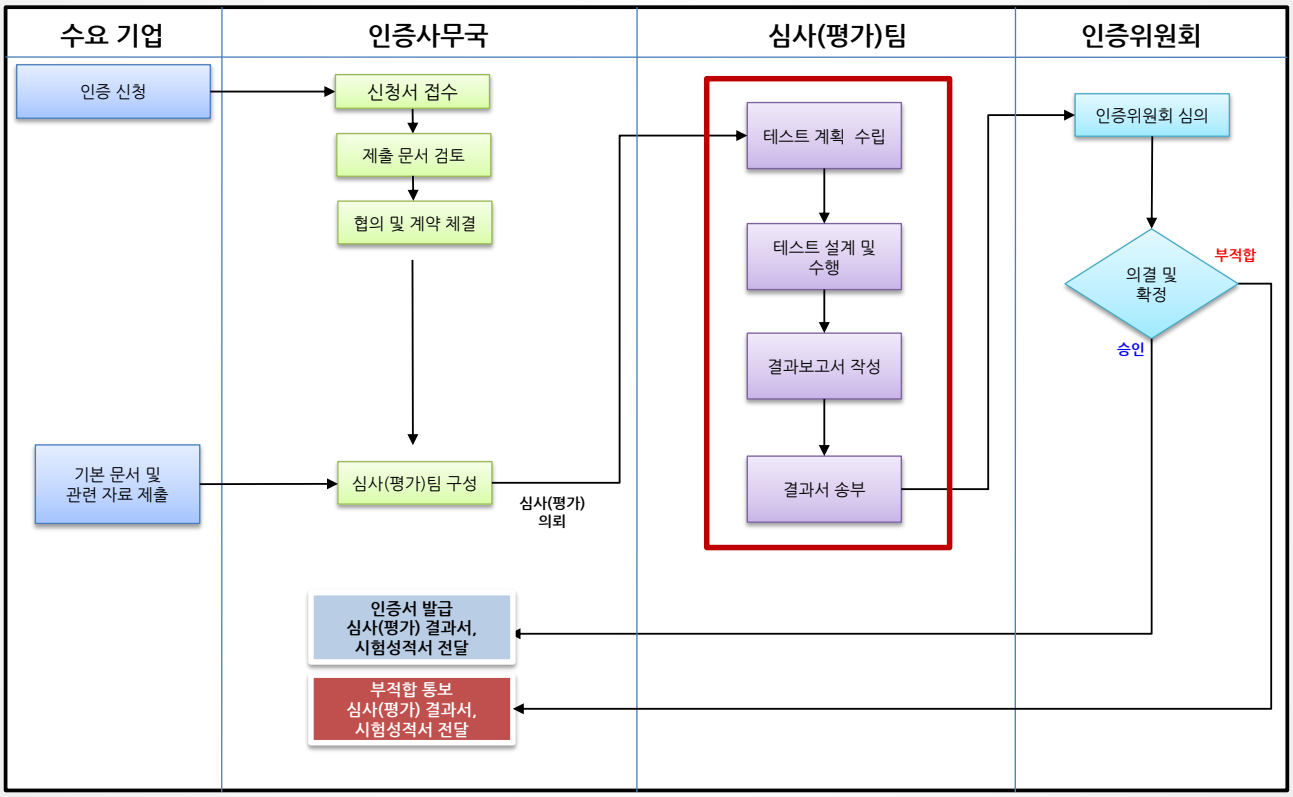
SW 인증 기준

기본적으로 ISO/IEC/IEE 29119 표준 테스트 프로세스에 근거해 테스트를 수행하며

- 결함 밀도(리스크와 결함 심각도를 고려한 종합적 관점),
- 결함 검출 추이,
- **(제품의 모든 기능 셋에 대해)커버리지 충족 여부를**

인증 기준으로 함

전체 SW 인증 진행 절차



인증 테스트 절차

- 주요 테스트 활동에 따른 일정

테스트 매니저

ISO/IEC/IEE 29119 프로세스

테스트 계획
- 정황 이해
- 리스크 식별/분석
- 테스트 전략 정의
- 테스트 계획서 작성/승인

테스트 모니터링&제어
- 테스트 진척 모니터링
- 종료 기준 평가
- 테스트 제어(이슈 대응)

테스트 종료
- 테스트 결과 정리
- 테스트 환경 정리
- 테스트 이슈 공유(회고)

인증 기준
- SW기능에 대한 커버리지 충족
- 결함 밀도
- 결함 검출 추이(결함 S-curve)

테스트 설계&구현
- 테스트 베이시스 리뷰
- 테스트 컨디션 도출
- 테스트 케이스 추가

인시던트 보고
- 인시던트 리뷰
- 인시던트 보고

테스트 환경 구현&유지
- 테스트 환경 구현
- 테스트 환경 유지

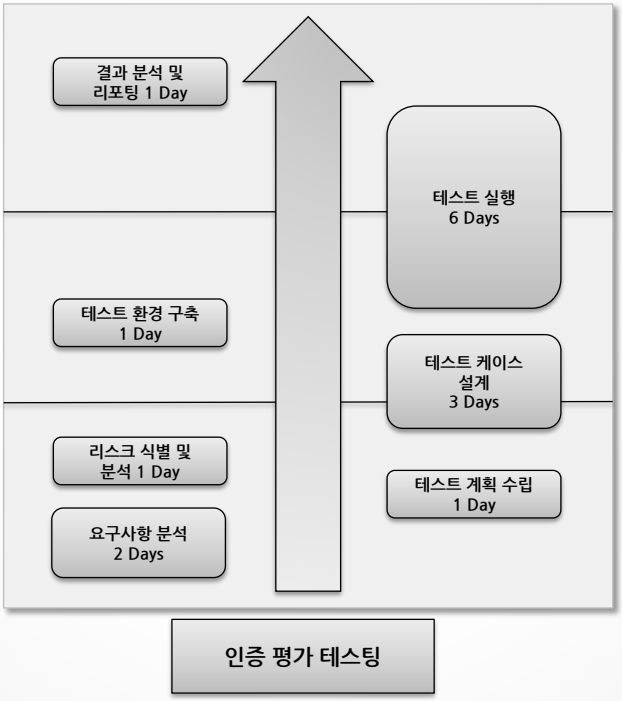
테스트 실행
- 테스트 실행 및 결과 기록
- 결과 비교(예상/기대 결과)

테스터

- SW 인증 개요
- SW 인증 취득 기준 및 절차
- **SW 인증을 위한 “테스트 사례”**
- SW 인증의 미래

사례 소개 - 테스트 계획 및 전략

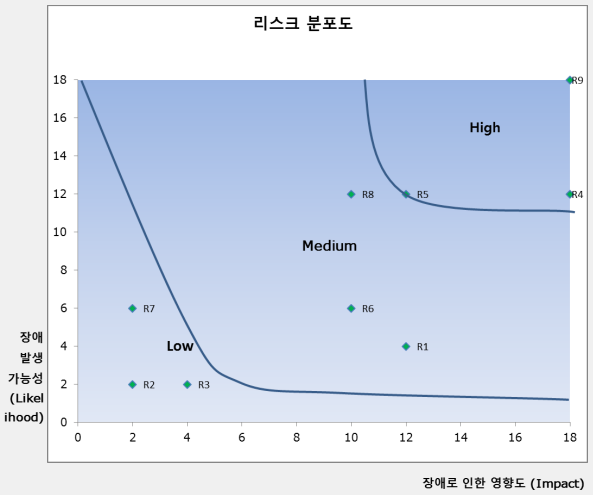
- 주요 테스트 활동에 따른 일정
  - 총 15MD(약 7일) 소요



사례 소개 - 테스트 계획 및 전략

- 리스크 분석

No.	리스크 아이템	리스크영역
RK_1	Main	Medium
RK_2	OO 소개(PC)	Low
RK_3	OO소식(PC)	Low
RK_4	OO정보(PC)	High
RK_5	전문지식(PC)	High
RK_6	참여마당(PC)	Medium
RK_7	마이페이지(PC)	Low
RK_8	회원(PC)	Medium
RK_9	통합검색(PC)	High



리스크 영역 별로 결정된 리스크 아이템 수
- High 3개
- Medium 3개
- Low 3개

사례 소개 - 테스트 계획 및 전략

- 리스크 기반 테스팅 전략 수립

리스크 영역	테스트 방법		테스트 엔지니어
	기능	비기능	
High	스크립트 테스트 탐색적 테스트	사용성 테스트 (닐슨 체크리스트 적용) 100%	유사 경력 6년 이상 고급 1명 중급 1명
Medium	탐색적 테스트	사용성 테스트 (닐슨 체크리스트 적용) 60%	유사 경력 3~4년 초급1
Low	탐색적 테스트	사용성 테스트 (닐슨 체크리스트 적용) 30%	유사 경력 3년~4년 초급 1

사례 소개 - 테스트 설계

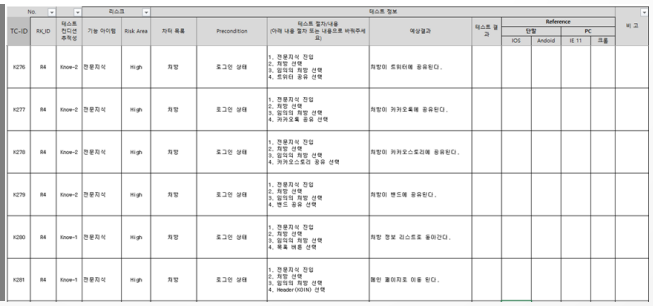
- 리스크 High 영역에 대해 명세기반 테스트 설계

경험기반을 토대로 동등분할 및 경계값 분석 기법 적용

높은 리스크 영역만 테스트 케이스 설계

리스크 아이템	리스크 영역	테스트 케이스 개수
OO정보	High	202개
전문지식	High	122개
통합검색	High	52개

설계된 테스트 케이스



사례 소개 - 테스트 설계

- 사용성 테스트 설계

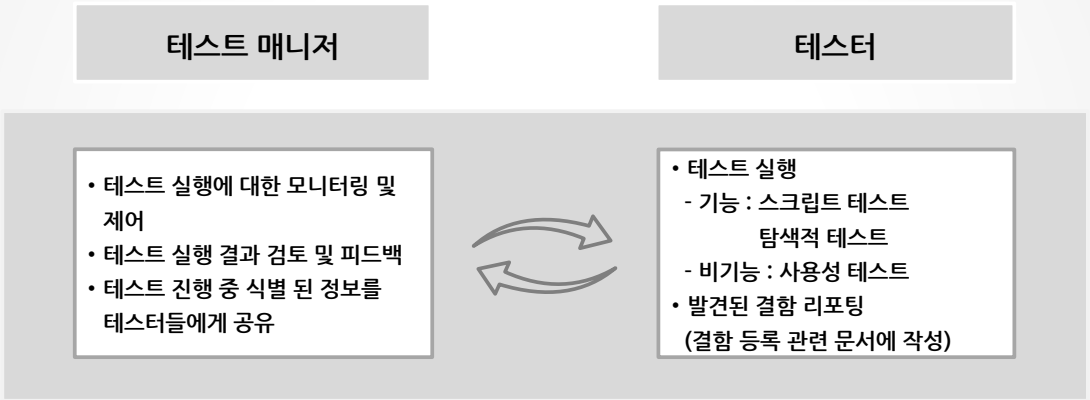
UI/UX 점검을 위한 **닐슨 체크리스트 문서를 기반**으로 적용 가능한 부분을 추출하여 **사용성 체크리스트 146개 문항** 설계

리스크 영역	사용성 체크리스트	비고
High	76개	
Medium	45개	
Low	25개	

사례 소개 - 테스트 환경 구축

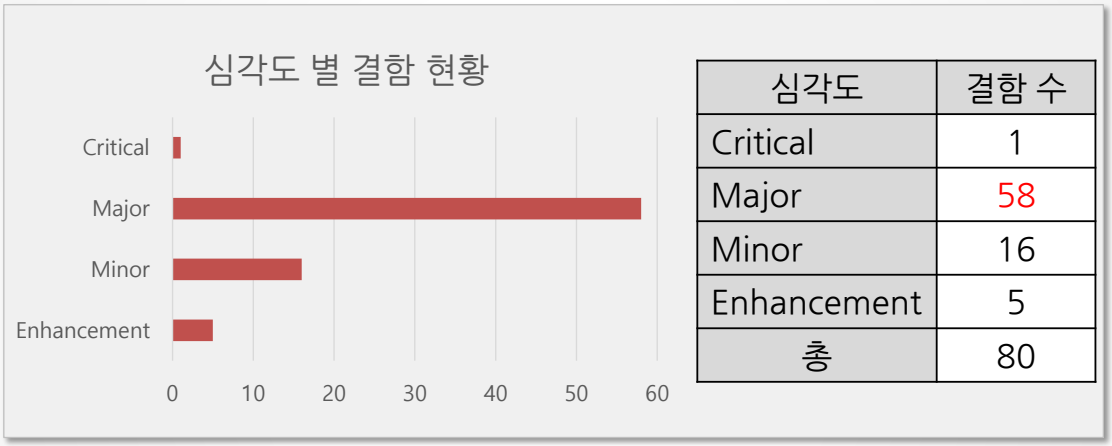
구분	모바일 앱	PC 웹	비고
테스트 실행 환경	iOS : 9.0 이상 Android : 5.0 이상	Internet Explorer 11 Chrome	스크립트 테스트는 각 환경 별로 모든 TC 실행
결함 관리 도구	Microsoft Excel		

사례 소개 - 테스트 실행



사례 소개 - 테스트 종료

- 테스트 결과 분석 - 심각도 별 결함 현황



심각도	결함 수
Critical	1
Major	58
Minor	16
Enhancement	5
총	80

특이 사항	리스크 High 영역에 대해 스크립트 테스트를 진행해 기본 기능에서 많은 결함을 발견했기 때문에 Major 심각도의 비중이 높음

사례 소개 - 테스트 종료

- 테스트 결과 분석 - 결함 밀도

		리스크 영역		
		Low	Medium	High
결함 심각도	Critical	0	1	0
	Major	3	10	45
	Minor	1	5	10
	Note	0	3	2

Vs.

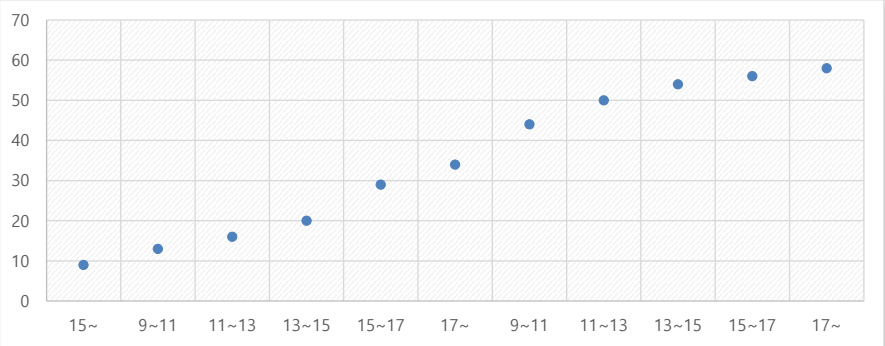
인증 기준 결함 밀도

		리스크 영역		
		Low	Medium	High
결함 심각도	Critical	0	0	0
	Major	2	2	0
	Minor	3	3	2
	Note	12	9	6

사례 소개 - 테스트 종료

- 테스트 결과 분석 - 리스크 High 영역의 결함 누적 추이

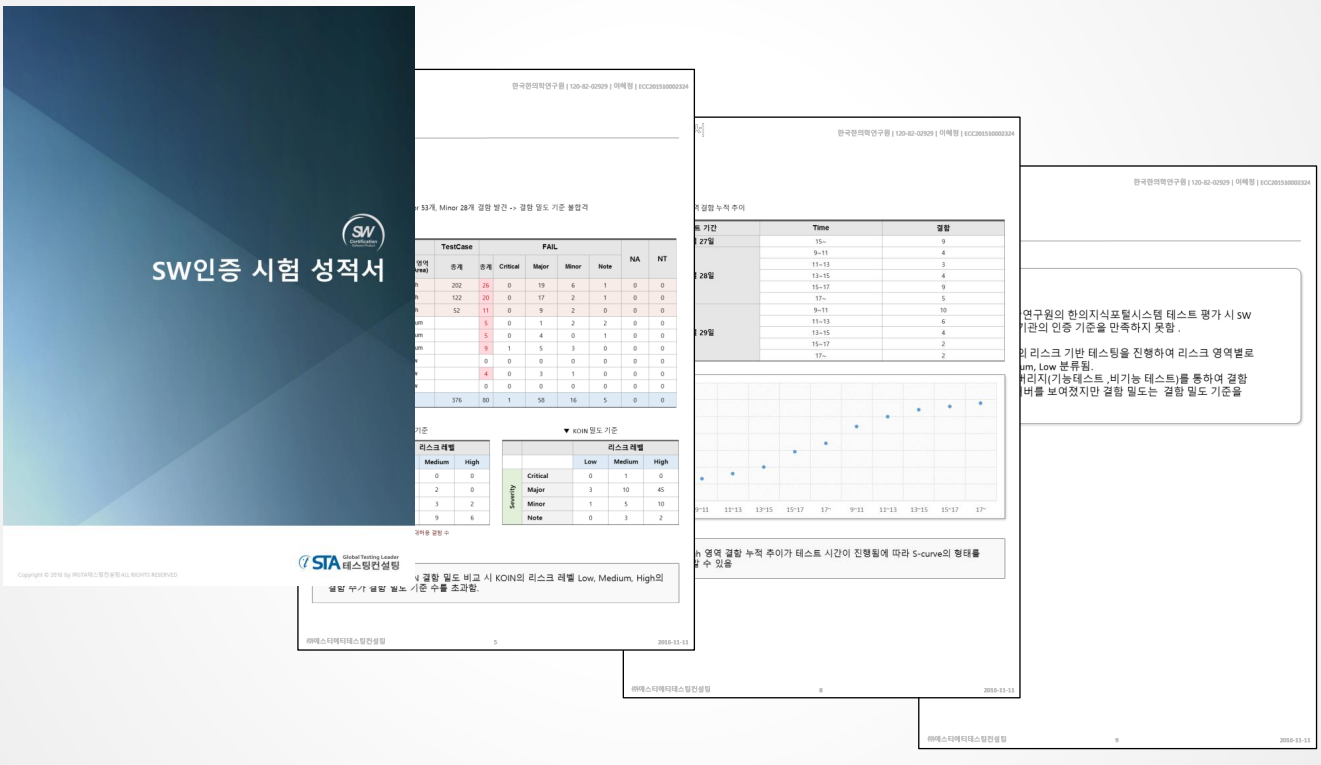
테스트 기간	Time	결함
XX월 27일	15~	9
XX월 28일	9~11	4
	11~13	3
	13~15	4
	15~17	9
	17~	5
XX월 29일	9~11	10
	11~13	6
	13~15	4
	15~17	2
	17~	2



❖ 리스크 High 영역 결함 누적 추이가 테스트 시간이 진행됨에 따라 S-curve의 형태를 나타냄을 알 수 있음

사례 소개 - 테스트 종료

- 시험 성적서 발행



- SW 인증 개요
- SW 인증 취득 기준 및 절차
- SW 인증을 위한 “테스트 사례”
- **SW 인증의 미래**

SW 인증 향후 계획

- **시험원 양성 교육**
  - ISO/IEC/IEEE 29119 자격 준비 과정
- **금년부터 KOLAS 인정기관 참여**
  - KOLAS 인정 기관인 동의대학교 부산IT융합부품연구소의 인증평가 참여로 인증 신뢰성 향상
- **테스트 자동화와 연계**
  - 오픈 소스 자동화(Fuzz Testing Tool), AI 자동화
- **글로벌 인정**
  - ISO 29119 표준화 조직의 의장인 Stuart Reid CTO의 주도

Q & A

감사합니다.

www.sten.or.kr
www.sta.co.kr

- SW 인증 신청 홈페이지
  - http://swcerti.sten.or.kr
- 인증 관련 상담 문의
  - 전화 070-7864-7708
  - 이메일 swcerti@sta.co.kr

김관호 수석
khkim@sta.co.kr