ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 44, No. 5, pp. 510-521, 2017. 5
https://doi.org/10.5626/JOK.2017.44.5.510

OOPT: 소프트웨어공학 교육을 위한 객체지향 소프트웨어 개발 방법론

(OOPT: An Object-Oriented Development Methodology for Software Engineering Education)

정 세 진 † 이 동 아 † 김 의 섭 † 장 천 현 †† 유 준 범 †††
(Sejin Jung) (Dong-Ah Lee) (Eui-Sub Kim) (Chun-Hyon Chang) (Junbeom Yoo)

요 약 소프트웨어 개발 프로세스(Software Development Process: SDP)는 소프트웨어공학 교육에서 가장 기초적이며 중심적인 역할을 한다. 모든 소프트웨어는 개발의 시작부터 마지막까지를 모두 포함하는 특정 SDP에 기반해서 개발된다. 따라서, SDP 교육은 소프트웨어공학의 제반 기술에 대한 이해를 도울 수 있다. 본 논문은 대학의 소프트웨어공학 수업에서 활용할 수 있는 소프트웨어 개발 방법론(프로세스)인 OOPT(Object Oriented Process with Traceability)를 소개한다. OOPT는 객체지향 소프트웨어를 개발하기 위한 방법론으로서, 각 단계마다 구체적인 요구사항과 산출물을 정의하고 있으며, 단위/시스템 시험 및 추적성 분석 등의 추가적인 내용들도 포함하고 있다. 본 논문은 OOPT에 대한 적용 사례로서 다년간의 건국대학교 컴퓨터공학과 소프트웨어공학 관련 수업들을 소개하고 있으며, 향후 개선 및 발전 방향을 포함한다.

키워드: OOPT, 소프트웨어공학 교육, 객체지향 개발방법론, 소프트웨어 개발방법론

Abstract The software development process (SDP) plays an important basic role in software engineering education. Every software is developed in accordance with a specific SDP which contains all phases of software development. SDP education helps students to understand the overall techniques and the process of software engineering. This paper introduces a software development methodology (i.e., process) – 'OOPT (Object Oriented Process with Traceability),' which was proposed for use in university software engineering classes. The OOPT is based on object-oriented software development, and it defines concrete requirements as well as outputs of each process/phases. It also contains the unit/system testing and a traceability analysis. We have used the OOPT in software engineering classes at Konkuk university for eight years. This paper conveys our experience as well as future extension and improvement plans.

Keywords: software engineering education, object-oriented software development, software development process/methodology, OOPT

† 학생회원 : 건국대학교 컴퓨터 정보통신공학과
jsjj0728@konkuk.ac.kr
ldalove@konkuk.ac.kr
atang34@naver.com
†† 종신회원 : 건국대학교 컴퓨터공학부 교수
chchang@konkuk.ac.kr
††† 정 회 원 : 건국대학교 컴퓨터공학부 교수(Konkuk Univ.)
jbyoo@konkuk.ac.kr
(Corresponding author임)

논문접수 : 2017년 1월 12일
(Received 12 January 2017)
논문수정 : 2017년 2월 6일
(Revised 6 February 2017)
심사완료 : 2017년 2월 19일
(Accepted 19 February 2017)

정보과학회논문지 제44권 제5호(2017. 5)

1. 서 론

소프트웨어 개발 프로세스(Software Development Process)는 1960년대의 소프트웨어 위기(Software Crisis)[1] 현상을 극복하기 위해 제안된 기법으로서, 소프트웨어공학의 시작점이 되는 근본 기술이다. 고성능 컴퓨터의 등장과 함께 더 많은/복잡한 기능을 수행하는 소프트웨어를 성공적으로 개발하기 위해서는, 다수의 개발자가 정해진 순서에 따라 협업하는 것이 필수적이라는 관찰을 통해, 최초의 소프트웨어 개발 프로세스인 폭포수 모델(Waterfall Model)[2]이 제안된 이후로 나선형 모델(Spiral Model), 반복적 모델(Iterative Model), 점진적 모델(Incremental Model) 등 다양한 모델[3]이 제안되었다.

현재는 이런 모델을 기반으로 소프트웨어 개발에 필요한 구체적인 사항들과 도구, 표기법 등을 추가한 것을 소프트웨어 개발방법론으로 통칭하는데, 크게 구조적 방법론(SASD: Structured Analysis and Structured Design)[4]과 객체지향방법론(OOAD: Object-Oriented Analysis and Design)[5]으로 분류할 수 있다. 구조적 방법론은 데이터 흐름을 기반으로 시스템을 top-down으로 분석하면서 소프트웨어 시스템을 개발하는 방법이며, C언어와 같은 절차지향 프로그래밍 언어(procedural programming language)에 적합하다. 반면, 객체지향방법론은 객체(Object Classes) 사이의 통신을 통해 시스템의 행위를 구현하는 방법론으로서, C++, Java와 같은 객체지향언어에 적합한 방법론이며, UML(Unified Modeling Language)을 사용한다. 다양한 응용 대상에 따라 다양한 객체지향 소프트웨어 개발 방법론[6-9]이 제안되었는데, 래셔널 통합 프로세스(RUP: Rational Unified Process)[10,11]를 현재 SI(System Integration) 산업계에서 가장 널리 사용되는 방법론으로 꼽을 수 있다. 기업에서는 래셔널 통합 프로세스를 각 상황에 맞게 수정(Tailoring)해서 사용하고 있다.

하지만, 래셔널 통합 프로세스를 교육과정에 그대로 적용하는 것은 교육에 다소 불필요한 내용들이 포함 될 수 있다. 이에 본 논문은 래셔널 통합 프로세스와 유사하지만 대학의 소프트웨어공학 수업에서 실습할 수 있는 수준으로 간결화된, 교육용 객체지향 소프트웨어 개발방법론인 OOPT(Object Oriented Process with Traceability)를 소개한다. OOPT는 대학의 소프트웨어공학 교육에서 필요한 다양한 요소들을 포함하고 있는데, 단위시험, 시스템시험, GUI, 추적성 분석 등이 그 예이다. OOPT를 소프트웨어공학 수업에 활용함으로써, 학생들은 소프트웨어공학의 필요성과 효과 등을 직접 체험할 수 있다. 또한, 다양한 소프트웨어공학 기술들을 직접 사용하고 적용함으로써, 소프트웨어공학이 이론이 아닌 실제적인/현장의 학문임을 직접 체험할 수 있다. OOPT는 건국대학교 컴퓨터공학과의 소프트웨어공학 관련 수업에서 2008년부터 지속적으로 활용되고 개선되어 왔다. 또한 본 논문은 OOPT의 추가적인 개선 및 발전 방향을 제시하고 그 동안의 경험을 공유함으로써, 여러 대학에서 소프트웨어공학 교육이 보다 효과적으로 수행되기를 기대한다.

본 논문의 구성은 다음과 같다. 2장은 소프트웨어 개발 프로세스와 대표적인 객체지향 소프트웨어 개발 방법론들을 소개한다. 3장은 OOPT를 자세히 소개하고, 4장은 건국대학교 소프트웨어공학 관련 수업에서 2008년 이후 OOPT를 이용하여 수행한 다양한 프로젝트들 중 'clone checker' 예제를 중점적으로 소개한다. OOPT의 구체적인 개선/발전 방향도 함께 정리하였다. 마지막으로 5장은 본 논문을 마무리한다.

2. 소프트웨어 개발 프로세스

2.1 소프트웨어 개발 모델

소프트웨어 개발 모델은 소프트웨어 개발 시 적용하는 모델로 폭포수 모델, 나선형 모델 등 다양한 종류가 있다. 폭포수 모델은 전통적인 소프트웨어 개발 모델로 구조적이고, 순차적인 접근 방법을 취한다[3]. 즉 앞 단계의 작업이 완료된 후 다음 단계의 작업이 진행되는 과정으로 구성되며, 각 단계는 요구사항 정의, 디자인, 개발, 테스팅 등으로 이루어져 있다. 폭포수 모델은 주로 요구사항이 명확히 정의되어 다음 단계로 넘어갈 수 있는 경우에 적합한 모델이며, 요구사항이 정해지지 않거나, 변경되는 경우에는 반복 및 수정에 어려움이 있다. 또한 다음 단계로 넘어가기 위해서는 다른 작업의 끝까지 기다려야 하는 상태가 발생할 수 있다.

나선형 모델은 위험을 최소화하기 위해 위험 분석을 포함한 개발 단계를 점진적으로 반복하여 개발하는 모델로 1988년 제안 되었다[9]. 폭포수 모델과 달리 반복적인 작업을 통한 개발이 필요한 경우에 효율적인 모델이다. 나선형 모델은 목표설정, 위험분석, 구현 및 테스트, 평가 및 다음 단계 진행으로 구성되며, 각 과정은 지속적이고 순차적으로 반복된다. 이처럼 여러 프로세스를 반복적으로 진행하여 개발을 수행하고 위험 분석을 통해 개발을 완성해 나가는 모델이다. 나선형 모델에서 위험분석은 요구사항의 변경 등 예측되는 위험 사항을 확인하고 대처 방안을 수립하는 단계이다. 다음 그림 1은 나선형 모델 프로세스에 대한 그림이다.

2.2 객체지향 소프트웨어 개발 방법론

객체지향 소프트웨어 개발 방법론은 다양한 소프트웨어 개발의 필요성 증가에 맞추어 실 세계의 구조를 정확

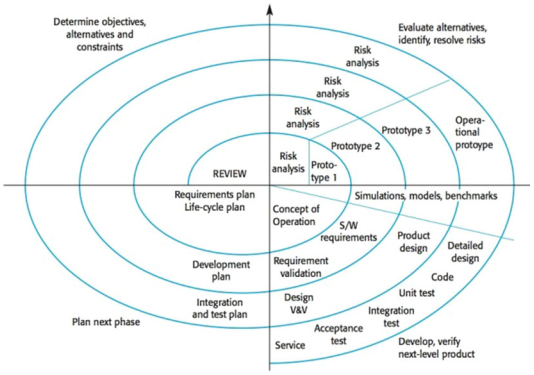

그림 1 나선형 모델 프로세스[3]
Fig. 1 Development process of a spiral model[3]

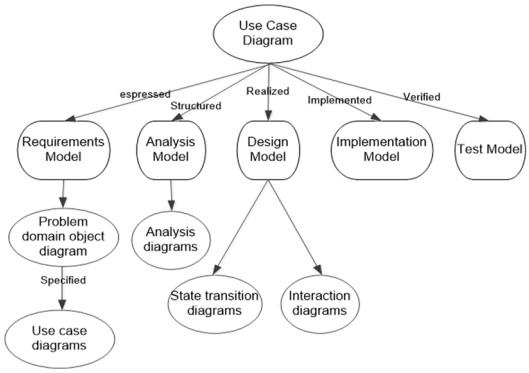

그림 2 OOSE 프로세스[7]
Fig. 2 The development process of OOSE[7]

하게 반영할 수 있고, 재사용을 통해 생산성을 향상 시킬 수 있다는 점으로 인해 제안되고 발전되었다. 초기에 제안된 개발 방법론 등으로는 OMT(Object Modeling Technique)[6], OOSE(Object-Oriented Software Engineering)[7], Fusion Methodology[8] 등이 있으며, 각각의 자체적인 표기 방법을 통해 디자인, 모델링 등의 과정을 구성하고 있다. 추후 표기 방법의 표준화를 위해 UML(Unified Modeling Language)이 사용 되었으며, UML을 사용한 소프트웨어 개발 방법론으로는 래셔널 통합 프로세스 등이 있다.

OMT는 소프트웨어 개발을 위해 객체 모델링 방법을 취하는 방법론으로, OMT의 모델링 방법은 UML의 이전 버전이라 할 수 있을 만큼 UML과 유사한 모델링 방법을 가지고 있다. OMT 방법론은 객체 모델(object model), 동적 모델(dynamic model), 기능 모델(functional model) 3가지 종류의 모델을 만드는 방식으로 진행되며 각 모델을 검증하고, 연결하는 형태로 마무리 된다. 객체 모델은 클래스 다이어그램과 유사한 형태의 클래스들을 모델로 작성한 것이고, 동적 모델은 시스템의 동작을 상태 다이어그램 형태로 표현한 모델이다.

OOSE는 유스 케이스를 기반으로 한 개발 방법론으로 요구사항, 분석, 디자인, 구현, 테스트를 진행하는 방법론이다. OMT와 마찬가지로 UML의 초기 버전에 가까운 다이어그램을 사용 하고 있다. OOSE의 프로세스는 시스템과 사용자를 표현한 유스 케이스 다이어그램을 작성하고, 이를 바탕으로 요구사항 정의 및 디자인, 개발을 진행하는 순서로 구성 되어있다. 다음 그림 2는 유스 케이스를 기반으로 한 OOSE 프로세스에 대한 그림이다.

2.3 래셔널 통합 프로세스(Rational Unified Process)

래셔널 통합 프로세스는 Rational에서 제안한 UML 기반의 객체지향 개발 방법론이다[10,11]. 래셔널 통합

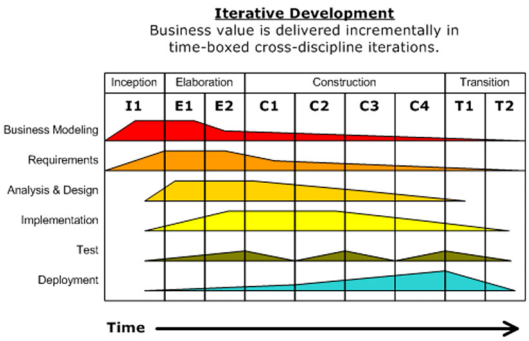

그림 3 래셔널 통합 프로세스 각 단계 및 순서[11]
Fig. 3 RUP phases and disciplines[11]

프로세스는 유스 케이스를 기반으로 사용자의 요구사항을 기본으로 반복적이고 점진적인 개발프로세스를 통해 시스템을 개발하는 UP(Unified Process) 프로세스 기반의 프레임워크 이며 개발 조직, 소프트웨어 프로젝트 팀 등이 필요에 따라 프로세스의 요소들을 선택하여 사용할 수 있도록 설계되어 있다.

그림 3에 나타난 바와 같이 래셔널 통합 프로세스는 요구사항 분석, 디자인, 구현, 테스팅, 배치(deployment)의 다양한 프로세스로 구성 되어 있으며, 이를 시작(inspection), 계획(elaboration), 구현(construction), 전달(transition) 페이즈에 거쳐 반복하며 필요에 따라 프로세스의 요소들을 조정하여 개발 프로세스를 구성할 수 있다.

2.4 관련 연구

기존의 대학 등에서 소프트웨어 공학 교육을 위해 교육용 개발 프로세스를 제안하거나, 기존의 프로세스들을 사용한 사례들이 있다. [12,13]과 같이 통합 프로세스(Unified Process)의 기초적인 구조를 교육에 사용하는 방법을 제안한 것과 [14,15]와 같이 새로운 교육용 개발

프로세스를 제안하는 사례들이 있다. 이외에도 다양한 개발 방법론들을 이용하거나 제안하는 리포트들이 존재한다[16,17].

[12,13,18]에서는 각각 통합 프로세스(unified process) 기반의 방법을 이용하여 소프트웨어 공학 교육에 적용한 사례에 대해 제시하고 있다. [12]에서는 통합 프로세스의 각 페이즈에 따라 디자인, 테스팅 등의 팀을 구성하고, 페이즈에 따라 프로젝트를 진행하는 방법을 취하고 있다. [18]에서는 소프트웨어 공학 교육에 래셔널 통합 프로세스를 사용하는 방법에 대해 제안하고, 그 경험을 설명하고 있다. [18]에서는 래셔널 통합 프로세스를 수정 하는 가이드라인 및 과정을 포함한 개발 프로세스를 사용하여 교육을 진행하였다. 위와 같은 사례들은 기존의 통합 프로세스를 그대로 사용하고 있다. 하지만 통합 프로세스를 교육과정에 그대로 적용하는 것은 교육목적에 불필요한 내용들이 있을 수 있다[15].

[15]에서는 교육용 소프트웨어 개발 프로세스를 개발시 필요한 요구사항들에 대해 정의하고, [13]에서는 [15]를 기반으로 기존의 상업용 개발 프로세스에서 적절한 수준의 개정을 거친 교육용 소프트웨어 개발 프로세스인 'Praxis'를 제안하고 있다.

'Praxis'의 프로세스는 디자인, 구현, 배치로 구성되어 있으며, 각각의 프로세스는 래셔널 통합 프로세스 등 기존의 프로세스에서 제안하는 요구사항 분석, 설계, 디자인, 구현 등의 내용을 요약해 제안하고 있다. 하지만 'Praxis'는 유저 인터페이스에 대한 고려가 부족한점이 있고, 설계 단계에서 상세한 내용 및 방법 제공에 대해서 학생들의 재량에 맞추도록 하는 등의 부족한점이 있다. 이처럼 소프트웨어 공학 교육을 위한 개발 프로세스 또는 프로세스 적용 방법을 제안하는 기존의 여러 연구들이 존재 한다. 본 논문에서 제안하는 OOPT는 해당 논문들과는 프로젝트의 주제를 직접 정의하는 점과, 요구사항 분석부터 테스팅까지의 추적성 확보, 객체지향 프로그래밍에서 GUI에 대한 직접적인 고려가 포함되는 등의 차이점을 가지고 있다.

3. Object Oriented Process with Traceability (OOPT)

본 논문에서 제안하는 OOPT는 래셔널 통합 프로세스를 대학 소프트웨어공학 수업에 활용 가능하도록 수정한 객체지향 소프트웨어 개발 방법론이다. 직접적인 소프트웨어 개발 외에도 다양한 소프트웨어공학 이론과 기술들을 습득할 수 있도록 추적성 분석, 테스팅 등 소프트웨어 공학교육에 필요한 내용들이 포함 및 추가되어 있으며, 수업의 주제에 따라 지속적으로 개선 가능하다.

OOPT는 개발 도중 요구사항 변경, 프로젝트 환경의 변경 등에 대처하고 피드백을 반영하기 위하여 반복적(Iterative)이고 점진적(Incremental)으로 소프트웨어를 개발할 수 있도록 구성되어 있으며, 크게 다음의 3개의 stage로 진행된다. 그림 4는 OOPT의 가장 상위 레벨의 프로세스에 대한 그림이다. Stage 1000 Plan and elaboration에서는 프로젝트의 요구사항 정의와 같은 전반적인 계획에 대하여 진행하고, Stage 2000 Build 단계에서는 실질적인 디자인 및 구현을, Stage 3000 Deployment 단계에서는 전체 시스템 구축에 대해 진행하며, 각 단계별 혹은 전체 프로세스별 반복하여 진행된다.

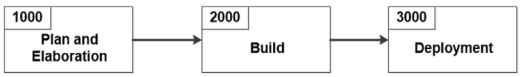

그림 4 OOPT 프로세스
Fig. 4 The development process of OOPT

3.1 Stage 1000: Plan and Elaboration

Stage 1000은 소프트웨어 개발 프로젝트를 기획하는 것으로서, 요구사항을 도출하고 프로젝트 계획서를 완성하는 것을 목표로 한다. 일반적으로 대학의 소프트웨어 공학 수업에서 실시하는 프로젝트들은 보통 미리 제시된 요구사항을 만족하는 소프트웨어를 개발하는 것을 목표로 하는 반면에, OOPT는 "OOO 그림판," "OOO 엘리베이터 컨트롤러," 등과 같이, 추상적으로 주어진 주제에 대해 구체적인 프로젝트 계획서와 요구사항(명세)을 직접 개발하는 것부터 시작하는 장점이 있다. 즉, 주어진 주제를 구현하는 것이 아니라, 주제를 직접 정하고 계획하는 단계부터 프로젝트를 시작하게 된다.

Stage 1000은 그림 5에 나타난 10 단계의 activity들로 구성되며, 각각의 상세한 내용 및 주요한 산출물은 표 1에 나타나 있다. Stage 1000에서 학생들은 각 단계의 activity 들을 통해 최종적으로 '프로젝트 개발 계획서'를 완성하게 된다. Activity 1001에서는 먼저 기본적인 내용들이 포함된 프로젝트 개발 계획서를 만든다.

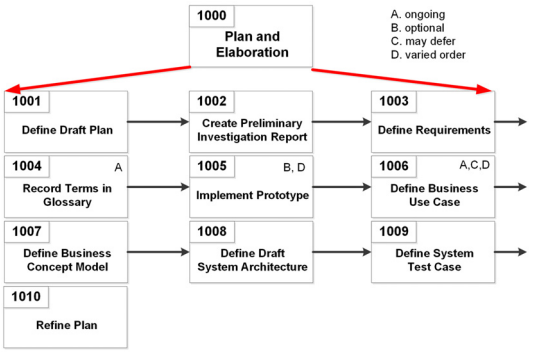

그림 5 OOPT의 stage 1000 activities
Fig. 5 Stage 1000 activities of OOPT

표 1 Stage 1000 activity 들의 명세
Table 1 Description of stage 1000 activities

Step	Activity description	Principal Output
1001.	Write a draft plan (e.g. schedule, budget, etc.)	Draft project plan
1002.	Write an investigation report (alternatives, needs, risk, etc.)	Investigation report
1003.	Write a requirement specification	Requirement specification
1004.	Dictionary of terms and any associated information	Term dictionary
1005.	Develop a prototype system (optional)	Prototype product
1006.	Write the business use cases	Business use case diagram, description
1007.	Identify "business concept" in the target domain	Business concept model
1008.	Construct a rough preliminary system architecture	A draft system architecture
1009.	Define test case of the system	System Test Cases
1010.	Refine the draft project plan	Refined project report

Activity 1002는 우리(학생팀)가 왜 이 소프트웨어를 직접 개발해야 하는가에 대한 당위성을 비즈니스 관점에서 분석해 보는 단계로서, 보통의 소프트웨어 개발 프로젝트에서는 다루지 못한 내용을 다수 포함한다.

Activity 1003에서는 기능/비기능 요구사항들을 도출해내며, 필요한 경우 프로토타입을 가볍게 개발해 볼 수도 있다(1005). Activity 1007은 1003에서 개발한 요구사항들을 상위 수준의 유스 케이스와 매핑하는 단계로서, OOPT에서 가장 중요한 시작점이 된다. OOPT는 각 요구사항과 유스 케이스가 1:1 관계를 이루도록 하며, 유스 케이스를 구체적으로 자세히 개발하지 않고 시스템 요구사항 수준으로만 한정한다. 현재 단계의 유스 케이스들은 시스템을 표현하는 상위 수준의 유스 케이스로써 이후 과정을 진행하면서 상세히 정의하고, 작성하는 과정을 거쳐 디자인에 사용된다.

시스템 시험(1009) 계획서도 Stage 1000에서 도출되어야 하는 중요한 산출물 중의 하나이다. 요구사항분석이 막 완료되어서 개발자가 개발할 시스템의 요구사항에 대해 잘 기억하고 있는 이 시점에서 시스템 테스트 케이스 (및 계획서)를 개발하는 것이 중요하다. 1010 단계를 거쳐서 최종적으로 "프로젝트 개발 계획서"가 완성되며, 이 문서를 기반으로 "보스"를 설득하기 위한 "발표자료"를 만들게 된다. 실제 수업에서는 기술적인 내용을 전달하는 technical presentation이 아닌, 상대방을 기술적으로 설득하는 business presentation을 하도록 지도한다. 이를 통해 졸업 후 입사했을 때, 주어진 업무만을 수행하면 되는 일반 사원이 아닌 새로운 과제를 만들어 내야 하는 관리자(Manager)가 갖추어야 할 덕목들을 미리 체험해 볼 수 있게 하는 것이 이 단계의 중요한 목적 중의 하나이다.

3.2 Stage 2000: Build

"프로젝트 개발 계획서"가 완성되고, (가상의) 보스로부터 과제승인을 성공적으로 획득했다면, 이제 구체적으로 소프트웨어 개발을 시작할 수 있다. Stage 2000 Build는 다음과 같이 여러 사이클로 구성될 수 있으며, 각 사이클은 온전한 6개의 페이즈로 구성된다. 다음 표 2와 그림 6은 stage 2000의 각 cycle 항목에 대한 설명 및 그림이다. 반복은 각 단계들이 페이즈 별로 반복되어 진행 된다.

표 2 Stage 2000 cycle 들의 명세
Table 2 Description of stage 2000 activities

Step	Activity description	Principal Output
2010	Refine or modify the plan	Revised plan
2020	Synchronize the plan activity	synchronized plan
2030	Analyze the system	Sequence diagram, etc.
2040	Design the system	Class diagram
2050	Construct the class, method, GUI etc.	Implementation
2060	Execution the tests of each activities	Testing reports

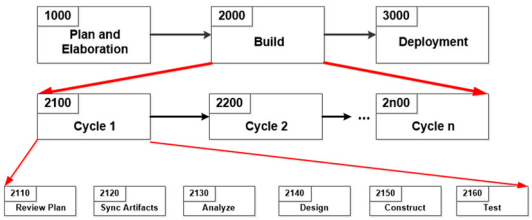

그림 6 OOPT의 stage 2000 cycle
Fig. 6 Stage 2000 activities of OOPT

Stage 2010 Revise plan과 2020 Synchronize Artifacts는 이전 사이클을 완료한 후, 수정이 필요한 내용이 확인되면, 이를 각 단계의 내용들에게 일괄적으로 반영하는 단계이다. 실제적인 개발은 2030 Analyze부터 시작하여 2040 Design, 2050 Construct 및 2060 Test 단계로 폭포수 모델과 같이 순차적으로 분석, 디자인, 구현, 테스팅이 진행된다. 각 단계는 보다 구체적인 Activity 들로 구성된다.

3.2.1 Stage 2030: Analyze

실제적인 개발의 첫 단계인 Stage 2030 Analyze는 9개의 순차적인 Activity로 구성되어 있다. 대부분의 전산

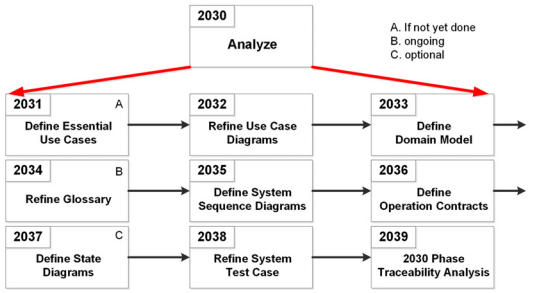

그림 7 OOPT의 stage 2030 activities
Fig. 7 Stage 2030 activities of OOPT

표 3 Stage 2030 activity 들의 명세
Table 3 Description of stage 2030 activities

Step	Activity description	Principal Output
2031	Add event flows to business use case	Essential use cases
2032	Validate and modify the use case diagram	Refined use case diagram
2033	Define domain concept model	A conceptual class diagram
2034	List and refines all the terms	A refined dictionary
2035	Illustrates events from actors to systems	A sequence diagram
2036	Define contracts for system operations	Operation contracts
2037	Describes all possible states of the system, use cases, or objects	A state diagram
2038	Refine system test plan and case	Refined system test plan
2039	Analysis the connection of results in 2030 step	Traceability analysis results

학과 소프트웨어 개발 수업/프로젝트는 이 단계에서부터 시작하거나 이 단계 결과물의 많은 부분을 제공한 후 시작하는 경향이 있다. 해당 페이즈는 대상 시스템(프로그램)과 사용자 간의 상호작용의 관계에 대한 동작을 포함하여 시스템의 기초적인 동작이나, 구성에 대해 분석하는 단계이다. 그림 7과 표 3은 stage 2030의 Activity 및 명세에 대한 그림과 표이다.

먼저 2031 Define Essential Use Case부터 시작하여 기존의 1007에서 개발했던 Business Use Cases를 보다 많은 정보를 담고 있는 Essential Use Case로 확장한다. Class Diagram의 원형을 개발(2033)한 후에는, 가장 중요한 작업인 System Sequence Diagram (SSD)을 정의하게 된다(2035). SSD는 시스템 수준으로 작성된 Sequence Diagram으로서, 개발할 소프트웨어 시스템은 블랙박스로 간주한 후, 이의 입출력만을 활용하여 외부 액터와 개발할 시스템 간의 event sequence를 찾는 작업이다.

SSD는 2031의 Essential Use Case 별로 작성되며, 최종 완성된 System Sequence를 모아서, 개발할 시스템이 제공해야 할 system operations으로 정의한다(2036). Activity 2039는 각 요구사항부터 유스 케이스, 시스템 동작 등의 도출/연결 관계를 모두 분석해서 시각적으로 표현하는 단계이다. 이와 같은 추적성 분석은 개발이 완료되는 테스팅 단계(2060)까지 지속적으로 반복해서 수행된다. 추적성 분석을 통해 요구사항부터 테스팅까지 시각적으로 구현되는 관계를 파악해 수정 사항 혹은 문제점 발생 시 이를 쉽게 파악하고 수정하는데 도움이 될 수 있다는 장점이 있다.

3.2.2 Stage 2040: Design

2040 Design 단계에서는 2030 Analyze에서 분석한 내용들을 보다 구체화하여, 구현을 시작하는데 필요한 모든 정보를 완성/준비하는 것을 목표로 한다. Stage 2040 단계에 대한 구성은 그림 8과 표 4에 나타나 있다. 먼저 개발할 소프트웨어의 GUI/UI를 고려하여 2031의 Essential Use Case를 Real Use Case로 확장(2041)한 후, 다양한 UML Diagram을 활용하여 클래스 다이어그램을 온전하게 완성한다. 'Sequence Diagram,' 'Collaboration

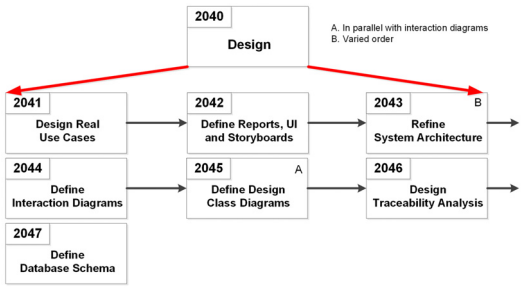

그림 8 OOPT stage 2040 activities
Fig. 8 Stage 2040 activities of OOPT

표 4 Stage 2040 activity 들의 명세
Table 4 Description of stage 2040 activities

Step	Activity description	Principal Output
2041	Define Real Use Cases	Real use cases
2042	Design UI component and storyboards	UI concept
2043	Refine draft system architecture	Package diagram
2044	Illustrate how objects interactions via messages	interaction diagram
2045	Describes details in conceptual class diagram	Class diagram
2046	Analyze the connection the results of 2040 steps	Traceability analysis result
2047	Design database, table, and records	A database schema

Diagram' 및 'Statecharts Diagram' 등은 모두 클래스 다이어그램의 내용들을 채우기 위해 사용되는 것으로서, 유스 케이스 당 다수 개의 다이어그램을 그림으로써(분석함으로써), 정의된 오브젝트(클래스) 간에 어떤 커뮤니케이션과 데이터가 필요한지 확인하고, 또 필요한 정의를 추가하게 된다.

2040 Design 단계에서는 GUI에 대한 충분한 고려를 통해, 2050 Construct(개발) 단계에서 있을 수 있는 디자인-구현 간의 Semantic Gap을 최소화해야 한다. JAVA나 Visual C++ 등을 통해 개발하는 소프트웨어는 GUI를 포함하는 것이 일반적이며, 그래픽관련 부분은 2030 Analyze 단계에서 고려되지 않는다. 따라서, 2040을 통해 컨트롤러나 main GUI 등의 Interface/Abstract 클래스를 가상으로 구성한 후, 실제 구현에서는 실 사용한 라이브러리 함수와 연결시켜 주는 작업이 추가적으로 요구된다. 이는 2044에서 정의하는 Sequence Diagram을 통해 분석되며, 최종적으로 2045 Design Class Diagram에 반영된다. 필요에 따라서 data-flow diagram이나 flow-chart 등을 이용해 추가적인 명세를 하는 것이 디자인에 유리할 수 있다.

Activity 2046에서는 이전 stage들과 마찬가지로 추적성 분석을 수행한다. 2040 stage에서 수행하는 추적성 분석은 이전 과정을 거치며 분석한 내용들이 클래스 다이어그램으로 연결되는 관계를 분석해서 시각적으로 표현하는 작업을 수행한다.

3.2.3 Stage 2050: Construct

2050 Construct 단계는 2040의 최종 산출물인 Class Diagram을 기반으로 실제 프로그램 코드를 작성하는 단계로 실제로 구현할 클래스와 method 정보를 이용해 단위시험 테스트 케이스 작성하는 과정까지 포함된다. 다른 단계에 비해 적용되는 기술/기법이 없는데, 이는 프로그램 개발(코딩)은 개발자의 능력에 크게 의존하는 특성에 기인한다. 중요한 점은, 2040 Design 단계의 다양한 산출물들을 활용하여, 다른 개발자가 독립적으로 문제없이 구현을 완료할 수 있어야 한다는 점이다. 개발자가 클래스 다이어그램을 구성하는 함수들의 실제 내용을 구성할 때, 2041 Real Use Case와 2044 Interaction Diagram 등을 참고하며, 이들에 구현에 필요한 충분한 내용이 포함되어 있어야 한다. 표 5와 그림 9는 OOPT 2050 stage의 활동들에 대해 설명한 그림이다.

2051 단계는 클래스를 구성하는 함수들을 구현할 때, 함수의 동작원리/내용/알고리즘 등을 포함하는 간략한 명세서를 포함하도록 하고 있으며, 이는 2055 Unit Testing용 테스트 케이스를 개발할 때, 유닛에 대한 명세서로서 활용될 수 있다. 2052 단계는 이전 단계에서 작성한 UI story board를 기반으로 하여 GUI를 통한 사용자와 시스템 간의 실제 상호작용에 대해 정의하고, 클래스 및

표 5 Stage 2050 activity 들의 명세
Table 5 Description of stage 2050 activities

Step	Activity description	Principal Output
2051	Implement class & methods by class diagram	Class & Methods description
2052	Define relations between GUI and operation	GUI implements results and description, refined class diagram
2053	Refined and implement the reports again	Analysis, design step reports
2054	Implement DB Schema	DB scheme
2055	Writing test code for unit testing	Unit test code

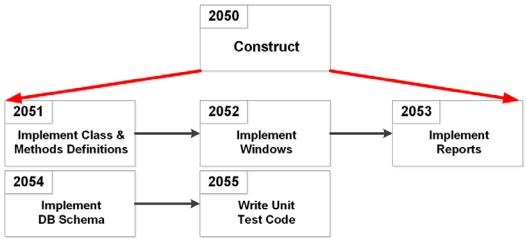

그림 9 OOPT stage 2050 activities
Fig. 9 Stage 2050 activities of OOPT

method를 연결하는 작업이다. 2052 activity의 작업을 통해서 UI 구성에 대한 명세서를 작성하게 된다. 2050 단계에서는 이전 단계들의 결과를 통해 구현하는 단계로 따로 추적성 분석을 수행하지 않는다.

3.2.4 Stage 2060: Testing

Stage 2060 단계에서는 앞에서 준비한 각 테스팅 계획서에 의거해서 해당 수준의 테스팅을 진행한다. Stage 2060 단계에서 진행되는 테스팅 종류는 총 6 가지로 구성되어 있지만, 대학 수업에서는 일반적으로 단위 시험(Unit Testing)과 시스템 시험(System Testing)이 사용된다. 이외의 다른 테스팅은 생략되는 것이 일반적이다. 다음 그림 10 및 표 6은 OOPT의 2060 stage의 활동들에 대한 설명이다.

단위 시험은 이전 stage에서 작성한 단위 시험 코드(Unit Testing Code)를 이용하여 수행한다(2061). 시스템 시험은 이전 단계들 에서 정의한 시스템 시험 계획(System Testing Plan)에 실질적으로 테스팅을 수행하기 위한 테스트 데이터(Test Data)를 생성하고, 수행하게 된다(2063). 특이한 점은, 최종적으로 2067 Testing Traceability Analysis를 수행하는 것이다. 요구사항 명세서의 요구사항과 유스 케이스부터 시작해서, 해당되는 Code/Class 및 해당되는 단위/시스템 테스트 케이스를 모두 연결하여 시각적으로 보여주는 작업이다. 이 작업을 통해, 전방/후방의 추적성 분석을 충분히 수행할 수 있다.

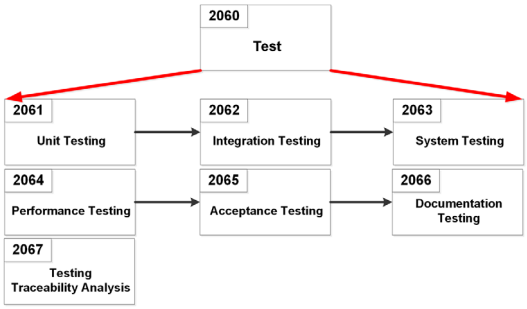

그림 10 OOPT Stage 2060
Fig. 10 Stage 2060 activities of OOPT

3.3 Stage 3000: Deployment

Stage 2000 Build에서 다수의(보통 대학 수업에서는 3회 정도) cycle을 수행한 후에는 최종적으로 Stage 3000 Deployment를 수행한다. 이는 개발한 프로그램을 고객의 특성에 맞게 부분 수정한 후 개발을 완료하는 것을 의미한다. 예를 들면, 국내 핸드폰 개발사에서 새 모델을 개발한 경우, 기본적인 기능은 같지만, SKT, KT 등의 통신사에 맞게 로고나 화면, 음악, 통신사 전용 프로그램 등을 추가적으로 설치해야 하는데, 이 작업은 Deployment에 해당된다. 따라서, 대학 수업에서는 이 단계를 수행하지 않는다.

표 6 Stage 2060 activity 들의 명세
Table 6 Description of stage 2050 activities

Step	Activity description	Principal Output
2061	Unit Testing	Unit testing reports
2062	Integration Testing	Integration testing reports
2063	System Testing	System testing reports
2064	Performance Testing	Performance testing reports
2065	Acceptance Testing	Acceptance testing reports
2066	Documentation Testing	Testing reports
2067	Testing Traceability Analysis	Traceability reports

4. 사례 연구

아래의 표 7은 건국대학교 컴퓨터공학과의 소프트웨어공학 관련 수업(소프트웨어 모델링 및 분석) 수업에서

표 7 OOPT를 이용해 진행한 프로젝트 명세
Table 7 Description of projects with OOPT

Year	Subject	Description	Team Subject (selected by each team)	Specification
2010	System for service	Develop the system for services.	- Safety web mail system	
2011	OOO coffee maker	Develop the coffee maker system.	- Smart DJ coffee maker - Customized coffee maker, Etc.	
2012	OOO data management system	Develop the specific data management system.	- Selectable parking navigation system - ASSIO data management system, Etc.	
2013	OOO Painter	Develop the specific-verifiable painter program.	- Children education paint - VAT paint, Etc.	OOPT had lack of the process about considering of GUI, while the focus of the project was GUI. → We modify the OOPT process to consider GUI implementation.
2014	OOO elevator simulator	Develop the simulator which is able to show operation (control) of the elevator.	- RE-ason Elevator Control simulator - No Wait ECS, Etc.	This project made confusion between elevator operation algorithm and elevator simulation. → Educating the importance of requirements analysis was possible.
2015	OOO child English learning program	Develop the English learning program which is able to show characteristics of each team for children.	- TALKIDS - English education for kids (Sound), Etc.	GUI was still difficult to design in OOPT. Opinions that writing document needs too many effort existed.
2016	OOO Clone Checker	Read 40 programs written in the C language and find the replication programs (This project does not use compiler techniques).	- Pleasant Clone Checker - One More Chance, Etc.	This project did not fit the OOAD because clone checking algorithm is the main issues.

OOPT를 활용하여 진행한 팀 프로젝트를 소개한다. 다양한 주제로 진행되어 왔으며, OOAD 방법론인 OOPT에 적합한 주제도 있었으나, 적합하지 않은 주제들도 수행하였다. 표 7에 나타난 바와 같이 해당 수업을 통해 'coffee maker system'부터 'clone checker'까지 매년 주제를 변경해 가면서 프로젝트 수업을 진행하였다.

2013년도에 진행한 'OOO Painter' 프로젝트는 학생팀별로 특정한 내용을 갖는 그림판을 OOPT를 통해 개발하는 프로젝트이었다. 이 경우 그림판에 대해 분석, 디자인, 개발하는데 GUI가 중심이 되어야 하는 점에 반해 OOPT가 상대적으로 GUI 표현이 약한 점을 파악 하게 되어 보완할 수 있는 기회가 되었다. 2014년도에 수행했던 'OOO elevator simulator' 프로젝트는 엘리베이터 컨트롤러를 시뮬레이션 할 수 있는 프로그램 개발을 목표로 진행을 하였는데, 많은 학생들이 요구사항을 명확히 분석하지 않아 구현 단계에서 많은 어려움을 겪는 등 요구사항 분석의 중요성을 학생들에게 전달할 수 있는 프로젝트였다. 이처럼 다양한 주제를 변경해가며 프로젝트를 진행하고, 학생들의 의견을 수렴하면서 OOPT에 대해 보완 작업을 진행하고 사례연구를 수행하였다.

4.1 사례연구: 'OOO Clone Checker'

2016년에 진행했던 'OOO Clone Checker'는 C 언어 코드를 입력으로 받아서 유사도를 검사하는 프로그램을 개발하는 프로젝트로 OOPT를 활용해 진행 하였다. 각 팀별로 유사도를 검사하는 알고리즘, 시각화하기 위한 방법 등에 대해 주제를 정해 수행하였다. 프로젝트의 진행은 총 6번의 결과 보고서 제출 및 발표를 거쳐 진행되었다. 진행 순서는 OOPT의 Stage 1000, 2030, 2040, 2050/60 및 2nd cycle, 3rd cycle 순으로 진행하여 OOPT의 개발 과정을 거치고 반복적(iteration)으로 진행하는 특징을 경험할 수 있도록 진행되었다.

그 중 하나에 대해 설명하면 다음과 같다. 해당 팀은 '쾌적한'을 주제로 '쾌적한 Clone Checker'를 목표로 프로젝트를 수행 하였다. '쾌적한'의 정의는 기능/비기능 요구사항에 대한 정의가 모두 포함되어 '비교 작업과 결과 출력의 쾌적함으로' 정의하고, 몇 초 이내의 비교시간 및 여러 개의 파일의 결과를 디스플레이를 통해 쾌적하게 보여주는 것을 목표로 하였다.

해당 프로젝트 팀은 반복적인 cycle을 진행하는 OOPT에 따라 각 stage를 여러 번 반복하여 진행하였다. Stage 1000단계는 7번, 2030 분석 단계는 5번, 2040 디자인 단계는 4번 마지막 2050, 2060 단계는 총 3번의 반복을 진행하였으며, 이는 일정이 진행되면서 다음 단계를 수행하고, 다시 반복하는 과정을 거쳐 진행하였다.

OOPT stage 1000을 거치며 22개의 functional requirements와 4개의 non-functional requirements를 개발하고 이에 맞추어 use-case diagram을 작성한 것을 확인 할 수 있다. 또한 stage 2030, 2040을 거치며 essential use case, real use case 정의 및 system sequence diagram, interaction diagram등을 통해 최종적으로 class diagram을 개발한 것을 확인할 수 있다. 그림 11은 각 stage의 주요 결과물들 중 일부에 대한 화면으로 각각 위에서부터 유스 케이스 다이어그램, 추적성 분석 결과이다.

Stage 2050, 2060 은 각각 구현 및 테스팅에 관련된 stage 들로써 최종적으로 테스팅을 거쳐 구현이 완료된다. 그림 12는 2050, 2060 단계를 거쳐 작성한 테스트 케이스 및 최종적으로 구현된 프로젝트의 화면이다.

하지만 'OOO Clone Checker' 프로젝트의 경우에는 프로젝트의 주된 이슈가 clone checking이기 때문에 내부 알고리즘 디자인, 데이터 흐름, 기능 흐름 등을 정의할 수 있는 activity가 있으면 더 좋을 것 같다는 의견이 있었다. 각 stage들 간의 연관성을 고려해 문서작업을 효율적으로 할 수 있으면 좋을 것 같다는 의견도 확인 할 수 있었다. 또한 학생들의 프로젝트 결과에서 추적성 분석이 시스템의 흐름, 연결 관계를 효과적으로 파악하는데 도움이 되지만, 시스템이 복잡해지고 프로세스가 진행되면서 많은 데이터가 추가되는 경우 시각적인 분석이 쉽지 않게 된다는 문제도 확인할 수 있었다.

4.2 OOPT 개선 방향에 대한 고찰

소프트웨어공학 수업을 위해 개발된 교육용 개발 방법론인 OOPT는 해당 수업에서 강조하는 내용/기법에 따라 다양한 branch를 만들어 활용할 수 있다. 그림 13은 소프트웨어 개발 생명주기에 따라 OOPT를 통해 개선하거나 변경시켜 적용할 수 있는 방법들에 대한 예제이다.

개선 및 변경을 위해 디자인/구현에 집중하여 디자인 시 구조를 강조하거나, GUI 혹은 알고리즘에 집중할 수 있는 사항들에 대해 추가적인 프로세스를 도입하는 등의 방법을 생각해 볼 수 있다. 또는 V-모델을 고려하여 테스팅을 강조하는 방향이나 전체적인 추적성을 강조하는 방향도 생각해 볼 수 있다. 또한 개발 생명주기 전체적으로 생각해서 오픈소스를 활용하거나, 오픈소스 프로젝트 기반, 오픈소스 개발 기반 혹은 전반적인 co-operation을 고려하는 방향으로도 개선 및 확장을 생각해 볼 수 있다. 이처럼 각 수업의 주제에 맞도록 여러 방향으로 개선된 프로세스를 통해 다양한 주제의 소프트웨어 공학 교육을 할 수 있을 것으로 생각된다.

5. 결론 및 향후 연구

본 논문에서는 건국대학교 컴퓨터공학과에서 소프트웨어공학 관련 수업에서 사용중인 객체지향 소프트웨어 개발 방법론인 OOPT를 자세히 소개하고, 지난 10년 동안 OOPT를 활용해 수행한 프로젝트들을 소개 및 분석

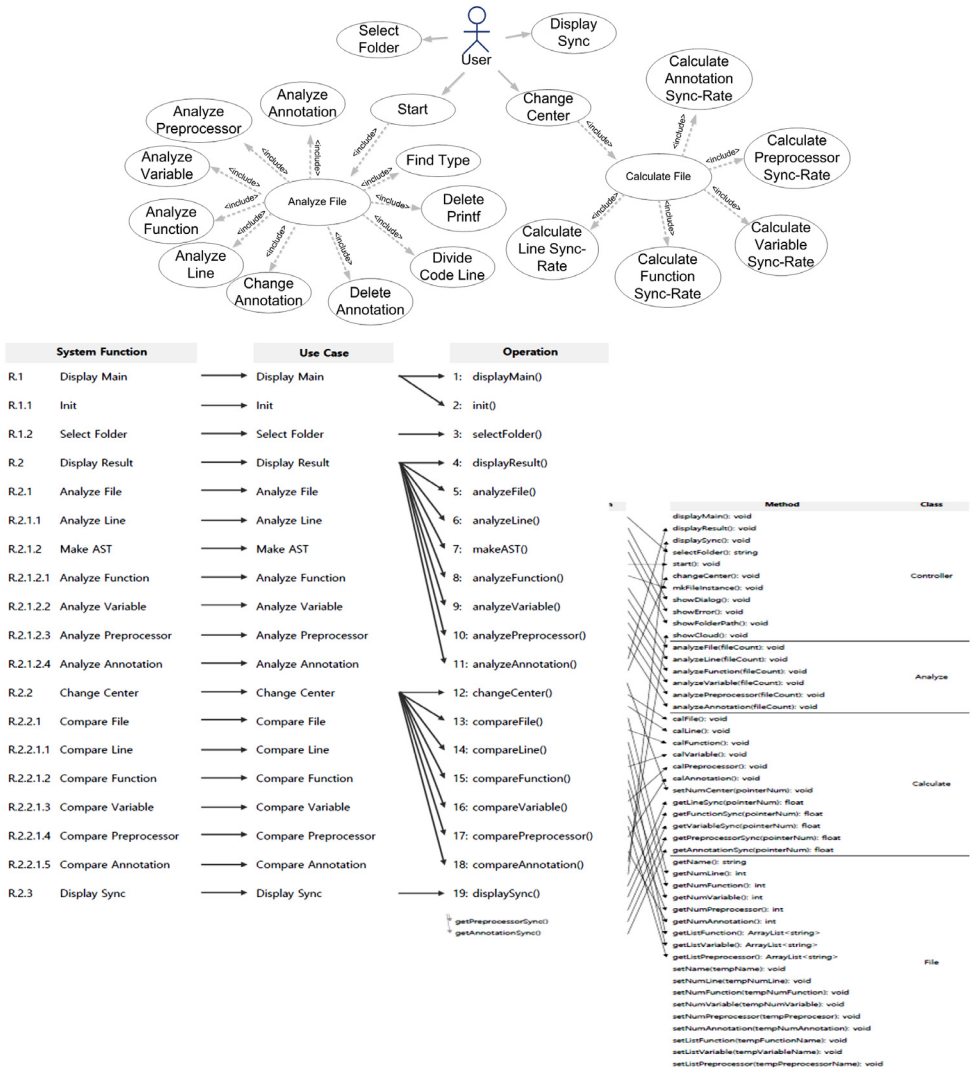

그림 11 각 stage 별 활동의 주요 결과화면(유스 케이스, 추적성 분석 결과)

Fig. 11 Screen dumps on results of OOPT stages (use case diagram, traceability analysis)

Test No.	Test 항목	Description	Use Case	System Function
1-1	폴더 선택 시험	사용자가 선택한 폴더의 경로가 변수 *Controller.folderPath*에 잘 들어갔는지 확인한다.	1. Select Folder	R.1
1-2	폴더 선택 시험	폴더 선택 버튼을 눌렀을 때 폴더 선택 창이 잘 나타나는지 확인한다.	1. Select Folder	R.1
1-3	폴더 선택 시험	사용자가 선택한 폴더의 경로가 정상적으로 출력되는지 확인한다.	1. Select Folder	R.1
1-4	폴더 선택 시험	비정상적인 폴더 경로를 선택하였을 경우 오류 메시지가 팝업 창으로 나타나는지 확인한다.	1. Select Folder	R.1
1-5	폴더 선택 시험	폴더 선택이 완료되었을 경우 '시작하기' 버튼이 활성화되는지 확인한다.	1. Select Folder	R.1

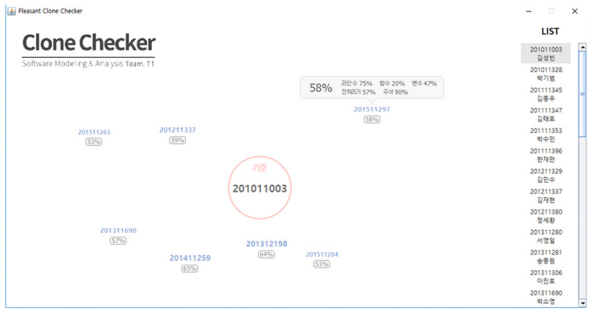

그림 12 쾌적한 clone checker 테스트케이스 및 최종 산출물

Fig. 12 A screen dump of the test case and the clone checker's final output

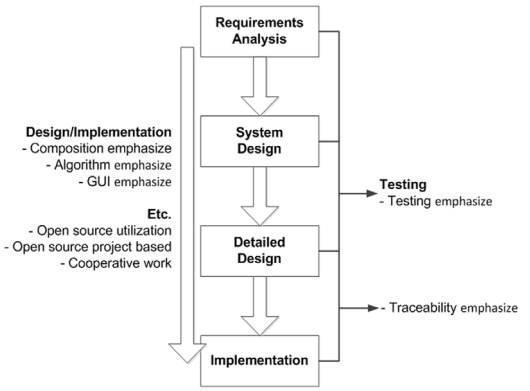

그림 13 소프트웨어 생명주기별 OOPT 개선 방향 및 방법 예제

Fig. 13 Example of OOPT improvement methods in accordance with the software lifecycle

하였다. OOPT는 다양한 단계와 cycle을 갖도록 구성되며, 소프트웨어 공학 관련 교육을 위해 요구사항 정의부터 다양한 내용이 포함되어 있다.

또한 수업을 통해 진행한 프로젝트들을 바탕으로 교육에 도움이 될 수 있는 개선 및 활용 방안에 대한 분석도 개진하였다. OOPT 또는 다양한 객체지향 소프트웨어 개발 방법론이 발전/개발되어, 소프트웨어 공학 교육에 효과적으로 사용되기를 기대한다. 향후 본 논문에서 소개한 여러 개선 방향들을 이용해 OOPT를 학부/대학원 수업 등을 통해 지속적으로 보완할 계획이다. 또한 추적성 분석과 관련해 효과적인 시각화 및 적용 방법에 대한 연구와 프로세스를 진행하며 수행되는 여러 작업들의 문서화에 대해 노력을 줄일 수 있는 방법을 보완할 계획이다.

References

[1] P. Naur, B. Randell, "Software Engineering: A Report on a Conference Sponsored by NATO science Committee," NATO, 1969.

[2] Winston W Royce, "Managing the development of large software systems," *Proceedings of IEEE WESCON*, Vol. 26, No. 8, pp. 328-338, 1970.

[3] Ian Sommerville, "*Software Engineering*," Addison-Wesely, Boston, 2007.

[4] FA Masoud, MU Shaikh, SH Mustafa, "SASD methodology from a practical point of view problems and suggestions for improvement," *WIT Transactions on Information and Communication Technologies*, Vol. 17, pp. 93-102, 1997.

[5] Grady Booch, "*Object-oriented Analysis and Design with Applications 3rd edition*," Addison-Wesley, Boston, 2007.

[6] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen, "*Object-Oriented Modeling and Design*," Prentice Hall, New Jersey, 1990.

[7] L. Jacobson, "*Object-Oriented Software Engineering: A Use Case Driven Approach*," Pearson Education, India, 1993.

[8] Derek Coleman, Patrick Arnold, Stephanie Bodoff, Chris Dollin, Helena Gilchrist, Fiona Hayes, Paul Jeremaes, "*Object-oriented development: the fusion method*," Prentice-Hall, Inc, 1994.

[9] Boehm B, "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, Vol. 21, No. 5, pp. 61-72, 1988.

[10] Per Kroll, Philippe Kruchten, Grady Booch, "The Rational Unified Process Made Easy a Practitioner's Guide to the RUP," Addison-Wesley, Boston, 2003.

[11] Philippe Kruchten, "The rational unified process: an introduction," Addison-Wesley Professional, Boston, 2004.

[12] Michael Halling, Wolfgang Zuser, Monika Kohle, Stefan Biffl, "Teaching the Unified Process to Undergraduate Students," *IEEE 15th Conference on Software Engineering Education and Training*, pp. 148-159, 2002.

[13] Jiang Li, "Teaching unified process in software design and development courses - A case study," *Journal of Computing Sciences in Colleges*, Vol. 24, No. 5, pp. 5-11, 2009.

[14] Wilson P. Paula Filho, "An Educational Software Development Process," *Proc. of the International Conference on Computer Science, Software Engineering*, Information Technology, E-Business and Applications, 2002.

[15] Wilson P. Paula Filho, "Requirements for an Educational Software Development Process," *ACM SIGCSE Bulletin*, Vol. 33, No. 3, pp. 65-68, 2001.

[16] P. Runeson, "Experience from teaching PSP for freshman," *Proc. of the 14th conference on Software Engineering Educational and Training*, pp. 98-107, 2001.

[17] N. Davis, J. Mullaney, "The Team Software Process(TSP) in Practice: A Summary of Recent Results," Technical Report CMU/SEI-2003-TR-014, Software Engineering Institute, 2003.

[18] G. Kelecic, Z. Car, "Teaching Software Process: An Experience in Implementing RUP in a Student Project," *Proc. of the 8th International Conference on Telecommunications*, pp. 479-484, 2005.

정 세 진

2015년 건국대학교 컴퓨터공학과 졸업(학사). 2016년 건국대학교 컴퓨터 정보통신 공학과 졸업(석사). 2016년~현재 건국대학교 컴퓨터 정보통신공학과 박사과정. 관심분야는 소프트웨어 공학, 위해도/안전성 분석

이 동 아

2010년 건국대학교 컴퓨터공학과 졸업(학사). 2012년 건국대학교 컴퓨터 정보통신 공학과 졸업(석사). 2012년~현재 건국대학교 컴퓨터 정보통신공학과 박사과정. 관심분야는 정형검증, 소프트웨어 확인 및 분석, 위해도 분석

김 의 섭

2012년 건국대학교 컴퓨터공학과 졸업(학사). 2015년 건국대학교 컴퓨터 정보통신 공학과 졸업(석사). 2015년~현재 건국대학교 컴퓨터 정보통신공학과 박사과정. 관심분야는 소프트웨어 공학, 정형검증

장 천 현

1979년 KAIST 전산학 전공 졸업(석사) 1988년 KAIST 전산학 전공 졸업(박사) 1984년~현재 건국대학교 컴퓨터공학부 교수. 관심분야는 프로그래밍 언어, 컴파일러, 객체지향, 실시간처리

유 준 범

2005년 KAIST 전자전산학과 전산학 전공 졸업(박사). 2008년 삼성 전자주식회사 통신연구소 책임연구원. 2008년~현재 건국대학교 컴퓨터공학부 부교수. 관심분야는 소프트웨어 공학, 안전성 분석, 정형 기법