대학생을 위한 Dependability 교육 – SW 테스팅

A Dependability Education Class for Undergraduate Students – SW Testing

유 준 범

건국대학교 컴퓨터공학부
서울시 광진구 화양동 건국대학교
jbyoo@konkuk.ac.kr

요약: 최근 국내 기술로 개발된 원자력발전소와 고속철도가 수출되고 독자적으로 인공위성을 발사하는 등 dependable 소프트웨어에 대한 관심이 증가하고 있다. 본 연구에서는 소프트웨어 dependability 를 보장하기 위한 기법 중 하나인 소프트웨어 테스팅(Software Testing)을 대학 4 학년을 대상으로 교육한 내용과 경험을 소개한다.[1]

핵심어: Dependability, 소프트웨어 테스팅, 교육

1. 서론

다양한 안전최우선시스템(Safety-Critical System)이 점차 국산화됨에 따라, 이들의 안전성(Safety)과 정확성(Correctness)에 대한 관심이 증가하고 있다 [1]. 원자력발전소, 인공위성, 고속철도, 미사일 시스템 등은 안전최우선시스템의 예로서, 작은 오작동은 단순한 미션 실패뿐만 아니라, 대규모의 인명피해와 물적 손실을 야기할 수 있다. 안전성과 정확성은 시스템의 dependability 로 통칭되며, 이러한 시스템을 dependable system 이라고 부른다 [2]. Dependable system 은 대부분 내장형시스템(Embedded System)이며, 소프트웨어가 dependability 의 중요한 역할을 담당한다. 따라서, 소프트웨어 dependability 는 시스템 dependability 를 보장하기 위한 필수 사항으로서, 국내외적으로 활발히 연구되고 있다.

소프트웨어 dependability 를 보장하기 위한 연구는 크게 소프트웨어 V&V(Verification & Validation) [3]과 안전성분석(Safety Analysis) [4]으로 구분할 수 있다. V&V 는 정형기법(Formal Methods) [5], 테스팅(Testing) [6] 및 프로그램 정적 분석(Static Analysis) [7]을 포함한다. 이들은 모두 대학원에서 집중적으로 연구되는 분야로서, 학부 학생에게는 해당 분야에 대한 소개만을 해왔다. 하지만, 국내외적으로 소프트웨어 dependability 에 대한 관심이 증폭됨에 따라, 학부 졸업생들도 해당 분야에서 종사할 가능성이 높아졌고, 다양한 소프트웨어공학 수업[2]을 통해 학생들의 학문 수준도 예전에 비해 상당히 상승되었다. 따라서, 이제는 학부 수업에서도 이러한 dependability 연구들을 자세히 소개하고 직접 실습함으로써, 학생들이 졸업 후에 dependability 관련 분야에서 차별된 전문성을 가질 필요가 있게 되었다.

이러한 필요성에 따라, 건국대학교 컴퓨터공학부에서는 2009 년부터 4 학년 학생을 대상으로 '소프트웨어 검증' 수업을 개설하여, dependability 기법과 이론을 교육하고 있다. 2009 년도에는 정형기법 (정형명세 및 검증)을 주제로, 정형기법의 기본 이론을 강의하고, NuSCR [8], SMV [9], UPPAAL [10] 등의 정형도구들을 사용하여 팀 프로젝트를 수행하였다. 하지만, 수업 내용이 학부 4 학년 학생들이 완벽하게 소화하기에 어려움이 있었다. 정형기법을 정확하게 사용하기 위해서는 기본 이론과 원리를 이해해야 하는데, 정형언어, 오토마타(Automata)나 논리(Logic)와 관련된 학부 수업의 부족함에 기인하는 것으로 판단된다.

2010 년도에는 dependability 보장 기법의 다른 큰 분야인 소프트웨어 테스팅을 주제로, 테스팅의 기본 이론과 함께 CTIP(Continuous Test & Integration Platform) [11]을 JAVA 프로그램을 대상으로 구축하고, 직접 테스팅을 수행하였다. 본 연구에서는 이 수업의 자세한 내용과 학생들이 수행한 팀 프로젝트 결과물 및 강의평가 내용을 소개한다. 본 논문의 구성은 다음과 같다. 2 장에서는 2010 년 소프트웨어 검증 수업 내용을 소개하며, 3 장에서는 학생들이 수행한 팀 프로젝트 결과와 학생들의 강의평가 결과를

[1] 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT 연구센터 지원사업의 연구결과로 수행되었음. (NIPA-2010-C1090-0903-0004, NIPA-2010-C1090-1031-0003)

[2] 건국대학교 컴퓨터공학부 2009 년 2 월 기준, '소프트웨어 공학개론', '소프트웨어 모델링 및 분석', '소프트웨어 설계방법론', '소프트웨어 검증' 4 개 교과목 개설

정리한 후, 4 장에서 논문을 마무리하였다.

2. 2010 소프트웨어검증 수업

2.1 이론 수업

2010 년도 '소프트웨어 검증' 수업[12]은 소프트웨어 테스팅의 기본 이론을 이해하고, 이를 실제 프로젝트에 적용함으로써, 졸업 후 현장 업무에 적용 가능한 능력을 갖추는 것을 목표로 한다. 이 수업은 총 14 명의 4 학년 학생이 수강하였다. 수업은 이론 강의와 팀 발표로 구성되었으며, 총 5 회의 팀 별 발표를 수행하였다. 이론 강의는 [13] 교재를 사용하였으며, 테스팅과 관련된 학부 수준의 내용만 발췌하여 강의하였다. 이론 강의는 소프트웨어 테스트 및 정적분석과 관련된 기본 이론들을 제공하며, 학생들 스스로 다양한 테스팅 기법들을 분류하고 취사 선택할 수 있는 능력을 갖추는 것을 목표로 하였다. 다음의 표 1 은 교재에서 수업한 강의 내용들이다.

표 1 이론강의 내용 (요약)

구분	강의 내용
기본이론	1. Software Test and Analysis in a Nutshell 2. A Framework for Test and Analysis 3. Basic Principles 4. Test & Analysis Activities Within a Software process
기본 테크닉	5. Finite Models 6. Dependence and Data Flow Models
기법 및 방법론	9. Test Case Selection and Adequacy 10. Functional Testing 11. Combinatorial Testing 12. Structural Testing 13. Data Flow Testing 14. Model based Testing 15. Testing Object-Oriented Software 16. Fault based Testing 17. Test Execution

2.2 실습 수업

팀 프로젝트로는, 근래에 테스팅 업계에서 화두가 되고 있는 CTIP(Continuous Test & Integration Platform) [11] 환경을 다양한 오픈 프로그램을 이용하여 구축하고, 실제 응용프로그램 [14]의 JAVA 소스 코드를 대상으로, 기능 기반의 단위 테스트(Functional Unit Testing)를 수행하였다. 아래의 그림 1 은 CTIP 을 소개한 그림이다. CTIP 을 구축하기 위해서는 다양한 기법과 도구들이 서로 원활하게 연동되도록 해야 한다. CI(Continuous Integration), 자동 Build, IDE(Integrated Development Environment), Unit Test 도구, 요구사항관리 도구, SCM(Software Configuration Management), Coverage 측정 도구 및 Compiler (SDK)들이 유기적으로 연동되어야 한다.

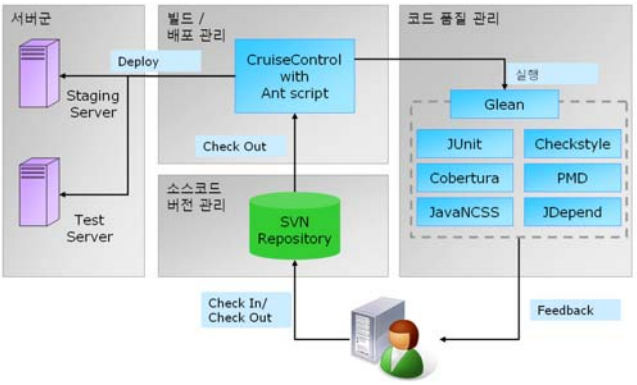

그림 1 CTIP 의 전체 구조 [11]

각 팀은 오픈 프로그램을 사용하여 CTIP 을 구축한 후, JAVA 로 구현된 응용 프로그램인 NuSRS [14]을 테스트하는 실습을 수행하였다. 아래의 그림 2 는 NuSRS 도구 화면이다. NuSRS 는 정형 요구사항 명세 언어인 NuSCR 의 명세 및 분석, 검증을 지원하기 위하여 개발된 도구이다. 실습에서는 2.0 버전을 사용하였으며, 현재 3.0 버전이 개발 중이다.

NuSRS 는 작성된 정형명세의 정확성 (Correctness)과 완전성 (Completeness)를 검사하는 자체 체킹 (Self-Checking) 기능, `Quick Check'을 내제하고 있다. 이 기능은 다음의 8 가지 항목의 검사를 수행한다.

① SDT Condition, Action 의 Undefined Variable
② SDT Condition, Action 에서 연산자 사용 잘못
③ SDT 에서 한 개의 Condition 에 둘 이상의 Action 할당
④ FSM, TTS Transition 의 Undefined Variable
⑤ FSM, TTS Transition 의 연산자 사용 잘못
⑥ FOD, FSM, TTS 에서 Transition 이 없는 노드
⑦ FSM, TTS 에서 Initial State 로부터 Unrechable 노드
⑧ FOD 에서 Output 변수와 연결된 노드 동일 명칭 체크

각 팀은 각자의 CTIP 환경에서, NuSRS 2.0 의 Quick Check 모듈이 명세서와 사용자의 요구에 맞게 구현되어 있는지 단위시험(Unit Test)을 통해 확인하였다. 각 팀은 기능시험 (Functional Testing)을 수행하였으며, Brute Force Testing 과 Combinational Selection Testing 등의 기능시험 기법을 직접 적용하여 테스트 케이스를 개발하였다. 개발된 테스트 케이스의 품질을 Coverage 개념을 사용하여 측정하며, 자동화 도구를 사용하여 시험을 실행 (Test Execution) 하였다. 또한, 시험 결과를 자동으로 정리하여, Fail 된 항목과 관련된 요구사항 명세를 보고서 형태로 정리하여 보여주고, 문제를 수정한 경우 자동

으로 rebuild 및 retest 한 결과도 함께 보여 주었다.

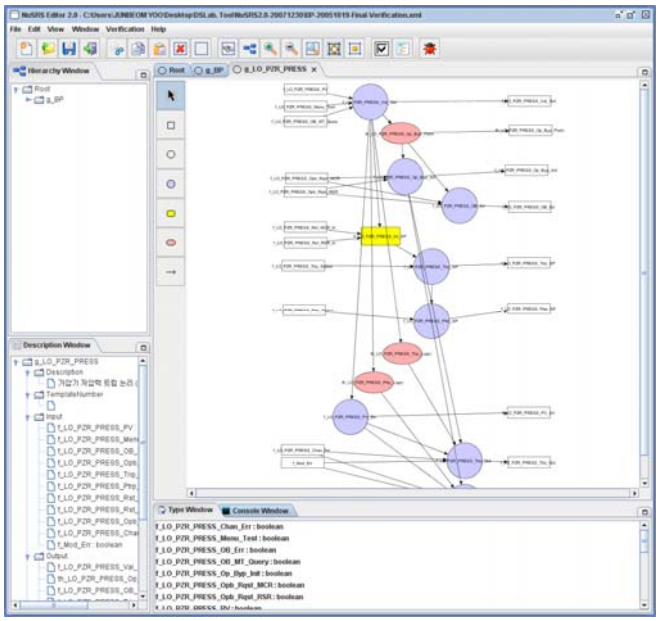

그림 2 NuSRS 도구 화면

3. 수업 결과 및 분석

모두 5 팀이 개별적인 팀 프로젝트를 수행하였다. 각 팀은 상이한 CTIP 환경을 구축하였으며, 개발된 테스트 케이스 및 시험 결과도 서로 상이한 결과를 보였다. 각 팀은 JAVA 프로그램 테스트를 위해 기본적으로 제안된 Eclipse, JUnit 및 JAVA SDK 외에는 서로 다른 도구와 기법들로 구성된 CTIP 환경을 구축하였다. 다음의 표 2 는 각 팀에서 그측한 CTIP 환경을 비교하여 보여 준다. 각 팀은 최소 7 종류 도구와 기법을 사용하여 CTIP 환경을 구축하였으며, Eclipse TPTP 와 같은 도구를 추가로 사용한 팀도 있다.

각 팀에서 사용한 도구들은 JAVA 개발환경인 Eclipse 에서 plug-in 형태로 제공되는 것들이 많다. 특히, 자동 빌드 도구인 Ant 를 Eclipse 와 연동하기 위해서는 복잡도가 상당히 높은 script 를 작성해야 하며, 이 부분에서 팀 별로 많은 시간과 노력이 필요했다. 추가로 사용한 도구 중 Eclipse TPTP 는 개발 중인 프로그램의 성능을 테스트하는 도구로서, 각 JAVA 메소드의 실행 시간과 CPU 점유율 등을 실시간으로 확인할 수 있는 도구이다. 또한, FitNesse 는 사용자의 인수 테스트를 위한 도구이다. 도구의 사용과 연동에 대한 자세한 내용은 과목 홈페이지 [12]에서 확인할 수 있다.

아래의 그림 3 은 팀 1 학생들이 Hudson CI 도구를 사용하여 단위시험을 수행한 결과를 정리한 결과를 보여주고 있다. 그림 4 는 개발한 테스트 케이스가 요구사항 명세의 내용을 얼마나 반영하는 가를 측정하는 도구인 JFeature 를 사용한 화면이며, 그림 5 는 팀 5 에서 Pairwise Testing 을 수행하기 위하여, Category-partition testing 을 통해 2,124 개의 테스트 케이스를 생성한 결과를 보여주고 있다. 팀 5 학생들은 Pairwise Testing 을 적용하여 2,124 개를 37 개로 축소하였다. 그림 6 은 소프트웨어 CM (Configuration Control)을 위하여 CM 소프트웨어를 PC 에 직접 설치하지 않고, 웹 사이트에서 제공하는 CM 서비스를 이용한 화면이다. 이러한 서비스를 이용하기 위해서는 테스트 대상인 JAVA 프로그램의 소스를 웹에 공개해야 한다.

표 2 팀 별 CTIP 구축환경 비교

	T1	T2	T3	T4	T5
CI	Hudson [15]	-	Trac [16]	-	Hudson
Automatic Build	Ant	Ant	Ant	Ant	Ant
IDE (Integrated Development Env.)	Eclipse	Eclipse	Eclipse	Eclipse	Eclipse
Unit Test	JUnit	JUnit	JUnit	JUnit	JUnit
Requirements Management	JFeature	JFeature	JFeature	JFeature	JFeature
CM (Configuration Management)	Subclipse	Subversion	Google Code / Subclipse	CVS	Subversion
Coverage	Cobertura [17]	Eclemma [18]	Clover [19]	CodeCover [20]	Cobertura
Compiler	Java SDK	Java SDK	Java SDK	Java SDK	Java SDK
Etc.	Eclipse TPTP	-	Eclipse TPTP / FitNesse	-	Randoop

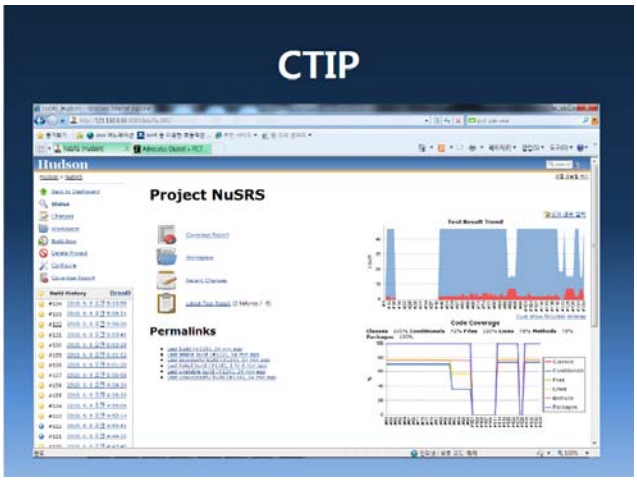

그림 3 Hudson CI 적용 결과 (T1)

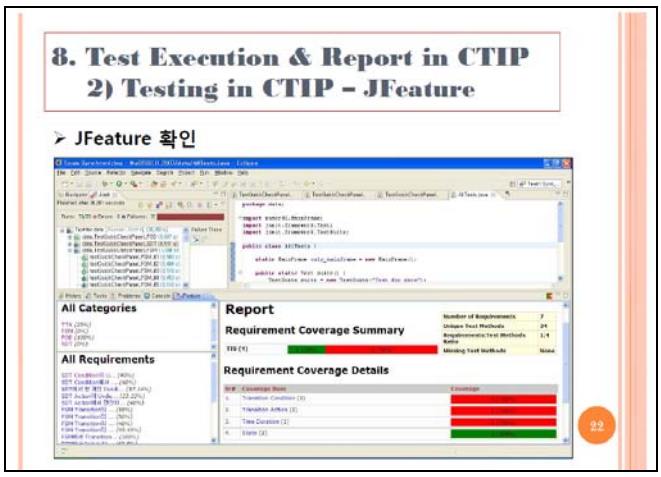

그림 4 JFeature 적용 화면 (T4)

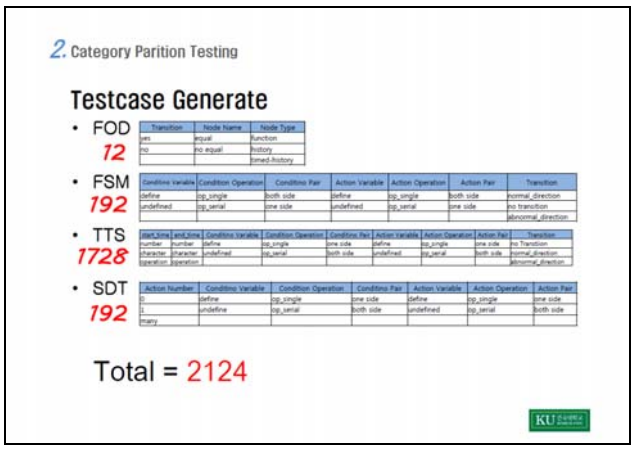

그림 5 Category-Partition Testing 적용 결과 (T5)

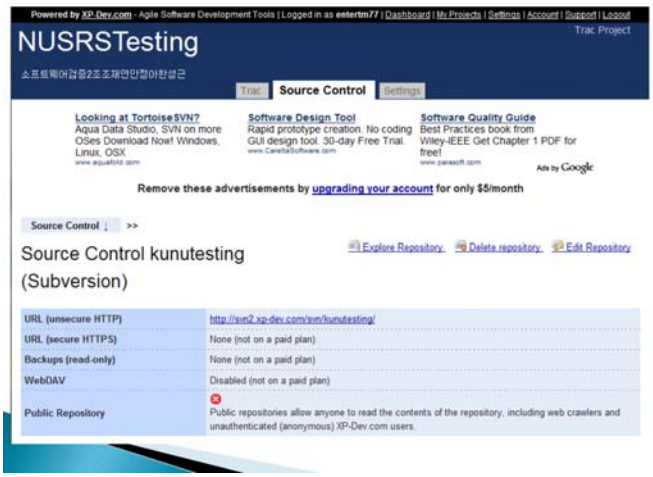

그림 6 Source Control (CM) 적용 화면 (T2)

이와 같이 팀 별로 구축된 CTIP 환경을 이용하여, 각 팀들은 다음의 세 가지 기능시험 (Functional Testing) 기법 중 2 가지 이상을 사용하여 단위시험을 2 회로 나누어 수행하였다. 특히, Pairwise Testing 은 자동화 도구를 사용하여 수행하도록 권장하였다.

(1) Brute Force Testing
(2) Category-Partition Testing
(3) Pairwise Testing

각 팀은 기능 요구사항 명세에서 명시된 8 가지 기능들을 효과적으로 단위시험하기 위하여, 수업에서 다룬 테스트 이론에 따라 먼저 'independently testable feature'를 추출하였는데, 모든 팀이 서로 다른 feature 를 추출하였으며, 이에 따라 모두 다른 방향의 단위 시험을 수행하였다. 이는 각 팀 별로 중요하게 생각하는 시험항목이 서로 상이할 수 있음을 보여주는 것으로서, 이론 수업시간에 언급한 "테스트 케이스를 개발하는 사람의 주관과 판단에 따라 기능 테스트를 통해 발견할 수 있는 오류의 항목이 변할 수 있다."는 내용에 대한 실증으로 학생들에게 이해되었다. 다음의 표 3 은 팀 별 단위시험 수행 내용과 특징을 정리한 표이다. T3 는 조사에 집중해서 발표를 기간 내에 수행하지 못하였다.

표 3 팀별 단위시험 수행 결과 (요약)

	1st round testing	2nd round testing
T1	Category-Partition Testing	Pairwise Testing
	44 test cases (2 fails)	15 test cases (7 fails)
	Spec.에 충실한 test 수행	PICT [21] 사용
T2	Brute Force Testing	Pairwise Testing
	18 test cases (2 fails)	37 test cases (7 Fails)
	-	TVG [22] 사용
T4	Category-Partition Testing	Pairwise Testing
	184 test cases	73 test cases (3 fails)
	중복되는 오류가 모두 검출되는 가에 집중함.	수동으로 직접 수행
T5	Brute Force Testing	Pairwise Testing
	47 test cases (7 fails)	37 test cases (12 fails)
	-	AllPairs [23] 사용

위의 표 3 에서 정리된 바와 같이, 1st round testing 에서 각 팀은 서로 다른 내용에 초점을 맞춘 단위시험을 수행하였다. 수행한 단위시험 케이스의 개수도 18 개에서 184 개까지 다양하며, Category-Partition Testing 또는 Brute Force Testing 을 이용하였다. 각 팀 별로 발견한 오류의 개수와 종류도 모두 상이하였으며, 이 내용은 현재 개발 중이 NuSRS 3.0 에 현재 적용 중이다. 2nd round testing 에서는 자동화 지원 도구를 이용한 Pairwise testing 을 수행하였다. 대부분 도구를 사용한 반면, T4 는 도구를 사용하면, 어렵게 수동으로 생성한 test cases 가 아까워서, 자동화 도구를 적용하지 않고, 나름의 이론을 세워 직접 Pairwise Testing 을 수행하였다.

5 회에 걸친 팀 별 발표와 강의평가를 통해 수집한 수강생들의 의견 및 수강생들의 동의를 얻은 내용들을 정리하면 다음과 같다.

(1) '소프트웨어 공학 개론' 수업을 통해 테스팅의 중요성은 알고 있었지만, 직접 실습해 봄으로써 그 중요성을 체험할 수 있었으며, 테스팅이 이론과 적용 측면에서 얼마나 어려운 학문 분야인가를 깨달았다.
(2) 테스팅을 위한 준비과정부터 테스트 실행(Test Execution)을 포함한 CTIP 전 과정을 수업함으로써, 취업 후 업무에서 실제 적용할 수 있는 경험을 축적했다.
(3) 테스팅은 자동화 도구가 중심이 되는 구조 시험 (Structural Testing) 뿐만 아니라, 전문가의 능력이 중시되는 기능시험 (Functional Testing) 도 중요하고 널리 사용된다는 사실을 이해했다.
(4) 테스트 전문가의 경험과 능력이 테스트 결과에 결정적인 영향을 끼침을 체험했으며, 졸업 후 일반적인 소프트웨어 엔지니어 보다는 테스팅과 같은 특화된 분야에 전문성을 가지는 전문가가 되어야겠다는 생각을 했다.

Dependability 를 보장하기 위한 기법 중의 하나인 테스팅을 학부 학생을 대상으로 강의한 경험을 정리하면 다음과 같다.

(1) 테스팅은 학부 4 학년 학생이 수강하기에 충분한 기법이다.
(2) 테스팅 이론과 함께 적절히 높은 수준의 팀 프로젝트가 필요하다.
(3) 기능시험과 구조시험을 포괄하는 팀 프로젝트의 개발이 필요하며, JAVA 와 C 언어를 모두 포함하면 좋겠다.
(4) 구조시험에서는 산업계의 여러 자동화 도구를 사용하는 실습을 수행하면 효과적일 것으로 판단된다.

4. 결론

최근 Dependability 의 중요성에 대한 사회적 인식이 증가함에 따라, 이를 대학생을 대상으로 교육해야 할 필요성이 대두되었다. 일반적으로 소프트웨어의 dependability 를 보장하기 위한 기법들은, 소프트웨어 공학의 여러 연구 분야들 중에서도 심화된 연구들로서, 대학원을 중심으로 연구되어 왔다. 본 논문에서는 건국대학교 컴퓨터공학부에서 4 학년 학생을 대상으로 하는 '소프트웨어 검증' 수업에서 dependability 기법 중의 하나인 소프트웨어 테스팅을 수업한 내용과 결과 및 경험을 소개하였다.

우려했던 바와는 달리, 학부 4 학년 학생들은 테스팅의 제반 이론과 함께 자율적인 팀 프로젝트를 효과적으로 수행할 수 있는 능력을 갖추고 있음을 확인할 수 있었으며, 팀을 중심으로 CTIP 의 구축 및 기능 기반의 단위시험 수행 등의 어려운 과제를 자발적으로 수행할 수 있는 능력이 있음을 알 수 있었다. 또한, 수업의 프로젝트 수행 내용을 다양하게 보강할 경우, 졸업 후, 소프트웨어 테스팅 전문가로서 활약할 수 있는 가능성을 볼 수 있었다. 수업의 자세한 내용과 수강생들의 발표 자료는 [12]에서 확인할 수 있다.

참고문헌

[1] N.G. Leveson, SAFEWARE, System safety and Computers, Addison Wesley, 1995.
[2] Ian Sommerville, SOFTWARE ENGINEERING (8th), 0000.
[3] B.Bérard, et. al., System and Software Verification", Springer, 2001.
[4] Lars Harms-Ringdahl, Safety Analysis, CRC Press, 2001.
[5] D.A. Peled, SOFTWARE RELIABILITY METHODS, Springer, 2001.
[6] Aditya P. Mathur, Foundations of Software Testing, Pearson Education, 2008.
[7] F. Nielson, H.R. Nielson, C. Hankin, Principles of Program Analysis, Springer, 2005.
[8] Junbeom Yoo, Taihyo Kim, Sungdeok Cha, Jang-Su Lee, Han Seong Son, "A Formal Software Requirements Specification Method for Digital Nuclear Plants Protection Systems," Journal of Systems and Software, Vol.74, No.1, pp73-83, 2005.
[9] K. L. McMillan. Symbolic Model Checking. Kluwer Academic Publishers, 1993.
[10] UPPAAL, http://www.uppaal.com/
[11] CTIP, http://www.sereform.com/?m=20090302
[12] http://dslab.konkuk.ac.kr/Class/2010/10SV/10SV.htm
[13] Mauro Pezzè and Michal Young, Software Testing and Analysis, WILEY, 0000.

[14] NuSRS, http://dslab.konkuk.ac.kr/Nuclear-Requirement/Nuclear-Requirement.htm
[15] Hudson, https://hudson.dev.java.net/
[16] Trac, http://trac.edgewall.org/
[17] Cobertura, http://cobertura.sourceforge.net/
[18] Eclemma, http://www.eclemma.org/
[19] Clover, http://www.atlassian.com/software/clover/
[20] Codecover, http://codecover.org/
[21] Pairwise Independent Combinatorial Testing (PICT), http://msdn.microsoft.com/en-us/testing/bb980925.aspx
[22] Test Vector Generator, http://sourceforge.net/projects/tvg
[23] AllPairs, http://engineering.meta-comm.com/allpairs.aspx