ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 46, No. 8, pp. 774-786, 2019. 8
https://doi.org/10.5626/JOK.2019.46.8.774

OSDEF: 객체지향 소프트웨어 개발 실습을 위한 통합 프레임워크

(OSDEF: An Integrated Framework for Practicing Object-Oriented Software Development)

정 세 진 †　　유 준 범 ††
(Sejin Jung)　　(Junbeom Yoo)

요 약 소프트웨어는 개발 프로세스에 따라 개발되는 것이 일반적이며 이러한 개발 프로세스는 소프트웨어 공학 교육에서 중심적인 역할을 한다. 객체지향 소프트웨어 개발 교육에서는 객체지향적 특성 확인을 위해 여러 예제들을 통해 교육과 실습이 진행된다. 하지만 프로그램의 규모에 따라 객체의 확인은 적합하지만 구현에 많은 노력이 필요하거나 소프트웨어 디자인과 구현의 관계를 파악하기에 어려운 점들이 나타난다. 본 논문에서는 이러한 점을 지원하고 객체지향 소프트웨어 개발 교육의 효과적인 실습을 위해 OSDEF 프레임워크를 제안한다. 이는 기존에 제안한 객체지향 방법론 기반의 소프트웨어공학 교육용 프로세스 OOPT를 바탕으로 하여 산출물 작성 및 관리와 추적성 분석 도구를 포함한다. 또한 임베디드 소프트웨어를 중심으로 층 구조를 효과적으로 실습하고, 실행해 볼 수 있는 환경을 제공해 개발 프로세스에 따라 디자인 및 실행에 대한 실습을 효과적으로 진행할 수 있을 것으로 기대한다.

키워드: OOPT, 소프트웨어공학 교육, 객체지향 개발방법론, 층 구조

Abstract Software development starts with a specific software development process (SDP) which contains the start and end of the development, SDP plays an important role in the software engineering education. Object-oriented software development education uses several examples that contain object-oriented characteristics into education and practices. However, there is an immense burden on the implantation phases as per the scale of the program, thus creating difficulties in the identification of the connected relations between software design and implementation. In the present work, we propose the OSDEF (Object-oriented based Software Development Education Framework) framework for proceeding an efficient software engineering education based on the OOPT (Object-Oriented Process with Traceability). The framework contains artifact management tools which can directly write development artifacts inside the tool, traceability analysis tool, and emulating environment for embedded software, which can practice a layered architecture in an efficient manner.

Keywords: software engineering education, object-oriented software development, OOPT, layered architecture

· 이 논문은 2018학년도 건국대학교의 연구년교원 지원에 의하여 연구되었음

† 학생회원 : 건국대학교 컴퓨터공학과
jsjj0728@konkuk.ac.kr
†† 정 회 원 : 건국대학교 컴퓨터공학과 교수(Konkuk Univ.)
jbyoo@konkuk.ac.kr
(Corresponding author임)

논문접수 : 2019년 3월 4일
(Received 4 March 2019
논문수정 : 2019년 5월 23일
(Revised 23 May 2019)
심사완료 : 2019년 5월 23일
(Accepted 23 May 2019)

정보과학회논문지 제46권 제8호(2019. 8)

1. 서 론

소프트웨어 개발 프로세스(Software Development Process)는 소프트웨어공학의 시작점이 되는 근본 기술로써 요구되는 기능과 복잡성이 증가하는 소프트웨어를 성공적으로 개발하기 위해서는 정해진 순서에 따라 협업하는 것이 필수적이라는 관찰을 통해 제안되었다. 개발 프로세스에는 폭포수 모델(Waterfall Model)[1], 나선형 모델(Spiral Model), 점진적 모델(Incremental Model) 등 다양한 모델[2] 들이 있다. 현재는 소프트웨어 개발 프로세스 모델을 기반으로 소프트웨어 개발에 필요한 사항들, 도구, 표기법 등을 소프트웨어 개발방법론이라 통칭하고 있으며, 대표적으로는 구조적 방법론(SASD: Structured Analysis and Structured Design)[3]과 객체지향 방법론 (Object-Oriented Analysis and Design: OOAD)[4]으로 구분할 수 있다.

객체지향 방법론은 객체(Object class)를 기반으로 각 객체사이의 통신을 통해 시스템의 행위/동작을 구현하는 방법론으로서 객체지향 언어에 적합한 방법론이다. 다양한 대상에 따라 여러 방법론이 제안되었으며 현재 가장 널리 사용되는 방법론으로는 래셔널 통합 프로세스(Rational Unified Process: RUP)[5]가 있다. OOAD 기반의 소프트웨어 개발 교육 진행 시 객체지향적 특성이 나타나는 예제들을 주로 사용하여 교육과 실습이 진행되는 것이 일반적이다[6]. 하지만 대상으로 하는 시스템의 규모에 따라서 객체지향 적인 특성을 잘 드러낼 수 있지만 구현에 많은 노력이 필요하거나, 실습의 결과물로 개발된 프로그램과 소프트웨어 디자인을 통합해 확인하는데 어려운 경우가 많이 존재한다.

본 논문에서는 이를 위해 기존에 제안한 객체지향 소프트웨어공학 교육용 방법론인 OOPT (Object-Oriented Process with Traceability)[7]를 바탕으로 효과적으로 실습을 수행할 수 있는 OSDEF (Object-oriented Software Development Education Framework) 프레임워크를 제안한다. OSDEF는 구현의 노력을 줄이고 소프트웨어 디자인과 UI간의 연결되는 아키텍쳐를 확인하는 구조를 제공한다. 최종적으로 실제 실행을 위한 에뮬레이션 환경을 제공함으로써 효과적으로 소프트웨어 architecture에 대한 실습을 수행할 수 있다.

또한 OSDEF에서는 OOPT에서 각 단계별로 개발 산출물을 외부의 도구를 활용해 개별적으로 작성, 관리하면서 발생하는 문제점을 보완하기 위해 이전에 연구한 OOPT에서의 추적성 분석 도구[8]를 확장하여 UML (Unified Modelling language) 다이어그램, 각종 산출문서를 직접 작성하고 추적성 분석을 수행할 수 있는 도구도 포함하고 있다. 이를 통해 산출물을 하나의 도구에서 직접 작성하고 관리하며, 자동으로 추적성을 분석할 수 있어 소프트웨어 프로세스에 따른 분석, 디자인의 전체적인 파악이 용이하다[8].

OSDEF는 [9]에 소개된 임베디드 소프트웨어를 위한 layered architecture 구조의 소프트웨어 시스템을 효과적으로 실습하고 실행해 볼 수 있도록 하였으며, 그 외 다른 information system, network-based multi-agency 시스템 등으로의 확장이 가능하다. 이를 바탕으로 교육에서 코딩 소요를 줄이고, 소프트웨어 architecture 설계에 대한 교육 내용이 더욱 효과적으로 진행될 것으로 기대 한다. 본 논문의 구성은 다음과 같다. 2장은 본 논문에서 대상으로 하는 OOPT에 대해 소개한다. 3장은 논문에서 제안하는 프레임워크에 대한 자세한 설명을, 4장은 프레임워크를 활용한 사례 연구에 대해 소개한다. 마지막으로 5장은 논문을 마무리한다.

2. 배경 지식

2.1 OOPT (Object-Oriented Process with Traceability) 개요

OOPT는 대학 소프트웨어공학 수업에 활용 가능하도록 래셔널 통합 프로세스를 수정하여 제안된 객체지향 방법론이다. 특히 기본적인 소프트웨어 개발 외에 다양한 소프트웨어공학 이론, 기술을 습득할 수 있도록 추적성 분석, 테스팅 등 필요한 내용들이 포함되어 있다. OOPT는 개발 도중 여러 변경 사항 등에 대처하고 피드백을 위해 반복적(Iterative)이고 점진적(Incremental)으로 소프트웨어를 개발할 수 있도록 구성되어 있으며 크게 'Plan and Elaboration', 'Build', 'Deployment' 3 stage로 구성된다.

Stage 1000 Plan and elaboration은 요구사항 정의 및 계획과 같은 프로젝트 개요를 정의하는 단계로 use case 작성, 테스트 계획 수립, 프로젝트 계획, 요구사항 정의 등의 활동을 수행한다. Stage 2000 Build 단계에서는 소프트웨어에 대한 실질적인 분석, 디자인 및 구현을 진행하는 단계로 그림 1과 같이 여러 사이클로 진행된다. 이는 반복적이고 점진적인 개발을 위해 구성되는 부분이다. 특히 Stage 2000 단계는 내부적으로 'refine plan', 'synchronize the plan activity', 'analyze', 'design', 'construction', 'testing'의 6 단계의 액티비티들이 그림 1과 같은 사이클에 따라 수행된다.

OOPT를 통해 반복적(iterative)이고 점진적(incremental)한 객체지향 개발에 대해 학습할 수 있으며, 다양한 액티비티들을 통해 소프트웨어를 분석하고 디자인하는 실습을 수행하여 개발프로세스에 따라 소프트웨어를 개발할 수 있다. 특히 use case 다이어그램, sequence 다이어그램, class 다이어그램과 같은 여러 UML 다이어그램

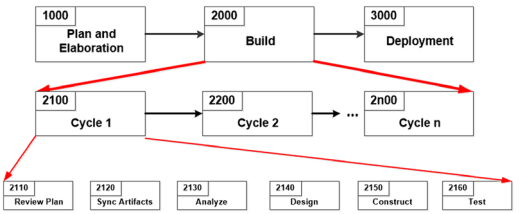


그림 1 OOPT의 stage 2000 cycle 활동
Fig. 1 Stage 2000 activities of the OOPT

표 1 OOPT Stage 2000 에서의 개발 산출물 타입
Table 1 Types of the development artifacts in the OOPT stage 2000

Artifact type	Kind	Note
Doc.	Table	
	Description text	
UML Diagram	Use case diagram	Activity 1006
	System architecture	Activity 1008
	Business concept model	Activity 1007
	Class diagram	Activity 2033, activity 2045
	Sequence diagram	Activity 2035, activity 2044
	State diagram	Activity 2037
	Package	Activity 2043

들을 활용하여 소프트웨어를 분석, 디자인함으로써 현재 가장 널리 사용되는 UML에 대해 학습 및 실습이 가능하다. 각 stage 및 activity에 대한 자세한 설명은 [6]에서 확인할 수 있다.

현재 OOPT에서는 표 1의 산출물들을 워드프로세서 도구와 독립적인 UML 도구를 사용하는 등 프로젝트를 진행하면서 개별적으로 작성 관리되고 있어 각 산출물의 버전 관리 및 추적성 분석 등에 많은 제약사항을 가지고 있다는 단점이 있다. 특히 추적성 분석에 있어 수많은 개발 산출물에서 추적성 분석을 위한 각 목록 작성 및 연결에 많은 노력의 소요가 발생한다.

2.2 관련 연구

소프트웨어공학 교육에서 객체지향 소프트웨어 개발은 학교, 회사 등에서 다양한 방법으로 교육이 진행되고 있다. 객체지향 소프트웨어 개발 교육은 객체지향 개념의 교육에서부터 개발 프로세스 및 UML 디자인 교육까지 다양한 형태로 진행되고 있으며[6,10], 기존의 UP 기반 개발 프로세스를 교육에 적합하도록 수정해 적용하는 사례들 또한 존재한다[11]. 특히 [12]에서는 확인결과 OOAD교육에 프로그래밍을 동시에 진행하는 것이 좋은 생산성을 보임을 확인하였다.

[13]에서도 교육 프로세스에서 디자인과 구현을 모두 진행하도록 하지만, 구현에 있어 디자인 결과물에 맞는 코드 프레임 작성만을 포함하고 있다. 이외에도 소개된 다른 사례들도 마찬가지로 구현의 내용을 포함해 코스를 진행하지만, 구현은 디자인 이후에 따로 진행하는 것으로 구성하고 있다[10,11]. UP를 수정해 적용하는 여러 교육 방법론 에도 구현에 대한 언급은 소프트웨어 프로세스에 맞추어 분석, 디자인대로 구현할 것으로만 언급하고 있다.

[14]에서는 객체지향 프로그래밍 교육을 위해 학생들에게 시뮬레이션 환경을 제공 도구에 대한 소개를 하고 있다. 해당 도구는 객체를 디자인하고 객체들 간의 관계를 정의하면서 시뮬레이션 함으로써 객체지향 프로그래밍에서 객체들의 관계를 학습하는데 좋은 환경을 제공하고 있다. 하지만 소프트웨어공학적 관점에서 소프트웨어 프로세스에 대한 내용은 부족하고, 시뮬레이션은 실제 동작보다는 객체들의 확인에만 중점을 두고 있다.

이처럼 OOAD와 관련된 교육에서는 일반적으로 UML을 활용한 디자인에 구현의 내용까지 같이 진행하고 있다. 하지만 대부분의 교육 진행은 디자인에 더 집중하고 있으며, 구현에 대해서는 작성한 디자인을 코드로 옮겨보는 것에서 그치고 있다[14]. 또한 UML 다이어그램 작성을 외부의 오픈소스 도구를 활용하면서 추적성 분석과 같은 점들이 부족하다.

3. 객체지향(Object-Oriented) 기반 SW 개발 교육 프레임워크

본 논문에서는 OOPT에서의 여러 제약사항 들을 보완하고 구현 및 실행이 포함된 실습을 지원하기 위해 OOPT를 바탕으로 객체지향 소프트웨어 개발 교육 프레임워크 'OODEF'를 제안한다. 제안하는 프레임워크는 개발 산출물의 직접 작성 및 추적성 분석 자동화와, 주요 예제로 많이 사용하는 임베디드 소프트웨어 (ESW: Embedded Software)를 위한 layered architecture 구조의 소프트웨어 시스템[9]에 대해 디자인과 UI의 연결을 통해 execution할 수 있는 에뮬레이션 환경을 구성하여 효과적인 교육 및 실습이 가능하도록 하였다.

그림 2는 OSDEF 프레임워크의 전체적인 개요에 대한 그림으로 총 3 파트로 구성된다고 할 수 있다. 그림 2의 위쪽은 임베디드 소프트웨어의 layered architecture에 대한 그림으로 일반적으로 UI, Business logic, technical service layer로 구성된다. 소프트웨어공학의 소프트웨어 개발 교육에서 주로 실습을 통해 작성하는 부분은 business logic layer이며 이 부분이 소프트웨어 분석, 디자인과 관계된다고 할 수 있다. UI layer와 technical service layer는 주변의 환경을 구성하는 계층으로 business logic layer와 연결되어 있는 주변의 계층이다.

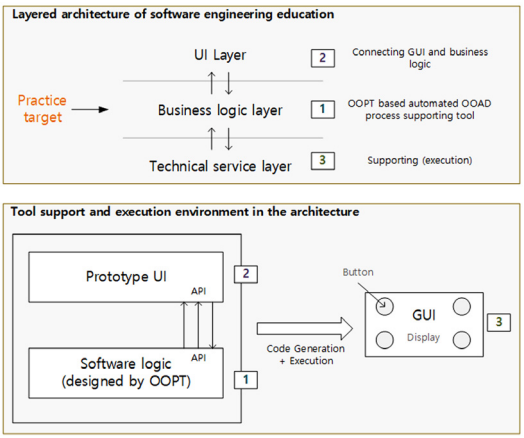


그림 2 OSDEF 프레임워크 개요
Fig. 2 An overview of the OSDEF framework

OSDEF는 이를 위해 software logic을 작성하는 (1) 작성 지원 도구와 UI와 소프트웨어 로직을 연결하는 (2) 연결 지원 환경, 코드를 생성하여 실행 환경을 제공하는 (3) 에뮬레이션 환경을 제공한다.

3.1 OOPT 지원 도구

OOPT는 소프트웨어 개발을 위해 분석 및 디자인과정을 진행하며 그 결과로 표 1과 같이 다양한 타입과 종류의 UML 다이어그램 및 문서들을 작성한다. 이 때 각 문서와 다이어그램의 내용들은 각각 밀접한 연관 관계를 갖고 진행되지만 독립적인 워드프로세서, UML 도구를 이용해 진행되어 각 관계에 대한 분석을 추가로 수행하거나, 관리가 어려운 점이 있다. OSDEF에서는 이를 보완하고자 지원도구를 통해 결과물을 내부적으로 직접 작성하도록 하며, 추적성 분석 또한 자동으로 지원한다.

3.1.1 산출물 작성 관리 도구

우선 OOPT를 구성하는 stage/activity 들 간에는 추적성 분석관계를 포함해 선후 관계가 존재하는 연결을 확인할 수 있다. 예를 들면 activity 1006에서 정의한 business use case를 essential use case, real use case를 작성하는 activity 2031, 2041 단계에서 이용하는 등의 관계로, 다음 표 2는 OOPT에서 존재하는 선후 관계에 대해 정의한 표이다. 표 2에 따라 OOPT 프로세스에 따른 작성 지원 도구 개발을 위해 자동 생성 또는 자동 연결이 필요한 activity 들의 관계에 대해 정의하였다.

해당 표에서 connection 관계는 activity 1 파트에서 작성한 항목을 바탕으로 activity 2 파트에서 작성한 내용과 작성자가 직접 연결하며 작성이 필요한 관계로써 항목 연결을 지원하도록 하였다. Connection 관계는 추적성 분석의 기초 관계로써 각 항목들이 디자인이 진행되며 다음 단계로 연결이 되는 지점을 나타낸다. Generation 관계는 분석, 디자인 진행 시 이전 단계의 산출물이 다음 단계에서 명시적으로 연결되어 상세 내용을 정의할 경우에 대한 관계이다. 본 논문에서 제안하는 프레임워크에서는 activity 내부적 관계를 포함해 해당 관계들이 성립하는 경우에 자동으로 이전 단계의 산출물 정보를 연결하여 산출물 관리가 용이하도록 하였다.

OOPT 지원 도구는 JAVA를 사용하여 개발되었으며 UML 작성을 위해 violet[15] 오픈소스 라이브러리를 도구에 맞게 수정하여 적용하였다. 도구의 주요 작성 과정은 다음과 같다. 그림 3은 지원 도구의 시작 화면으로 OOPT의 각 stage/activity에 따른 navigator를 통해 관리할 수 있다. 이 때 Stage 1000 단계부터 순차적으로 진행할 시 표 2에 따라 연결된 결과물들을 자동으로 추출해 리스트 및 작성 환경을 구성한다.

그림 4는 Stage 1000 단계의 use case 다이어그램 작성 및 요구사항과 use case 간의 연결을 작성하는 화면으로 Stage 1000 단계의 앞부분에서 작성한 요구사항의 항목을 자동으로 생성하여 각 요구사항 별로 use case

표 2 OOPT activity에서 자동 연결, 생성 지원 항목
Table 2 Lists of relation for automated connection, and generation in the OOPT

Activity 1	Activity 2	Need operation
1003 Requirements	1006 business use case	Connection
1006 Business use case	2031 Essential use case	Generation
2031 Essential use case	2041 Real use case	Generation
1003 Requirements	1009 System test case	Connection
2031 Essential use case	2035 System sequence diagram	Connection
2035 System sequence diagram	2036 System operation	Generation
2031 Essential use case	2038 System test case	Connection
2041 Real use case	2044 Interaction diagram	Connection
2035 System sequence diagram	2044 Interaction diagram	Connection
2045 Class diagram	2051 Class & method definition	Generation
2038 System test case	2063 System testing	Generation

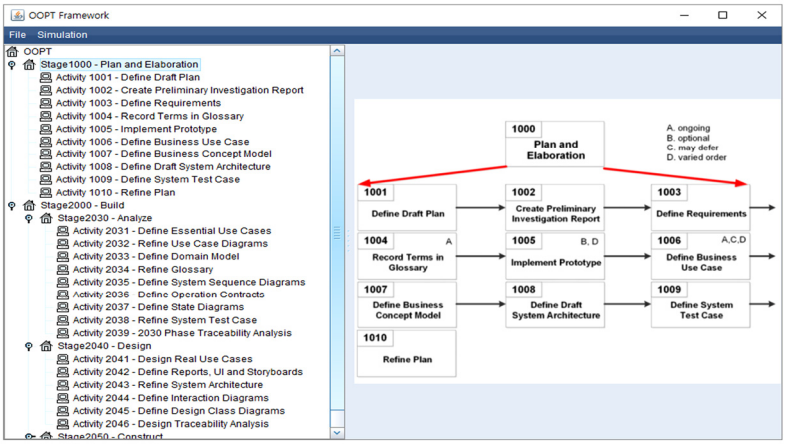
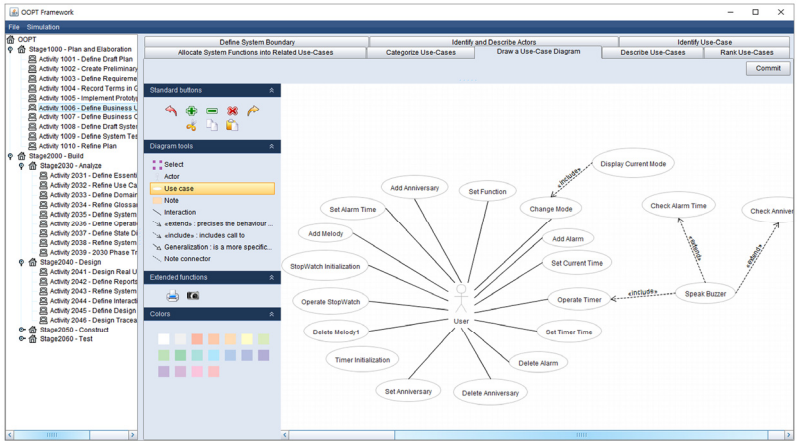
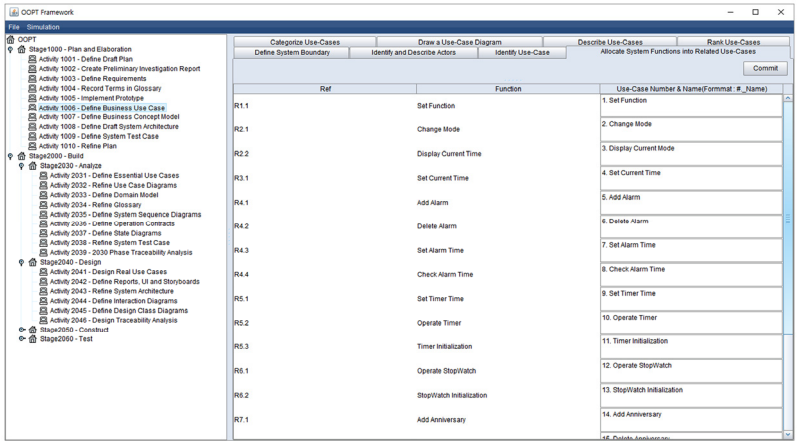


그림 3 도구 시작 전체 화면

Fig. 3 An overview of the tool



(a) Use case 다이어그램 작성 화면



(b) Use case 및 요구사항 연결

그림 4 Use case diagram 및 요구사항 연결 화면

Fig. 4 A screen example of writing use case diagram and requirements

를 할당하는 것을 지원한다. 이를 통해 작성한 각 요구사항을 빠짐없이 use case를 할당할 수 있다.

그림 5는 Stage 2030 단계의 시스템 시퀀스 다이어그램을 작성하는 화면으로 use case의 리스트를 제공해 각 use case 별로 시퀀스 다이어그램을 작성하도록 한다. 시스템 시퀀스 다이어그램은 use case 별로 use case를 달성하기 위해서 사용자가 시스템과 상호작용하는 system operation을 작성하는 과정으로 각 use case 별로 작성되어야 하기 때문에 자동으로 연결을 통해 리스트를 제공하고 있다.

그림 6은 system operation을 작성하는 화면으로 앞의 그림 5의 시퀀스 다이어그램에서 작성한 system operation들의 정의 및 description을 작성하는 화면이다. 이때 앞의 시퀀스 다이어그램에 작성한 항목들을 마찬가지로 자동으로 불러와 항목을 제공한다. 그림 7도 시스템의 내부적인 동작에 대해 작성하는 interaction diagram을 그림 5와 같이 use case 별로 작성하도록 항목을 생성해 준다. 그 외 2031, 3041 단계에서 use case를 상세화 하는 activity, 2051과 같이 method를 작성하는 activity 등도 연결 관계를 고려하여 항목을 생성하고 있다.

그림 8은 클래스 다이어그램을 작성하는 화면으로 그림 8을 통해 작성한 클래스 다이어그램이 2040 디자인의 최종 산출물이 된다. 이처럼 OOPT 지원 도구는 다양한 UML 다이어그램 및 문서 항목 작성을 지원하고 OOPT 각 stage/activity 간의 연결 관계를 고려해 자동으로 항목을 생성하며 작성을 지원함으로써 소프트웨

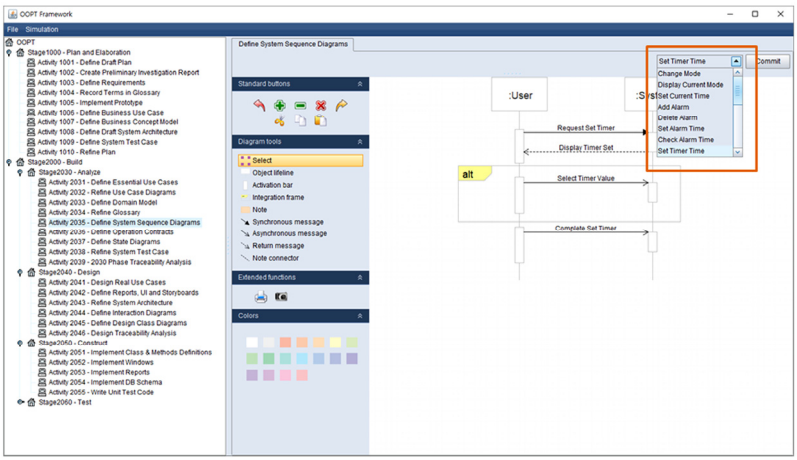


그림 5 시스템 시퀀스 다이어그램 작성 화면

Fig. 5 A screen example of writing a system sequence diagram

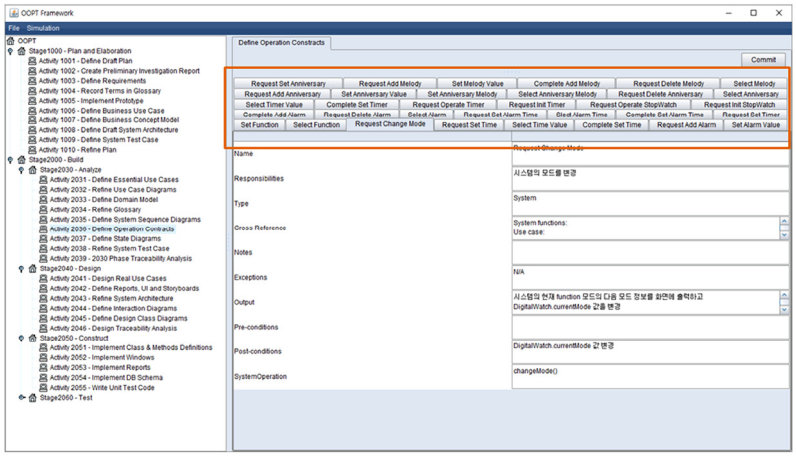


그림 6 System operation 작성 화면

Fig. 6 A screen example of writing a system operation

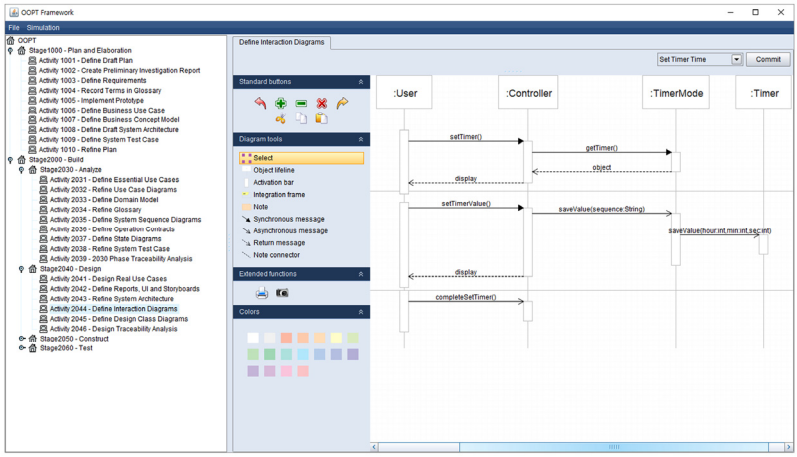


그림 7 시퀀스 다이어그램을 활용한 interaction 다이어그램 작성 화면

Fig. 7 An example of writing an interaction diagram using UML sequence diagram

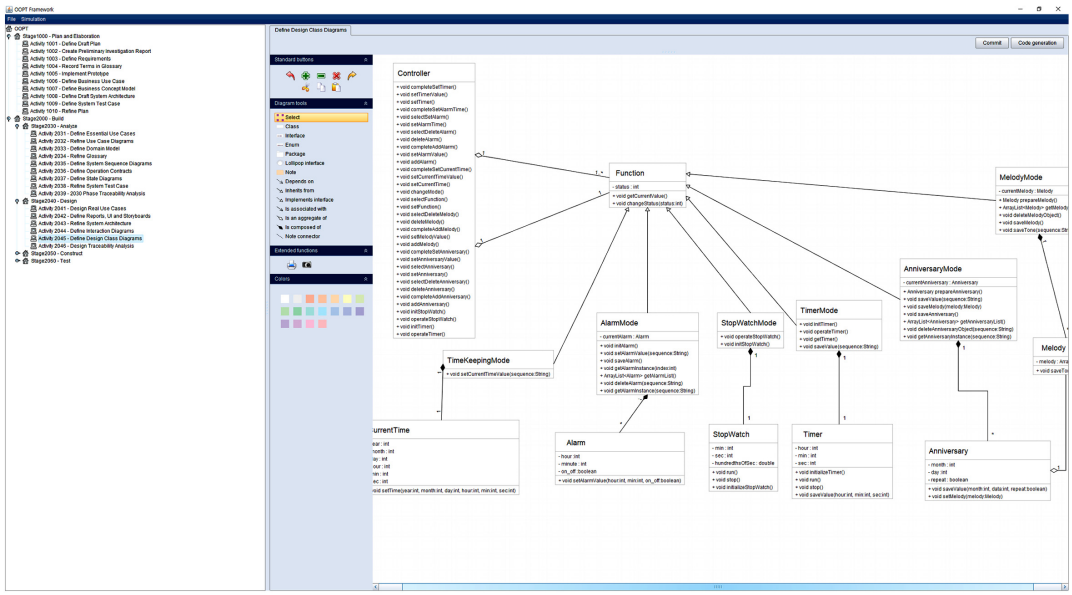


그림 8 클래스 다이어그램 작성 화면

Fig. 8 A screen of writing a class diagram

어 분석, 디자인 실습을 진행하는데 도움이 된다.

3.1.2 OOPT 개발 산출물의 추적성 분석

위 절에서 설명한 것과 같이 OOPT의 각 액티비티의 개발 산출물은 연결 항목들이 존재하며 또한 추적성을 분석할 수 있다. 추적성 분석은 소프트웨어의 요구사항부터 구현까지 그 작성 관계를 분석하는 것으로 중간에 요구사항의 변경이나, 유지 보수의 변경으로 인한 그 인과 관계 및 변경 지점을 찾는데 효과적이다.

OOPT 각 액티비티의 개발 산출물을 작성하고, 그 추적성을 분석하기 위해서는 많은 노력이 필요 하다. 본 논문에서 제안하는 프레임워크에서는 각 산출물의 작성을 지원하며 존재하는 추적성을 자동으로 분석할 수 있도록 하였다. 이를 위해 본 논문에서는 OOPT 프로세스에서 추적성 분석에 사용되는 산출물을 분류하고 그 관계를 정의하였다. 표 3은 OOPT의 각 산출물 중 추적성 분석에 활용되는 주요 산출물에 대한 표로 text, UML, test로 분류하여 정리하였다.

추적성 관계는 기본적으로 'trace to' 관계를 가지며

표 3 추적성 분석에 사용되는 산출물 분류
Table 3 Classification of the development artifacts used for traceability analysis

Artifact kinds of the OOPT	Type of Entity
Functional requirement	<<Textual requirement>>
Use case	<<UML diagram>>
System test case	<<Test artifact>>
Unit test case	<<Test artifact>>
Essential use case specification	<<Textual requirement>>
System sequence diagram	<<UML diagram>>
System operation	<<Textual requirement>>
Real use case specification	<<Textual requirement>>
Interaction diagram	<<UML diagram>>
Class diagram	<<UML diagram>>
Method/class description	<<Textual requirement>>

한쪽의 산출물이 반대의 산출물로 나타나는 추적 관계를 의미한다. 추적성 분석의 관계는 표 2에 나타난 사용 관계와 유사하다. 다음 그림 9는 도구를 통해 도출한 추적성 분석 결과에 대한 화면으로, functional requirement 부터 class/method 및 테스트 케이스까지의 추적 과정을 자동으로 도출하여 추적성 분석에 소요되는 노력을 줄이고 정확한 분석을 할 수 있다.

Textual requirement 타입의 산출물은 각 산출물 종류별 대표적인 text (e.g. 요구사항 이름, use case 이름)를 기준으로 하고, UML 다이어그램도 마찬가지로 이름을 기준으로 하였다. 반면 test artifact의 경우에는 사용자가 연결한 다른 산출물의 텍스트를 이용하여 추적성을 분석하는 모델을 구성하고 분석을 수행하였다. 이를 이용해 요구사항부터 구현까지 소프트웨어의 개발 과정의 산출물 및 그 관계를 한눈에 확인할 수 있다는 장점이 있다.

3.2 ESW의 layered architecture 학습 환경 구성

지금까지 소프트웨어공학 교육의 이론적 부분을 지원하는 OOPT 작성 지원 도구에 대해 설명하였다. 이 절에서는 OSDEF 프레임워크에서 OOPT를 통한 소프트웨어 디자인 결과물을 바탕으로 UI 연결 및 execution을 지원하는 환경 구조에 대해 설명한다. 소프트웨어 architecture는 용도에 따라 여러 종류가 사용되지만, 현재 OSDEF에서는 여러 다양한 architecture 중 layered architecture를 갖는 임베디드 소프트웨어를 대상으로 하며 OOO Dgitial_Watch 개발을 지원한다.

본 논문에서 제안하는 프레임워크를 통해 layered architecture 상에서 business logic layer 부분의 소프트웨어를 직접 분석, 디자인하고, UI layer는 프레임워크를 통해 제공받아 일정 규칙에 따라 연결을 수행하는 것으로 layered architecture 구조에 대해 학습하고, 코드를 생성하여 직접 실행해 볼 수 있다. 따라서 UI 구현에 대한 이슈를 줄이고, 소프트웨어 디자인과 UI 연결을 통해 구현의 결과와 디자인을 통합해서 쉽게 확인해 볼 수 있다. 이는 그림 2의 (2), (3) 파트에 해당한다. 이 때 UI와 로직의 연결은 OOPT '2035 define system sequence diagram' 액티비티를 통해 개발한 외부 사용자가 시스템을 사용하

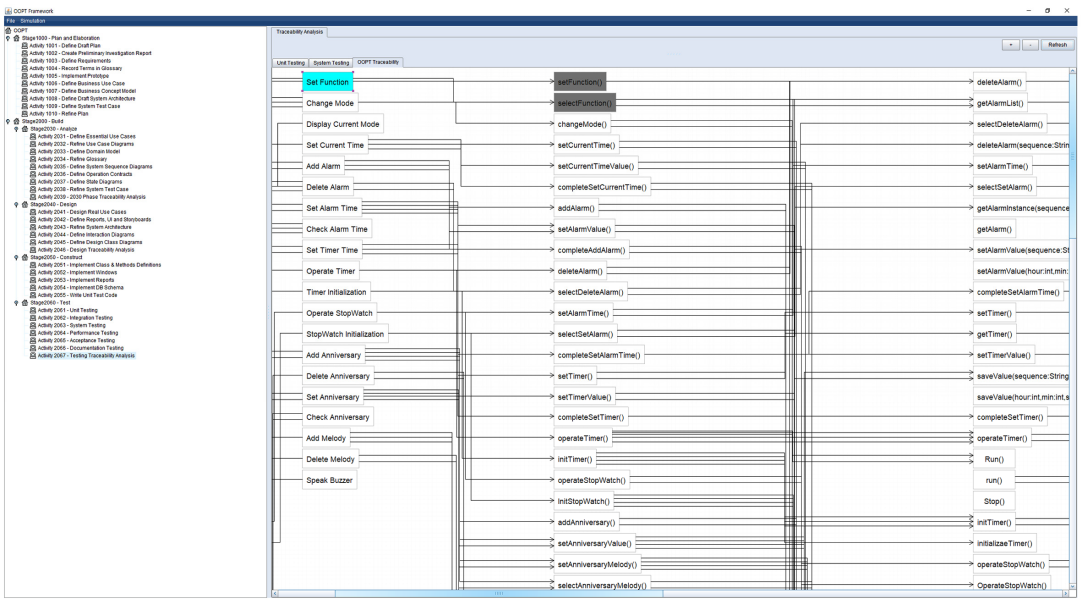


그림 9 도구를 통한 추적성 분석 결과[7]
Fig. 9 The traceability analysis result by the tool

는 system operation을 기준으로 하여 작성한 디자인 결과를 그대로 확인하는데 도움이 된다.

이 때 2036, 2044 activity 들에서 정의하는 system operation의 수준에 따라 UI에서 필요한 정보가 결정된다는 점을 고려해야 한다. 시스템의 수준에 따라 UI에서 하나의 입력(e.g. button click, text input, Etc.)에 하나의 system operation이 할당되기도 하지만, 하나의 입력에 조건에 따라 다른 system operation이 할당될 수도 있다. 일반적으로 후자의 경우는 system UI가 고정되어 있을 경우에 주로 나타나지만 layered architecture에 따라 각 layer의 정의 수준에 따라 달라진다. 본 논문에서는 이를 고려하여 UI와 system operation을 연결하는 중간에 작동 조건(condition) 작성 과정을 도입하였다.

본 논문에서 제안하는 system operation 연결의 순서는 다음과 같다.

1. 정의된 UI의 버튼 선택
2. 연결할 system operation 선정
3. System operation 별 작동 조건 작성
4. 코드 생성(generation)

그림 10은 각 버튼 별로 선택한 system operation에 대해 작동 조건을 작성하는 화면이다. 화면의 중간 위 부분에 조건을 작성하게 되면 해당 조건이 포함된 UI layer의 코드를 자동으로 생성해 각 system operation의 실제 동작 부분만 코딩할 수 있도록 하였다. Condition에 대한 작성 규칙은 class diagram의 attribute을 기준으로 하도록 하며, 논리 연산자, 이름 규칙, 비교 연산자, 값 순으로 한다. Condition 작성 규칙을 정규식으로 표현하면 "^([AND|OR|NOT]*)([0-9a-zA-Z_]*)([==|>|<|>=|<=]+)([0-9]+)*$"이다. 작성 규칙에 따라 작성한 system operation을 실행하는 코드의 생성 화면은 그림 11과 같다. 선택하여 조건을 작성한 각각의 system operation에 따라 조건문을 활용하여 할당된 system operation을 실행할 수 있도록 작성한다. 이 때 사용자는 정해진 코드에 system operation을 표현하는 메소드의 내용만 작성함으로써 UI와 실제 동작을 연결할 수 있게 된다.

자동으로 생성되는 코드는 미리 설정한 UI 실행 코드와 class diagram 상의 logic class, UI 설정을 위해 필요한 API 메소드, 그리고 그림 11에서 설명한 연결 코드로 구성된다. 그림 12는 생성한 코드를 실행한 화면으로, 사용자는 system operation 별로 시스템이 할 동작을 작성하며 구현 단계를 진행하고, 제공되는 메소드들을 활용해 UI의 디스플레이 값을 변경할 수도 있다. 이 과정을 거쳐 사용자는 system operation의 몸체만 작성 후 실행해 볼 수 있어 구현의 소요를 줄이고, system operation의 개념을 학습할 수 있다.

3.3 프레임워크 확장 방향성 연구

이처럼 본 논문에서는 객체지향 소프트웨어 개발 교육을 위해 개발 프로세스 지원 도구와 layered architecture로 구성되는 임베디드 소프트웨어를 대상으로 한 layer 연결 및 에뮬레이션 환경으로 구성된 프레임워크를 제안하였다. 표 4는 OOPT와 비교해 OSDEF의 차이점 및 개선점에 대한 표로 OSDEF에서는 대부분의 개발 산출물 작성 및 관리를 프레임워크에서 제공하는 도구를 통해 지원함으로써 개별적인 외부의 워드 프로세서 및 UML

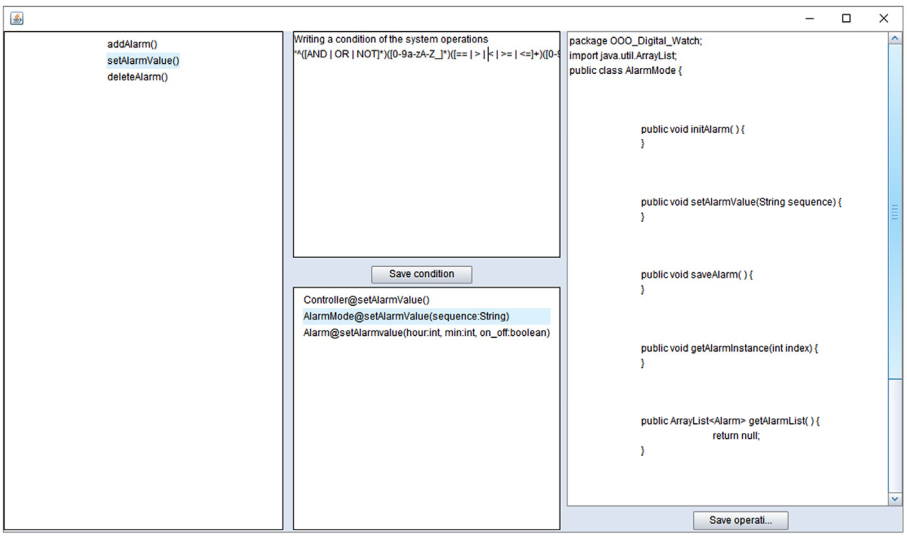


그림 10 System operation 별 UI 작동 조건 작성 화면

Fig. 10 A screen of writing an operation condition for each system operation

그림 11 코드 생성 결과

Fig. 11 The result of the code generation

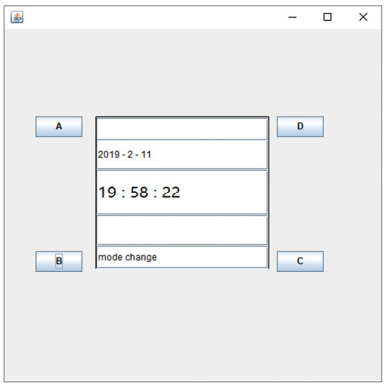


그림 12 코드 생성 후 실행 화면

Fig. 12 An execution result with generated code

도구를 활용해 작성하던 OOPT와 비교해 산출물의 관리 및 작성에 용이함을 가지고 있다. 또한 추적성 분석에 있어서도 일부 자동 분석과 코드 생성 등의 작업을 프레임워크에서 수행할 수 있다는 차이점이 있다.

하지만 본 논문에서 대상으로 하는 architecture는 UI와 business logic 이 구분되는 layered 구조만을 대상으로 하고 있다. 하지만 소프트웨어에는 다양한 architecture가 있으며 이 부분에 대해 확장이 가능하다. 예를 들면 정보를 가지고 처리하는 information system이나 network를 통해 여러 통신을 처리하는 네트워크 시스템 등 다양한 architecture를 대상으로 확장을 할 수 있다.

하지만 이 부분에 대해서는 하드웨어가 고정된 임베디드 소프트웨어와는 달리 UI에 대해 다양한 변화와 반영이 필요하다. 따라서 prototype UI를 상정해 두는 것만으로는 부족하기 때문에 2042 activity의 UI storyboard를 작성하는 부분에서 추가적인 고려와 디자인 요소가 필요하다. 또한 교육을 위해 다양한 추가 환경을 고려할 수 있다. 특히 오픈소스 소프트웨어를 활용한 개발을 진행할 때 오픈소스 소프트웨어가 갖는 위치를 포함해 컴포넌트 기반의 디자인 작성 등에 대한 점도 고려하여 프레임워크를 구성해 볼 수 있다. 본 논문에서 제안하는 코드 생성 부분에서 오픈소스를 포함해 코드를 구성하는 방안을 고려해볼 수도 있다고 생각한다.

4. 사례 연구

본 논문에서는 'OOO digital watch'를 예제로 하여 제안하는 프레임워크를 활용해 OOPT 지원 도구 및 system operation 연결, execution 환경의 진행 및 교육의 활용 가능성을 확인하였다. Digital watch는 작은 시계에 들어가는 임베디드 소프트웨어이며, UI는 4개의 버튼과 display로 고정되어 구성된다. 기본 기능으로는 현재 시각 확인, 알람, 스톱워치가 있으며 'OOO' 항목을 위해 자유롭게 추가 기능을 고려하여 디자인하도록 진행하였다. 'OOO'은 기존의 실습 수업과 달리 학생들이 직접 요구사항을 정의하며 자유로운 추가 분석을 할 수 있도록 하는 부분으로 본 논문에서는 '기념일을 알려주는' 으로 정의하였다. 본 논문에서는 해당 예제를 대상으로 처음 시작인 요구사항 분석부터 코드 생성까지 진행하였다.

총 20개의 기능 요구사항과 각 요구사항에 할당되는 use case를 분석하였고 그 결과는 그림 4와 같다. 각각의 use case에 대해 brief 형태로 시나리오의 상세 사항을 지원 도구의 작성 파트에 작성하여 진행하였다. 특히 본 논문에서 제안하는 지원 도구에서는 기능 요구사항과

표 4 OSDEF 와 OOPT의 개선점 비교

Table 4 The differences and improvements of the OSDEF compared to OOPT

	OOPT	OSDEF
Description & UML writing (use case, Etc.)	Using external tool	Framework support
Development artifact management	Individual artifact management	Manage once with internal support
Traceability analysis	manually	Automatically
Code generation & architecture layer connection	Depends on external UML tools & manually	Supports semi-automatically practice in framework
GUI emulation support	X	O

이에 맞는 use case를 연결하는 작성을 지원해 모든 기능 요구사항을 빠뜨리지 않고 디자인하는데 도움을 줄 수 있다. 그 외 순차적으로 OOPT의 activity에 따라 use case의 상세화(essential/real use case), 시스템 시퀀스 다이어그램 작성, system operation 도출, 및 interaction diagram 작성을 진행하였으며 그 결과의 일부를 각각 그림 5~그림 8에서 확인할 수 있다. OOPT 지원 도구를 통해 이처럼 요구사항 작성부터 클래스 다이어그램, test case 작성까지 작성하여 손쉽게 관리할 수 있다.

특히 각 그림에 표시한 바와 같이 이전의 activity에서 작성한 항목을 뒤의 activity에서 자동으로 제공하는 틀에 넣는 부분이 많은 도움이 된다고 할 수 있다. 기존의 OSDEF 프레임워크를 통하지 않고 OOPT 방법론을 적용한 실습에서는 파워포인트와 오픈소스 UML 도구를 따로 사용하기 때문에 버전 관리 및 보고서 작성 관리에 별개로 작성된 항목들을 하나로 만드는 등 불편한 점이 개선되었다. 기존의 방법론을 사용한 프로젝트의 예제들은 [7]에서 확인할 수 있다.

결과적으로 총 38개의 system operation이 도출되었으며 이는 하나의 use case의 시나리오를 달성하기 위해 외부의 사용자가 시스템과 여러 번 상호작용이 필요함을 의미한다. 표 5는 각 use case의 system operation 별 버튼 할당에 대한 예시로 아래 표와 같은 형태로 각 system operation 별로 버튼을 할당하고 조건을 작성하였다. 그림 13은 버튼 B에 할당하기 위해 선택된 system operation 및 조건 작성 화면이고, 그림 14는 System operation 할당 후 조건을 작성하여 생성한 버튼 B 부분 코드의 화면이다.

표 5 System operation 별 버튼 할당 예시

Table 5 An example of assigning button into the system operation

Use case	System operation	Btn
Change Mode	changeMode	A
Set Current Time	setCurrentTime	B
	setCurrentTimeValue	C, D
	completeSetCurrentTime	B
Operate Timer	operateTimer	C
Add Melody	addMemody	B
	setMelodyValue	C, D
	completeAddMelody	B

```
if (btnA == true) {
    //@btnA start
        systemOperation.changeMode();
    //@btnA end
} else if (btnB == true) {
    //@btnB start
    if(SystemOpeartions_gen.Function_status==1){
        systemOperation.setFunction();
    }
    else if(SystemOpeartions_gen.Function_status==2){
        systemOperation.setCurrentTime();
    }
    else if(SystemOpeartions_gen.Function_status==2 && SystemOpeartions_gen.TimeKeepingMode_isActive==true){
        systemOperation.completeSetCurrentTime();
    }
    else if(SystemOpeartions_gen.Function_status==3){
        systemOperation.addAlarm();
    }
    else if(SystemOpeartions_gen.Function_status==8 && SystemOpeartions_gen.AlarmMode_isActive==true){
        systemOperation.setAlarmValue();
    }
    else if(SystemOpeartions_gen.Function_status==3 && SystemOpeartions_gen.AlarmMode_isActive==true){
        systemOperation.completeAddAlarm();
    }
    else if(SystemOpeartions_gen.Function_status==4){
        systemOperation.setTimer();
    }
    else if(SystemOpeartions_gen.Function_status==4 && SystemOpeartions_gen.TimerMode_isActive==true){
        systemOperation.completeSetTimer();
    }
    else if(SystemOpeartions_gen.Function_status==6){
        systemOperation.addAnniversary();
    }
    else if(SystemOpeartions_gen.Function_status==9 && SystemOpeartions_gen.AnniversaryMode_isActive==true){
        systemOperation.selectAnniversaryMelody();
    }
    else if(SystemOpeartions_gen.Function_status==6 && SystemOpeartions_gen.AnniversaryMode_isActive==true){
        systemOperation.selectDeleteAnniversary();
    }
    else if(SystemOpeartions_gen.Function_status==7){
        systemOperation.addMelody();
    }
    else if(SystemOpeartions_gen.Function_status==7 && SystemOpeartions_gen.MelodyMode_isActive==true){
        systemOperation.completeAddMelody();
    }
    else if(SystemOpeartions_gen.Function_status==0 && SystemOpeartions_gen.MelodyMode_isActive==true){
        systemOperation.selectDeleteMelody();
    }
```

그림 14 버튼 A, B 부분 코드 생성 화면

Fig. 14 A screen of the generated code about buttons A and B

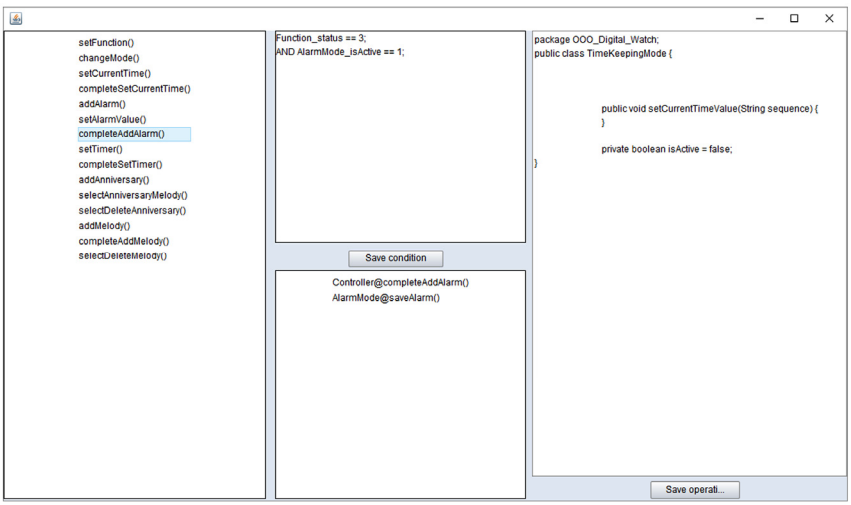


그림 13 버튼 B에 해당하는 system operation 별 조건 작성 화면

Fig. 13 A screen of the writing conditions of system operation about button B

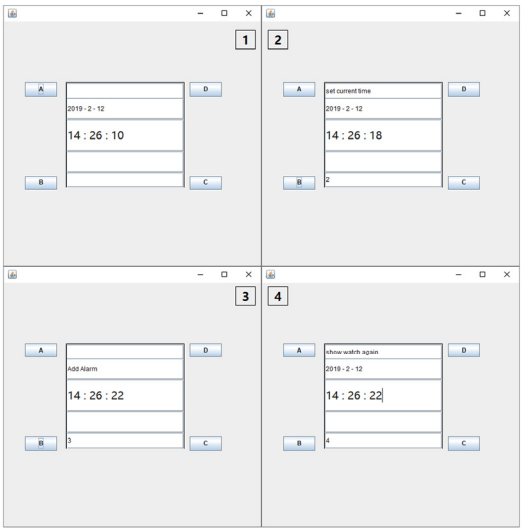


그림 15 실행 화면(일부 코드 작성 후)
Fig. 15 An execution result of the digital watch project

그림 15는 최종적으로 system operation의 동작 코드 부분을 label text를 이용해 코딩하고 실행한 화면의 일부이다. 이처럼 주변 환경의 코드를 생성하고 실제 로직만을 작성하여 실행함으로써 동작의 결과 확인 및 system operation으로 연결되는 layered architecture에 대해 효과적으로 확인할 수 있다. 또한 디자인과 구현의 연결에 대해서도 명확한 system operation을 기준으로 하여 쉽게 확인할 수 있다. OOPT 방법론의 stage 2050에서는 이 부분에 대해 사용자와 시스템 사이에 GUI와 관련된 class를 넣는 형태로 정의만을 하였기 때문에 system operation을 통한 layered architecture에 대한 고려가 부족하였다. 본 논문에서 제안하는 OSDEF에서 제공하는 system operation의 조건과 실제 GUI 코드와의 연결을 통해 직접 작성하여 개발이 필요한 GUI 코드와 연결 부분을 실제 층 구조와 유사한 구조로 자동 생성하여 층 구조에 맞는 구현을 쉽게 확인할 수 있다.

작성 중 use case, system operation, method 등의 이름을 정확하게 일관성 있도록 작성하지 않을 경우 그 연결 관계 파악이 어려워 리스트 제공이 어렵다는 한계점이 존재하지만, 이 점들은 분석, 디자인의 일관성 유지 및 추적성 분석 차원에서 필요한 작업으로 실제 실행까지 포함한 실습에 도움이 될 것으로 기대한다.

5. 결론 및 향후 연구

본 논문에서는 객체지향 소프트웨어 개발 교육의 효과적인 실습을 위해 객체지향 소프트웨어 개발 실습 프레임워크 OSDEF를 제안하였다. OSDEF는 객체지향 개발 교육용 개발 프로세스인 OOPT를 바탕으로 하여 작성 지원 도구 및 코드 생성, layered architecture를 대상으로 하는 execution 환경을 제공한다. 이를 통해 OO 교육에서 구현의 소요를 줄이고, 구현과 디자인의 연결 관계를 쉽게 확인하는 등의 장점을 가지고 실습을 효과적으로 수행할 수 있다.

향후 임베디드 소프트웨어 외에 다양한 소프트웨어 architecture를 중심으로 확장하는 연구를 수행할 계획이며, 이에 맞추어 UI 부분을 직접 디자인하고 연결하는 연구를 수행해 다양한 소프트웨어 개발 교육에 도움이 되는 방법을 연구할 계획이다. 또한 개발한 프레임워크를 건국대학교 컴퓨터공학과의 소프트웨어공학 관련 수업 (소프트웨어 모델링 및 분석)을 통해 적용하여 활용 방법에 대해 지속적으로 보완할 계획이다.

References

[1] Winston W Royce, "Managing the development of large software systems," *Proceedings of IEEE WESCON*, Vol. 26, No. 8, pp. 328-338, 1970.
[2] Ian Sommerville, *"Software Engineering,"* Addison-Wesely, Boston, 2007.
[3] F. A. Masoud, M. U. Shaikh, and S. H. 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.
[4] Grady Booch, *"Object-oriented Analysis and Design with Applications 3rd edition,"* Addison-Wesley, Boston, 2007.
[5] Per Kroll, Philippe Kruchten, and Grady Booch, *"The Rational Unified Process Made Easy a Practitioner's Guide to the RUP,"* Addison-Wesley, Boston, 2003.
[6] Danijela BOBERIC-KRSTICEV, Danijela TESENDIC, *"Experience in Teaching OOAD to Various Students," Informatics in Education*, Vol. 12, No. 1, pp. 43-58, 2013.
[7] Sejin Jung, Dong-Ah Lee, Eui-Sub Kim, ChunHyon Chang, and Junbeom Yoo, "OOPT: An Object-Oriented Development Methodology for Software Engineering Education," *Journal of KIISE: Software and Applications*, Vol. 44, No. 5, pp. 510-521, 2017. (in Korean)
[8] Jaehyun Kim, Hungu Ha, Sejin Jung, and Junbeom Yoo, "A Framework for Traceability analysis in OOPT, a Development process for education," *2018 Korea Software Congress*, pp. 1734-1736, 2018. (in Korean)
[9] Craig Larman, *"Applying UML and Patterns: An introduction to Object-Oriented Analysis and Design and Iterative Development 3rd Edition,"* Addison-Wesley, Boston, 2004.

[10] Suvarna G. Kanakaraddi, Jayalaxmi G. Naragund, and Ashok K. Chikaraddi, "Active Learning Method for Teaching OOAD Course," *2013 IEEE International Conference in MOC, Innovation and Technology in Education*, pp. 47-52, 2013.
[11] G. Kelecic and Z. Car, "Teaching Software Process: An Experience in Implementing RUP in a Student Project," *Proceedings of the 8th International Conference on Telecommunications*, pp. 479-484, 2005.
[12] Justin C. Debuse and Tony Stiller, "Technologies and Strategies for Integrating Object-Oriented Analysis and Design Education with Programming," *19 th Australian Conference on Software Engineering*, pp. 97-103, 2008.
[13] Rafael Rivera-lopez, Estela Rivera-Lopez, and Abelardo Rodriguez-leon, "Another Approach for the Teaching of the Foundations of Programming using UML and Java," *Proceedings of the 3rd WSEAS International Conference on Computer Engineering and Applications*, pp. 279-283, 2009.
[14] Micaela Esteves and Antonio Jose Mendes, "A Simulation Tool to Help Learning of Object-Oriented Programming Basics," *34th ASEE/IEEE Frontiers in Education Conference*, 2004.
[15] Violet UML editor [Online], Available: http:// alexdp. free.fr/violetumleditor/page.php (downloaded 2019, Feb. 07)

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

유 준 범

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