J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr

코드 가시화 툴체인 기반 UML 설계 추출 및 검증 사례☆

Best Practices on Validation and Extraction of Object oriented Designs with Code Visualization Tool-chain

이 원 영[1] 김 영 철[2*]
Won-Young Lee Robert YoungChul Kim

요 약

본 논문은 역공학 기반 소프트웨어 가시화를 통해 설계 개선 실현과 고품질화에 초점을 두고 있다. 앞으로의 4차 산업의 다양한 영역에서 새로운 기술과 복잡한 소프트웨어가 대두됨에 따라 안정성과 신뢰성을 겸비한 소프트웨어 검증이 이슈화되고 있다. 간단한 연산 소프트웨어부터 기계 학습기반의 데이터 지향 소프트웨어까지 다양한 소프트웨어 고품질화를 위한 역공학 기반 UML 설계 추출 및 가시화 방법을 제안한다. 이를 기반으로 목표 설계에 대한 정확도를 확인하고 코드 내부 복잡도 식별을 이용하여 설계 개선을 통한 소프트웨어 품질 향상을 기대한다.

☞ 주제어 : 역공학, 소프트웨어 리팩토링 가시화, UML,, 툴체인, 자바, 객체지향

ABSTRACT

This paper focuses on realizing design improvement and high quality through visualization of reverse engineering-based software. As new technologies and complex software emerge in various areas of the fourth industry in the future, software verification with both stability and reliability is becoming an issue. We propose a reverse engineering-based UML design extraction and visualization for high-quality software ranging from simple computational software to machine learning-based data-oriented software. Through this study, it is expected to improve software quality through design improvement by checking the accuracy of the target design and identifying the code complexity.

☞ keyword : Reverse Engineering, Software Refactoring, Visualization, UML, Tool-chain, Java, Object-Oriented

1. 서 론

4차 산업혁명으로 로봇 공학, 인공 지능, 사물 인터넷 등 다양한 분야에서 많은 소프트웨어가 대두됨에 따라 그에 걸맞은 고품질화를 위한 기술이 요구되고 있다. 또한, 소프트웨어의 유지보수를 어렵게 만드는 특성인 비가시성 (Invisibility)을 통한 복잡성 (Complexity)을 구분하기 어렵다[1]. 특히 IT 벤처/중소업체의 경우에 빈번한 개발자의 이직으로 요구사항 및 설계문서의 부재로 유지보수 체계가 부실하다[2]. 또한 소프트웨어 패치(patch)는 소프트웨어 복잡도를 증가시켜 더 큰 문제를 야기한다. 최근에는 소프트웨어 산업진흥법에 의한 소프트웨어 분리발주 제도가 새롭게 제정되었다. 기존에 시행되던 일괄발주는 하드웨어 및 소프트웨어 구매, 분석 및 설계, 개발 등 모든 단계를 묶어서 진행하고 단일 소프트웨어 사업자가 진행하였다. 하지만 분리발주는 각 단계마다 다른 발주기관이 사업에 참여하여 시스템을 분석하고 진행한다. 이에 설계 단계에서는 설계를 체계적으로 검증할 방안과 지표가 필요하다[11].

본 연구는 구현에 따른 설계의 순환으로 소프트웨어 설계 완성도를 개선 및 검증하여 소프트웨어 품질을 높인다. 소프트웨어 개발 라이프 사이클의 설계 단계에서 검증을 함으로써 개발 단계로 진행되기 전에 오류를 점검하고 수정할 수 있도록 한다. 또한 코드 내부의 복잡도(클래스 간 의존 관계와 순차적 관계 다이어그램)를 가시화하여 소프트웨어의 비가시성과 복잡성을 극복하고자 한다.

본 논문은 2절에서 관련 연구인 소프트웨어 역공학, 리팩토링, 가시화 프로세스를 소개하고, 3절에서 역공학

1 Cyber Security Team, Defense Agency for Technology and Quality, Daejeon, 35409, Korea.
2 Software Engineering Lab, Hongik University, Sejong, 30016, Korea.
* Corresponding author (wylee@dtaq.re.kr)
[Received 26 February 2021, Reviewed 12 March 2021(R2 8 October 2021), Accepted 19 November 2021]
☆ 본 논문은 2020년도 한국인터넷정보학회 추계학술발표대회 우수논문 추천에 따라 확장및 수정된 논문임.

기반 UML 설계 검증을 위한 가시화 프로세스를 소개한다. 4절에서는 연구에 대한 결론 및 향후 연구를 언급하고 5절에서 결론을 제시하며 마무리한다.

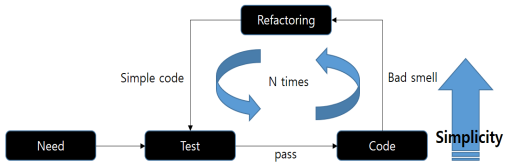
2. 관련 연구

2.1 역공학

소프트웨어 역공학은 개발되어진 소프트웨어를 분석하여 하위 산출물로부터 상위 단계의 문서나 설계도면과 같은 산출물을 복원하는 작업이다[3]. 이는 상용화되거나 기존 개발된 소프트웨어에 대한 자료와 정보를 설계 수준에서 분석할 수 있게 하여 유지보수성을 향상시킨다. 또, 기존 시스템 정보를 Repository에 보관하여 자동화 도구(CASE)의 사용을 용이하게 한다. 이를 통해 재문서화를 통하여 현존하는 시스템의 지식을 재 획득하고 현존 시스템 설계를 재사용할 수 있다. 하지만 노후한 시스템의 경우, 하위 단계에서의 소스코드부터 상위 단계의 소프트웨어 구조까지 파악하는 데에 많은 시간이 소요되기 때문에 많은 비용과 시간이 요구될 수 있다.

2.2 소프트웨어 리팩토링

리팩토링(Refactoring)은 유지보수 생산성 향상을 위해 기능을 변경하지 않고 소스코드를 수정 및 보완하는 소프트웨어 품질 향상 기법이다. 외부 동작을 변경하지 않으면서 내부 구조를 개선하는 방법으로, 소프트웨어 시스템을 수정하는 프로세스이다[4]. Dirk Riehle는 리팩토링은 생활 시스템의 지속적인 보수와 유사하다고 언급했다[5]. 리팩토링의 목표는 지속적인 손상에 대해 제한된 비용 내에서 합리적인 작동을 유지하는 것이다. 이는 소프트웨어 코드에서 중복되거나 불필요한 복잡성을 제거하여 최적화하고 간결화하여 가독성을 높이는 것과 같은 방식이다. 그림 1은 소프트웨어 리팩토링의 과정이다. 테스트 단계에서 모든 테스트를 통과했다면 그 코드에 대해 발생할 수 있는 bad smell을 파악한다. 잠재적인 오류를 제거하기 위하여 코드를 간결하게 수정하고 이를 다시 테스트하는 과정을 무한 반복한다. 다만, 테스트에 통과하지 못했거나 소스 코드의 수정이 필요할 경우에는 리팩토링을 수행하지 않는다. 이 과정의 반복으로 잠재적인 오류를 제거할 수 있으며 코드를 이해하기 쉽게 만들기 때문에 수정에 용이해진다. 결과적으로 리팩토링으로 설계를 검증할 수 있고, 코드를 간결화하여 높은 가독성과 이해성을 가질 수 있다. 또 잠재적인 오류를 제거하여 소프트웨어를 최적화할 수 있다.

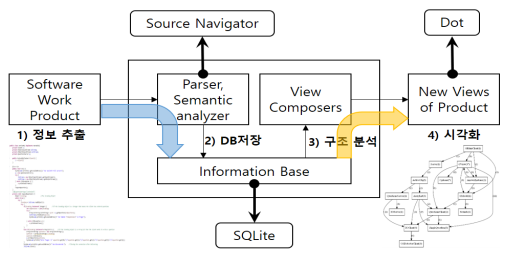


(그림 1) 리팩토링 절차
(Figure 1) Process of Refactoring

2.3 소프트웨어 가시화

소프트웨어 개발 및 관리에서 가장 어려운 점이 소프트웨어 비가시성이다. 비가시성은 소프트웨어 개발 과정 전반에서 발생하는 다양한 문제의 파악을 어렵게 한다. 이는 소프트웨어 가시화를 통해 극복이 가능하다. 소프트웨어 개발 과정을 시각화를 이용하여 효율적으로 관리하고 전체 과정을 파악하여 소프트웨어 개발의 품질을 관리할 수 있다. 또, 소프트웨어 개발 과정의 투명성을 확보함으로 인해 소프트웨어 개발 문제를 조기 검출하여 유지보수 비용 절감으로 기업의 경쟁력이 확보될 수 있다.

2.4 기존연구의 Tool-Chain



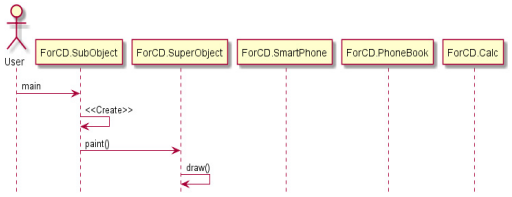
(그림 2) 기존연구의 Tool-chain[6-10]
(Figure 2) Tool-chain of Related research

그림 2는 기존 연구[6 - 10]에서 구현한 코드 가시화 도구 구성도이다. 소스 코드 파싱도구로 Source Navigator를 사용하고, 분석된 데이터를 데이터베이스에 저장한다. 그리고 필요한 데이터들만 추출하여 오픈소스 Dot으로 그래프화 한다.

3. 역공학 기반 UML 설계 복원

3.1 소프트웨어 가시화

기존 Tool-chain에서 사용한 Source Navigator는 C와 Java를 모두 분석해주는 오픈소스 도구이다. 전체 29가지의 소스코드 분석 파일들을 제공하지만, Java 코드를 분석하여 도출되는 결과 파일은 cl(Class), in(Inheritances), iv(Instance variables, lv(Local variables), md(Method definitions)로 총 6개의 파일뿐이다. 그리고 업데이트 되며 새로운 문법들이 생기는 Java에 비해 Source Navigator는 업데이트가 이루어지지 않는다. 이러한 이유로 Source Navigator를 사용하여 Java 코드를 정적 분석하기에는 분석되는 데이터가 미흡하다. 이를 해결하기 위해 Java 전용 파서인 Java parser를 사용하여 새로운 Tool-chain을 개발하였다. Java parser는 Java 소스코드를 AST(Abstract Symbol Tree)구조로 분석하는 오픈소스로 빠르고 사용이 간편한 도구이다.



(그림 3) PlantUML 결과물(순차 다이어그램)
(Figure 3) Output of PlantUML(Sequence Diagram)

```
@startuml
actor User
User -> ForCD.SubObject : main
ForCD.SubObject -> ForCD.SubObject :  <<Create>>
ForCD.SubObject -> ForCD.SuperObject : paint()
ForCD.SuperObject -> ForCD.SuperObject : draw()
@enduml
```

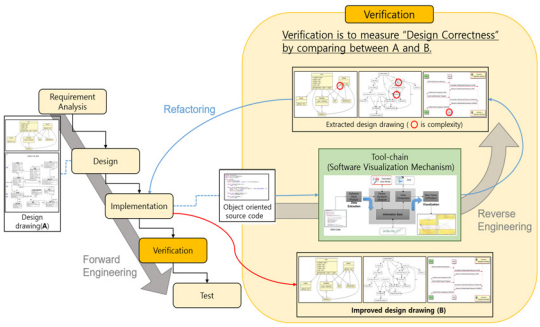
(그림 4) PlantUML 순차 다이어그램 스크립트
(Figure 4) Sequence Diagram script of PlantUML

기존에 사용하던 Dot 프로그램을 PlantUML로 대체가 필요하다. Dot은 하나의 그래프 모델을 다양한 알고리즘을 통해 가시화하지만, PlantUML은 UML 다이어그램만을 가시화하여 각 다이어그램마다 다른 모델을 사용한다. PlantUML은 UML 작성을 도와주는 오픈소스 툴로 간단한 언어로 객체간의 관계를 정의하여 UML을 생성한다. 그로 인해 다이어그램마다 스크립트 언어가 다르지만, 하나의 도구로 여러 가지의 다이어그램을 가시화할 수 있다. 이 도구는 순차 다이어그램, 유스케이스 다이어그램, 클래스 다이어그램, 액티비티 다이어그램, 컴포넌트 다이어그램, 상태 다이어그램, 객체 다이어그램 등 다양한 다이어그램을 지원한다. StarUML, Eclipse의 AmaterasUML 등의 무료 UML도 있지만, PlantUML은 GUI로 UML을 작성하지 않고 텍스트로 작성하므로 Git과 같은 형상관리 툴 적용이 가능하다. 손쉬운 소스코드 관리로 협업이 가능하며 언제든지 변경 이력을 확인하고 이전 내역으로도 복구가 가능한 장점이 있다. 그림 3은 PlantUML으로 출력한 순차 다이어그램의 일부분이다. 수평 축에 엑터와 클래스가 나열되어 있으며 수직 방향으로 생명선이 점선으로 그려진다. 수평 방향의 실선은 호출 메시지이고 점선은 반환 메시지를 나타낸다. 그림 4는 그림 3의 순차 다이어그램을 나타내기 위한 스크립트 문이다. 메시지를 호출하는 클래스들과 수신하는 메시지, 반환 값들이 입력된다.

마지막으로 새로 개발한 Tool-chain에는 데이터베이스 단계가 없다. 데이터베이스를 사용하면 데이터를 따로 저장하기 때문에 자바 소스코드의 전체적인 무게가 가벼워지지만 복잡한 쿼리문을 적용시켜야 한다. 하지만 그림 2의 Tool-chain은 Java 소스코드를 Java parser로 분석하는 Java 기반이기 때문에 데이터베이스를 설치해야하는 번거로움을 없애고 Java 프로그램 하나만으로 사용이 가능하여 데이터 관리차원이나 개발하는 입장에서 편리하다. 게다가 PlantUML로 그림을 그리는 작업도 역시 Java로 스크립트를 작성하기 때문에 불편함이 없다. 데이터베이스 쿼리문같은 경우는 개발자 취향대로 원하는 자바 함수를 사용하여 대체할 수 있고 나중에 필요하다면 선택적으로 데이터베이스 작업을 처리해도 된다는 장점이 있다.

3.2 Java parser와 PlantUML을 이용한 설계 복원

그림 5는 역공학을 기반으로 설계도를 복원하여 검증하는 프로세스이다. 순공학 프로세스에서 소프트웨어를 설계한 설계도면(A)와 역공학 프로세스를 통해 소프트웨어 Tool-chain을 통해 얻어진 설계도면(B)을 비교하여 '디자인 정확도'를 검증한다. 두 설계의 비교를 통해 현재 설계(B)의 복잡도를 측정하고 설계를 개선해야할 부분을 확인할 수 있다. 이상적인 설계도면(A)에 가까워지도록 소프트웨어 리팩토링을 수행하여 설계 복잡도를 낮출 수 있다. 결과적으로 '설계→구현→설계' 과정의 반복으로 소프트웨어 설계를 검증하여 완성도를 개선하여 소프트웨어 품질을 높일 수 있다.



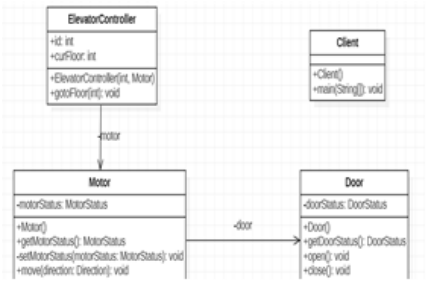
(그림 5) 역공학 기반 설계 검증
(Figure 5) Design Validation based on reserve engineering

아래는 소프트웨어 개발 프로세스에 적용하여 역공학 기반으로 설계를 검증하는 사례이다. 엘리베이터 부품이 다른 경우엘리베이터 작동 예제를 사용하여 검증한다. 제일 먼저 소프트웨어 개발 프로세스에서 요구사항 분석단계를 거쳐야 한다. 표 1은 엘리베이터 작동 프로그램을 개발하는 데 필요한 요구사항이다. 건물 A, B의 엘리베이터는 서로 다른 회사 부품을 사용하고 엘리베이터는 운행을 하기 전에 반드시 문을 닫아야 한다. 엘리베이터는 모터를 통해 운행되며 반드시 엘리베이터 문이 닫혀있는지 열려있는지 상태를 알려주어야 한다.

(표 1) 엘리베이터 요구사항
(Tabel 1) User requirement for elevator

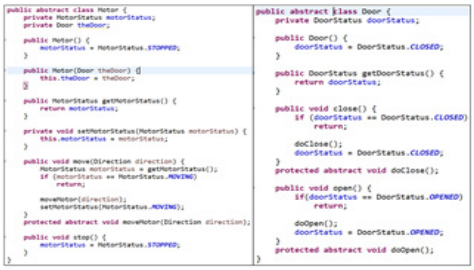
번호	요구사항
1	건물 A에는 LG의 부품을 사용하고 건물 B에는 현대의 부품을 사용한다.
2	같은 회사 모터와 문을 사용하도록 한다. 예를 들어 LG모터를 사용하는 엘리베이터는 LG문을 사용한다.
3	엘리베이터는 위, 아래로 이동한다.
4	엘리베이터 상태는 작동중인 상태와 멈춰있는 상태 두 가지이다.
5	엘리베이터는 이동하기 전에 문을 닫아야 한다.
6	엘리베이터는 모터를 구동해서 이동한다.
7	문이 열려 있으면 문 상태를 '열림'으로 설정하고 닫혀 있으면 '닫힘'으로 설정한다.

그림 6은 표 1의 요구사항을 기반으로 설계 단계에서 그린 클래스 다이어그램이고 이를 설계도 A라고 칭한다. 요구사항을 반영하여 Motor 클래스와 Door 클래스를 생성한다. Motor 클래스에는 모터의 상태를 MOVING과 STOPPED 두 가지 상태로 나타내고 엘리베이터 작동 방향은 위, 아래로 설정한다. 그리고 LGMotor 클래스와 HyundaiMotor 클래스는 Motor 클래스를 상속하여 구현하는 구조로 설계한다. 마찬가지로 Door 클래스는 엘리베이터가 작동되기 전에 닫혀야 하고 문의 상태를 알려주는 기능을 설계도에 표현한다. 그리고 LGDoor 클래스와 HyundaiDoor 클래스는 Door 클래스를 상속하여 구현한다.



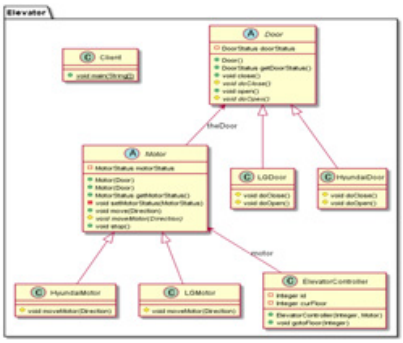
(그림 6) 엘리베이터 클래스 다이어그램
(Figure 6) Class Diagram for elevator

다음으로 그림 7은 설계도를 바탕으로 프로그램 개발 단계에서 작성한 Motor 클래스와 Door 클래스의 소스 코드이며, 그림 8은 개발단계의 산출물로 얻은 소스 코드를 역공학 기반의 Tool-chain으로 복원한 클래스 다이어그램 설계도면이다. 아래의 클래스 다이어그램을 설계도 B라고 칭한다. 이제 설계도 A와 B의 대조를 통해 설계도 A가 잘 설계되었는지 확인할 수 있다. 또, 설계도 B를 통해 설계 복잡도를 가늠할 수 있으며 설계 복잡도가 높은 부분은 소스 코드를 리팩토링하여 설계도 A와 같이 이상적인 설계도를 가질 수 있도록 개선할 수 있다.

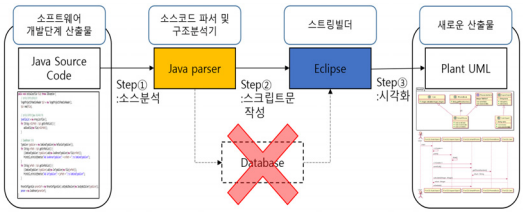


(그림 7) 엘리베이터 구현 코드
(Figure 7) Source code for elevator

그림 9은 객체지향 코드로부터 UML을 복원하기 위한 역공학기반 소프트웨어 가시화 Tool-chain 구조이다. 아래는 Java parser와 PlantUML을 이용하여 설계를 복원하는 과정이다.



(그림 8) 역공학 기반 Tool-chain 복원 설계도
(Figure 8) Recovered design using Tool-chain based on reserve engineering



(그림 9) 객체지향 Tool-chain
(Figure 9) Object-oriented Tool-chain

- Step 1에서 자바 소스코드를 Java parser에 입력하여 소스코드를 분석한다.
- Step 2에서 Java parser가 분석한 결과를 기존처럼 각각의 class/method/.. 테이블화하여 데이터베이스에 저장하지 않고 Eclipse내의 ModelManager.java 클래스를 통해 DiagramRecover.model Package 내 Mclass, MMember, MOperation, MOperationCall 모듈형태로 메모리에 저장한다. 즉, Java parser로 분석된 데이터들이 클래스와 메소드의 호출관계, 리턴값, 매개변수, 클래스 이름, 멤버변수, 변수 타입, 메소드 타입 등등으로 저장된다.
- Step 3에서 저장된 데이터를 기반으로 원하는 구문 규칙을 생성하여, 설계를 가시화 한다. 즉, 내부 메모리에 저장된 데이터(Mclass, MMember, MOperation, MOperationCall)들을 PlantUML 스크립터 구문을 프로그램화 하여 SequenceDiagramScriptMaker, ClassDiagramScriptMaker 클래스를 통해 UML을 설계한다.
- Step 4: 자동으로 UML 설계를 가시화 한다.

4. 객체지향 코드 설계복원 사례

그림 10은 객체지향 Tool-chain에 적용한 input의 일부분으로 클래스 간의 상속관계를 보여주기 위한 샘플 예제이다.

- step 1: source code 입력

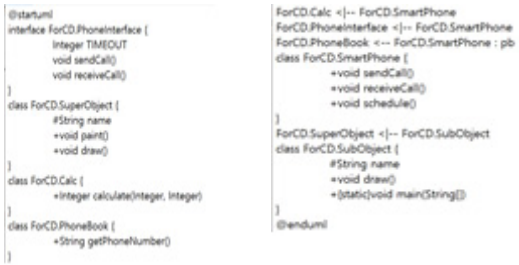
```
public class SubObject extends SuperObject {
    protected String name;
    public void draw() {
        name = "Sub";
        super.name = "Super";
        super.draw();
        System.out.println(name);
    }
    public static void main(String [] args) {
        SuperObject b = new SubObject();
        b.paint();

        SmartPhone phone = new SmartPhone();
        phone.sendCall();
        System.out.println("3과 5를 더하면 " + phone.calculate(3, 5));
        phone.schedule();
    }
}
```

(그림 10) input 소스코드
(Figure 10) Source code for Input

위의 샘플코드를 설계 복원 tool-chain에 분석할 타겟 소스로 설정하고 Java parser를 통해 구문을 분석한다.

- step 2: 분석한 결과는 다른 도구를 사용하지 않고 Eclipse 스트링빌더에 저장하여 바로 메모리에서 사용할 수 있다. MClass 클래스는 클래스에 대한 정보로 접근자, 클래스 이름, 패키지 이름, 메소드 이름, 부모 클래스 이름, 객체들의 이름을 가지고 있다. Mmember 클래스에는 변수 타입이랑 이름 정보를 가지고 있다. MOperation 클래스는 메소드를 호출하는 정보를 알 수 있고, MOperationCall 클래스는 호출된 메소드의 정보를 알 수 있다. 마지막으로 ModelManager 클래스는 위에 나열한 클래스들에 저장된 모든 분석 데이터들을 가진다.
- step 3: 저장된 분석 데이터들을 open source인 PlantUML을 통해 클래스 다이어그램과 순차 다이어그램 스크립트 구문에 맞게 작성한다. 그림 11, 12는 작성된 스크립트 구문이다. 수집된 데이터와 호출관계를 바탕으로 클래스 다이어그램 스크립트로 작성되고 수집된 데이터와 호출 및 반환관계를 바탕으로 작성된 순차 다이어그램 스크립트이다. Tool-chain이 소스코드 분석을 마치면 sequenceDiagramScript.txt 파

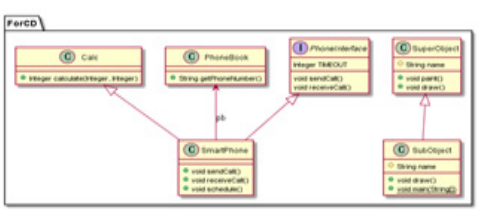
일을 생성하고 맨 앞에는 시작을 알리는 @startuml로 시작하며 @enduml로 끝난다.



(그림 11) 클래스 다이어그램 스크립트
(Figure 11) Class Diagram script

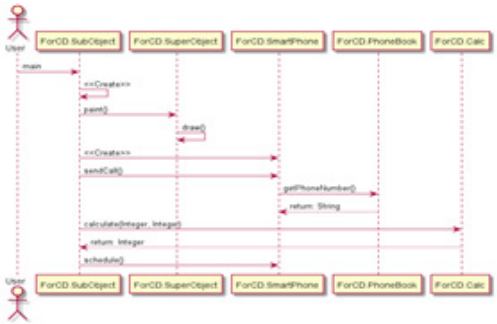
(그림 12) 순차 다이어그램 스크립트
(Figure 12) Sequence Diagram script

- step 4: 각각의 스크립트 구문에 의해 자동으로 UML 설계를 가시화하는 단계이다. 그림 13, 14는 각각 복원한 클래스 다이어그램과 순차 다이어그램이다. 클래스 다이어그램을 복원하기 위해 클래스 명, 메소드 식별자, 변수, 메소드 명 등 필요한 정보를 수집한다. 또, 상속, 연관, 의존 관계 등을 표현할 클래스 간의 관계에 대한 정보도 수집한다. 클래스 간의 상속관계는 일반화관계에서, 의존관계는 호출관계로부터 추출할 수 있다.



(그림 13) 복원 클래스 다이어그램
(Figure 13) Recovered Class Diagram

반면에 순차 다이어그램은 클래스 다이어그램에서 분석된 정보를 깊이 우선으로 탐색하여 스크립트를 생성한다. 호출하는 클래스와 메소드, 호출되는 클래스와 메소드의 정보를 통해 그려진다. 그리고 메소드의 리턴 타입을 분석하여 반환 값도 고려한다.



(그림 14) 복원 순차 다이어그램
(Figure 14) Recovered Sequence Diagram

5. 결론 및 향후 연구

최근 급속한 시장 환경 변화와 소프트웨어의 특징인 비가시성 (Invisibility), 복잡성 (Complexity)로 인하여 소프트웨어 품질향상이 어렵다. 이는 소프트웨어 생산성과도 연관이 된다. 이 연구를 통해 오픈소스 기반 코드 정적 분석기를 사용하여 기업의 소프트웨어 자산을 쉽게 구축할 수 있어 인력 문제를 해소할 수 있다. 그리고 IT 벤처/중소기업의 유지보수를 위한 지원도구로 활용되어 비용과 시간절감에 효과가 있다. 또, 실제 운영되는 코드를 통한 최신 설계도 생산이 가능하여 유지보수에 도움을 주고 설계 구조의 개선, 코드 복잡도 및 성능 분석 등이 가능해질 것이다. 역공학을 통해 추출된 설계도는 소프트웨어 설계에 대한 복잡도 가시화를 통해 모듈 간의 복잡도를 확인할 수 있어 모듈 결합력을 낮추고 응집도를 높혀 설계에 대한 소프트웨어 품질을 높일 수 있다. 이러한 검증 과정을 통해 기존에 발견하지 못한 문제점을 찾아내어 잠재적인 오류에 대한 문제해결이 가능하며 빠른 유지보수가 가능하고 급변하는 소프트웨어 기술에 대처가 가능하다. 본 논문에서 구현한 역공학 기반의 가시화 프로그램을 통해 설계를 검증하고 그에 따른 유지보수 비용을 줄일 수 있다. 향후에는 각 모듈에 대한 소프트웨어 설계 복잡도 메트릭스를 제안하여 품질을 측정하는 연구

를 진행할 예정이다.

참고문헌(Reference)

[1] So Young Moon, R. YoungChul Kim, "Code Structure Visualization with A Tool-Chain Method," International Journal of Applied Engineering Research, ISSN 0973-4562, Vol.10 No.99, 2015. http://selab.hongik.ac.kr/selab/data/papers/201512214-218_so-young.pdf

[2] Changjae Kim, Jaewon Park, "A Software Maintenance Capability Maturity Model Based on Service", Korea Institute of Information Technology, pp.173-184, 2014. http://dx.doi.org/10.14801/kiitr.2014.12.5.173

[3] Chikofsky, Elliot J., and James H, Cross. "Reverse engineering and design recovery: A taxonomy," Software, IEEE, 7(1), pp. 13-17, 1990. https://doi.org/10.1109/52.43044

[4] Martin Fowler, "Principles of Refactoring," in Refactoring, 2nd ed, Seoul, Korea.

[5] Bäumer, Dirk, and Dirk Riehle. "Product Trader." In Pattern Languages of Program Design 3, Mass.: Addison-Wesley, 1998.

[6] Won Young Lee, So Young Moon, R. Young Chul Kim, "The Constructing & Visualizing Practices in Effective Static Analyzer for analyzing the Quality of Object Oriented Source Code," Korea Information Processing Society, Vol. 38, No.2, pp.704-707, 2019. https://doi.org/10.3745/PKIPS.y2019m10a.704

[7] Bokyung Park, Haeun Kwon, Hyeoseok Yang, Soyoung Moon, Youngsoo Kim, R. Youngchul Kim, "A Study on Tool-Chain for statically analyzing Obect Oriented Code," Korea Computer Congress, pp.463-465, 2014. https://www.dbpia.co.kr/Journal/articleDetail?nodeId=NODE02444062

[8] Geon-hee Kang, HyunSeung Son, Youngsoo Kim, Young B. Park, R. Young Chul Kim, "Improving Static Code Complexity with Refactoring technique based on SW visualization," Korea Information Processing Society, Vol.21, No.2, pp.650-653, 2014. https://doi.org/10.3745/PKIPS.y2014m11a.650

[9] Haeun Kwon, Bokyung Park, R. Youngchul Kim, "Automatically Extracting Structural and Behavioral Designs From Object Oriented Programming", Korean institute of smart media, Vol.4, No.2, pp.129-131, 2015.

[10] Haeun Kwon, Bokyung Park, Young S. Kim, R. Youngchul Kim, "Extracting Use Case Design from Source Code based on Reverse engineering," Vol. 5 No. 1, pp.289-291, 2016.

[11] [Software Industry Information Total System[Website]. (2020 Sep 25). https://www.swit.or.kr/SO/SOI/soiIntro.jsp

◐ 저 자 소 개 ◑

이 원 영(Won-young Lee)
2014년 홍익대학교 컴퓨터정보통신공학과
2019년 홍익대학교 일반대학원 전자전산공학과(공학석사)
2020년 11월~2021년 국방기술진흥연구소 미래전력선행연구팀 연구원
2022년~현재 국방기술품질원 사이버보안기술팀 연구원
관심분야 : 소프트웨어공학, 설계검증, 사이버보안, etc.
E-mail : wylee@dtaq.re.kr

김 영 철(Rober YoungChul Kim)
2000년~2001년 LG산전 중앙연구소 Embedded System 부장
2001년~현재 홍익대학교 소프트웨어융합학과 교수
관심분야 : 소프트웨어공학, 인공지능, etc.
E-mail : bob@hongik.ac.kr