역공학 기반 UML 설계 복원을 위한 소프트웨어 가시화

이원영[1], 정세준[2], 권하은[3], 김영철[4]
[1,2,4]홍익대학교 소프트웨어공학연구실, [3]한국정보통신기술협회
e-mail : {[1]leewy, [2]bvcx79, [4]bob}@selab.ac.kr, [3]khe0402@tta.or.kr

Software Visualization for restoring UML Design based on Reverse Engineering

[1]Won Young Lee, [2]Se Jun Jung, [3]Ha Eun Kwon, [4]R. Young Chul Kim
[1,2,4]Hongik University Software Engineering Lab, [3]TTA

요 약

앞으로의 4차 산업의 다양한 영역에서 새로운 기술과 복잡한 소프트웨어가 대두되고, 안정성과 신뢰성을 겸비한 소프트웨어로부터 기계 학습기반의 데이터 지향 소프트웨어까지 다양한 소프트웨어 고품질화가 이슈화되고 있다. 본 연구에서는 역공학 기반 소프트웨어 가시화를 통해 설계 개선 실현과 고품질화에 있다. 이를 위해, 역공학 기반으로 UML 설계 추출 Tool-chain 구축 방법 및 가시화를 적용을 제안한다. 이는 객체지향 코드 내부 복잡도를 낮추기 위해 설계 개선을 통한 소프트웨어 품질 향상에 기여 할 수 있을 것으로 기대한다.

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 설계 문서의 부재로 유지보수체계가 없는 것이 실상이다[2]. 이러한 상황으로 소프트웨어를 지속적으로 패치(patch) 및 유지보수가 소프트웨어 복잡도를 증가시켜 더 큰 문제를 야기한다. 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.

본 논문은 2장에서 관련 연구인 소프트웨어 역공학, 리팩토링, 가시화 프로세스를 소개하고, 3장에서 역공학 기반 UML 설계 검증을 위한 가시화 프로세스를 소개한다. 4장에서는 연구에 대한 결론 및 향후 연구를 언급하며 마무리한다.

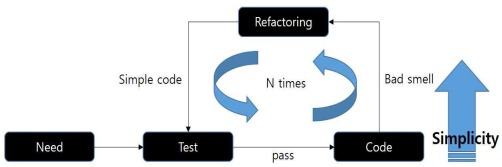
2. 관련 연구

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.

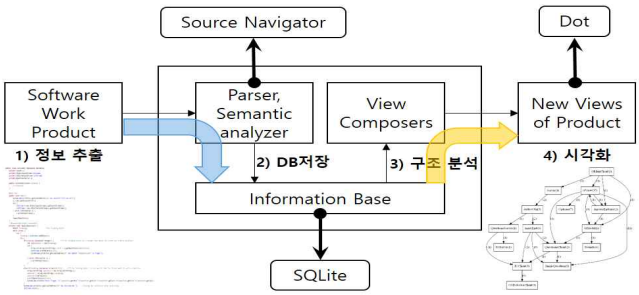
2.2 소프트웨어 리팩토링

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]. 그림 1은 소프트웨어 리팩토링의 과정이다.



(그림 1) 리팩토링 절차

2.3 기존연구의 Tool-chain



(그림 2) 기존연구의 Tool-chain[6, 7, 8, 9]

그림 2는 기존 연구[6, 7, 8, 9]에서 구현한 코드 가시화 도구 구성도이다. 소스 코드 파싱도구로 Source Navigator를 사용하고, 분석된 데이터를 데이터베이스에 저장한다. 필요한 데이터들만 추출하여 오픈소스 Dot으로 그래프화 한다.

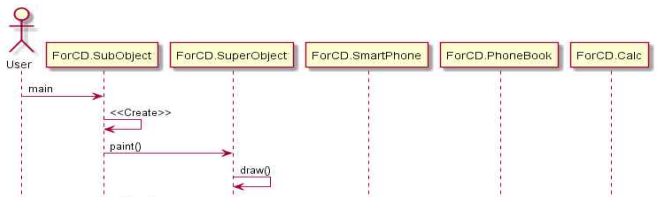
3. 역공학 기반 UML 설계 복원

3.1 기존 Tool-chain과의 차이점

기존 Tool-chain에서 사용한 Source Navigator는 C와 Java를 모두 분석해주는 오픈소스 도구이다. 전체 29가지의 소스

*) 본 논문은 2019년도 산업통상자원부의 '창의산업융합 특성화 인재양성사업(과제번호 N0000717)과 2020년도 정부재원(과학기술정보통신부 여대학원생 공학연구팀제 지원사업)으로 과학기술정보통신부와 한국여성과학기술인지원센터의 지원을 받아 연구되었습니다.

코드 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. 기존에 사용하던 Dot 프로그램을 PlantUML로 대체가 필요하다. 그림 4는 그림 3의 순차 다이어그램을 나타내기 위한 스크립트 문이다. 메시지를 호출하는 클래스들과 수신하는 메시지, 반환 값들이 입력된다.



(그림 3) PlantUML으로 출력한 순차 다이어그램 일부분

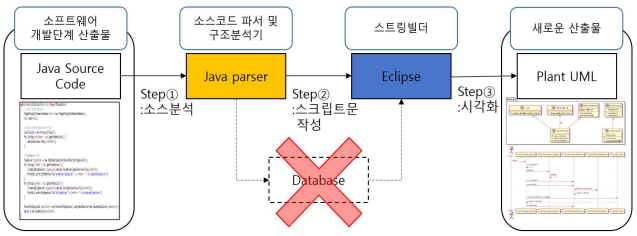
```
@startuml
actor User
User -> ForCD.SubObject : main
ForCD.SubObject -> ForCD.SubObject :  <<Create>>
ForCD.SubObject -> ForCD.SuperObject : paint()
ForCD.SuperObject -> ForCD.SuperObject : draw()
@enduml
```

(그림 4) PlantUML에서 사용하는 순차 다이어그램 스크립트

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.

3.2 Java parser와 PlantUML을 이용한 설계 복원

그림 5는 객체지향 코드로부터 UML을 복원하기 위한 소프트웨어 가시화 Tool-chain 구조이다. 아래는 Java parser와 PlantUML을 이용하여 설계를 복원하는 과정이다.



(그림 5) 객체지향 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 프로그램화하여 SequenceDiagramScriptMaker, ClassDiagramScriptMaker 클래스를 통해 UML을 설계한다.
- Step 4: 자동으로 UML 설계를 가시화 한다.

4. 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. 본 논문에서 구현한 역공학 기반의 가시화 프로그램을 통해 설계를 검증하고 그에 따른 유지보수 비용을 줄일 수 있다.

참 고 문 헌

[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.

[2] Changjae Kim, Jaewon Park, "A Software Maintenance Capability Maturity Model Based on Service", Korea Institute of Information Technology, 173-184, 2014.

[3] Chikofsky, Elliot J., and James H, Cross. "Reverse engineering and design recovery: A taxonomy." Software, IEEE 7.1, pp. 13-17, 1990.

[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, edited by R. Martin, F. Buschmann, and D. Riehle. Reading, 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, 704-707, 2019.

[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, 463-465, 2014.

[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, 650-653, 2014.

[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, 129-131, 2015.