# 나쁜 코딩 습관 개선을 위한 코드 가시화 연구

박지훈°, 김영철
홍익대학교 소프트웨어공학 연구실
e-mail : {pjh°, bob}@selab.hongik.ac.kr

# A Study on Software Visualization for Bad Smell Coding Improvement

Ji-Hoon Park°, R. Young-chul Kim
Software Engineering Lab., Hong-ik University

요 약

최근 소프트웨어 시장의 규모와 위상은 급속도로 성장하고 있다. 하지만 커지는 산업에 비해서 소프트웨어의 품질에 대한 인식은 아직 미흡하다. 국내 중소기업의 경우 인력, 비용 등의 여러 측면에서 어려움이 존재한다. 구현 중심의 개발이 이루어지면서 소프트웨어의 품질보다 빠른 개발에 초점을 두어 저 품질의 소프트웨어를 양산하고 있다. 본 논문에서는 이를 해결하기 위한 가시화를 통해 나쁜 코딩 습관을 개선하여 소프트웨어의 품질향상을 도모한다. 가시화를 통해 소스 코드에서 개선되어야 할 부분을 찾아내 소프트웨어의 품질을 향상시킬 수 있을 것으로 기대한다.

## 1. 서론

근래에 접어들면서 소프트웨어의 시장규모가 급속하게 성장하고 있다. 실제로 전체 IT 산업에서의 규모는 1조 달러로 전체 IT 산업의 30% 수준이며, 8000억 달러 수준인 하드웨어 시장보다 큰 것으로 나타났다. 그 이유는 소프트웨어의 중요성이 날이 갈수록 강조되고 있기 때문이다[1]. 그러므로 얼마나 좋은 품질의 소프트웨어를 생산해낼 수 있는지가 중요하게 여겨진다.

소프트웨어 공학적 접근의 개발을 한다면 해결되겠지만 국내 중소기업의 경우 인력, 비용 등의 여러 측면에서 어려움이 존재한다. 게다가 구현 중심의 개발이 이루어지면서 내부 구조화가 진행되고 있지 않고 있다[2]. 따라서 소프트웨어의 비가시성이라는 특징 때문에 관리자는 물론 개발자까지도 소프트웨어의 품질 관리 및 유지보수에 어려움이 따른다. 본 논문에서는 기존연구에서 역 공학을 통해 소스 코드를 가시화한다. 소프트웨어 가시화를 통하여 소스 코드의 결합도 및 나쁜 냄새(Bad Smell)를 추출하는 방법을 소개한다[3,4,5]. 오픈 소스인 Source Navigator를 이용하여 소스 코드의 정보를 얻어 결합도와 나쁜 냄새(Bad Smell)를 가시화 하였다. 가시화된 정보들을 이용하여 소스 코드에서 개선해야 할 부분을 찾아낸다. 궁극적으로 소프트웨어의 품질을 향상시킬 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구로 나쁜 냄새(Bad Smell)에 대한 개념을 설명한다. 3장에서는 나쁜 코딩습관을 가시화시켜주는 도구에 대해 설명한다. 마지막으로 4장에서는 결론 및 향후연구를 설명한다.

## 2. 나쁜 냄새(Bad Smell)

나쁜 냄새(Bad Smell)는 마틴 파울러의 'Refactoring'이라는 책에서 처음 쓰여진 비유이다. 소스 코드를 작성하다보면 마주치는 나쁜 코딩 습관들을 정리했다. 본 논문에서는 마틴 파울러의 총 22가지의 나쁜 냄새(Bad Smell) 중 표 1처럼 5가지를 소개할 것이다[6]. 이 항목에 적합한 부분이 소스 코드에 존재한다면 그것은 나쁜 코딩습관이 있는 소스 코드이다.

| No. | 항목 |
|---|---|
| 1 | Long Parameter List (LPL) |
| 2 | Feature Envy (FE) |
| 3 | Lazzy Class (LC) |
| 4 | Data Class (DC) |
| 5 | Message Chains (MC) |

< 표 1 > 나쁜 냄새(Bad Smell)의 항목

이러한 나쁜 냄새(Bad Smell)를 해결하기 위해 다음의 몇 가지 방법들을 지켜야 한다.

첫째로, 비슷한 코드가 여기저기에 존재한다면 프로그램을 수정하거나 유지보수 할 경우 어려움이 따른다. 이런 경우 겹쳐 있는 코드를 메소드 추출이나 클래스 추출이라는 리팩토링을 검토하는 것이 좋다.

둘째로, 메소드가 너무 길면 이해하기 어려우므로 메소드 추출이라는 리팩토링을 하는 것이 좋다.

셋째로, 클래스가 너무 크다는 이유로 클래스를 지나치게 많이 추출하면 안 된다. 커다란 클래스와 마찬가지로

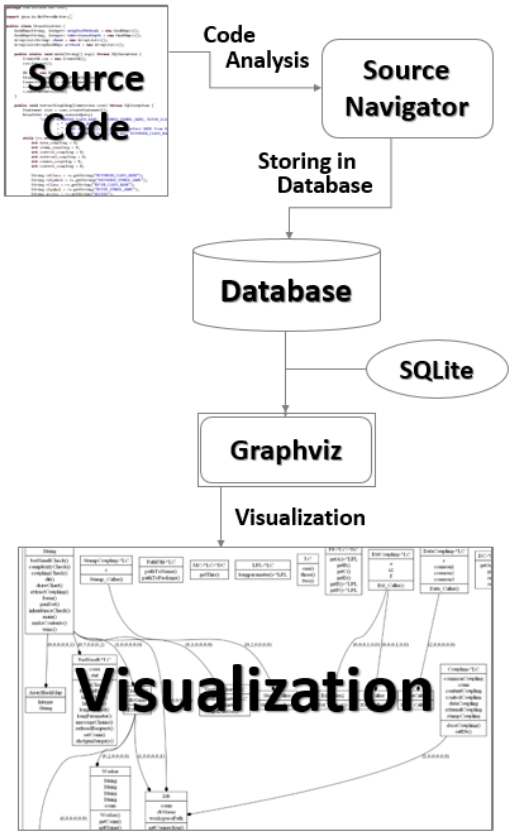
너무 많은 수의 클래스도 이해하는데 어려움이 따른다. 클래스나 메소드의 인라인화라는 리팩토링을 검토한다.

넷째로, 메소드의 이름을 개념과 어울리게 지어야 한다. 메소드명의 변경은 대표적인 리팩토링이다.

마지막으로, 필드나 클래스의 이름을 너무 많이 공개하면 안 된다. 필드의 캡슐화라는 리팩토링으로 필드를 감추거나 Refactory Method로 생성자를 치환하는 것을 검토해야 한다.

## 3. 제안 프로세스

본 연구에서는 기존 연구[3]의 결합도 추출에 더해서 나쁜 냄새(Bad Smell)를 추가적으로 추출하였다. 그림 1은 나쁜 코딩 습관을 개선할 수 있는 가시화 프로세스를 표현한 그림이다. JAVA code를 Source Navigator에서 분석한 후 Database에 저장한다. 저장된 데이터를 이용하여 결합도 및 나쁜 냄새(Bad Smell) 패턴을 찾아내어 코드의 내용을 그래프로 가시화 할 수 있다. 표 1에서 소개한 5가지의 항목에 대해서 측정할 수 있다. Source Navigator의 SNDB 파일들에서 Database로 추출해내는 정보는 다음과 같다. 클래스이름과 그 안의 변수와 메소드이름이 있다. 또한 메소드의 리턴값, 접근자, 파라미터 종류와 개수도 있다. 변수들의 이름과 리턴값, 접근자가 있다. 마지막으로 상속관계에 대한 정보도 들어있다. 이렇게 추출해낸 Sourece Navigator의 SNDB파일들의 정보를 정제하여 그래프에 필요한 나쁜 냄새(Bad Smell)를 가시화 시킨다.

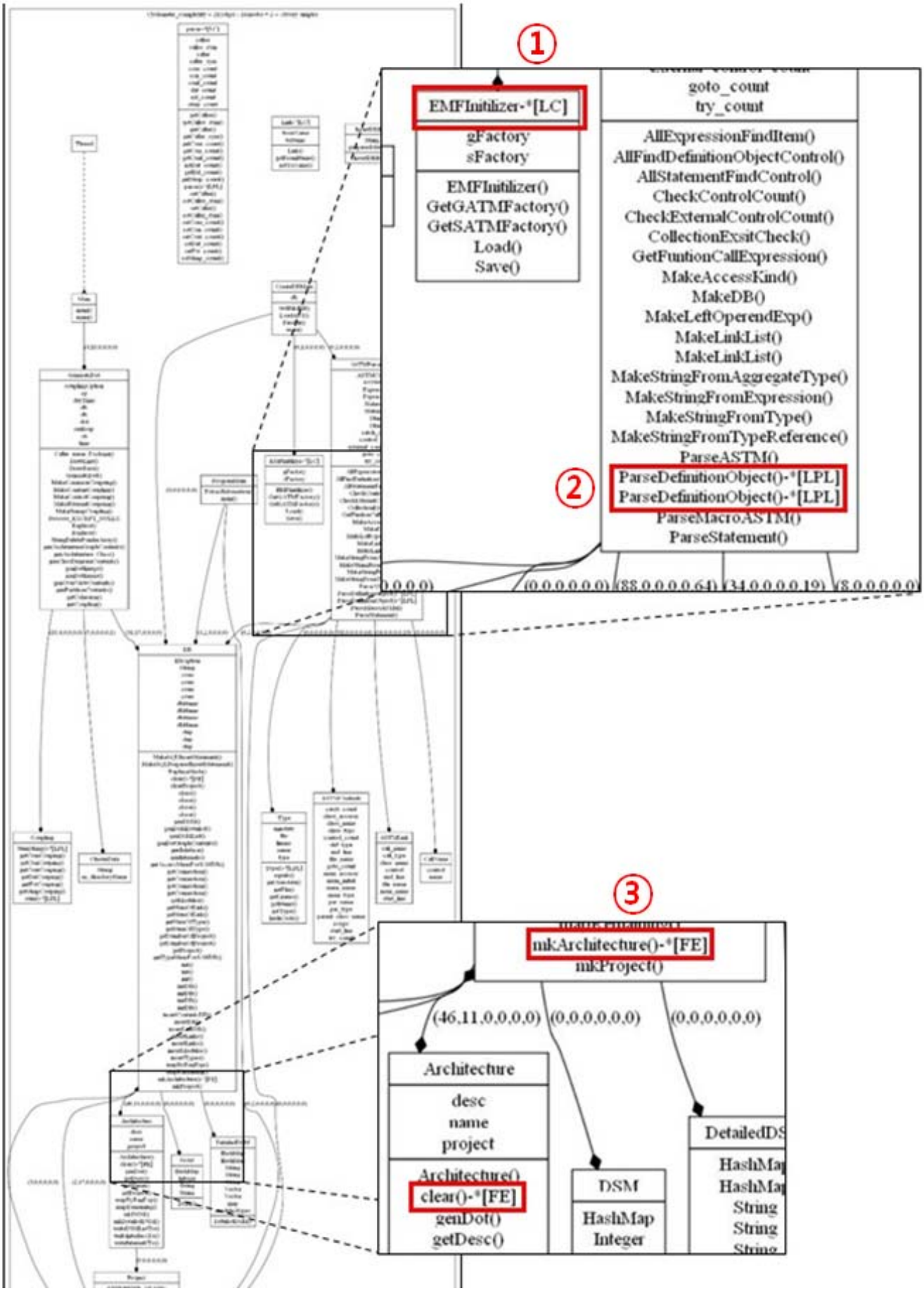



< 그림 1 > 가시화 프로세스 구성도

다음은 추출하는 나쁜 냄새(Bad Smell)코드의 정의이다[4]. 각 조건에 맞게 소스 코드가 작성되어 있으면 가시화된 그래프에 표시를 해준다. 총 5가지의 항목을 조사하였다. 각각의 항목에 대해서 검출조건과 해결방안을 제시한다.

- Long Parameter List (LPL) : Method의 Parameter List가 길다면 이해하기 어렵다. 또한 다른 데이터가 필요할 때마다 계속 고쳐야 할 상황이 발생하기 때문에 Parameter List는 짧은 것이 좋다. Parameter가 5개 이상인 메소드의 이름 뒤에 '-*[LPL]'를 마킹 해놓았다.
- Feature Envy (FE) : 객체지향 관점에서 중요한 한 가지는 데이터와 데이터를 사용하는 프로세스를 하나로 묶는 기술이다. 하지만 메소드가 자신의 클래스보다 다른 클래스에 관심을 가지는 경우가 발생한다. 호출하는 메소드 중에 다른 객체의 get으로 시작하는 메소드가 5개 이상이다. 이런 경우 이름 뒤에 '-*[FE]'를 마킹하였다.
- Lazy Class (LC) : 클래스는 항상 유지하기 위한 비용이 발생한다. 그렇기 때문에 항상 그에 맞는 충분한 일을 해야 한다. 한 번도 사용되지 않는 메소드가 하나라도 존재하는 클래스는 LazyClass라고 판단하였다. 하지만 이런 조건으로 파악할 경우 main 메소드를 호출하는 메소드가 없기 때문에 예외로 main 메소드는 기준에 포함시키지 않았다. 그래프에서는 클래스 이름 뒤에 '-*[LC]'를 마킹하여 가시화하였다.
- Data Class (DC) : 초기 단계에 public으로 각 필드에 대한 get / set 메소드만 가지고 다른 것은 아무것도 없는 클래스를 작성하는 경우가 있다. 후에 이런 클래스는 데이터만 저장하고 거의 대부분 다른 클래스에 의해 조작된다. 클래스이름 뒤에 '-*[DC]'를 마킹하였다.
- Message Chains (MC) : 메시지 체인은 클라이언트가 어떤 객체를 얻기 위해 다른 객체에 물어보고 다른 객체는 또 다른 객체에 물어보고 또 다시 다른 객체에 물어보는 경우를 정의한다. 이런 경우 중간의 어떤 관계가 변한다면 클라이언트 코드도 변경해야하는 결과를 초래한다. 호출당한 메소드가 get*()인데 호출당한 메소드가 다시 다른 객체의 get*()를 호출하는 경우를 메시지 체인이라고 판단하였다. 메소드이름 뒤에 '-*[MC]'를 마킹하여 표시했다.

그림 2는 실제 소스 코드를 이용하여 가시화를 적용한 그래프이다. 예시로 사용된 코드는 학과 게시판 소스 코드이다. 결과를 보면 Long Parameter List와 Lazy Class, Feature Envy가 해당되는 각 클래스와 메소드를 확인할 수 있다. ①번을 보면 EMFInitializer클래스의 Save메소드가 생성되어 있지만 실제 구동 시에는 아무도 호출하지 않는다. 클래스 내부에 사용되지 않는 잉여 코드가 존재하므로 LazyClass를 적용시켜 클래스이름 옆에 [LC]가 표시되었다. ②번을 보면 ParseDefinitionObject메소드의 파라미터




< 그림 2 > 실제 학교 게시판 소스 코드를 이용한 Bad Smell 추출 그래프

개수가 총 6개(String class_type, String class_acceses, String class_name, String parent_class_name, Definition Object defObject, String scope)이다. 개수가 4개를 넘어갔기 때문에 재사용하기 어려운 메소드라 만단하고 [LPL]를 표시하였다. ③번은 mkArchitecture메소드 같은 경우 아키텍처를 만들때마다 sql문을 작성해야 한다. 그 과정에서 아키텍처의 getName, getDecs, getProject 등 총 5번 다른 객체의 get*메소드를 호출한다. 5번 이상 호출하였기 때문에 [FE]를 표시하였다. DataClass와 MessageChains처럼 추출되지 않은 패턴이 많을수록 좋은 품질의 소스 코드라고 할 수 있다. 이처럼 소스 코드를 내부 구조화 시켜 복잡한 소스 코드를 정제할 수 있다.

## 4. 결론 및 향후연구

커져가는 소프트웨어 시장 속에서 품질은 경쟁력이 될 수 있다. 하지만 많은 개발자들이 품질이 중요시되지 않는 구현중심의 개발방법을 사용하고 있다. 본 논문에서 제안하는 가시화 프로세스를 적용한다면 이미 만들어진 소프트웨어의 경우 역 공학기법을 통하여 소스 코드를 가시화 시킬 수 있다. 그로 인해 결합도와 나쁜 냄새(Bad Smell)를 찾아내 소프트웨어의 품질을 향상시킬 수 있다. 또한 제작중인 소프트웨어의 경우에도 개발과 동시에 소스 코드를 가시화할 수 있다. 이를 통해 실시간으로 개발상황을 확인할 수 있고 개발 후의 막대한 유지보수비용을 절감할 수 있다.

현재는 나쁜 냄새(Bad Smell)의 총 22가지 항목 중 5가지만 표현하였지만 점차 나머지 항목들도 패턴을 찾아내어 가시화할 수 있도록 연구할 예정이다. 그 후에도 새로운 품질지표를 찾아내어 소프트웨어의 고품질화를 도모할 계획이다.

## ACKNOWLEDGMENT

이 논문은 2015년 교육부와 한국연구재단의 지역혁신창의인력양성사업의 지원을 받아 수행된 연구임(NRF-2015H1C1A1035548)

## 참고문헌

[1] NIPA, "SW 개발 품질관리 매뉴얼(SW Visualization)", SW 공학센터, pp. 3, 2013.

[2] 조현훈, 최용락, 류성열, “McCabe 및 BP Win도구를 이용한 소프트웨어 역공학 사례연구”, 정보과학회논문지 2000 제 6권 제5호, pp.528-535

[3] 박지훈, 권하은, 강건희, 이근상, 김영철, “코드 가시화를 통한 나쁜 코딩 습관 개선 방안 연구”, 한국인터넷방송통신학회 2015 종합학술대회 논문집 제13권 제1호 pp.47-48

[4] 강건희, 손현승, 김영수, 박용범, 김영철 “SW 가시화 기반 리펙토링 기법 적용을 통한 정적 코드 복잡도 개선”, 한국정보처리학회, 제 21권, 제2호, pp.646-649, 2014

[5] 권하은, 박보경, 이근상, 박용범, 김영수, 김영철, “코드 가시화부터 모델링 추출을 통한 역공학 적용”, 한국정보처리학회, 제21권, 제2호, pp.650-653, 2014

[6] Martin Fowler, "Refactoring", Addison-Wesley, 2002