Jaccard 매트릭스를 이용한 중복 코드패턴 추출 자동화

박지훈[O1], 박영식[2], 김영철[3]
홍익대학교 소프트웨어융합학과 소프트웨어공학연구실
{pjh[1], pys[2], bob[3]}@selab.hongik.ac.kr

Automatic Extraction of Duplicate Code Patterns with Jaccard Metrics

Jihoon Park[O1], Youngsik Park[2], R. Youngchul Kim[3]
SE Lab, Dept. of Software and Communications Engineering, Hongik University

요 약

소프트웨어는 끊임없이 변화할 수 있기 때문에 크기가 커질수록 처음 의도한 디자인대로 구현되기가 쉽지 않다. 그로인해 개발자만이 프로그램에 대한 내용을 알고 있고 새로 들어온 개발자나 다른 사람들은 원본의 디자인만으로 이해하기가 쉽지 않다. 이렇게 레거시 코드가 되어버린 소프트웨어들은 리팩토링을 통하여 복잡도를 줄이고 원래의 디자인을 찾아야 한다. 본 논문에서는 클러스터링을 이용한 리팩토링으로 클래스를 추출하여 중복 코드인 부분을 하나의 클래스로 재사용하는 방안을 제안한다. 중복 코드가 추출된 클래스들은 응집도가 증가하고 재사용된 중복 코드로 인해 런타임시 전체 소프트웨어의 복잡도가 감소한다. 결과적으로 소프트웨어의 디자인을 개선하고 가시화함으로써 소프트웨어의 품질을 높일 수 있을 것으로 기대한다.

1. 서 론

소프트웨어 시스템은 지난 10년 동안 크기가 크게 늘어나면서 끊임없이 변화하고 있다. 이처럼 소프트웨어가 너무 커지면 구조가 처음 의도한 디자인대로 구현되었다는 것을 보장하기가 어려워진다. 그래서 소프트웨어의 개발은 여러 단계의 프로세스를 거쳐서 개발되어야 한다[1]. 하지만 국내 중소기업의 개발자들은 보통 개발시간 및 비용에 대한 압박으로 인하여 중요한 설계 원칙을 놓치고 있다. 이를 해결하기 위해서 역공학을 통한 코드 가시화 방법을 사용할 수 있다[2,3]. 코드 가시화를 통해 소프트웨어의 전체적인 아키텍처를 알아볼 수 있다. 그로인해 리팩토링으로 코드의 복잡도를 개선한다면 원래의 디자인을 찾아갈 수 있다[4,5].

본 논문에서는 Jaccard 매트릭스를 통한 클러스터링으로 리팩토링을 한다. 응집된 클래스를 추출하여 설계를 간결하게 한다. 또한 추출된 클래스 중 중복 코드가 있다면 하나의 클래스로 만들어 런타임시 객체가 중복 생성되지 않도록 한다. 소프트웨어에도 이미 잘 연구된 클러스터 분석 알고리즘을 채택하고 적용할 수 있다면 유용하다[6]. 클러스터링을 통하여 객체지향 프로그래밍에서 클래스를 의미 있는 클래스들로 분해한다. 이를 통하여 분해된 클래스 중에 중복된 클래스가 존재한다면하나의 클래스로 통합하여 다른 클래스들이 재사용할 수 있다.

2장에서는 관련 연구로 객체지향에 사용되는 클러스터링에 대해 알아본다. 3장에서는 클래스 추출을 통한 중복 객체 추출 결과를 설명한다. 4장에서는 결론 및 향후 연구이다.

2. 관련 연구

2.1 객체지향에서의 클러스터링

일반적으로 소프트웨어를 이해하고 설계를 복구하기 위해서는 지식 기반 접근을 해야 한다. 지식 기반 접근으로는 기존 지식을 사용하여 소스 코드로부터 이해할 수 있다. 하지만 대형 시스템은 소스 코드 전체에서 파악하기 힘들 만큼의 지식 집합이기 때문에 적합하지 않다[7]. 구조 기반 접근으로 소프트웨어를 클러스터링 할 수 있다. 구조 기반으로 접근하기 위해서는 소스 코드를 클러스터링 엔티티로 지정해야한다[8]. 지정된 엔티티를 사용해서 클러스터링을 할 수 있다.

2.2 Jaccard 매트릭스

본 논문에서 사용하는 Jaccard 매트릭스는 소프트웨어 재구성에서 좋은 결과를 나타낸다[10]. Jaccard 매트릭스는 다음과 같은 수식을 사용한다.

$$JM_{a,b} = 1 - \frac{|a \cap b|}{|a \cup b|} \quad (1)$$

a와 b의 합집합에서 교집합을 제외한 값이 Jaccard 매트릭스의 거리이다. 이 수식을 사용하기 위해선 a와 b의 엔티티를 정의하여 집합을 만들어야한다. 소프트웨어에서 엔티티의 정의는 다음과 같다. 속성은 속성에 엑세스하는 모든 메서드를 엔티티로 갖는다. 메서드는 자기 자신과 자신이 엑세스하는 속성들을 엔티티로 갖는다. 이 경우 Jaccard 거리가 0이라면 a와 b는 같다고 볼 수 있다[9].

3. 중복 코드를 재사용 클래스로 통합

3.1 소스 코드에서 엔티티 추출

그림 1은 엔티티를 추출하기 위한 샘플 코드이다. 그

```java
public class School {
    private String name;
    private int classNumber;
    private String workspacePath;
    private Connection conn;

    public void setPath(String workspacePath) {
        this.workspacePath = workspacePath;
    }
    public String allocateStudent(String newName, int number) throws SQLException {
        ResultSet rs = conn.createStatement().executeQuery("select " + newName + "from TABLE");
        while (rs.next()) {
            if (rs.getString(newName).equals(name)) {
                classNumber = number;
            }
        }
        return name + ", " + classNumber;
    }
    public void getConnection() {
        Connection conn = null;
        try {
            Class.forName("org.sqlite.JDBC").newInstance();
            conn = DriverManager.getConnection("jdbc:sqlite:/" + workspacePath);
        } catch (Exception ex) {
        }
        this.conn = conn;
    }
}
```

```java
public class Company {
    private String employee;
    private String workspacePath;
    private Connection conn;

    public void setPath(String workspacePath, String dbName) {
        this.workspacePath = workspacePath;
    }
    public void changeDepartment(String name, String department) throws SQLException {
        ResultSet rs = conn.createStatement().executeQuery("select " + name + "from TABLE");
        while (rs.next()) {
            conn.createStatement().executeUpdate(
                "update TABLE set department=" + department + "where name=" + employee);
        }
    }
    public Connection getConnection() {
        Connection conn = null;
        try {
            Class.forName("org.sqlite.JDBC").newInstance();
            conn = DriverManager.getConnection("jdbc:sqlite:/" + workspacePath);
        } catch (Exception ex) {
        }
        return conn;
    }
}
```

그림 1 클러스터링을 위한 샘플 중복 코드

림 1을 보면 School과 Company 클래스가 존재한다. 두 클래스 모두 학생과 부서의 정보를 얻기 위해 데이터베이스 접속이 필요하다. 이럴 경우 개발자는 데이터베이스 접속 메서드를 그대로 복사해서 사용하는 경우가 생긴다. 결국 런타임시 getConnection()가 불필요하게 두 번 실행된다. 이를 해결하기 위해 클래스로 추출하여 효율성을 높일 수 있다. School은 4개의 변수 3개의 메서드를 가지고 있고, Company는 2개의 변수 3개의 메서드를 가지고 있다. 정리해보면 다음과 같이 엔티티를 구성할 수 있다. 추출된 엔티티를 이용하여 각 변수와 메서드 간의 응집도를 Jaccard 거리 매트릭스로 구분할 수 있다.

- sa1 - name
- sa2 - classNumber
- sa3 - workspacePath
- sa4 - conn
- sm1 - setPath
- sm2 - allocateNumber
- sm3 - getConnection

- ca1 - employee
- ca2 - workspacePath
- ca3 - conn
- cm1 - setPath
- cm2 - changeDepartment
- cm3 - getConnection

3.2 Jaccard 매트릭스를 통한 Jaccard 거리 생성

표 1 School 클래스의 Jaccard 거리 매트릭스

	sa1	sa2	sa3	sa4	sm1	sm2
sa2	*0.67*					
sa3	*1*	*1*				
sa4	*0.8*	*0.8*	*0.6*			
sm1	*1*	*1*	*0.33*	*0.8*		
sm2	*0.5*	*0.5*	*1*	*0.67*	*1*	
sm3	*1*	*1*	*0.5*	*0.6*	*0.75*	*0.83*

표 2 Company 클래스의 Jaccard 거리 매트릭스

	ca1	ca2	cm1	cm2
ca2	*0.8*			
cm1	*0.33*	*1*		
cm2	*1*	*0.33*	*1*	
cm3	*0.5*	*0.5*	*0.75*	*0.75*

표 1과 2는 Jaccard 거리 매트릭스를 통한 수치 값이다. 표 안의 값들은 수식 1을 이용하여 계산한 값이다. 다음 값들을 이용하여 Dendrogram을 작성하면 임계값을 측정할 수 있다. 측정된 임계값은 추출 클래스의 범위이다.

3.3 Dendrogram을 통한 임계값 측정

그림 2는 표 1과 2의 거리 매트릭스 값을 이용하여 Dendrogram을 그린 것이다. 클러스터링을 통해 클래스를 추출하기 위해서는 어느 정도의 기능을 가지고 있는 클래스를 식별해야 한다. 그래야 리팩토링을 했을 때 문제

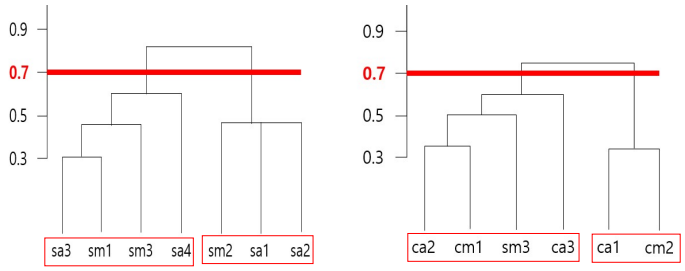

그림 2 School과 Company의 Dendrogram

가 없다. 조건은 적어도 2개 이상의 엔티티가 포함되어야 하고, 1개 이상의 메서드가 포함되어야 한다[9]. 두 클래스 모두 임계값을 0.7로 하면 위의 조건을 만족하는 클러스터링이 성립된다.

3.4 클러스터링을 통한 중복 코드 통합

그림 3은 클러스터링을 하였을 경우 중복코드가 통합된 그래프이다. 임계값을 통해 클래스를 추출하면 Student와 Company 클래스는 2개의 클래스로 나눠진다. 하지만 나눠진 클래스 안의 엔티티 노드들의 관계를 보면 중복 코드가 있기 때문에 하나로 통합된다.

4. 결론 및 향후 연구

본 논문에서는 클러스터링을 통해 클래스를 추출하고 추출된 클래스 중 중복코드를 통합하는 방법을 제안한다. 이 방법을 이용하면 클래스의 응집도를 높일 수 있

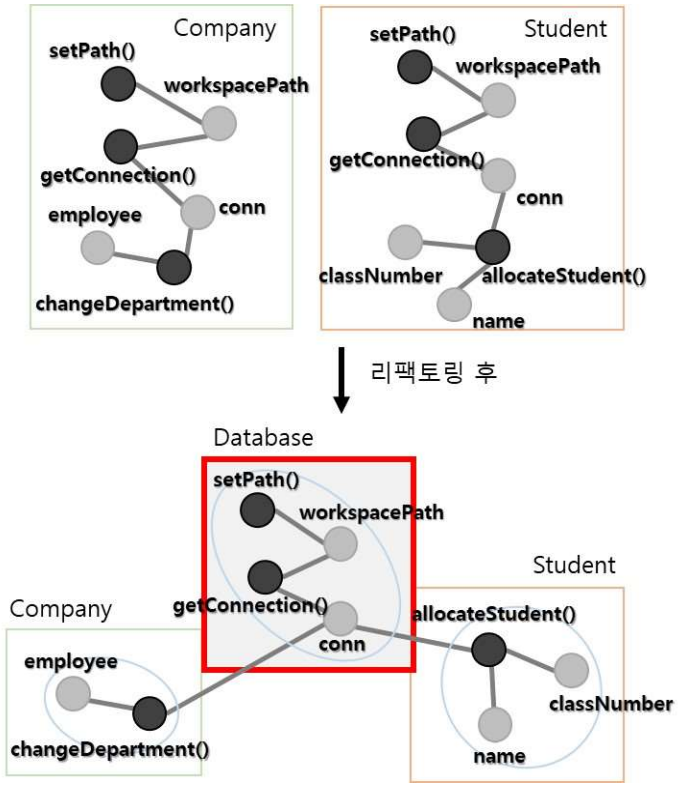

그림 3 클러스터링을 통한 중복 코드 추출

다. 또한 기존에는 중복된 코드만 찾아서 리팩토링할 경우 응집도가 낮아지고 결합도가 높아질 수 있었다. 하지만 이 방법을 사용하면 응집도 높은 클래스를 추출하기 때문에 퍼포먼스와 설계 모두 효율성을 높일 수 있다.

하지만 완전한 중복 코드가 아니라면 클러스터링에 문제가 있을 수 있다. 따라서 본 연구는 copy&paste기능에 의존한 개발자의 코드에 적합하다고 볼 수 있다. 향후 연구에서는 메서드 추출을 접목하여 더욱 응집한 클래스를 추출할 것이다. 그로인해 완전한 중복 코드가 아니라도 추출을 통하여 통합 객체로 생성할 수 있도록 연구할 것이다.

ACKNOWLEDGEMENT

이 논문은 2017년도 정부(교육부) 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업(NRF-2017R1D1A3B03035421)과 2018년도 정보통신산업진흥원의 정보통신, 방송 연구개발사업(개방형OS 환경개발 및 보급, 확산 사업)의 지원을 받아 수행된 연구임(S1113-18-1001)

참 고 문 헌

[1] NIPA SW공학센터, "SW개발 품질관리 매뉴얼(SW Visualization)", 2013.

[2] 박지훈, 박보경, 이근상, 김영철, "기존 자바파서 확장 기반의 코드 정적 분석기 구현", 한국정보처리학회, 제24권 제1호, pp.641-644, 2017.

[3] 박지훈, 김영철, "객체지향 내의 절차식 코드 스타일을 리팩토링하기 위한 가시화 도구 구현", ICT플랫폼학회, Vol. 5 No.3, pp.28-32, 2017.

[4] B. Boehm, H. D. Rombach, M. V.Zelkowitz, "Foundations of Empirical Software Engineering", Springer, pp.426-427, 2005.

[5] Martin Fowler, "Refactoring: Improving The Design of Existing Code", Addison-Wesley, 2002.

[6] T. A. Wiggerts, "Using clustering algorithms in legacy systems remodularization", Proceedings of the Fourth Working Conference on Reverse Engineering IEEE Computer Society Press, pp33-43, 1997.

[7] V. Tzerpos, R. C. Holt, "Software Botryology Automatic Clustering of Software Systems", Proceedings of the International Workshop on Large-Scale Software Composition, 1998.

[8] M. Fokaefs, N. Tsantalis, A. Chatzigeorgiou, J. Sander, "Decomposing Object-Oriented Class Modules using an Agglomerative Clustering Technique", IEEE International Conference on Software Maintenance, 2009.

[9] N. Tsantalis and A. Chatzigeorgiou, "Identification of Move Method Refactoring Opportunities", IEEE Treansations on Software Engineering, 35(3), pp347-367, 2009.

[10] O. Maqbool, H. A. Babri, "The Weighted Combined Algorithm: A Linkage Algorithm for Software Clustering", Proceedings of the 8th European Conference on Software Maintenance and Reengineering IEEE, pp15-24, 2004.