코드 재사용을 위한 기능 단위 모듈의 유사도 분석 기법 연구

나학연°, 이종호, 류성열
숭실대학교 컴퓨터학과, 대신정보통신
hacyunna@selab.soongsil.ac.kr, jhlee@dsic.co.kr,
syrhew@computing.soongsil.ac.kr

A Study on Similarity Analysis of Function Unit Module for Reusing Code

Hac Yun Na°, Jong Ho Lee, Sung Yul Rhew
Dept. of Computing, SoongSil University, DaiShin Information & Communications

요 약

소프트웨어 재사용은 이전의 개발 경험을 새로운 소프트웨어 개발 과정에서 재적용 하는 것으로, 소프트웨어 개발 환경 및 관리 과정에서 생산성 향상에 기여할 수 있다. 본 논문에서는 객체지향 언어로 개발된 프로그램에서, 하나의 클래스에 있는 여러 메소드들 안에 중복된 코드가 있는 경우 이러한 중복된 코드의 검출을 통해 재사용의 근거로 제시하고자 한다. 그 방법으로 McCabe 의 유사도 분석 기법을 이용하였고, 그 과정에서 나타난 문제점을 해결하고자, 새로운 검색 요소로 구성된 최적의 유사도 분석 기법을 제안하였다. 그리고 분석 결과를 재사용하기위한 문서화 작업의 기준도 마련하였다. 기준에 맞게 작성된 문서들은 코드 수준의 재사용 가능 라이브러리로 저장하여 다음 개발에 직접 적용한다면, 비용 및 시간을 절약하는 효과를 기대할 수 있다.

1. 서 론

소프트웨어 재사용은 이전의 개발 경험을 새로운 소프트웨어 개발 과정에 재적용 하는 것으로 소프트웨어 개발 환경 및 관리 과정에서 생산성 향상에 기여할 수 있다.[1]코드 수준의 재사용은 라이브러리가 새로운 개발에 직접 적용할 수 있는 풍부한 코드들로 구성되어 있는 경우 큰 효과를 기대할 수 있다. 대규모의 프로그램에서는 재사용 가능한 코드가 존재할 수 있다. 다른 표현이지만, 일반적으로 프로그램의 크기가 클수록, 복잡해질수록 비슷한 기능을 하는 코드의 중복이 생길 수 있다. 이런 중복된 코드는 부분적으로 유사한 기능을 갖는 형태일 수 있으며, 전체적으로 같은 형태일 수도 있다. 과거에 작성된 프로그램의 경우 그 당시 개발에 참여했던 개발자들이 현재 존재하지않거나, 문서화가 잘 되어 있지 않다면 유지보수하기가 매우 어렵다. 그리고 대규모의 프로그램은 단독으로 작성되기는 힘들고, 보통 공동으로 개발되기 때문에 중복된 기능 단위 모듈이 생길 수 있다.[2]특히 객체지향 프로그램언어로 작성된 프로그램에서 코드의 중복은 비효율적일 수 있다. 예를 들어 객체지향 프로그램 언어인 C++은 상속(inheritance)이라는 기능을 제공하고 있는데, 이렇게 프로그램 언어 자체에서 제공하는 기능을 사용하지 않고, 유사한 기능을 하는 모듈들을 중복으로 작성한다면, 프로그램의 무게가 늘어날 뿐만 아니라 효율면에서도 떨어질 것이 분명하다.[5] 본 논문에서는 이러한 유사 코드 재사용을 위한 필수적인 유사도 분석 기법을 제시한다.

2. 관련 연구

2.1. Duplicated Code

만약 여러 군데에서 같은 코드 구조를 발견한다면, 그래서 그러한 것들을 통합하는 방법을 찾아낸다면, 프로그램의 효율을 더욱더 높일 수 있을 것이다. 중복된 코드가 발생하는 가장 간단한 문제는 같은 클래스의 두개의 메소드에서 똑 같은 표현이 있을 때 이다. 이 문제를 해결하는 방법은 일단 메소드 추출(Extract Method)을 하고 같은 곳에서 코드를 호출하게끔 한다. 또 다른 중복된 코드의 문제는 두개의 형제 서브 클래스에서 똑 같은 표현이 있을 때 이다. 마찬가지로 메소드 추출 방법을 사용해서 중복을 제거하고, 상위 클래스로 올리는 과정(Pull Up Field)을 수행한다. 만약 그 코드가 비슷하기 만하고 똑 같지 않다면, 다른 비트로부터 비슷한 비

트를 분리하기 위해 메소드 추출 방법을 사용해야 한다. 검출된 메소드들은 유형별 메소드(Template Method)로 사용될 수 있다. 만약 그 메소드가 다른 알고리즘을 갖고 같은 일을 한다면, 두개의 알고리즘에 대해 좀더 명확한 개념을 갖을 수 있으며, 이러한 알고리즘을 치환해서 사용할 수 있다.[6]

2.2. McCabe Tool

역 공학 도구들은 유지보수 행위 동안 복잡한 소프트웨어 시스템들을 분석하고 이해하는 과정에 있어서 소프트웨어 엔지니어들을 지원한다. 그러한 도구들의 기능은 편집 및 브라우징(browsing) 기능들로부터 텍스트 및 그래픽 보고서들까지 다양하다. 역 공학을 지원하는 제품으로 McCabe 가 있는데, 이는 McCabe & Associates, Inc 사에서 개발한 소프트웨어 역공학 도구로써 기존의 소스 코드를 분석하는 역공학 및 코드에 대한 시험을 수행하고, 유사 모듈을 검색하는 도구로써 분석 결과를 다양한 그래픽 및 텍스트 형태로 제공한다. 이를 활용하면 소스 코드의 구조와 흐름 및 복잡도, 유사도 등을 명확히 살펴 볼 수 있는 자료를 얻을 수 있다.[3,4]

3. 유사도 분석 기법

3.1. 기존 McCabe 의 유사도 분석 기법

McCabe 의 유사 모듈 검색에서 제공되는 분석 기법의 단점은 다음과 같다.[7]

- McCabe Metrics Comparison : 단순히 McCabe 메트릭인 복잡도만을 비교하기 때문에 복잡도가 같다고 해서 소스 코드가 일치하지는 않다.
- Graph Comparison : 유사도 100%이하는 모두 일치하지 않는 것으로 나오기 때문에 비슷한 기능에 대한 검색은 불가능하다.
- Control Structure Comparison : Graph Comparison 과 거의 유사하다.
- General Comparison : 특성 중 nl (number of lines)의 가중치가 60%로 다른 특성보다 높게 책정되어있어 모듈 내 주석의 유무, 줄넘김 문자의 유무에 따라 유사도의 차이가 크다.

이런 이유로 최적의 유사도 측정을 위한 접근이 필요한데 그 내용은 다음 절에서 설명하기로 한다.

3.2. 중복 코드 검출을 위한 유사도 분석 기법 제안

- 모듈 비교에서 유사도의 필요충분 조건
  - 프로그램안에 재사용 가능한 코드 또는 중복 코드를 발견해야 한다.
  - 한 모듈의 두 가지 버전의 유사함을 검증할 수 있어야 한다.
  - 수정된 모듈이 원본 모듈과 유사함을 검증할 수 있어야 한다.
- 모듈 비교에서 검색 요인의 기준으로 고려해야 할 특성
  - metrics
  - data elements
  - parameters
  - module names and types
  - control structures
  - flowgraphs

이들 특성들은 모듈의 속성을 구분 짓는 요소로써 이는 유사도 측정을 위한 지표가 된다. 위의 모듈의 특성을 만족시키는 검색 요인의 조합이 필요한데, 그 것들은 표 1 에 잘 나타나 있다.

표 1. Optimal Compare 의 요소

Element	Description
Actual_parameter_type	호출된 모듈에 전달되는 파라메터의 데이터 형식
Branch	주어진 모듈 안에서 발생한 분기의 수
Call_pair	다른 모듈에 대한 호출의 수
Called_module_names	호출된 모듈의 이름
Called_module-type	호출된 모듈의 형식
Code	단지 코드와 여백을 포함하고 있는 코드의 소스 라인
Control_structure	그래프의 제어 구조
Dec	모듈에 있는 불리언 결정(참, 거짓)에 대한 결과의 수
Formal_parameter_type	함수로 전달되는 정형 파라메터의 형식
Global_data_element_count	모듈에서 사용되는 범용 데이터 요소의 수

3.3. 분석 결과 및 평가

유사도 분석을 통해 얻은 분석 결과는 크게 두 가지로 분류할 수 있는데, 이는 문서화의 기준으로 이를 통한 문서들의 라이브러리화는 재사용에 직접적으로 적용 가능하다.

- 형태적 요소

형태적 요소에서는 파라메터의 개수, 파라메터 유형, 주석, 함수명, 지역변수, 리턴값 등 소스 코드의 외형적인 일치를 나타내고 있다.

- 기능적 요소

기능적 요소에서는 함수명, 지역변수는 다르나 사용된 로직의 알고리즘의 유사로 인한 내부적인 일치를 나타내고 있다.

결과 산출물은 재사용을 하기엔 가공할 요소가 많다. 이러한 부분을 형태적 요소와 기능적 요소로 구분하여 문서화를 한다면 추후 재사용에 용이할 것이다.

- 평가

평가를 위해서 공통 모듈을 사용했다. 이 모듈은 General Compare 로 시행했을 때 보여준 결과 중 함수명만 틀리고 기능은 갖은 모듈을 선택하여, 비교하기 위해 선택되었다. 비교를 위해 선택된 모듈은 그림 1 과 같다.

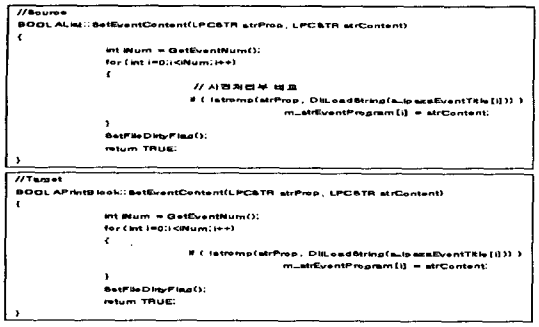

그림 1. 공통 비교 모듈

General Compare 로 유사도 분석을 시행한 경우는 유사도 측정치가 75%로 평가됐다. 그런 오차가 생긴 가장 큰 이유 중 하나는 소스코드 라인에 비중을 많이 두었기 때문이다. 먼저 소스쪽의 라인 수는 주석 포함해서 12 줄이고, 타켓쪽은 10 줄이다. 다음으로 함수명의 차이를 둘 수 있다. 하지만 내부의 로직은 유사하므로 75%의 유사도가 나온 것이다. 보통 McCabe 에서 75%이상은 유사한 것으로 취급한다. Optimal Compare 로 유사도 분석을 시행한 경우는 실제 가중치를 모든 검색 요인에 동등하게 주고 시행했기 때문에 어떤 특정 요인의 영향을 덜 받게 하였다. 그래서 주석을 제거한 실제 소스 라인만의 비교로 인해 함수명의 차이를 보인 95%의 결과치를 보여주었다. 그리고 소스코드의 실질적인 비교가 가능하여, 전체 유사도 측정치를 높였다.

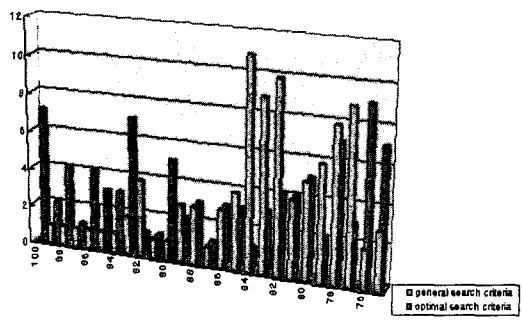

그림 2. 비교 분석 차트

위의 그림 2 에서 보는 것과 같이 General Compare 의 결과 중 가장 많이 몰려 있는 곳이 77~84%이다. 이런 수치가 나온 이유는 소스코드 라인의 비교에서 주석의 유무와 프로그래머의 임의로 넘긴 줄이 실제 비교에 들어가 전체 유사도에 영향을 주었기 때문이다. 그러나 Optimal Compare 에서는 그러한 요소를 제거하고 형태적 요소와 기능적 요소의 적절한 사용과 특정 요소의 가중치를 높이지 않고 동등하게 했기 때문에 라인 수만 같을 때 높았던 유사도가 적절한 측정값으로 나타남을 보여주고 있다. 가장 큰 차이는 소스코드에 충실한 비교를 통해 정확도를 높였다.

4. 결론 및 향후 연구

본 논문에서는 소프트웨어 재사용 과정에서 필요한 유사도 분석의 기법을 설명하고 있다. 기존의 작성된 프로그램에서 중복된 코드의 제거로 인한 프로그램의 성능 향상도 코드 재사용을 위한 환경이 될 수 있다. 이런 중복된 코드를 일일이 눈으로 찾는다는 것은 어렵고, 많은 시간이 요구되는 일이다. 개발 기간의 속도에 따라 프로젝트의 성패 여부가 결정되는 이때, 소프트웨어 유지보수 도구의 사용은 당연한 것이다. 본 논문에서도 유지보수를 위해 McCabe 를 사용하였다. McCabe 에서 제공되는 유사도 분석을 위한 정확한 사전 지식이 없을 때 측정된 유사도의 신뢰도는 떨어질 것이다. 그러므로 본 논문은 코드수준의 유사도 분석을 위한 검색 기준의 정립을 통해 이후 소프트웨어 유지보수 작업에 도움을 주고자 함이다. 검색 기준을 통해 얻어진 산출물은 동일한 모듈 이름과 기능을 하는 형태적 요소와 다른 모듈 이름이지만 비슷한 기능을 하는 기능적 요소로 분류할 수 있다. 추후 분석된 자료를 가공할 수 있는 도구의 개발을 통해 재사용을 위한 문서화가 필요하다.

5. 참고 문헌

[1] R.Alan Whitehurst, " *Systemic Software Reuse Through Analogical Reasoning,*" Ph.D. thesis, University of Illinois at Urbana-Champaign, 1995.

[2] Rugaber, S., " *White Paper on Reverse Engineering,*" Technical Report, Georgia Institute of Technology, March, 1994.

[3] Berndt Bellay and Harald Gall, " *An Evaluation of Reverse Engineering Tools,*" Technique University of Vienna Information Systems Institute Distributed Systems Department, March 13, 1997.

[4] Berndt Bellay and Harald Gall, " *A Comparison of four Reverse Engineering Tool,*" Technique University of Vienna Information Systems Institute Distributed Systems Department, September 19, 1997.

[5] Stephen R.Schach, " *Classical and Object-Oriented Software Engineering with UML and C++,*" Fourth Edition, Vanderbilt University, 1998, Pages 171-175.

[6] Martin Fowler, " *Refactoring: Improving the Design of Existing Code,*" OOPSLA, 1999.

[7] McCabe Associates, " *Using the Visual Reengineering ToolSet,*" June 1998