컴포넌트 의존성 추적을 통한 최적 테스트케이스 추출에 관한 연구

천승민[O] 송영재
경희대학교 전자계산공학과
smcheon@orgio.net yjsong@khu.ac.kr

A study on most suitable test case abstraction through component relativity chase

Seung-Min Cheon[O] Young-Jae Song
Dept. of Computer Engineering, KyungHee University

요 약

현재 소프트웨어 개발에서 증가적으로 채택되고 있는 컴포넌트 기반 소프트웨어 엔지니어링(CBSE)에서, 새로운 프레임 웍에서의 컴포넌트의 효율적인 동작을 위한 컴포넌트 변형이나, 컴포넌트 통합은 활발히 연구가 진행중이나, 변형이나 통합으로 인하여 발생 할 수 있는 오류들을 테스팅하는 방법이과, 툴은 부족한 현실이다. 본 논문에서는 컴포넌트의 통합, 변경시 발생할수 있는 에러들을 쉽게 검출하기 위하여 본 논문에서 제안한 CCG(Component Calling Graph)를 이용하여 컴포넌트간의 의존성을 추적하고 selection 알고리즘을 거친 테스트 케이스 추출을 통한 컴포넌트의 효율적인 테스팅 방법을 제안하였다.

1. 서론

컴포넌트 개발과, 컴포넌트 통합의 두 가지 단계를 가지고 있는 컴포넌트 기반 소프트웨어 엔지니어링(CBSE)의 애플리케이션 개발을 위한 컴포넌트 기반 소프트웨어 프로세스 모델은 컴포넌트의 분석과 컴포넌트 획득, 컴포넌트의 합성, 단위 테스트, 통합 테스트, 시스템 테스트의 과정을 거친다. 이러한 컴포넌트 기반 소프트웨어가 다른 플랫폼에서 원활한 동작을 하기 위해 컴포넌트에 변형이 가해지거나, 새로운 기능을 갖기 위하여 컴포넌트 통합이 이루어 졌을 경우에, 변형되거나 통합된 새로운 컴포넌트는 새로운 테스트 과정을 거쳐야 한다. 일반적인 소프트웨어 개발에서의 Testing 레벨은 3가지로 구분될 수 있는데, 첫째는 단위 테스팅(Unit Testing), 둘째는 통합 테스팅(Integration Testing), 셋째는 시스템 테스팅(System Testing)이다. 단위 테스팅은 코드 개발자와 프로젝트 개발팀의 멤버가 수행하는데, 여기서는 작성된 함수나 서브루틴 등의 각각의 Unit들을 테스트하게 된다. 단위 테스팅의 성공 조건은 각각의 개별 Case들의 올바른 선택과 이에 대한 코드의 존재 여부에 대해서 시험을 하게 되는데, 컴포넌트 기반 개발에서는 이러한 Unit Testing을 하지 않고 바로 통합 테스트를 하게 된다.

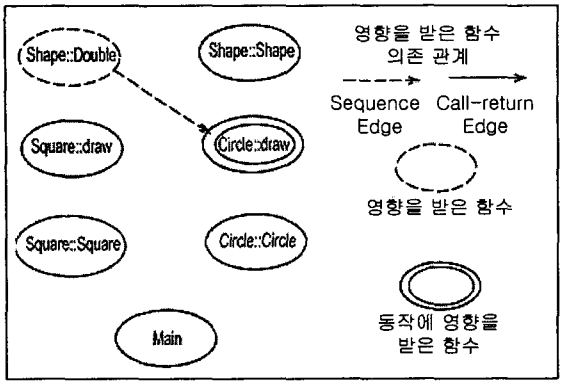
컴포넌트 테스팅의 기존연구[1,2,3]중, [1]은 컴포넌트의 테스팅을 위하여 발생할 수 있을만한 오류를 삽입하는 오류 삽입방법을 이용함으로서 에러를 체크하는 방법을 제시하고 있으며, [2]는 컴포넌트 내의 반복문의 횟수를 사용자가 True의 T와 False의 F의 조합을 입력하여 체크해 봄으로써 테스팅을 수행한다. 마지막으로, [3]은 뮤테이션 기법을 이용하여 컴포넌트의 테스팅을 수행하는 방법을 제시하고 있는데, 여기서 뮤테이션 기법이란 테스트 하려는 프로그램 P가 있을 때 이를 약간 변형한 뮤턴트 프로그램 P'을 만들어 P와 P'을 차별화 할 수 있는 테스트 데이터를 선택하는 기법이다.

컴포넌트를 테스트하기 위한 기존의 컴포넌트 테스트 방법들은 컴포넌트에 수정을 가하거나 조립을 한 개발자가 컴포넌트의 의존성을 체크하여 재테스팅을 한다면 불필요한 테스트 케이스의 생성을 방지할 수 있고, 이로 인하여 컴포넌트 테스팅에 들어가는 시간과 비용을 줄일 수 있을 것이다.

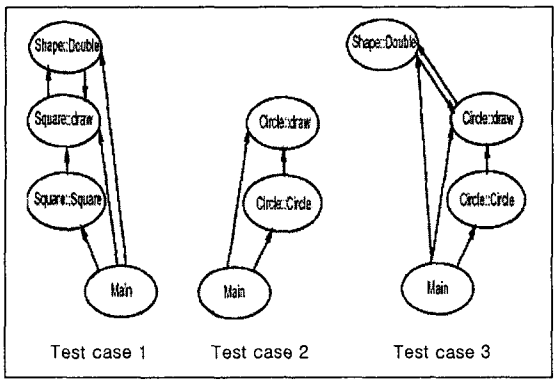
본 논문에서는 수정된 컴포넌트와 영향을 받은 컴포넌트들을 식별하기 위해 컴포넌트들의 실행 순서를 추적하는 CCG(Component Calling Graph)를 제안하고, 발생할 수 있는 오류들을 Graph를 이용해 추적하였다. 이러한 정보와, 기존의 테스트 케이스 추출알고리즘을 이용하여, 수정된 프로그램을 테스트하기 위한 새로운 test case들을 발생시킴으로써 효율적으로 컴포넌트를 테스팅 할 수 있을 것이다.

2. 관련연구

기존 연구로는 객체 지향 프로그램에서 함수들의 의존관계 및 함수의 호출 관계를 도식화한 연구가 있었다[4]. 이 연구에서는 영향을 받은 변수들과, 영향을 받은 함수들을 식별한 후, [그림 1]과 같이 함수들과의 의존관계를 그래프로 나타내고 [그림 2]는 각각의 테스트 케이스에 대한 FCG(Function Calling Graph)를 보여준다.



[그림 1] 영향을 받은 함수 의존 그래프



[그림 2] 3가지 테스트 케이스들에 대한 FCG

이렇게 작성된 FCG로 인해서 변수들과 함수들 그리고 함수들의 종속관계 및 함수들 사이의 의존관계에 기반을 둔 수정이 일어날 때 각 테스트 케이스에 대한 함수들의 실행순서 추적을 용이하게 하고, 이 논문에서 제시한 알고리즘을 이용하여 수정된 프로그램에서의 테스트케이스 추출 시 불필요한 테스트 케이스를 제외한 테스트 케이스를 추출함으로서 프로그램의 테스트를 효율적으로 수행할 수 있도록 하였다.

본 논문에서는 컴포넌트내부의 함수 및 변수들의 의존성과 컴포넌트간의 의존성을 본 논문에서 제시하는 CCG(Component Calling Graph)를 UML을 이용하여 작성함으로서 이해도를 높이고, 컴포넌트의 테스트 케이스 추출을 위한 의존성의 추적을 용이하게 할 수 있을 것이다.

3. 컴포넌트의 의존성 추적

컴포넌트의 의존성을 추적하기 전에 소스코드의 내용을 소스코드의 내용을 볼 수 있는 White-box 컴포넌트의 경우와 소스코드의 내용을 볼 수 없는 Black-box 컴포넌트의 경우를 나누어서 살펴보아야 할 것이다.

White-box 컴포넌트의 경우 소스코드의 내용이 확인 가능하므로 아래의 CCG를 쉽게 적용할 수 있지만 소스코드의 내용을 볼 수 없는 Black-box컴포넌트의 경우에는 컴포넌트간의 연결을 위한 인터페이스를 이용하여 의존성을 추적할 수 있다.

3.1 영향을 받은 컴포넌트 종속 관계

컴포넌트의 의존성을 추적하기 위한 첫 단계는 컴포넌트 내부의 영향을 받은 변수들, 함수들, 그리고 컴포넌트 종속 관계들을 식별하는 것이다. 이것을 수행하기 위해서 우리는 수정 작업에 의해 영향을 받은 변수들을 우선 찾는다.

영향을 받은 변수 :

- 변수 x의 값을 계산하기 위하여 사용된 계산식 CE가 컴포넌트 내의 함수 f에서 수정되었다면 이 변수는 직접적으로 영향을 받은 것이다.
- 컴포넌트 내의 함수 f의 변수 x가 변수 y를 사용하고, 변수 y를 다른 함수 f'에서 사용을 할 때, 영향을 받았다고 한다.
- 변수 y의 값이 함수의 파라미터의 값을 계산하기 위해 사용되고, 함수의 반환 값이 변수 x의 값을 계산하기 위해 사용되었다.

영향을 받은 함수 :

컴포넌트 내부의 함수 f가 f에서 영향을 받은 변수 x와 y를 사용하였다면 영향을 받은 것으로 여겨진다. 수정된 후의 컴포넌트의 출력이 수정되기 전과 같다면 그 test case를 재 테스트할 필요가 없다.

영향을 받은 함수의존 관계 :

만약 아래 조건을 만족한다면, 컴포넌트 내부의 함수 f1과 f2사이의 함수 종속 관계는 영향을 받은 것으로 간주한다.

- 컴포넌트 내의 함수 f1이 f2에서 정의된 변수 x를 사용하고, x가 f2에서 영향을 받는다.

3.2 수정된 컴포넌트의 오류

예를 들어 컴포넌트 C1에 "i = 0"을 추가한 후에, 우리는 그 컴포넌트에서 어떠한 문제도 찾을 수 없지만, 만약 컴포넌트 C1이, 컴포넌트 C2, C3와 함께 배치된다면, 치명적인 오류가 발생할 것이다.

<table>
<tr><td rowspan="2">C1
I1 -> i = 0;
I2 -> return i;</td><td>C2
I3 -> j = 10</td></tr>
<tr><td>C3
I4 -> k = C2::I3() / C1::I2();</td></tr>
</table>

[그림 3]. 수정된 컴포넌트

이때 발생할 수 있는 컴포넌트의 오류를 CCG를 이용하여 의존 관계를 추적하면, 오류가 발생하는 컴포넌트와 연관된 컴포넌트의 오류를 찾기가 용이해질 것이다.

3.3 Component Calling Graph(CCG)

CCG는 컴포넌트들의 실행순서를 간단하게 기록하는 것이다. 아래에서 CCG를 사용한 접근을 나타내고 그것의 장점을 보인다.

CCG는 방향성 그래프이다($G_{CCG} = (V_{CCG}, E_{CCG})$). 각 노드 $v_i \in V_{ccg}$는 컴포넌트에서 하나의 함수이다.

그리고 $e_{ij} = (v_i, v_j) \in E_{CCG}$ 는 v_j가 실행되기 전에 실행하는 v_i인, 두 함수 v_i와 v_j를 나타낸다.

아래의 조건들 중 하나라면 vi는 vj 전에 실행한다.

1. 컴포넌트 내의 함수 vi가 다른 컴포넌트 안의 함수 vj를 호출한다면 vi는 함수 vj에 의해 호출된다. (call-return edge in G_{CCG}).
2. vi와 vj가 같은 컴포넌트 안에서의 함수이고, vi가 vj전에 실행하고, 그 컴포넌트 내부에서 vi와 vj사이에 실행되는 다른 함수는 없다

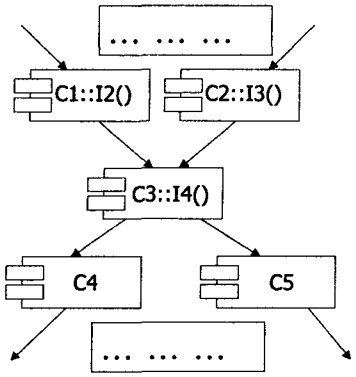


그림 4. Component Calling Graph(CCG)

[그림 4]의 CCG로 컴포넌트 C3가 영향을 받는 컴포넌트들의 추적할 수 있을 것이다.

3.4 Selection Algorithm

각 테스트 케이스에 대한 CCG를 갖는다면, 그 새로운 알고리즘은 다음과 같을 것이다.

첫째, 실행에서 실제적으로 영향을 받은 변수들을 식별한다. 그 후, 그것은 함수 f로부터 닿을 수 있는 적어도 하나의 행위에 영향을 받은 함수인 f와 적어도 하나의 수정된 함수 f를 포함하는 테스트 케이스들 중에서 선택을 한다. 만약 그렇다면, 재테스트 되어야만 할 것이다. 이 selection 알고리즘은 재테스트 pool로부터 더 많은 테스트 케이스들을 제거 할 것이고, 그로 인하여 더 정밀해 질 것이다.

Algorithm : Selection Algorithm by using CCG

Input :

Original 컴포넌트 C, 수정된 버전 C'

T : 컴포넌트 C를 테스트 하기위하여 사용된 test suite 각 test case는 그것의 CCG를 갖는다.

Output :

T' : retest될 필요가 있는 test case들을 포함하는 T의 subset

Step 1 : T'={}

test suit T에서 각 test cast ti는 아래를 실행한다.

컴포넌트에서 함수 fi에 대해 CCG에 기반을 둔, fi가 닿을 수 있는 같은 컴포넌트 안에서 다른 모든 함수들을 찾는다.

Step 2 :

각 함수에서 영향을 받은 변수들을 식별한다. 각 수정된 함수 ft, S_{ft} = 변수들은 수정에 의하여 영향을 받는다.

```
For 모든 함수들 Sfn = ∅;
For 각 컴포넌트:
  Worklist = { 컴포넌트에서 수정된 모든 함수 };
  while ( Worklist ≠ ∅ ) {
    Worklist로부터 vi를 선택하고 제거
    For vi에 연결된 모든 노드 vj
      S = Svi - vj의 N-list;
      If(Svi∩vj의 U-list = ∅)
      Then S = S∪vj의 A-list
      If(S ⊄ Svj) Then Svj = S∪Svj;
        Worklist에 vj를 추가
  }
A-list : 함수 f에서 정의되고 영향을 받은 모든 변수들
N-list : 함수 f에서 정의된 모든 변수들
U-list : 함수 f에서 사용된 모든 변수들
```

알고리즘 1. Selection Algorithm

4. 결론 및 향후 연구 방향

본 논문에서 제시한 CCG를 이용하여 컴포넌트의 의존성을 유추하고, 선택 알고리즘을 이용하여 재테스팅을 한다면 불필요한 테스트 케이스의 생성을 방지함으로써 더욱 효율적인 테스팅을 수행할 수 있을 것이다.

이 연구에 대한 향후 과제로는 Black-box 컴포넌트의 ¡인터페이스에 관한 의존성 추출을 통한 CCG의 도식화 작업과 컴포넌트 내부함수의 의존성에대한 CCG의 도식화 작업에 대한 연구가 계속 있어야 할 것이다.

5. 참고 문헌

[1] Hoijin Yoon, Byoungju Choi, "Inter-class Test Technique between Black-box-class and White-box-class for Component Customization Failures" Software Engineering Conference, 1999. (APSEC '99) Proceedings. Sixth Asia Pacific, 1999 Page(s): 162-165

[2] Gary A Bundell, Gareth Lee, John Morris, Kris Parker "A Software Component Verification Tool" Software Methods and Tools, 2000. SMT 2000. proceedings. International Conference on, 2000 Page(s): 137-146

[3] 마유승, 장윤규, 권용래, 뮤테이션 기법을 이용한 컴포넌트의 테스팅, 한국 정보과학회 가을 학술발표 논문집. 1999년

[4] Ye Wu, Mei-Hwa Chen and Howard M.Kao, "Regression Testing on Object-Oriented Programs" Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on, 1999 Page(s): 270-279

[5] Ye Wu, Dai Pan and Mei-Hwa Chen "Techniques for Testing Component-Based Software"

[6] Ye Wu, Dai Pan and Mei-Hwa Chen "Techniques of Maintaining Evolving Component-Based Software"

[7] John Cheesman; John Daniels, "UML Components"