Worm 코드를 이용한 정적분석 도구의 설계

김상영[0] 이영식 황선명
대전대학교 컴퓨터공학과
{jayusop[0], umss}@zeus.dju.ac.kr, sunhwang@dju.ac.kr

Design of a Statics analysis tool through Worm Code

Sang-Young Kim[0] Young-Sik Lee Sun-Myung Hwang
Dept. of Computer Engineering, Daejeon University

요 약

개발자는 품질 요구사항을 만족하는 소프트웨어를 만들 책임이 있기 때문에 최종 제품의 품질뿐만 아니라 중간 제품의 품질에도 관심을 갖는다. 이러한 이유에서 품질관리를 위해서는 소프트웨어 품질 테스팅이 필요로 하게 되는데 테스팅의 방법에는 크게 WhiteBox Testing과 BlackBox Testing으로 나누어 진다.

본 논문에서는 C++ 언어에 적용할 수 있는 정적 분석 도구를 설계하였으며, 이 도구의 특징은 테스팅 도구에 테스터의 의도를 전달 할 수 있는 테스트 제어 언어를 정의하고, 또한 이 정의 언어를 사용하여 Worm 코드를 삽입하여 테스팅 할 수 있는 정적 테스팅 도구를 설계하였다.

1. 서 론

품질의 중요성이 상업이나 산업분야에서 높이 인식됨에 따라 국내외적으로 다양한 표준이 상업, 산업, 군사, 원자력 산업 등의 요구를 충족시키기 위하여 개발되어 구매자와 공급자 조직 간의 계약 용도로 사용되어지고 있다. 물론 소프트웨어도 다른 분야와 마찬가지로 품질의 중요성은 굳이 표현을 할 필요성이 없다. 어떤 제품에 있어어도 품질의 중요성은 아무리 강조해도 지나치지 않는다. 소프트웨어의 품질을 확보하고 결함을 찾아내기 위해 수행되는 일련의 작업인 소프트웨어 테스팅을 통하여 기존에 개발된 소프트웨어 시스템이나 또는 현재 개발중인 시스템에 대하여 고품질화를 실현할 수 있는 것이다.

개발자는 품질 요구사항을 만족하는 소프트웨어를 만들 책임이 있기 때문에 최종 제품의 품질뿐만 아니라 중간 제품의 품질에도 관심을 갖는다. 또한 관리자의 입장에서도 제한된 비용과 인적자원과 시간내에 품질 최적화가 이루어지기를 원하게 된다. 생명주기 후반에 발견되어지는 결함은 초기에 발견되어 수정되어지는 것 보다 훨씬 많은 비용이 들며 재 작업과 손상이 개발비용의 30%에 달하고, 생명주기 비용의 50%에 달하는 소요가 이루어질 수 있다[1].

이러한 이유에서 품질관리를 위해서는 소프트웨어 품질 테스팅이 필요로 하게 되는데 테스팅의 방법에는 크게 WhiteBox Testing과 BlackBox Testing으로 나누어 진다. 소프트웨어 제품이 개발되어지는 과정에서는 정적 분석을 통한 테스팅인 WhiteBox Testing 이 이루어지며, 이러한 방법들은 많이 연구되어져 왔다. 그러나 원시코드를 대상으로 실제 수행 시 처리되어지는 수행경로나 변수들의 값의 검사는 테스팅의 중요 검사 요소로 볼 수 있으나 검사 할 수 있는 마땅한 도구가 거의 없다. 또한 현재 제품화되어 있는 테스팅 도구는 기계어 수준에서 검사가 되어지는 수준이다[2].

본 논문에서는 기존의 테스팅 도구를 분류하고 분석하며 원시코드 내에 프로그램의 수행경로와 테스터가 요구하는 변수에 대하여 값을 검사할 수 있는 도구를 설계하도록 하였다. 또한 검사시 사용되어질 수 있는 테스트 언어를 정의하였다.

2 테스팅 도구 사례 연구

2.1 시험도구의 필요성

요즘 대부분의 응용 소프트웨어들은 시험하기가 복잡하여 더 많은 시험활동을 요구하고 있다. 다양한 플랫폼의 지원, 네트워크를 지원하는 분산환경에서의 수행, 그리고 비순차적이면서 사용자에게 풍부한 정보와 융통성을 제공하는 GUI의 제공 등은 개발자 및 시험 담당자로 하여금 대상 소프트웨어에 대한 기능적이고 기술적인 품질을 확보할 수 있는 시험을 요구하고 있다. 또한 사용자 인터페이스의 사용성, 반응시간 및 데이터의 정확성을 보장하고 사용자에 따른 서로 다른 목적 및 환경에서 응용 소프트웨어를 수행할 수 있도록 하기 위해서는 가능한 많은 시험경로를 거치도록 시험케이스를 설계하여야 한다. 이러한 상황에서 시험 수행시간 단축과 효율적 시험을 위해 디버거와 같은 단순한 시험도구에서 벗어나 다양한 자동화 시험도구의 적용이 필수적이다. 시험도구를 적용하지 않은 시험과 비교해 볼 때, 자동화 시험도구를 적용한 경우는 오류 발견 및 수정 작업을 더욱 효과적이고 효율적으로 수행할 수 있다([그림 2-1] 참조).

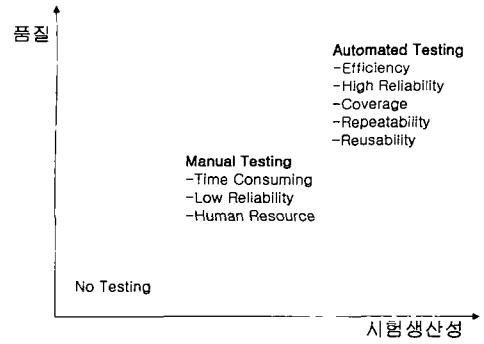
자동화 시험도구는 시험데이터 및 시험 스크립트 작성을 편리하게 지원해 주기 때문에 시험설계 시간 단축은 물론, 일단 만들어진 정보를 이용하여 누구나 반복적으로 동일한 시험을 수행할 수 있게 한다.

즉, 여러 사람이 시험을 하여도 그 결과가 동일하여 시험수행의 신뢰성을 보장할 수 있으며, 유지보수 후 변경사항에 대한 재확인 및 파급효과로 인한 부가적인 문제점을 쉽게 발견할 수 있게 한다.

2.2 시험 도구의 분류

자동화 시험도구는 도구의 적용방법에 따라 시험관리도구, 시험수행도구, 품질평가도구 등으로 분류될 수 있고 시험수행도구는 다시 회귀시험, 부하시험, 커버리지시험 등을 수행하는 도구로 분류될 수 있다([표 2-1] 참조)[2].

시험관리도구는 시험계획, 시험케이스, 시험 절차 및 시험 일정을 관리하고 시험수행 결과에 대한 통계 및 보고서를 생성한다.



[그림 2-1] Manual 대 Automated Testing

시험수행도구는 시험 경로의 생성, 사용자 없이 수행할 수 있도록 입력 및 수행내역을 캡처하고 재생하는 기능을 제공하며 시스템에 부하를 주기 위한 물리적 또는 논리적 부하 태스크의 생성/수행 기능과 시험의 충분성 또는 완전성을 보장하기 위한 커버리지 측정 기능을 제공한다.

품질평가도구는 특정 시험방법론을 적용하여 원시프로그램의 품질을 부여하여 준다. 즉, 적용 메트릭(모듈 복잡도, 코딩지침 준수도....)에 의한 원시프로그램의 제어흐름 및 데이터흐름 구조를 분석하여 시험용이성 및 변경용이성 등의 유지보수성 측면의 품질을 평가한다.

현재 상용화된 자동화 시험도구로서 소프트웨어 품질 향상을 위해 보편적으로 사용되고 있는 대표적인 시험도구들을 살펴보면 [표 2-1]과 같다.

2.3 품질평가 및 시험수행도구

소프트웨어 시험시 적용한 품질평가 및 시험수행도구는 소프트웨어 구조와 모듈간의 호출관계 등을 표현하고 제어흐름그래프 생성을 통한 모듈의 복잡도 및 시험경로를 생성하는 정적분석기능과 시험된 경로와 시험되지 않은 경로를 제시하고 프로그램간의 차이를 자동으로 비교, 분석해 주는 동적분석기능을 포함하고 있으며 현재 많은 응용 소프트웨어에 적용하고 있는 객체지향 프로그래밍에 대한 분석기능도 포함하고 있다[2].

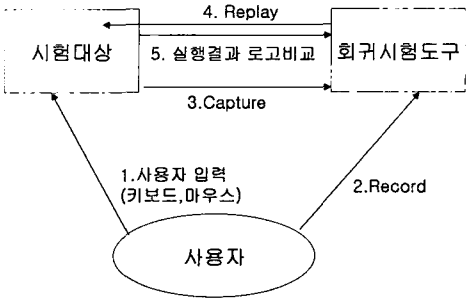
정적분석기능은 원시프로그램을 실행하지 않고 제어흐름과 데이터 흐름 등의 내부구조를 분석하여 변경용이성과 시험용이성을 분석해 주며, 단위시험 및 통합시험에 대한 시험경로를 제시하고 동적분석기능은 블랙박스시험에 의해 수행되지 않은 시험경로를 보여준다

2.4 회귀시험 도구

소프트웨어 시험시 적용한 회귀시험 도구는 키보드, 마우스 등을 이용한 사용자 입력 사항을 기록하여 시험스크립트를 생성한다([그림 2-2] 참조)[3].

이렇게 생성된 스크립트를 가지고 시험자는 동기화(Synchronization) 포인트를 삽입하여 응용프로그램 실행시 스크립트의 실행을 지연하여 윈도우나 비트맵 또는 상대방 등이 화면에 나타나기를 기다리도록 하거나 GUI Object, Bitmap과 Text 등에 대한 검증을 위한 GUI Check 포인트를 추가할 수 있다.

회귀시험 도구는 작성된 스크립트를 자동적으로 재수행(Replay)하여 실제 수행결과와 예상결과의 차이를 비교하여 시험결과 로그로 남긴다.



[그림 2-2] 회귀시험 도구의 동작

시험자는 회귀시험 도구가 제공하는 시험결과 로그와 해당 검증화일을 참고로 하여 발견된 오류의 수정을 효과적으로 수행할 수 있다.

GUI를 제공하는 시스템에 대하여 회귀시험 도구를 적용하여 사용자 인터페이스의 사용 편리성을 효과적으로 평가할 수 있

[표 2-1] 시험도구 분류

분 류		특 징	시험도구명
시험관리도구		test planning, test procedure and bug management, cause-effect diagram of specification based test case generation etc.	SQA Manager, SoftTest, StP/T, McCabe Visual Testing Tool Set, TesMaster, TESTByes
시험수행도구	회귀시험	automated GUI client or server testing, fully OO and script-driven approach, integrated with test repository, OO scenario recording, wedget-based testing, support embedded and real-time environment etc	SQA Partner, TestRunner, SQA Robot, XRunner, WinRunner, CAPBAK/X, CAPBAK/MSW, CAPBAK/NI, Visual Test 4.0, QC/Replay, winVue,er, Final Exam Internet Test, preVue
	부하시험	stress & performance testing, SL and graphic capture between client and server, support distrubuted networking etc.	Load Runner, PurePerformix, SQA LoadTest, preVue-X C/S, Final Exam Internet Load Test, TestView, ITF
	커버리지 시험	coverage analysis(branch coverage, path coverage, code coverage, function coverage)	PureCoverage, Logiscope, McCabe Visual Tool Set, Final Exam Test Advisor
품질평가도구		source code analysis, calculate code metric, generate control flow graph etc.	McCabe Visual Tool Set, Visionsoft/PROJECT
기타 (메모리오류추적 등)		runtime memory error detect, parallel real-time application monitoring, bottleneck analysis etc.	Purify, Quantify, TestGen, safec, PARTAMOS

었으며 특히 육안으로는 정확하게 식별하거나 판단하기 어려운 대용량 이미지 변환 및 전송된 파일간의 동일성에 대한 자동검증으로 시험결과에 대한 객관성을 높일 수 있었다. 그리고 사용자 정보서비스의 다양한 요구 유형을 시험케이스로 작성하여 동시에 시험을 수행함으로써 시험시간을 단축할 수 있었다.

또한, 오류 수정에 따른 버전 변경이 일어난 경우 이전에 수행한 시험스크립트를 재 수행함으로써, 변경에 따른 시스템의 영향범위 및 변경의 정확성을 효과적으로 확인할 수 있다.

3 테스팅 도구의 설계 및 구현

3.1 PEV의 정의

바이너리 형태의 프로그램은 원시코드와는 달리 동적으로 경로의 검색 및 수행결과 확인 그리고 내부 변수의 검색등에 상당히 어려움이 있다. 개발자가 프로그램 개발시 컴파일러의 도움으로 디버깅 정보를 이용하여 변수의 검사나 수행 Step 검사등은 가능하나 이것은 난해하다는 한계점이 있어왔다. 그래서 기존의 개발도구나 테스팅 도구들과는 차별화된 방법을 통하여 프로그램을 수행검사 할 수 있는 도구를 설계 및 구현하였다.

본 논문에서는 PEV(Program Evaluator)라 명명하였으며 원시코드에 프로그램을 제어할 수 있는 제어문을 삽입하는 방법으로 테스트를 수행하도록 하였다.

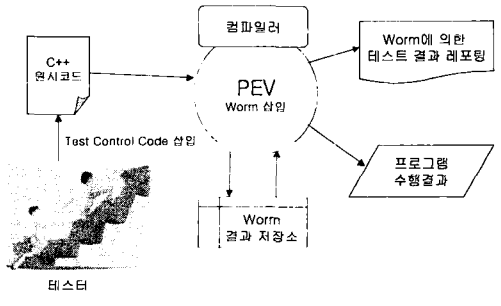
3.2 PEV의 설계

PEV는 프로그램의 원시코드에 테스트 제어문(Test Control Statement)를 삽입하고, 확장된 원시코드를 PEV에 지정하게 되면 PEV는 테스트 제어문에서 지정한 내용에 대하여 컴파일 가능한 웜(Worm Code)을 삽입하고 기존의 테스트 제어문을 삭제후 컴파일 및 테스트를 수행하도록 설계되어졌다. PEV의 테스팅 절차는 [그림 3-1]와 같으며, PEV의 내부 구조는 [그림 3-2]와 같다.

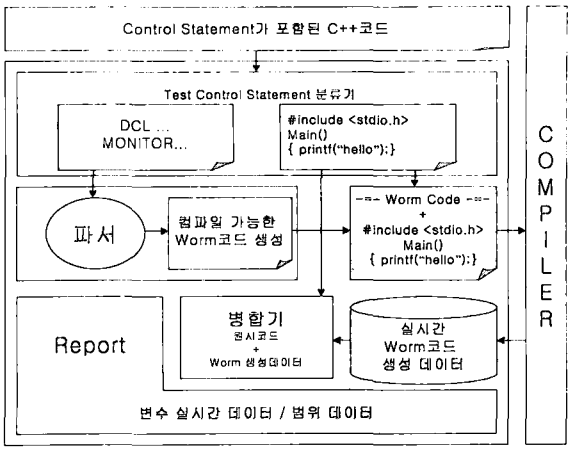
PEV로 테스트를 수행하기 위해서는 원시코드내에 테스트 제어문이 필요한데 [표 3-1]과 같은 문법체계를 가지고 있다.

[표 3-1] Test Control Statement 문법체계

```
L → test_control_language
test_control_language → dcl | monitor
dcl → "DCL CHECK"  variable
variable → alphabet | number | variable
alphabet → a | b | ... | y | z | A | B | ... | Y | Z
number → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
monitor → "MONITOR" variable { exec | range }
exec → "EXEC TIME" exenum "LINE" exenum
exenum → number | exenum
range → "RANGE" exenum to exenum
```



[그림 3-1] PEV의 테스팅 절차



[그림 3-2] PEV의 내부구조

- Test Control Statement 분류기 : 테스트 대상으로 입력받은 코드를 C++ 문법과 Test Control Statement와 분류하여 각각의 임시 저장소에 저장하고 분류되어진 Test Control Statement를 파서의 입력 데이터로 보내준다.
- 파서 : Test Control Statement 분류기로부터 추출되어진 구문을 파서에서 [표 3-1]의 구문 법에 의해 토큰 분석을 한 다음 해당하는 구문에 맞는 Worm 코드를 작성하고, 작성된 코드를 Test Control Statement 분류기에서 분류되어진 C++의 코드의 적당한 위치에 삽입시킨 후 임시 파일을 만들어 컴파일러의 입력으로 보내어지고 컴파일 되어진 바이너리 파일은 수행되면서 Worm에 의해서 실시간으로 테스터가 지정한 값들을 수집하여 외부 저장소에 저장을 한다.
- 병합기 : 바이너리가 수행되면서 수집되어진 테스트 데이터의 내용과 C++ 원시코드의 내용을 병합 가공한다.
- 레포트 : 병합기에서 가공되어진 내용을 테이블 형태로 출력하여 테스터가 검색할 수 있도록 해준다.

5. 결 론

프로그램의 테스팅은 개발 환경이 나아지고 프로그래밍 언어가 발전될수록 점점 더 복잡해지고 어려워져 가고 있다. 이러한 환경속에서 보다 나은 테스팅 환경을 테스터에게 제공하는 것은 소프트웨어공학의 소프트웨어 테스팅 부분에 있어서 많은 발전을 가져와 줄 수 있다. 그러나 우리의 주위에 보면 상당히 많은 도구들이 존재하고 있으며, 그러한 도구들의 각각의 특색에 맞게 프로그래밍 언어별로 구성되어져 있다.

본 논문에서는 C++ 언어에 적용할 수 있는 정적 분석 도구를 설계하였으며 이 도구의 특징은 테스팅 도구에 테스터의 의도를 전달 할 수 있는 테스트 제어 언어를 정의하고 또한 이 정의 언어를 사용하여 Worm 코드를 삽입하여 테스팅 할 수 있는 테스팅 도구를 설계하였다.

6. 참고문헌

[1] Brian Henderson-Sellers, "Object-Oriented Metrics," Prentice Hall, 1996.
[2] Binder, "Testing Object-Oriented System," Addison Weskey, 1999.
[3] Graham Titterington, "Software Testing Tools," Ovum Press, 1997.