제26회 한국정보처리학회 추계학술발표대회 논문집 제13권 제2호 (2006. 11)

클래스기반 객체지향소프트웨어 테스트 프레임워크

정일재*, 박상필*, 염희균*, 황선명*
*대전대학교 컴퓨터공학과
e-mail:sky7681@nate.com

Object Oriented Software Testing Framework Based on Class

Il-Jae Jung*, Sang-Pil Park*,
Hee-Gyun Yeom*, Sun-Myung Hwang*
*Dept of Computer Engineering, Daejeon University

요 약

소프트웨어의 테스트는 소프트웨어의 품질결정을 위한 필수항목으로서 체계적인 방법과 절차에 따라서 진행되어야 한다. 많은 노력으로 개발되고 판매되는 소프트웨어서도 오류가 발견되며, 발견된 오류의 절반이상이 개발과정에서 발생 하므로 개발단계부터 결함을 제거할 수 있는 소프트 테스트가 필요하다. 소프트웨어의 테스트는 소프트웨어가 좋은가를 검증하는 것이 아니라 요구된 명세서의 구현과 구현된 소프트웨어에 결함이 있는가를 찾는 행위이며, 소프트웨어 개발에 최종적인 검토이다. 본 연구에서는 객체지향 소프트웨어의 품질을 향상시키기 위하여 클래스 기반의 단위테스트 수행 시 "Error Seeding" 전략을 통한 동적테스트를 수행하고, 테스팅 과정에 대한 관리 및 성능 테스트를 제공하는 도구를 제안한다. 본 프레임워크는 테스트 유형을 테스트 수행자가 결정하고 테스트를 수행하므로 결과판독이 정확하고 빠른 장점이 있다.

1. 서론

1.1 연구배경 및 목적

소프트웨어 제품의 품질이 제품선택의 가장 중요한 요소로 부상되면서 소프트웨어 사용자들은 개발사에 지속적인 소프트웨어의 품질 향상을 요구하고 있다. 또한 여러 측면에서 소프트웨어 품질에 대한 객관적인 평가와 수행을 통하여 소프트웨어 품질에 대한 인증이 실시되고 있다.

이러한 소프트웨어 품질 인증에 소프트웨어의 테스트는 중요한 요소이며, 명세서, 설계, 그리고 코드 생성의 최종적 검토를 나타낸다[1]. 또한 시스템의 복잡성, 다양한 요구사항, 편리한 접근성, 치밀한 보안성, 기타 위험 요소 배제를 위해 프로젝트 개발 조직의 테스트에 대한 많은 노력이 요구된다.

그러나 이러한 노력을 통하여 개발되고 판매된 소프트웨어에서도 오류가 발견되며, 발견된 오류의 절반 이상이 개발과정에서 발생하므로 초기 개발단계부터 결함을 제거할 수 있는 소프트웨어 테스트 인프라의 구축이 필요하며, 인프라의 구축은 손실 비용을 효과적으로 줄일 수 있다[5].

효율적인 테스트를 위한 테스트 지원 도구 개발 역시 인프라 구축에 일환으로 볼 수 있다.

본 논문에서는 객체지향 소프트웨어 테스트 수행 시 캡슐화, 정보은닉, 상속성, 다형성과 동적 바인딩으로 발생될 수 있는 문제점을 극복하고자 "클래스 테스트 프레임워크(Class Test Framework : CLT프레임워크)"를 구현하고, CLT프레임워크를 적용하여 적용할 수 있는 세분화된 객체지향테스트 프로세스를 제안한다.

2. 관련연구

2.1 객체지향소프트웨어의 테스트

소프트웨어 테스트는 일정한 기간동안 관리할 수 있는 범위내의 노력으로 가능한 한 많은 수의 오류를 발견하는 것이다. 이런 기본 목적은 객체지향 소프트웨어에 대해서도 그대도 적용되지만, 객체 지향 소프트웨어의 특성에 의해 테스트 전략과 전술이 변경된다.

2.2 객체지향소프트웨어의 특징

객체지향소프트웨어는 속성과 함수의 집합인 객체를 사용하여 프로그램을 개발하는 방법으로 시스템을 쉽게 구축할 수 있으며 확장, 유지 보수가 용이해지며, 재사용과 변경이 용이하다.

객체는 실세계 사물을 표현한 모듈로 객체를 사용하면 캡슐화, 정보은닉, 상속성, 다형성, 동적 바인딩 등의 개념이 사용 가능하다.

2.3 객체지향 소프트웨어 테스트 종류

객체지향 소프트웨어 테스트 종류에는 모델, 단위, 통합, 시스템테스트로 구성되어져 있다. 본 프레임워크에서 사용되는 테스트는 단위 테스트이다.

2.3.1 단위테스트

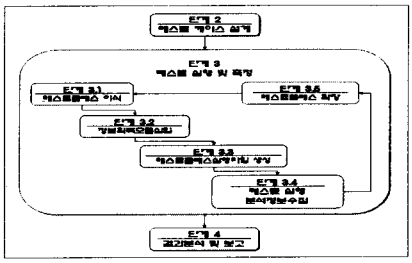
객체 지향 소프트웨어의 단위는 캡슐화된 클래스 또는 객체를 의미한다. 다시 말해 캡슐화는 클래스들과 객체들의 정의를 이끌어 내고, 각각의 클래스(객체)가 속성(데이터)과 데이터를 조작하는 동작(메소드, 서비스)들을 패키지로 묶는 것을 의미한다. 이러한 관점에서 테스트가 가능한 가장 작은 단위는 클래스 또는 객체이다.

객체 지향 소프트웨어에서 단위테스트는 클래스에서 여러 개의 다른 동작을 포함할 수 있고, 특정한 동작은 여러 개의 다른 클래스들의 부분에 존재할 수 있기 때문에 단위 테스트의 의미가 많이 바뀐다. 구조적 소프트웨어서 하나의 동작을 분리하여 테스트 하는 것이 아니라 클래스, 객체의 일부로써 테스트되어야 한다.

3. CLT프레임워크의 설계

3.1 테스트프로세스 절차의 세분화

CLT프레임워크의 적용을 위한 테스트 프로세스를 (그림 1)과 같이 세분화하여 제안한다.



(그림 1) 테스트 수행 및 측정의 세부 프로세스

- 단계 3.1 테스트클래스 이식

CLT프레임워크에서 개발프로그램의 단위테스트를 위하여 테스트하고자하는 클래스를 이식한다. 이때 각 클래스들은 재사용을 고려하여 설계되었는지, 통합테스트를 위하여 테스터가 준비해야할 사항을 관찰할 수 있다.

- 단계 3.2 정보획득모듈 삽입

이식된 클래스의 메소드, 속성들의 테스트를 위하여 정보획득모듈을 삽입한다. 삽입된 모듈들은 테스트의 실행 시 클래스의 상태, 메소드의 수행 경로 등 모듈의 특성에 맞는 로그를 생성하게 된다.

- 단계 3.3 테스트클래스 실행파일 생성

삽입이 완료되면 클래스를 실행하기 위한 실행파일을 생성한다.

- 단계 3.4 테스트 실행 및 분석정보 수집

생성된 테스트클래스 실행파일을 수행하면 클래스의 메소드 및 데이터에 접근할 수 있는 유저인터페이스를 제공하고, 준비된 테스트 케이스를 적용하여 각 메소드 등을 수행한다. 이때 삽입된 정보획득모듈들은 별도의 창에 수행되는 로그를 보여주며 이를 저장한다. 저장된 로그를 이용하며 CLT프레임워크의 분석 모듈을 이용하여 메소드의 수행 경로, 속성의 모니터링, 객체의 상태 변화를 분석할 수 있다.

- 단계 3.5 테스트클래스 확장

목적한 테스트클래스의 단위 테스트가 완료되면 추가적으로 다른 클래스를 이식한다. 이러한 클래스의 단위테스트를 반복적으로, 점차적으로 확장하여 통합 테스트를 실시한다.

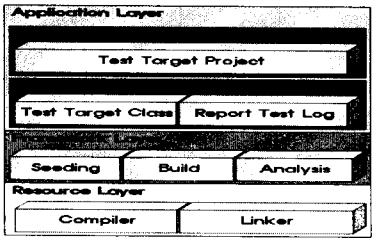
3.2 CLT프레임워크 아키텍처

CLT프레임워크는 크게 "애플리케이션 레이어", "프레임워크 레이어", "리소스 레이어"로 되어있다.

- 애플리케이션 레이어 : 실행이 가능한 프로그

램 형태로 구분한 것으로 테스트 소스와 테스트 타깃 레이어로 세분된다.

- **테스트 소스 레이어** : 테스트의 대상되는 개발된 객체지향 소프트웨어 계층
- **테스트 타깃 레이어** : 테스트 대상 소프트웨어로부터 단위테스트를 수행하기위해 분리된 클래스를 기반으로 생성된 테스트클래스실행 프로그램이다. 실제적인 테스트케이스를 적용하여 클래스테스트를 수행하는 계층이다.
- **프레임워크 레이어** : 테스트 소스에서 분리된 테스트 대상클래스에 대하여 테스트 정보획득모듈을 삽입하고, 클래스테스트실행파일 생성을 수행한다. 또한 테스트 수행을 통해 수집된 정보를 분석하여 테스트 결과를 보고하는 계층이다.
- **리소스 레이어** : 프레임워크에서 클래스테스트실행파일을 생성하기위한 외부 환경을 제공하는 계층이다



(그림 2) CLT프레임워크 아키텍처

3.3 CLT프레임워크의 기능 설계

CLT프레임워크의 기능은 테스트클래스 이식, 정보획득모듈삽입 및 클래스테스트실행프로그램 생성을 수행할 수 있는 테스트 준비기능과 테스트수행 후 수집된 정보를 바탕으로 테스트 결과를 분석하는 테스트 분석기능으로 구분된다.

3.3.1 테스트 준비기능

테스트 준비기능은 앞서 설명한 바와 같이 대상 클래스를 테스트하기 위한 준비 기능이다. 이러한 기능을 수행하면서 테스트 대상클래스에 대한 참조성과 재사용성을 확인할 수 있다.

3.3.2 테스트 분석기능

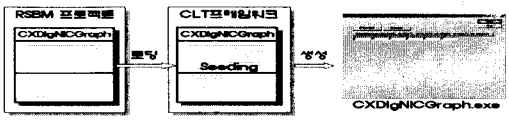
테스트 분석기능은 정보수집모듈을 삽입한 후 테스트를 수행하여 수집된 정보를 바탕으로 테스트 결과를 분석하는 기능이다.

4. CLT프레임워크 적용 테스트

CLT프레임워크를 이용한 객체지향 소프트웨어 적용 평가를 위하여 테스트를 수행하고 그 결과를 분석한다.

4.1 테스트 준비

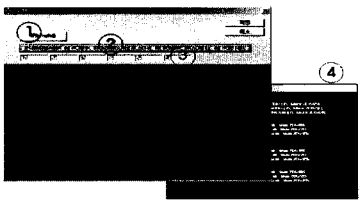
CLT프레임워크를 이용하여 RSBM 프로젝트의 테스트 대상 클래스인 CXDlgNICGraph를 이식하고 테스트 수행 방법에 제시된 정보획득모듈을 노드포인트를 설정하여 삽입한다. 삽입이 완료되면 CXDlgNICGraph.exe 테스트 실행파일을 생성한다.



(그림 3) 테스트 목적 실행 파일 생성 개념도

4.2 테스트 수행

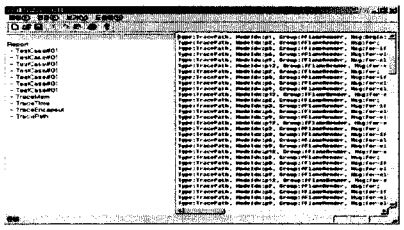
테스트를 수행하기 위해 CLT프레임워크에서 생성한 CXDlgNICGraph.exe를 이용하여 앞서 정의된 테스트 케이스를 적용하여 테스트를 수행한다.



(그림 4) 실시간 테스트 수행 화면

4.3 테스트 결과 분석

준비된 테스트케이스를 모두 수행하고 충분한 테스트를 수행하였으며, 테스트를 중단하고 CLT프레임워크를 실행하여 로그파일을 분석한다.



(그림 5) CLT프레임워크의 테스트결과 분석 수행

4.4 다른 테스트 도구와 비교

CLT프레임워크와 현재 개발되어 사용되는 테스트 도구와 기능을 [표 1]과 같이 각 도구들과 비교할 수 있다.

[표 1] 대표적인 테스트 도구와 CLT프레임워크의 비교

구분	CLT	jUnit	Load Runner	이지트랙	Robort
Language	c++	java	web	n/a	web
지원 API	MFC	n/a	web	n/a	web
Target	Binary	Binary	n/a	n/a	n/a
방 법	반자동	반자동	자동	n/a	자동
성 능	○	○	○	n/a	○
경 로	○	○	×	n/a	×
재사용 성	○	×	×	n/a	×
다 형 성	○	×	×	n/a	×
추상화/캡슐화	○	×	×	n/a	×
경계값 추적	○	○	×	n/a	×
스트레스테스트	×	×	○	n/a	○
메모리릭	○	×	×	n/a	×
테스트 레포팅	△	×	○	○	○
비 고				테스트 문서관리	

4.4 적용 장단점

CLT프레임워크는 기본적으로 클래스를 최소 단위로 테스트하며 클래스 내부에 대한 테스트를 시작으로 점진적으로 통합테스트를 수행한다. CLT프레임워크를 적용한 장단점을 요약하면 [표 2]과 같다.

[표 2] CLT프레임워크 적용 장점 및 개선필요사항 요약

구 분	내 용
장점	- 테스트 타깃클래스 실행파일을 생성 실시간으로 테스트 진행 · 테스트케이스 일괄 입력 가능 · 실시간 모니터링 - 테스트 결과정보 형태를 테스터가 직접 정의 가능하므로 가독성 높음 - 여러 가지 형태의 테스트 수행 가능 - 노드포인트 자동검색, 입력 가능하여 짧은 테스트 준비시간 - 테스트분석 모듈화로 짧은 결과분석시간 - 노드포인트 개념으로 결함 검출시 결함지역 쉽게 접근 가능
개선 필요	- 정보획득모듈삽입으로 테스트 타깃클래스 성능 저하 · 정보획득모듈삽입설계 주위 - 추가산출물 발생 (테스트타깃클래스실행파일, 정보분석파일) - 다양한 인터페이스 제공 요구됨 - 세분화된 프로세스로 추가적인 작업 요구됨 · CLT프레임워크(생성), 클래스실행파일(실행), CLT프레임워크(분석)

5. 결론

소프트웨어의 테스트는 소프트웨어가 좋은가를 검증하는 것이 아니라 요구된 명세서의 구현과 구현된 소프트웨어에 결함이 있는가를 찾는 행위이다. 이를 통하여 소프트웨어의 완성도를 높여가는 행위로 소프트웨어의 품질보장에 대한 기본적인 접근 프로세스이며, 최종적 검토이다.

소프트웨어의 테스트 연구는 계획, 설계, 테스트케이스, 수행 방법 등에 대하여 많은 연구가 이루어지고 있으나 소프트웨어에 대한 높은 이해를 가진 테스터나 개발자에게 의존되어 온 것이 사실이다.

본 논문에서는 객체지향소프트웨어 테스트를 어렵게 하는 문제점과 단위, 통합 테스트 수행의 효율성, 테스트 결과에 대한 가독성 등을 높이기 위하여 CLT프레임워크(CLass Test Framework)을 구현하였다.

또한 지속적으로 다양한 인터페이스를 반영할 수 있는 방법에 대한 연구와 테스트 수행정보를 획득하기 위해 테스트 클래스 내부에 삽입되는 정보 획득 모듈의 정형화가 필요하다. 또한 결과분석 모듈의 신뢰성 확보를 위한 지속적인 연구도 수행되어야한다.

참고문헌

[1] Pressman, R., Software Engineering: A Practitioner's Approach, McGraw-Hill, 2003.

[2] Davis, A., 201 Principles of Software Development, McGraw-Hill, 1995

[3] Patton, R., Software Testing, Sams, 2000.

[4] Lindland, O.I., et al. "Understanding Quality in Conceptual Modeling," IEEE Software, vol.11, no.4, July 1994

[5] NIST, "The Economic Impacts of Inadequate Infra-structure for Software Testing", 2002.5

[6] 한국정보통신기술협회, 소프트웨어테스트 전문기술 기초분야, 한국정보통신기술협회, 2005

[7] 한국정보통신기술협회, 소프트웨어테스트 전문기술 응용분야, 한국정보통신기술협회, 2005

[8] 소프트웨어테스트 엔지니어네트워크, "STEN Journal Vol. Online eBook",

[9] 퍼슨넷, "제3자 소프트웨어 테스팅 사례(Case on 3'rd Party S/W testing)", 스텐컨퍼런스 발표

[10] SQA G. S/W Center, "Manual vs. Automated Test에 대한 사례 연구 소개", 2003

[11] 황선명, Regular Expression에 의한 복잡도 측정과 소프트웨어 품질보증을 위한 툴의 구축, 1987