□특집□

형식명세에 의한 소프트웨어 테스팅 자동화

한 규 정† 정 연 대††

◈ 목 차 ◈

1. 서 론

소프트웨어 시스템의 개발 과정은 그 자체가 많은 에러를 유발시키는 요소를 내포하고 있다. 그 이유는 시스템 개발자가 그 자신의 일을 수행하고 의사를 전달하는 데에는 한계가 있기 때문이다. 따라서 소프트웨어 테스팅은 소프트웨어의 요구 사항에 따라 실행하는 지를 확인하는 소프트웨어 품질 보증을 위한 하나의 평가 요소로서 중요시되고 있다. 그러므로 테스팅이 소프트웨어 생명 주기에 있어서 전체의 노력에 약 40% 이상을 차지하는 것은 이상한 일이 아니다. 특히 인간의 생명에 관련된 소프트웨어의 테스팅, 가령 비행 또는 항공 통제나 핵 반응감시 등에서의 비용은 일반 소프트웨어 개발보다 3~5배의 비용이 소요된다. 일반적으로 소프트웨어 테스팅은 소스 코드를 기반으로 하는 프로그램 테스팅과 명세를 기반으로 하는 명세 기반 테스팅으로 구분된다. 본 연구에서는 일반적인 테스팅 방법론과 형식 명세(formal spec)를 기반으로 하는 형식 명세 테스팅에 대해 알아보고, 또한 일반적으로 현업에서 사용하는 테스팅 툴의 기능과 그 유형, 비용 및 기술 수준에 대해서 논한다.

※ 이 논문은 1997년도 시스템공학연구소의 지원을 받은 'LOTOS 명세로부터의 테스트 케이스 자동 생성' 연구의 일부분임.

† 종신회원 : 공주교육대학교 컴퓨터 교수,
공주교육대학교 전자계산 소장
†† 정 회 원 : 시스템공학연구소 책임연구원

2. 테스팅 전략

프로그램의 테스트 계획의 실행은 요구 사항에서 제안된 활동(activity)을 확인해야 하는데 프로그램 상에서 모든 에러를 발견하여 100%의 프로그램 정확성(correctness)을 보장하기는 불가능하

다. 따라서 이상적인 테스팅을 수행하기 위한 전략들이 연구되었는데, 그 중 Meyers [1]는 좋은 테스팅 전략에 대해 다음과 같은 몇 가지 규칙을 제시하고 있다.

- 테스팅은 오류를 발견하기 위한 하나의 프로그램 실행과정이다.
- 좋은 테스트 케이스(test case)는 미처 발견하지 않은 오류를 발견할 가능성을 높인다.
- 성공적인 테스팅이란 아직 발견하지 않은 오류를 찾아내는 것이다.

프로그램의 테스팅은 크게 두 가지 전략들로 접근한다. 그 하나는 프로그램의 원시 코드를 분석하지 않고 명세에 크게 의존하여 프로그램을 기능 중심으로 테스팅하는 명세 기반 테스팅(specification-based testing)으로서, 기능적 테스팅(functional testing) 혹은 블랙 박스 테스팅(black box testing)이라고도 한다. 또 다른 하나는 프로그램의 원시 코드의 내부 구조를 분석하여 테스팅하는 프로그램 기반 테스팅(program-based testing)으로서, 구조적 테스팅(structured testing) 혹은 화이트 박스 테스팅 (white box testing) 등이 같은 의미로 사용된다. 그리고 테스팅의 대상에 따른 구분으로 단위 모듈(unit module)에 국한하여 수행시 단위 테스팅(unit testing)이라 하고, 통합 모듈(integrated module)을 대상으로 수행시 통합 테스팅(integrated testing)이라 한다. 또한 프로그램의 변경시 변경된 모듈을 대상으로 테스팅을 수행할 때는 재테스팅(retesting)이라고 한다 [2].

2.1 구조적 테스팅

구조적 테스팅은 다른 말로 White-Box 테스팅이라고도 하는데 이는 프로그램의 내부 구조를 고려한다. 가령 논리적 경로, 조건 반복과 같은 부분이 테스팅 대상이다. 따라서 어디에 관심을 구하느냐에 따라 테스팅의 방법론들이 달라진다. 그 중 프로그램의 모든 문장을 수행하도록 하는 문장 커버리지(statement coverage), 분기(branch)를 수행하는 분기 커버리지, 데이터 흐름(data flow) 관점인 데이터 흐름 커버리지 등 다양하다. 이러한 구조적 테스팅 기법으로 프로그램의 100% 정확한 프로그램을 보장할 것 같았으나 불행하게도 매우 작은 프로그램이라 할지라도 가능한 논리 경로의 수가 매우 많게 됨으로 불가능하다. 가령 소모성 테스팅(exhaustive testing)으로 경로수가 1014 인 프로그램의 경로 그래프(flow graph)가 있다고 하자. 이런 경로 그래프에는 루프를 포함한 테스트 경로(test paths)를 대상으로 각 테스트 케이스에 대해 검증하는데 걸리는 시간을 5 초라고 가정시, 모든 경로에 대한 검증은 약 10억 년이 걸린다. 이렇듯 모든 경로에 대한 검증은 불가능하다.

그러므로 테스팅 방법들의 일관된 수행 순서는 다음과 같다.

첫째, 테스트 대상이 되는 소스 프로그램에 대해 경로 그래프를 구축한다.

둘째, 경로 그래프 상에 테스트 경로를 생성한다.

셋째, 생성된 테스트 경로를 대상으로 테스트 데이터를 생성한다.

넷째, 생성된 테스트 데이터를 동적으로 실행하여 출력을 분석한다.

이러한 네 가지 단계들 모두가 테스팅에 관련된 연구 부분들이 되어 왔고, 궁극적으로 효율적인 테스트 데이터 생성에 큰 관심이 집중 되었다.

2.2 기능적 테스팅

기능적 테스팅은 Black-Box 테스팅이라고 하며 프로그램의 개발 명세나 프로그램이 수행하려는 의도에서 테스트 케이스를 추출하여 바라진 결과와 실행결과를 비교하게 된다. 이런 기능 위주의 테스팅은 전술한 바와 같이 명세 기반 테스팅과

매우 밀접하다. 3장에서는 이러한 명세 기반 테스팅중 형식 명세를 중심으로 한 테스팅 방법론을 알아본다.

2.3 테스팅 기술 수준 측정

<표 1>은 여러분 회사의 테스팅 기술 수준을 측정할 수 있는 질문서이다. 체크한 결과에 따라 회사의 기술 수준을 점검할 수 있다[3]. '아니오'로 체크한 결과에 따라 그 회사의 수준은 다음과 같다.

(1) 17~20개인 경우

테스트를 예술(arts)로서 생각하는 그룹으로 테스팅에 대한 보다 많은 관심을 가져야 한다.

(2) 13~16개인 경우

테스트를 기교(crafts)로서 생각하는 그룹으로 테스트 과정과 같은 테스트 연습의 과정을 실시하고 있다. 이 그룹에서는 이런 테스트 연습과 같은 형태를 테스트 접근 방법의 차원으로 끌어가야 한다.

(3) 9~12개인 경우

정의된 테스트 과정을 가지고 테스트하는 그룹이다. 테스트 접근 방법을 확립하고 있다. 그러나 보다 조직적인 테스트접근 방법과 테스트 연습이 필요하다.

(4) 5~8개인 경우

우수한 테스트 그룹이다. 훌륭한 테스트 접근과 테스트 연습을 가지고 있다. 그러나 조직내 테스트 과정의 평가나 향상이 필요하다.

(5) 0~4개인 경우

세계적인 수준의 테스트 그룹이다.

3. 형식 명세 기반 테스팅

3.1 명세 기반 테스팅

이 방법은 전술한 것과 같이 명세에서 테스트 케이스를 유도한다. 명세는 소프트웨어 제품에 대한 속성이나 예상되는 행위(behavior)를 서술하게 된다. 작성 형태는 명세언어로 작성된 형식(formal) 형태이거나 회의 도중에 서술된 메모형태의 비형식(informal)의 형태도 있을 수 있다.

3.2 명세 기반 테스팅의 발전

1970년대 중반/후반까지는 테스팅 분야에는 전반적으로 별로 관심이 없었다. 그러나 1980년대에 들어서서 형식(formal) 언어의 발달로 관심을 갖게 되고, 통신 소프트웨어와 소프트웨어 툴이 발전함에 따라 시스템과 툴사이의 인터페이스의 표준화가 요구되게 되었다. 이런 연유로 형식 명세 언어로 발전하게 되었고 자동화된 명세기반 테스팅(automated specification- based testing)의 연구가 태동하게 되었다. CASE 도구가 점차 발전함에 따라 어렵게만 느껴졌던 형식 명세를 쉽게 접근할 수 있게 되었다. 즉 다이어그램의 표현을 형식 명세로, 그리고 형식 명세에서 테스트 케이스를 생성하는 과정을 이루게 되었다. 1980년대 말에는 산업계 및 학계에서는 점차 프로그램 기반(Code - based) 테스팅의 한계와 형식 언어의 장점을 인식하여 툴지원(tool assisted), 명세기반(spec-based) 테스팅에 관심을 갖게 되었다 [4].

3.3 명세 기반 테스팅의 과정

명세 기반 테스팅의 과정은 보통 3단계로 구성된다. 즉 테스트 케이스의 생성(CREATE) → 실행(EXECUTE) → 평가(EVALUATE) 의 과정이다.

- CREATE: 테스트 케이스를 생성한다.
- EXECUTE: 테스트 케이스를 실행한다.
- EVALUATE :

테스트 케이스의 실행이 테스트 대상 시스템의 구조, 입/출력, 기능들을 모두 만족하게 수행했는지를 평가한다.

<표 1> 테스트 기술 수준 측정 질문서

아 이 템	응 답			
	예	아니오	무응답	코멘트
1. 여러분의 조직에 누군가가 테스트 과정을 담당하고 있는 가?				
2. 테스트 계획 표준을 가지고 있고 사용하고 있는가?				
3. 단위 테스트 표준을 가지고 있고 사용하고 있는가?				
4. 테스트 보고서 표준을 가지고 있고 사용하고 있는가?				
5. 테스트 계획과 실행 프로세스가 전체 소프트웨어 개발과 병행적으로 수행되고 있는가? (즉, 개발이 시작될 때, 테스트도 시작되고, 개발이 종료될 때 테스트도 종료되는가?)				
6. 명세가 올바르게 구현되었는지를 확인하는가?				
7. 명세가 올바르게 구현되었는지 뿐만 아니라 고객/사용자의 기대에 맞게 구현되었는 지를 확인하는가?				
8. 요구사항이나 설계와 같이 내부 개발물의 정확성이나 완전성을 검증하는가?				
9. 테스터가 실행을 위해 소프트웨어 개발 팀에게 결함을 보고하는가?				
10. 테스터는 테스트 계획을 개발하기전에 비지니즈 위험을 인식하고 있는가?				
11. 테스트되려는 각 소프트웨어 시스템에 대해 수립된 테스트 목적을 측정할 수 있는가?				
12. 만약 그렇다면 이런 측정가능한 목적들은 비니지스 위험과 밀접한 관계를 갖고 있는가?				
13. 테스트 기록에서 결함이 발견되거나, 요약되거나 개발과 테스트 과정에서 향상을 위해 사용되는가?				
14. 테스터가 그전 결함의 경험을 기반으로 예측되는 결함을 정의하는가?				
15. 테스트 과정을 향상시키기 위한 과정을 가지고 있는가?				
16. 결함이란 용어를 사용하는가?				
17. 실패가 없는 소프트웨어를 생성하기위해 테스트 과정의 효과성을 평가하는 데이터의 사용이나 요약 기록 등을 하는가?				
18. 테스트 과정의 평가와 계획을 위한 메트릭스를 사용하는가?				
19. 테스터를 훈련시키는 과정을 가지고 있는가?				
20. 자동화된 테스트 툴을 테스트 과정에서 사용하고 있는가?				

3.4 명세 기반 테스팅(Specification-based testing) 의 전략

테스트 케이스 정의는 ANSI/IEEE Standard 829로서 다음과 같이 7가지 부분으로 구성된다.

(1) Test case spec identifier
다른 테스트 케이스와 구별되는 유일한 이름이다.

(2) Test items
이 테스트 케이스가 수행할 행위 혹은 함수의 리스트이다.

(3) Input spec
이 테스트 케이스가 실행하기 위한 입력값 혹은 이름의 리스트이다.

(4) Output spec
이 테스트 케이스의 실행 결과 값에 대한 리스트이다.

(5) Environmental needs
이 테스트 케이스가 수행하는 데 필요한 소프트웨어나 하드웨어이다.

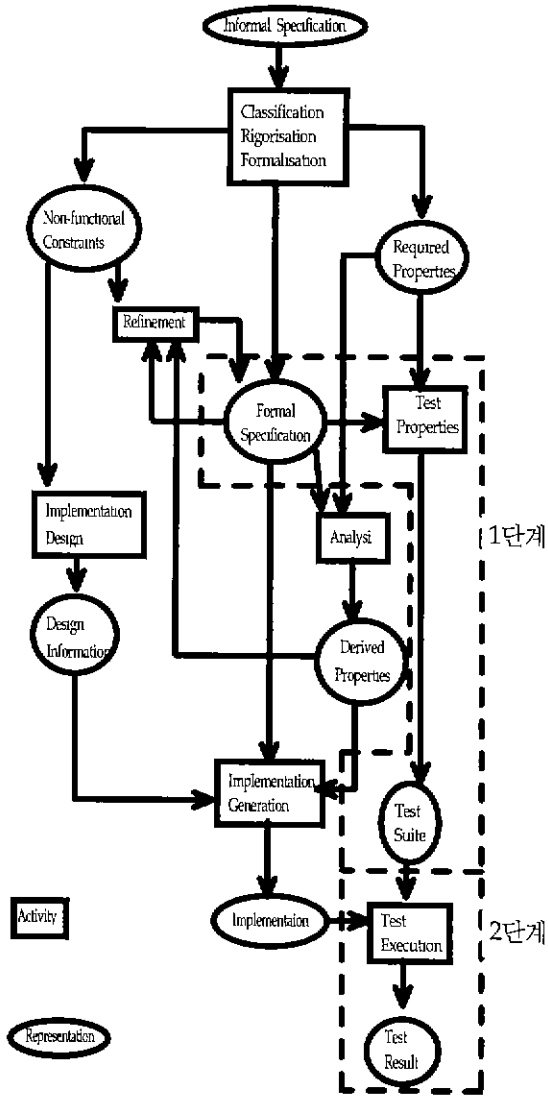
(6) Special procedural requirements
이 테스트 케이스가 실행하는데 필요한 프로시듀어(procedure) 혹은 절차 상의 제한이다

(7) Intercase dependencies
테스트 케이스가 실행되기 전에 반드시 실행해야할 테스트 케이스의 리스트이다.

자동 테스트 케이스 생성기(automated test case generator)에 반드시 필요한 정보는 (1)부터 (4)까지이고 일반적으로 (5)부터 (7)은 배제하는 경우가 많으며 테스트 계획(test plan)이나 테스트 과정(test procedure) 등의 문서화 자료로 만드는 경우가 많다.

3.5 일반적인 형식 방법론에서의 명세 테스팅 위치

일반적으로 형식 방법론(Formal method)으로 소프트웨어를 개발시 명세 테스팅의 위치는 (그림 1)과 같다. 1단계는 형식명세(Formal Spec)에서 테스트 유도(Test Derivation)가 이루어지고 테스트 스위트(Test Suite)를 생성하는 과정이다. 2단계는 테스트 스위트를 통해 테스트 실행(Test Execution)을 하여 테스트 결과(Test Result)를 얻는다.



(그림 1) 일반적인 formal methodology에서의 테스팅

3.6 LOTOS 명세 테스팅

형식 언어중 LOTOS(Language Of Temporal Ordering Specification)는 명세언어로서 형식 기반의 언어이다. 이 언어는 ACT ONE과 같은 추상 자료형을 포함하는 대수적 명세언어이며, CSP와 같은 프로세스 대수의 개념을 갖고 있다. 그 주요 특징은 데이터 타입잉과 행위 모델링을 통합하고 있으며, 비결정적(non-determinism)인 것과 병행성을 제어한다. 또한 OSI 지향적이다[5, 6]. LOTOS 명세의 대상이 되는 응용영역은 주로 프로토콜 시스템이 될 수 있고, 이런 프로토콜 시스템의 특성은 기존의 소프트웨어 시스템과는 <표 2> 와 같은 점에서 다르다.

<표 2> 기존 시스템과 LOTOS 명세 시스템과의 차이점

소프트웨어 시스템 구분 / 항 목	기존의 소프트웨어 시스템	LOTOS명세 소프트웨어시스템
state transition 형태의 명세	initial state-> final state 로서 잘 명세	terminate가 일어나지 않을 수 있음 (reactive 시스템의 명세는 단지 initial state와 final state를 명세하는 것만으로는 적합하지가 않음)
시스템 이름	transformational system이라고 함	reactive system(최종 결과물을 얻는 것 보다 시스템 환경과의 상호작용의 유지에 더 관심을 갖고 있음)
예제 시스템	배치, off-line data processing, numerical package	OS나 프로세스 제어 시스템
기타		-초기화에서의 입력과 종료시의 출력이 제한되어 있지 않고 일부 입력은 중간 출력과정에 의존하기도 한다. -계속되는 behavior를 잘 나타내어야 함

프로토콜 시스템에서의 테스팅은 적합성 테스팅(Conformance Testing)이라고 한다. 여기서 적합성(conformance)이란 시스템이 명세에 따른 구현이 타당한가를 의미하는 것이다. 따라서 적합성 테스팅이란 표준(standard)에서 주어진 요구에 구현이 적합하게 따르고 있는지를 검사한다. 표준이란 OSI 프로토콜에서 외부적으로 관찰가능한 행위(observable behaviour) 명세이고 이런 표준은 TTCN(Tree and Tabular Combined Notation)에서 명세된 표준화된 테스트 스위트(test suite)를 가지고 있다. TTCN는 테스트에 관한 명세화된 언어로 보면 된다. 만약 구현이 이런 테스트 스위트에서의 테스트 케이스에 대해 모두 통과(pass)되면, 시스템은 표준을 따랐음을 나타낸다. 표준은 오직 관찰가능한 행위에 관한 요구사항만 나타난다. 따라서 적합성 테스팅을 실행시 구현의 자세한 사항은 나타나지 않는다. 이전까지는 이런 표준의 개발은 자연언어로 혹은 상태표(state table)로 서술되었다. 따라서 테스트 스위트는 반드시 수작업으로 개발되었다. 문제점은 이런 테스트 스위트가 표준에서 명세된 것처럼 요구사항과 같은 집합을 명세할 수 있는지에 관한 사항이다. 즉 형식화된 검증방법이 없었다. FDTs(Formal Descriptions Techniques)로 표현된 표준 명세는 이전까지의 이런 문제를 해결한다.

즉 형식명세(formal spec)에서 자동적으로 테스트 스위트를 유도한다.

4. 테스팅 툴의 적용

이 장에서는 일반 테스팅 툴의 기능과 그 유형 및 기술, 비용, 기술 수준을 소개한다 [3].

4.1 테스팅 툴의 소개

다음은 현존하는 테스팅 종류와 그 기능을 간략하게 소개한 것이다.

(1) 인수 테스트 기준(Acceptance Test Criteria)

사용자가 인수하기 전에 제작목적에 맞는 반드시 성취되어야 할 시스템 표준을 개발한다.

(2) 경계값 분석(Boundary-Value Analysis)

응용 시스템을 세그멘트 단위로 나누는데 테스팅은 이런 세그멘트의 경계에서만 이루어진다.

(3) 원인-효과 그래프(Cause Effect Graphing)

처리되는 각 이벤트의 효과를 보여주기 위해 사용하는데 발생되는 이벤트는 처리의 결과에 의해 이벤트들로 분류된다.

(4) 체크리스트(Checklist)

기능이나 미리 정해진 영역을 검토하는데 사용하기 위한 질문들의 모임이다.

(5) 코드 비교(Code Comparison)

동일 프로그램의 두 버전사이의 차이점을 인식하는데 사용한다. 목적코드나 소스코드가 그 대상이 될 수 있다.

(6) 컴파일러-기반 분석(Compiler-based Analysis)

컴파일러에 의해서 나오는 진단을 이용하거나 프로그램의 컴파일 시기에 프로그램의 결함을 인식하기 위해 컴파일러에 첨가되는 진단루틴을 이용한다.

(7) 복잡도-기반 메트릭스 테스팅(Complexity-based Metric Testing)

컴퓨터 프로그램의 복잡도나 복잡한 로직를 평가하는 방법의 테스팅 완전성을 인식하는 데 사용되며 아주 높은 예측관련성을 가진 것을 개발하는 통계나 수학을 사용한다.

(8) 확인/검사(Confirmation/Examination)

사용자와 같은 서드파티의 계약에 의해 시스템의 여러 측면의 정확성을 검증한다. 또는 문서의 존재를 검사하기 위해 사용한다.

(9) 제어 흐름 분석(Control Flow Analysis)

로직 문제를 발견하기 위해 프로그램 내에 제어 로직을 검사한다. 이를 위해 프로그램을 그래프 형식으로 표현하는 것이 필요하다.

(10) 정확성 증명(Correctness Proof)

처리의 정확성을 정의하기 위한 가설이나 문장의 집합을 포함한다. 이런 가설들은 응용시스템이 이런 정확성 문장에 따라 처리를 실행하는 가를 결정하기 위해 테스트된다.

(11) 커버리지-기반 메트릭스 테스팅(Coverage-Based Metric Testing)

응용 시스템의 몇 퍼센트가 테스트 과정에 의해 커버되는지를 보여주기 위해 수학적 관련성을 사용한다. 결과 메트릭은 테스트 과정의 효과성을 예측하는데 유용하다.

(12) 자료 사전(Data Dictionary)

자료 원소를 기록하는 문서 툴이며 자료 원소의 속성은 일부 구현에서 시스템의 자료 편집을 확인하는 테스트데이터를 생성할 수 있다.

(13) 자료 흐름 분석(Data Flow Analysis)

프로그램에서 사용되는 데이터가 적절히 정

의되고, 정의된 데이터가 적절히 사용되는 지를 검사하는 방법이다.

(14) 설계-기반 기능 테스팅(Design-based Functional Testing)

응용 시스템 내에 기능 등은 적절한 요구사항이 지원되어야 한다. 이런 과정은 테스트 목적을 위한 이런 설계-기반 기능 테스팅을 인식한다.

(15) 설계 검토(Design Reviews)

검토는 시스템 개발 과정 중에 발생되며, 보통 시스템 개발 방법론에 따라 수행된다. 설계 검토의 주요 목적은 설계 방법론에 따라 준수되었는 지를 확인하는 것이다.

(16) 데스크 검사하기(Desk Checking)

요구 사항, 설계, 혹은 개인에 의해 수행되었던 작업의 검사로서 프로그램의 원안자에 의해 검토된다.

(17) 재해 테스트(Disaster test)

복구 과정을 테스팅하기위한 기초로서 재해를 미리 결정하는 과정이다. 테스트 그룹은 재해를 발생시키고, 시뮬레이트한다.

(18). 에러 추측하기(Error Guessing)

가장 에러 발생 가능한 것을 추측함으로서 미리 결정하여 시스템이 이런 테스트 조건을 조절할 수 있는지를 확인한다.

(19) 실행 명세(Executable Specs)

명세 언어가 테스트 가능한 프로그램으로 컴파일 될 수 있도록 하는 시스템 명세 작성이 가능한 특별한 언어가 요구된다. 컴파일된 명세는 최종 구현 프로그램보다는 덜 자세하고 덜 정확하다. 그러나 명세에 대해 적절한 기능성이나 완전성을 평가하는데는 무리가 없다.

(20) 소모성 테스팅(Exhaustive Testing)

모든 가능한 프로그램의 경로와 조건을 평가하는 충분한 테스팅이다.

(21) 사실-발견하기(Fact Finding)

문서의 정확성의 보증을 제공하거나 테스트의 수행에 필요한 정보로서 정보의 획득과정이 요구된다.

(22) 흐름도(Flowchart)

(23) 인스팩션(Inspections)

잠재적 결함을 인식하기 위해 시스템 개발 생명 주기의 각 단계에 의해 생성되는 매우 구조화된 step-by-step의 검토이다.

(24) 계측도구 삽입(Instrumentation)

미리 결정된 이벤트 에러가 발생한 빈도를 결정하기 위해 카운터나 모니터를 프로그램에 삽입한다.

(25) 통합 테스트 기능(Integrated Test Facility)

테스트 데이터의 소개를 제작 환경으로 허용하는 개념으로, 응용프로그램은 제작환경내에서 동시에 테스트 될 수 있다.

(26) 매핑(Mapping)

테스트되는 동안 컴퓨터 프로그램의 어떤 부분이 실행되는 지를 분석하는 과정이며 프로그램 내에 각 문장이나 루틴의 실행 빈도를 나타낸다.

(27) 모델링(Modeling)

설계 명세가 시스템 목적을 성취했는 지를 결정하기 위해 응용 시스템의 환경이나 그 기능을 시뮬레이팅하는 방법이다.

(28) 병렬 작동(Parallel Operation)

2개의 프로세스간의 차이를 알기 위해 동시의 시간 프레임내에서 옛날버전과 새버전을 실행한다.

(29) 병렬 시뮬레이션(Parallel Simulation)

테스트에 생성되는 결과가 합리적인지를 결정하기 위해 컴퓨터 시스템의 세그멘트에 있어서 좀 덜 정확한 버전을 개발한다.

(30) 피어 검토(Peer Review)

시스템의 개발 주기 관점에서 검토를 한다. 즉 효율성, 효과성, 설계나 구현의 경제성에 반하여 표준, 프로시듀어, 지침 등의 준수성을 검토한다.

(31) 위험 메트릭스(Risk Matrix)

위험의 인식을 통해 제어의 적절성을 테스트한다.

(32) SCARF(System Control Audit Review File)

운영 시스템을 시간 주기를 두고 평가하거나 특별한 엔티티의 오퍼레이션을 비교한다.

(33) 점수주기(Scoring)

테스트되는 응용영역에 문제성 기준을 점수를 주는 방법으로 결정함으로서 응용 시스템의 어느 부분이 반드시 테스트되어야 할 것인가를 결정하는 방법이다. 이런 과정에서는 테스팅 정도를 결정할 수 있는 데 가령 고 위험도의 시스템은 저 위험도의 시스템보다 좀더 테스트되어야 한다는 것을 결정할 수 있다.

(34) 스냅샵(Snapshot)

처리과정중에 미리 정의된 포인트에서 컴퓨터 메모리의 상태를 프린트하는 방법이다.

(35) 기호 실행(Symbolic Execution)

테스트 데이터 없이 프로그램을 테스팅 하는 것으로 프로그램의 기호 실행 결과는 프로그램의 로직의 완전성을 평가하는데 사용하는 문장 표현의 결과를 나타낸다.

(36) 시스템 로그(System Logs)

얼마나 시스템이 잘 수행되었는 지를 결정하기 위한 목적으로 컴퓨터 시스템의 운영 과정중에 얻어지는 정보이다.

(37) 테스트 데이터(Test Data)

응용 시스템의 테스팅의 목적으로 생성되는 시스템 트랜젝션이다.

(38) 테스트 데이터 발생기(Test Data Generator)

테스트 목적을 위해 테스트 데이터를 자동적으로 발생할 수 있는 소프트웨어 시스템이다.

(39) 트래이싱(Tracing)

데이터 처리 등에 의해 컴퓨터 프로그램에서 생성되는 경로의 표현이다.

(40) 유틸리티 프로그램(Utility Programs)

범용 소프트웨어 패키지들을 응용 시스템의 테스팅에 사용 가능하다.

(41) 볼륨 테스팅(Volume Testing)

시스템 기능이 한계점이나 그 능력을 초과했을 경우를 조사하기 위해 미리 정의된 시스템 한계를 테스트하는 테스트 데이터의 특별한 유형의 생성이다.

(41) 워크-스루(Walk-Through)

응용 시스템의 실행 시뮬레이션을 사용하여 테스트 팀에게 응용 시스템을 설명하기 위해 분석가나 프로그래머에게 질문하는 과정이다.

4.2 테스팅 도구의 유형/비용/기술 수준

다음의 <표3>은 앞절에서 살펴본 테스팅 도구의 유형/비용/기술수준 등을 나타낸 것으로 유형은 첫째, 도구가 수동적(manual)으로 작동하는 지, 자동적(automatic)으로 수행하는 지를 나타낸다. 둘째, 정적(static)도구인지 동적(dynamic) 도구인지, 셋째, 구조적(structural) 방법을 사용하는 지 기능적(functional) 방법을 사용하는 지를 나타낸다. 또한 도구의 비용은 고비용 혹은 중간비용 혹은 저 비용인지를 3단계로 구분하였다. 기술수준은 사용자단계, 시스템 단계, 프로그램 단계, 기술적 단계의 4단계중 어느 것을 요구하는 지를 나타내었다.

<표 3> 테스팅 도구의 유형/비용/기술 수준

이 름	유 형	비 용	기 술
인수 테스트 기준	수동 동적 구조적	저비용	사용자 단계
경계값 분석	수동 동적 기능적	저비용	시스템 단계 프로그램 단계
원인-효과 그래핑	수동, 자동 동적 기능적	중간비용	시스템 단계
체크 리스트	수동 정적 구조적, 기능적	저비용	사용자 단계 시스템 단계 프로그램 단계 기술적 단계
코드 비교	자동 동적 구조적	중간비용	프로그램 단계
컴파일러- 기반 분석	자동 정적 구조적	저비용	프로그램 단계
복잡도- 기반 메트릭스 테스팅	자동 정적 구조적	고비용	프로그램 단계 기술적 단계
확인/검사	수동 정적 구조적, 기능적	저비용	시스템 단계
제어 흐름 분석	자동 정적 구조적	중간비용	프로그램 단계
점확성 증명	수동, 자동 정적 구조적, 기능적	고비용	시스템 단계 프로그램 단계
커버리지- 기반 메트릭스 테스팅	자동 정적, 동적 구조적	고비용	프로그램 단계 기술적 단계
자료 사전	자동 동적 구조적, 기능적	저비용	프로그램 단계
자료 흐름 분석	자동 정적 구조적	중간비용	프로그램 단계
설계- 기반 기능 테스팅	수동 동적 기능적	중간비용	시스템 단계 프로그램 단계
설계 검토	수동 정적 구조적	중간비용	시스템 단계
데스크 검사	수동 정적, 동적 구조적, 기능적	저비용	시스템 단계 프로그램 단계
재해 테스트	수동 구조적	중간비용	시스템 단계
에러 추측하기	수동 동적 기능적	저비용	시스템 단계
실행 명세	수동 동적 기능적	고비용	시스템 단계
소모성테스팅	자동 동적 기능적	고비용	프로그램 단계
사실-발견하기	수동 정적 구조적, 기능적	저비용	시스템 단계

이 름	유 형	비 용	기 술
흐름도	수동, 동적 정적 구조적, 기능적	저비용	시스템 단계 프로그램 단계
인스팩션	수동 정적 구조적, 기능적	고비용	시스템 단계
계측도구 삽입	자동 동적 구조적	중간비용	프로그램 단계 기술적 단계
통합 테스트 기능	자동 동적 구조적, 기능적	중간비용	사용자 단계 시스템 단계
매핑	자동 동적 구조적	중간비용	시스템 단계 프로그램 단계
모델링	자동 동적 구조적	고비용	시스템 단계 프로그램 단계
병렬 작동	자동 기능적	고비용	기술적 단계
병렬 시뮬레이션	자동 동적 기능적	고비용	시스템 단계 프로그램 단계
피어 검토	수동 정적 구조적	중간비용	사용자 단계 시스템 단계 프로그램 단계 기술적 단계
위험 메트릭스	수동 정적 기능적	중간비용	사용자 단계
SCARF(System Control Audit Review File)	자동 동적 기능적	고비용	사용자 단계 시스템 단계 프로그램 단계
스코어주기	수동 정적 기능적	저비용	사용자 단계
스냅샵	자동 동적 구조적, 기능적	중간비용	프로그램 단계
기호 실행	자동 정적 구조적	고비용	프로그램 단계
시스템 로그	자동 동적 구조적	중간비용	시스템 단계 프로그램 단계
테스트 데이터	자동 동적 구조적, 기능적	고비용	시스템 단계 프로그램 단계
테스트 데이터 발생기	자동 동적 구조적, 기능적	중간비용	프로그램 단계
트래이싱	수동, 자동 정적, 동적 구조적, 기능적	중간비용	시스템 단계 프로그램 단계
유틸리티 프로그램	자동 동적 구조적, 기능적	중간비용	프로그램 단계
볼륨 테스팅	수동 동적 기능적	중간비용	시스템 단계 프로그램 단계

5. 결 론

본 연구는 2가지 관점에서 서술되었다. 그 하나는 일반적인 테스팅인 프로그램 위주의 테스팅전략이 아닌 명세 기반 테스팅의 관점으로 그 중 형식 명세 테스팅에 관한 사항이었다. 또다른 하나는 일반적인 테스팅툴의 기능 소개와 그들의 유형, 비용 및 기술 수준을 알아보았다.

일반적으로 소프트웨어 테스팅은 소스 코드를 기반으로 하는 프로그램 테스팅과 명세를 기반으로 하는 명세 기반 테스팅으로 구분된다. 그러나 요즘 형식 방법론으로 시스템을 개발하는 관점에서의 명세 기반 테스팅의 연구도 또한 활발하다. 특히 인간의 생명에 관련된 소프트웨어의 테스팅, 가령 항공 통제나 핵 반응 감시등에서의 비용은 일반 소프트웨어 개발보다 더 많은 비용이 요구된다. 따라서 이런 시스템 개발의 경우 형식 명세로 개발하여 자동 소스코드 생성 및 이에 대한 형식 명세 테스팅의 연구가 많이 이루어지고 있는 것이다. 본 연구에서는 일반적인 테스팅 방법론과 형식 명세를(formal spec) 기반으로 하는 형식 명세 테스팅에 대해 알아보았고, 그중 형식 언어중 LOTOS 언어로의 적합성 테스팅 방법에 대해서도 일부 소개하였다.

또한 일반적으로 현업에서 사용하는 테스팅 툴의 기능과 그 유형, 비용 및 기술 수준을 살펴보았다. 우리 나라의 실정에는 소프트웨어 테스팅·자동화의 비용 등을 고려할 때 약간은 수준이 높을 수 있다고 본다. 그러나 이와 같은 테스팅 툴을 필요한 영역에 따라 선택하여 적합하게 활용하는 관점을 지향해야 할것이다.

참고문헌

[1] [Myers] Myers, G. J, The Art of Software Testing. New York: John Wiley & Sons, 1979

[2] [Boris] Boris Beizer, Software Testing Techniques, Vsn Nostrand Reinhold,1990

[3] [Bolognesi] T. Bolognesi and E. Brinksma, Introduction to the ISO Specification language LOTOS, Comput. Networks and ISDN Syst. 14(1), 1987, pp.27-59.

[4] [Robert] Robert M.Poston, Automatic Specification-Based Software testing, IEEE Computer Society Press, 1996

[5] [William] William Perry, Effective Methods for Software Testing, Wiley QED, 1995

[6] [van] van Eijk et.a.l :The Formal Description Techniques LOTOS,North-Holland, Amsterdam, 1989

한 규 정

1986년 중앙대학교 화학과 졸업 (이학사)
1988년 중앙대학교 전자계산학과 대학원 졸업 (이학석사)
1991년 중앙대학교 전자계산학과 대학원 졸업 (공학박사)
1992년-현재 공주교육대학교 컴퓨터 교수
1997년-현재 공주교육대학교 전자계산 소장
관심분야 : 소프트웨어공학(특히, 객체지향 시스템, 테스팅, 형식방법론), 프로토콜 공학(프로토콜 테스팅)

정 연 대

1978년 서강대학교 수학과 졸업 (이학사)
1984년 한국과학기술원 경영과학 이수
1988년 서강대학교 경영대학 경영대학전공 (석사)
1978년-현재 시스템공학연구소 선임, 책임연구원
1992년-1996년 시스템공학연구소 정보베이스 연구실장
1996년-현재 시스템공학연구소 개발자동화 연구실장
관심분야 : 소프트웨어공학(형식방법론, 개발자동화, 객체 지향, 재사용 등), 데이터 베이스