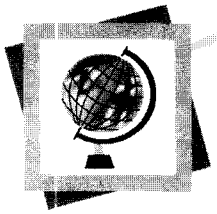
| 특집 05 |



소프트웨어 시험 단계별 자동화 지원 도구

배현섭
(슈어소프트테크(주))

목 차

1. 서 론

소프트웨어의 용도가 확대되고 중요성이 점점 증가함에 따라 품질관리에 대한 요구가 한층 커지고 있다. 소프트웨어 품질관리를 위한 다양한 방법들 중에서 가장 널리 사용되는 기법이 소프트웨어 테스팅 기법이다. 과거의 소프트웨어 테스팅은 대부분 개발자나 품질 관리자에 의해서 수작업으로 진행되었다. 그러나 소프트웨어의 규모가 방대해지고 입출력이 복잡해짐에 따라 소프트웨어 테스팅에 소요되는 시간과 비용이 급격하게 증가하고 있다. 이를 완화하기 위해서 최근 소프트웨어 테스팅 자동화에 대한 관심이 크게 높아지고 있다[1,6].

소프트웨어 테스팅 자동화 도구는 테스팅에 포함되는 다양한 과정-테스트 관리, 소스 코드 리뷰 및 인스펙션, 테스트 설계 및 개발, 테스트 수행 등-을 하드웨어 혹은 소프트웨어적으로 자동화할 수 있도록 지원한다[2,3,4]. 따라서 효과적인 테스팅 자동화 도구의 도입은 소프트웨어 테스팅 과정의 생산성 및 신뢰성 향상에 도움을 줄 수 있고 나아가서 소프트웨어 품질 향상에 기여할 수 있다.

이 논문에서는 현재 상용화된 다양한 테스팅 자동화 도구를 그 특징 및 용도에 따라서 분류하고 대표적인 기능을 소개한다. 논문의 2장에서는 자동화 도구에 대한 분류 기준을 제시하며 3장에서는 분류 기준에 따른 대표적인 도구들을 소개한다. 4장에서는 시험 단계에 따른 자동화 도구 도입 전략을 설명하고 마지막으로 5장에서는 현재 자동화 도구의 한계 및 향후 발전 방향을 제시한다.

2. 테스팅 자동화 도구 분류 기준

이 장에서는 소프트웨어 테스팅 자동화 도구에 대한 분류 기준을 제시한다. 테스팅 자동화 도구에 대한 분류 기준은 테스팅 단계, 테스트 프로세스, 테스팅 방법 등에 따라서 다양하게 나뉠 수 있지만[8,9], 여기서는 테스팅 단계에

따른 분류 기법을 택한다. Fewster와 Graham은 (그림 1)에서 보는 바와 같이 소프트웨어 개발 주기에 따른 테스팅 자동화의 분류 기준을 제시했다[3].

이 논문에서는 Fewster와 Graham의 분류 기준을 다음과 같이 세분화하고 빠진 항목을 추가하여 자동화 도구를 분류하고자 한다.

■ 명세 기반 테스트 설계 도구

소프트웨어에 대한 명세로부터 테스트 프로시저, 테스트 데이터, 테스트 드라이버, 테스트 예상결과 등을 생성하는 도구

■ 코드 기반 테스트 설계 도구

소스 코드로부터 테스트 프로시저, 테스트 데이터, 테스트 드라이버, 테스트 스텁 등을 생성하는 도구

■ 테스트 관리 도구

테스트 계획 수립, 요구 사항 및 버그 추적 관리 등을 지원하는 도구

■ 정적 분석 도구

프로그램을 수행하지 않고 분석하는 도구로서 복잡도 분석 도구를 포함

■ 리뷰 및 인스펙션 도구

소스 코드 및 설계 문서를 분석해서 주어진 가이드라인, 규칙을 검사하고 문제점을 발견하는 도구

■ 커버리지 측정 도구

주어진 테스트 케이스에 의해서 프로그램이 얼마나 많이, 얼마나 샅샅이 테스팅 되었는지 평가하는 도구

■ 동적 분석 도구

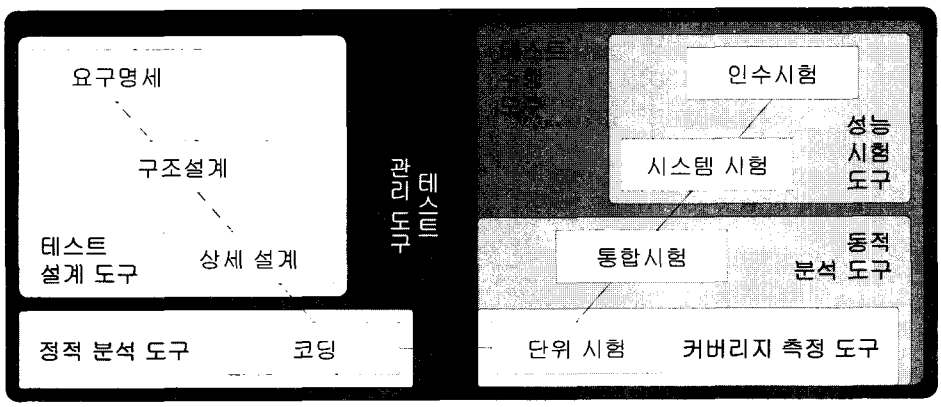
프로그램 수행 중에 시스템의 상태를 평가하는 도구로서 메모리 누출 평가 도구 등이 포함

■ 성능, 로드, 시뮬레이션 시험 도구

시스템 부하를 생성하고 부하에 따른 프로그램의 반응시간, 메모리 사용량 등을 평가하는 도구

■ 테스트 수행 도구

주어진 테스트 케이스를 자동 수행하고 수행 결과를 예상 결과와 비교하는 기능을 제공하는 도구로서, 단위 시험, 통합 시험, 시스템 시험, 인수 시험의 모든 단계에서 사용 가능하며 현재 녹화 후 재수행(capture and replay) 도구가 대표적으로 사용됨



(그림 1) 개발 단계에 따른 테스팅 자동화 도구 분류

IEEE 829에서 제시하는 테스트 문서화 프레임워크에 따르면 테스팅 과정에서 테스트 플랜, 테스트 케이스, 테스트 프로그램, 테스트 로그, 테스트 수행 결과 보고서 등이 생성되어야 한다[5]. 이 때 테스트 케이스는 입력 데이터, 테스트 프로시저, 예상 결과를 포함하고 있어야 하며, 테스트 프로그램은 테스트 드라이버, 테스트 스텁, 테스트 비교기 (test comparator) 등을 포함한다. 다음의 <표 1>은 앞에서 제시한 자동화 도구 분류 기준과 IEEE 829에 명시된 테스트 산출물 간의 대응 관계를 보여준다.

〈표 1〉 테스트 도구 분류와 IEEE 829 대응

분류	IEEE 829 산출물 자동화
명세 기반 테스트 케이스 설계 도구	-테스트 케이스 자동 생성: 입력 데이터, 테스트 프로시저, 예상 결과 자동 생성 가능
코드 기반 테스트 케이스 설계 도구	-테스트 케이스 자동 생성: 입력 데이터, 테스트 프로시저 자동 생성 가능 -테스트 프로그램 자동 생성: 테스트 드라이버, 테스트 스텁, 테스트 비교기 자동 생성 가능
테스트 관리 도구	-테스트 플랜 생성 및 관리 지원
정적 분석 도구	-정적 분석 결과 보고서 자동 생성
리뷰 및 인스펙션 도구	-리뷰 및 인스펙션 결과 보고서 자동 생성
커버리지 측정 도구	-테스트 프로그램 자동 생성: 모니터링을 위한 테스트 탐침 자동 삽입 -테스트 결과 보고서 생성: 커버리지 측정 결과 자동 축적
동적 분석 도구	-테스트 프로그램 자동 생성: 메모리 모니터링을 위한 탐침 자동 삽입 -테스트 결과 보고서 생성: 메모리 테스팅 측정 결과 자동 축적
성능, 로드, 시뮬레이션 도구	-테스트 결과 보고서 생성: 성능 분석 결과 자동 축적
테스트 수행 도구	-테스트 케이스 자동 생성: 입력 데이터, 테스트 프로시저, 예상 결과 자동 생성 가능

3. 테스팅 자동화 도구 소개

이 장에서는 앞 장에서 제시한 분류 기준에 해당하는 대표적인 소프트웨어 테스팅 자동화 도구들을 소개하고 특징을 기술한다[7].

〈표 2〉 명세 기반 테스트 케이스 설계 도구

도구명	특징
Datamacs	-조합(permutation)을 통한 테스트 데이터 생성 지원 -메인프레임 환경 지원
Datatect	-조합(permutation)을 통한 테스트 데이터 생성 지원 -윈도우즈 환경 지원
QualityArchitect	-UML의 시퀀스 다이어그램으로부터 테스트 케이스 생성 -통합 시험 지원
Tau	-SDL, UML 명세로부터 TTCN 형식의 테스트 케이스 생성

앞의 <표 2>에 설명한 도구들은 명세로부터 테스트 케이스를 생성하는 과정을 자동화하는 도구들이다. 이 도구들은 테스트 데이터에 대한 명세를 바탕으로 데이터를 자동 조합(permutation)하는 기능을 제공하거나 시퀀스 다이어그램으로부터 객체 간의 상호작용 시퀀스를 자동 생성하여 통합 시험을 지원할 수 있다.

〈표 3〉 코드 기반 테스트 케이스 설계 도구

도구명	특징
C++Test/Jtest	-클래스의 메소드 단위 시험을 위한 테스트 드라이버 자동 생성 -테스트 데이터는 랜덤 데이터 이용
CodeScroll-API Tester	-C 프로그램: 함수별 시험을 위한 테스트 케이스 및 테스트 프로그램 자동 생성 -C++/Java 프로그램: 클래스의 메소드 시험을 위한 테스트 케이스 및 테스트 프로그램 자동 생성

<표 3>에서 기술한 코드 기반 테스트 케이스 자동 생성 도구는 소스 코드에 대한 분석을 바탕으로 테스트 케이스를 자동 생성하는 도구로서 대부분 C, C++, Java 등 보편적으로 사용되는 언어들을 지원한다. 일반적으로 코드 기반 테스트 케이스 자동 생성 도구들은 테스트 프로시저, 테스트 입력 데이터를 자동 생성하며 테스트 수행을 위한 테스트 프로그램도 자동 생성한다.

〈표 4〉 테스트 관리 도구

도구명	특징
QADirector	-테스트 플랜에 따른 스케줄링 지원 -테스트 케이스 관리 및 추적 지원
TestDirector	-테스트 플랜에 따른 스케줄링 지원 -테스트 케이스 관리 및 추적 지원
ClearQuest	-버그 추적 지원

<표 4>에서 기술한 테스트 관리 도구들은 소프트웨어 개발자, 테스트 엔지니어, 시스템 관리자 간의 정보 공유를 지원하기 위한 도구들로서 테스트 케이스 관리 및 스케줄링을 지원하고 버그 이력을 추적 관리하는 기능을 제공한다.

〈표 5〉 정적 분석 도구

도구명	특징
McCabe QA	-소스 코드 복잡도 분석
Discover	-소스 코드 복잡도 분석
McCabe Reengineer	-코드 구조 분석 및 역공학을 통한 설계 정보 추출
RESORT	-소스 코드 복잡도 분석 -역공학을 통한 설계 정보 추출 및 레거시 코드를 객체 지향 코드로 변환

<표 5>에 기술된 정적 분석 도구는 프로그램을 수행하지 않고 분석하여 복잡도를 분석하거나 역공학 기법을 이용하여 설계 정보를 자동 추출하는 도구들이다. 정적 분석은 좁은 의미에서 테스팅과는 다른 범주로 해석될 수 있으나 이 논문에서는 넓은 의미에서 테스팅 자동화 도구를 분류하였다.

〈표 6〉 리뷰 및 인스펙션 도구

도구명	특징
Review Pro	-소프트웨어 리뷰 프로세스 지원
CodeWizard	-소스 코드에 대한 규칙 및 가이드라인 체크
CodeScroll - Code Inspector	-소스 코드에 대한 규칙 및 가이드라인 체크

리뷰 및 인스펙션 도구 역시 정적 분석 도구와 마찬가지로 프로그램을 직접 수행하지 않고 주어진 규칙을 검사하는 도구다. 리뷰 및 인스펙션은 개발 초기 단계에서 아직 구현이 다 끝나지 않은 상태에서 오류를 검출할 수 있으므로 적은 비용으로 높은 효과를 얻을 수 있다. 또한 테스팅에 의해서 에러가 발견되는 경우에는 그 원인을 파악하기 위한 디버깅이 수반되어야 하는 반면에 리뷰 및 인스펙션에 의해서 발견된 에러는 그 원인을 즉시 수정할 수 있으므로 디버깅도 매우 간단하다는 장점이 있다. 그러나 리뷰 및 인스펙션에 의해서 자동으로 검출할 수 있는 에러는 대부분 프로그램의 의미(semantics)를 배제한 것이므로 테스팅과 인스펙션은 상호 보완적인 관계를 유지하는 것이 바람직하다.

다음의 <표 7>은 커버리지 측정 도구를 기술한다. 커버리지 측정 도구는 그 자체로 테스팅을 진행하는 도구라기 보다는 테스팅 과정에서 보조적으로 사용되는 도구로서 테스팅의 효과를 평가하고 종료 시점을 결정하기 위한 기준이 될 수 있다.

〈표 7〉 커버리지 측정 도구

도구명	특징
McCabe Test	-코드 커버리지 측정 지원
CodeScroll - API Tester	-코드 커버리지 측정 지원

<표 8>에 나타난 동적 분석 도구는 프로그램 수행 중에 발생할 수 있는 비정상적 상황을 모니터링하기 위한 도구로서 대부분 메모리 문제를 체크한다.

〈표 8〉 동적 분석 도구

도구명	특징
BoundsChecker	- 메모리 영역 침범 검사
Purify	- 메모리 누출 모니터링

<표 9>는 성능, 시뮬레이션, 로드 시험 도구를 보여준다. 이 도구들은 시스템에 특정한 형태의 부하를 가하고 그에 따른 소프트웨어의 반응시간 및 메모리 사용량 등을 시험한다.

〈표 9〉 성능, 시뮬레이션, 로드 시험 도구

도구명	특징
LoadRunner	-클라이언트/서버 환경에서 부하 생성 시험
QALoad	-클라이언트/서버 환경에서 부하 생성 시험
e-Load	-웹 환경에서 부하 생성 시험
Performance Studio	-클라이언트/서버 환경에서 부하 생성 시험

<표 10>은 테스트 수행 자동화 도구들을 포함하고 있으며 현재 보편적으로 사용되는 녹화 후 재수행 도구들을 포함한다.

〈표 10〉 테스트 수행 도구

도구명	특징
Silk Test	-녹화 후 재수행 기법을 통한 테스트 수행
QARun	-녹화 후 재수행 기법을 통한 테스트 수행
Robot	-녹화 후 재수행 기법을 통한 테스트 수행
WinRunner	-녹화 후 재수행 기법을 통한 테스트 수행

마지막으로 <표 11>은 이와 같은 기준에 해당하지 않는 특별한 형태의 테스팅 자동화 도구들을 기술한다. 특히, 최근에 임베디드 소프트웨어의 활성화와 더불어 기존 소프트웨어 테스팅 자동화 도구를 임베디드 소프트웨어 테스팅에 적용하고자 하는 시도가 활발하게 진행되고 있다.

〈표 11〉 그 외의 테스팅 자동화 도구

도구명	특징
TestQuestPro	-임베디드 소프트웨어에 대한 녹화 후 재수행 도구
Test RealTime	-임베디드 소프트웨어에 대한 커버리지 모니터링
CodeScroll - Embedded Tester	-임베디드 소프트웨어에 대한 코드 기반 시험 자동화

4. 테스트 자동화 도구 도입 전략

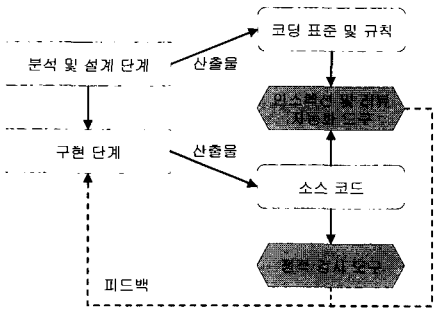
이 장에서는 지금까지 분류하여 소개한 테스팅 자동화 도구들의 효과를 극대화하기 위한 도입 전략을 제시한다. 도입 전략은 코딩 단계, 단위 시험 단계, 통합 시험 단계, 시스템 시험 단계로 나누어 설명한다.

■ 코딩 단계

부분적으로 소프트웨어의 구현이 이루어진 상태이므로 본격적인 시험에 앞서서 인스펙션 및 리뷰 도구를 이용해서 소스 코드의 정적 품질 검사를 수행한다. 그리고 일부 오류 유형에 대해서 정적 분석을 통해서 오류를 검출한다. 다음 (그림 2)에서 보는 바와 같이 인스펙션 및 리뷰를 위한 표준 및 규칙은 분석 및 설계 과정에서 품질 보증 활동을 통해서 구축되며 구현 단계에서 소스 코드가 산출되면 이에 대한 인스펙션 및 정적 검사가 수행된다. 인스펙션 및 정적 검사를 통한 기대 효과는 인스펙션 및 정적 검사의 기준이 되는 코딩 표준, 규칙, 검사 항목에 따라서 크게 달라진다. 이 도구들은 소스 코드 형상관리 도구와 연동하여 사용하는 것이 바람직하다.

인스펙션이나 정적 검사는 자동화를 통해서 생산성을 가장 크게 향상시킬 수 있는 영역 중의

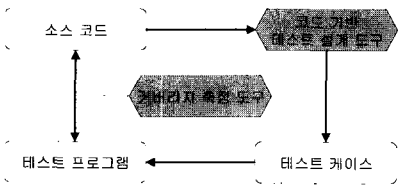
하나다. 이는 인스펙션 및 정적 검사가 반복 작업을 많이 포함하고 있기 때문이다. 반면에 인스펙션과 정적 분석을 통해서 기대할 수 있는 품질 향상 효과는 제한적이다. 일반적으로 정적인 방법을 통해서 향상시킬 수 있는 품질 항목은 유지보수성(maintainability)과 이식성(portability)이 포함되며 부분적으로 신뢰성(reliability) 및 효율성(efficiency)의 향상을 기대할 수 있다.



(그림 2) 코딩 단계의 자동화 도구 도입 전략

■ 단위 시험 단계

단위 시험에서는 코드 기반의 시험 전략이 가장 널리 사용된다. 따라서 단위 시험의 자동화를 위한 효과적인 전략은 코드 기반 테스트 설계 도구를 활용하여 시험 생산성 및 효과를 높일 수 있다. 또한 준비된 테스트 케이스의 효과 및 종료 시점을 결정하기 위해서 소스 커버리지 도구를 활용한다. (그림 3)에 나타난 것처럼 소스 코드로부터 테스트 케이스를 생성하고 이를 수행하는 과정에서 커버리지를 모니터링 한다. 이 과정 역시 소스 코드를 기반으로 진행되므로 코드에 대한 형상관리 도구와 연동되는 것이 바람직하다.

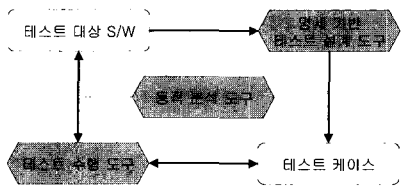


(그림 3) 단위 시험 자동화 도구 도입 전략

단위 시험 자동화에 대한 기대 효과는 커버리지 향상을 통해서 확인할 수 있다. 즉, 단위 시험을 자동화함으로써 수작업으로 단위 시험을 수행할 때에 비해서 짧은 시간 내에 더 많은 영역을 테스팅할 수 있다. 단위 시험의 커버리지를 향상시키려면 다양한 경로를 시험할 수 있는 테스트 데이터를 생성하는 방법이 매우 중요하다. 소스 코드 기반 테스팅 자동화 도구들은 랜덤 데이터 생성, 심볼릭 수행을 통한 데이터 생성, 동적 파티션을 통한 데이터 생성 기법 등을 통해서 테스트 데이터를 자동 생성한다[1,8]. 단위 시험은 소프트웨어의 기능 및 신뢰성 향상효과를 가진다.

■ 통합 시험 단계

단위 시험을 통과한 소스 코드들은 설계를 기반으로 통합이 진행된다. 통합 시험에서는 프로그램 구성 요소 간의 인터페이스 시험을 수행하고 메모리 누출 문제 등에 대해서 동적으로 분석한다. (그림 4)는 통합 시험 단계에서 적용 가능한 자동화 도구 도입 전략을 보여준다. 명세 기반 시험 설계 도구를 활용하여 테스트 케이스를 확보하고 이를 자동 수행하는 도구를 도입함으로써 자동 수행 환경을 갖출 수 있다. 이 과정에서 메모리 등에 대한 동적 분석 도구가 연동될 수 있다. 특기할만한 점은 테스트 수행 도구의 경우에 대부분 녹화 후 재수행 방식을 택하고 있는데 이 경우에 녹화 기법을 통해서 테스트 엔지니어가 직접 테스트 케이스를 만들 수 있다. 따라서 통합 시험을 위한 테스트 케이스는 명세 기반 테스트 설계 도구를 통해서 확



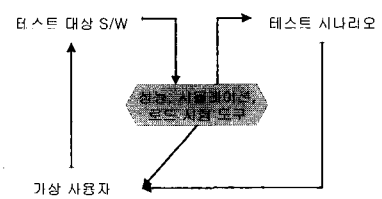
(그림 4) 통합 시험 자동화 도구 도입 전략

보할 수도 있고 녹화 기법을 통해서 테스트 엔지니어가 직접 만들 수도 있다.

통합 시험은 보통 여러번 반복적으로 수행되며 소프트웨어 시험 단계 중 가장 많은 시간을 차지하는 부분 중의 하나다. 따라서 자동화를 통한 생산성 향상 효과를 크게 기대할 수 있다. 그러나 현실적으로 녹화 후 재수행 도구를 사용해서 만들어진 테스트 케이스는 사용자 인터페이스에 대한 변경이 발생하지 않는 경우에 한해서 재사용 가능하다. 즉 자동화 도구를 도입해서 생산성을 높이려면 통합 시험 단계에서 더 이상 GUI에 대한 변경이 발생하지 않아야 한다는 제약사항이 있다. 통합 시험은 소프트웨어 기능 및 신뢰성 향상 효과를 가진다.

■ 시스템 시험 단계

통합 시험이 종료되면 시스템 시험 단계에서는 소프트웨어의 성능 및 로드에 대한 시험을 수행하며 이 과정에서 시뮬레이션이 필요한 경우가 많다. 따라서 성능, 시뮬레이션, 로드 시험 도구가 사용된다. 다음의 (그림 5)는 성능, 시뮬레이션, 로드 시험 도구를 통한 시스템 시험 자동화 전략을 보여준다.



(그림 5) 시스템 시험 자동화 도구 도입 전략

시스템 시험에서 수행되는 성능 및 로드 시험은 일정 수준 이상이 되면 수작업으로 수행하는 것이 불가능해진다. 예를 들어서, 동시 접속자 수가 1천명인 경우에 대한 로드 시험을 하고자 하는 경우에 수작업으로는 불가능하며 자동화 도구가 필수적이다. 즉, 시스템 시험 단계는 자동화 도구의 도입이 가장 효과적인 단계 중의 하나다. 시스템 시험은 소프트웨어 신뢰성과 효율성을 향상시킨다.

■ 기타 자동화

테스트 관리 도구 및 버그 추적 도구는 특정 테스팅 단계에 국한되지 않고 모든 단계에 걸쳐서 활용될 수 있다. 또한 소프트웨어의 특성에 따라서 별도의 시험이 필요한 경우가 있다. 예를 들어, 임베디드 소프트웨어의 경우에는 타겟 보드에서의 시험이 필요하며 웹 기반 소프트웨어는 링크 시험이 필요하고 탑재될 하드웨어 플랫폼이 고정되지 않은 소프트웨어의 경우에는 이식성에 대한 시험이 필요하다. 이와 같은 시험은 아직 자동화 정도가 미진하며 부분적인 자동화가 진행되고 있는 실정이다.

실제 소프트웨어 시험 과정에서는 이 장에서 도식화 한 4가지 그림에 해당하지 않는 독특한 시험 요구사항이 발생하는 경우가 빈번하다. 이 경우에 기존에 상용화 된 자동화 도구를 도입해도 테스팅 요구사항을 해결하지 못하는 경우가 있다. 따라서 테스팅 자동화 도구에 대한 자체 개발 혹은 커스터마이징 필요성이 제기되는 경우가 많다. 그러나 현재는 대부분의 자동화 도구들이 외국에서 개발되었으며 소스 코드를 공개하지 않고 있기 때문에 상용 자동화 도구들을 커스터마이징할 수 있는 방법은 거의 없는 실정이다.

5. 테스트 자동화 도구 발전 방향

이 논문에서는 소프트웨어 테스팅 자동화 도구를 테스팅 단계별로 분류하는 기준을 제시하고 이 기준에 맞추어 현재 상용화 된 대표적인 자동화 도구들을 분류하였다. 소프트웨어 테스팅 자동화 도구들은 테스트 케이스 설계 및 생

성 과정에서 사용되는 도구, 정적 분석용 도구, 테스트 수행 과정에서 사용되는 도구, 그리고 테스트 관리 도구를 포함한다. 또한 이 도구들을 코딩 단계, 단위 시험 단계, 통합 시험 단계, 시스템 시험 단계에서 도입하기 위한 전략을 제시하였다.

효과적인 테스팅 자동화 환경을 구축하려면 테스트 관리 도구, 정적 분석 도구, 테스트 케이스 설계 도구, 테스트 수행 도구가 통합 운용되어야 한다. 그러나 기존의 테스팅 도구들은 개발 회사에 따라서 서로 다른 데이터 구조와 서로 다른 GUI를 가지므로 쉽게 통합되지 못하는 문제점이 있다. 향후 테스팅 자동화 도구들의 통합 운용을 촉진하기 위해서 테스팅 자동화 도구 통합을 위한 프레임워크의 구축이 필수적이다. 즉 개방형 데이터 구조와 GUI를 가지는 방향으로 테스팅 자동화 도구가 발전되어야 할 것이다.

참고문헌

[1] B. Beizer, Software Testing Techniques, 2nd Ed., International Thomson Computer Press, 1990.

[2] E. Dustin, J. Rashka, and J. Paul, Automated Software Testing, Addison Wesley, 1999.

[3] M. Fewster, D. Graham, Software Testing Automation: Effective use of test execution tools, ACM Press, Addison Wesley, 1999.

[4] L. Hayes, The Automated Testing Handbook, 1995.

[5] IEEE 829, Standard for Software Test Documentation, IEEE, 1998.

[6] OVUM, Software Testing Tools, OVUM, 1999.

[7] M. Sowers, "Software Testing Tools Summary", Software Development Technologies Inc. White Paper, 2002.

[8] 배현섭, "CodeScroll™: 코드 기반 소프트웨어 테스트 자동화 도구", 소프트웨어공학회지 15권 3호, 2002.

[9] 배현섭, "코드 기반 분산/병렬 소프트웨어 테스팅 자동화 도구", 소프트웨어공학회지 16권 3호, 2003.

저자약력

배 현 섭

1993년 KAIST 전산학과 학사
1995년 KAIST 전산학과 석사
1999년 KAIST 전산학과 박사
1999년~2000년 한국전자통신연구원(ETRI) 컴퓨터소프트웨어연구소 선임연구원
2000년~현재 슈어소프트테크(주) CTO
E-mail: hsbae@suresofttech.com