에이전트 기반의 객체지향 소프트웨어 테스트 도구인 TAS의 구현 및 분석

(Implementation and Analysis of the Agent based Object-Oriented Software Test Tool, TAS)

최 정 은 † 최 병 주 ††
(Jeongeun Choi) (Byoungju Choi)

요 약 컴퓨터 분야에서 에이전트의 개념은 전자 상거래, 정보 검색과 같은 많은 어플리케이션에 응용되어 중요시되고 있지만, 소프트웨어 테스트 분야에 에이전트의 개념이 적용되는 것은 드문 일이었다. 테스트 에이전트 시스템(TAS)은 에이전트의 개념을 소프트웨어 테스트 분야에 적용한 새로운 도구로, 'User Interface Agent', 'Test Case Selection & Testing Agent', 그리고 'Regression Test Agent'로 구성되어 있다. 이들 세 개의 에이전트들은 각각 지능성을 나타내는 규칙들을 가지고 객체지향 프로세스를 따라 자율적으로 테스트를 진행한다. 이 시스템은 두 가지 측면에서 장점을 가지고 있다. 첫째는 자율적으로 테스트를 진행시켜 테스터의 간섭을 최소화한다는 것이고 둘째는 지능적으로 중복이 없고 일관성 있는 효율적인 테스트케이스를 선택하여 테스트 시간을 감소시키면서 오류 검출 능력은 향상된다는 것이다. 본 논문에서는 사례를 중심으로 실행 과정을 기술하여 TAS를 구성하는 세 개의 에이전트들의 자율적인 행동으로 테스트가 진행되는 것을 보여 TAS가 테스터의 간섭을 최소화한다는 것을 보인다. 그리고 4가지 유형의 실험을 수행하여 테스트 시간의 단축과 오류 검출 효과 향상을 기술한다.

Abstract *The concept of an agent has become important in computer science and has been applied to the number of application domains such as electronic commerce and information retrieval. But, no one has proposed yet in software test. The test agent system applied the concept of an agent to software test is new test tool. It consists of the User Interface Agent, the Test Case Selection & Testing Agent, and the Regression Test Agent. Each of these agents, with their intelligent rules, carry out the tests autonomously by employing the object-oriented test processes. This system has 2 advantages. Firstly, since the tests are carried out autonomously, it minimizes tester interference and secondly, since redundant-free and consistent effective test cases are intellectually selected, the testing time is reduced while the fault detection effectiveness improves. In this paper, by actually showing the testing process being carried out autonomously by the 3 agents that form the TAS, we show that the TAS minimizes tester interference. By also carrying out the 4 different types of experiments on the RE-Rule, CTS-Rule, overall TAS experiment, and the fault-detection effectiveness experiment on the RE-Rule, we show the cut-down on the testing time and improvement in the fault detection effectivity.*

1. 서 론

소프트웨어의 품질을 평가하고 문제점을 발견하기 위한 테스트는 소프트웨어의 생명주기에서 매우 중요하다. 하지만, 단위 테스트부터 시스템 테스트까지 점진적으로 테스트를 할 수 있는 테스트 통합 환경을 가진 테스트 도구는 많지가 않다. 또 테스트 대상에 대해 테스트를 수행하기 위한 계획을 세우고, 테스트 선정 기준에 따라 테스트케이스를 선택하고, 테스트 결과를 분석하는 등의 테스트 프로세스를 따라 테스트를 진행시키는 도구들도 많지가 않다. 자동화된 테스트 도구가 테스트에 도움을 주기는 하지만, 테스트 수행에 테스터의 간섭을 필요로

· 본 연구는 한국과학재단 목적기초연구(과제번호 2000-0-303-02-3) 지원으로 수행되었음
† 학생회원 : 이화여자대학교 컴퓨터학과
982COG25@ewha.ac.kr
†† 종신회원 : 이화여자대학교 컴퓨터학과 교수
bjchoi@ewha.ac.kr
논문접수 : 2000년 9월 27일
심사완료 : 2001년 8월 9일

한다. 그리고 자동화된 테스트 도구로 많은 양의 테스트 케이스가 생성되어 불필요하게 테스트 시간이 많이 걸리게 된다. 테스트 과정에서 적은 비용과 시간으로 가능한 한 많은 에러를 발견하는 것은 중요하다[1,2].

테스터의 간섭을 최소화하고, 테스트 시간을 절약 할 수 있고, 스스로 판단하여 테스트를 수행할 수 있는 테스트 도구가 소프트웨어 개발 현장에 필수적이다. 따라서 우리는 테스트 프로세스를 따라 테스트를 수행하면서 테스터의 간섭을 최소화하고, 단위 테스트부터 시스템 테스트까지 점진적으로 테스트를 수행 할 수 있는 테스트 통합 환경을 제공하고 에이전트의 개념[3,4,5,6]을 적용한 에이전트 기반의 객체지향 소프트웨어 테스트 방안[7,8]을 제안하였다. 즉, 에이전트 기반의 객체지향 소프트웨어 테스트 방안의 구조 및 에이전트로써의 특성, 이 방안으로 제안한 테스트 에이전트 시스템(TAS)을 구성하고 있는 3개의 에이전트 즉, 'User Interface Agent', 'Test Case Selection & Testing Agent', 'Regression Test Agent'의 특성들은 논문[7]에서 제안함으로써, 에이전트 기술을 소프트웨어 테스팅 작업에 접목할 수 있는 이론적 배경을 제시하였다. 본 논문에서는 제안된 에이전트 기반의 객체지향 소프트웨어 테스트 방안을 근거로 테스트 도구인 TAS를 구현하고, 실험을 통하여 TAS의 성능을 평가한다.

TAS는 두 가지 측면에서 장점을 가지고 있다. 첫째는 자율적으로 테스트를 진행시켜 테스터의 간섭을 최소화한다는 것이고 둘째는 지능적으로 중복이 없고 일관성 있는 효율적인 테스트케이스를 선택하여 테스트 시간을 감소시키면서 오류 검출 능력은 향상된다는 것이다. 테스트를 수행하는 데에 있어서 테스트케이스를 선택 것은 테스트 수행 시간과 오류 검출 능력과 관련이 되기 때문에 중요한 문제이다. 따라서 본 논문에서는 중복이 없는 테스트케이스를 선택하는 규칙인 'RE-Rule'의 효과와 일관성 있는 테스트케이스를 선택하는 규칙인 'CTS-Rule'의 효과, 그리고 TAS 전체 시스템의 성능을 분석한 결과를 분석하고, RE-Rule에 의해 선택된 테스트케이스의 오류 검출 능력을 실험한 결과를 분석한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 개발되어 있는 자동화 도구를 기술하고, 3장에서는 TAS의 이론적인 바탕을 기술한다. 4장에서는 TAS가 가지는 에이전트의 특성인 자율성을 Prototype을 통해 분석하고, 5장에서는 실험 결과를 기술한다. 마지막으로 6장에서 결론을 제시한다.

2. 기존 테스트 도구

기존 테스트 도구들은 그림 1에서처럼 테스트 프로세스를 관리하는 도구, 테스트 수행에 테스트 기법을 적용하는 도구, 테스트 프로세스 관리와 테스트 기법을 적용하는 도구로 나눌 수 있다.

타입 A의 테스트 프로세스를 관리하는 도구(QADirector[9,10]등)는 테스트 계획 단계부터 테스트케이스 선정, 테스트 수행, 테스트 결과 분석 과정을 자동화 한 도구이다. 그러나, 이 도구들은 특정 테스트 기법에 따라 테스트를 수행하는 기능은 독립적으로 가지고 있지 않다. 또, 테스트 프로세스를 관리하는 것도 단계별로 테스터의 간섭에 의해 진행된다. 따라서 테스터의 간섭 없이 자율적으로 테스트 프로세스를 진행시키는 도구가 필요하다. 그림 1의 타입 B는 테스트케이스 선정 기법을 적용하는 도구들이다. 이러한 도구들은 단위 테스트, 통합 테스트, 시스템 테스트 등의 전체 테스트 레벨에 대한 테스트케이스 선정 및 실행을 위한 도구들(VisionSoft/TEST[9,11] 등)과 단위 테스트, 모듈 테스트, 클래스 테스트, 또는 시스템 테스트 등 한 개의 레벨에 대한 테스트 도구들(Proteum[9,12,13] 등)이 있다.
그림 1의 타입 C에 속하는 도구들은 테스트 프로세스 관리와 테스트 기법을 적용하는 도구들이다. 이 도구들에는 테스트 프로세스를 관리하면서 모든 테스트 레벨에 대해 테스트를 수행하는 도구(TestDirector[9,13]등)이거나, 테스트 프로세스를 관리하면서 한 레벨에 대한 테스트를 하는 도구들(Panorama[9,15,16,17,18,19]등)이다. 타입 B와 타입 C의 도구들 역시 테스트 진행에 테스터의 간섭이 필수적이다.

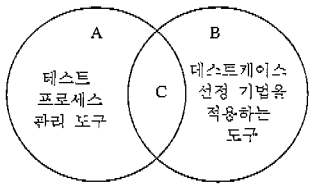


그림 1 기존 테스트 도구 분류

우리가 제안한 TAS는 테스트 프로세스를 관리하고, 단위 테스트, 통합 테스트, 시스템 테스트까지 테스트를 수행 할 수 있는 테스트 통합 환경을 제공한다. 또, 테스터의 역할을 최소화하여 자율적으로 테스트가 진행될 수 있는 테스트 도구이다. 표 1은 기존의 테스트 도구들의 타입과 TAS를 비교한 것이다.

표 1 기존 테스트 도구와 TAS 비교

	Type A	Type B	Type C	TAS
테스트 프로세스 관리 기능	○	×	○	○
특정 테스트 기법 적용	×	○	○	○
테스트 레벨에 따른 테스트 통합 환경 제공	×	△	×	○
테스트 진행 방법	수동적	수동적	수동적	능동적 자율적
테스터가 테스트 작업에 소비하는 시간	많다	많다	보통	적다

소프트웨어 테스트 분야에 에이전트의 개념을 도입하면, 테스트의 작업을 나누어 에이전트화 하여, 에이전트 각각이 독립적으로 수행이 되어, 분산환경에서도 쉽게 테스트를 진행 할 수 있게 된다. 또, 에이전트의 지능성을 가짐으로써, 테스트 수행을 효율적으로 진행 할 수 있다. 즉, 테스트 정보 입력, 테스트 항목 선택, 테스트 케이스 선택, 테스트, 리그래션 테스트로 진행이 되는 테스트 작업을 밀접한 관계가 있는 과정들끼리 묶어 이를 에이전트화 하면, 독립적으로 각자의 기능을 수행하면서, 서로간의 통신을 통해 원활한 테스트 작업을 진행시킬 수 있다. 따라서, TAS는 자율적으로 테스트를 진행시키면서, 지능성을 나타내 주는 여러 규칙들로 효율적인 테스트케이스를 선택하면서, 테스트 시간을 감소시키는 도구이다.

3. 테스트 에이전트 시스템(TAS)

테스트 에이전트 시스템인 TAS는 에이전트의 특성인 자율성, 사회성, 지능성을 갖추고 있어 기존의 테스트 도구에 비하여 테스터의 작업을 대행해 주는 새로운 개념의 테스트 도구이다. TAS의 구성은 그림 2와 같이 테스터와 정보를 주고 받는 "User Interface Agent", 중복이 없고 일관성 있는 테스트케이스를 선택하고 테스트를 수행하는 "Test Case Selection & Testing Agent", 그리고 리그래션 테스트를 담당하는 "Regression Test Agent"인 3개의 에이전트로 이루어 졌다. TAS에 대한 자세한 이론적인 내용은 논문[7]에 기술되어 있고 이 장에서는 TAS의 핵심 내용을 기술한다.

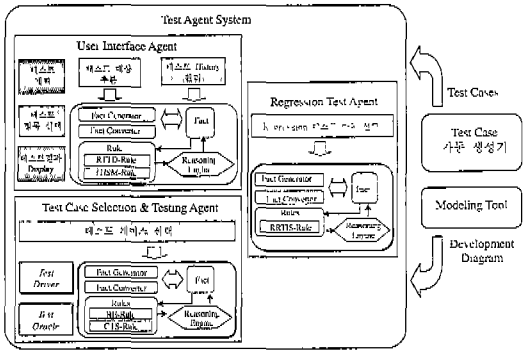


그림 2 TAS의 구성도

3.1 User Interface Agent

User Interface Agent는 테스터와 TAS와의 인터페이스 역할을 하는 에이전트이다. 이 에이전트의 구조는 그림 2와 같이 테스트 대상을 추론하여 테스트를 계획하는 부분, 테스트 대상이 리그래션 테스트인지를 추론하는 부분, 테스트 History의 메모리 관리를 위해 메모리 상황을 추론하는 부분과 테스트 결과를 화면으로 보여 주는 부분, RTTD Rule과 THSM-Rule을 적용하기 위해 필요한 Fact들을 만들고, 변환시켜 주는 부분으로 구성되어 있다. User Interface Agent는 리그래션 테스트 대상 판단규칙(RTTD-Rule)과 테스트 히스토리 크기 관리 규칙(THSM-Rule)로 지능성을 나타낸다. RTTD-Rule은 테스트 대상이 테스트 History로부터 리그래션 테스트 대상인지를 판단하는 규칙이고, THSM-Rule은 Test History 크기를 관리하는 규칙이다. 이 규칙들을 술어 논리[20]로 표현한 것은 다음과 같다.

<RTTD-Rule>

▣ ∀ x, y, Test-target (x) ∧ History(y) ∧ Member(x, History(y)) ⊃ Reg-target(x)

▣ ∀ x, y, Test-target(x) ∧ History(y) ∧ ¬ Member(x, History(y)) ⊃ NReg-target(x)

<TIISM-Rule>

▣ ∀ x, Occurrence-order1 (x) ⊃ Due(x, 30)

▣ ∀ x, Occurrence-order2 (x) ⊃ Due(x, 20)

▣ ∀ x, Occurrence-order 3(x) ⊃ Due(x, 10)

3.2 Test Case Selection & Testing Agent

Test Case Selection & Testing Agent는 수많은 테스트케이스들 가운데 중복이 없고 일관성 있는 테스트케이스를 선택하여 테스트를 수행하는 것이 목적이다. Test Case Selection & Testing Agent는 그림 2와 같이 테스트케이스를 선택하는 부분, 테스트를 수행하는 Test Driver, 수행 후 테스트 결과를 테스트 예상 결과와 비교하는 Test Oracle, RE-Rule과 CTS-Rule을 적

용하기 위해 필요한 Fact들을 만들고, 변환시켜 주는 부분으로 구성되어 있다. Test Case Selection & Testing Agent는 중복 제거 규칙(RE-Rule)과 일관성 있는 테스트케이스 선택 규칙(CTS-Rule)을 통하여 중복됨이 없고 일관성 있는 테스트케이스를 지능적으로 선택한다. RE-Rule은 메소드 순서 형태의 테스트케이스들에서 중복된 테스트케이스를 제거하는 규칙이고, CTS Rule은 하나의 테스트케이스가 여러 개의 메소드로 이루어졌을 때, 각 메소드의 테스트 데이타를 일관성 있게 선택하는 규칙이다. 이 규칙들을 술어 논리[20]로 표현한 것은 다음과 같다.

<RE-Rule>

▣∀ x, y, (Type1-testcase1(x)∧Type1-testcase2 (y)∧Subset(x, y))
⊃Redundancy(x, y)

▣∀ x, y, (Type1-testcase1(x)∧Type1-testcase2 (y)∧Subset(y, x))
⊃ReverseRedundancy(y, x)

▣∀ x, y, (Type1-testcase1(x)∧Type1-testcase2(y)∧ ¬(Subset(x, y))
∧ ¬(Subset(y, x)))⊃NRedundancy-testcase(x)

<CTS-Rule>

▣∀ x, y, (Type2-testcase1(x)∧Type2-testcase2(y)
∧Equal(x, First(y))∧Subset(x, y))⊃Consistency-testcase(x, y)

▣∀ x, y, (Type2-testcase1(x)∧Type2-testcase2(y)
∧Member(x, Rest(y))∧Subset(x, y))⊃Consistency-testcase(x, y)

▣∀ x, y, (Type2-testcase1(x)∧Type2-testcase2(y)
∧Equal(y, First(x))∧Subset(y, x))⊃Consistency-testcase(x, y)

▣∀ x, y, (Type2-testcase1(x)∧Type2-testcase2(y)
∧Member(y, Rest(x))∧Subset(y, x))⊃Consistency-testcase(x, y)

3.3 Regression Test Agent

Regression Test Agent는 리그래션 테스트 수행을 위해 리그래션 테스트 항목을 선택하는 기능을 가진 에이전트로서 선택된 리그래션 테스트 항목과 관련된 모든 테스트 항목에 대한 리그래션 테스트를 수행하는 것이 목적이다. 이 에이전트는 그림 2와 같이 리그래션 테스트 항목을 선택하는 부분, 리그래션 테스트 관련 항목 찾는 규칙이 수행되는 추론 엔진부분, RRTIS-Rule을 적용하기 위해 필요한 Fact들을 만들고, 변환시켜 주는 부분으로 구성되어 있다. Regression Test Agent의 지능성을 나타내는 RRTIS-Rule은 리그래션 테스트의 관련 항목을 찾는 규칙(Regression test Related Test Item Search Rule)이다. 좀 더 자세한 내용을 논문[7]을 참고한다. 이 규칙을 술어 논리[20]로 표현한 것은 다음과 같다.

<RRTIS-Rule>

Intra-method test level, Class-test level

▣∀ x, y, (Regtestitem (x)∧Relatedlist(y)∧Subset(x, Relatedlist(y)))
⊃Related-regtestitem(y)

▣∀ x, y, (Regtestitem (x)∧Relatedlist(y)∧Subset(Relatedlist(y), x))
⊃Related-regtestitem(y)

Inter-class test level-inheritance

▣∀ x, y, z, (Regtestitem(x)∧Related-regtestitem(y)∧Relatedlist(z)
∧Equal(Last(z),Seconedelement(y))⊃Related-regtestitem(z)

Inter-class test level-association, Component integration-test level

▣∀ x, y, z, (Regtestitem (x)∧Related-regtestitem(y)∧Relatedlist(z)
∧Equal(First(z), Last(y))⊃Related-regtestitem(z)

System-test level

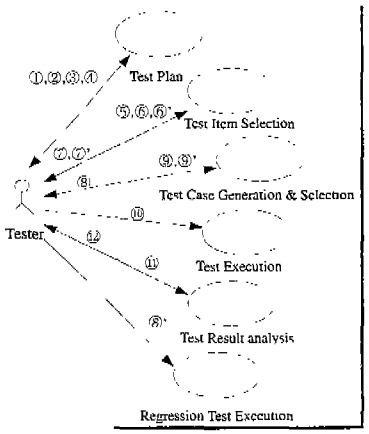
▣∀ x, y, z, (Regtestitem (x)∧Related-regtestitem(y)∧Relatedlist(z)
∧Equal(Fourthelement(z), Last(y))⊃Related-regtestitem(z)

4. TAS의 자율성 분석

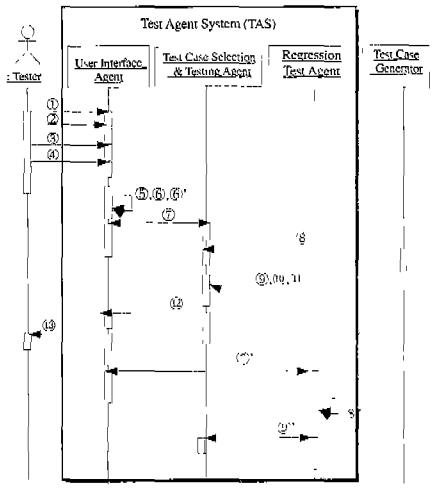
TAS는 자율적으로 테스트를 진행 시켜 테스터의 간섭을 최소화한다는 장점을 가지고 있다. 이러한 장점을 설명하기 위해 이장에서는 보통의 테스트 도구를 사용한 테스트 작업과 TAS를 통한 테스트 작업을 비교 설명하고 Prototype을 통해 TAS의 자율성을 분석한다.

소프트웨어 테스트는 테스트 계획을 세우고, 테스트 항목을 선택하고, 테스트케이스를 생성하고, 테스트를 실행하고, 테스트 결과를 분석하는 작업으로 구성한다. 리그래션 테스트의 경우는 리그래션 테스트케이스를 선택하여 테스트를 실행한다. 그림 3은 보통의 테스트 도구를 사용할 경우와 TAS를 사용하여 테스트 작업을 하였을 때의 진행 흐름을 도식화 한 것이다. 그림 3 (a)에서 ⑤ ~ ⑪ 작업이 테스터에 의해 진행되는 것이 그림 3 (b)에서는 TAS 안에서 자율적으로 진행된다. 일반적인 테스트 도구가 수동적인 것에 비해 TAS는 자율성을 가짐으로써 능동적으로 테스트 작업이 진행하게 된다. 이를 위해 TAS는 수동 동작이나 내부 상태 변화 등에 대한 제어권도 가지고 있다.

TAS는 Windows환경에서 JDK1.2.1(Java Development Kit)로 구현하였고, 에이전트의 지능성을 나타내는 User Interface Agent의 *'RTTD-Rule'*과 *'THSM-Rule'*, Test Case Selection & Testing Agent에서의 *'RE-Rule'*과 *'CTS-Rule'*, 그리고 Regression Test Agent의 *'RRTIS-Rule'*은 자바와 연동이 용이한



(a) 보통의 테스트 도구를 사용한 테스트 작업



(b) TAS를 통한 테스트 작업

Items

① Input Test Date, Tester Name
② Input Test Plan, Development diagram, Test target
③ Display Test Agent and Select Execution Mode ④ Select Test Level
⑤ Reason whether target is regression test target (RTTD-Rule) ⑥ Select test item
⑥' Select regression test item ⑦ Communication ⑦' Communication ⑧ Test Cases
⑧' Search for related regression test item(RRTIS-Rule)
⑨ Select Test Case(RE-Rule, CTS-Rule) ⑨' Communication
⑩ Execute testing (Test Driver) ⑪ Compare test result(Test Oracle)
⑫ Test Result ⑬ Display test result

그림 3 테스트 작업 비교

JESS4.4 (Java Expert System Shell) 전문가 시스템을 이용하여 구현하였다.

테스터 행동

TAS를 시작하면 테스터는 현재 날짜와 테스터의 이름을 입력한다. 이것은 그림 3에서 ①에 해당하는 것이다. 또, 테스터는 그림 3의 ②에 해당하는 것인 테스트에 필요한 정보(테스트 계획서, 개발 다이어그램)와 테스트 대상을 입력한다. 테스트 대상은 크게 2가지 형태가 있을 수 있다. 하나는 객체지향 프로그램이고, 다른 하나는 컴포넌트 기반 프로그램이다. TAS는 2가지 형태의 프로그램에 대한 테스트를 O-O Test[21], CBD Test[22]로 나누어 하도록 설계하였다. 현재 구현된 TAS는 객체지향 프로그램만을 테스트 대상으로 한다. 테스트케이스의 형태를 알기 위하여 테스트 계획서가 필요하고, 테스트 항목간의 소속을 알기 위해 개발 다이어그램이 필요하다. 그림 4는 테스트 계획서, 테스트 대상, 개발 다이어그램의 화일을 선택한 후의 화면이다. 테스터는 필요한 정보를 입력 후 테스트 진행을 위해 그림 4에서 Next 버튼을 누른다. 이때 Exit 버튼은 테스트 에이전트 시스템을 종료하는 것이다.

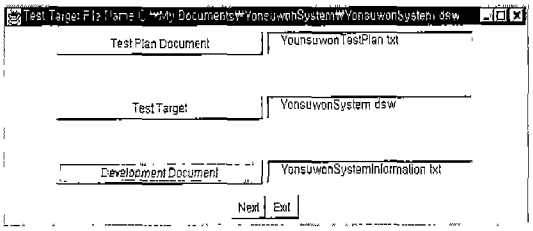


그림 4 테스트 정보 입력 화면

테스트가 행하는 일로는 테스터에 필요한 정보를 입력하는 것 외에 그림 3의 ③에 해당하는 것인 테스트 진행 과정을 Display하는 방법과 진행 방법을 결정하는 Mode를 설정한다. 이것은 그림 5와 같은 화면이다. Display방법은 테스트 수행 과정의 화면을 전부(ALL) 다 볼 것인지 기본(Basic)으로 할 것인지를 선택한다.

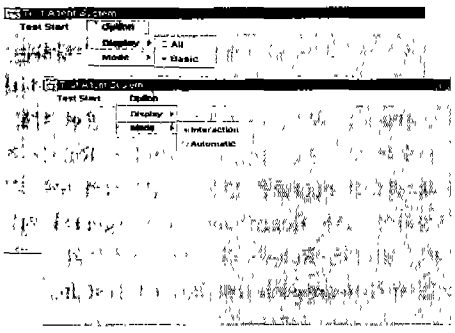


그림 5 Display & Mode 설정 화면

Mode설정은 테스터의 간섭 여부를 설정하는 것으로 전혀 간섭을 하지 않고 자동적(Automatic)으로 수행 할 것인지, 한 테스트 레벨에 대한 테스트가 끝나면 그 결과를 확인하고 다음 레벨을 수행하게 테스터가 간섭(Interaction)한 것인지를 선택한다.

테스터는 그림 3의 ④에 해당하는 것으로 테스트 레벨을 선택한다. 그림 6의 왼쪽 화면은 테스트를 객체지향 프로그램을 할 지 컴포넌트 기반 프로그램을 할 지 선택하는 화면이다. 위에서 기술하였듯이 객체지향 프로그램을 테스트 대상으로 하는 것은 O-O Test를 선택하고, 컴포넌트 기반 프로그램을 테스트 대상으로 하면 CBD Test를 선택한다. 그림 6의 오른쪽 화면은 객체지향 프로그램을 테스트하는 O-O Test 버튼을 선택 후 객체지향 테스트 프로세스의 테스트 레벨이 나온 화면이다. 테스트 레벨은 Unit(Component Test), Integration Test, System Test로 나누어 있다. Unit(Component Test)는 단위 메소드 테스트, 클래스 테스트, 클래스 통합 테스트를 하는 것이고, Integration Test는 컴포넌트 통합 테스트를 하는 것이다. 이 화면에서 테스트 레벨을 테스터가 선택하면, TAS는 테스트 계획을 세우고 테스트 항목 선택, 효율적인 테스트케이스 선택, 테스트 결과 분석하는 과정을 자율적으로 수행한다.

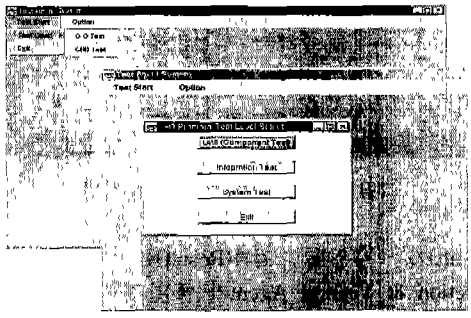


그림 6 테스트 레벨 선택 화면

TAS 행동

그림 6에서 테스터가 테스트 레벨을 선택하면 그림 3의 ⑤에 해당되는 것으로 테스트하려는 대상이 리그래션 테스트 항목인지 아닌지를 User Interface Agent가 RTTD-Rule에 의해 추론한다. 추론된 결과에 따라 리그래션 테스트가 아니면 그림 3의 ⑥에 따라 테스트 항목을 선택하고 리그래션 테스트이면 그림 3의 ⑥'에 따라 리그래션 테스트 항목을 선택한다. 테스트 항목 선정은 테스터가 간섭을 하여 일부분을 선택 할 수도 있고 TAS가 자율적으로 선택 할 수도 있다. 그림 7의 테스트 항목 선택 화면에서 Select 버튼은 테스터가 테스트 할 항목을 왼쪽 창에서 선택하는 것이고, Select All 버튼은 왼쪽 창에 표시된 테스트 항목 모두가 테스트 할 항목이 된다. 테스터가 아무 반응을 안 보이면 자동적으로 Select All이 되어 표시된 모든 테스트 항목이 테스트 할 항목이 된다.

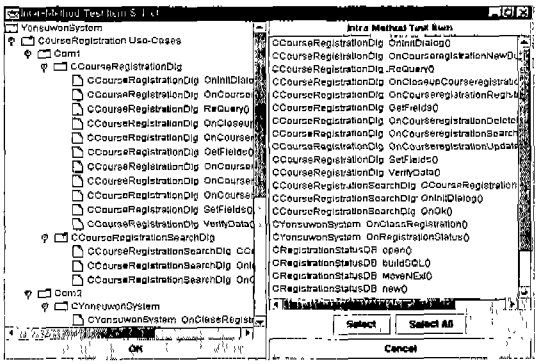


그림 7 테스트 항목 선택 화면

그림 8은 그림 3에서 ⑥'에 해당하는 리그래션 테스트 항목을 선택하는 화면이다. 그림 8에서 리그래션 테스트 항목을 선택하면, Regression Test Agent가 그림 3의 ⑧'에 해당하는 RRTIS-Rule에 의해 리그래션 테스트 항목과 관련 있는 항목을 추론한다. 그림 9는 RRTIS-Rule에 의해 추론되어진 리그래션 테스트 관련 항목들이다. 그림 10은 테스트가 진행되는 과정을 나타내 주는 화면이다. 이는 그림 3의 ⑦,⑦',⑨'에 해당하는 에이전트 들간의 서로 통신하는 모습과 그림 3의 ⑤~⑪에 해당하는 작업을 테스터의 간섭 없이 자율적으로 진행되는 과정을 보인 것이다.

그림 7에서 테스터나 User Interface Agent에 의해 테스트 항목이 선택되면, Test Case Selection & Testing Agent는 그림 3의 ⑧처럼 TAS의 테스트케이

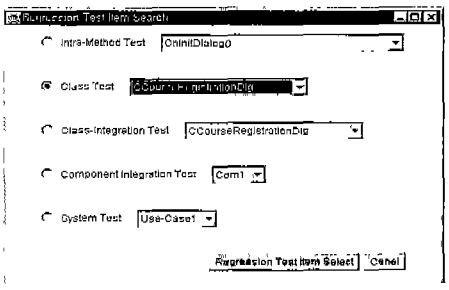


그림 8 리그래션 테스트 항목 선택 화면

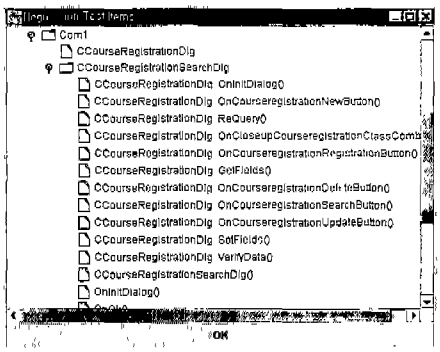


그림 9 RRTIS-Rule에 의해 추론 된 테스트 항목 결과 화면

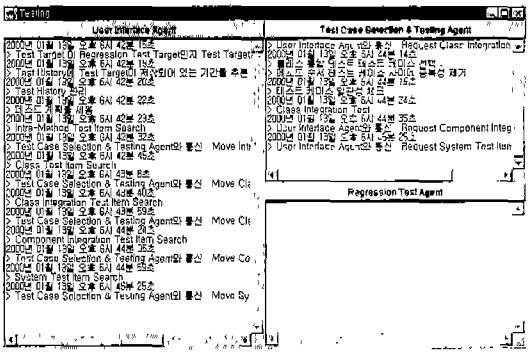


그림 10 테스트 진행 화면

스 생성기로부터 테스트케이스 풀을 입력받고, 그림 3의 ⑨에 해당하는 것인 RE-Rule과 CTS-Rule에 의해 중복이 없고 일관성 있는 테스트케이스를 추론한다. 추론된 테스트케이스로 Test Case Selection & Testing Agent는 그림 3의 ⑩에 해당하는 작업을 수행한다. 그리고, Test Case Selection & Testing Agent는 ⑩의 수행결과로 그림 3의 ⑪에 해당하는 테스트 결과를 분석하는 일을 한다. 이로부터 나온 결과가 그림 3의 ⑫인 테스트 결과이다.

그림 11, 그림 12는 그림 3의 ⑬에 해당하는 것으로 테스트 결과 화면이다. 결과 화면에는 단위 메소드 테스트인 경우, 그림 11과 같이 테스트 항목, 테스트케이스 수, 테스트케이스, 테스트케이스의 품질 측정치(Measure), 결과(Result)가 나온다. 클래스 테스트, 클래스 통합 테스트, 컴포넌트 통합 테스트 결과 화면에는 그림 12와 같이 테스트 항목, 테스트케이스 수, 메소드 순서 형태의 테스트케이스, 테스트 데이타 형태의 테스트케이스, 테스트케이스의 품질 측정치(Measure), 결과

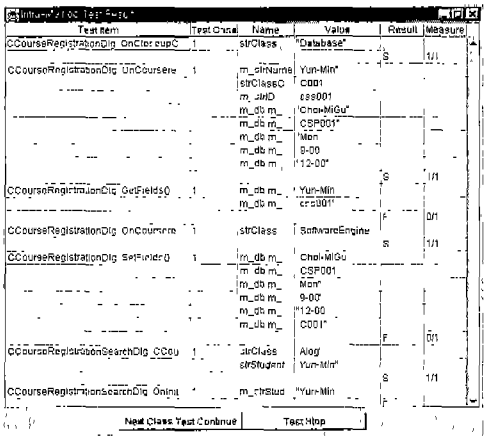


그림 11 클래스 테스트 결과 화면

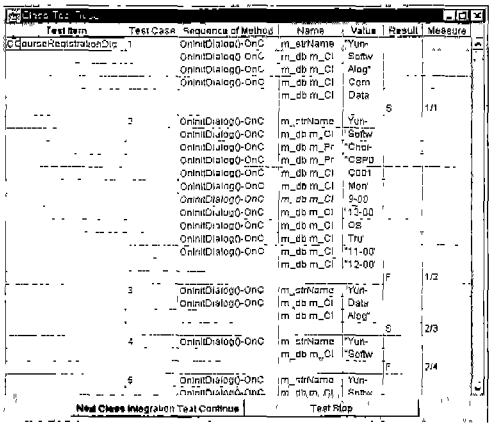


그림 12 클래스 테스트 결과 화면

(Result)가 나온다. 결과(Result)는 Test Case Selection & Testing Agent가 예상 결과와 테스트 결과를 비교해서 테스트 수행 결과가 같으면 S로 틀리면 F로 표시된다. 테스트케이스 품질 측정치(Measure)는 테스트 항목 당 총 테스트케이스 수를 Result가 S인 테스트케이스 수로 나눈 수이다.

5. 실험에 의한 TAS 분석

TAS가 가지는 두 가지 장점 중 자율적으로 테스트를 수행해 테스터의 간섭을 최소화한다는 것은 Prototype을 통해 4장에서 분석하였고, 두 번째 장점인 중복이 없고 일관성 있는 효율적인 테스트케이스를 지능적으로 선택하여 테스트 시간을 감소시키면서 오류 검출 능력도 향상된다는 것을 분석하기 위해 이 장에서

는 4가지 실험에 대한 결과를 기술한다.

TAS는 객체지향 테스트 프로세스[21]를 따라 테스터의 일을 대행해 주고, 테스터의 간섭을 최소화 시켜준다. 또한, 이 시스템의 Test Case Selection & Testing Agent는 RE-Rule과 CTS-Rule를 통하여 자동 생성된 많은 양의 테스트케이스에서 중복이 없고 일관성 있는 테스트케이스를 지능적으로 선택함으로써 테스트 시간을 단축시킨다. 3가지의 객체 지향 프로그램에 대해 TAS가 테스트 시간을 단축시킨다는 것을 증명하기 위한 다음 3가지 유형의 실험을 하였다.

- RE-Rule에 의한 실험
- CTS-Rule에 의한 실험
- TAS에 의한 전체 실험

TAS는 RE-Rule에 의해 선택된 중복이 없는 테스트케이스들에 대해 CTS-Rule에 의해 일관성 있는 테스트케이스를 선택한다. 이때 TAS에서 RE-Rule에 의해 제거되는 테스트케이스들이 오류 발견에 보탬이 되지 않는 쓸모 없는 테스트케이스라는 것을 보여주는 실험을 수행하였다. RE-Rule을 적용하지 않은 테스트케이스들이 오류를 검출하는 수와 RE-Rule에 의해 선택되는 테스트케이스들의 오류를 검출하는 수를 비교하여 다음과 같은 실험을 하였다.

- RE-Rule과 오류 검출 효과(RE-Rule and fault detection effect)

TAS 시스템은 객체지향 프로그램을 테스트 대상으로 하기 때문에, RE-Rule과 CTS-Rule은 클래스 테스트 레벨, 클래스 통합 테스트 레벨, 그리고 컴포넌트 통합 테스트 레벨 등 모든 객체지향 테스트 레벨에 대한 테스트 케이스를 선택할 때 적용이 된다. 본 실험은 클래스 테스트 레벨에 한정하여 실험을 수행하였다.

5.1 테스트 대상

테스트 대상은 프로그램 특성에 따라 기본적인 프로그램, DB를 연동하는 객체지향 프로그램, 분산 객체지향 프로그램의 세 가지 경우에 대한 프로그램을 선정하여 실행하였다. 선정된 프로그램 및 내용은 다음과 같다.

- **기본적인 프로그램 : Stack Program**

Stack C++ Program은 Push, Pop을 포함한 Stack 프로그램이다.

- **DB를 연동하는 객체지향 프로그램 : 연수원 관리 프로그램**

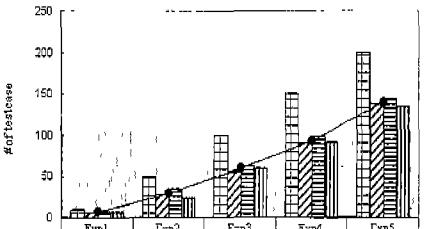
연수원 관리 C++ 프로그램은 연수생의 교과과정 등록, 강사들에 의한 교육과정선정, 연수생, 강사의 신상명세 데이타를 검색 관리하는 프로그램이다.

- **분산 객체지향 프로그램 : Communication Program**

Communication C++ program은 분산되어 있는 다른 컴퓨터와의 통신을 지원하는 프로그램이다.

5.2 RE-Rule에 의한 실험

그림 2의 테스트케이스 자동 생성기[1]에 의해 생성된 200개의 테스트케이스로부터 테스트케이스 수에 따라 5번의 실험을 하였다. 실험 1, 2, 3, 4, 5는 각각 테스트케이스의 개수를 10개, 50개, 100개, 150개, 200개로 하였다. 그림 13은 각 실험별로 3개의 각 프로그램에서 RE-Rule에 의해 선택되는 테스트케이스의 수를 막대그래프로 표현하였다. 3개의 프로그램에서 선택된 테스트케이스 수의 평균은 꺾은선 그래프로 나타내었다.



	Exp1	Exp2	Exp3	Exp4	Exp5
Overall number of test cases	10	50	100	150	200
Stack Program	5	28	55	90	138
Tensuwon Program	7	35	62	98	140
Communication Program	7	25	60	92	135
Average	6.33	29.33	59	93.33	138.67

그림 13 전체 테스트케이스 수와 RE-Rule에 의해 선택된 테스트케이스 수

그림 14는 각 실험에 대하여 RE-Rule을 적용함으로써 테스트케이스가 감소된 비율을 나타낸다. 각 실험별로 3개의 프로그램의 평균적으로 감소되는 비율은 실험 1에서는 36.7%, 실험 2에서는 44%, 실험 3에서는 41%, 실험 4에서는 37.8%, 실험 5에서는 30.7% 감소된 것을 알 수 있다. 각 프로그램별로 테스트케이스 수가 감소되는 비율은 테스트케이스가 많아질수록 비슷해져 간다. 즉, 실험 5에서 감소 비율이 Stack Program은 31%, 연수원 관리 프로그램은 29%, Communication Program은 28.5%로 비슷한 결과가 나왔다. 따라서 RE-Rule을 적용시키는 것이 특정 프로그램에만 국한되지 않음을 알 수 있다.

전체적으로 3개의 프로그램에 대한 5번의 실험을 통하여 RE-Rule에 의해 테스트케이스 수가 평균 37.5% 감소되었다. 이것은 Test Case Selection & Testing

1) 이 테스트케이스는 뮤테이션 테스트 기법[22]에 의해 선정된 메소드 순서 형태의 테스트케이스 임.

Agent가 RE-Rule을 통하여 총 생성된 테스트케이스 가운데 평균적으로 37.5%가 줄어든 테스트케이스 만을 선택하므로, 그 결과 테스트 수행 시간을 결과적으로 단축시킬 수 있게 되었다.

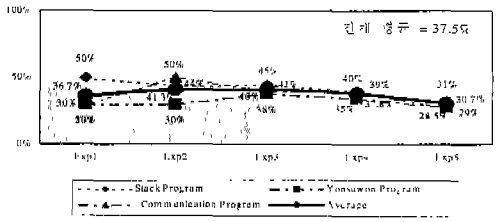


그림 14 RE-Rule에 의해 감소된 테스트케이스 수 비율

5.3 CTS-Rule에 의한 실험

그림 13에서 Exp5의 경우, 3가지 테스트 대상 프로그램, Stack Program, 연수원 관리 프로그램, Communication Program에 대하여 RE-Rule에 의해 감소된 테스트케이스 수는 200개 가운데 각각 138개, 143개, 153개였다. 본 실험에서는 CTS-Rule에 의해 감소되는 테스트케이스 데이타의 수의 비율을 실험하였다. Test Case Selection & Testing Agent는 CTS-Rule을 통하여 RE-Rule에 의해 선정된 각 테스트케이스별로 이 테스트케이스를 구성하는 메소드들에 대한 일관성 있는 테스트 데이타를 선정한다. 따라서 RE-Rule에 의해 감소된 테스트케이스에 대한 총 테스트 데이타 수 가운데 CTS-Rule에 의해 선정되는 일관성 있는 테스트 데이타의 선정 비율을 실험하였다. 그 결과는 Stack Program은 12.01%, 연수원 관리 프로그램은 23.11%, Communication Program은 20.11%이고 평균적으로 18.41%가 선정됨을 알 수 있었다.

그림 15와 같이, 이 실험으로부터 CTS-Rule에 의해 감소된 비율이 Stack Program은 87.99%, 연수원관리 프로그램은 76.89%, Communication Program은 79.89%이고, 테스트 시간이 CTS-Rule에 의해 평균적으로 81.59% 줄어들 수 있음을 알 수 있었다.

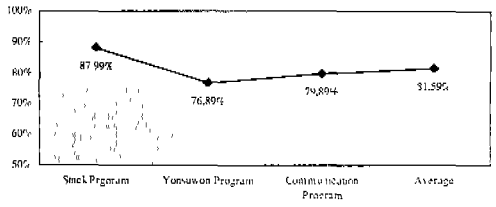


그림 15 CTS-Rule에 의해 감소된 데이타 수의 비율

5.4 TAS에 의한 전체 실험

TAS의 총 테스트 수행 시간을 RE-Rule과 CTS-Rule이 적용되었을 때와 적용되지 않았을 때를 비교하였다. 이 실험은 테스트케이스 수: 10, 50, 100, 150, 200에 따라 5번 수행하였다. 표 2는 각 실험에 대해 3가지 프로그램을 대상으로 측정한 테스트 시간을 나타낸다. 예를 들어 Stack Program의 경우, RE-Rule과 CTS-Rule을 적용하지 않았을 때와 규칙을 적용했을 때의 테스트 수행 시간은 각각 실험 1에서는 10초와 5초, 실험 2에서는 각각 45초와 28초, 실험 3에서는 90초와 50초, 실험 4에서는 125초와 75초, 실험 5에서는 190초와 120초로 측정되었다.

표 2 테스트 시간 측정 결과

	Exp1	Exp2	Exp3	Exp4	Exp5
Stack Program	5(10)	28(45)	50(90)	75(125)	120(190)
Yonsuwon Program	6(10)	35(50)	60(90)	80(130)	120(200)
Communication Program	7(10)	26(51)	60(102)	86(148)	130(205)
Average	6(10)	29.7(48.7)	56.7(94)	80.3(134.4)	123.3(198.3)

각 실험에 대한 테스트 시간이 감소된 비율을 그림 16과 같이 꺾은선 그래프로 나타내었다. 각 실험에 대해 3개 프로그램의 감소된 비율의 평균은 실험 1이 40%, 실험 2가 39%, 실험 3이 41%, 실험 4가 40%, 실험 5가 38%이다. 따라서, 이 실험으로부터 TAS를 사용함으로써 총 테스트 시간은 평균적으로 39.6% 감소함을 알 수 있었다.

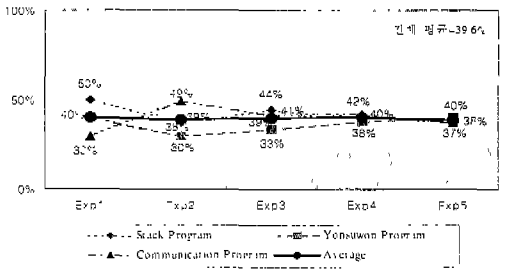


그림 16 TAS에 의한 테스트 시간 감소 비율

5.5 RE-Rule과 오류 검출 효과

3개의 프로그램에 각각 20개의 오류를 삽입하여 그림 2의 테스트케이스 자동 생성기에 의해 생성된 200개의

테스트케이스와 3개의 각 프로그램에서 5.2절에서 RE-Rule에 의해 선택된 테스트케이스에서 오류를 발견하는 수를 비교하였다. 규칙이 적용되지 않은 200개의 테스트케이스로 실험을 한 경우가 Case 1이고 RE-Rule에 의해 선택된 테스트케이스로 실험을 한 경우가 Case 2이다. 이 실험에서 각 Case와 프로그램별로 사용된 테스트케이스 수는 표 3과 같다.

표 3 RE-Rule과 오류 검출 효과 실험 결과

실험 대상 프로그램		실험 대상 테스트케이스 수	오류 삽입 수	오류 검출 수	오류 검출 비율(%)
Stack Program	Case 1	200	20	18	9%
	Case 2	138	20	18	13%
Yonsuwon Program	Case 1	200	20	19	9.5%
	Case 2	143	20	19	13.3%
Communication Program	Case 1	200	20	18	9%
	Case 2	135	20	18	13.3%

표 3은 20개의 오류를 3개의 프로그램에 삽입하여 Case 1과 Case 2에서 오류를 검출하는 수를 나타낸 것이다. 표 3에서 RE-Rule을 적용하지 않고 200개의 테스트케이스로 실험한 Case 1에서 Stack Program이 18개, Yonsuwon Program이 19개, Communication Program이 18개의 오류가 각각 검출되었다. RE-Rule을 적용한 Case 2에서도 Case 1과 같은 수의 오류가 검출되었다. 따라서 각 Case별 실험 대상 테스트케이스 개수에 대한 오류 검출 개수의 비율인 오류 검출 비율은 Stack Program에서는 Case 1 경우는 9%이지만, Case 2 경우에는 13%로 4% 검출 비율이 높은 효과가 나타났다. Yonsuwon Program, Communication Program에서도 Case 2의 검출 비율이 모두 약 4% 더 높은 효과가 나타났다. 그림 17은 표 3의 오류 검출 비율을 막대 그래프로 나타내었다.

200개의 테스트케이스에서의 오류 검출 수와 RE-Rule에 의해 선택된 테스트케이스에서의 오류 검출 수는 같았다. 하지만, 세 개의 프로그램에 대한 오류 검출 비율은 200개의 테스트케이스 수인 Case 1의 경우는 평균적으로 9.2%, Case 2의 경우는 200개의 테스트케이스 수가 RE-Rule에 의해 각각 138개, 143개, 135개로 감소되어서 오류 검출 비율은 평균적으로 13.2%로 증가됨을 알 수 있다. 결과적으로 RE-Rule에 의해 선택된 테스트케이스로 테스트를 실행하면, 5.2절에서처럼 테스트 시간의 감소뿐만 아니라 오류 검출 효과도 향상시킬 수 있음을 알 수 있었다.

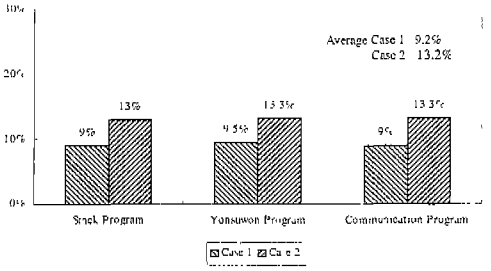


그림 17 오류 검출 비율

6. 결 론

TAS 시스템은 단위 테스트로부터 시스템 테스트까지 점진적으로 테스트를 수행 할 수 있는 테스트 통합 환경을 지원하며 테스터의 간섭을 최소화하고 테스트 프로세스를 따라 자율적으로 테스트를 진행하는 도구이다.

TAS는 두 가지 측면에서 장점을 가지고 있다. 첫째는 자율적으로 테스트를 진행시켜 테스터의 간섭을 최소화한다는 것이고, 둘째는 중복이 없고, 일관성 있는 효율적인 테스트케이스를 지능적으로 선택하여 테스트 시간을 감소시키면서 오류 검출 능력은 향상된다는 것이다.

첫 번째 장점을 분석하기 위해 본 논문에서는 TAS를 사례를 중심으로 실행 과정을 기술하여 TAS를 구성하는 세 개의 에이전트들의 자율적인 행동으로 테스트 진행이 이루어지는 것을 보였다. 즉, Windows 환경에서 JDK1.2.1로 구현된 TAS가 테스트 대상 입력으로부터 테스트 결과 분석까지의 작업이 자율적으로 진행되는 과정을 Prototype을 통해 보여 주었다.

테스트 시간 단축과 오류 검출 효과를 향상시키는 두 번째 장점을 분석하기 위해 본 논문에서는 RE-Rule에 의한 실험, CTS-Rule에 의한 실험, TAS에 의한 전체 실험, RE-Rule과 오류 검출 효과 실험 등 4가지 유형의 실험을 수행한 실험 결과를 기술하였다. RE-Rule에 의한 실험으로부터 Test Case Selection & Testing Agent가 RE Rule을 통하여 테스트 수행 시간을 평균 37.5% 단축시킬 수 있었다. 많은 양의 메소드 순서 테스트케이스에는 테스트케이스 사이에 중복이 되어 불필요한 테스트 시간을 소비하게 되는 경우도 있다. 하지만, 이 RE-Rule을 사용하면 중복을 제거한 테스트케이스로 테스트 시간도 단축시킬 수 있다. CTS-Rule에 의한 실험에서 Test Case Selection & Testing Agent가 CTS-Rule에 의해 선택되는 테스트케이스 수의 비율이 평균 18.41%이므로 테스트 시간은 81.59% 줄어 들 수

있음을 알 수 있었다. TAS에 의한 전체 실험으로부터 TAS의 총 수행 시간을 RE-Rule과 CTS-Rule이 적용되었을 때와 적용되지 않았을 때를 측정한 결과 테스트 시간이 평균적으로 39.6% 감소함을 알 수 있었다. 마지막으로 RE-Rule과 오류 검출 효과 실험으로부터 TAS에서 RE-Rule에 의해 제거되는 테스트케이스들이 오류 발견에 보탬이 되지 않는 쓸모 없는 테스트케이스라는 것을 보였다. 결국, RE-Rule에 의해 선택된 테스트케이스로 테스트를 실행하면 테스트 시간의 감소뿐만 아니라 오류 검출 효과도 향상시킬 수 있음을 알 수 있었다. 현재 TAS는 객체지향 프로그램 테스트만을 행하고 있는데, 향후 컴포넌트 기반 프로그램 테스트로 확장이 필요하다. TAS는 소프트웨어공학의 CASE 도구에서 테스트 측면에 에이전트 개념을 적용한 연구였다. 향후 테스트 분야 외에 소프트웨어공학의 분석, 디자인 등 모든 분야의 CASE 도구들에 에이전트의 개념을 접목시키는 연구가 필요하다.

참 고 문 헌

[1] Huo yan Chen, T. H. Tse, F. T. Chan, and T.Y. Chen, In Black and White: An Integrated Approach to Class-Level Testing of Object-Oriented Programs, ACM Transactions on Software Engineering and Methodology, Vol.7, No.3, pp250-295, July 1998

[2] Pankaj Jalote, Mallaku G.Caballero, Automated Test case Generation for Data Abstraction, Proceedings of COMPSAC, pp205-210, 1988.

[3] Hyacinth S.Nwana, Software Agent: An Overview, Knowledge Engineering Review, vol11, No3, pp1-40, Sept. 1996.

[4] Franklin S. and Graesser A. Is it an agent, or just a program?: A taxonomy for autonomous agents, Proc. of Third International Workshop on Agent Theories, Architect Theories, Architectures, and Languages. 1996

[5] Nicholas R. Jennings, Intelligent Agents: Theory and Practice, Michael Wooldridge. Knowledge Engineering Review January 1995.

[6] Stephen Richard Allen. Concern Processing in Autonomous Agents, Submitted PhD Thesis, University of Birmingham School of Computer Science, 2000.

[7] 최정은, 최병주, "에이전트 기반의 객체지향 소프트웨어 테스트 방안", 정보과학회 논문지: 소프트웨어 및 응용, 제27권 제11호 2000년 11월 pp.1106-1114.

[8] Jeongeun Choi, Byoungju Choi, "Test Agent System Design," 1999 8th IEEE International Fuzzy Systems Conference Proceedings, pp.326-331, Volume I, August, 1999.

[9] Testing Tool Information (http://www.evolutif.co.uk/cast/main.html)

[10] http://www.soft.com/AppNotes/Scribble/index.html

[11] Proteum-A Tool for the Assessment of TestAdequacy for C Programs User's Guide, by Marcio Eduardo Delamaro and Jose' Carlos Maldonado, SERC-TR-168-P, April, 1996.

[12] http://www.clark.net/pub/dickey/atac/atac.html

[13] http://softtest.com/pages/prod_st.htm

[14] http://www.mccabe.com

[15] http://www.obsoft.com/Product/ObjCov.html

[16] http://www.softwareautomation.com/www/overview.htm

[17] http://www.rational.com/products/visual_test/prodinfo/whitepapers

[18] http://www.vtsoft.com/products/hightest.html

[19] http://www.btree.com/pc/validorempc.htm

[20] Thomas Dean, James Allen, Yiannis Aloimonos, Artificial intelligence Theory and Practice, The Benjamin/Cummings publishing company, pp.71-119, 1995.

[21] Mina Rho, Byoungju Choi,Test Process in the Object-oriented Software Development Life Cycle bases on the Test Standards, Proceedings of Asia-Pacific Workshop on Software Process Improvement, pp17-32, 1997

[22] Hoijin Yoon, Byoungju Choi, Inter-class Test Technique between Black-box-class and White-box-class for Component Customization Failures, Sixth Asia Pacific Software Engineering Conference Proceedings, pp.162-165, 1999.

최 정 은

1998년 2월 서울산업대학교 전자계산학과 학사 2001년 2월 이화여자대학교 대학원 컴퓨터학과 석사. 현재 삼성전자 CTO전략실 소프트웨어센터 연구원. 관심분야는 소프트웨어공학, 객체지향 소프트웨어 테스팅, 에이전트, 객체지향 소프트웨어 개발 프로세스, 요구사항 프로세스

최 병 주

정보과학회논문지 : 소프트웨어 및 응용 제 28 권 제 3 호 참조