효율적인 소프트웨어 제품라인 회귀시험을 위한 자동화된 코드 기반 시험 방법†
(Efficient Code-based Software Product Line Regression Testing)

정 필 수‡ 강 성 원§
(Pilsu Jung) (Sungwon Kang)

요 약 소프트웨어 제품라인 개발은 제품군의 개발을 위하여 공통적인 부분과 가변적인 부분을 분리 개발함으로써 중복개발을 피하여 효율적으로 제품군을 개발하는 개발 패러다임이다. 소프트웨어 제품라인 개발에서 제품군을 생성하기 위해 사용되는 소스코드를 제품라인 코드 베이스라고 부르고, 제품라인 코드 베이스가 변경되어 제품군의 제품들이 영향을 받을 때 영향 받은 제품들을 시험하는 활동을 제품라인 회귀시험이라고 한다. 이 때 제품군의 각 제품을 개별적으로 시험하는 대신, 변경과 무관한 시험을 파악하여 피할 수 있다면 효율적인 제품라인 회귀시험이 가능해 질 것이다. 본 논문은 이런 방법으로 소프트웨어 제품라인 회귀시험을 효율적으로 수행하는 자동화된 방법인 SRTS를 소개한다. 이 방법은, 먼저 제품라인 코드 베이스와 시험 항목을 공통성과 가변성을 기반으로 나누고 변경에 영향을 받는 시험 항목을 식별하여 선택한 후, 선택된 시험 항목만을 재실행함으로써 불필요한 시험을 줄인다.
키워드: 소프트웨어 제품라인, 제품군, 회귀시험

Abstract Software product line development is a development paradigm that efficiently develops a product family by avoiding redundant development based on separation of the common part and the variable part of the product family. In software product line development, the source code that is used to produce a product family is called a product line code base, and when the product line code base is changed and the products of the product family are affected by the change, the activity of testing the affected products is called a product line regression testing. For product line regression testing, instead of conducting regression testing individually on each product of the product family, a more efficient regression testing would be possible if unnecessary testing that are irrelevant to the change can be avoided. This paper introduces SRTS, which is an automated method to efficiently perform software product line regression testing. SRTS divides the product line code base and test cases based on commonality and variability. Then SRTS identifies and selects the test cases affected by the change. Finally, it reduces unnecessary testing by rerunning only the selected test cases.
Key words: Software Product Line, Product Family, Regression Testing

1. 서론

소프트웨어 회귀시험은 소프트웨어가 변경되었을 때 변경된 부분을 재시험하여 이전에 시험된 코드에 결함이 없도록 검증하는 활동이다[1]. 이 활동은 소프트웨어가 변경될 때마다 반복적으로 수행되어야 하므로 전체 시험 비용에 큰 비중을 차지한다[2].

소프트웨어 제품라인 개발에서 제품군을 생성하기 위해 사용되는 소스코드를 제품라인 코드 베이스라고 부른다. 소프트웨어 제품라인을 위한 회귀시험은 제품라인 코드 베이스가 변경되었을 때 제품군을 재시험하는 시험활동을 말한다. 제품라인 회귀시험을 수행하기 위해, 제품군이 변경되기 전에 통과(pass)한 기존 시험 항목들을 모두 재실행할 수 있다. 그러나 이 방법은 제품군이 변경될 때마다 변경과 무관한 시험 항목들이 불필요하게 실행되기 때문에 매우 비효율적이다. 따라서 제품라인 코드 베이스의 변경에 영향을 받은 시험 항목들을 식별하여 선택하고 이들 만을 재실행함으로써 불필요한 시험 비용을 줄일 필요가 있다. 이런 목적으로 소프트웨어의 변경된 부분을 재시험하기 위해 기존 시험 항목들 중 어떤 시험 항목들을 선택해야 하는가 하는 문제를 회귀시험 항목 선택 문제(Regression test selection)라고 한다[1].

회귀시험 항목 선택 문제를 다룬 방법으로는 지금까지 아키텍처 기반의 제품라인 회귀시험 항목 선택 방법들[3][4]이 제안되었다. 그러나 이 방법들은 아키텍처가 철저히 관리된 제품라인에서만 적용 가능 하거나[4] 또는 시험 전문가의 개입을 요구 한다[3]. 따라서 과거 연구

† 본 연구는 한국연구재단의 지원(NRF-2017M3C4A7066210)으로 수행하였음
‡ 비회원 : 삼성전자 DIT센터
psjung416@gmail.com
§ 종신회원 : 한국과학기술원 전산학부 교수
sungwon.kang@kaist.ac.kr
논문접수 : 2020년 09월 13일
심사완료 : 2020년 11월 02일

```
public class Calculator{
    public double sum(double n1, double n2){
        return n1 + n2;
    }
    public double sub(double n1, double n2){
        return n1 - n2;
    }
    public double mult(double n1, double n2){
        return n1 * n2;
    }
    public double div(double n1, double n2){
        return n1 / n2;
    }
}
public class TestCalculator{
    @Test
    public void t1(){
        assertEquals(
          10.0, mult(div(sum(3.0,7.0),
                    sub(4.3, 2.3)), 2.0));
    }
}
```

P1

```
public class Calculator{
    public double sum(double n1, double n2){
        return n1 + n2;
    }
    public double sub(double n1, double n2){
        return n1 - n2;
    }
    public double div(double n1, double n2){
        return n1 / n2;
    }
    public double mod(double n1, double n2){
        return n1 % n2;
    }
}
public class TestCalculator{
    @Test
    public void t2(){
        assertEquals(
          1.0, mod(div(sum(3.0,7.0),
                    sub(4.3, 2.3)), 2.0));
    }
}
```

P2

```
public class Calculator{
    public double sum(double n1, double n2){
        return n1 + n2;
    }
    public double sub(double n1, double n2){
        return n1 - n2;
    }
    public double mult(double n1, double n2){
        return n1 * n2;
    }
    public double mod(double n1, double n2){
        return n1 % n2;
    }
}
public class TestCalculator{
    @Test
    public void t3(){
        assertEquals(
          0.0, mult(mod(sum(3.0,7.0),
                    sub(4.3, 2.3)), 2.0));
    }
}
```

P3

그림 2. 계산기 제품군의 소스코드

중에는 자동화된 소스코드 기반의 회귀시험 항목 선택 방법은 없었고, 이러한 방법이 최초로 논문 [5]에서, 제품라인 코드 베이스에 변경이 발생했을 때 회귀시험을 효율적으로 수행하는 자동화된 방법 SRTS(Space based Regression Test Selection for software product lines)으로 발표되었다.

본 논문은 SRTS방법을 소개한다. 이 방법은, 먼저 제품라인 코드 베이스와 시험 항목을 공통성과 가변성을 기반으로 나누어 파티션 테이블을 만들고 제품라인 코드 베이스의 변경에 영향을 받은 시험 항목을 테이블로부터 식별하여 선택한 후, 선택된 시험 항목만을 재실행함으로써 불필요한 시험을 줄인다.

본 논문의 구성은 다음과 같다. 제 2 절에서는 제품라인 회귀시험의 개요를 소개한다. 제 3 절에서는 SRTS방법을 소개한다. 끝으로, 제 4절에서는 SRTS 방법의 의의를 설명하고 향후 연구 방향을 제시한다.

2. 제품라인 회귀시험

제품라인의 제품군은 가변성 모델과 바인딩 정보를 기반으로 정의되고 생성된다. 가변성 모델은 제품군의 공통성과 가변성을 표현하는 모델이다. 제품라인 내의 공통적인 피처를 필수적(Mandatory) 피처로 표현하고 제품 간의 구별되는 차이점을 선택적(Optional) 피처, 택일적(Alternative) 피처 등으로 표현한다. 그림 1(a)는 계산기 제품라인의 가변성 모델을 보여준다. 덧셈 연산과 뺄셈 연산을 포함한 공통 소스코드 c1은 모든 제품의 코드에 공통적으로 포함된다. 그러나 특수 연산(Special op)을 위한 제품의 코드는 c2 와 c3를 포함하거나 c3 와 c4를 포함하거나 c2 와 c4 를 포함하고, 제품 시험을 위해 적용되는 시험 항목은 t1, t2, t3 중 하나가 된다. 바인딩은 제품라인의 공통 산출물로부터 각 제품 별 산출물 인스턴스를 생성하기 위해 반드시 필요한 정보로서, 이로부터 제품라인 코드 베이스 및 시험 항목 중 어떤 부분이 각 제품에 포함되는지 알수있다. 그림 1(b)는 계산기 제품군을 생성하기 위한 바인딩 정보를 보여준다. 그림 1(b)에서 제품 P1을 생성하기 위한 바인딩 정보는BP1인데, BP1은 특수 연산을 위한 구현으로 c2와 c3를 바인드 해주고 시험 항목으로는 t1을 바인드 해준다.

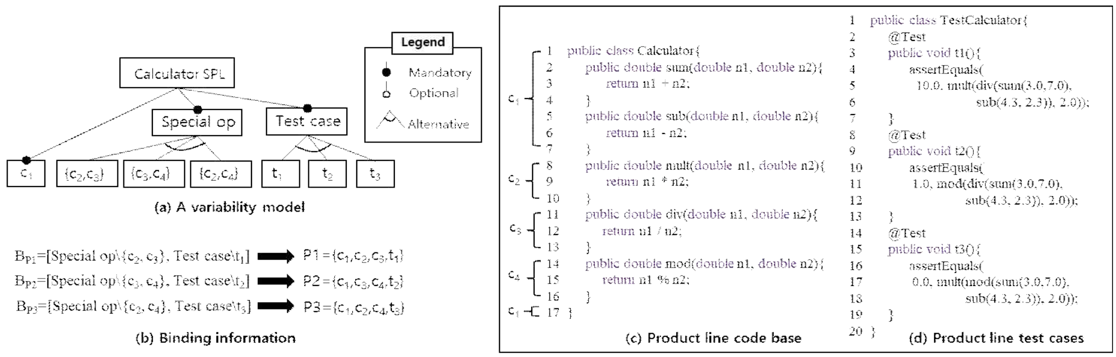

그림 1. 가변성 모델, 바인딩 정보, 제품라인 코드 베이스, 제품라인 시험 항목

그림 1(c)는 제품라인 코드 베이스, 그림 1(d)는 제품라인 시험 항목을 보여준다. 이들은 각각 제품군을 생성하기 위해 필요한 모든 소스코드를 포함하고, 제품군을 시험하는 모든 시험 항목들을 포함한다. 제품군의 소스코드와 시험 항목은 제품라인 코드 베이스와 제품라인 시험 항목에 바인딩 정보를 적용하여 생성된다. 예를 들어, 그림 1(b)의 P1을 위한 바인딩 정보BP1를 제품라인 코드 베이스와 제품라인 시험 항목에 적용하면, 제품라인 코드 베이스의 c1, c2, c3가 조합되어 P1이 생성되고 시험 항목 t1이 P1의 시험을 위해 선택된다. 그림 2는 제품라인으로부터 생성된 제품군의 소스코드와 시험 항목을 보여준다. 제품군이 생성되면 각 제품은 해당 제품을 위해 선택된 모든 시험 항목을 실행함으로써 시험된다.

제품라인에 새로운 피처가 추가되거나 기존 피처가 수정되면 제품라인 코드 베이스가 변경될 수 있다. 변경된 제품라인 코드 베이스를 재시험하는 가장 간단한 방법은 기존 시험 항목들을 모두 재실행하는 전수 시험(Retest-all)을 하는 것이다[4]. 소프트웨어 제품라인의 회귀 시험을 위한 전수 시험 방법은 새로운 결함을 탐지하는데 효과적이지만 변경에 무관한 많은 시험 항목들을 실행하게 되므로 매우 비효율적이다.

전수 시험의 대안으로, 단일 제품을 위한 기존의 회귀 시험 방법을 제품군의 각 제품에 적용할 수 있는데, 이 회귀 시험을 스토브파입 시험(Stove-pipe Test)이라고 부르기로 한다. 이 방법은 변경에 영향을 받지 않는 시험 항목을 각 제품 별로 식별하여 배제함으로써 전수 시험에 비해 제품군의 회귀시험 비용을 많이 줄일 수 있다. 그러나 코드 기반 회귀시험 방법들은 소스코드와 시험 항목을 분석하기 위해 많은 비용을 요구하고, 단일 제품을 위한 회귀시험 절차를 제품 수만큼 반복 적용하게 되기 때문에 제품 수가 많은 제품군에서는 실용적이지 않다.

3. 효율적인 소프트웨어 제품라인 회귀시험 방법

하나의 제품 라인 시험 항목은 동일한 소스코드를 갖는 제품들을 시험하기 위해 재사용될 수 있다. 하지만 서로 다른 소스코드를 시험하기 위해 다수의 제품에 재사용되는 경우, 제품마다 다른 시험 산출물(예를 들어, 시험 커버리지, 시험 결과 등)을 생성할 수 있기 때문에 재사용의 효과가 크게 줄어들 수 있다. 또한, 시험 항목이 멀티쓰레딩, 비동기 함수, 랜덤 함수 등과 같은 비결정적인 기능을 시험하는 경우, 그 시험 항목은 동일한 제품에서 실행될 때마다 다른 결과를 낼 수 있다. 따라서 본 연구는 SRTS를 소개하기 이전에 이 두 가지 측면들에 대해 다음의 가정을 한다.

제품라인 시험 항목의 결정적 실행 가정. 제품라인 시험 항목은 1) 같은 제품에서 여러 번 실행될 때 항상 같은 코드 부분을 시험하고 같은 시험 결과를 산출한다. 또한 2) 그 시험 항목을 사용하는 제품들에서 각각 실행될 때 항상 같은 코드 부분을 시험하고 같은 시험 결과를 산출한다.

위 가정의 1)은 개별 제품에 대한 시험 항목의 결정적 실행에 대한 가정이고, 2)는 제품군에 대한 시험 항목의 결정적 실행에 대한 가정이다. 가정의 2)와 관련하여, 이론적으로는 하나의 시험 항목이 서로 다른 두 제품에서 다른 코드 부분을 시험하더라도 재사용 가능하다. 그러나 이 경우, 시험 항목의 실행 결과가 제품마다 달라질 수 있고, 제품라인 코드 베이스가 변경되었을 때 시험 항목의 관리가 매우 복잡해진다. 따라서 시험 항목은 적용될 때 동일한 코드 부분을 시험하는 제품들에서만 재사용되어야 한다고 가정한다.

제품라인의 회귀시험을 효율적으로 수행하기 위해, SRTS는 공통성과 가변성을 기반으로 제품라인 소스코드와 시험 항목들을 나눈다. 바인딩 정보를 이용하여 각 소스 코드 조각이 포함된 제품 집합을 식별하면 제품라인 코드 베이스와 시험 항목들을 자동으로 여러 개의 코드 영역과 시험 스위트로 나뉜다. 그림 3과 4는 7개의 코드 영역으로 나눠진 제품라인 코드 베이스와 시험 항목을 보여준다. 예를 들어, 그림 1(b)의 바인딩 정보에 따라, c1은 모든 제품에 포함되므로 코드 영역 G에 포함되고, c2는 P1과 P3에 포함되므로 코드 영역 E에 포함되고, c3

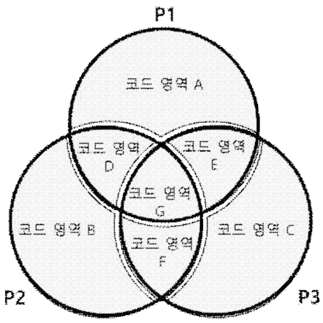

그림 3. 공통성과 가변성을 기반으로 나누어진 제품라인 코드 베이스

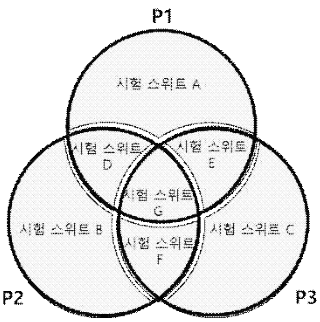

그림 4. 공통성과 가변성을 기반으로 나누어진 제품라인 시험 항목

는 P1과 P2에 포함되므로 코드 영역 D에 포함된다. 마찬가지로, t1은 P1에만 포함되므로 시험 스위트 A에 포함되고, t2는 P2에만 포함되므로 시험 스위트 B에 포함된다. 우리는 모든 제품에 공통으로 포함되는 코드 영역의 소스코드(예를 들어, 그림 3의 코드 영역 G)를 공통 소스코드라고 부르고 그 외의 소스코드(예를 들어, 그림 3의 코드 영역 A, B, C, D, E, F)를 가변 소스코드라고 부른다.

3.1. 공통 소스코드의 변경을 위한 회귀시험

Ekstazi[6]는 단일 제품을 위한 최신의 자동화된 코드 기반 회귀시험 방법이다. 이 방법은 각 시험 항목의 커버리지 정보를 수집한 후, 이를 활용하여 제품 변경 시 변경된 소스코드를 시험하는 시험 항목을 식별하여 재실행한다. 시험 항목의 커버리지 정보를 수집하기 위해, 프로그램에 바이트 코드를 삽입(BCI: Byte-Code Instrumentation)하여 시험 항목이 실행될 때 커버리지 정보가 수집되도록 한다. 그러나 BCI는 많은 비용을 요구하기 때문에 제품군의 회귀시험을 위해 각 제품에 BCI를 반복적으로 수행하는 것은 실용적이지 않다. 따라서 한 제품에서 수집한 커버리지 정보를 다른 제품에 재사용하여 BCI의 반복 수행을 줄일 필요가 있다.

그림 5와 같이, t1, t2, t3가 회귀시험을 위해 선택되고 각 시험 항목이 시험하는 제품이 화살표의 방향과 같을 때, P1 과 P2 에 BCI 를 수행하면 모든 시험 항목의 커버리지 정보와 시험 결과를 얻을 수 있다. P3을 회귀시험하기 위해서 t3가 필요하지만, 제품라인 시험 항목의 결정적 실행을 가정하면 t3는 P1으로부터 얻은 커버리지 정보와 시험 결과가 P3로부터 수집되는 것과 같을 것이므로 재사용될 수 있다. 따라서 P3 에 BCI 를 수행할 필요가 없고 t3는 P3에서 실행될 필요가 없다.

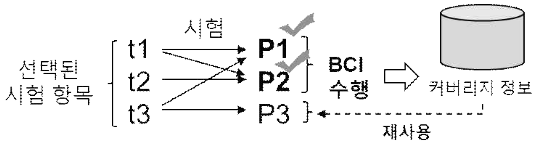

그림 5. 제품군의 회귀시험을 위한 바이트 코드 삽입

3.2. 가변 소스코드의 변경을 위한 회귀시험

하나의 시험 항목은 두 개 이상의 제품에서 동시에 실행될 수 없다. 다시 말해서, 하나의 시험 항목은 단 하나의 제품을 구성하는 코드 영역들에서만 실행될 수 있다. 이 사실을 이용하면 제품군의 소스코드와 시험 항목의 분석 없이 회귀시험 항목수를 효과적으로 줄일 수 있다. 예를 들어, 만약 코드 영역 A가 변경되어 회귀시험을 수행해야 하는 경우, 코드 영역 A는 P2와 P3에 포함되지 않는 코드 영역이므로 P2와 P3에서 실행 가능한 시험 항목들은 코드 영역 A를 절대로 시험하지 않을 것이다. 따라서 시험 스위트 B, C, D, E, F, G는 변경에 무관하므로 제거할 수 있다. 마찬가지로, 코드 영역 F가 변경된 경우, 코드 영역 F는 P1에 포함되지 않으므로 P1에서 실행가능한 시험 항목들은 코드 영역 F를 절대로 시험하지 않을 것이다. 따라서 시험 스위트 A, D, E, G를 제거할 수 있다. 표 1은 각 코드 영역이 변경될 때, 회귀시험을 위해 제거할 수 있는 시험 스위트들을 보여준다.

표 1. 그림 3의 제품군에 대한 회귀시험 선택

변경된 코드 영역	제거될 시험 스위트
코드 영역 A	시험 스위트 B, C, D, E, F, G
코드 영역 B	시험 스위트 A, C, D, E, F, G
코드 영역 C	시험 스위트 A, B, D, E, F, G
코드 영역 D	시험 스위트 C, E, F, G
코드 영역 E	시험 스위트 B, D, F, G
코드 영역 F	시험 스위트 A, D, E, G
코드 영역 G	-

이 접근 방법은 가변 소스코드의 변경을 위한 회귀시험에는 효과적이지만 공통 소스코드가 변경되었을 경우에는 효과적이지 않다. 예를 들어, 코드 영역 G가 변경되었을 경우에는 이 방법을 사용해서 제거할 수 있는 시험 스위트가 없다. 따라서 3.1절에서 설명한 회귀시험 항목 선택 방법을 적용하는 것이 좋다.

3.3. 방법의 평가

이 절의 SRTS의 평가는 논문 [5]에 나오는 내용을 요약한 것이다. SRTS의 검증을 위해, SPL2go[7] 오픈 소스 저장소로부터 Java 언어로 구현된 여섯 개의 제품라인 프로젝트를 선정하였다. 표 2는 대상 제품라인의 이름과 각 제품라인으로부터 생성한 제품군의 코드 규모(SLOC), 제품군의 크기(|PSPL|), 공통 소스코드의 비율(Com.), 시험 항목의 수(|TSPL|), 시험 항목의 총 초 단위의 실행 시간(ET)을 보여준다.

표 2. 대상 제품라인과 제품군

대상 제품라인	SLOC	$\lvert P_{SPL} \rvert$	Com.	$\lvert T_{SPL} \rvert$	ET
MobileMedia	3,196	4	44.2%	1,139	6.9
TankWar	4,845	5	62.9%	782	2.6
Prevayler	5,109	3	72.5%	1,306	9.9
		5	72.5%	2,543	23.2
MobileRSS Reader	16,664	3	93.1%	3,247	11.1
		5	93.1%	5,383	22.2
Lampiro	30,158	4	98.8%	6,236	128.3
		6	98.6%	9,346	188.0
BerkeleyDB	44,994	3	69.3%	10,297	23.0
		5	69.3%	18,407	46.8
		7	69.3%	26,290	55.7

대상 제품라인의 시험 항목은 EvoSuite 도구[8]를 활용하여 생성하였고, 제품군의 변경된 버전은 PIT 도구[9]를 활용하여 제품라인 별로 50개를 생성하였다.

SRTS의 효용성을 검증하기 위해, 단일 제품을 위한 회귀시험 방법을 제품군에 적용하는 전통적인 제품군 회귀시험 방법과 비교하였다. 단일 제품을 위한 회귀시험 방법은 Ekstazi[6]를 채택하고, 이를 제품군의 각 제품에 적용하는 방법을 Ekstazi_SPL이라고 부른다. Ekstazi는 변경에 영향을 받는 시험 항목을 모두 선택하는 방법이고, Ekstazi_SPL은 제품군의 각 제품에 Ekstazi를 반복 적용하는 것이므로 Ekstazi_SPL 또한 변경에 영향을 받는 시험 항목을 모두 선택한다. SRTS와 Ekstazi_SPL의 비교 항목은 선택된 결함 노출 시험 항목(fault-revealing test case)의 수, 선택된 시험 항목의 총 개수, 총 회귀시험 수행 시간(회귀 시험 항목을 선택하고 선택된 시험 항목을 실행하는 시간)이고 50개의 변경된 버전을 대상으로 실험한 결과의 평균 값을 비교하였다.

실험 결과, SRTS와 Ekstazi_SPL 방법은 모든 결함 노출 시험 항목을 선택하였다. 이 결과가 나타난 이유는 변경된 코드를 시험하는 시험 항목을 모두 선택하는 방법은 항상 모든 결함 노출 시험 항목을 선택하게 되고[10], 두 방법은 모두 제품라인 시험 항목의 결정적 실행 가정 하에 위 조건을 만족하는 방법들이기 때문이다.

표 3은 대상 제품군에 대해, 선택된 시험 항목의 총 개수, 회귀시험 총 시간에 관하여 SRTS와 Ekstazi_SPL 방법을 비교한 결과이다. 표 3에서 괄호 안의 숫자는 제품군을 구성하는 제품의 수이다. SRTS는 모든 대상 제품군에 대해 Ekstazi_SPL보다 총 회귀시험 시간을 14.8% ~ 49.1% 줄였다. 특히, Prevayler(3), Prevayler(5), MobileRSSReader(3), Lampiro(4) 제품군에서 적은 수로 나눠진 코드 영역으로 인해 SRTS가 Ekstazi_SPL보다 적은 수의 시험 항목을 줄였음에도 불구하고 더 많은 회귀시험 시간을 줄였다. 이러한 결과는 SRTS가 회귀시험 절차의 불필요한 반복을 줄이고 소스코드와 시험 항목의 불필요한 분석을 효과적으로 줄인다는 것을 의미한다.

4. 결론

본 논문은 소스코드를 기반으로 제품라인 회귀시험을 효율적으로 수행하는 자동화된 방법인 SRTS를 소개하였다. SRTS는 공통 소스코드의 변경과 가변 소스코드의 변경을 별도로 다룬다. 공통 소스코드가 변경되었을 때 한 제품의 시험 산출물을 재사용하여 불필요한 회귀시험 절차의 반복을 줄이고, 가변 소스코드가 변경되었을 때 제품군의 제품라인 코드베이스와 시험 항목을 공통성과 가변성을 기반으로 분할하여 소스코드와 시험 항목의 분석 없이 변경에 무관한 시험 항목들을 제거한다.

실험 결과에 나타나듯이, SRTS를 적용하면 시험 항목을 선택하고 선택된 시험 항목을 실행하는 총 회귀 시험 시간을 Ekstazi_SPL과 비교하여 4.8%~49.1%만큼 줄일 수 있다. 이 효과는 제품군을 구성하는 제품 수가 많을수록, 제품 간의 공통부분이 많을수록 커지는 경향을 보인다. 따라서 SRTS는 다수의 유사한 제품들을 효율적으로 개발하기 위해 개발 산출물을 최대한 많이 재사용한 제품군에서 큰 효과를 보인다.

본 논문에서 소개한 SRTS에 관련된 후속 연구로는 SRTS의 효용성을 대규모의 제품라인을 대상으로 검증하는 연구와 SRTS를 확장하여 회귀시험을 위해 선택된 시험 항목들을 실행할 때 발생하는 불필요한 비용을 줄이는 연구가 필요하다.

표 3. 실험 결과

대상 제품라인	코드 영역 수	선택된 시험 항목의 수			총 회귀시험 수행 시간		
		SRTS	Ekstazi_SPL	Savings	SRTS	Ekstazi_SPL	Savings
MobileMedia(4)	11	10.7%	12.7%	15.6%	28.0%	39.0%	28.1%
TankWar(5)	17	28.1%	34.5%	18.7%	80.5%	158.0%	49.1%
Prevayler(3)	4	29.2%	20.0%	-46.0%	48.6%	59.1%	17.1%
Prevayler(5)	6	31.4%	23.4%	-34.3%	49.5%	58.1%	14.8%
MobileRSSReader(3)	5	23.4%	23.4%	-0.2%	58.4%	87.6%	33.3%
MobileRSSReader(5)	16	22.9%	23.4%	1.8%	48.0%	84.8%	43.4%
Lampiro(4)	10	16.3%	16.2%	-0.6%	26.2%	36.4%	28.0%
Lampiro(6)	16	16.1%	16.3%	1.2%	24.2%	37.7%	35.9%
BerkeleyDB(3)	4	24.4%	25.4%	3.9%	61.1%	71.7%	14.8%
BerkeleyDB(5)	16	24.0%	32.4%	26.0%	55.5%	75.0%	26.0%
BerkeleyDB(7)	31	20.9%	32.0%	34.6%	50.8%	72.1%	29.6%

참 고 문 헌

[1] Yoo, S., Harman, M., 2012. Regression testing minimization, selection and prioritization: a survey. Software Testing, Verification and Reliability. 22 (2), 67 - 120.

[2] Chittimalli, P. K., & Harrold, M. J., 2009. Recomputing coverage information to assist regression testing. IEEE Transactions on Software Engineering, 35(4), 452-469.

[3] Neto, P.A.d.M.S., do Carmo Machado, I., Cavalcanti, Y.C., de Almeida, E.S., Garcia, V.C., de Lemos Meira, S.R., 2010. A regression testing approach for software product lines architectures. In: Proceedings of the Fourth Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), pp. 41 - 50.

[4] Lity, S., Nieke, M., Thüm, T., & Schaefer, I., 2019. Retest test selection for product-line regression testing of variants and versions of variants. Journal of Systems and Software, 147, 46-63.

[5] Jung, P., Kang, S., & Lee, J., 2019. Automated code-based test selection for software product line regression testing. Journal of Systems and Software, 158, 110419.

[6] Gligoric, M., Eloussi, L., Marinov, D., 2015. Practical regression test selection with dynamic file dependencies. In: Proceedings of the International Symposium on Software Testing and Analysis, pp. 211 - 222.

[7] SPL2go. Available online: http://spl2go.cs.ovgu.de/ (accessed on September 2020).

[8] Fraser, G., Arcuri, A., 2011. EvoSuite: automatic test suite generation for object-oriented software. Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, pp. 416-419.

[9] PIT. Available online: http://pitest.org/ (accessed on September 2020).

[10] Rothermel, G., Harrold, M.J., 1996. Analyzing regression test selection techniques. IEEE Transactions on Software Engineering. 22 (8), 529 - 551.

정 필 수

2012년 충남대학교 컴퓨터공학과 졸업 (학사)

2014년 한국과학기술원 전산학과 졸업 (석사)

2020년 한국과학기술원 전산학과 졸업 (박사)

관심분야는 소프트웨어 아키텍처, 소프트웨어 제품라인 공학, 소프트웨어 재사용

강 성 원

1982년 서울대학교 사회과학대학 졸업

1989년 University of Iowa 전산학 석사

1992년 University of Iowa 전산학 박사

1993 ~ 2001년 한국통신(KT) 선임연구원

2001 ~ 2009년 한국정보통신대학교 전산학과 교수

2009 ~ 현재 한국과학기술원 전산학부 교수