RT-Selection: 텍스트 차이점과 변경 영향 분석을 이용한 회귀 테스트 선택 기법

(RT-Selection: A Regression Test Selection Technique using Textual Differencing and Change Impact Analysis)

김 의 섭 † 이 동 아 † 유 준 범 ††
(Eui-Sub Kim) (Dong-Ah Lee) (Junbeom Yoo)

요 약 회귀 테스팅은 소프트웨어의 변경이 기존 기능에 피해를 주지 않았다는 신뢰를 제공하기 위해 수행하는 테스팅 활동이다. 가장 간단하고 기본적인 방법은 기존의 모든 테스트 케이스를 이용해 다시 테스팅을 수행하는 것이지만 이 방법은 많은 시간과 비용을 필요로 한다. 회기 테스팅 중 회귀 테스트 선택(Regression test selection) 방법은 기존 테스트 케이스 중 변경을 테스트할 수 있는 테스트 케이스만을 선택하여 다시 수행하는 방법이다. 다시 수행하는 테스트 케이스의 수를 줄임으로써 비용 절감 효과를 얻을 수 있다. 본 논문은 회귀 테스트 선택의 효과적인 수행을 지원하는 기법으로 RT-Selection을 제안한다. RT-Selection은 두 가지 접근법으로 구성되어 있다. 첫째, 변경을 식별하기 위해 텍스트 차이점을 사용한다. 둘째, 테스트 케이스에 의해 실행되는 소프트웨어의 부분을 식별하기 위해 변경 영향 분석을 사용한다. 본 논문은 RT-Selection의 모든 과정과 이를 지원하는 가이드라인, 추론 규칙을 보여 주고, RT-Selection을 이용해 수행한 케이스 스터디를 보여준다.

키워드: 회귀 테스팅, 회귀 테스트 선택, 텍스트 차이점, 변경 영향 분석

Abstract Regression testing is a testing activity that is performed to provide confidence that changes do not harm. One of simple and basic regression testing techniques is retest-all, however, it requires a lot of time and cost. The regression test selection technique identifies changes and selects a subset of previous test cases to retest the changed software. The technique reduces the number of test cases, so that it is able to reduce the time and cost for the regression testing. This paper proposes the RT-Selection that effectively performs the regression test selection. This technique consists of two approaches. First, it uses text differencing to fine the changes. Second, it uses change impact analysis to fine the software riffle that is a trace of test case in the software. This paper shows the overall process of RT-Selection and guidelines and inference rule and then it shows the case study with RT-Selection to show the feasibility of RT-Selection.

Keywords: regression testing, regression test selection, textual differencing, change impact analysis

· 이 논문은 2013학년도 건국대학교의 연구년교원 지원에 의하여 연구되었음

† 학생회원 : 건국대학교 컴퓨터공학부
atang34@konkuk.ac.kr
ldalove@konkuk.ac.kr
†† 정 회 원 : 건국대학교 컴퓨터공학부 교수
jbyoo@konkuk.ac.kr
(Corresponding author임)

논문접수 : 2013년 12월 23일
심사완료 : 2014년 4월 12일

정보과학회논문지: 소프트웨어 및 응용 제41권 제6호(2014.6)

1. 서 론

회귀 테스팅(Regression testing)은 소프트웨어의 변경이 기존 소프트웨어의 기능이나 변경되지 않은 나머지 부분에 피해를 주지 않았다는 신뢰성을 제공하기 위해 수행하는 테스팅 활동이다[1]. 회귀 테스팅의 가장 간단하고 기본적인 방법은 기존의 테스트 케이스를 이용하여 변경이 이루어진 소프트웨어에 다시 수행하는 것이다(retest-all). 하지만 이 방법은 많은 시간과 비용(테스터의 시간과 노력, 자원 등)을 필요로 한다. 한 예로 [2]에서는 30,000개의 기능 테스트 케이스를 수행하기 위해 1,000시간의 기계운전 시간이 필요하고, 추가적으로 테

스터가 이를 감독하는데 100시간이 필요하다고 했다. 문제는 현재 소프트웨어의 크기와 복잡도가 점점 증가하고 있어, 테스팅 비용 또한 계속 증가 하고 있다는 점이다. 따라서 이런 테스트 비용을 줄이기 위해 전략적인 테스팅 방법이 필요하고, 본 논문은 회귀 테스트 선택(Regression Test Selection) 기법을 이용하여 회귀 테스팅 시 테스트 비용을 줄일 수 있는 기법으로 RT-Selection을 제시한다.

Rothermel와 Harrold[3]는 회귀 테스팅 문제를 다음과 같이 정의 했다. “기존 프로그램을 P, 변경이 생긴 프로그램을 P', 기존 P에 수행했던 테스트 케이스 집합을 T라 할 때, 회귀 테스팅은 P와 P'가 일치함을 T를 이용해서 보여 주는 것이다”. 이와 같은 작업을 수행하기 위해 필요한 작업을 두 가지로 정의할 수 있다. 첫째, 소프트웨어의 변경을 식별하는 작업, 둘째, p와 p'가 일치함을 보여주기 위한 T의 서브 셋(subset)을 찾는 작업이다.

RT-Selection은 위 두 작업을 텍스트 차이점과 변경 영향 분석을 통해 수행한다. 첫째, RT-Selection은 소프트웨어의 변경을 식별하기 위해 텍스트 차이점을 이용한다. 텍스트 차이점은 변경이 이루어지기 전과 후, 두 소프트웨어의 소스 코드를 텍스트 단위의 비교를 통해 변경을 식별한다. 텍스트 단위의 비교는 변경을 보다 정확하고 쉽게 식별 할 수 있는 장점이 있다.

둘째, RT-Selection은 소프트웨어의 변경 전과 후의 일치성을 보장하기 위한 서브 셋을 찾기 위해 변경 영향 분석을 이용한다. Arnold와 Bohner[4]은 변경 영향 분석을 다음과 같이 정의 했다. “주어진 변경으로부터 잠재적으로 영향을 받는 소프트웨어의 부분을 식별하고, 식별된 부분으로부터 영향을 받는 부분들을 식별하는 것”. RT-Selection은 잠재적인 영향과 그 영향으로부터 영향을 받은 부분을 식별하기 위해 변경이 이루어진 지점에 변경을 식별할 수 있는 풋 프린트(footprint)의 삽입을 제안한다. 풋 프린트는 변경이 이루어지기 전 코드의 분석을 통해 삽입하게 되는데, 풋 프린트가 삽입된 코드와 기존 수행 했던 테스트 케이스를 이용해 다시 테스팅을 수행하면 자연스럽게 풋 프린트 리스트를 얻을 수 있다. 생성된 풋 프린트 리스트는 테스터가 테스트 케이스에 의해 실행되는 소프트웨어의 부분과 실행된 요소들 사이의 영향 관계를 쉽고 정확하게 식별 가능하게 하는 기능을 한다. 첫 번째 활동과 두 번째 활동의 결과물을 이용해 추론 작업을 거치고 나면 테스터는 소프트웨어의 변경을 테스트 할 수 있는 최적화된 회귀 테스팅의 테스트 케이스 셋을 얻게 된다.

우리는 RT-Selection을 이용하여 본교 대학원의 Software Test Management 수업과 본교 대학의 Introduction to Software Engineering 수업을 통해 케이스 스터디를 수행하였다. 기존의 테스트 케이스를 총 4가지의 타입(Obsolete, NewSpec, Re-testable, Re-useable)의 테스트 케이스로 분류할 수 있었고, 이를 이용해 비용절감적인 회귀 테스팅을 수행할 수 있을 것이라 기대할 수 있다.

본 논문은 다음과 같이 구성된다. 2장에서는 과거 제시되었던 회귀 테스팅 기법에 대해 설명한다. 3장에서는 RT-Selection의 자세한 과정을 설명한다. 4장에서는 RT-Selection을 이용해 수행한 케이스 스터디를 보여준다. 5장에서는 결론 및 향후 계획에 대해 소개한다.

2. 관련 연구

소프트웨어 개발 시 소프트웨어는 요구사항 변경이나 오류 수정, 업데이트 등과 같은 이유로 수 많은 변경이 일어난다[5]. 소프트웨어의 변경은 기존에 존재하지 않았던 새로운 버그나 에러를 생성 할 가능성이 있다. 변경에 의해 새로운 버그나 에러가 생성되지 않았다는 것을 확인 하는 활동이 회귀 테스팅이다. 하지만 회귀 테스팅은 비용이 많이 들어가는 활동이다. 따라서 비용을 낮추기 위한 기법이 필요했고, 과거 이런 비용을 줄이기 위한 여러 기법들이 제시 되어 왔다.

그 중 널리 알려진 기법 중 하나가 회귀 테스트 선택(RTS-Regression test selection)이다[6-11]. 회귀 테스트 선택은 변경이 일어난 소프트웨어를 다시 테스트 하기 위해 기존 테스트 케이스들 중 변경을 테스트할 수 있는 테스트 케이스를 선택하는 방법이다. 기존 테스트 케이스 중에 다시 테스트 해야 되는 케이스를 선택하기 때문에 회귀 테스팅 시 수행 비용을 낮출 수 있는 장점을 가지고 있다.

RTS를 구현한 예 중 하나로 F. I. Vokolos와 P. G. Frankl 가 제시한 Pythia가 있다[11]. Pythia는 변경이 생기기 전과 후의 코드의 텍스트 차이점을 이용해 회귀 테스팅을 위한 테스트 케이스를 선택한다. 순전히 텍스트의 차이점만을 기반으로 한 Pythia는 Rothermel & Harrold[6]의 control flow graph 기반 분석을 텍스트 차이점으로 대체한 것인데, 기존 존재하는 도구를 이용하여 구현이 쉽다는 장점을 가지고 있지만, 정확도가 떨어질 수 있다는 문제점을 가지고 있다. 반면 slicing 기반 RTS 기술들은 대체로 backward slicing을 이용해서 정확한 impact analysis를 하는 것을 목표로 하는데[12,13], 정확도에도 불구하고 slicing이 가지는 한계점(예를 들어 코드 크기의 제약) 때문에 널리 이용되지 못했다.

RT-Selection은 선택된 테스트 케이스의 분류(Obsolete, NewSpec, Re-testable, Re-useable)를 제공함으로써 단순 텍스트 차이점 비교의 단점인 정확도의 문제를 극

복하고, 자동화 도구의 구현을 가능하게 하는 방법들을 제시함으로써 slicing의 한계점(코드 크기에 대한 제약)을 극복할 수 있는 방법을 제시한다.

RTS 이외에 테스팅 비용을 줄일 수 있는 예로는 Minimization, Prioritization을 들어 볼 수 있다. Minimization은 중복되거나 테스트 요구사항을 만족하지 못하는 불필요한 테스트 케이스를 선택해 제거함으로써 테스트 케이스의 수를 줄여 회귀 테스팅 시 비용을 낮추는 방법이다[14-16]. Prioritization은 테스트 케이스에 우선순위를 부여하여 테스팅이 달성해야 하는 수준을 최소한의 테스트 케이스를 통하여 달성하고자 하는 방법이다. 현실적으로 모든 상황에 대해 테스트 케이스를 만들고 테스팅 하는 것은 불가능하기 때문에 적절한 테스트 케이스의 수로 충분한 결과를 얻기 위한 높은 수준의 커버리지를 가지고 있는 테스트 케이스를 선별할 필요가 있는데, 이런 선별 작업을 지원하는 것이 prioritization이다[17-20].

3. RT-Selection의 과정

RT-Selection의 과정은 총 7단계로 이루어져 있다. 전체적인 흐름을 그림 1을 통해 확인할 수 있고, 각 단계에 대한 자세한 내용을 앞으로 설명한다.

3.1 (1 단계) 정형화된 형태 만들기

정형화된 형태로 만든다는 것은 서로 다른 코드를 구문적으로 동일한 스타일의 코드로 만드는 것이다. RT-Selection은 소프트웨어의 변경을 텍스트 차이를 이용해 식별하기 때문에 불필요한 텍스트 차이를 줄일 필요가 있다. 예를 들면, 텍스트 단위의 비교는 괄호('(')의 위치나 띄어쓰기, 공백, 내려쓰기 등 사소한 코딩 스타일 차이 하나도 모두 비교의 대상이 된다. 하지만 이런 단순 스타일의 차이는 실제 소프트웨어의 기능과 관련된 변경이 아니기 때문에 의미가 없지만, 변경으로 인식되기 때문에 테스터의 추가적인 분석을 요구하게 된다. 따라서 이런 문제점을 해결하기 위해 상이한 두 코드를 하나의 정형화된 형태로 일치시켜 테스터의 분석에 대한 부담을 덜어줄 필요가 있다.

그림 2의 (a)와 (b)는 기능이 일치하지만 다른 스타일로 작성된 코드이다. 두 코드는 기능이 일치할 뿐만 아니라 정형화된 형태로 만들게 되면 정확하게 일치 하다는 것을 (c)를 통해 확인할 수 있다. 하지만 (a)와 (b) 두 코드를 이용해 텍스트 비교를 하게 되면, 수 많은 차이점이 생성된다. 이런 차이는 소프트웨어의 기능적인 변경이 아니기 때문에 분석이 불필요하지만, 변경으로 식별되어 테스터의 추가적인 분석을 요구하게 된다. 따라서 이런 불 필요한 작업을 줄이기 위해 하나의 정형화된 형태로 만들어줄 필요가 있다.

Guideline 1. RT-Selection은 기능적으로 의미 없는 부분에 의한 텍스트 차이점(예를 들면, 공백이나 띄어쓰기 등)을 제거하기 위해 하나의 동일한 스타일로 만들어야 한다. 하나의 정형화된 스타일로 만들 수 있다면 어떤 스타일로 정형화 하는지, 또는 어떤 도구를 사용하여 정형화 하는지에 대해서는 제약을 두지 않는다. 따라서 기존의 도구 및 방법을 재 사용할 수 있다.

3.2 (2 단계) 코드 분석과 풋 프린트 삽입

풋 프린트란 코드 상에서 "어떤 요소가 어떤 요소에 영향을 주는지 또는 받는지"에 대한 기록이다. 소프트웨어는 실행 중에 다양한 요소들이 서로 영향을 주고받게 되는데, 어떤 요소들이 상호 영향을 주고 받는지를 기록하여 테스터가 이를 알 수 있도록 하는 것이 풋 프린트의 역할이다. 풋 프린트를 삽입하기 위해서는 소프트웨어의 코드를 정적으로 분석할 필요가 있다. RT-Selection에서 관심을 갖고 분석해야 되는 요소를 다음과 같이 정의한다.

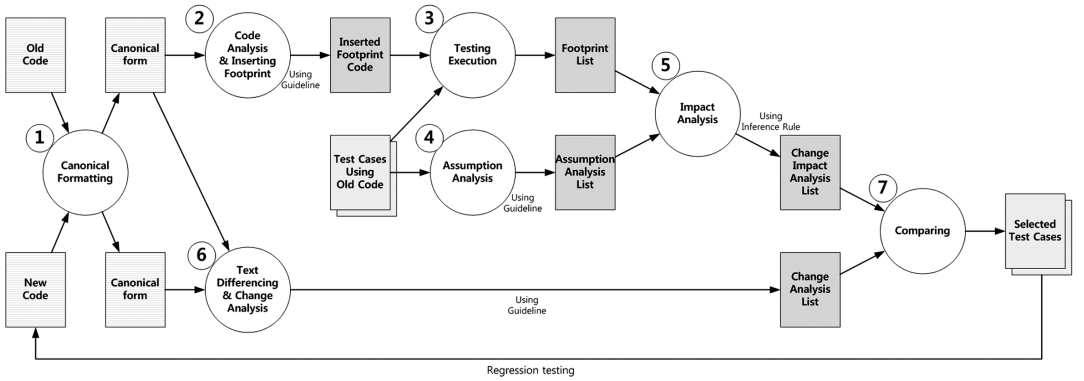


그림 1 RT-Selection의 전체 과정

Fig. 1 An overview of the RT-Selection technique

```c
#include <stdio.h>

int Factorial(int n){
	int i,result=1;
	for (i=1; i<=n; i++)
	{ result=result*i; }
	return result;
}
int Combination(int n,int r){
	int a=Factorial(n);
	int b=Factorial(n-r)
		*Factorial(r);
	int result=a/b;

	return result;
}
```

(a) A source file in style A

```c
#include <stdio.h>

int Factorial (int n)
{
	int i, result = 1;
	for (i = 1 ; i <= n ; i++)
	{
		result = result * i;
	}
	return result;
}
int Combination (int n, int r)
{
	int a = Factorial (n);
	int b = Factorial (n - r)
		* Factorial (r);
	int result = a / b;

	return result;
}
```

(b) A source file in style B

```c
#include <stdio.h>

int Factorial(int n) {
	int i, result = 1;
	for (i = 1; i <= n; i++) {
		result = result * i;
	}
	return result;
}
int Combination(int n, int r) {
	int a = Factorial(n);
	int b = Factorial(n - r) * Factorial(r);
	int result = a / b;

	return result;
}
```

(c) A source file in style B

그림 2 스타일이 다른 두 코드에 정형화를 수행한 예
Fig. 2 An example of the old, new and canonically formatted cods)

- 배치문, 함수 반환문, 함수 호출문, 조건문의 조건부, 반복문의 제어부

상기 요소들을 식별하고 해당 구문의 의미에 맞는 풋 프린트를 삽입하는 활동이 코드 분석과 풋 프린트 삽입이다. 예를 들면 *"a = b + 1"* 와 같은 코드가 있을 경우, 먼저 해당 구문이 배치문임을 식별한 뒤, 해당 구문에 맞는 풋 프린트를 선정(*"a ← b"*), 이를 해당 코드에 삽입한다. 여기서 기호 *"←"*는 영향을 주었다는 것을 의미한다. 의미를 해석하자면, "요소 a는 요소 b에 의해 영향을 받았다." 또는 "요소 b가 요소 a에 영향을 주었다." 라는 의미로 해석할 수 있다. 삽입된 풋 프린트들은 3단계 '테스트 수행'을 통해 자연스럽게 풋 프린트의 리스트 형태로 결과를 얻게 된다.

그림 3은 그림 2(c)의 코드에 풋 프린트가 삽입된 모습을 보여준다. RT-Selection은 코드 분석과 적절한 풋 프린트 선택, 삽입을 보다 체계적으로 수행 할 수 있도록 다음과 같은 가이드라인을 제시한다. 이를 이용해 테스터는 보다 체계적인 코드 분석과 풋 프린트 삽입을 진행할 수 있을 것이고, 또한 이 가이드라인은 곧 자동화 도구 구현 시 도구의 핵심요구사항이 될 것이다.

Guideline 2. 만약 코드가 *"Left element = Right element"*

```c
#include <stdio.h>

int Factorial(int n) {

	int i, result = 1;

	printf("Factorial(); 1_for_condition <- i\n");
	printf("Factorial(); 1_for_condition <- n\n");
	for (i = 1; i <= n; i++) {

		printf("Factorial(); result <- 1_for_condition\n");
		printf("Factorial(); result <- i\n");
		result = result * i;
	}

	printf("Factorial(); Factorial() <- result\n");
	return result;
}
int Combination(int n, int r) {

	printf("Combination(); a <- Factorial()\n");
	printf("Combination(); Factorial() <- n\n");
	int a = Factorial(n);

	printf("Combination(); b <- Factorial()\n");
	printf("Combination(); Factorial() <- n\n");
	printf("Combination(); Factorial() <- r\n");
	int b = Factorial(n - r) * Factorial(r);

	printf("Combination(); result <- a\n");
	printf("Combination(); result <- b\n");
	int result = a / b;

	printf("Combination(); Combination() <- result\n");
	return result;
}
```

그림 3 풋 프린트가 삽입된 코드의 예
Fig. 3 The old code with inserted footprints

와 같은 형태의 배치문일 경우 *"Left element ← Right element"*와 같은 풋 프린트를 삽입한다. 코드가 *"return something"*와 같은 형태의 함수 반환문일 경우 *"function name()← something"*와 같은 풋 프린트를 삽입한다. 코드가 함수 호출문일 경우 *"function name() ← parameter"* 을 삽입한다.

Guideline 3. 만약 코드가 if나 switch 등과 같은 조건문일 경우 *"(ordinal number)_(id)_condition ← used element in condition statement"*와 같은 풋 프린트를 삽입한다. 여기서 *ordinal number*는 해당 조건문이 해당 함수에서 사용된 순서를 나타내고, *id*는 *if* 나 switch와 같은 해당 조건문을 식별할 수 있는 식별자를 나타내고, *condition*은 해당 조건문의 조건부를 나타낸다. 여기서 영향을 주는 요소는 조건문의 조건부에 사용된 요소이다. 추가로, 만약 조건문의 기능부에 배치문이 사용될 경우 해당 배치문의 풋 프린트에 *"Left element ← (ordinal number)_(id)_condition"* 를 추가로 삽입한다.

Guideline 4. 만약 코드가 for나 while 등과 같은 반복문일 경우 *"(ordinal number)_(id)_condition ← used element in condition statement"*와 같은 풋 프린트를 삽입한다. 나머지 기본적인 부분은 Guideline 2와 동일하다.

Guideline 5. Guideline 1~3가 중복적으로 사용될 경우 풋 프린트를 열거를 하여 사용한다. 예를 들면 조

건문 안에 조건문 안에 조건문…일 경우 *"Left element ← (ordinal number)_(id)_condition_(ordinal number)_(id)_ condition…"* 와 같이 열거하여 사용한다.

3.3 (3 단계) 테스트 수행

RT-Selection에서 이루어지는 테스팅은 유닛 테스팅(Unit testing)을 기본으로 삼고 있다. 유닛 테스팅은 소프트웨어의 컴포넌트나 컴포넌트의 집합을 테스트 하는 것으로써 입력과 입력에 대한 결과를 예측함으로써 해당 유닛의 기능을 확인하는 테스팅이다. 기본적으로 유닛 테스팅은 테스팅하고자 하는 해당 유닛을 제외한 소프트웨어의 나머지 부분에 대해서는 무시를 하기 때문에 해당 유닛 안에서의 변경에만 집중하여 테스팅 할 수 있는 장점이 있다. 따라서 RT-Selection은 유닛 안에서의 변경에 집중하여 테스팅을 수행할 수 있는 유닛 테스팅을 기본 테스팅 방법으로 삼고 있다.

테스터는 기존 테스트 케이스를 이용하여 2단계의 결과인 풋 프린트가 삽입된 코드에 기존에 수행했던 유닛 테스팅을 다시 수행하면 된다. 테스팅이 끝나면 테스터는 자연스럽게 그림 4(a)와 같은 풋 프린트 리스트를 얻을 수 있다. 풋 프린트 리스트는 해당 테스트 케이스가 소프트웨어의 어떤 부분을 수행하였는지, 수행 중에 어떤 요소들이 영향을 주고 받았는지 식별 가능하게 한다.

하지만 생성된 풋 프린트 리스트는 많은 중복을 가질 수 있다. 예를 들어 반복문이나 재귀함수와 같은 내용이 유닛 안에 있을 경우, 반복된 횟수만큼의 중복된 풋 프린트들을 생성 하게 될 것이다. 따라서 보다 효율성 있는 테스팅을 진행하기 위해 이런 중복을 제거할 필요가 있다. 그림 4(b)는 중복을 제거한 최적화된 결과를 나타낸다.

Guideline 6. 만약 풋 프린트 리스트에 중복된 풋 프린트가 있을 경우 중복된 풋 프린트를 제거하여 하나의 풋 프린트만 남긴다.

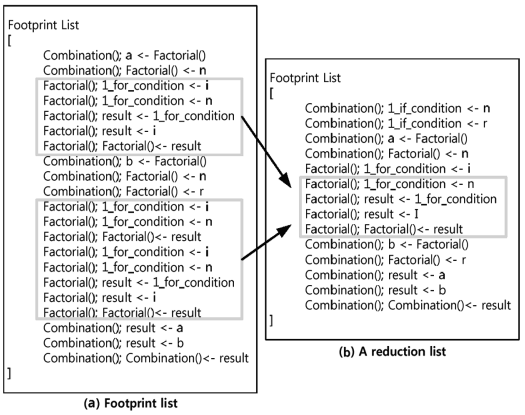


그림 4 풋 프린트 리스트와 중복을 제거한 예
Fig. 4 An example of footprint list and reduced result

3.4 (4 단계) 가정 분석

가정 분석이란 유닛 테스트를 통해 확인하고자 하는 값에 영향을 미치는 요소를 분석하는 작업이다. 다시 말해, 가정이란 하나의 유닛 테스트 케이스로부터 테스터가 확인하고자 하는 값이라 할 수 있다. 일반적으로 유닛 테스트 케이스는 그림 5(a)와 같은 형태로 작성하게 되는데, 하단의 가정문(Assumption())을 통해 확인하고자 하는 값을 확인할 수 있다.

RT-Selection은 특별히 변경이 기존 기능에 영향을 주었는지에 관해 관심이 있다. 이를 확인하기 위해 RT-Selection은 소프트웨어의 변경으로부터 영향을 받은 테스트 케이스를 식별하고, 식별된 테스트 케이스를 이용하여 회귀 테스팅함으로써 기존 기능이 영향을 받지 않았다는 것을 증명할 필요가 있다.

가정 분석은 변경으로부터 영향을 받은 테스트 케이스를 식별하기 위한 첫 번째 작업으로써, 하나의 테스트 케이스 내에서 확인하고 싶은 값(가정문)에 영향을 미치는 요소가 무엇인지를 식별 하는 과정이다. 추후 풋 프린트 리스트와 연계해서 추론을 하게 되면 변경이 테스트 케이스에 영향을 주었는지를 식별할 수 있게 된다.

그림 5(a)는 유닛 테스팅을 위한 테스트 케이스를 나타내고 (b)는 가정 분석을 통해 얻은 요소를 보여준다. (a)의 가정문 *"Assumption(expected_value==2)"*을 보면 expected_value가 2와 동일한지 확인하는 것을 확인할 수 있다. 또한, 가정문 바로 위 코드에서 expected_value에 Combination() 함수의 반환값이 영향을 주는 것을 확인할 수 있다(Guideline 1). 위와 같은 분석을 반복 수행하여, 테스트 케이스 내에 더 이상 영향을 주는 요소가 없을 때까지 진행한다. 마지막으로 테스터는 테스트 케이스 Unit_test_1의 결과에 영향을 미치는 요소가 Combination()의 반환값이라는 것을 식별하게 된다. 식별된 요소를 리스트 형태로 만들면 가정 분석 리스트가 된다(그림 5(b)).

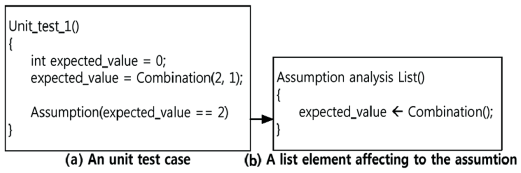


그림 5 유닛 테스트 케이스와 가정 분석 리스트의 예
Fig. 5 An example of a unit test case and an assumption analysis list

Guideline 7. 만약 테스트 케이스 내에 Guideline 2~5와 같은 구성을 보일 경우, 영향을 주었다는 기호 "←"를 이용하여 요소간의 영향을 모두 기록한다.

3.5 (5 단계) 변경 영향 분석

변경 영향 분석은 소프트웨어의 어떤 요소가 테스트 케이스의 가정에 영향을 주는지 추론을 통해 변경에 의해 영향을 받은 테스트 케이스를 식별 하는 활동이다. 다시 말해 3단계에서 생성된 풋 프린트 리스트 중 실제 테스트 케이스의 가정에 영향을 주는 요소만 찾는 것이 변경 영향 분석이다. 3단계에서 생성된 풋 프린트 리스트는 테스트 케이스에 의해 실행된 모든 요소들의 단순 나열이다. 하지만 이 요소가 모두 가정에 영향을 주었다는 것을 의미하는 것은 아니다. 이 중 일부 요소들 만이 가정 분석 리스트의 요소에 영향을 주게 되는데, 이는 풋 리스트 요소 중에 하나가 변경에 의해 영향을 받았다 해도, 이 요소가 테스트 케이스의 가정에 영향을 주지 않았을 수 있는 것이다. 만약 변경이 테스트 케이스의 결과에 영향을 주지 않았다면, 해당 테스트 케이스는 변경된 부분을 테스팅 하지 않는다는 것을 의미한다. 따라서 회귀 테스팅 시 이전과 동일한 결과를 낼 것이라 자연스럽게 예상할 수 있고, 이런 테스트 케이스는 다시 테스트할 필요가 없는(회귀 테스팅을 할 필요가 없는) 테스트 케이스라 할 수 있다.

비용 절감적인 회귀 테스팅을 위해 변경을 테스팅 할 수 있는 최소한의 테스트 케이스만을 선택하는 것이 RT-Selection의 목적이기 때문에 실제 영향을 주지 않은 테스트 케이스는 필요가 없다. 따라서 보다 정확하고 확실한 테스트 케이스를 선택하기 위해 테스트 케이스의 가정에 실제로 영향을 주는 요소를 찾아 테스트 케이스에 영향을 주는지 분석하는 작업이 필요하다.

RT-Selection은 실제로 영향을 주는 요소들을 찾기 위해 추론 규칙(Inference rule)을 사용한다. 추론 규칙이란 전제로부터 결론을 얻는 행위로써, 전제를 얻고 그 의미를 분석해서 결론을 얻는다[21]. RT-Selection에서 쓰인 추론 규칙은 가정 분석 리스트의 요소들과 풋 프린트 리스트를 이용하여 가정 분석 리스트의 요소에 실제로 영향을 주는 요소들을 하나씩 찾아 내는 것이다.

그림 6(a)를 보면 추론 규칙의 과정을 확인할 수 있다. 마지막 줄에 있는 *"expected_value ← Combination()"*는 가정 분석 리스트에 적혀진 요소로써 추론의 시작점이다. 이를 바탕으로 풋 프린트의 요소인 *"Combination() ← result"*을 통해 Combination()에 영향을 미치는 요소가 result임을 추론 할 수 있다. 이어서 *"result ← a"*와 *"sum ← b"*을 통해 *sum*에 영향을 주는 요소가 *a*와 *b*임을 추론 할 수 있다. 더 이상 추론할 요소가 없을 때까지 계속 추론을 하고, 추론한 요소들을 모아 리스트를 만들면 변경 영향 분석 리스트를 작성할 수 있다. 추론을 하기 위해 RT-Selection에서 사용한 기호는 총 3가지 이고, 표 1을 통해 표기방법과 그 의미를 확인할 수 있다.

표 1 추론 규칙을 위한 구문과 의미

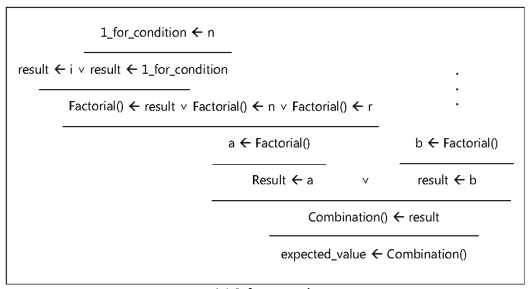
Table 1 Syntax and descriptions for inference rule

syntax	Description
a ← b	The element a affects the Element b
A ∨ B	premises A or premises B
A ⊢ B	If the premises A then premises B

Guideline 8. 추론을 하기 위한 시작점은 가정 분석 리스트의 요소이다.

Guideline 9. 풋 프린트의 기호 "←"의 왼쪽 편 요소를 영향을 받는 요소로 식별하고, 오른쪽 편을 영향을 주는 요소로 식별한다. 하나의 풋 프린트에서 오른쪽 편의 요소가 다른 풋 프린트의 왼쪽 편에 존재할 경우 두 풋 프린트의 관계는 추론이 가능한 관계이다.

Guideline 10. 영향을 주는 요소가 두 개 이상일 경우 기호 "∨"을 이용해 표현할 수 있고, 두 요소를 모두 추론한다.



(a) Inference rule

```
Change impact analysis list
[
        Factorial(); n
        Factorial(); r
        Factorial(); 1_for_condition
        Factorial(); i
        Factorial(); result
        Combination(); Factorial()
        Combination(); a
        Combination(); b
        Combination(); result
        Combination()
]
```

(b) Change impact analysis list

그림 6 추론규칙과 변경 영향 분석 리스트의 예

Fig. 6 An inference rule and corresponding change impact analysis list

3.6 (6단계) 텍스트 차이점과 변경 분석

텍스트 차이점은 소프트웨어의 변경된 부분을 식별하기 위해 수행하는 것으로 1단계에서 정형적으로 만든 두 코드를 이용해 텍스트 간에 차이가 있는 부분을 식별한다. 텍스트 비교를 통한 차이점 도출은 코드를 기반으로 하기 때문에 무엇보다도 정확하고 명확하게 변경을 식별할 수 있는 장점이 있다. 현재 텍스트 차이점을 찾을 수 있는 상용도구와 알고리즘이 많이 개발되었고 쉽게 접근

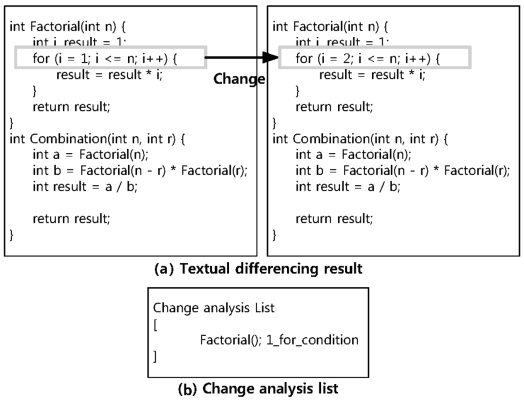


그림 7 텍스트 비교와 변경 분석 리스트의 예
Fig. 7 An example of change analysis list

가능하다(예, Unix의 diff 도구[22]). RT-Selection은 어떤 도구나 알고리즘을 사용하는지 제약을 두지 않는다.

변경을 식별한 뒤, 변경으로 인해 정확히 어떤 요소가 영향을 받았는지 식별된 변경을 분석해야 한다. 이런 변경을 분석하는 작업이 바로 변경 분석이다. 그림 7의 (a)의 두 코드를 대상으로 텍스트 비교를 수행하면 *"for(int i = 1; i < n; i++)"*이 *"for(int i = 2; i < n; i++)"*로 변경되었다는 것을 식별할 수 있다. 여기서 변경으로부터 영향을 받은 요소는 for문의 제어부이다. 영향을 받은 요소들을 모아 리스트 형태로 정리한 것이 변경 영향 분석 리스트이다.

추가적으로 RT-Selection은 추후에 테스트 케이스 선택 시 도움을 주기 위해 2개의 태그를 사용하는 것을 제시한다. 태그의 종류는 "Obsolete", "NewSepc" 두 종류로 해당 태그에 대해 Guideline 12, 13을 통해 그 의미와 사용에 대해 살펴볼 수 있다.

Guideline 11. 변경으로부터 영향을 받는 요소는 다음 중 하나이다. 배치문의 왼쪽 요소, 조건문의 조건부, 반복문의 제어부. 요소의 작성은 Guideline 2~5의 풋 프린트와 동일하게 작성한다.

Guideline 12. 만약 함수의 파라메터가 변경된 경우, 또는 존재하고 있던 함수가 사라진 경우 *"Obsolete"*라는 태그를 삽입 한다. 추후 테스트 케이스 선택 시 해당 태그가 변경 분석 리스트에 나타나는 경우, 해당 테스트 케이스는 더 이상 새로운 코드에 적용 가능하지 않기 때문에 선택의 대상에서 제외한다.

Guideline 13. 만약 기존에 없던 요소(배치문, 함수 반환문, 함수 호출문, 조건문, 반복문)가 새로운 코드에 생겼을 경우 *"NewSpec"*이라는 태그를 삽입한다. 추후 테스트 케이스 선택 시 해당 태그가 변경 분석 리스트에 나타나는 경우, 해당 테스트 케이스는 회귀 테스팅을 위해서 선택해야 하는 테스트 케이스이고 특별히 테스터의 주의를 필요한 하는 테스트 케이스이다.

3.7 (단계 7) 일치성 비교

일치성 비교는 RT-Selection의 마지막으로 단계로서 회귀 테스팅을 위한 테스트 케이스를 선택 및 분류하는 단계이다. 일치성 비교는 각각의 리스트에 동일한 요소가 존재하는지 단순 비교를 통해 확인하는 활동이다. 단계 5의 변경 영향 분석 리스트와 단계 6의 변경 분석 리스트를 이용하여 상호간에 일치하는 요소가 있는지 확인한다. 만약, 일치하는 요소가 존재 한다면 해당 테스트 케이스는 회귀 테스팅을 위한 테스트 케이스로 분류 및 선택 되게 된다.

그림 8을 보면 변경 영향 분석 리스트와 변경 분석 리스트에 *"Factorial(); 1_for_condition"*라는 요소가 동시에 나타나는 것을 확인할 수 있다. 이렇게 상호간에 동일한 요소가 존재한다는 것은 다음과 같이 말할 수 있다. "해당 테스트 케이스는 변경이 이루어진 부분을 실행하는 테스트 케이스(변경 분석 리스트에 해당 요소가 존재)이고, 변경이 테스트 케이스의 결과에 영향을 미치기 때문에(변경 영향 분석 리스트에 해당 요소가 존재)는 회기 테스팅을 통해 다시 테스트해야 되는 테스트 케이스이다".

Guideline 14. *"Obsolete"* 태그가 있을 경우 해당 테스트 케이스는 더 이상 새로운 소프트웨어에 사용할 수 없는 테스트 케이스로써 추후 사용이 불가능한 테스트 케이스로 분류한다.

Guideline 15. *"NewSpec"* 태그가 있을 경우 해당 테스트 케이스는 회귀 테스팅을 위한 테스트 케이스로 선택될 뿐만 아니라, 추가적으로 소프트웨어의 명세를

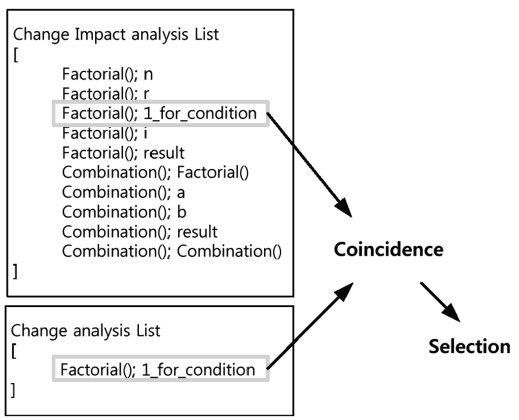


그림 8 변경 영향 분석 리스트와 변경 분석 리스트의 비교를 통해 일치하는 요소를 찾는 예
Fig. 8 A comparing example to identify coincided elements between change analysis list and change impact analysis list

확인하거나 테스트 케이스의 요구사항을 확인해 볼 필요가 있는 테스트 케이스로 분류할 수 있다.

Guideline 16. "*Obsolete*" 와 "*NewSpec*" 가 동시에 있을 경우 우선순위를 "*Obsolete*" 로 한다. 따라서 두 태그가 동시에 나올 경우 해당 테스트 케이스를 회귀 테스팅에 사용이 불가능한 테스트 케이스로 분류한다.

4. 케이스 스터디

우리는 RT-Selection을 이용하여 본교 대학원의 *Software Test Management(SWTM)* [23] 수업과 본교 대학의 *Introduction to Software Engineering(SE)* [24] 수업을 통해 케이스 스터디를 수행하였다. SE 수업을 통해 학생들은 Structured Analysis and Structured Design(SASD) 방법론을 이용하여 소프트웨어를 설계부터 디자인, 구현하는 방법을 배우고 이를 이용하여 Digital Watch System(DWS) 프로그램을 만들었다. SWTM 수업 학생들은 SE 수업 학생들이 만든 DWS 프로그램을 대상으로 시스템 테스팅과 유닛 테스팅을 수행 하였다. 또한, SE 수업 학생들은 SWTM 수업 학생들의 테스팅 결과를 반영하여 수정된 프로그램을 반복하여 만들었다.

4.1 DWS 시스템

DWS 프로그램은 C언어를 기반으로 하고, 콘솔에 그래픽을 출력하는 프로그램이다. 구현하고자 하는 DWS의 실제 모습은 그림 9와 같다. DWS 프로그램은 시계의 버튼 기능을 키보드의 a, b, c, d의 입력으로 대체, 시간을 표현하는 디스플레이 부분은 콘솔의 문자를 출력하는 것으로 대체, 백라이트는 출력되는 문자의 색을 바꾸는 것으로 대체, 알람음은 컴퓨터의 비프음으로 대체하여 구현되었다. 상태 모드는 현재 시간을 출력해 주는 디스플레이 모드, 스탑워치 기능을 수행할 수 있는 스탑워치 모드, 시간을 설정할 수 있는 설정 모드를 가지고 있는 프로그램이다.

4.2 RT-Selection 결과

우리는 총 9팀에 대해 RT-Selection을 적용해 보았다. 그 결과를 표 2를 통해 확인할 수 있다. 표 2의 첫 번째 행의 Total test case for old version은 변경이 생기기 전 수행하였던 기존 테스트 케이스의 수이다. 두

그림 9 DWS 프로그램 예제[25]
Fig. 9 An example of DWS program [25]

번째 행의 Obsolete test cases는 "Obsolete"태그가 나타나는 테스트 케이스로써 더 이상 새로운 소프트웨어에 적용할 수 없는 테스트 케이스이다. 세 번째 행의 NewSpec test cases는 "NewSpec"태그가 나타나는 테스트 케이스로써 회귀 테스팅을 위해 선택해야 하는 테스트 케이스이고 특별히 주의를 기울여야 하는 테스트 케이스 이다. 네 번째 행의 Re-testable test cases는 소프트웨어의 변경에 의해 영향을 받는 부분을 실행하는 테스트 케이스로써, 새로운 소프트웨어가 변경에 영향을 받지 않았다는 것을 보장하기 위해 회귀 테스팅 시 반드시 다시 수행해야 하는 테스트 케이스이다. 마지막 행의 Candidate for regression testing은 NewSpec 테스트 케이스와 Re-testable 테스트 케이스 합친 테스트 케이스의 수로써 RT-Selection이 찾으려 하는 회귀 테스팅 시 다시 테스팅 해야 하는 테스트 케이스이다.

결과를 자세히 살펴보면, T4의 경우 변경이 이루어지기 전 소프트웨어를 위해 작성된 테스트 케이스의 수는 총 72였고, RT-Selection을 적용한 결과 72개중 18개의 테스트 케이스에서 "Obsolete"라는 태그가 삽입되어 나타났다. 이는 회귀 테스팅을 위해 선택할 필요가 없는 테스트 케이스로 분류 된다. 72개중 2개의 테스트 케이스에서 변경 영향 분석 리스트와 변경 분석 리스트에서 동일한 요소가 식별되었다. 이는 해당 테스트 케이스가 변경이 이루어진 부분을 실행하였고 변경이 테스트 케이스의 결과에 영향을 미쳤다는 것을 의미한다. 따라서 회귀 테스팅을 통해 다시 테스팅 되어야 하는 테스트 케이스로 선정할 수 있다. 나머지 52개의 테스트 케이스는 변경된 부분을 실행하지 않았거나 변경이 테스트 케이스의 결과에 영향을 주지 않았다는 것을 의미하기 때

표 2 RT-Selection을 이용한 케이스 스터디 결과
Table 2 A result of case study using RT-Selection

	T1	T2	T3	T4	T5	T6	T7	T8	T9
Test cases for old version	55	49	50	72	54	57	35	73	50
Obsolete test cases	-	-	4	18	-	-	3	18	9
NewSpec test cases	18	-	-	-	-	11	-	-	-
Re-testable test cases	-	-	-	2	5	-	7	-	5
Candidate for regression testing	18	-	-	2	5	11	7	-	5

문에 다시 수행할 필요가 없는 테스트 케이스로 분류된다.

T1과 T6는 반대로 “NewSpec”태그만 나온 것을 확인 할 수 있다. 이와 같은 경우 회귀 테스팅 시 해당 테스트 케이스를 반드시 수행해야 되는 것은 물론이고, 테스터는 보다 신뢰성 있는 회귀 테스팅을 수행하기 위해 소프트웨어 또는 테스트 케이스의 요구사항을 다시 한 번 확인해야 한다는 결론을 얻을 수 있다.

T2와 같은 경우는 회귀 테스팅 시 필요 없는 테스트 케이스나 다시 테스트 해야 될 테스트 등 아무런 분류가 되지 않은 것을 확인할 수 있다. 이는 소프트웨어에 변경이 발생하지 않았거나 또는 변경이 발생 했지만 해당 변경을 테스트 하는 테스트 케이스가 존재 하지 않다는 것을 의미한다. T2의 결과를 미루어보아 만약 T2가 만든 프로그램에도 변경이 생겼다면, 기존 작성된 테스트 케이스가 T2의 소프트웨어를 테스팅 하기에 충분하지 않은 커버리지를 가지고 있었다는 것을 간접적으로 알 수 있다. 무엇보다 먼저 소프트웨어를 충분히 테스트할 수 있는 테스트 케이스가 있어야 의미 있는 회귀 테스팅을 수행 할 수 있다는 것을 간접적으로 알 수 있는 부분이다.

5. 결론 및 향후 연구

회귀 테스팅은 소프트웨어의 변경이 새로운 버그나 에러를 만들었는지 확인하는 활동이다. 하지만 소프트웨어의 규모와 복잡성이 점차 커지면서 회귀 테스팅의 비용 역시 커지고 있다. 이 문제를 해결하기 위해 본 논문은 텍스트 차이점과 변경 영향 분석 방법을 사용하여 비용 절감적으로 회귀 테스팅을 수행할 수 있는 방법인 RT-Selection을 제시했다. RT-Selection은 총 7개의 단계로 이루어져 있고, 각각의 단계를 보다 체계적으로 수행 할 수 있도록 총 16개의 가이드라인과 추론 규칙을 포함하고 있다. 우리는 RT-Selection을 이용해 케이스 스터디를 수행하였고 적용 가능성을 보였다.

자동화 도구 구현 측면에서, RT-Selection은 다음과 같은 특징 및 장점을 가지고 있다. 첫째, RT-Selection은 기존의 도구를 재 사용하여 수행할 수 있다. 1단계의 ‘정형화된 형태 만들기’와 6단계의 ‘텍스트 차이점과 변경 분석’과 같은 경우, 기존 존재하는 도구를 재 이용하여 이를 수행할 수 있다(예, eclipse의 code formatting 기능[26], BeautyJ[27], Linux/Unix의 diff 도구[22], 온라인 사이트 [28] 등). 둘째, 분석 결과는 리스트 형태로 출력이 되기 때문에 컴퓨터를 이용한 단순 비교가 가능하다. 5단계의 ‘변경 영향 분석’과 7단계의 ‘일치성 비교’와 같은 경우, 리스트의 요소들에 대한 단순 비교를 통해 결과를 쉽게 얻을 수 있다. 셋째, 자연스러운 결과 획득이 가능하다. 3단계의 ‘테스트 수행’과 같은 경우, 기존 테스트의 단순 재 수행을 통해 결과를 쉽게 얻을 수 있다. 넷째, 본 논문에서 제시한 16개의 가이드라인은 자동화 도구 구현 시, 도구의 핵심 요구사항으로 역할을 하게 된다. 2단계의 ‘코드 분석과 풋 프린트 삽입’과 4단계의 ‘가정 분석’과 같은 경우, 가이드라인 2~5는 구현될 도구의 핵심 요구사항으로 작용할 것이다. 다섯째, RT-Selection 은 구현될 도구의 형태나 환경에 대한 제약이 없다. 구현은 개발자의 익숙한 언어와 환경을 이용하여 구현 가능하다. 또한, 사용자의 의한 추가적인 guideline 정의를 통해 RT-Selection의 기능 또한 확장이 가능하다.

현재 우리는 자동화 도구를 구현 계획 중에 있으며, 자동화 도구의 개발이 완료된다면, RT-Selection의 효율성이 보다 크게 증가할 것이라 기대하고 있다.

References

[1] S. Yoo, M. Harman, "Regression testing minimization, selection and prioritization: a survey," *journal of Software Testing, Verification and Reliability*, vol.22, no.2, pp.67-120, 2012.

[2] H. Do, S. Mirarab, L. Tahvildari, G. Rothermel, "The effects of time constraints on test case prioritization: A series of controlled experiments," *Journal of IEEE Transactions on Software Engineering*, vol.36, no.5, pp.593-617, Sep/Oct. 2010.

[3] G. Rothermel, M. J. Harrold, "Selecting tests and identifying test coverage requirements for modified software," *Proc. of the ACM SIGSOFT international symposium on Software testing and analysis (ISSTA)*, pp.169-184, 1994.

[4] R. S. Arnold, S. A. Bohner, "Impact analysis - towards a framework for comparison," *Proc. of the Conference on the Software Maintenance (CSM)*, pp.292-301, Sep. 1993.

[5] I. Sommerville, SOFTWARE ENGINEERING (8th), Pearson College Div, 2006.

[6] G. Rothermel, M. J. Harrold, "A safe, efficient regression test selection technique," *journal of ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol.6, no.2, pp.173-210, Apr. 1997.

[7] J. Bible, G. Rothermel, D. S. Rosenblum, "A comparative study of coarse-and fine- grained safe regression test-selection techniques," *Journal of ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol.10, no.2, pp.149-183, Apr. 2001.

[8] S. Biswas, R. Mall, M. Satpathy, S. Sukumaran, "Regression test selection techniques: A survey," *Journal of Informatica: An International Journal of Computing and Informatics*, vol.35, no.5, pp.289-321, Sep. 2011.

[9] G. Rothermel, M. J. Harrold, "Analyzing regression

test selection techniques," *Journal of IEEE Transactions on Software Engineering*, vol.22, no.8, pp.529-551, Aug. 1996.
[10] E. Emelie, S. Mats, R. Per, "Empirical evaluations of regression test selection techniques: a systematic review," *Proc. of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM)*, pp.22-31, Oct. 2008.
[11] F. Vokolos, P. Frankl, "Pythia: a regression test selection tool based on textual differencing," *Proc. of the ceedings of the International ConfeReliability, Quality and Safety of Software-Intensive Systems (ENCRESS)*, pp.3-21, May. 1997.
[12] D. Binkley, "Reducing the cost of Regression Testing by Semantics Guided Test Case Selection," *Proc. Conf. Software Maintenance*, pp.251-260, Oct. 1995.
[13] H. Agrawal, J. Horgan, E. Krauser, and S. London, "Incremental Regression Testing," *Proc. Conf. Software Maintenance*, pp.348-357, Sept. 1993.
[14] W. E. Wong, J. R. Horgan, S. London, A. P. Matur, "Effect of test set minimization on fault detection effectiveness," *Proc. of the 17th International Conference on Software Engineering (ICSE)*, pp.41-41, Apr. 1995.
[15] J. A. Jones, M. J. Harrold, "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage," *Journal of IEEE Transactions on Software Engineering*, vol.29, no.3, pp.195-209, Mar. 2003.
[16] M. J. Harrold, R. Gupta, M. L. Soffa, "A methodology for controlling the size of a test suite," *Journal of ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol.2, no.3, pp.270-285, Jul. 1993.
[17] G. Rothermel, R. H. Untch, C. Chu, M. Harrold, "Prioritizing Test Cases For Regression Testing," *Journal of IEEE Transactions on Software Engineering*, vol.27, no.10, pp.929-948, Oct. 2001.
[18] J. M. Kim, A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," *Proc. of the 24rd International Conference on Software Engineering (ICSE)*, pp.119-129, May. 2002.
[19] Z. Li, M. Harman, R. M. Hierons, "Search algorithms for regression test case prioritization," *Journal of IEEE Transactions on Software Engineering*, vol.33, no.4, pp.225-237, Apr. 2007.
[20] A. Srivastava, J. Thiagarajan, "Effectively prioritizing tests in development environment," *Proc. of ACM SIGSOFT Software Engineering Notes*, vol.27, no.4, pp.97-106, Jul. 2002.
[21] Wikipedia, Rule of inference, [Online]. Available: http://en.wikipedia.org/wiki/Rule_of_inference
[22] Wikipedia, Diff, [Online]. Available: http://en.wikipedia.org/wiki/Diff
[23] D. S. Laboratory, Introduction of software engineering, [Online]. Available: http://dslab.konkuk.ac.kr/Class/201 2/12SE/12SE.htm
[24] D. S. Laboratory, Software testing management, [Online]. Available: http://dslab.konkuk.ac.kr/Class/201 2/12SM/12SM.htm
[25] Digital Watch Library, QW-1219, [Online]. Available: http://www.digitalwatchlibrary.com/images/casio_manuals/qw1219.pdf
[26] Eclipse - The Eclipse Foundation opens source community website, [Online]. Available: http://www.eclipse.org
[27] BeautyJ - Java Source Code Transformation Tool, [Online]. Available: http://beautyj.berlios.de/index.html
[28] An Online Tool to do a 'quick and dirty' diff of two text or code fragments, [Online]. Available: http://www.quickdiff.com/

김 의 섭
2012년~현재 건국대학교 컴퓨터·정보통신공학과 석사과정. 관심분야는 소프트웨어 공학, 안전성 분석, 정형기법, 회귀 테스팅

이 동 아
2012년~현재 건국대학교 컴퓨터·정보 통신공학과 박사과정. 관심분야는 소프트웨어 공학, 안전성 분석, 정형기법

유 준 범
2005년 KAIST 전자전산학과 전산학전공 박사. 2005년~2008년 삼성전자주식회사 통신연구소 책임연구원. 2008년~현재 건국대학교 컴퓨터공학부 부교수. 관심분야는 소프트웨어 공학, 안전성 분석, 정형기법