인사이드 이슈

소프트웨어공학센터 경영지원TF팀

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

2014. 9. 30. [제108호]

김의섭(Eui-Sub Kim), 이동아(Dong-Ah Lee), 유준범(Junbeom Yoo)

Journal of KIISE. Software and applications v.41 no.6

Contents

정보통신산업진흥원 부설
nipa 소프트웨어공학센터

Ⅰ. 서론

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 기존 테스트 케이스를 이용하여 변경이 이루어진 소프트웨어를 다시 테스팅 하는 것이다. 하지만 이 방법은 많은 시간과 비용(테스터의 시간과 노력, 자원 등)을 필요로 하는 단점이 있다. 문제는 소프트웨어의 크기와 복잡도가 점점 증가함에 따라 테스팅 비용 또한 증가 하고 있다는 점이다. 따라서 이런 테스트 비용을 줄이기 위한 전략적인 방법이 필요한 상황이고, 본 원고는 특히 회귀 테스팅 시 드는 비용을 줄이기 위해 회귀 테스트 선택(Regression Test Selection) 기법을 이용하여 비용 절감적인 회귀 테스팅을 수행할 수 있는 RT-Selection을 제시한다.

Rothermel와 Harrold[3]는 회귀 테스팅을 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 T의 서브 셋을 이용해서 보여 주는 것.” 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 셋을 찾는 작업이다. RT-Selection은 위 두 작업을 텍스트 차이점과 변경 영향 분석을 이용해 수행한다.

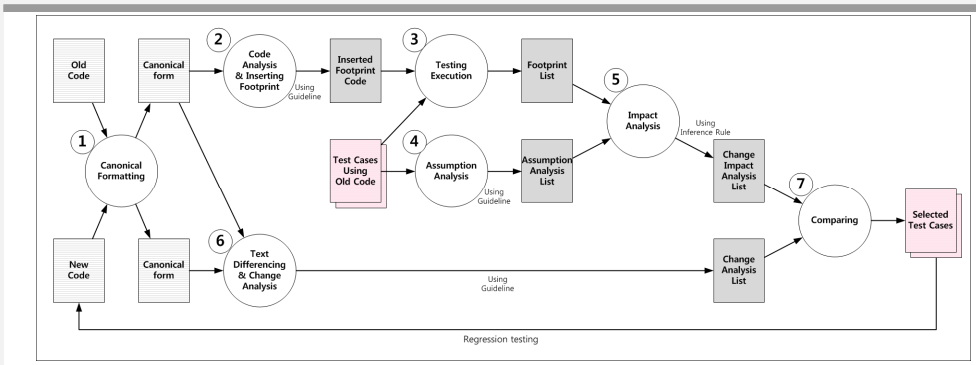
첫째, RT-Selection은 소프트웨어의 변경을 식별하기 위해 텍스트 차이점을 이용한다. 텍스트 차이점은 변경이 이루어지기 전과 후, 두 소프트웨어의 소스 코드를 대상으로 텍스트 단위의 정적인 비교를 통해 식별한다. 둘째, RT-Selection은 소프트웨어의 변경 전과 후의 일치성을 보장하기 위한 서브 셋 T를 찾기 위해 변경 영향 분석[4] 을 이용한다. 변경 영향 분석이란 변경으로부터 잠재적으로 영향을 받는 소프트웨어의 부분을 식별하고, 식별된 부분으로부터 영향을 받는 부분들을 다시 식별해 나가는 방법이다.

RT-Selection은 소프트웨어의 영향 관계를 표현 할 수 있도록 풋 프린트(footprint)의 삽입을 제안한다. 풋 프린트가 삽입된 코드와 기존 테스트 케이스를 이용, 다시 테스팅을 수행하면 풋 프린트 리스트를 얻을 수 있다. 또한, 텍스트 비교를 통해 식별된 변경과 풋 프린트 리스트를 이용, 추론 작업을 거치면 회귀 테스팅을 위한 테스트 케이스 셋을 얻을 수 있다.

II. RT-Selection의 과정

제시하는 RT-Selection은 총 7단계로 구성되어 있다. 그림 1은 각 단계들의 흐름을 보여주며, 앞으로 각 각 단계에 대한 자세한 내용을 설명한다.

그림 1_RT-Selection의 전체 과정



(1단계) 정형화된 형태 만들기

정형화된 형태로 만든다는 것은 서로 다른 코드를 구문적으로 동일한 형태로 만드는 것이다. RT-Selection은 소프트웨어의 변경을 식별하기 위해 텍스트 차이를 이용하는데, 단순 텍스트 차이는 불필요한 요소 까지 비교한다는 단점이 있다. 예를 들면, 괄호의 위치나 띄어쓰기, 공백, 내려쓰기 등 사소한 코딩 스타일 차이가 모두 비교 대상이 된다. 하지만 이런 단순 스타일 차이는 실제 소프트웨어의 기능에 영향을 주지 않기 때문에, 변경이 생겨도 다시 테스팅을 할 필요가 없다. 따라서 이런 불필요한 요소를 제거하기 위해 스타일이 상이한 두 코드를 하나의 정형화된 형태로 일치시켜 줄 필요가 있다.

그림 2의 (A)와 (B)는 기능이 일치하지만 서로 다른 스타일로 작성된 코드이다. 또한, 정형화된 형태 (그림 2의 (C)) 로 만들면 정확하게 일치 하다는 것을 확인할 수 있다. 하지만 (A)와 (B) 두 코드를 이용해 텍스트 비교를 하게 되면, 수많은 차이점이 생성된다. 이런 불필요한 작업을 줄이기 위해 하나의 정형화된 형태로 만들어줄 필요가 있다.

Guideline 1. RT-Selection 은 기능적으로 의미 없는 부분에 의한 텍스트 차이점을 제거하기 위해 하나의 정형화된 스타일로 만드는 것을 제안한다. 하나의 정형화된 스타일로 만들 수 있다면 어떤 스타일로 정형화 하는지, 어떤 도구를 사용하여 정형화 하는지에 대해서는 제약을 두지 않는다.

그림 2_스타일이 다른 두 코드에 정형화를 수행하는 예

```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 the canonical form

(2단계) 코드 분석과 풋 프린트 삽입

풋 프린트란 코드 상에서 "어떤 요소가 어떤 요소에 영향을 주는지 또는 받는지"를 표현 하는 식별자 이다. 소프트웨어는 실행 중에 다양한 요소들이 서로 영향을 주고받게 되는데, 그 중 우리가 관심을 갖는 요소를 중심으로 상호 어떤 영향을 주고받았는지를 알 수 있도록 하는 것이 풋 프린트의 역할이다. 풋 프린트의 삽입을 위해서는 코드를 정적으로 분석 할 필요가 있다. RT-Selection에서 관심을 갖고 분석 하는 요소는 다음과 같다. (배치문, 함수 반환문, 함수 호출문, 조건문의 조건부, 반복문의 제어부.)

상기 요소들을 식별하고 해당 구문에 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 해석 할 수 있다.

그림 3_풋 프린트가 삽입된 코드의 예

```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;
}
```

Guideline 2. 코드가 "Left element = Right element"와 같은 형태의 배치문일 경우 "Left element ← Right element"와 같은 풋 프린트를 삽입한다. 코드가 "return something"과 같은 형태의 함수 반환문일 경우 "function name() ← something"와 같은 풋 프린트를 삽입한다. 또한, 코드가 함수 호출문일 경우 "function name() ← parameter"을 삽입 한다.

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

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

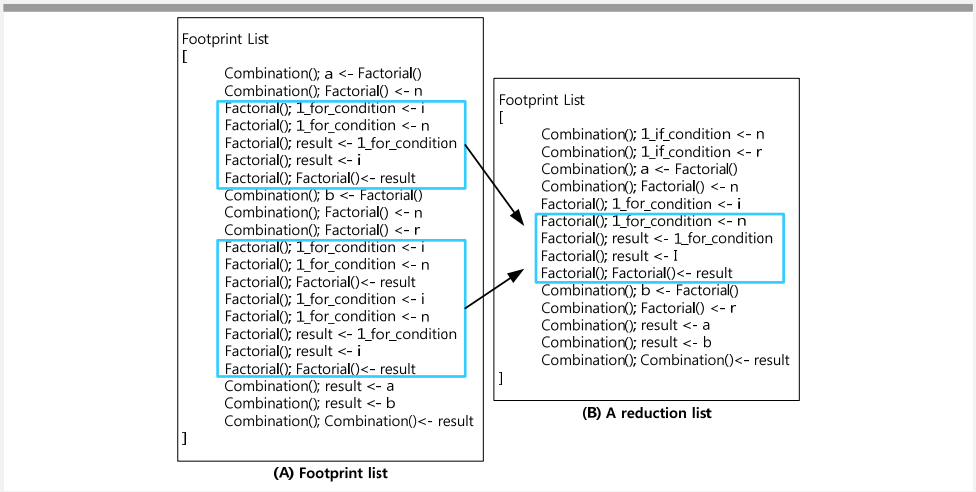
Guideline 5. Guideline 2 ~ 4가 중복적으로 사용될 경우 풋 프린트를 열거를 하여 사용

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

(3단계) 테스트 수행

RT-Selection에서 전제로 삼고 있는 테스팅 방법은 유닛 테스팅(Unit testing)이다. 유닛 테스팅은 테스팅 하고자 하는 유닛에만 관심을 갖고 있기 때문에 해당 유닛만 집중하여 테스팅 할 수 있다는 장점이 있다. 기존 테스트 케이스와 풋 프린트가 삽입된 코드를 이용하여 유닛 테스팅을 수행하면 자연스럽게 그림 3의 (A)와 같은 풋 프린트 리스트를 얻을 수 있다. 풋 프린트 리스트는 해당 테스트 케이스가 소프트웨어의 어떤 부분을 수행하였는지, 수행 중에 어떤 요소들이 영향을 주고받았는지 식별 가능하게 한다.

그림 4_풋 프린트 리스트와 중복을 제거한 예



하지만 생성된 풋 프린트 리스트는 다수의 중복을 가질 수 있다. 예를 들어 반복문이나 재귀함수와 같은 내용이 유닛 안에 있을 경우, 반복된 횟수만큼의 중복된 풋 프린트들을 생성 하게 될 것이다. 따라서 이런 중복을 제거할 필요가 있다. 그림 4의 (B)는 중복을 제거한 최적화된 결과를 나타낸다.

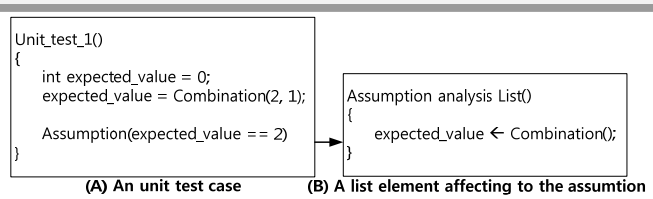
Guideline 6. 풋 프린트 리스트에 중복된 풋 프린트가 있을 경우 이를 제거한다.

(4단계) 가정 분석

가정 분석이란 유닛 테스트를 통해 확인하고자 하는 값에 영향을 미치는 요소를 테스트 케이스 내에서 분석하는 작업이다. 일반적으로 유닛 테스트 케이스는 그림 5의 (A)와 같은 형태로 작성하게 되는데, 하단의 가정문 (Assumption())을 통해 확인하고자 하는 값을 확인 할 수 있다. 그림 5의 (A)는 유닛 테스팅을 위한 테스트 케이스이고, (B)는 가정

분석을 통해 얻은 요소이다. (A)의 가정문 "Assumption(expected_value==2)"은 expected_value가 2와 동일한지 확인하고 있다. 가정문 바로 위 라인에서 expected_value에 calc() 함수의 반환 값이 영향을 주는 것을 볼 수 있다(Guideline 2). 위와 같은 분석을 테스트 케이스 내에 더 이상 영향을 주는 요소가 없을 때까지 반복 수행한다. 예제의 경우 최종적으로 결과에 영향을 미치는 요소가 Calc()의 반환 값 이라는 것을 식별 할 수 있다. 식별된 요소를 모아 리스트 형태로 만들면 가정 분석 리스트가 된다(그림 5의 (B)).

그림 5_유닛 테스트 케이스와 가정 분석 리스트의 예



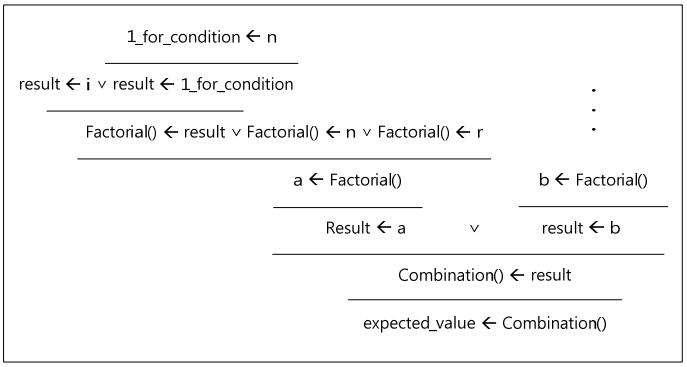
Guideline 7. 테스트 케이스 내에 요소가 Guideline 2~4와 같은 구성을 보일 경우, 영향을 주었다는 기호 '←'를 이용하여 요소간의 영향을 기록한다.

(5단계) 변경 영향 분석

변경 영향 분석은 소프트웨어의 어떤 요소가 테스트 케이스의 가정에 영향을 주었는지 풋 프린트 리스트를 이용해 식별 하는 작업이다. 풋 프린트 리스트란 테스트 케이스에 의해 실행된 소프트웨어의 요소들 간 상호 영향을 나타내는 지표인데, 3단계에서 생성된 풋 프린트 리스트는 테스트 케이스에 의해 실행된 요소들의 단순 나열이다. 만약 풋 리스트 요소 중에 하나가 소프트웨어의 변경에 의해 영향을 받았다 하더라도, 이 요소가 테스트 케이스의 가정에 영향을 주지 않는 요소라면 회귀 테스팅 시 이전과 동일한 결과는 낼 것이라 예상할 수 있다. 이런 테스트 케이스는 다시 테스트 할 필요가 없는(회귀 테스팅 시 위해 불필요한) 테스트 케이스라 할 수 있다. 비용 절감적인 회귀 테스팅을 위해 변경을 테스팅 할 수 있는 최소한의 테스트 케이스만을 선택하는 것이 RT-Selection 의 목적이기 때문에 이와 같은 테스트 케이스는 회기 테스팅을 위해 선택할 필요가 없다.

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[5]. RT-Selection 에서 쓰인 추론 규칙은 가정 분석 리스트의 요소들과 풋 프린트 리스트를 이용하여 가정 분석 리스트의 요소에 실제로 영향을 주는 요소들을 하나씩 찾아내는 것이다.

그림 6_추론 규칙과 변경 영향 분석 리스트의 예



(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 6의 (A)를 보면 추론 규칙의 과정을 확인 할 수 있다. 마지막 줄에 있는 "expected_value ← Combination()"는 가정 분석 리스트의 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 sum에 영향을 주는 요소가 a와 b임을 추론 할 수 있다. 더 이상 추론할 요소가 없을 때까지 계속 추론을 하고, 추론한 요소들을 모아 리스트를 만들면 변경 영향 분석 리스트를 작성 할 수 있다. 추론을 하기 위해 RT-Selection에서 사용한 기호는 총 3가지이고, 표 1을 통해 표기방법과 그 의미를 확인 할 수 있다.

표 1_추론 규칙을 위한 구문과 의미

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 요소, 오른쪽 편을 영향을 주는 요소로 식별한다. 하나의 풋 프린트에서 오른쪽 편에 요소가 다른 풋 프린

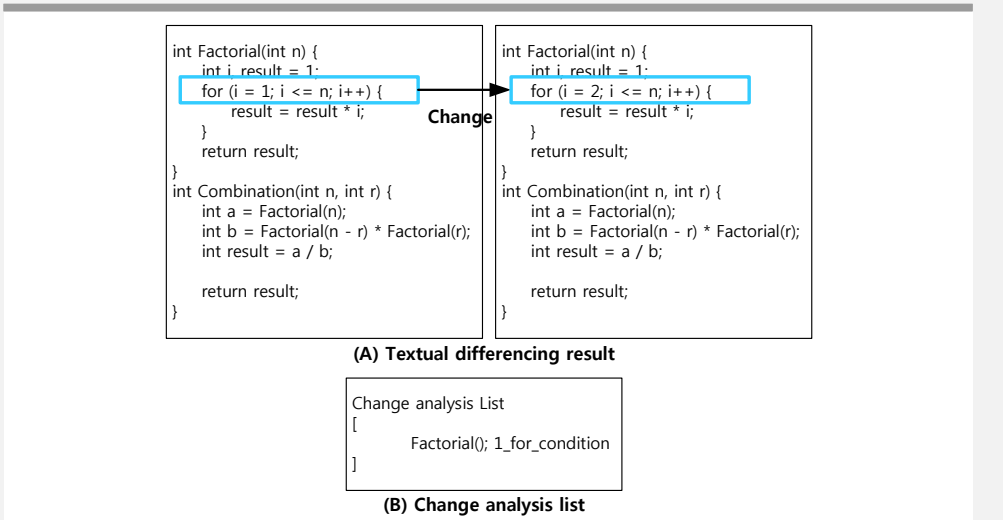
트의 왼쪽 편에 존재할 경우 두 풋 프린트의 관계는 추론이 가능한 관계이다.

Guideline 10. 영향을 주는 요소가 두 개 이상일 경우 기호 'V'을 이용해 표현하고, 두 요소를 모두 추론한다.

(6단계) 텍스트 차이점과 변경 분석

텍스트 차이점은 소프트웨어의 변경된 부분을 식별하기 위해 수행하는 것으로 1단계에서 정형적으로 만든 두 코드를 이용해 텍스트 간 차이가 있는 부분을 식별한다. 텍스트 비교를 통한 차이점 도출은 코드를 기반으로 하기 때문에 정확하고 기계적으로 변경을 식별 할 수 있다는 장점이 있다. 현재 텍스트 차이점을 찾을 수 있는 상용도구와 알고리즘이 많이 개발 되었고 쉽게 접근 가능하다 (예, Unix의 diff 도구 [6]). RT-Selection은 어떤 도구나 알고리즘의 사용에 제약을 두지 않는다.

그림 7_텍스트 비교와 변경 분석 리스트의 예



변경을 식별한 뒤, 변경으로 인해 정확히 어떤 요소가 영향을 받았는지 식별된 변경을 분석 할 필요가 있다. 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 변경 되었다는 것을 식별 할 수 있다. 여기서 변경으로부터 영향을 받은 요소는 for문의 제어부임을 식별 할 수 있다 (Guideline 2~5). 추가적으로 RT-Selection은 추후에 테스트 케이스 선택 시 도움을 주기 위해 2개의 태그를 사용하는 것을 제안한다. 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 작성 방법은 Guideline 2~5를 따른다.

Guideline 12. 유닛의 파라메터가 변경된 경우, 또는 존재하고 있던 유닛이 사라진 경우 "Obsolete" 라는 태그를 삽입 한다. 추후 테스트 케이스를 선택 시 해당 태그가 나타나는 테스트 케이스는 더 이상 새로운 코드에 적용 가능 하지 않기 때문에 제외한다.

Guideline 13. 기존에 없던 요소(배치문, 함수 반환문, 함수 호출문, 조건문, 반복문)가 유닛에 생겼을 경우 "NewSpec" 태그를 삽입한다. 추후 테스트 케이스를 선택 시 해당 태그가 나타나는 테스트 케이스는 회귀 테스팅을 위해서 반드시 선택해야 하는 테스트 케이스이고 특별히 테스터의 주의를 필요한 하는 테스트 케이스로 분류한다.

(7단계) 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 확인 가능하다. 5단계의 변경 영향 분석 리스트와 6단계의 변경 분석 리스트를 이용하여 상호간에 일치하는 요소가 있는지 확인한다. 일치하는 요소가 존재 한다면 해당 테스트 케이스는 회귀 테스팅을 위한 테스트 케이스로 분류 및 선택 할 수 있다.

그림 8_변경 영향 분석 리스트와 변경 분석 리스트의 비교를 통해 일치하는 요소를 찾는 예

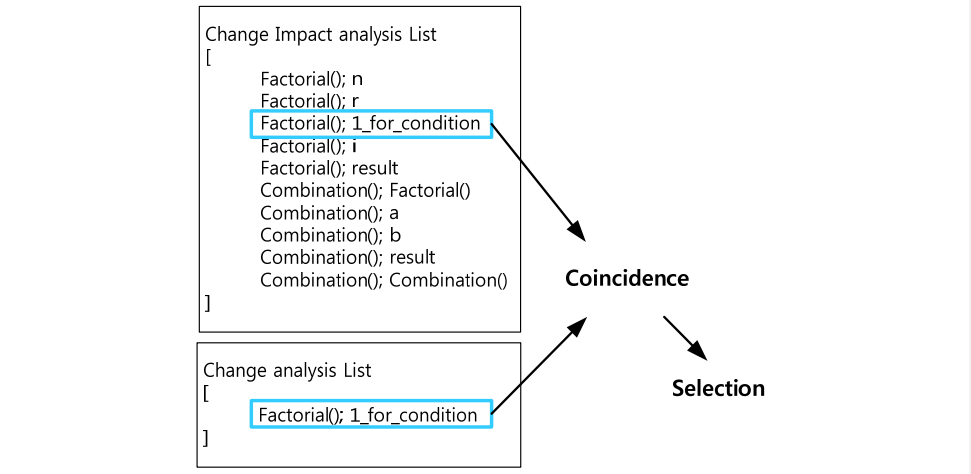


그림 8을 보면 변경 영향 분석 리스트와 변경 분석 리스트에 "Calc(); 1_for_condition" 라는 요소가 동시에 나타나는 것을 확인할 수 있다. 상호간에 동일한 요소가 존재한다는 것을 다음과 같이 말할 수 있다. "해당 테스트 케이스는 변경이 이루어진 부분을 실행하는 테스트 케이스(변경 분석 리스트에 해당 요소가 존재)이고, 소프트웨어의 변경이 테스트 케이스의 결과에 영향을 미치기 때문에(변경 영향 분석 리스트에 해당 요소가 존재)는 회기 테스팅을 통해 다시 테스트해야 되는 테스트 케이스이다".

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 선택하고, 추가적으로 소프트웨어의 명세 또는 테스트 케이스의 요구사항을 확인해 볼 필요가 있는 테스트 케이스로 분류 한다.

Guideline 16. “Obsolete”와 “NewSpec”가 동시에 있을 경우 “Obsolete”의 우선순위가 높다.

Ⅲ. 케이스 스터디

우리는 RT-Selection을 이용하여 본교 대학의 Introduction to Software Engineering 수업을 통해 개발된 Digital Watch System(DWS) 프로그램을 대상으로 케이스 스터디를 수행 하였다.

3.1 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 콘솔 출력으로 대체, 백라이트는 출력되는 문자의 색을 바꾸는 것으로 대체, 알람은 컴퓨터의 비프음으로 대체하여 구현된 프로그램이다.

그림 9_DWS 시스템 예제

3.2 RT-Selection 결과

우리는 총 9팀에 대해 RT-Selection을 적용하였고, 표 2를 통해 결과를 확인 할 수 있

다. 표 2의 첫 번째 행은 변경이 생기기 전 수행하였던 기존 테스트 케이스의 수이고, 두 번째 행은 "Obsolete"태그가 나타나는 테스트 케이스로써 더 이상 새로운 소프트웨어에 적용 할 수 없는 테스트 케이스이다. 세 번째 행은 "NewSpec"태그가 나타나는 테스트 케이스로써 회귀 테스팅을 위해 선택해야 하는 테스트 케이스이고 특별히 주의를 기울여야 하는 테스트 케이스 이다. 네 번째 행은 회귀 테스팅 시 반드시 다시 수행해야 하는 테스트 케이스 이다. 마지막 행은 NewSpec 테스트 케이스와 Re-testable 테스트 케이스 합친 테스트 케이스의 수로써 RT-Selection이 찾으려 하는 회귀 테스팅을 위한 테스트 케이스이다.

표 2_RT-Selection을 이용한 케이스 스터디 결과

구분	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

결과를 자세히 살펴보면, T4의 경우 변경이 이루어지기 전 소프트웨어를 위해 작성된 테스트 케이스의 수는 총 72개였고, RT-Selection을 적용한 결과 72개중 18개의 테스트 케이스에서 "obsolete"라는 태그가 삽입되어 나타난 것을 볼 수 있다. 이는 회귀 테스팅을 위해 선택할 필요가 없는 테스트 케이스로 분류 할 수 있다. 또한, 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 선정 할 수 있다. 나머지 52개의 테스트 케이스는 변경된 부분을 실행하지 않거나 변경이 테스트 케이스의 결과에 영향을 주지 않았다는 것을 의미하기 때문에 다시 수행할 필요가 없는 테스트 케이스로 분류 할 수 있다.

T1과 T6는 "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 할 필요가 있다는 것을 의미한다.

T2의 경우, 회귀 테스팅 시 필요 없는 테스트 케이스나 다시 테스트해야 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 하

지 않다는 것을 의미한다. 하지만 만약 T2가 만든 프로그램에도 변경이 생겼다면, 기존 작성된 테스트 케이스가 T2의 소프트웨어를 테스팅하기에 충분하지 않았다는 것을 간접적으로 알 수 있다. 무엇보다 먼저 소프트웨어를 충분히 테스트할 수 있는 테스트 케이스를 만들어야 의미 있는 회귀 테스팅도 수행 가능하다는 것을 확인 할 수 있다.

Ⅳ. 결론 및 향후 연구

회귀 테스팅이란 소프트웨어의 변경이 새로운 버그나 에러를 만들었는지 확인하는 활동이다. 하지만 소프트웨어의 규모와 복잡성이 점차 커지면서 회귀 테스팅의 비용 역시 커지고 있다. 이를 해결하기 위해 본 원고는 텍스트 차이점과 변경 영향 분석 방법을 사용하여 비용 절감적으로 회귀 테스팅을 수행 할 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 보였다. 우리는 RT-Selection을 이용해 케이스 스터디를 수행하였고 적용 가능성을 보였다. 현재 우리는 자동화 도구를 구현 계획 중에 있으며, 자동화 도구의 개발이 완료 된다면, RT-Selection의 효율성이 보다 크게 증가 할 것이라 기대하고 있다.

* 본 원고는 해당 논문의 요약본임에 따라 자세한 내용은 논문원문을 참조하시기 바랍니다.

참고 자료

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. Wikipedia, Rule of inference, [Online]. Available: http://en.wikipedia.org/wiki/ Rule_of_inference
6. Wikipedia, Diff, [Online]. Available: http://en.wikipedia.org/wiki/Diff