RT-Selection : A Regression Test Selection Technique Using Textual Differencing and Change Impact Analysis

Eui-Sub Kim[†1] Dong-Ah Lee[†2] Junbeom Yoo[†3]

Abstract: Regression testing intended to provide confidence that newly introduced changes do not obstruct the behaviours of the existing and unchanged parts of the software. One of simple and basic regression testing techniques is retest-all, but it requires lot of time and cost. The regression test selection technique selects a subset of previous test cases to retest the changed software. It, therefore, can reduce the time and cost through reducing the number of test cases for regression testing. This paper proposes a new regression test selection technique, RT-Selection, which can perform the regression test more effectively than the existing selection techniques. It can be explained in two approaches. First, it uses textual differencing to fine change. Second, it uses change impact analysis to fine the software riffle to trace test cases. RT-Selection helps testers efficiently select a subset from previous test cases for regression testing. We also propose 4 Guidelines and inference rules to support this technique. Guidelines and inference rules help testers can perform this technique more systemically. We performed a case study to show feasibility of the proposed technique with graduate and undergraduate software engineering classes in Konkuk University.

Keywords: Regression testing, Regression test selection, Textual differencing, Change impact analysis

1. Introduction

Regression testing is performed when changes are made to existing software. The purpose of regression testing is to provide confidence that the newly introduced changes do not obstruct the behaviours of the existing, unchanged part of the software [1]. One of simple and basic regression testing techniques is retest-all. The retest-all performs all test cases again. However, the cost of testing has been exponentially rising on account of the complexity and size of the modern software. In order to reduce the cost, various strategic regression testing techniques have been proposed in the past. One of the widely known techniques is regression test selection which can reduce the time and cost (see [2][3][4]).

The regression test selection technique selects a subset of previous test cases to retest the changed software only. By reducing the number of test cases, it is able to perform cost effective regression testing. [5] defined the regression test selection problem as follows: Given a program P, a modified version P', and a test set T used previously to test P, regression testing techniques attempt to make use of T to gain sufficient confidence in the correctness of P'. In order to accomplish the sufficient confidence of the correctness between of P and P', we found two necessary activities such as (1) identification of modifications and (2) selection of subsets of test cases.

This paper proposes a new regression test selection technique, RT-Selection, which reflects the two necessary activates. First, it uses the textual differencing to identify the modifications. The textual differencing is a result from the comparison between statements of old and new versions of code. Textual differencing results help testers identify changed point and elements affected by previous changes.

Second, it uses a change impact analysis approach to select a subset of test cases. [6] defined that change impact analysis is determination of potential effects to a subject system resulting from a proposed software change. To understand the software with respect to the change, a tester must ascertain parts of the software that will be affected by the change and examine them for possible further impacts [7]. The RT-Selection proposes to use a footprint that will be inserted to the old code in order to identify the changed parts of the software (riffle). If one of the test cases is performed with the old code inserted footprints, a footprint list would be produced. The tester, then, can be aware of the trace of test cases by the footprints list.

With the support of the two techniques, the RT-Selection can help testers select a subset of test cases efficiently for regression testing. We performed a case study to show feasibility of the RT-Selection with graduate and undergraduate software engineering classes in Konkuk University. We obtain 18 obsolete test cases, 2 re-testable test cases which mean that those are test cases selected by RT-Selection and 52 not necessary to retest test cases from 72 whole test cases. RT-Selection produces the 3 type test cases. Those type help testers can classify test cases for regression testing.

This paper is organized as follows. Section 2 provides backgrounds such as regression testing and test cases selection techniques. In section 3, we introduce the detail process proposed technique. Section 4 reports the result of the case study, and Section 5 concludes the paper and presents our future work

2. Backgrounds

A regression test selection technique chooses, from an existing test set, tests that are deemed necessary to validate modified software [4]. The RT-Selection is based on this technique. Many selective regression testing techniques have been introduced. If you want to see this techniques, you could refer to the literature [2][3][4].

Pythia [8] developed by F. I. Vokolos and P. G. Frankl is a regression testing tool, which realize the regression test selection technique. It selects some test cases from the whole test cases, based on textual deference.

TestTube[9] developed by Chen, Rosenblum, and Vo is a tool, which combines static and dynamic analysis to perform

† Konkuk University, 120 Neungdong-ro, Gwangjin-gu, Seoul 143-701, Korea
a) atang34@konkuk.ac.kr
b) ldalove@konkuk.ac.kr
c) jbyoo@konkuk.ac.kr

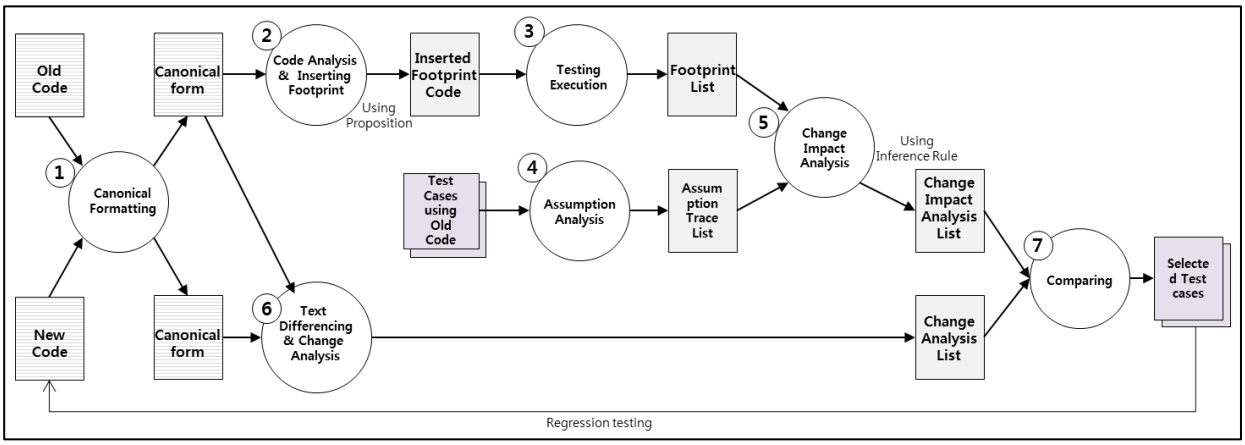

Figure 1 An overview of the RT-Selection technique

selective retesting of software written in C. It identifies which functions, types, variables and macros are covered by each test unit in a test suite and which entities were changed to create the new version. Using the coverage and change information, TestTube selects the test cases for retesting.

3. The RT-Selection Technique

This section presents an overview of the RT-Selection technique as described in Figure 1. It consists of 7 phases as follows:

3.1 (Phase 1) Canonical Formatting

Canonical formatting is task that each different style forms convert into canonical form. Both of code (old and new) may not be same even if a function of the software were equivalent since developers have own coding styles individually. Textual differencing, however, compares old and new code in terms of text one by one (ex, blank lines, comment lines, and different coding styles so on.). Thus, it is necessary for testes to convert into canonical form in order to obtain correct results.

(A) and (B) in Figure 2 show two source codes in different styles, but both have the same behavior. Furthermore, they are exactly same after conversion into the canonical form. To convert canonical form, any method, any form or any style is not restriction. Only necessary task is to make canonical form with the same style

```
int init(int a,int b)
{
        if(a<b) return b-a;
        else return 0;
}
int Calc(int a,int b){
        int i;

        int sum=init(a,b);
        for(i=a;i<b;i++)
                sum=sum+1;

        return sum;
}
```

(A) A source file in style A

```
int init(int a, int b){
        if(a<b)
                return b-a;
        else
                return 0;
}
int Calc(int a, int b){
        int i;

        int sum=init(a, b);
        for(i=a; i<b; i++)
                sum = sum+1;
        return sum;
}
```

(B) A source file in style B

```
int init(int a, int b) {
        if (a < b)
                return b - a;
        else
                return 0;
}
int Calc(int a, int b) {
        int i;

        int sum = init(a, b);
        for (i = a; i < b; i++)
                sum = sum + 1;
        return sum;
}
```

(C) A source file in the canonical form

Figure 2 An example of the old, new and canonically formatted codes

3.2 (Phase 2) Code Analysis and Inserting Footprint

Code analysis is task to understand the meaning about sentence of code. The purpose of code analysis is to identify both affected and affecting elements. And, inserting footprint is task to insert appropriate footprints that will notify testers about the interrelation of elements. Elements, testers should focus on, are follows: assignment, function return, function call, condition statement and iteration statement. For example, given sentence is "a = b + 1", it is element to insert the footprint since it is assignment. In the case, the footprint, "a b" (it is meaning that the variable "a" is effected by variable "b" or the variable "b" is effecting the variable "a"), should be inserted above the sentence

```
int init(int a, int b) {

        printf("init(); 1_if; 1_if_condition <- a\n");
        printf("init(); 1_if; 1_if_condition <- b\n");
        if (a < b) {
                printf("init(); 1_if; init() <- a\n");
                printf("init(); 1_if; init() <- b\n");
                printf("init(); 1_if; init() <- 1_if_condition\n");
                return b - a;
        } else {
                printf("init(); 1_else; init() <- 1_if_condition\n");
                return 0;
        }
}
int Calc(int a, int b) {
        int i;

        printf("Calc(); init() <- a\n");
        printf("Calc(); init() <- b\n");
        printf("Calc(); sum <- init()\n");
        int sum = init(a, b);

        printf("Calc(); 1_for; 1_for_condition <- a\n");
        printf("Calc(); 1_for; 1_for_condition <- b\n");
        for (i = a; i < b; i++) {
                printf("Calc(); 1_for; sum <- 1_for_condition\n");
                sum = sum + 1;
        }

        printf("Calc(); Calc() <- sum\n");
        return sum;
}
```

Figure 3 The old codes with inserted footprints

Figure 3 shows an example of the footprint insertion. It inserted footprints into the canonically formatted code in Figure 2. The insertion, however, takes much time and cost as described in Figure 3. The RT-Selection proposes 4 guidelines, which can help testers perform this phase more systematically, as follows:

Guideline 1. If the sentence includes assignment, function return and function call, the necessary footprint is as follows:

- Assignment: "*Left element = Right element*"
  - Footprint: "*Left element ← Right element*"
- Function return: "*return something*"
  - Footprint: "*function name () ← something*"

- Function call
  - Footprint: "*function name ()* ← *parameter*"

Guideline 2. If the sentence includes a condition statement (ex., If the sentence includes a condition statement (ex., if, switch, etc.), the necessary footprint is as follows:

- Condition statement:
  - Footprint: "(*ordinal number)_(context)_condition* ← *used element in condition statement*"

The ordinal number is a sequence number used in the function. The context is identification of the statement (ex., if, switch, so on). In addition, if one of the elements in Guideline 1 is included at the function of condition statement, the necessary footprint is as follows:

- Element of Guideline 1 in the condition statement
  - Footprint: "*Left element* ← (*ordinal number)_(id)_ condition*"

Guideline 3. If the sentence includes an iteration statement (ex., for, while, etc.), the necessary footprint is as follows:

- Iteration statement:
  - Footprint: "(*ordinal number)_(id)_condition* ← *used element in condition of iteration statement*"

And, the other things refer the 'Guideline 2.'

Guideline 4. The expression of footprint with Guidelines 1-3 can be enumerated as follows.

- Enumeration:
  - Footprint: "*Left element* ← *(ordinal number)_(id)_ condition_(ordinal number)_(id)_condition; ....*"

3.3 (Phase 3) Testing Execution

Testing execution is task to perform the unit testing with existing test cases to the old code. When testing is done, testers can obtain footprint list such as (A) of Figure 4 which is trace of test case in the software components. But, it may have many duplicated elements. Thus, if testers want the compact size, testers can eliminate duplicated footprint (B) of Figure 4.

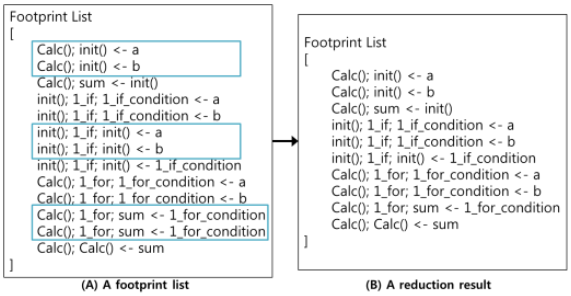

Figure 4 An example of footprint list and reduced result

3.4 (Phase 4) Assumption Analysis

Assumption analysis is task to identify what elements will affect to the assumption of unit test case. For unit testing, testers can use more than one element such as unit, global or instance elements. However, all of elements do not affect the assumption because some of them are merely used for initialization or setting so on. Thus, testers should identify elements affecting the assumption exactly.

Figure 5 depicts the test case of unit testing (A) and assumption analysis list (B). In this case, the assumption is "expected_value == 3." Using this assumption, testers have to obtain the result as "expected_value ← Calc()" because 'expected_value' affects to the assumption, and 'Calc()' affects the 'expected_value.'

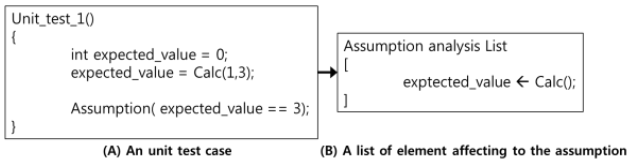

Figure 5 An example of an unit test case and an assumption analysis list

3.5 (Phase 5) Change Impact Analysis

Change impact analysis (CIA) is task to identify exact elements that affect to one of assumption analysis list. The footprint list is merely collection all of elements executed by test case. However, those all does not affects to the elements in the assumption list. In older words, if the element did not affect to the assumption, the element would not important element even if it was affected by change. Therefore, it is necessary to fine actual affecting elements.

(A) of Figure 6 depicts inference process. The expected value of the bottom line is a start point that is one of elements in the assumption analysis list. And the next is to identify associated elements step by step. The collection of associated elements is change impact analysis list. (B) of Figure 6 depicts the change impact analysis list.

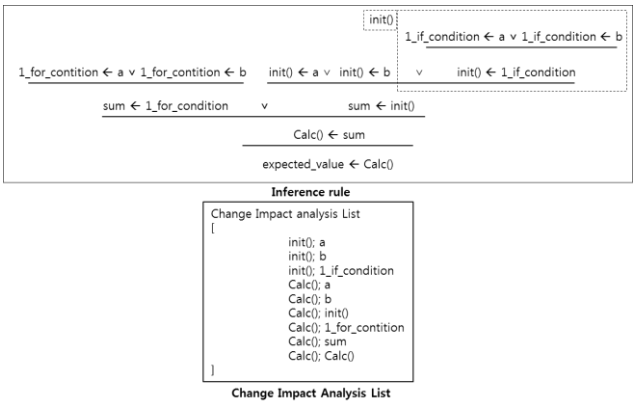

Figure 6 An inference rule and corresponding change impact analysis list

3.6 (Phase 6) Textual Differencing and Change Analysis

The textual differencing is a result from the comparison between statements of old and new versions of a source code. Textual differencing compares old and new code in terms of text one by one. Change analysis is task to analyze what elements is affected by change using result of textual differencing.

(A) and (B) of Figure 7 depict the change detection by textual differencing. (C) of Figure 7 depicts the result of interpretation about sentence of change point. In addition, if sentence include change of the type of variable or parameter of function, testers could insert tag "obsolete" to the result. The tag helps testers select test cases more easily.

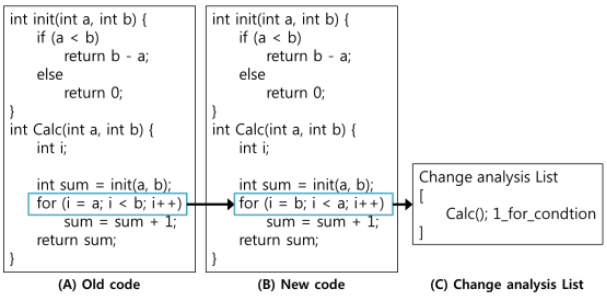

Figure 7 An example of change analysis list

3.7 (Phase 7) Comparing

Comparing is task to identify coincided elements between impact analysis list and change analysis list. If elements of both results coincided, testers could select the test case for regression testing. Figure 8 depicts an example of coincidence. In addition, if change analysis list have the tag "obsolete," tester could ignore this test case since this test case is useless any more for the new software.

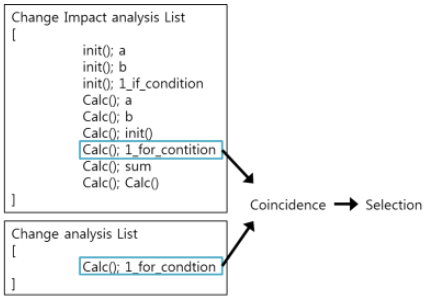

Figure 8 A comparing result

4. Case Study

We, a test team, have applied the RT-Selection to the software developed team in the undergraduate class. The development team released the softwares reflecting test results. A purpose of the undergraduate class is development of a digital watch system (DWS). The development teams reflected results of the testing by testing team to fix faults. We have performed the RT-Selection to DWS developed by development team.

The result is that the total number of test cases for DWS is 72. After applying the RT-Selection, 18 test cases have a tag "Obsolete," which mean that they do not need to retest because the format have changed or have no target units in the new version. We obtain 2 test cases that have coincident elements in both change impact analysis list and change impact analysis list. It means that those test cases executed affected element and the element affects the assumption of test case. Thus, those test cases must retest to provide confidence between old and new software. The other test cases are not need to retest, because they are not executing the affected element or affected element are not affecting the assumption. In conclusion, we could able to obtain 18 obsolete test cases, 2 re-testable test cases and 52 test cases that is not necessary to retest.

5. Conclusion and Future work

Regression testing is one of test activities to check whether changes of software make new bugs and errors. But, regression testing has problem in terms of cost. To solve this problem, we propose a regression test selection technique, RT-Selection, for reducing regression testing cost; it uses the text differencing and change impact analysis. This technique has 7 phases and some of Guidelines and inference rules to support the systemic procedure. We performed case study with DWS software. We had classified the existing test cases to the 3 types. It is useful to perform a regression testing. We are now planning to implement a set of automation tools for the RT-Selection. It would increase usability of the RT-Selection, and be more helpful for regression testers.

References

1) S. Yoo, M. Harman: Regression testing minimization, selection and prioritization: a survey, Software Testing, Verification and Reliability, Vol.22, No.2, pp.67-120 (2012).
2) J. Bible, G. Rothermel, D. S. Rosenblum: A comparative study of coarse-and fine- grained safe regression test- selection techniques, ACM Transactions on Software Engineering and Methodology (TOSEM), Vol.10, No.2, pp.149-183 (2001).
3) S. Biswas, R. Mall, M. Satpathy, S. Sukumaran: Regression test selection techniques: A survey, Informatica: An International Journal of Computing and Informatics, Vol.35, No.5, pp.289-321 (2011).
4) G. Rothermel, M. J. Harrold: A safe, efficient regression test selection technique, ACM Transactions on Software Engineering and Methodology (TOSEM), Vol.6, No.2, pp.173-210 (1997).
5) G. Rothermel, M. J. Harrold: Selecting tests and identifying test coverage requirements for modified software, Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis, pp.169-184, ACM (1994)
6) R. S. Arnold, S. A. Bohner: Impact analysis-towards a framework for comparison, Software Maintenance, 1993. CSM-93, Proceedings, Conference on, pp.292-301, IEEE (1993).
7) S. A. Bohner: Extending software change impact analysis into cots components, Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE, pp.175-182, IEEE (2002).
8) FI. Vokolos, PG. Frankl: Pythia: a regression test selection tool based on textual differencing, Reliability, quality and safety of software-intensive systems, pp.3-21, Springer (1997)
9) Y-F. Chen, D.S. Rosenblum, K-P. Vo: TestTube: A System for Selective Regression Testing, Proc. 16th Int. Conf. on Software Engineering, pp. 211-220 (1994)

Acknowledgments This research was supported by a grant from the Korea Ministry of Strategy, under the development of the integrated framework of I&C conformity assessment, sustainable monitoring, and emergency response for nuclear facilities.