Comparison and Analysis of Three Regression Testing Methods

MengqiuQin[a] andHainiCai[b]

Department of Software Engineering, Chongqing University, Chongqing,400000, China

[a]qinmengqiu@cqu.edu.cn, [b]hainicai@cqu.edu.cn

Keywords:RTS, code-based regression test, model-based regression test, UML

Abstract. Regression Test Selection (RTS) techniques have been proposed to increase the effectiveness and reduce the cost of regression testing. Researchers stated numerous types of approaches for selecting regressing test cases, and those methods can be grouped into two categories, code-based and model-based. This paper presents one code-based and two model-based regression test strategies, and also evaluates and analyzes the features, advantages and drawbacks of each technique.

Introduction

Regression testing is expensive and time consuming, but of vital importance in ensuring the quality of continually changing software systems. Regression testing has applied in modified version of software to maintain that the changed version behaves as intended, and examines whether the new part has some negative effects on its original behaviors [1].

In order to keep the effectiveness and reduce the cost of regression test, RTS (Regression Test Selection) has been proposed. RTS techniques select a subset out of the original test suite to retest; hence, we can just focus on the aspects which have been modified, instead of rerunning all the test cases.However, the RTS approaches are not always effective in reducing the cost of Regression testing. Firstly, RTS may be incorrect. It is possible to omit important test cases and select unnecessary test cases by using some inefficient methods. What’s more, RTS may be unprofitable. Obviously, the process, selecting test cases, itself will costs some time and resource; so if the cost of RTS is more than the cost of rerunning the whole test suite, the approach is definitely valueless. Hence, it is quite necessary to evaluate the correctness and the profitableness of a RTS approach to ensure it is effective to reduce the cost and time consumed.

This paper presents three different techniques for regression testing based on Control-flow graphs, UML model and activity diagrams. The first one belongs to code-based RTS techniques, and the other two methods are kinds of model-based. According to these three approaches, this paper compared the main ideas of each in order to find the advantages and the disadvantages of code-based and model based strategies.

The rest of this paper is organized as follow: section 2 describes the technique background and evaluates each approach as well. Section 3 provides the comparison and analysis in order to show the similarity and difference of these three techniques, and we can also see the advantages and drawbacks of each of them in this part. And then, section 4 shows some related works in regression testing fields. Finally, conclusions are drawn in section 5.

Background and Approach Evaluation

Generally, the method of regression test selection could be divided into two aspects, code-based and model-based [2]. Code-based approaches select test cases based on different versions of code. By comparing the original and the modified program, the changed part of code can be located and test suite of retesting can be generated. While, as is implied by the name, model-based RTS techniques are based on the model generations. Modified model elements can be find after comparing different model versions of programs, and the test cases which have relationships with the changed components are those cases need to be retest.

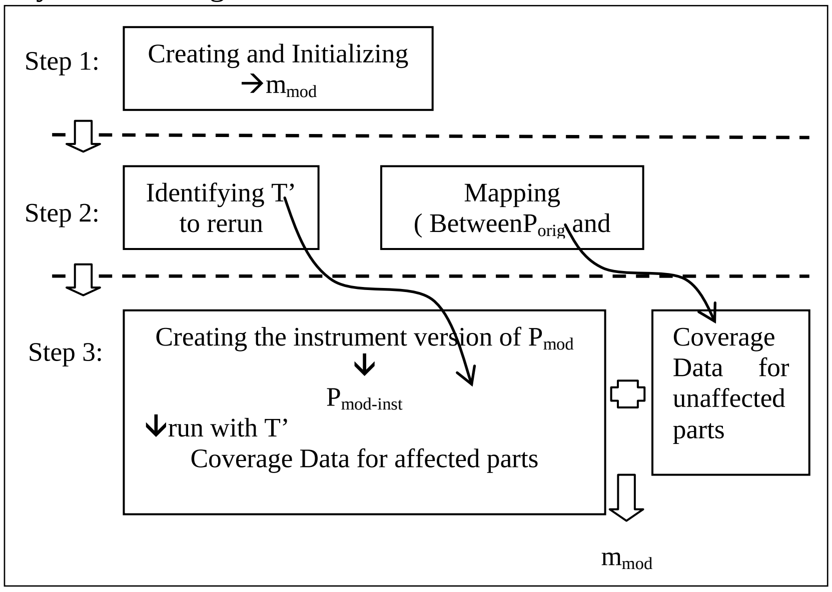
In this section, I will illustrate one code-based and two model based RTS approaches for analysis of the contributions and drawbacks for each method.

Control-flow graph based test selection approach (Code based).

Pavan Kumar Chittimalli and Mary Jean Harrold presented a RTS algorithm based on Control-flow graph and using re-computing coverage information [1]. This is a code-based approach which can compute an updated coverage data by comparing different versions of programs.

The authors of this paper used coverage data to identify which test cases should be done for modified program versions. Using outdated coverage data is lack of accuracy, because it would easily select unnecessary test cases and omit important test cases; while the updated coverage data is expensive and costly, since it reruns the entire test suite. The purpose of this algorithm is to select a test suite with has the same coverage data with rerunning all test cases.

An efficient technique named DEJAVOO has been used in this algorithm. DEJAVOO not only create Control-flow graphs according to the original and modified programs, but labeled the affected parts nodes or edges in the control flow graphs. As proposed in the paper, the inputs of the algorithm include Porig (original program), Pmod(modified program), T(Set of test cases ran on Porig), m(coverage matrix for Porig), and the output of the algorithm is mmod. The main steps of this algorithm are shown in the fig.1: First, a coverage matrix for original program has been created and initialized, which has been illustrated as mmod. The second step is to identify the T' to rerun from T by using the method DEJAVOO, and to establish a mapping which stores the relationships between Porig and Pmod. Finally the third step is creating the Pmod-inst which represents the instrument version of Pmod. After running Pmod-inst with T' identified in step2, the coverage data for affected part has been done. And as well, the coverage data for unaffected part can be gotten through the mapping between original and modified versions. So, by combining the two parts together, the whole coverage data matrix for Porig has been determined, and the coverage data matrix can be directly used into regression test selection.



Step 1:
Creating and Initializing
→m_{mod}
Step 2:
Identifying T' to rerun
Mapping (BetweenP_{orig} and
Step 3:
Creating the instrument version of P_{mod}
$P_{mod\text{-}inst}$
run with T'
Coverage Data for affected parts
Coverage Data for unaffected parts
m_{mod}

Fig.1 Main steps of algorithm using re-computing coverage information

UML model based test selection approach. Unified Modeling Language (UML) is widely used in regression test selection model-based methods. As described by L. C. Briand, Y. Labiche and G. Soccar [4], the UML based RTS method is more close to the design or architecture information. They presented an approach use three kinds of UML diagrams including class diagrams， sequence diagrams and use case diagrams. By using this approach, test cases can be generated automatically. Additionally, they also proposed a tool named RTSTool to apply the method.

In order to support this RTS technique, test cases has been defined as three different categories: reusable, retestable and obsolete test cases. Reusable test cases are those test cases do not need to be rerun. While, retestable test cases need to be rerun to keep the safety and accuracy of regression testing. And the obsolete test cases are those that need to be remove from the test suite, because the class or sequence they related to can no longer be execute in the new version of program.

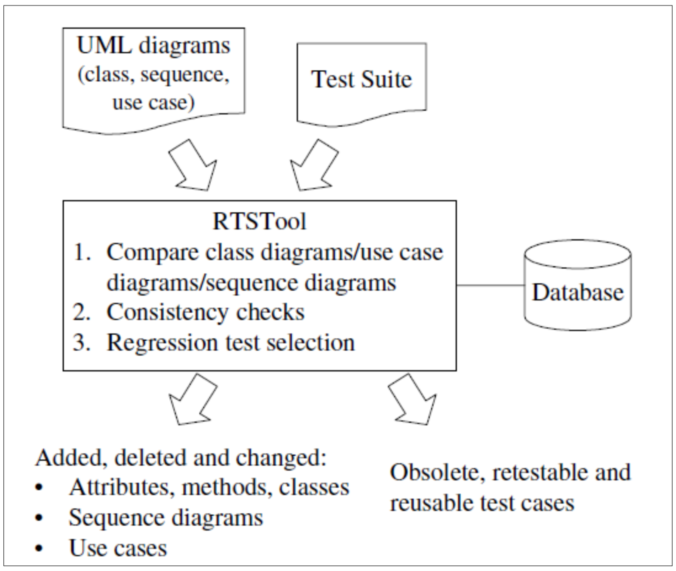

UML diagrams (class, sequence, use case)
Test Suite
RTSTool
1. Compare class diagrams/use case diagrams/sequence diagrams
2. Consistency checks
3. Regression test selection
Database
Added, deleted and changed:
• Attributes, methods, classes
• Sequence diagrams
• Use cases
Obsolete, retestable and reusable test cases

Fig.2 Main steps of UML model based testing selection approach [4]

As illustrated in fig.2, there are three main steps of this RTS approach. Firstly, the tool RTSTool needs to establish the mapping between the sequence diagram and test cases. Secondly, the attributes, methods and relationships between different classes should be checked, and all of the elements will be labeled with added, deleted and changed tag. Similarly, the changed methods and use cases will be found by comparing the sequence or use case diagrams of different versions. Then, according to the results of the second step, it is easy to classify test cases into three categories: obsolete, retestable and reusable test cases.

We can see from the case study and result analysis in the survey that this method is quite successful and has a high performance in automation and efficiency. Because the instance they used is a case on system level, and any test case in that level would generally related to the sequence of methods [4].

Activity diagram based test selection approach. Nan Ye, Xin Chen and their team members presented a RTS technique based on activity diagram. According to their method, a technique named C-FT (feedback-directed test cases generation) [5] has been embedded into the activity diagram based approach to get a regression test suite automatically.

Similarly with the UML based method mentioned in part 2.2, they grouped the test cases into affected, unaffected, removed and new test cases. And the results can be used to decide the test suite.

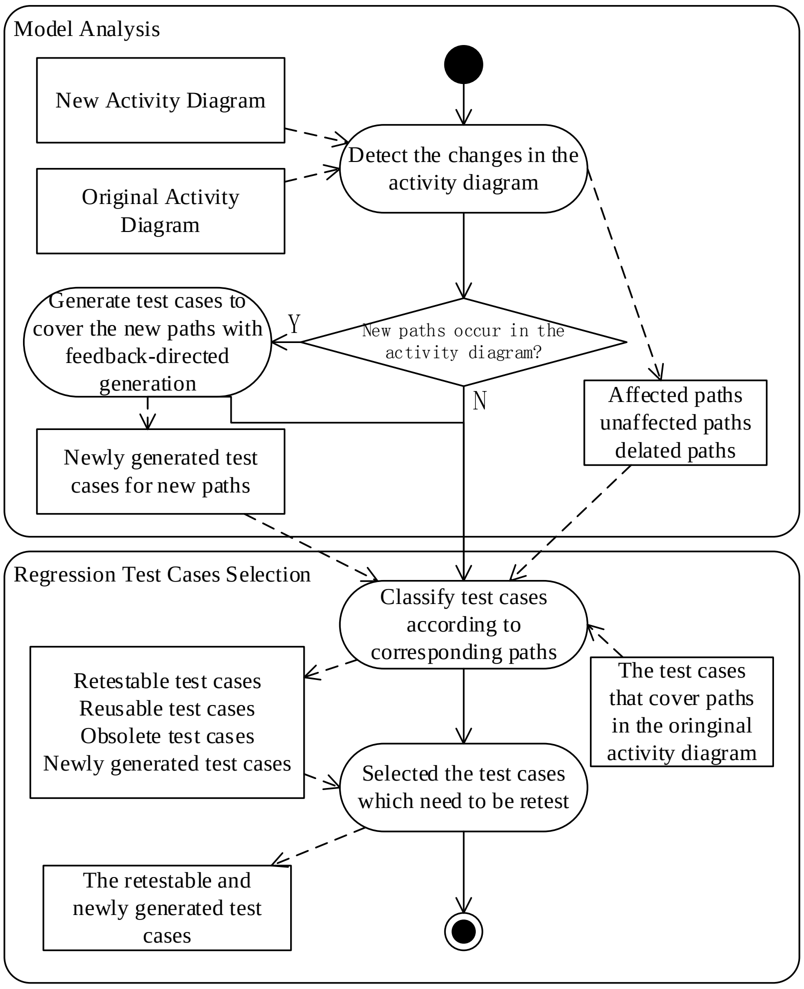

Model Analysis
New Activity Diagram
Original Activity Diagram
Detect the changes in the activity diagram
Generate test cases to cover the new paths with feedback-directed generation
Y
New paths occur in the activity diagram?
N
Affected paths unaffected paths delated paths
Newly generated test cases for new paths
Regression Test Cases Selection
Classify test cases according to corresponding paths
Retestable test cases Reusable test cases Obsolete test cases Newly generated test cases
The test cases that cover paths in the oringinal activity diagram
Selected the test cases which need to be retest
The retestable and newly generated test cases

Fig.3 The workflow of automatic regression test selection based on activity diagrams [5]

We can see from the fig.3 that the approach consists of two main parts：model analysis and regression test case selection. In model analysis part, they identified the paths in diagram as affected, unaffected, removed and new path, by comparing the original activity diagram and the modified diagram. And in the part of regression test case selection, test cases could be classified into different groups according to the corresponding paths. We can only rerun the affected and newly generated test case on new version of program instead of retest the entire test suite of the old program.

With this method, the cost will be reduced efficiently, especially for some slightly updating between program versions the benefits would be quite significant. Besides, the accuracy would be higher than rerunning the whole old test suite, because they considered the test cases generated from new added paths.

Comparison and analysis

Sinilarity

According to the three approaches mentioned in section 2, no matter it is code-based or model-based, their main ideas are pretty much the same. The common mode of these approaches is identifying the changed part and unchanged part separately by comparing different versions on code or model. And then they generate the new test cases depend on the changed part, while as for the test cases related to unchanged part, they just need to keep the same with the original.

Additionally, the ways for test case measurement used in these approaches are similar. The first approach use coverage data to measure test case, and the those coverage data can be classified by DEJAVOO[1] to decide which part remains the same with the original and which part contains danger edges. The UML-based approach divided the test cases into three categories: reusable, retestable and obsolete. Similarly, the third approach use affected, unaffected, removed and newly generated test cases to group different part of output test suite.

Differences.

Even though these three approaches are similar with each other in some aspects, the differences of them need to be considered. After comparing the other two methods, each of the three approaches have its own features, advantages and drawbacks as follow:

Control-flow graph (Code) based. This code-based method focuses on analyzing control flow and data flow of source program. Pretty similar to white-box testing, this approach is more direct, detailed and specific than the model based RTS methods. It is quite beneficial that the output test cases associate directly to code or control flow, and it is easily accessible and of highly accuracy. However, with the increasing size and complexity of software system, code-based technologies are losing their competitiveness on efficiency.

UML model based. This approach depends on the relationships between model elements and test cases, and test suite is generated according to the model architecture. As a black-box testing, the advantage of this method is more efficient and effective, because it is closer to the architecture and design. When it comes to the larger size system, the benefits could be more magnificent. Nevertheless, the test cases determined in this UML-based technique could not be directly used without some extra manual works. In this case, by using the three kinds of UML graphs, we can have a global analysis for the system architecture, and as shown in the section of case study [4], the result of this approach is quite accurate. However, if we just use one kind of UML diagrams, maybe some relationships between two program versions would be omitted. What's more, most UML-based strategies are suitable for object oriented programs or well organized systems, so it may be useless in other cases.

Activity diagram based. The activity diagram based approach is a kind of model-based strategies, because activity diagram is one of the common forms of UML diagram. But owing to the specialty of activity graphs, it is closer to the side of code and details. As we can see from the case presented by Nan Ye, Xin Chen and their team members [5], the approach associates to the executions and calls inside functions. So it contains features in both code-based and model-based strategies. This activity diagram based RTS approach is like gray-box testing in some aspects. By combining the characteristics and advantages of both model-based and code-based techniques, the activity diagram based approach generates the regression test suite according to the relationships between model elements and test cases, and the output test suite can be used directly into regression testing.

Related works

There are large amount of researches related to regression testing. Some of test case generations are based on model design [7][8][9]; while some others are associate to original source code[10][11]. Most of these model-based technologies generated their test suite from the designs and architectures, without using the source program. They required extra works of the transformation from abstract test cases into these cases can directly used. And like the control flow based approach mentioned in this paper, the code-based techniques are more concrete, but has the limitation in large size and complex program.

Conclusions

With the development of information industry, software systems have become larger, more complex and continually updatable than ever before. The increasing size and complexity cause the problem of highly cost and time consuming in regression testing. Both cede-based and model-based strategies have been commonly used in RTS. Code-based approaches select test cases based on the different versions of code. The changed regions of code or control data flows can be located and test suite of retesting would be generated, after comparing the different program versions. While, model-based RTS techniques are based on the model generations. This kind of strategies is closer to the architecture and design, and modified test suites are generated through the relationships between model elements and test cases.

To sum up, the results of this code-based approach can be directly used into regression test, and at the same time, a specific algorithm has been proposed to generate the affected test cases. Compared with code-based approach, the model-based is more efficient but more abstract, and they may require heavy manual efforts to get more specific test suite. The RTS strategy could be chosen according to the size, the categories, and the specific requirements of target programs. For example, if the program is objected-oriented, well-organized and with large size, a UML-based approach should be recommended; while if the target program is of smaller size or the results are required to be used directly, it is better to use an approach closer to code.

References

[1] P. K. Chittimalli, M. J. Harrold. Re-computing Coverage Information to Assist Regression Testing. IEEE International Conference on Date 2-5 Oct. 2007

[2] C. Bharati1, S. Verma. Analysis of Different Regression Testing Approaches. International Journal of Advanced Research in Computer and Communication Engineering Vol. 2, Issue 5, May 2013

[3] A. Orso, N. Shi, and M. J. Harrold. Scaling regression testing to large software systems. In Proceedings of the 12th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE 2004), pages 241–252, November 2004

[4] L. C. Briand, Y. Labiche, G. Soccar. Automating Impact Analysis and Regression Test SelectionBased on UML Designs. Proceedings of the International Conference on Software Maintenance (ICSM.02), 2002

[5] Nan Ye, Xin Chen, Peng Jiang, Wenxu Ding and Xuandong Li. Automatic Regression Test Selection based on Activity Diagrams. Fifth International Conference on Secure Software Integration and Reliability Improvement Companion, 2011

[6] Md. Imrul Kayes. Test Case Prioritization for Regression Testing Based on Fault Dependency. 2011 IEEE

[7] Y. G. Kim, H. S. Hong, D. H. Bae and S. D. Cha, Test cases generation from UML state diagrams. In IEE Proceedings: Software, vol. 146, no. 4, The Institution of Engineering and Technology(IET), 1999, pp. 187-192

[8]D. Seifert, S. Helke, and T. Santen, Test case generation for UML statecharts. In Proceedings of the 5th International Andrei Ershov Memorial Conference on Perspectives of System Informatics(PSI 2003), Lecture Notes in Computer Science 2890, Springer, 2003, pp. 93-109.

[9] M. Chen, X. Qiu, W. Xu, L. Wang, J. Zhao, and X. Li, UML activity diagram-based automatic test case generation for Java programs. In The Computer Journal, vol. 52, no. 5, Oxford University Press, 2009, pp. 545-556.

[10] Gregg Rothermel, Mary Jean Harrold, and Jeinay Dedhia, Regression Test Selection for C++ Software. Journal of Software Testing, Verification, and Reliability, Vol. 10, No. 2, June2000.

[11] G. Rothermel, M.J Harrold, Selecting Tests and identifying Test Coverage Requirements for Modified Software,In Proceeding of the ACM international Symp. On Software,pp-169-184, August 1994.