Jtop: Managing JUnit Test Cases in Absence of Coverage Information

Lingming Zhang, Ji Zhou, Dan Hao, Lu Zhang*, Hong Mei
Key Laboratory of High Confidence Software Technologies, Ministry of Education
Institute of Software, School of Electronics Engineering and Computer Science, Peking University,
Beijing, 100871, P. R. China
{zhanglm07, zhouji07, haod, zhanglu, meih}@sei.pku.edu.cn

Abstract

Test case management may make the testing process more efficient and thus accelerate software delivery. With the popularity of using JUnit for testing Java software, researchers have paid attention to techniques to manage JUnit test cases in regression testing of Java software. Typically, most existing test case management tools are based on the coverage information. However, coverage information may need extra efforts to obtain. In this paper, we present an Eclipse IDE plug-in (named Jtop) for managing JUnit test cases in absence of coverage information. Jtop statically analyzes the program under test and its corresponding JUnit test cases to perform the following management tasks: regression test case selection, test suite reduction and test case prioritization. Furthermore, Jtop also enables the programmer to manually manipulate test cases through a graphical user interface.

1 Introduction

Test suite reuse, in the form of regression testing, is prevalent in software construction [5, 8]. Regression testing is a time-consuming task, taking up as much as one-half of the cost in software maintenance [4, 8]. Test case management, including regression test case selection, test suite reduction and test case prioritization, has been intensively studied. However, few practical tools have been used by the industry because most existing management techniques focus on test case management based on coverage information. Techniques based on coverage information may have the following disadvantages. First, before applying these techniques, testers need to run instrumented source code to collect coverage information, and the process may be time-consuming. Second, coverage information can be of large volume for large programs, and thus the storage and management of coverage information may be a burden for developers. Third, changes of source code in software evolution may make source code inconsistent with previous collected coverage information. Finally, developers may modify some existing test cases or add new test cases in regression testing, making previous coverage information inconsistent with test cases in the test suite. These techniques may thus be negatively impacted.

Nowadays, the JUnit testing framework has been widely used for testing Java software. Large programs usually accumulate numerous JUnit test cases during the period of evolution, and developers may need tools to manage JUnit test cases. Different from traditional test case formats, the format of each JUnit test case is a piece of executable source code that contains a sequence of method invocations. Therefore, it is possible to extract the static call graph of each JUnit test case and use this information to guide management tasks. In this paper, we present an Eclipse IDE plug-in (named Jtop), aiming to facilitate the management of JUnit test cases based on static analysis of the program under test and its corresponding JUnit test cases.

2 Approach of Jtop

Shown in Figure 1, Jtop consists of seven modules. "Analysis Engine" is the basis of other modules. It statically analyzes the program under test and JUnit test cases to provide information for high level modules. "GUI Display Support" displays the manipulation process and results of other modules in various GUI display views. "Test Suite Generation"[1] turns the manipulation results of automatic or manual operation into a executable JUnit test suite. Besides, the modules framed by broken lines in Figure 1 are shown in detail as follows.

*Corresponding Author.

[1]Test suite generation here has a different meaning from test case generation. In Jtop, we refer to the process of selecting a subset of test cases from the whole set of test cases and prioritizing the selected test cases as test suite generation.

1527-1366/09 $29.00 © 2009 IEEE
DOI 10.1109/ASE.2009.22

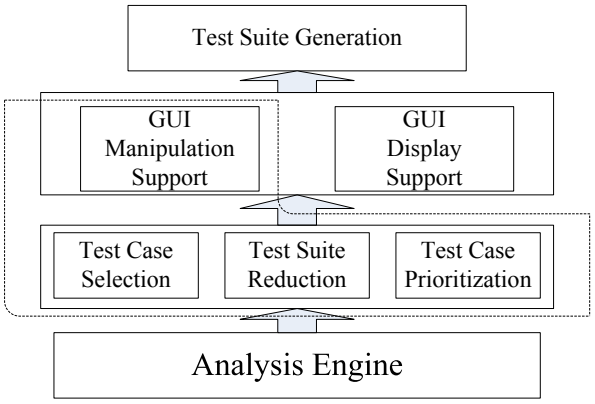

Figure 1. Hierarchy of *Jtop*

2.1 Regression Test Case Selection

Typically, techniques for regression test case selection aim to select a subset of the test suite in order to test software changes. Formally, safe regression test case selection can be defined as finding a subset of test cases $T \subseteq TS$, such that $(\forall t)$ $[t \in M \cap TS \rightarrow t \in T]$. In this definition, TS denotes the original test suite, M denotes the set of test cases influenced by the modifications. In the literature, several techniques have been proposed, including symbolic execution [9] and dependence graph based approaches [6]. However, most existing techniques cannot work without coverage information.

Jtop statically analyzes the structure of the program under test and JUnit test cases to construct the *relevant* relation between them. The $relevant$ relation is defined as follows:

Definition 1 *An element*[2] s *of the program under test is* relevant *to a test case* t*, iff* $\exists$ m *such that* $m \in \varpi(s) \bigwedge m \in \hbar(t)$.

In this definition, $\varpi(s)$ denotes the set of methods that are offsprings of element s in code hierarchy; $\hbar(t)$ is the set of methods directly and indirectly called by test case t in the static call graph. Note that Definition 1 is symmetric. That is to say, Jtop can find *relevant* test cases for certain elements of the program under test, and find *relevant* elements of the program under test for certain test cases.

Using Definition 1 for regression test case selection, Jtop takes the modified elements of the program under test as input and returns all the *relevant* test cases as the test selection results.

[2] An element denotes a unit of source code, which may be a package, a class, a method, and so on.

2.2 Test Suite Reduction

Harrold et al. [2] defined test suite reduction as finding a representative set of test cases $T \subseteq TS$, such that $(\forall r_i)[r_i \in R \rightarrow (\exists t_j)(t_j \in T)(t_j \text{ satisfies } r_i)]$. In this definition, TS denotes the original test suite, and R denotes a set of testing requirements (denoted as r_1,r_2...r_n) that must be obtained to provide the desired testing. However, the problem to achieve the maximum reduction has been shown to be NP-complete [1].

Test suite reduction are first proposed in regression testing where coverage information is available [3]. However, the disadvantages of coverage information based techniques (see Section 1) make it difficult to build practical coverage information based tools. To address this problem, Jtop achieves test suite reduction based on Definition 1. In order to find an approximation to the maximum reduction, we use the hitting set heuristic [2] to guide the test suite reduction. Different from traditional test suite reduction, we used the relevant relation to substitute coverage information. That is to say, in test suite reduction of Jtop, we deem test case t to cover element s iff s is relevant to t.

2.3 Test Case Prioritization

Generally speaking, test case prioritization techniques reorder test cases to maximize some objective function. For example, one such function is the rate of fault detection, indicating how quickly faults are exposed in testing. Rothermel et al. [8] gave the definition for the test case prioritization as finding $T' \in PT$, such that $(\forall T'')[(T'' \in PT)(T'' \neq T') \rightarrow f(T') \geq f(T'')]$. In the definition, PT denotes the set of all possible permutations of a given test suite T, and f denotes a function from PT to real numbers.

Test case prioritization has been intensively studied by both the industry and the academic communities. However, most existing prioritization techniques are based on coverage information and have disadvantages as depicted in Section 1. To address this problem, in our recent work [10], we proposed Jupta, a JUnit test case prioritization approach based on Definition 1. According to our experimental study, although Jupta does not need coverage information, it can perform approximately as effective as widely used coverage based prioritization techniques. Now we have implemented Jupta as a module of Jtop.

2.4 GUI Manipulation Support

In regression testing, programmers may have previous knowledge that some JUnit test cases may have higher priorities than others. To take advantage of this knowledge, Jtop provides a graphical user interface for users to manipulate test cases manually. In Jtop, a user can move or remove

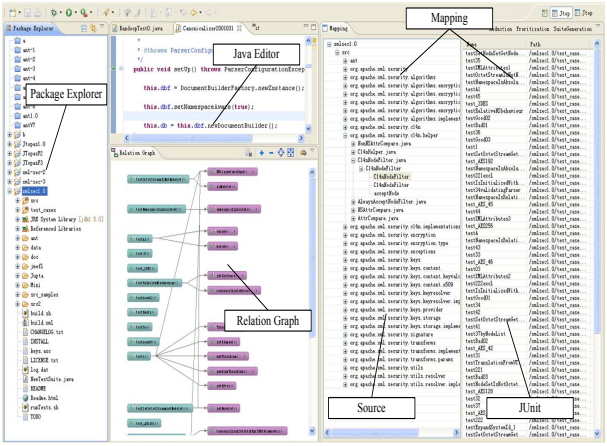

Figure 2. A glance at Jtop perspective

JUnit test cases for any generated test suites to obtain his or her preferred test case permutation.

3 Status

As a visual test case management tool, Jtop has been implemented as an Eclipse IDE plug-in. As shown in Figure 2, Jtop perspective contains "Package Explorer", "Java Editor", "Mapping", and "Relation Graph" four views. "Package Explorer" displays all the programs in workspace. "Java Editor" shows the source code of programs. Consisting of two subviews: "Source" and "JUnit", "Mapping" view provides graphical user interfaces for regression test selection, test suite reduction, test case prioritization, as well as manual manipulation. Based on automatic analysis, "Relation Graph" displays the *relevant* relation between the program under test and JUnit test cases using a colored directed graph.

We are beginning to use Jtop on Java programs from Subject Infrastructure Repository (abbreviated as SIR[3]) to evaluate the effectiveness of our approach. Jtop is now available from our project page[4].

4 Related Work and Conclusion

Although the JUnit testing framework has been widely used in Java software development, few practical JUnit management tools have been proposed. Existing test case management techniques can be categorized into three areas: regression test case selection [6, 7, 9], test suite reduction [2, 3], and test case prioritization [8]. Most of these techniques focus on using previously acquired coverage information.

However, these coverage based techniques may be difficult to use in real-world software development due to the burden of storing and managing coverage information. Different from the existing management techniques, we implement Jtop, an Eclipse IDE plug-in to manage JUnit test cases in absence of coverage information.

[3]SIR is repository providing Java and C programs for controlled experimentation of program analysis and testing approaches. It is accessible from http://sir.unl.edu/.

[4]http://code.google.com/p/pku-jtop/

Acknowledgments

This research is sponsored by the National Basic Research Program of China under Grant No. 2009CB320703, the Science Fund for Creative Research Groups of China under Grant No. 60821003, the National Natural Science Foundation of China under Grant No.90718016 and 60803012.

References

[1] M. Garey, D. Johnson, et al. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman San Francisco, 1979.

[2] M. Harrold, R. Gupta, and M. Soffa. A methodology for controlling the size of a test suite. *ACM TOSEM*, 2(3):270–285, 1993.

[3] J. Hartmann and D. Robson. Approaches to regression testing. In *Proc. of ICSM*, pages 368–372, 1988.

[4] H. Leung and L. White. Insights into regression testing. In *Proc. of ICSM*, pages 60–69, 1989.

[5] A. Onoma, W. Tsai, M. Poonawala, and H. Suganuma. Regression testing in an industrial environment. *Communications of the ACM*, 41(5):81–86, 1998.

[6] G. Rothermel and M. Harrold. A safe, efficient algorithm for regression test selection. In *Proc. of ICSM*, pages 358–367, 1993.

[7] G. Rothermel and M. Harrold. Empirical studies of a safe regression test selection technique. *IEEE TSE*, 24(6):401–419, 1998.

[8] G. Rothermel, R. Untch, C. Chu, and M. Harrold. Test case prioritization: An empirical study. In *Proc. of ICSM*, pages 179–188, 1999.

[9] S. Yau and Z. Kishimoto. A method for revalidating modified programs in the maintenance phase. In *Proc. of COMPSAC*, pages 272–277, 1987.

[10] L. Zhang, J. Zhou, D. Hao, L. Zhang, and H. Mei. Prioritizing JUnit test cases in absence of coverage information. In *Proc. of ICSM (to appear)*, 2009.