FAST ABSTRACT: Static Detection of Redundant Test Cases: An Initial Study

Nuo Li
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
nli3@ncsu.edu

Patrick Francis, Brian Robinson
ABB Corporate Research
Raleigh, NC, USA
{patrick.francis, brian.p.robinson}
@us.abb.com

Abstract

As software systems evolve, the size of their test suites grow due to added functionality and customer-detected defects. Many of these tests may contain redundant elements with previous tests. Existing techniques to minimize test suite size generally require dynamic execution data, but this is sometimes unavailable. We present a static technique that identifies test cases with redundant instruction sequences, allowing them to be merged or eliminated. Initial results at ABB show that 7%-23% of one test suite may be redundant.

1. Introduction

Software systems often change throughout their lifecycle, with added functionality or defect corrections. These changes are often accompanied by additional test cases or modifications of existing ones. Because of this, the test suite may soon grow to be very large and costly to execute. However, some of these test cases may be unnecessary and at least partially redundant.

For example, the tester may use an existing test as a template when creating a new one, and retain parts of it that are unnecessary. The tester may also inadvertently create tests for functionality that is already tested in the existing suite. To cope with this problem, minimization methods exist that seek to find a subset of an existing test suite that is equally effective as the whole.

However, traditional test suite minimization techniques often rely on obtaining code coverage information [1] [2] or other test execution profiles [3] . In some industrial situations, though, it may be difficult or infeasible to do this. For example, the tests may require some specialized hardware or other resource that the test maintainer has only limited access to. Alternatively, the software under test may be highly timing-dependant, such that a coverage tool may disrupt the execution behavior.

In this paper, we present an initial study at ABB on a static method for detecting potentially redundant test cases. This method textually analyzes the instructions within each test case to identify groups that can be merged or eliminated.

2. Approach

At ABB, the software in question is designed to control mechanical systems. This software implements its own programming language, which the user employs to program the system. These programs consist of procedures, grouped into modules, with each procedure consisting of a list of instructions. These instructions may, for example, set a signal or position a device.

A test case (TC) is implemented in a high-level automation framework and consists of a list of commands. These commands perform actions such as copying files, setting variables, loading configurations, and loading and running a program module.

In order to identify test cases that may be redundant, we analyze the commands included in each test case, as well as the instructions in the procedures it loads. These commands/instructions are automatically extracted and any parameters are ignored. The sequences are then compared to each other according to a similarity metric and clustered into groups of mutually similar sequences. The members of a given group are considered to be potentially redundant and subject to a manual inspection. Ideally, each group could be merged into a single test case.

We apply this technique at both the procedure and test case levels. We also combine the two to identify groups of test cases that are similar to each other and also load modules with similar procedures.

To calculate the similarity between two sequences, we consider three different metrics: Manhattan distance (MD), unigram cosine similarity (UCS), and bigram cosine similarity (BCS). The MD is calculated using the counts of the instructions present in the two sequences. Assume two procedures with sets of instructions $S_1=\{x_1, x_2, \ldots, x_n\}$ and $S_2=\{y_1, y_2, \ldots, y_n\}$, where each x_i or y_i is the count of instruction type i present in that procedure. The MD is then calculated as $\sum|x_i-y_i|$ $(i=1...n)$. We consider two sequences similar if their MD is less than 1.

1071-9458/08 $25.00 © 2008 IEEE
DOI 10.1109/ISSRE.2008.50

UCS is based on traditional cosine similarity [4] . This metric calculates the proportion of instructions that two TCs/procedures have in common. Using S_1 and S_2 from above, the UCS is given as $|S_1 \cap S_2| / (|S_1| * |S_2|)^{1/2}$. If two sequences have the same set of instructions, then their UCS will be equal to 1.

UCS focuses on the different kinds of instructions used in TCs/procedures but ignores the order of those instructions. BCS incorporates this ordering by considering pairs of sequential instructions, or "bigrams". It uses the same formula as for UCS, but with S_1 and S_2 composed of bigrams, instead of single instructions. If two sequences are identical then their BCS will be 1.

3. Study Results

In the examined test suite, there are 4218 test cases and 1637 test modules, which contain 6246 procedures. Table 1 shows the results for each metric applied to both the procedure and test case levels. For each metric, it shows the number of similar groups found, the total number of procedures/test cases in those groups, and the potential redundancy. This last value is the percentage of procedures/test cases that may be redundant, assuming that the members of each group can be reduced to a single entity.

Table 1. Redundant groups by metric

		Procedures	Test cases
	Total	6246	4218
MD <= 1	# Groups	814	219
	# Members	4733	3473
	Potential Redundancy	62.7%	77.1%
UCS = 1	# Groups	957	296
	# Members	3937	3606
	Potential Redundancy	47.7%	78.5%
BCS = 1	# Groups	193	135
	# Members	796	1118
	Potential Redundancy	9.7%	23.3%

From these results, we can see that both the MD and UCS metrics identify a large portion of the test suite that may be redundant. The BCS metric detects a much smaller portion of redundant procedures and test cases, only 9.7% and 23.3%, respectively. Since the BCS metric incorporates the ordering of the instructions, this indicates that many procedures/test cases have similar sets of instructions, but in different sequences.

For all three metrics, the potential redundancy is higher among test cases than procedures. Much of this may be due to test cases that have similar action sequences, but that load different modules. To eliminate these false positives, we filtered the test case groups to include only those test cases that are similar to each other and also load modules with similar procedures. These results are shown in Table 2.

Table 2. Test cases with similar actions and modules

	# Groups	# Members	Potential Redundancy
MD <= 1	187	1151	22.9%
UCS = 1	228	1302	25.5%
BCS = 1	66	363	7.0%

After this filtering process, an average of 66.1% fewer test cases are identified as potentially redundant. Eliminating these false positives greatly reduces the amount of manual effort needed to verify whether the test cases are redundant.

4. Conclusions and Future Work

Traditional approaches for test suite minimization usually rely on coverage information collected during dynamic execution. Since this information is unavailable for one of ABB's test suites, we applied a static analysis technique to detect redundant test cases based on their instruction sequences and counts. We used three kinds of metrics to evaluate the similarity among test cases. After filtering the results to include only test cases that are similar to each other and load similar modules, we detect that 7%-23% of test cases are potentially redundant. Manually checking a sample of the detected groups verifies that significant opportunities exist for merging and removing these test cases.

As future work, we shall manually check the detected similar test cases and procedures, and consult with domain experts to verify their redundancy. This should also lead to further improvements in the metrics and detection method. Furthermore, we must also verify that the reduced test suite is as effective as the whole one.

5. References

[1] Black, J., Melachrinoudis, E., and Kaeli, D. Bi-Criteria Models for All-Uses Test Suite Reduction. In *Proceedings of the 26th International Conference on Software Engineering* (May 23 - 28, 2004). IEEE Computer Society, Washington, DC, 106-115.

[2] Harrold, M. J., Gupta, R., and Soffa, M. L. A methodology for controlling the size of a test suite. *ACM Trans. Softw. Eng. Methodol.* 2, 3 (Jul. 1993), 270-285.

[3] Leon, D. and Podgurski, A. A Comparison of Coverage-Based and Distribution-Based Techniques for Filtering and Prioritizing Test Cases. In *Proceedings of the 14th International Symposium on Software Reliability Engineering* (November 17 - 21, 2003). IEEE Computer Society, Washington, DC, 442.

[4] Salton, G. and McGill, M.J. *Introduction to Modern Information Retrieval*. McGraw-Hill, (1983), 104-109.