A Survey of Coverage-Based Testing Tools

Article *in* The Computer Journal · January 2006

DOI: 10.1093/comjnl/bxm021 · Source: OAI

3 authors, including:

Jenny li
University of Technology Sydney
46 PUBLICATIONS **558** CITATIONS

SEE PROFILE

David Weiss
Iowa State University
89 PUBLICATIONS **5,128** CITATIONS

SEE PROFILE

A Survey of Coverage Based Testing Tools

Qian Yang, J. Jenny Li, and David Weiss
Avaya Labs Research, 233 Mt. Airy Rd., Basking Ridge NJ 07920
{yangqian, jjli, weiss}@research.avayalabs.com

ABSTRACT

Test coverage is sometimes used as a way to measure how thoroughly software is tested. Coverage is used by software developers and sometimes by vendors to indicate their confidence in the readiness of their software. This survey studies and compares 17 coverage-based testing tools focusing on, but not restricted to coverage measurement. We also survey additional features, including program prioritization for testing, assistance in debugging, automatic generation of test cases, and customization of test reports. Such features make tools more useful and practical, especially for large-scale, real-life commercial software applications. Our initial motivations were both to understand the available test coverage tools and to compare them to a tool that we have developed, called eXVantage[1] (a tool suite that includes code coverage testing, debugging, performance profiling, and reporting). Our study shows that each tool has its unique features tailored to its application domains. Therefore this study can be used to pick the right coverage testing tools depending on various requirements.

Categories and Subject Descriptors

D.2.5 [Software Engineering]: Testing and Debugging — testing tools

General Terms: Reliability

Keywords: Code coverage, coverage-based testing tool, prioritization, automate test case generation, dominator analysis, eXVantage

[1] eXVantage stands for eXtreme Visual-Aid Novel Testing and Generation. eXVantage is a testing tool jointly developed by Avaya Research Labs and University of Texas at Dallas.

AST'06, May 23, 2006, Shanghai, China.

1. Introduction

Because of the increasing competitive pressure among numerous software vendors, the request for high quality software has increased. Software quality is a key differentiator in industries whose products rely on software for their operation. Examples of such industries include DVD player manufactures, auto makers, and avionics, where a small software defect may introduce considerable negative financial, customer relations, or even safety consequences. As a result, there is continuing pressure in these industries to improve software quality.

Software testing is a practice often used to determine and sometimes improve software quality. It is also a very labor and resource intensive process that often accounts for more than 50% of the total cost of software development [1]. Indeed, understanding the time and resources that should be allocated to testing involves a trade-off among budget, time and quality [2]. Finding an effective and efficient software testing tool could be a life-saver for a project or a company. Yet there is no single test tool suitable for all possible systems and industry sectors. Deciding what criteria to apply when selecting a specific tool for a project is quite tricky. For example, some tools integrate seamlessly with your choice of IDE (e.g. Eclipse) and provide user-friendly interfaces to ease unit testing in the development stage, but have scalability issues. Those tools are suitable for a small project, but not a large-scale real-life commercial application that sometimes includes a large percentage of legacy code. Other tools provide great testing granularity, but the performance overhead inevitably prevents them from being useful in real-time or embedded systems.

This survey studies various criteria that practitioners should consider when picking a coverage based testing tool; it encompasses the following issues.

First, the topic of testing is very broad. One widespread, but oversimplified model to categorize testing approaches is the structural/behavioral, or white-box/black-box model. Structural tests, also known as white-box tests, are based on how a system operates. They involve a detailed knowledge of the implementation of a system. Behavioral tests, also known as black-box tests, are based on what a system is required to do. They use typical user scenarios without delving into the code. Because of their different views of the system, black box and white box test tools are not comparable.

We focus this survey on tools that measure testing coverage. Coverage-based testing provides a way to quantify the degree of thoroughness of white-box testing.

Second, there are many available test tools, both commercial and open-source software. We selected only those with code coverage capabilities. We found 16 tools that fit our category. Information about them is available in the public domain. In-house or private coverage-based test tools are not in the scope of this study.

This survey is conducted with a dual purpose. Besides studying the 16 public coverage-based test tools, our goal was to evaluate an in-house tool suite, eXVantage (a tool suite for code coverage testing, debugging, performance profiling, etc.). In addition to coverage measurement, test tools often provide some other functions, such as rule checking, profiling, and debugging assistance. As stated previously, we exclude tools that only conduct static analysis, load testing or functional testing without coverage measurement. We compiled descriptions of each test tool based on the information in the public domain, but our descriptions were not reviewed by the venders providing the software. The descriptions are factored into many different categories covering important functions and features of the testing tools.

The rest of this paper is organized as follows. Section 2 discusses some important aspects of coverage measurement, including programming languages, granularity, overhead and so on. Section 3 presents various coverage criteria supported by each tool. It also illustrates how some of the tools do prioritization to get "good test cases" in an efficient way for complex software suites. Section 4 discusses automating test case generation in the context of coverage-based testing. It also covers the user interfaces of the tools. GUI and batch-mode versions provide different benefits. A comprehensive reporting component facilitates the communication among the team members or to the customers. Our observations appear in section 5.

2. Coverage Measurement

All tools included in this survey have coverage measurement capability, but may apply only to a limited set of programming languages, some to C/C++ only, some to Java only, some to both, and some to other languages such as FORTRAN, COBOL, or JavaScript. Table 1 shows a complete list of the tools and the languages that they support.

The selection of languages reflects each company's target industries. Companies that provide tools for system software, or embedded software vendors tend to focus more on supporting C/C++. Such tools are designed to introduce minimum performance overhead so that the tool is usable in real-time environments, e.g. CodeTest [8]. TCAT claims that its TCAT C/C++ Version 3.2 maintains its overhead for execution size ratio at 1.1-1.8 and execution speed ratio at 1.1-1.5 [18]; Semantic Designs claims 1.1-1.3, varying according to language and compiler, among the best in our survey.

Tool Name:	C++/C	Java	Other
Agitar [3]		X	
Bullseye [5]	X		
Clover [6]		X	.net
Cobertura [7]		X	
CodeTest [8]	X		
Dynamic [9]	X		
EMMA [21]		X	
eXVantage [4]	X	X	
gcov [10]	X		
Insure++ [11]	X		
Intel [12]	X		
JTest [15]		X	.net
JCover [13]		X	
Koalog [14]		X	
PurifyPlus [16]	X	X	Basic, .net
Semantic Designs (SD) [17]	X	X	C#, PHP, COBOL, PARLANSE
TCAT [18]	X	X	

Table 1. Coverage Tools and the Languages To Which They Apply

Another very market-conscious decision for C/C++ tool suppliers is the selection of supported platforms, which is not an issue for Java testing tools. DMS's customers are mostly medium to large Solaris software development shops [9]. It supports a limited number of operating systems. BullseyeCoverage supports a wide range of platforms among code coverage analyzers. Semantic Designs has a general foundation and process for instrumenting source code in not-widely-used languages on a variety of platforms [22].

Source code instrumentation, used by most of the tools including BullseyeCoverage, Insure ++, Intel Code Coverage Tool, Semantic Design, and TestWork, requires recompilation time, but provides more direct results and is more adaptable to a wide variety of processors and platforms. It can't be used for 3rd party code when the source code is not available. Tools such as DMS's Dynamics, use runtime instrumentation, which makes them feasible in a production environment. They may be more efficient in term of compilation time, but less portable. Agitar's Agitator runs the code in a modified Java Virtual Machine (JVM), which is also a dynamic instrumentation approach.

Our tool, eXVantage, uses source code instrumentation for C/C++ and bytecode instrumentation for Java. As compared to the other 16 tools, it has the least instrumentation overhead because it uses a sophisticated global dominator analysis method to reduce the number of instrumented probes for coverage measurement. The analysis method is based on the control flow diagram of a program under test. [20]

In practice, debugging always follows testing. Some of the coverage tools provide debugging assistance, such as Agitar, Dynamic, JCover, JTest, and Semantic Designs. Their solutions are all different. For example, Agitar provides a snapshot and stack trace to help developers to track the cause of the bugs. JCover has the ability to do coverage differencing and comparison to expose the error code. Semantic Designs provides slicing and dicing operations on test coverage data via the GUI to allow code executed/not executed by arbitrary combinations of test runs to be easily isolated visually. eXVantage uses a dynamic execution slicing approach. It creates an execution slice for each test case and reads results from a testing oracle to generate a bug localization report automatically whenever a failed test is detected. [23]

3. Coverage Criteria and Prioritization

There is a large variety of coverage measurement criteria: statement coverage (line coverage), decision coverage (branch coverage), path coverage, function/method coverage, class coverage and so on. Picking the right measurement requires balancing usability with thoroughness. Some tools provide various levels of code coverage information. Table 2 gives a list of tools with their coverage measurement criteria. Two tools, Koalog, and Intel, are not on the list because we cannot find their coverage criteria in the public domain.

While using thorough testing to help build low-defect software is the goal, in practice most applications are tested with low coverage. Code coverage of 60% to 70 % is often considered acceptable because of the difficulty in increasing code coverage past 60%. Hence, providing assistance in achieving high code coverage in an effective way is one of the most important features a good code coverage tool can provide. Among the tools we surveyed, only a handful suggest the availability of this feature.

Agitator from Agitar [3] shows risk or complexity scores of classes or methods, which in turn help testers to focus on more error-prone parts of the code, when time or resources are limited.

Cobertura [7], a free Java tool that calculates the percentage code accessed by tests, shows the McCabe cyclomatic code complexity of each class, and the average cyclomatic code complexity for each package and for the overall product. Cyclomatic complexity essentially represents the number of paths through a particular section of the code, such as a method in an object-oriented language. It is helpful in pinpointing areas of code that may require additional attention during testing, maintenance or refactoring phase.

	Line	Decision	Method	Class
Agitar [3]	X	X	X	X
Bullseye [5]		X	X	
Clover [6]	X	X	X	
Cobertura [7]	X	X		
CodeTest [8]	X	X		
Dynamic [9]	X	X	X	
EMMA [21]	X		X	X
eXVantage [4]	X	X	X	
gcov [10]	X			
Insure++ [11]	X			
JCover [13]	X	X	X	X
JTest [15]	X	X		
PurifyPlus [16]	X		X	
SD [17]	X	X	X	X
TCAT [18]	X	X	X	X

Table 2. Levels of Coverage Measurement Provided By Tools

Some other tools, such as Dynamic Suite [9], can run in the production mode or even at customer sites to obtain the information on which features or modules are being used. The tool has so little performance impact that it can be used in the field during normal system operation to collect operation profiles of the target system without interfering with its normal operation. This operation profile information can guide future testing and therefore help prioritize testing efforts.

eXVantage derives prioritization through a sophisticated dominator analysis [19]. It enhances conventional dominator analysis to include the impact of function/method calls, which increases the scalability of the tool. In addition, it relaxes the constraint of “guaranteed” to “at-least” coverage relationship to improve testing performance without losing accuracy [20]. eXVantage prioritization identifies the part of the code that can increase the code coverage the most. As by-products of this dominator analysis, dependency and control flow graphs of source code (when source code is available) are generated. They help visualize the testing coverage results, which will be discussed in more detail in the next section.

4. Automatic Test Generation and Reporting

Another important feature for comparison is automation. Software testing is a very resource consuming task. Automation is one way to cut down time and cost. Automation of testing process includes many steps, such as test case generation, test execution, and test oracles.

The approach to test oracle automation often relies on the system behavior specification and is mostly used in functional testing. Oracles use a system specification to verify the correctness of the target software test results. No full-scale system oracle exists today to achieve this goal. In general, a significant amount of human intervention is still needed to define oracles. Coverage testing is not linked directly with a test oracle, but an automatic test oracle could speed up debugging efforts when tests fail.

Another important automation area is test generation, which is more tightly linked with code coverage. Code-coverage based test generation is an important research area. None of the tools in our list can generate test cases for C/C++ code, but Parasoft, Agitar, and eXVantage claim the capability of generating Java test cases automatically. Parasoft has its patented test case generation technology. Agitator from Agitar [3] provides a certain level of automation by combining test suite generation and execution.

Agitator does Software Agitation, which is defined as "an automated way of exercising software code and providing observations about its behavior". That is, Agitator creates instances of the classes being exercised, calling each method with sets of input data (provided by static analysis of the source code to cover both boundary conditions and normal conditions), and analyzing the results. Subsequently, a set of summary observations about its behavior is presented to developers who can decide whether the observation is an assertion or a bug. Assertions would be kept for later regression tests or code refactoring.

All the agitation results are stored in XML files, which can be shared among the team, but would not be useful for other testing tools. The generated tests are not explicitly given to the users. Agitator supports tests created with JUnit by running them as part of each agitation, reporting outcome and coverage. Therefore, test cases that can drive testing efforts independent of any testing tools are not available to the testers.

Parasoft claims it has patented automated test case generation technology. Besides unit testing, the company provides solutions for web service, functional, rule compliance, security, and performance testing. It is possible that it generates random test data without taking into consideration increasing code coverage.

eXVantage generates tests to cover high priority blocks. It includes four steps: 1) use various approaches to rank the priorities of each target source code line and pick those with the highest priority; 2) identify paths going through the highest priority points, called hot-spots; 3) collect and solve constraints on the path, and 4) generate test data to execute the path and render test data into test cases in the same programming language as the original target program.

Besides automation, a friendly graph interface is also an important feature for comparison. The user interface can be a decisive element for a tool's usability. The first impression of a software tool is very important to users in their tool selection. There are two aspects of the user interface in this case: deployment and report generation.

Some tools have both a GUI version and a batch mode to suit the requirements of different users. Developers usually like to use the GUI version and the integrators usually like the batch mode version. Java tools, such as Agitar, Clover, Cobertura, eXVantage, and Parasoft, include plug-ins for one of the most popular IDE, Eclipse, which makes the integration in the development stage as transparent as possible. Apache Ant is selected as the build tool for some tools, such as Agitar, Clover, Cobertura, Koalog, and Parasoft, because it is very commonly used for Java projects.

One part of the GUI display or the output of the batch mode is the coverage report. Most commercial products include sophisticated report generation components, some of which are graph-based and some file-based. See table 3 for a list of report formats.

	GUI	File-based	Notes
Agitar [3]	X		
Clover [6]	X	X	PDF, XML
Cobertura [7]	X	X	XML
Dynamic [9]		X	
JCover [13]	X	X	XML, CSV
Koalog[14]	X	X	CSV, LaTex, XML
JTest [15]	X	X	Group reporting system
PurifyPlus [16]	X		
SD [17]	X	X	Test coverage vector file, XML
TCAT [18]		X	
eXVantage [4]	X	X	Customizable

Table 3. Tool Reporting Formats

Agitar has an innovative way to display coverage reports. Besides coverage information, it uses Agitar Management Dashboard to monitor and manage developer testing efforts. It is a comprehensive reporting tool, allowing users to input

test targets for better management, and providing rule checking functionality.

eXVantage, on the other hand, emphasizes web-based reporting. Its database reporting feature allows the recording of historic data and scaling up to very large software systems. Historic data allows the observation of trends, which can be used for future predictions of testing.

5. Conclusive Observations

In this survey, we compare 17 coverage-based testing tools. We also study other related functionalities, which we believe are indispensable for a testing tool to provide programmers and testers an integrated solution. Our study includes comparison of three features: 1) code coverage measurement, 2) coverage criteria, and 3) automation and reporting.

Our result shows each tool has its pros and cons depending on its application domain(s). For example, Semantic Designs's differentiator is their parsing capability for various languages, including obsolete ones. The strong point of Agitar is mutation-based data input to achieve very high code coverage running on their provided platform. The eXVantage tool suite, on the other hand, differentiates from other tools mostly in automatic unit test generation. It can automatically generate test cases to reach high priority points in the program, a step toward testing automation. eXVantage also provides a framework and code base for developers or testers to add more tests. Furthermore, it provides a solution for time and memory efficient instrumentation, done on either the source code or Java byte code level. eXVantage also applies an innovative way to do testing prioritization, through recovery of dependency and control flow diagrams by the application of a global dominator analysis.

Overall, much research in the area of software coverage testing has been realized and used in industrial software production. We hope our work will contribute to more usage of tools to improve software testing.

6. References

[1] Myers, Glenford J., The art of software testing, New York : Wiley, c1979.

[2] Yang, M.C.K.; Chao, A. Reliability-estimation and stopping-rules for software testing, based on repeated appearances of bugs; IEEE Transactions on Reliability, vol.44, no.2, p. 315-21, 1995

[3] Agitar (http://www.agitar.com/)

[4] Avaya eXvantage (http://www.research.avayalabs.com/user/jjli/eXVantage.htm)

[5] BullseyeCoverage (http://www.bullseye.com/index.html)

[6] Clover (http://www.cenqua.com/clover/)

[7] Cobertura (http://cobertura.sourceforge.net/)

[8] CodeTest (http://www.metrowerks.com/MW/Develop/AMC/CodeTEST/default.htm)

[9] Dynamic, DMS (http://dynamic-memory.com/)

[10] gcov (http://gcc.gnu.org/onlinedocs/gcc-3.0/gcc_8.html)

[11] Parasoft Insure ++ (http://www.parasoft.com/jsp/products/home.jsp?product=Insure&itemId=63)

[12] Intel code coverage tool (http://www.intel.com/cd/software/products/asmo-na/eng/219794.htm)

[13] JCover (http://www.mmsindia.com/JCover.html)

[14] Koalog (http://www.koalog.com/php/kover.php)

[15] Parasoft Jtest (http://parasoft.com/jsp/home.jsp)

[16] Purify Plus (http://www-306.ibm.com/software/awdtools/purifyplus/)

[17] Semantic Designs (http://www.semdesigns.com/index.html)

[18] TCAT (http://www.evalidator.org/TestWorks/)

[19] H. Agrawal, Dominators, super block, and program coverage: SIGACT Symposium on Principles of Programming Languages- POPL'94, Portland, Oregon, 1994, pp.25-34

[20] J. J. Li, Prioritize code for testing to improve code coverage of complex software: Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium, pp: 75- 84

[21] EMMA (http://emma.sourceforge.net/)

[22] I. D. Baxter, Branch Coverage for Arbitrary Languages Made Easy (http://www.semanticdesigns.com/Company/Publications/TestCoverage.pdf)

[23] W. E. Wong and J. J. Li, "An Integrated Solution for Testing and Analyzing Java Application in an Industrial Setting," in Proceedings of The 12th IEEE Asia-Pacific Software Engineering Conference (APSEC), pp. 576-583, Taipei, Taiwan, December 2005