A UNIFIED WEB-BASED FRAMEWORK FOR JAVA CODE ANALYSIS AND EVOLUTIONARY AUTOMATIC TEST-CASES GENERATION

Anastasis A. Sofokleous, Panagiotis Petsas
Department of Computer Science, University of Cyprus, 75 Kallipoleos Str., P.O.Box 20537, CY1678, Nicosia, Cyprus

Andreas S. Andreou
Department of Electrical Engineering and Information Technologie, Cyprus University of Technology
31 Archbishop Kyprianos Str., 3036 Lemesos, Cyprus

Keywords: Automatic test-case generation, Edge/condition coverage, k-Tuples coverage, Genetic algorithms.

Abstract: This paper describes the implementation and integration of code analysis and testing systems in a unified web-enabled framework. The former analyses basic programs written in Java and constructs the control-flow, data-flow and dependence graph(s), whereas the testing system collaborates with the analysis system to automatically generate and evaluate test-cases with respect to control flow and data flow criteria. The present work describes the design and implementation details of the framework and presents preliminary experimental results.

1 INTRODUCTION

Most work on software testing uses control and data flow graphs for examining the program under testing, extracting paths of interest and retrieving other important program information that can improve the efficiency of the testing process. Program models, such as control-flow graphs, can present useful information to the experienced programmers. Combined with automated test-case generation tools, control flow graphs can assist in the evaluation of testing adequacy, e.g. with the definition and use of a code coverage criterion. Code Analysis Systems can be used in both random and dynamic test data generators (Bertolino 2007, Godefroid *et al* 2005). Several test data generation schemes have been proposed, the most recent of which use genetic algorithms for exploring the huge search spaces of input values and determine the near to optimum solutions (Ghiduk *et al* 2007).

Many approaches have been proposed over time as standalone systems, console application and tools, presenting the core features of software testing systems. This work integrates core software testing subsystems into a web-enabled framework and expands the functionality of previous studies (Sofokleous and Andreou 2008b).

The rest of the paper is organized as follows: Section 2 presents related work and compares similar frameworks and tools. Section 3 describes in detail the proposed testing framework and section 4 introduces the reader to the supporting prototype software application and describes the evaluation procedure and some preliminary experimental results. Section 5 concludes the paper and proposes future research steps.

2 RELATED WORK

The proposed framework analyzes source-code written in Java and creates and visualizes control-flow, dataflow and dependence graphs. Dedicated modules of the framework generate test-cases in relation to a variety of coverage adequacy criteria defined on the control-flow and dataflow graphs of the selected source-code. Most test-data generation approaches use the source code to analyze the program under testing and guide the test process (Andrews *et al* 2006), whereas other approaches

A. Sofokleous A., Petsas P. and S. Andreou A. (2010).
A UNIFIED WEB-BASED FRAMEWORK FOR JAVA CODE ANALYSIS AND EVOLUTIONARY AUTOMATIC TEST-CASES GENERATION.
In *Proceedings of the 12th International Conference on Enterprise Information Systems - Databases and Information Systems Integration*, pages 407-410
DOI: 10.5220/0002911704070410

generate test-cases using the binary code or the Java Bytecodes, often with the assistance of the JVM Debugger Interface. This framework extends previous work, and specifically, it integrates components which analyze the code and create the control-flow, data-flow and dependence graphs (Sofokleous and Andreou 2008b, Sofokleous and Andreou 2008a). The proposed framework utilizes genetic algorithms for generating test cases according to the to edge/condition and the k-tuples criteria (Zhu *et al* 1997, Ntafos 1984). Genetic algorithms (GA) have gained their reputation in the area of test data generation in the last ten years as they are able to search a large search space of possible program inputs and reach to a set of near to optimal solutions, resulting in high code coverage.

This work studies and compares some of the most popular software testing and analysis frameworks currently available: Clover – CL (Clover 2009), Java Quality Solution - JQS (Java Quality Solution 1996), Parasoft Jtest – JT (Jtest 2007), Java Coverage Validator – JCV (Java Coverage Validator 2002), Quilt – QU (Quilt 2001), JCover – JC (JCover 2009), ispace – IS (ispace 2006) and Byecycle – BC (Byecycle 2008). Most of these tools focus on test-case generation and code coverage; ispace and Byecycle focus on creating dependence graphs.

Clover (Clover 2009), Jtest (Jtest 2007) and Java Coverage Validator (Java Coverage Validator 2002) frameworks execute on the bytecode-level unlike our framework which uses directly the source code. Analyzing the source-code is more difficult compared to bytecode-based analysis, which is neither programming language-specific, nor is associated with a certain programming style. Source code based analysis, however, allows better comprehension and maintenance of the program under testing, as it is closer to the programmer's logic and the business level (e.g. requirements). Many compilers offer graphs constructed on intermediate, bytecode or low code. The framework presented here works on the source code to create the code graphs necessary for its operations.

Java Quality Solution (Java Quality Solution 1996) and Quilt (Quilt 2001) are capable of creating data-flow and control-flow graphs, respectively. In terms of testing coverage, most of the tools use statement testing coverage; *JCover* and *JTest* use up to branch coverage. The majority of these frameworks present the coverage results in reports, e.g. exercised code, in terms of LOC, and the selected test cases. Some tools, such as Clover, JTest and JCover produce reports in HTML/XML and/or charts and diagrams. Our framework presents the results using both reports and graphs created during execution. Some of the presented frameworks utilize built-in rules to minimize design mistakes, regression testing, thread integrity testing, etc.

Byecycle (Byecycle 2008) is an Eclipse (Eclipse 2009) plug-in and can create a dependence graph illustrating the dependencies between packages, classes and interfaces. *ispace* (ispace 2006) creates dependence graphs as well, but these graphs show dependencies not only in an inter-package and inter-class level but also in more depth, and specifically between methods of a class. The proposed framework offers an even more analytic dependence graph; it can explore in more depth the methods of a class and can how the dependencies among statements, methods, inputs of methods, constructors, classes, control structures and outputs.

3 THE FRAMEWORK

The framework uses a program analyzer that creates and visualizes Control Flow Graphs (CFG), Data Flow Graphs (DFG) and Program Dependence Graphs (PDG) at the class level. The testing system uses genetic algorithms in order to generate test cases according to control flow or/and data flow criteria. The test data generation uses a set of adequacy criteria, e.g. the branch/condition coverage criterion and the k-tuples data flow criterion.

The program analysis and test data generation systems of the framework work closely together so as to satisfy the coverage criterion selected as follows:

The user invokes the Load Wizard (User Level, Figure 1) to select the source code and the type of graph(s) The Testing System Level of Figure 1 describes the testing layer. The option of testing is enabled after the analysis of the source code and the creation of the graph(s). Depending on the graph created, different types of coverage criteria can be used; the DF test-case Generation and the CF test-case Generation Systems are currently supported. The main difference between these two systems is the form of the fitness function that evaluates the generated test data, which, of course, as adapted to reflect each time the criterion used.

The two systems search for the near to optimum test cases and present the results to the user; apart from the optimum set of test cases, the user may view the coverage percentage, the evolutions and time needed and the coverage achieved with direct mapping of execution on the graphs . A test case/path may be

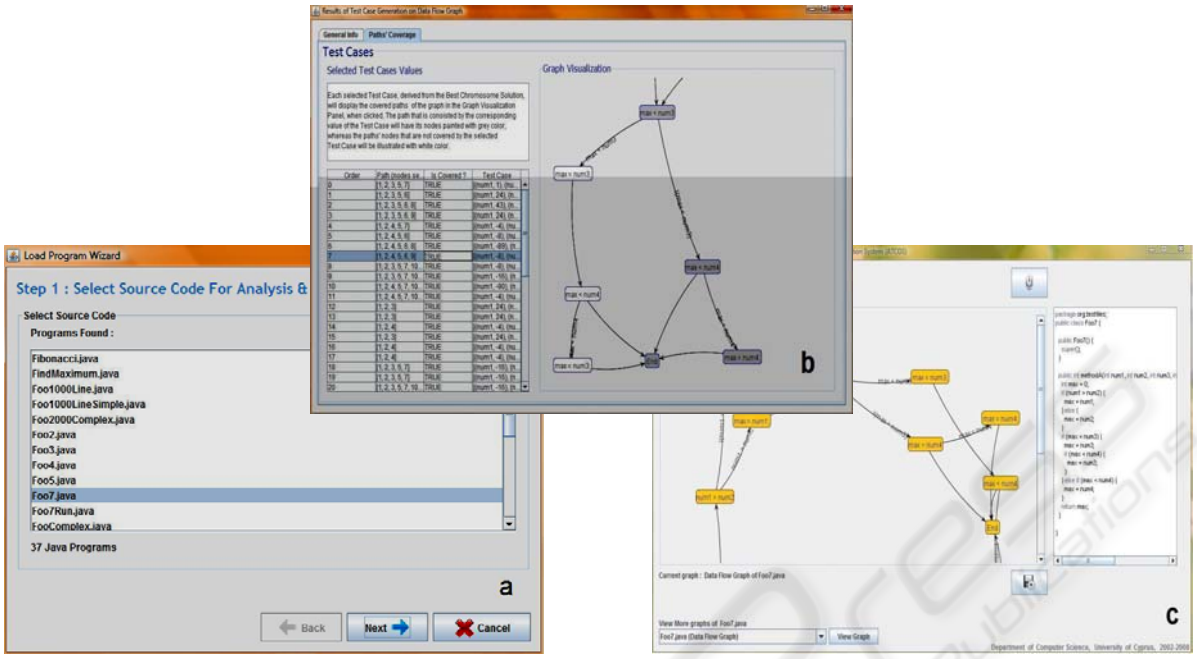

Figure 1: The prototype software application: a) Load Wizard: Step 1 (Source Code selection), b)Viewing test-cases and code coverage (Covered Paths), c) Viewing Graph and java syntax-highlighted editor.

highlighted on the graph through an interactive user interface.

4 PROTOTYPE SOFTWARE APPLICATION AND EVALUATION

The framework can run over the internet (online BPAS 2009). Using the Load Wizard (see figure 1), the user can select the JAVA source code for testing and the settings of the analysis, such as the type of graph to produce. The framework allows the user to select a program from a list of JAVA files; the source code of the selected program is downloaded over the internet and the framework utilizes the analysis modules and creates and visualizes the selected graphs. The graph is displayed in an interactive graph panel. Then the user may customize settings related to the testing process. When the test-case generation is over, the results are presented to the user. The results include information regarding the GA's execution and details of all of the successful test-cases produced in the chromosomes. This section presents the preliminary results of experiments carried out with the framework on a pool of standard and randomly generated JAVA programs. Each experiment cycle reported in this section assumes the following phases: sample source code downloading, parsing, graph construction and presentation of the graph in the Graph Panel. Note that this framework is able to parse large programs in terms of LOC, construct and graphically represent graphs of source code consisting of thousands of LOC (our experiments tested source code of up to 2000 LOC - see Table 1).

Table 1: Experiments on a selection of sample java programs with varying LOC.

LOC	Graph Creation Time (in milliseconds)		
	CFG	DFG	PDG
9	1279	1014	1342
12	1341	998	1294
16	1404	1092	1420
31	1545	1186	1466
35	1810	1451	1700
100	2606	24788	2512
5000	64768	61335	63578

5 CONCLUSIONS

This paper presented a web enabled framework that integrates program analysis and testing systems. Experiments conducted with a selection of input-programs show the efficiency of the framework on the unit testing.

Future work will attempt to enhance the framework by utilizing other forms of computational

intensive or/and intelligent techniques for automatic test data production like the Particle Swarm Optimization and Fuzzy Logic. Furthermore, future steps will consider integration of the framework with widely-used development tools and projects such as NetBeans, (Eclipse 2009), JBuilder and Project Maven.

REFERENCES

Andrews, J. H., Briand, L. ., Labiche, Y. and Namin, A. S., 2006, Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria, *IEEE Transactions on Software Engineering*, 32(8), pp. 608-624.

Bertolino, A., 2007, Software Testing Research: Achievements, Challenges, Dreams, in: *Proceedings of the 29th International Conference on Software Engineering (ICSE 2007): Future of Software Engineering (FOSE '07)*, Minneapolis, MN, USA, May 2007, (IEEE Computer Society: Los Alamitos, CA, USA), pp 85-103.

BYECYCLE, 2008, *byecycle, http://byecycle.sourceforge.net/* [Date accessed: 2 June 2009].

CLOVER, 2009, clover, http://www.atlassian.com/software/clover/ [Date accessed: 15 April 2009].

ECLIPSE, 2009, eclipse, http://www.eclipse.org [Date accessed: 15 May 2009].

Ghiduk, A. S., Harrold, M. J. and Girgis, M. R., 2007, Using Genetic Algorithms to Aid Test-Data Generation for Data-Flow Coverage, in: Proceedings of the 14th Asia-Pacific Software Engineering Conference (APSEC '07), Nagoya, Japan, December, (IEEE Computer Society: Washington, DC, USA), pp 41-48.

Godefroid, P., Klarlund, N. and Sen, K., 2005, Dart: Directed Automated Random Testing, in: Proceedings of the 2005 ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI '05), Chicago, IL, USA, June 2005, (ACM Press: New York, NY, USA), pp 213-223.

ISPACE, 2006, ISPACE, http://ispace.stribor.de/index.php?n=Ispace.Home [Date accessed: 28 April 2009].

JAVA COVERAGE VALIDATOR, 2002, java coverage VALIDATOR - software verification, http://www.softwareverify.com/java/coverage/feature.html [Date accessed: 18 April 2009].

JAVA QUALITY SOLUTION, 1996, java quality solution, http://www.parasoft.com/jsp/solutions/java_solution.jsp [Date accessed: 5 May 2009].

JCOVER, 2009, JCOVER, http://www.codework.com/JCover/product.html [Date accessed: 23 May 2009].

JTEST, 2007, jtest, http://www.parasoft.com/jsp/products/home.jsp?product=Jtest [Date accessed: 18 April 2009].

Ntafos, S. C., 1984, On required element testing, IEEE Transactions on Software Engineering, 10(6), pp. 795-803.

ONLINE BPAS, 2009, online BPAS, http://www.cs.ucy.ac.cy/~cs04pp2/dist/launch.html [Date accessed: 15 January 2010].

QUILT, 2001, QUILT, http://quilt.sourceforge.net/ [Date accessed: 3 May 2009].

Sofokleous, A. and Andreou, A., 2008a, Dynamic Search-based test data generation focused on data flow paths, in: Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS 2006), Barcelona, Spain, June, (INSTICC Press: Porto, Portugal), pp 27-35.

Sofokleous, A. A. and Andreou, A. S., 2008b, Automatic, evolutionary test data generation for dynamic software testing, The Journal of Systems & Software, 81(11), pp. 1883-1898.

Zhu, H., Hall, P. and May, J., 1997, Software Unit Test Coverage and Adequacy, ACM Computing Surveys, 29(4), pp. 366-427.