EvoSuite at the SBST 2015 Tool Competition

Gordon Fraser
University of Sheffield
Sheffield, UK

Andrea Arcuri
Scienta, Norway
and University of Luxembourg

***Abstract*—EvoSuite is a mature research prototype that automatically generates unit tests for Java code. This paper summarizes the results and experiences of EvoSuite's participation at the third unit testing competition at SBST 2015. An unfortunate issue of conflicting dependency versions in two out of the nine benchmark projects reduced EvoSuite's overall score to 190.6, leading to the overall second rank.**

***Keywords*-test case generation; search-based testing; testing classes; search-based software engineering**

I. Introduction

This paper describes the results of applying the EvoSuite test generation tool [2] to the benchmark used in the tool competition at the International Workshop on Search-Based Software Testing (SBST) 2015. Details about the competition and the benchmark can be found in [14]. In this competition, EvoSuite ranked second with a score of 190.6.

II. About EvoSuite

EvoSuite [2], [6] automatically generates test suites for Java classes, targeting branch coverage and other coverage criteria (e.g., mutation testing [8]). EvoSuite works at the Java bytecode level, i.e., it does not require source code. It is fully automated and requires no manually written test drivers or parameterized unit tests. For example, when EvoSuite is used from its Eclipse plugin, a user just needs to select a class, and tests are generated with a mouse-click.

EvoSuite has been evaluated on millions of lines of Java code [9], both open-source code and close-source code provided by one of our industrial partners. In the previous two editions of the unit testing tool competition, EvoSuite ranked first [4], [5].

EvoSuite uses an evolutionary approach to derive these test suites: A genetic algorithm evolves candidate individuals (chromosomes) using operators inspired by natural evolution (e.g., selection, crossover and mutation), such that iteratively better solutions with respect to the optimization target (e.g., branch coverage) are produced. For details on this test generation approach we refer to [6]. To improve performance further, we are investigating several extensions to EvoSuite. For example, EvoSuite can employ dynamic symbolic execution [13] and memetic algorithms [10] to handle the cases in which our genetic algorithm may struggle.

Table I
Classification of the EvoSuite unit test generation tool.

Prerequisites	
Static or dynamic	Dynamic testing at the Java class level
Software Type	Java classes
Lifecycle phase	Unit testing for Java programs
Environment	All Java development environments
Knowledge required	JUnit unit testing for Java
Experience required	Basic unit testing knowledge
Input and Output of the tool	
Input	Bytecode of the target class and dependencies
Output	JUnit test cases (version 3 or 4)
Operation	
Interaction	Through the command line, and Eclipse plugin
User guidance	manual verification of assertions for functional faults
Source of information	http://www.evosuite.org
Maturity	Mature research prototype, under development
Technology behind the tool	Search-based testing / whole test suite generation
Obtaining the tool and information	
License	GNU General Public License V3
Cost	Open source
Support	None
Does there exist empirical evidence about	
Effectiveness and Scalability	See [6], [9].

As the generated unit tests are meant for human consumption [11], EvoSuite applies various post-processing steps to improve readability (e.g., minimising) and adds test assertions that capture the current behavior of the tested classes. To select the most effective assertions, EvoSuite uses mutation analysis [12]. EvoSuite can also be used to automatically find faults such as undeclared thrown exceptions and broken code contracts [7]. For more details on the tool and its abilities we refer to [2], and for more implementation details we refer to [3].

III. Competition Setup

We configured EvoSuite to use a dynamic timeout of maximum 10 minutes per class for the search, with an earlier

stop if the fitness value did not increase for two minutes. The fitness function to drive the genetic algorithm was based on a combination of line coverage, branch coverage, and weak mutation testing [8]. We enabled the post-processing step of test minimization, but to reduce the time spent we included all assertions rather than filtering them with mutation analysis [12]. In practice, this may not result in the most readable or maintainable test cases, but neither of these two attributes is measured by the SBST contest metric.

IV. Benchmark results

The results of EvoSuite on the benchmark classes are listed in Table II. On average, EvoSuite achieved 55.36% line coverage, 47.19% branch coverage[1], and 41.02% mutation score. On average, EvoSuite took 6 minutes and 11 seconds per class to generate test cases. The generated test suites take on average 2.1 seconds to run per class.

A. Issues Encountered

Out of 63 classes under test (CUTs), EvoSuite did not manage to obtain any coverage for 18, i.e., 28% of all classes. For 14 of them (seven in the twitter4j project and seven in the hibernate project), this is due to a mismatch in libraries on the classpath. In particular, EvoSuite does bytecode instrumentation using the ASM library (currently using version 5.0.3). However, the projects of the CUTs had their own (older) version of ASM; for example Twitter4J has an indirect dependency to ASM 3.2.

Because the API of ASM has changed over different versions, this leads to errors like: "java.lang.IncompatibleClassChangeError: class org.objectweb.asm.tree.ClassNode has interface org.objectweb.asm.ClassVisitor as super class". Note: EvoSuite can be applied to its own library dependencies through its use of customized classloaders. However, bytecode instrumentation is also performed in the generated JUnit files (e.g., to support environment testing based on mock objects [1]), which leads to a runtime dependency to ASM. Consequently, EvoSuite generates tests for those 14 classes, but then all these tests fail due to the above mentioned exception. An easy solution would be to ship EvoSuite with its own ASM version using a different package name (e.g., by using the JarJar tool[2]). If we had handled this issue properly before the competition, this would have changed the outcome: Excluding classes from the twitter4j and hibernate, the overall score of EvoSuite would be 191.584, whereas the first ranked tool (GRT) would have a score of 164.464. Consequently, this issue clearly is the main factor affecting EvoSuite's overall result.

[1]Using Cobertura's definition of branch coverage, which only counts conditional statements, not edges in the CFG.

[2]https://code.google.com/p/jarjar/

For the other four classes with 0% coverage, EvoSuite failed to generate any tests for other reasons. For `CharMatcher`, there was an issue in how EvoSuite handled timeouts, which resulted in EvoSuite's master process killing the client process before tests were written to disk. For `CycleHandler` and `WikipediaInfo`, EvoSuite ran into an issue when trying to resolve the generic type parameters of some dependency classes; this also affects `Page` with 1.49% coverage. This issue could be avoided by omitting generic type parameters, as the Java compiler would only issue warnings about such missing parameters. However, as EvoSuite is aiming to produce readable tests, we feel it is important to properly handle Java Generics. Finally, for `Response` the constructor requires a parameter of type `java.net.HttpURLConnection`, which is an abstract class without concrete subclasses. As EvoSuite does not produce stubs automatically, it therefore failed to instantiate `Response` objects.

V. Conclusions

With an overall score of 190.6, EvoSuite achieved the second highest score of all tools in the competition. The score calculated for the best tool is 203.7: a very close call. In particular, if considering only projects without a configuration issue in the classpath of the target projects, EvoSuite would have scored first with a score of 191.6. The underlying issue can be easily fixed for future runs of the competition.

To learn more about EvoSuite, visit our Web site:

`http://www.evosuite.org`

References

[1] A. Arcuri, G. Fraser, and J. P. Galeotti, "Automated unit test generation for classes with environment dependencies," in *IEEE/ACM Int. Conference on Automated Software Engineering (ASE)*. ACM, 2014, pp. 79–90.

[2] G. Fraser and A. Arcuri, "EvoSuite: Automatic test suite generation for object-oriented software." in *ACM Symposium on the Foundations of Software Engineering (FSE)*, 2011, pp. 416–419.

[3] ——, "EvoSuite: On the challenges of test case generation in the real world (tool paper)," in *IEEE Int. Conference on Software Testing, Verification and Validation (ICST)*, 2013.

[4] ——, "Evosuite at the SBST 2013 tool competition," in *International Workshop on Search-Based Software Testing (SBST)*, 2013, pp. 406–409.

[5] ——, "Evosuite at the second unit testing tool competition." in *Fittest Workshop*, 2013.

[6] ——, "Whole test suite generation," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 276–291, 2013.

[7] ——, "1600 faults in 100 projects: Automatically finding faults while achieving high coverage with evosuite," *Empirical Software Engineering (EMSE)*, 2014.

Table II
DETAILED RESULTS OF EVOSUITE ON THE SBST BENCHMARK CLASSES. TIME IS EXPRESSED IN MINUTES.

Class	JUnit Files	Generation Time	Execution Time	% Line Coverage	% Branch Coverage	% Mutation Score
com.google.gdata.data.AttributeHelper	1.00	9.94	0.03	92.56	92.89	62.50
com.google.gdata.data.DateTime	1.00	3.62	0.02	83.31	71.67	86.25
com.google.gdata.data.Kind	1.00	4.03	0.04	57.07	48.86	46.74
com.google.gdata.data.Link	1.00	11.17	0.08	74.94	72.75	63.06
com.google.gdata.data.OtherContent	1.00	3.60	0.04	56.46	49.24	64.58
com.google.gdata.data.OutOfLineContent	1.00	8.96	0.04	89.75	85.12	29.17
com.google.gdata.data.Source	1.00	10.43	0.05	64.37	63.07	16.23
net.sf.javaml.core.AbstractInstance	1.00	3.31	0.02	13.28	5.36	0.00
net.sf.javaml.core.Complex	1.00	0.28	0.02	100.00	0.00	100.00
net.sf.javaml.core.DefaultDataset	1.00	3.87	0.02	95.31	97.50	57.80
net.sf.javaml.core.DenseInstance	1.00	3.92	0.02	98.87	100.00	100.00
net.sf.javaml.core.Fold	1.00	4.74	0.02	91.44	96.67	79.17
net.sf.javaml.core.SparseInstance	1.00	3.06	0.02	98.37	87.50	97.06
net.sf.javaml.tools.data.ARFFHandler	1.00	2.47	0.02	75.42	70.83	48.15
twitter4j.ExceptionDiagnosis	1.00	3.81	0.01	0.00	0.00	0.00
twitter4j.GeoQuery	1.00	3.80	0.01	0.00	0.00	0.00
twitter4j.OEmbedRequest	1.00	4.47	0.01	0.00	0.00	0.00
twitter4j.Paging	1.00	3.35	0.01	0.00	0.00	0.00
twitter4j.TwitterBaseImpl	1.00	11.48	0.01	0.00	0.00	0.00
twitter4j.TwitterException	1.00	5.12	0.01	0.00	0.00	0.00
twitter4j.TwitterImpl	1.00	16.35	0.01	0.00	0.00	0.00
com.puppycrawl.tools.checkstyle.api.AbstractLoader	1.00	2.28	0.03	77.00	50.00	30.00
com.puppycrawl.tools.checkstyle.api.AnnotationUtility	1.00	3.02	0.02	53.74	50.00	40.91
com.puppycrawl.tools.checkstyle.api.AutomaticBean	1.00	2.75	0.04	67.50	50.00	18.60
com.puppycrawl.tools.checkstyle.api.FileContents	1.00	5.50	0.03	97.08	91.67	80.67
com.puppycrawl.tools.checkstyle.api.FileText	1.00	2.94	0.03	88.62	85.26	91.84
com.puppycrawl.tools.checkstyle.api.ScopeUtils	1.00	14.16	0.09	72.63	50.33	24.04
com.puppycrawl.tools.checkstyle.api.Utils	1.00	3.35	0.03	93.99	92.31	90.48
com.google.common.base.CharMatcher	0.00	31.32	0.00	0.00	0.00	0.00
com.google.common.base.Joiner	1.00	3.37	0.03	86.52	97.10	79.79
com.google.common.base.Objects	1.00	2.63	0.03	98.13	95.37	94.59
com.google.common.base.Predicates	1.00	4.16	0.02	44.39	21.88	30.68
com.google.common.base.SmallCharMatcher	1.00	4.06	0.02	97.60	92.31	0.00
com.google.common.base.Splitter	1.00	4.88	0.02	94.35	91.35	76.56
com.google.common.base.Suppliers	1.00	3.07	0.02	60.38	58.33	56.06
org.hibernate.search.SearchException	1.00	0.26	0.01	0.00	0.00	0.00
org.hibernate.search.Version	1.00	0.24	0.01	0.00	0.00	0.00
org.hibernate.search.backend.BackendFactory	1.00	7.65	0.01	0.00	0.00	0.00
org.hibernate.search.backend.FlushLuceneWork	1.00	2.28	0.01	0.00	0.00	0.00
org.hibernate.search.backend.OptimizeLuceneWork	1.00	2.28	0.01	0.00	0.00	0.00
org.hibernate.search.util.logging.impl.LoggerFactory	1.00	0.27	0.01	0.00	0.00	0.00
org.hibernate.search.util.logging.impl.LoggerHelper	1.00	0.25	0.01	0.00	0.00	0.00
de.tudarmstadt.ukp.wikipedia.api.CategoryDescendantsIterator	1.00	2.51	0.05	8.24	0.00	0.00
de.tudarmstadt.ukp.wikipedia.api.CycleHandler	0.00	32.02	0.00	0.00	0.00	0.00
de.tudarmstadt.ukp.wikipedia.api.Page	1.00	2.75	0.06	1.49	6.00	2.67
de.tudarmstadt.ukp.wikipedia.api.PageIterator	1.00	3.76	0.06	47.72	46.83	38.54
de.tudarmstadt.ukp.wikipedia.api.PageQueryIterable	1.00	3.01	0.05	18.57	9.30	0.00
de.tudarmstadt.ukp.wikipedia.api.Title	1.00	2.98	0.02	89.51	83.33	98.81
de.tudarmstadt.ukp.wikipedia.api.WikipediaInfo	0.00	32.02	0.00	0.00	0.00	0.00
org.asynchttpclient.AsyncHttpClient	1.00	6.81	0.05	67.51	63.89	63.08
org.asynchttpclient.AsyncHttpClientConfig	1.00	10.24	0.06	93.73	61.11	83.03
org.asynchttpclient.FluentCaseInsensitiveStringsMap	1.00	5.59	0.02	98.72	95.13	87.21
org.asynchttpclient.FluentStringsMap	1.00	4.00	0.02	98.76	94.38	86.05
org.asynchttpclient.Realm	1.00	8.00	0.02	97.55	95.20	89.93
org.asynchttpclient.RequestBuilderBase	1.00	8.06	0.03	93.96	89.89	43.05
org.asynchttpclient.SimpleAsyncHttpClient	0.50	31.34	0.60	32.64	20.77	0.00
org.scribe.model.OAuthConfig	1.00	0.58	0.02	100.00	100.00	100.00
org.scribe.model.OAuthRequest	1.00	3.49	0.02	100.00	100.00	80.00
org.scribe.model.ParameterList	1.00	2.65	0.02	99.61	99.07	91.30
org.scribe.model.Request	1.00	3.18	0.02	62.15	40.91	28.68
org.scribe.model.Response	0.00	2.20	0.00	0.00	0.00	0.00
org.scribe.model.Token	1.00	2.35	0.02	100.00	100.00	77.27
org.scribe.model.Verifier	1.00	0.12	0.02	100.00	0.00	50.00
Average		6.19	0.03	55.36	47.19	41.02

[8] ——, "Achieving scalable mutation-based generation of whole test suites." *Empirical Software Engineering (EMSE)*, 2014.

[9] ——, "A large-scale evaluation of automated unit test generation using evosuite," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 24, no. 2, p. 8, 2014.

[10] G. Fraser, A. Arcuri, and P. McMinn, "A memetic algorithm for whole test suite generation," *Journal of Systems and Software*, 2014.

[11] G. Fraser, M. Staats, P. McMinn, A. Arcuri, and F. Padberg, "Does automated white-box test generation really help software testers?" in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. ACM, 2013, pp. 291–301.

[12] G. Fraser and A. Zeller, "Mutation-driven generation of unit tests and oracles," *IEEE Transactions on Software Engineering (TSE)*, vol. 28, no. 2, pp. 278–292, 2012.

[13] J. P. Galeotti, G. Fraser, and A. Arcuri, "Improving search-based test suite generation with dynamic symbolic execution," in *IEEE Int. Symposium on Software Reliability Engineering (ISSRE)*, 2013.

[14] U. Rueda, T. E. Vos, and I. Prasetya, "Unit testing tool competition - round three," in *International Workshop on Search-Based Software Testing (SBST)*, 2015.