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 EVO SUITE

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.

III. COMPETITION SETUP

We configured EvoSuite to use a dynamic timeout of maximum 10 minutes per class for the search, with an earlier stopping criterion in the case of problems where it was not possible to make progress. Details on the configuration of EvoSuite and the tool setup can be found in [14].

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].
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% mu-
tation 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 used with 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 pack-
age 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 (GRU)
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, Evo-
Suite failed to generate any tests for other reasons. For
CharMatcher, there was an issue in how EvoSuite han-
dled 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 Engineer-
2. G. Fraser and A. Arcuri, “EvoSuite: Automatic test suite
generation for object-oriented software,” in ACM Symposium
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
5. ———, “Evosuite at the second unit testing tool competition.”
in Fittest Workshop, 2013.
6. ———, “Whole test suite generation,” IEEE Transactions on
7. ———, “1600 faults in 100 projects: Automatically finding
faults while achieving high coverage with evosuite,” Empiri-
cal Software Engineering (EMSE), 2014.


