Java Unit Testing Tool Competition - Ninth Round

Sebastiano Panichella
panc@zhaw.ch
Zurich University of Applied Science (ZHAW)
Zurich, Switzerland

Alessio Gambi
alessio.gambi@uni-passau.de
University of Passau
Passau, Germany

Fiorella Zampetti
fiorella.zampetti@unisannio.it
University of Sannio
Benevento, Italy

Vicenzo Riccio
vincenzo.riccio@usi.ch
Software Institute - USI
Lugano, Switzerland

ABSTRACT

We report on the results of the ninth edition of the Java Unit Testing Competition. This year, six tools, Randoop, UtBot, Kex, Tardis, Evosuite and EvosuiteDSE, were executed on a benchmark with (i) new classes under test, selected from three open-source software projects, and (ii) the set of classes from three project considered in the eight edition. We relied on an improved Docker infrastructure for the execution of the different tools and the subsequent coverage and mutation analysis. Given the high number of participants, we considered two time budgets for test case generation: thirty seconds an two minutes. This paper describes our methodology, the statistical analysis of the results together with the contestant tools, and the challenges faced while running the competition experiments.

CCS CONCEPTS

• **Software and its engineering** → *Search-based software engineering*; **Automatic programming**; **Software testing and debugging**.

KEYWORDS

tool competition, benchmark, software testing, test case generation, unit testing, Java, JUnit

ACM Reference Format:
Sebastiano Panichella, Alessio Gambi, Fiorella Zampetti, and Vicenzo Riccio. 2021. Java Unit Testing Tool Competition - Ninth Round. In *IEEE/ACM 43rd International Conference on Software Engineering Workshops (ICSEW'21), May 25–28, 2021, Madrid, Spain.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

This year, the ninth edition of the Java Unit Testing Competition, has as participants Randoop [13], UtBot, Kex [4], TARDIS [6], Evosuite [9] and EvosuiteDSE [2]. Each tool, except for TARDIS, has been executed with a time budget of thirty seconds and two minutes on 98 classes under test selected from three new open source projects and three open source projects already used in the previous edition [8] of the competition. As regards TARDIS, due to limitations related to the implemented algorithm as being highlighted by the authors, we have used a time budget of five minutes.

We have compared the competitors' tools by using both line and branch coverage metrics, as well as, mutation analysis to evaluate the potential of the generated test suites in revealing fault, for each time budget. Both the execution of the tools for generating test suites, and their evaluation, computing code coverage metrics together with performing mutation analysis, has been carried out by using a dockerized infrastructure hosted on Github[1].

The remainder of the report is structured as follows. Section 2 describes the benchmark being adopted once having described the selection criteria. Section 3 briefly describes the contestants' tools, while Section 4 presents the methodology for running the competition. The results are detailed in Section 5, and Section 6 concludes the report with remarks and ideas for future improvements.

2 THE BENCHMARK SUBJECTS

The extraction of the classes under test (CUT) to use as benchmark for unit test case generators has been conducted considering several factors: (i) inclusion of different application domain [9], (ii) replicability, so open source projects may be preferable, and (iii) no trivial classes (e.g., classes without branches) [14]. We focused on GitHub projects (i) relying on Maven or Gradle as build framework, and (ii) including JUnit4 test suites. We used three out of four projects from the eight edition [8], by using a recent version of them, and we added three new ones. Specifically, we picked:

- *Guava (v29.0)* (https://github.com/google/guava), a set of Java libraries widely used within Google;
- *Seata (v1.3.0)* (https://github.com/seata/seata), a distributed transaction solution;
- *Spoon (v7.2.0)* (https://github.com/INRIA/spoon), a meta programming library to analyze and transform Java source code [11];
- *FastJSON (v1.2.66)* (https://github.com/alibaba/fastjson), a JSON parser and generator for Java;
- *Okio (v1.16.0)* (https://github.com/square/okio) a I/O library for Android, Kotlin and Java;

[1] https://github.com/JUnitContest/junitcontest

ICSEW'20, May 25–28, 2021, Madrid, Spain

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: Characteristics of the benchmark.

Project	Cand.	Samp.
Guava	274	25
Seata	29	6
Spoon	179	15
FastJSON	120	20
Okio	35	7
Weka	1165	25
Total	1802	98

- *Weka (v3.8)* (https://github.com/Waikato/weka-3.8), i.e., a workbench for machine learning.

Based on the time and resources available for running out the competition, we have only sampled a limited number of CUTs using the approach adopted in the last edition [8]. Once having removed the classes in which the whole set of methods have a McCabe's cyclomatic complexity lower than five, we randomly sampled 98 classes from the six projects proportionally to their overall number of testable classes. During the random process, once identified a class, we ran Randoop with a time budget of 10 seconds to exclude classes where Randoop cannot provide any test case.

Table 1 reports, for each project, the total number of candidate CUTs together with the number of classes used as benchmark.

3 THE TOOLS

Six tools are competing in this ninth edition: Randoop [13], UtBot, Kex [4], TARDIS [6], Evosuite [9] and EvosuiteDSE [2].

Randoop generates unit tests using a feedback-directed random test generation, collecting information from the execution of the tests as they are generated to reduce the number of redundant and illegal tests [13]. Differently, **UtBot**, a tool implemented by Huawei (Russian Research Institute, Saint Petersburg Research Center, Software Analysis Team), relies on symbolic execution extracting the information about the execution paths identified inside the method to derive the constraints that need to be met for traversing a desired path. By using the SMT (Satisfiability Modulo Theory) solver, UtBot builds a model (i.e., a set of parameter values for the method under test) satisfying the above constraints with the aim of finding a model satisfying all possible execution paths of the method under test. Similarly, **Kex** [4] works as a symbolic execution engine and uses SMT solvers to perform the constraint solving. By analyzing jar files, it constructs the control flow graph for each method and tries to cover each basic block in each method by generating sufficient input data. Finally, by using a novel backward search algorithm, namely Reanimator, Kex constructs valid test cases from generated input parameters. **TARDIS** [6], with the aim of maximizing branch coverage) performs concolic (i.e., mixed concrete-symbolic) execution on a set of initial tests to generate constraints on the program inputs that may determine the execution of a program path towards an uncovered branch. It then solves these constraints and generates new test cases by exploiting a search-based algorithm. The generated tests are fed back to concolic execution in order to discover new paths. Similarly to TARDIS, **EvosuiteDSE** [2] uses a pure dynamic symbolic execution approach along with a generational search exploration strategy in an attempt to maximize branch coverage. It also uses Evosuite's previously developed test suite post-processing techniques (test suite minimization). Finally, **Evosuite** [9] uses an evolutionary algorithm to evolve a set of unit tests satisfying a given set of test objectives.

4 THE CONTEST METHODOLOGY

The methodology followed to run the competition is similar to the one adopted in the eight edition [8]. Due to time constraints and amount of resources required to compare the various tools, we decided to focus on two different time budgets (i.e., 30 and 120 seconds), as well as, on a limited set of classes in the case of more computational demanding tools.

Public contest repository. The complete contest infrastructure is released under a GPL-3.0 license and is available on Github [1]. The repository also contains the benchmark (i.e., CUTs), detailed reports and data for the ninth edition, as well as, for previous ones.

Execution environment. The infrastructure performed a total of 6,250 executions (2,800 in the previous edition). In principle, without considering the resources needed for the execution of each competitors, we planned to have 98 CUTs x 6 tools x 2 time budgets x 10 repetitions, resulting in 11,760 executions in total to use for statistical analysis. However, only for Randoop, Evosuite and EvosuiteDSE we were able to run the planned number of executions. As regards the other three competitors' tools we realized that, (i) Kex was not able to produce any test case for four out of six projects in our benchmark, so we only have results for 30 CUTs belonging to Guava and Seata; (ii) TARDIS, as reported by original authors, cannot provide useful results for time budgets less than two minutes, for this reason we randomly picked 30 CUTs from our benchmark and executed it using a time budget of five minute; finally (iii) Utbot requires too much memory and disk space so we executed it only on 50% of the total number of CUTs in our benchmark. The executions were run in parallel using Docker on four servers with similar characteristics: Linux Flavor with 8 CPU cores, 16 GB of RAM and 240 GB memory.

Test generation and time budget. Once accounted for each tool constraints, we executed each tool ten time against each CUT for each time budget in order to reduce the randomness of the generation processes [7]. Furthermore, we considered two different time budgets, i.e., 30 and 120 seconds (except for TARDIS, where we experimented a time budget of five minutes).

Metrics computation. As for last year [8], the time budget used for mutation analysis has been set to five minutes for each class under test, while the timeout considered for each mutant has been set to one minute. The mutants has been sampled among the ones generated by PITest [5] using the following procedure: for CUTs with more than 200 mutants we randomly kept only 33% of them, while for CUTs with more than 400 mutants we sampled 50% of them for our analysis. Finally, for coverage metrics, we focused on lines and branches coverage by relying on JaCoCo [3].

Statistical analysis. Similarly to previous editions[8], we used statistical tests to support the results. Specifically, we use the Friedman test for assessing whether the scores over the different CUTs and time budgets achieved by the competitors tools are significantly different from each other; then we use the post-hoc Conover's test to determine for which pair of tools the significance actually holds, once having adjusted them with the Holm-Bonferroni procedure.

5 RESULTS

Table 2 presents, for each tool and for each time budget the minimum, mean, median and maximum number of test cases being generated. As expected, in almost all the cases, while increasing the time budget given for generation purposes, the number of test cases being produced increase. Furthermore, for 19 CUTs in our benchmark, at least one tool was not able to generate any test case.

Table 2: Statistics on number of test cases generation for each tool and each time budget.

Tool	Time budget	Min	Mean	Median	Max
Randoop	30	0	1502	754	9428
	120	0	2801	1429	18115
Evosuite	30	0	70	65	229
	120	0	56	49	313
EvosuiteDSE	30	0	5	2	29
	120	0	16	2	356
Utbot	30	0	341	461	595
	120	0	4589	6575	6729
Kex	30	0	28	14	150
	120	0	30	19	150
TARDIS	300	0	1388	1395	2796

Going deeper on the evaluation of the test cases being generated by each tool, we observed cases for which our infrastructure was not able to compute metrics in terms of both coverage and mutant analysis. By removing those cases from our analysis, Figure 1, Figure 2 and Figure 3 show the ratio of lines, branches and mutants being covered by Randoop, Evosuite, EvosuiteDSE, Utbot and Kex, for each specific time budget. Note that the mutation coverage is the ratio between the mutants that were killed by at least one test and the total number of mutants. Unsurprisingly, while increasing the time budget, also the median line, branch and mutant coverage (slightly) increase. This is more evident for Utbot and Evosuite (see figure 3). Specifically, for Utbot the number of mutants being killed over the total number of mutants moves from 34.3% (30 seconds) up to 40.1% with a time budget of 120 seconds, while for Evosuite increases from 0% up to 72.7%. Moreover, Evosuite achieves, on average, a higher coverage and mutation score for all the projects followed by Utbot. Specifically, for a time budget of 120 seconds, Evosuite and Utbot achieve (i) a line coverage of 93.2% and 31% compared to 20% of Randoop, 8% of Kex and 6.2% of EvosuiteDSE; (ii) a branch coverage of 87.5% and 30.1% compared to 13%, 1% and 0% of Randoop, Kex and EvosuiteDSE; and (iii) a mutation score of 72.7% and 40.1% compared to 8%, 2.7% and 7.1% obtained while using Randoop, Kex and EvosuiteDSE. Finally, while Evosuite is the tool that performs best, EvosuiteDSE is the worst on the CUTs selected for this ninth edition of the competition. It is important to know that these results can be influenced by the specific project versions and classes under tests considered this year as well as the usage of 30 seconds and 2 minutes as time budgets.

Due to resources and time limitations, we were able to generate test cases by using a time budget of five minutes only for three out of six tools, namely EvosuiteDSE, Utbot and Evosuite. Figure 4 shows the comparison between them while looking at line, branch and mutant coverage ratio. The graph clearly shows a trend between those three tools with EvosuiteDSE performing the worst

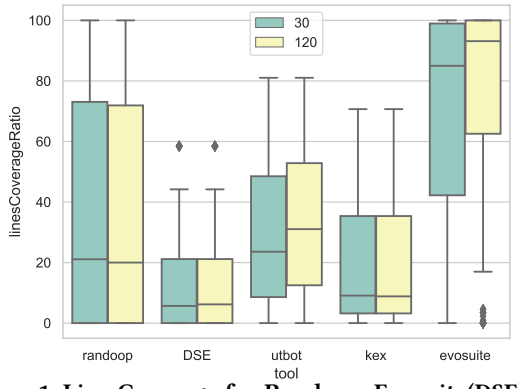

Figure 1: Line Coverage for Randoop, Evosuite(DSE), Utbot, Kex and Evosuite for 30 and 120 seconds.

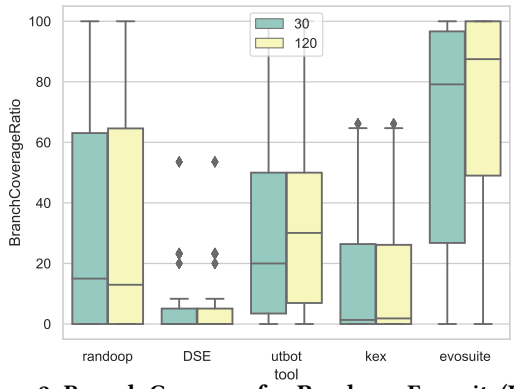

Figure 2: Branch Coverage for Randoop, Evosuite(DSE), Utbot, Kex and Evosuite for 30 and 120 seconds.

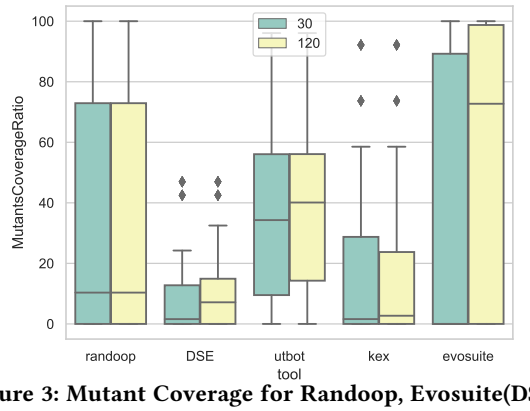

Figure 3: Mutant Coverage for Randoop, Evosuite(DSE), Utbot, Kex and Evosuite for 30 and 120 seconds.

and Evosuite performing the best. Specifically, the median values for line, branch and mutant coverage are doubled while comparing Evosuite with Utbot. For instance, EvosuiteDSE has a median line coverage ratio of 6.2% compared to 31.1% of Utbot and 92.9% of Evosuite.

For next year we plan to investigate this further, by including changes in the infrastructure that allow parallel executions, thus allowing the execution of different and greater time budgets (e.g., 5

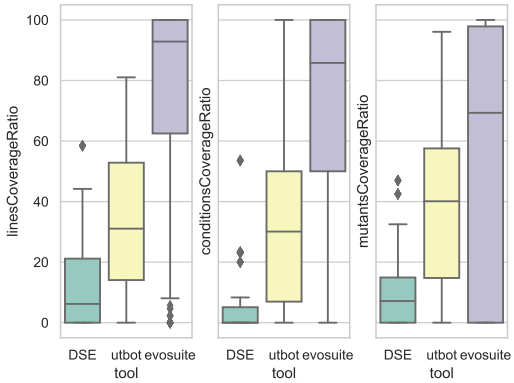

Figure 4: Coverage for Evosuite(DSE), Utbot and Evosuite on a time budget of 5 minutes.

and 10 minutes), for a more complete comparison of the various tools.

Even if, in principle we could compare TARDIS performance with the ones of Evosuite, EvosuiteDSE and Utbot for a time budget of five minutes only on a subset of the CUTs, we realized that, while computing the metrics for TARDIS, our scripts were not able to produce useful values on the 30 sampled CUTs.

Finally, Table 3 reports the final score and ranking achieved by the tools at different search budgets as well as the ranking produced by the Friedman test. It is important to mention that in doing the comparison we limited the results to the ones available for the six competitors' tools.

Table 3: Final score and ranking for each tool.

Tool	Score	Ranking
Evosuite	292.05	1.43
Randoop	121.04	2.56
EvosuiteDSE	47.14	3.33
Utbot	87.76	3.53
Kex	44.21	4.15

6 CONCLUSIONS AND FINAL REMARKS

This year was the ninth edition of the Java Unit Testing Competition. Compared to previous editions, this year we have five competitors, namely EvosuiteDSE, Kex, Utbot, TARDIS and Evosuite, and Randoop as a baseline, among which the best performing one is Evosuite followed by Utbot, while, quite surprisingly, EvosuiteDSE seem to perform the worst on the CUTs selected for this year.

The dockerized version of the infrastructure allowed us to distribute the execution on four different servers. However, most of the tool being submitted required too much RAM for generation (around 13GB for Kex, and 10 GB for TARDIS), as well as a huge amount of disk space for storing the results. As an example, Kex and Utbot required 140-150GB for storing generated test cases and intermediate results while executing on the whole set of CUTs in our benchmark. The above constraints limited us in properly parallelizing the execution in terms of automatic generation of test cases.

The two-steps procedure used to select the different CUTs proved to be useful again this year. It allowed us to discover configuration issues in the competition infrastructure (e.g., wrong class-paths) and avoid several of the difficulties encountered last year.

As future directions we envision several possibilities: we need to (i) properly verify the reasons why for some CUTs in our benchmark, our infrastructure was not able to produce both coverage and mutation analysis data; (ii) include additional criteria than the coverage and mutation analysis [10] for evaluation purposes; (iii) experiment with tools supporting the testing of more complex application (e.g., cloud-based systems [12]); and (iv) consider to extend the infrastructure to support other languages (e.g., Python).

ACKNOWLEDGMENTS

Sebastiano Panichella and Fiorella Zampetty gratefully acknowledges the Horizon 2020 (EU Commission) support for the project *COSMOS* (DevOps for Complex Cyber-physical Systems), Project No. 957254-COSMOS). We thank all participants of the unit-test tool competition of this and previous years, which continuously sustain the evolution and maturity of automated testing strategies.

REFERENCES

[1] 2021. Contest Infrastructure. https://github.com/JUnitContest/junitcontest. [Online; accessed 23-02-2021].
[2] 2021. EvoSuite DSE Module. https://github.com/ilebrero/evosuite/releases/tag/DSE.SBSTToolCompetition2021. [Online; accessed 23-02-2021].
[3] 2021. JaCoCo. https://www.jacoco.org/jacoco/trunk/doc/. [Online; accessed 23-02-2021].
[4] 2021. Kex. https://github.com/vorpal-research/kex/tree/sbst-21. [Online; accessed 23-02-2021].
[5] 2021. PiTest. http://pitest.org/. [Online; accessed 23-02-2021].
[6] 2021. Tardis (meTAheuRistically-driven DynamIc Symbolic execution). https://github.com/pietrobraione/tardis. [Online; accessed 23-02-2021].
[7] Andrea Arcuri and Lionel Briand. 2014. A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering. *Softw. Test. Verif. Reliab.* 24, 3 (May 2014), 219–250. https://doi.org/10.1002/stvr.1486
[8] Xavier Devroey, Sebastiano Panichella, and Alessio Gambi. 2020. Java Unit Testing Tool Competition: Eighth Round. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. 545–548.
[9] Gordon Fraser and Andrea Arcuri. 2014. A Large-Scale Evaluation of Automated Unit Test Generation Using EvoSuite. *ACM Transactions on Software Engineering and Methodology* 24, 2 (dec 2014), 1–42. https://doi.org/10.1145/2685612
[10] G. Grano, C. Laaber, A. Panichella, and S. Panichella. 2019. Testing with Fewer Resources: An Adaptive Approach to Performance-Aware Test Case Generation. *IEEE Transactions on Software Engineering* (2019), 1–1.
[11] Fitsum Kifetew, Xavier Devroey, and Urko Rueda. 2019. Java Unit Testing Tool Competition - Seventh Round. In *2019 IEEE/ACM 12th International Workshop on Search-Based Software Testing (SBST)*. IEEE, 15–20. https://doi.org/10.1109/SBST.2019.00014
[12] Diego Martin and Sebastiano Panichella. 2019. The cloudification perspectives of search-based software testing. In *Proceedings of the 12th International Workshop on Search-Based Software Testing, SBST@ICSE 2019, Montreal, QC, Canada, May 27, 2019*, Alessandra Gorla and José Miguel Rojas (Eds.). IEEE / ACM, 5–6. https://doi.org/10.1109/SBST.2019.00009
[13] Carlos Pacheco and Michael D Ernst. 2007. Randoop: Feedback-Directed Random Testing for Java. In *Companion to the 22nd ACM SIGPLAN conference on Object oriented programming systems and applications companion - OOPSLA '07*, Vol. 2. ACM Press, 815. https://doi.org/10.1145/1297846.1297902
[14] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2018. Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets. *IEEE Transactions on Software Engineering* 44, 2 (2018), 122–158. https://doi.org/10.1109/TSE.2017.2663435