# Java Unit Testing Tool Competition - Eighth Round

Devroey, Xavier; Panichella, Sebastiano; Gambi, Alessio

**DOI**
10.1145/3387940.3392265

**Publication date**
2020

**Document Version**
Accepted author manuscript

**Published in**
Proceedings - 2020 IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW 2020

**Citation (APA)**
Devroey, X., Panichella, S., & Gambi, A. (2020). Java Unit Testing Tool Competition - Eighth Round. In *Proceedings - 2020 IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW 2020* (pp. 545-548). ACM DL. https://doi.org/10.1145/3387940.3392265

# Java Unit Testing Tool Competition - Eighth Round

Xavier Devroey
x.d.m.devroey@tudelft.nl
Delft University of Technology
Delft, The Netherlands

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

## ABSTRACT

We report on the results of the eighth edition of the Java unit testing tool competition. This year, two tools, EvoSuite and Randoop, were executed on a benchmark with (i) new classes under test, selected from open-source software projects, and (ii) the set of classes from one project considered in the previous edition. We relied on an updated infrastructure for the execution of the different tools and the subsequent coverage and mutation analysis based on Docker containers. We considered two different time budgets for test case generation: one an three minutes. This paper describes our methodology and statistical analysis of the results, presents the results achieved by the contestant tools and highlights the challenges we faced during the competition.

## 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:**
Xavier Devroey, Sebastiano Panichella, and Alessio Gambi. 2020. Java Unit Testing Tool Competition - Eighth Round. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3387940.3392265

## 1 INTRODUCTION

This year is the eighth edition of the Java unit testing tool competition. This year's participants are EvoSuite [10] and Randoop [8]. Each tool has been executed with a time budget of one and three minutes on 70 classes under test: 10 classes coming from last years' edition [6], and 60 classes taken from three projects never considered in past editions of the competition. We compare the tools for each time budget using well-established structural statement and branch coverage metrics, and mutation analysis to assess the fault revealing the potential of the generated test suites. The evaluation was carried out using our dockerized infrastructure that includes tooling to execute the different tools, compute code coverage metrics, and perform mutation analysis. Additionally, the infrastructure includes statistical analysis scripts to compare and rank the different test case generation tools. This year, we implemented several improvements to our infrastructure, including parallelizing the execution of mutation analysis, fixing the issues reported during last year's edition of the competition [6], and implementing a novel Docker image to execute the Java unit test case generators. The infrastructure is open-source and available on GitHub at https://github.com/JUnitContest/junitcontest.

In the remainder of this report, Section 2 presents the benchmark and selection procedure, Section 3 briefly describes the participating tools, Section 4 presents the methodology, Section 5 reports this year's results, and Section 6 concludes with remarks and ideas of future improvements.

## 2 THE BENCHMARK SUBJECTS

Choosing subjects for benchmarking unit test case generators should take into consideration several factors. The classes under test should be (i) a representative sample of real-world software covering different application domains [4]; (ii) preferably open-source to ease replicability of the study; and, (iii) should be not trivial [11] (e.g., classes should have branches in their methods and should require different types of input).

Taking these aspects into account, and considering also the need to automate the analysis, we focused on GitHub repositories that satisfy the following criteria: the project (i) can be built using Maven or Gradle, and (ii) includes JUnit 4 test suites. In addition to those, we included classes from a project selected in last year's edition which were particularly challenging for the competing tools. As a result, we selected the following projects:

- *Fescar/Seata* (https://github.com/seata/seata), an easy-to-use, high-performance, open source distributed transaction solution.
- *Guava* (https://github.com/google/guava), a set of core Java libraries extending the standard Java API.
- *PdfBox* (https://github.com/apache/pdfbox), an Apache Java API to work with PDF documents.
- *Spoon* (https://github.com/INRIA/spoon/), a library for analyzing and transforming Java source code, also used in last year's edition of the competition [6].

Considering all the classes in each project is not possible as it would require an extensive amount of time and resources for the competition. Following the lead of past editions [6], we sampled a limited number of Classes under test (CUTs) using a two-step procedure. In the first step, we computed McCabe's cyclomatic

*ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea*

**Table 1: Characteristics of the benchmark.**

| Project | Cand. | 1m | 3m | Filt. | Samp. |
|---|---|---|---|---|---|
| Fescar/Seata | 43 | 43.0m | 2.2h | 26 | 20 |
| Guava | 275 | 4.6h | 13.8h | 114 | 20 |
| PdfBox | 303 | 5.1h | 15.2h | 242 | 20 |
| Spoon | 437 | 7.3h | 21.9h | N.A. | 10 |
| **Total** | 1094 | 17.7h | 53.3h | 382 | 70 |

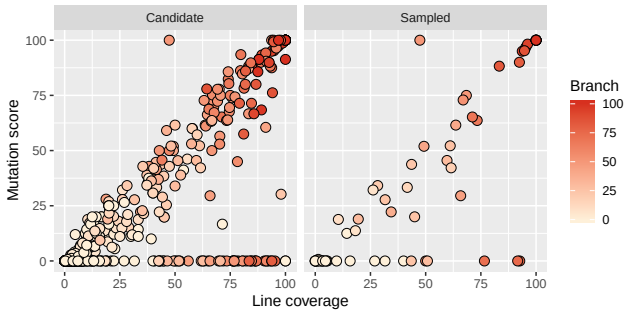


**Figure 1: Line coverage, branch coverage and mutation score for the candidate and selected CUTs.**

complexity for all methods and classes in each project using JavaNCSS[1] and filtered out classes that contain only methods with a complexity lower than five. Removing those classes reduces the chances to sample classes with few branches that can easily be covered by randomly generated tests [11]. This gave us a set of 1,094 candidate CUTs. In the second step, we executed Randoop with a time budget of 10 seconds against each candidate CUT (except for Spoon classes) and filtered out classes for which Randoop could not generate a single test. This reduces the chances of running into technical difficulties during the execution of the different tools. After filtering, this resulted in a set of 382 classes (Spoon classes excluded) from which we randomly selected 20 CUTs from each project. We considered 10 classes from Spoon which were used in the past edition of the competition [6].

Table 1 reports the main characteristics of the selected projects with the number of classes under tests. For each project, Table 1 details the number of candidate CUTs (Cand.), and, to give an idea of the time required for test case generation, the total estimated test case generation time when considering budgets of 1 (1m) and 3 minutes (3m) per candidate class. Finally, Table 1 gives the number of CUTs after filtering out CUTs for which Randoop could not generate a single test (Filt.), and the number of sampled CUTs (Samp.). Figure 1 reports the line and branch coverage, and the mutation score for the test cases generated by Randoop for all candidate CUTs (on the left), and the 60 sampled CUTs (on the right).

## 3 THE TOOLS

Two tools are competing in this eighth edition: Randoop [8], and EvoSuite [2, 10, 11]. Randoop relies on a *feedback-driven random testing* strategy [9], collecting information from the execution of the tests as they are generated to avoid redundant and illegal tests, to generate regression tests capturing how the system behaves as-is. EvoSuite uses an evolutionary algorithm to evolve a set of unit tests satisfying a given set of test objectives (for instance, covering the different branches of a CUT, or weakly killing a set of mutants) [4].

## 4 THE CONTEST METHODOLOGY

The methodology adopted in this year's edition is similar to the one adopted last year [6]. However, due to delays in the agenda, time constraints, and engineering concerns, we decided to drop the combined analysis of the results (evaluating the complementary of the different tools), and the comparison of the generated test cases with the manually written test suites. The following paragraphs described the main steps and modifications of the contest methodology.

**Public contest repository.** The complete contest infrastructure is released under a BSD3 license and is available on GitHub https://github.com/JUnitContest/junitcontest/. We improved the infrastructure by highly parallelizing the mutation analysis execution and stopping the test execution as soon as a mutant is killed, correcting several bugs related to coverage computation which were reported during last year's edition of the competition [6], and revising the procedure to update the Docker image readily usable to execute the Java unit test case generators. Additionally, the repository hosts the benchmarks, detailed reports and data of this year's as well as previous years' editions.

**Execution environment.** The infrastructure performed a total of 2,800 executions (4,560 in the previous edition): 70 CUTs × 2 tools × 2 time budgets × 10 repetitions for statistical analysis. The executions were run in parallel using Docker on two servers: one Linux Ubuntu (v 4.4.0-174-generic) with 40 CPU cores (Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz) and 128 GB memory, and one with one Linux Ubuntu (version id 19.10) with 8 CPU cores (Intel Core Processor (Broadwell, no TSX, IBRS) CPU v6 @ 2.49GHz) and 160 GB memory.

**Test generation and time budget.** Each tool was executed ten times against each CUT for each time budget to take randomness of the generation processes into account [1]. We considered two different time budgets: 1 and 3 minutes.

**Metrics computation.** As for last year, we kept the strict mutation analysis time budget of 5 minutes per CUT, and a timeout of 1 minute for each mutant, and we sampled the mutants generated by PITest [2]. We applied a random sampling of 33% for CUTs with more than 200 mutants, and a sampling of 50% for CUTs with more than 400 mutants. This year, however, we did not encounter difficulties while computing coverage metrics, thanks to the fixes applied to the competition infrastructure. We updated JaCoCo to the latest version (version 0.8.5) to compute line and branches coverage.

**Combined analysis and comparison with manually written tests.** Due to delays and time constraints for the execution of the combined analysis and publication of the present report, we could not perform the combined analysis and the comparison with manually written tests. The combined analysis gathers all the tests generated by the different tools and performs a coverage and mutation analysis, but requires an extensive amount of time to be

[1] Available at https://github.com/codehaus/javancss.

[2] http://pitest.org/

**Table 2: Overall scores and rankings obtained with the Friedman test**

| Tool | Score | Ranking |
|---|---|---|
| EvoSuite | 406.14 | 1.26 |
| Randoop | 310.75 | 1.74 |

performed as it re-executes all the tests. One possible improvement to the existing infrastructure would be to collect the full coverage and mutation analysis reports from JaCoCo and PIT to avoid re-executing all the combined test suites.

The comparison between automatically generated and manually written tests represents a great source of information for research. Unfortunately, this process is largely manual in the current competition infrastructure and poses several challenges. For instance, Google Guava contains several and diverse extensions to the Java standard API and therefore constitutes a great candidate for the evaluation of the unit test generation capabilities of a tool. However, the developers chose to keep Guava's test suite in a dedicated Maven module to allow for the tests to depend on Guava itself, posing several challenges (e.g., dependencies management) to the competition infrastructure. Those limitations should be investigated further to (partially) automate the comparison.

**Statistical analysis.** Similarly to previous editions [6], we used statistical tests to support the results: the Friedman test to assess whether the scores over the different CUTs and time budgets (70 CUTs × 2 budgets = 140 data points) achieved by alternative tools differ significantly from each other; and the post-hoc ConoverâĂŹs test for pairwise multiple comparisons to determine for which pair of tools the significance actually holds. We used the confidence level $\alpha = 0.05$, and p-values obtained with the ConoverâĂŹs test were further adjusted with the Holm-Bonferroni procedure, which is required in case of multiple comparisons.

## 5 RESULTS

Figure 2 presents the results of the coverage and mutation analysis for EvoSuite and Randoop for the two time budgets considered. Mutants coverage denotes mutated statements that could be covered by at least one test, while the mutation score is the classical ratio between mutants that were killed by at least one test to the total number of mutants.

Out of 700 executions (70 CUTs × 10 executions), EvoSuite achieved, on average, a higher coverage and mutation score for all the projects compared to Randoop. The average coverage and mutation score of the tests generated by EvoSuite slightly increase (from 55.9% to 57.0% for line coverage, 50.8% to 51.7% for branch coverage, from 32.6% to 33.8% for the mutation score) when increasing the time budget from one to three minutes. For Randoop, the average coverage and mutation score of the test cases remain stable (from 42.4% to 42.7% for line coverage, from 32.6% to 33.1% for branch coverage, and from 30.9% to 31.1% for the mutation score).

Table 2 presents the final score and ranking achieved by the tools at different search budgets as well as the ranking produced by the Friedman test. The total score is split in (resp.) 199.73 (standard deviation 33.72) and 206.40 (standard deviation 45.64) for EvoSuite, and (resp.) 152.77 (standard deviation 1.55) and 157.98 (standard deviation 3.44) for Randoop, for time budgets of (resp.) one and three minutes. Figure 3 presents the detailed score achieved by the generated tests for each execution of EvoSuite and Randoop. The p-values produced by the post-hoc Conover's procedure is lower than

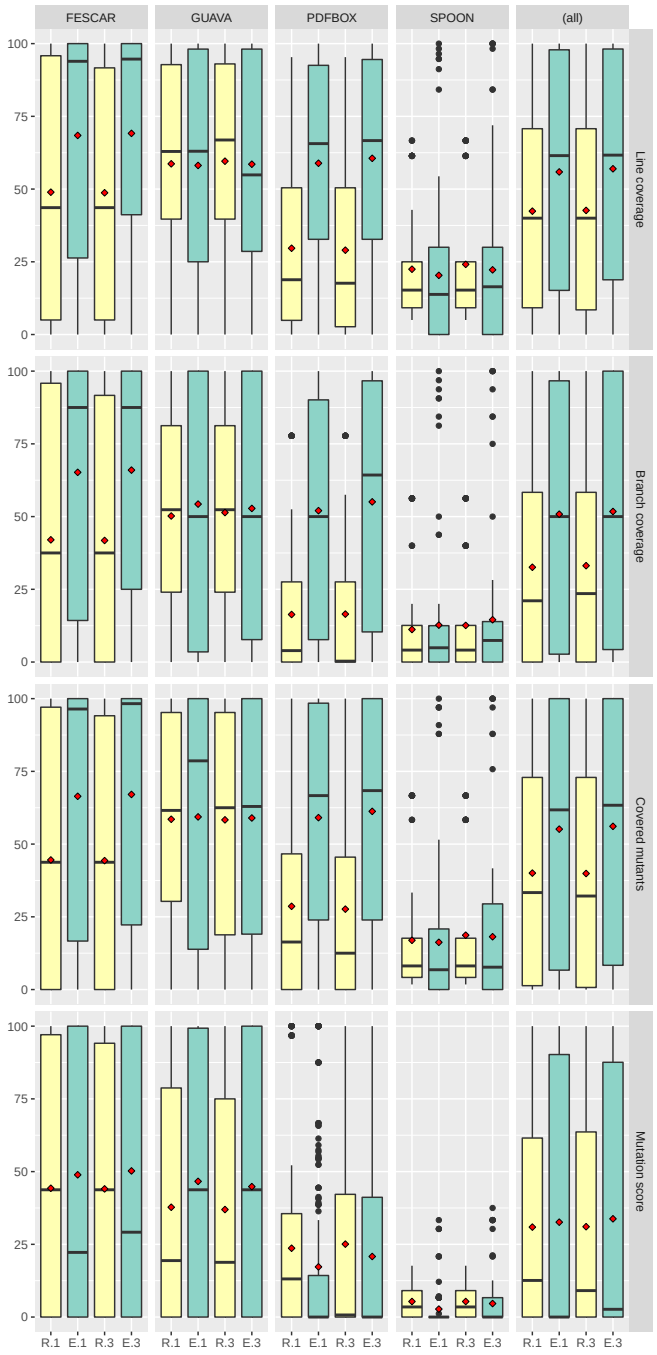


**Figure 2: Line, branch and mutants coverage, and mutation score for EvoSuite (E.) and Randoop (R.) with a time budget of one (1) and three (3) minutes on the different CUTs, grouped by project.**

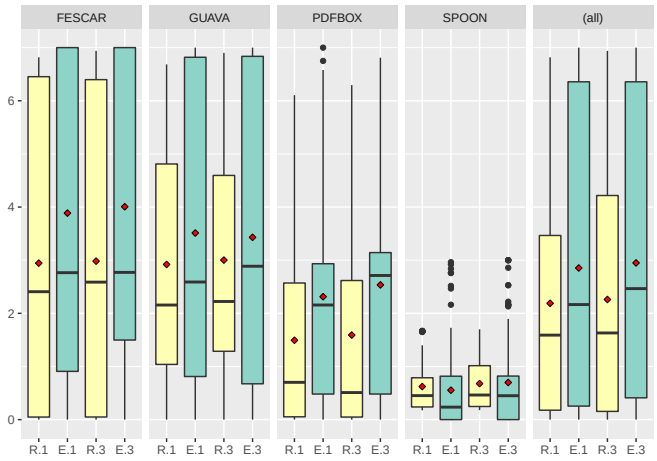


**Figure 3: Scores achieved by EvoSuite (E.) and Randoop (R.) with a time budget of one (1) and three (3) minutes on the different CUTs, grouped by project.**

0.006. Those results are consistent with other independent evaluations [4, 11], as well as previous results of the competitions [7], and show that bugs discovered both in the competition infrastructure [6] and in the EvoSuite implementation [3] could be fixed.

The full results are available in the contest infrastructure repository at https://github.com/JUnitContest/junitcontest/tree/master/publications.

## 6 CONCLUSIONS AND FINAL REMARKS

This year was the eighth edition of the Java unit testing tool competition. EvoSuite was improved compared to last year and showed several improvements in the results.

Among the several improvements made to the competition infrastructure, we parallelized the execution of the mutation analysis for the different CUTs. We also corrected several bugs reported during the seventh edition of the competition, leading to an improvement in the results of the various tools. Finally, we revised the procedure to update the readily usable Docker image. 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. Unfortunately, the configuration (and validation of the configuration) of the competition infrastructure remains mainly a manual process and should be partially automated in the future.

The dockerized version of the infrastructure allowed us to distribute the execution on two different servers. Of course, this could have side effects on the performance of the individual tools running on different hardware. However, the impact could be toned down by limiting the amount of resources used by the executions. Future editions of the competition should include those limits in the Docker image configuration file to bring reproducibility of the results one step further in future evaluations.

Future directions for future unit testing tool competitions go toward (i) the comparison of tools by considering additional criteria than the coverage and mutation analysis [5]; (ii) exploring the possibility to consider other languages (e.g., Python) in further competitions; (iii) extend even further the dockerized version of the infrastructure, making it available as service to researchers of the community.

## ACKNOWLEDGMENTS

Special thanks to Mitchell Olsthoorn for his help with the setup of the server. This research was partially funded by the EU Horizon 2020 ICT-10-2016-RIA âĂIJSTAMPâĂİ project (No.731529). 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] Andrea Arcuri and Lionel Briand. 2014. A HitchhikerâĂŹs Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering. *Softw. Test. Verif. Reliab.* 24, 3 (May 2014), 219âĂŞ250. https://doi.org/10.1002/stvr.1486

[2] Andrea Arcuri, Jose Campos, and Gordon Fraser. 2016. Unit Test Generation During Software Development: EvoSuite Plugins for Maven, IntelliJ and Jenkins. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 401–408. https://doi.org/10.1109/ICST.2016.44

[3] Jose Campos, Annibale Panichella, and Gordon Fraser. 2019. EvoSuite at the SBST 2019 Tool Competition. In *2019 IEEE/ACM 12th International Workshop on Search-Based Software Testing (SBST)*. IEEE, 29–32. https://doi.org/10.1109/SBST.2019.00017

[4] 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

[5] 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.

[6] 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

[7] Urko Rueda Molina, Fitsum Kifetew, and Annibale Panichella. 2018. Java unit testing tool competition. In *Proceedings of the 11th International Workshop on Search-Based Software Testing - SBST '18*. ACM Press, 22–29. https://doi.org/10.1145/3194718.3194728

[8] 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

[9] Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball. 2007. Feedback-Directed Random Test Generation. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 75–84. https://doi.org/10.1109/ICSE.2007.37

[10] Annibale Panichella, JosÃĬ Campos, and Gordon Fraser. 2020. EvoSuite at the SBST 2020 Tool Competition. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20)*. Seoul, Republic of Korea. https://doi.org//10.1145/3387940.3392266

[11] 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