Java Unit Testing Tool Competition - Seventh Round

Kifetew, Fitsum ; Devroey, Xavier; Rueda, Urko

DOI
10.1109/SBST.2019.00014

Publication date
2019

Document Version
Accepted author manuscript

Published in
2019 IEEE/ACM 12th International Workshop on Search-Based Software Testing (SBST)

Java Unit Testing Tool Competition - Seventh Round

Fitsum Kifetew
Fondazione Bruno Kessler
Trento, Italy
kifetew@fbk.eu

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

Urko Rueda
Research Center on Software Production Methods
Universitat Politècnica de València
Valencia, Spain
urueda@pros.upv.es

***Abstract*—We report on the results of the seventh edition of the JUnit tool competition. This year, four tools were executed on a benchmark with (i) new classes, selected from real-world software projects, and (ii) challenging classes from the previous edition. We use Randoop and manual test suites from the projects as baselines. Given the interesting findings of last year, we analyzed the effectiveness of the combined test suites generated by all competing tools and compared; results are confronted with the manual test suites of the projects, as well as those generated by the competing tools . This paper describes our methodology and the results, highlight challenges faced during the contest.**

***Index Terms*—tool competition, benchmark, mutation testing, automation, unit testing, Java, statistical analysis, combined performance**

I. Introduction

After a successful sixth edition in 2018 [11], this year, we celebrate the seventh edition of the Java unit testing tool competition, involving five tools (two of them introduced this year) and a fresh set of classes under test (CUTs). The participant tools are: EvoSuite [1], T3 [9], [10], Sushi [3], [4], Tardis [2], and Randoop [7].

While Sushi and Tardis are new entries to this year's edition, EvoSuite and T3 were participants in previous competitions but have been since improved by the respective authors. For Randoop, used as a baseline, we have updated it to the latest version (version 4.1.1). Furthermore, following last year's initiative, we performed a *combined analysis* in which we constructed test suites by putting together all the tests generated, for a particular CUT, by all participating tools. Last year, the combined analysis gave better performance than any single tool [11], indicating that the different tools were complementary in the areas of the solution space they covered. This year, we also combined the test suites generated by all tools and analyze the resulting performance. Finally, we compared the results achieved by the tools against manually-written test suites included in the original projects from which our CUTS were extracted.

For the comparison, we used well-established *structural coverage* metrics, namely *statement* and *branch* coverage, which we computed by using JaCoCo[1]. Additionally, we applied *mutation analysis* to assess the fault revealing potentials of the test suites generated by the tools. We used PITest[2] to compute the mutation scores of the various test suites (either automatically generated or manually-written).

We used similar time budgets as last year [11]: 10, 60, 120, and 240 seconds. Such a range of time budgets allows us to assess the capabilities of the tools in different usage scenarios, while staying within the tight time period and computational resources available for running the experiments.

The rest of this report is organized as follows. Section II describes the benchmark, Section III describes participating tools and baselines. Section IV presents the methodology while Section V presents the results. Concluding remarks are given in Section VI.

II. The benchmark Subjects

Benchmark preparation should ideally take into consideration several factors. The benchmark should be a representative sample of real-world software [5]; preferably be open-source and cover different application domains [5]; the classes should not be trivial [8] (e.g., classes with only branchless methods) and should handle different types of input. Taking these aspects into account, we focused on GitHub repositories that satisfy the following criteria: (i) can be built using Maven, and (ii) contains JUnit 4 test suites. We also included CUTs from last year's edition which proved to be challenging for the competing tools. As a result, we selected the following new projects:

- *Antlr4*[3]: is a tool able to generate lexical analyzers and parsers for structured text or binary files written with a given grammar. For the competition, we focused on the *antlr* and *antlr-runtime* modules.
- *AuthzForce*[4]: is an Attribute-Based Access Control (ABAC) framework compliant with OASIS XACML 3.0. For the competition, we focused on the *authzforce-ce-core* module.
- *Fescar*[5]: is a distributed transaction framework for microservices architecture.

[1]https://www.jacoco.org/jacoco

[2]http://pitest.org/

[3]https://github.com/antlr/antlr4

[4]https://github.com/authzforce/core

[5]https://github.com/alibaba/fescar/

TABLE I
CHARACTERISTICS OF THE BENCHMARK. UPPER HALF: SUBJECTS NEW THIS YEAR; LOWER HALF: SUBJECTS KEPT FROM LAST YEAR

Project	#CUTs	10s	4m	# Sampled CUTs
Antlr4	370	61.6m	24.6h	20
AuthzForce	70	11.6m	4.6h	10
Fescar	75	12.5m	5.0h	10
Imixs-Workflow	83	13.8m	5.5h	20
Spoon	437	72.8m	29.1h	10
Dubbo	235	39.1m	15.7h	1
FastJason	217	36.2m	14.5h	2
Okio	44	7.3m	2.9h	2
Webmagic	162	27.0m	10.8h	2
Zxing	268	44.7m	17.9h	1

- *Imixs-Workflow*[6]: is an open source workflow engine. For the competition, we focused on the *imixs-workflow-core* and *imixs-workflow-engine* modules.
- *Spoon*[7]: is a library for analyzing and transforming Java source code.

Table I summaries the main characteristics of the selected projects. Antlr4, AuthzForce, Fescar, Imixs-Workflow, and Spoon are newly added this year, while the remaining come from last year [11]. The total number of CUTs in each project ranges between 44 (Okio) and 437 (Spoon) classes. Since considering all CUTs in each project is infeasible due to the extremely large amount of time and resources the competition would require, following the experience of previous editions, we sampled a small number of CUTs from each project as reported in Table I. Basically we sampled 10 CUTs from each new project/module and kept 8 of the challenging CUTs from last year's edition, for a total of 78 CUTs. For the selection, we followed a similar procedure as in the previous editions [11]. First, we computed McCabe's cyclomatic complexity for all methods and classes in each project using JavaNCSS[8]. Then, we removed classes that contain only methods with a complexity lower than three. This filter reduces the chance of sampling very trivial classes with either no branches or that can be fully covered with few randomly generated tests [8].

As a further filtering mechanism, we generated test cases using the previous version of Randoop on all candidate CUTs with a time budget of 10 seconds. This is an additional step we took to ensure that the CUTs are not too trivial. We ordered the filtered CUTs by (i) the number of branches (descending), (ii) the number of branches covered by Randoop (ascending), (iii) the number of lines (descending), and (iv) the cyclomatic complexity (descending). Finally we picked the top 10 CUTs from each of the 7 projects/modules, resulting in 70 CUTs. Furthermore, we picked the most challenging eight classes from previous year's benchmarks. This resulted in 78 Java classes[9], whose number of branches ranges between 4 and 512, while number of lines ranges between 10 and 820, and number of mutants generated by PIT ranges between 4 and 316.

This year, however, we discovered several issues with the new CUTs when computing metrics (coverage and mutation score) after test generation. In particular, the library we use in our contest infrastructure for computing code coverage (JaCoCo) had difficulty in measuring coverage for several of the CUTs in the benchmark. We upgraded JaCoCo from version 0.6.3 to 0.8.3 (the latest version) and re-executed metrics computation on all test cases generated by all tools. This fixed some of the issues but still metrics computation failed for several classes. Consequently, we were forced to drop CUTs from four of the projects/modules (*antlr*, *antlr-runtime*, *imixs-workflow-core*, *imixs-workflow-engine*) from the benchmark. The final number of CUTs for which execution was completed successfully were 38 CUTs.

[6]https://github.com/imixs/imixs-workflow

[7]https://github.com/INRIA/spoon/

[8]https://github.com/codehaus/javancss

[9]https://github.com/PROSRESEARCHCENTER/junitcontest/tree/master/bin/benchmarks_7th

TABLE II
SUMMARY OF TOOLS

Tool	Technique	Static analysis
EvoSuite [1], [8]	Evolutionary algorithm	yes
Sushi [3], [4]	Evolutionary + Symbolic executon	yes
Tardis [2]	Evolutionary + Concolic testing	yes
T3 [9], [10]	Random testing	no
Randoop [7]	Random testing	no

III. THE TOOLS

A total of five tools are considered in this year's edition: EvoSuite, Randoop, Sushi, Tardis, and T3 (see Table II). EvoSuite and T3 have also participated in previous editions of the contest, however for the current edition, besides bug fixes, they have introduced important improvements to their respective test generation mechanisms. Sushi and Tardis are new tools participating in the contest for the first time this year. Randoop is used as a baseline and has been updated to its latest version (4.1.1).

As shown in Table II, EvoSuite, Sushi, and Tardis use evolutionary algorithms for evolving test suites. However, Sushi and Tardis further exploit (dynamic) symbolic execution to enhance the test generation process [3]. On the other hand, T3 and Randoop follow a random testing strategy.

Baselines: As baseline, we use tests generated by Randoop, as well as manually written test suites of the CUTs available from their respective projects. It should be noted that, we use manual tests as baseline to give an idea of how automatically generated test suites fair with respect to human written tests. It is, however, difficult to draw direct parallels between the two as manual test suites are typically evolved and improved overtime, and it is hard to estimate how much (human) effort has been spent in writing the test suites.

IV. CONTEST METHODOLOGY

The methodology adopted in this year's edition is mostly similar to that of last year's [11]. Here we highlight the main

changes introduced this year:

→ **Public contest repository**[10]**.** The full contest infrastructure was already published on GitHub as an open source project in the previous editions. In this year's edition, to further minimize the effort and time required to configure the infrastructure, we packaged the contest infrastructure as a docker container so that participants could easily test their tools. We hope that making the infrastructure open and easily accessible will contribute to keeping the contest alive and attract new participants. Furthermore, all the resources of the previous two editions were published in the GitHub repository, including benchmarks, reports, detailed data, etc. Details of this year's edition will also be made available in the online repository.

→ **CUTs**. We selected 78 CUTs for the benchmark (70 new, 8 from last year) as described in Section II, however the final number of CUTs was 38 due to problems faced during metrics computation.

→ **Execution frame**. A total of 4560 executions were performed (5664 in the previous edition): 38 CUTs x 5 tools x 4 time budgets x 6 repetitions for statistical analyses. Similar to last year, the executions were run in a cluster environment running Sun Grid Engine (SGE). We have used five physical nodes, each with 12 CPU cores (Intel(R) Xeon(R) CPU E5-2440 @ 2.40GHz) and 64GB RAM. In contrast to previous editions in which all tools were run in parallel, in this edition, each tool was run on a dedicated physical node for all time budgets and repetitions.

→ **Test generation**. Similar to the previous edition, for each tool, the test generation was repeated a total of 6 times, to account for the inherent randomness of the generation processes. Unlike the previous edition where the execution sporadically hang during tests generation, this year no such problem was observed.

→ **Metrics computation**. We kept the strict mutation analysis time budget of 5 minutes per CUT, and a timeout of 1 minute for each mutant. Moreover, we sampled the mutants generated by PITest: 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 had difficulty in computing metrics for several CUTs in our benchmark, in particular CUTs from *antlr*, *antlr-runtime*, *imixs-workflow-core*, and *imixs-workflow-engine*. The problems were caused by the coverage computation using JaCoCo. Updating JaCoCo to its latest version could not solve the issues. Since our mutation analysis depends on coverage results computed by JaCoCo, the failures in coverage computation also led to failures in mutation analysis. Eventually we had to exclude 40 CUTs from our benchmark due to these difficulties. The test generation phase, however, was completed successfully, and all test cases generated have been communicated to the respective tool authors for eventual analysis. Due to the limited amount of time between tool submission and results notification, we could not perform the metrics computation with a different tool for coverage analysis. For the next edition of the contest, based on the data collected this year, we will help the new organizers to investigate and correct the issues.

→ **Combined analyses**. We performed the combined analysis by putting together all the tests generated by all tools for a given CUT and time budget. Metrics computation was performed on the combined test suite in the same way as for the individual tools, however the computational cost increases to the sum of the costs required to evaluate the test suites generated by the individual tool. Due the high computational costs for the combined analyses, we were only able to do it for budgets of 10 and 240 seconds.

→ **Time budgets**. Similar to the previous edition [11], we considered four search budgets: 10, 60, 120 and 240 seconds.

→ **Statistical Analysis**. We used statistical tests to support the results. First, we used the Friedman test to compare the scores achieved by the different tools over the different CUTs and time budgets. In total, each tool produced (38 CUTs $\times$ 4 budgets) = 152 data points, corresponding to the average scores achieved across six independent repetitions. Second, we applied the post-hoc Conover's test for pairwise multiple comparisons. While the former test allows us to assess whether the scores achieved by alternative tools differ statistically significantly from each other, the latter test is used to determine for which pair of tools the significance actually holds.

Note that for all aforementioned statistical tests, we used the confidence level α=0.05; p-values obtained with the Conover's test were further adjusted with the Holm-Bonferroni procedure, which is required in case of multiple comparisons.

A. Threats to Validity

Conclusion validity. As in previous editions, we perform statistical analyses for significance. The number of repetitions (6 runs) could be considered low for rigorous statistical analysis. The could be improved in future editions by either allocating more computational resource or increasing the time span available for the experiments. Effects of multiple comparison are mitigated by adjusting p-values via Holm-Bonferroni.

Internal validity. The contest infrastructure has been improved over the last seven years. It is released as an open-source project to foster improvement and bug fixes from the community. There could be failures in computing metrics for some tools and CUTs due to issues in the interaction between the contest infrastructure and libraries (e.g., JaCoCo or PITest). The benchmarks used in the competition were hidden from the participants, but they were able to test their tools via docker using some of the CUTs from past editions.

Construct validity. The scoring formula used to rank the tools —which is identical to the past edition— assigns a higher weight to the mutation coverage. Also, we apply a time window of 1 minute per mutant and a global timeout of 5 minutes per CUT, as well as a random sampling of the mutants to reduce the costs of metrics computation. Note that the set of sampled mutants for each CUT is kept the same for all tools and search budgets.

[10]https://github.com/PROSRESEARCHCENTER/junitcontest

TABLE III
AVERAGE (MEAN) COVERAGE METRICS AND OVERALL (SUM) SCORES OBTAINED ACROSS ALL CUTS.

Tool	Budget (in sec.)	cov_i			cov_b			cov_m			Score	Std.dev
		Min	Mean	Max	Min	Mean	Max	Min	Mean	Max		
evosuite	10	0.00	0.31	1.00	0.00	0.25	1.00	0.00	0.29	1.00	41.42	7.93
randoop		0.00	0.25	0.70	0.00	0.16	0.83	0.00	0.20	0.90	35.58	0.40
sushi		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
tardis		0.00	0.01	0.18	0.00	0.00	0.04	0.00	0.00	0.15	0.63	0.55
t3		0.00	0.35	0.89	0.00	0.28	1.00	0.00	0.02	0.34	34.48	2.04
evosuite	60	0.00	0.34	1.00	0.00	0.30	0.96	0.00	0.29	1.00	78.75	16.21
randoop		0.00	0.24	0.70	0.00	0.15	0.83	0.00	0.20	0.90	37.99	0.36
sushi		0.00	0.07	0.61	0.00	0.02	0.32	0.00	0.06	0.60	12.33	2.81
tardis		0.00	0.12	0.51	0.00	0.08	0.49	0.00	0.11	0.77	25.01	7.39
t3		0.00	0.35	0.84	0.00	0.28	1.00	0.00	0.02	0.29	36.17	3.16
evosuite	120	0.00	0.28	1.00	0.00	0.26	0.97	0.00	0.24	1.00	65.89	17.67
randoop		0.00	0.24	0.70	0.00	0.15	0.83	0.00	0.21	0.90	39.37	0.55
sushi		0.00	0.07	0.62	0.00	0.03	0.33	0.00	0.07	0.60	13.52	2.42
tardis		0.00	0.09	0.52	0.00	0.06	0.49	0.00	0.09	0.80	19.82	4.47
t3		0.00	0.36	0.89	0.00	0.28	1.00	0.00	0.02	0.34	37.49	2.85
evosuite	240	0.00	0.30	1.00	0.00	0.27	0.99	0.00	0.25	1.00	69.37	18.48
randoop		0.00	0.24	0.70	0.00	0.16	0.83	0.00	0.21	0.90	41.46	0.41
sushi		0.00	0.08	0.61	0.00	0.03	0.26	0.00	0.07	0.56	13.99	4.73
tardis		0.00	0.10	0.51	0.00	0.06	0.49	0.00	0.09	0.80	21.35	7.05
t3		0.00	0.35	0.85	0.00	0.29	1.00	0.00	0.02	0.34	37.14	3.85

External validity. To mitigate the effects on the low number of subjects in the benchmark, we attempted to increase the benchmark size to 78 this year. However, due to technical challenges with JaCoCo, we had to eventually eliminate much of the CUTs from the benchmark. Nonetheless, the number of active participant tools has increased compared to the previous edition. Future editions of the contest should work towards expanding the benchmark size, of course also considering the amount of time for computing the results and number of tools involved.

V. CONTEST RESULTS

Detailed results for all tools, as well as manual and combined analysis, for all budgets, averaged over six repetitions, can be found in an online appendix [6]. Table III summarizes the average (mean) instruction coverage (cov_i), branch coverage (cov_b), and strong mutation coverage (cov_m) achieved across all CUTs over the four search budgets. Table III also shows the overall scores, which are computed as the sum of the average scores achieved across all CUTs and for each search budget, separately. As expected, the maximum coverage metrics and the scores tend to increase as the time given for test generation increases. However, the average coverage does not appear to be much affected by the time budget. This could be due to the fact that with increased budget the tools tend to generate too many tests, eventually increasing the possibility of timeouts and potential problems with coverage/mutation computation.

Comparison with manual and combined suites. We present in Table VI summary of results for all tools as well as results of the combined analysis and manual test suites, for all 38 CUTs in the benchmark. Figure 1 shows a plot of the performances. Due to lack of space, Table VI reports results only for the 10 second budget, the full details can be found in the online technical report [6]. As can be seen from Table VI, for some of the CUTs the manual suites has a very low coverage (FESCAR, AUTHZFORCE, and WEBMAGIC), while for others (SPOON) the manual suites achieve a higher coverage than the combined ones.

TABLE IV
OVERALL SCORES AND RANKINGS OBTAINED WITH THE FRIEDMAN TEST

Tool	Score	Ranking
t3	145.27	2.30
evosuite	255.43	2.38
randoop	154.34	2.51
tardis	66.80	3.73
sushi	39.84	4.09

TABLE V
PAIRWISE COMPARISON ACCORDING TO THE POST-HOC CONOVER'S TEST

	evosuite	randoop	sushi	t3
randoop	0.83	-	-	-
sushi	8.08e-19	4.38e-18	-	-
t3	0.14	0.11	1.31e-26	-
tardis	1.9e-13	7.57e-13	0.22	3.32e-20

Confirming previous observations [11], combined analysis was able to achieve better results compared to the individual tools. However, this year, combined analysis did not always outperform manual test suites.

We also observe from Table III and Table VI that `sushi` and `tardis` were not able to achieve good coverage for the small time budgets. Eventually the results for these two tools improved as the budget increased. However compared to the other tools they generally achieved low coverage. This could be attributed to the fact that the techniques employed in these tools perform complex symbolic analyses and execution, besides search-based algorithms [3]. Given a search budget grater than 240 seconds, the tools could have generated more effective test cases. Unfortunately, this was not possible in the context of the contest, giving the limited amount of time available for running the experiments. To assess this hypothesis, we executed these two tools with a budget of 480 seconds[11], and results confirm our hypothesis, but the improvements were still marginal. In future contests, we encourage the tool authors to optimize the efficiency of the tools in using the allocated time budget, and we recommend future contest organizers to consider larger time budgets, if resources allows it.

Final scores and statistical results. Table IV presents the overall scores achieved by the tools at different search budgets as well as the ranking produced by the Friedman test. According to the test, some tools turn out to be statistically different in terms of scores (p-value as low as 10^{-26}), while other tools did not differ significantly (e.g., EvoSuite vs T3 or vs Randoop). Table V reports the p-values produced by the post-hoc Conover's procedure. Note the p-values are adjusted with the Holm-Bonferroni correction procedure as required in case of multiple pairwise comparisons.

[11]Since these two tools did not produce much tests, metrics computation finished sooner than for the other tools. We were able to run them with a larger time budget. For the other tools however we could not do it due to shortage of time.

TABLE VI

RESULTS FOR MANUAL, AVERAGED COMBINED, AND TOOLS ON 10S BUDGET

	manual			**combined 10s**			**t3 10s**			**evosuite 10s**			**sushi 10s**			**tardis 10s**		
CUT	cov_i	cov_b	cov_m	cov_i	cov_b	cov_m	cov_i	cov_b	cov_m	cov_i	cov_b	cov_m	cov_i	cov_b	cov_m	cov_i	cov_b	cov_m
AUTHZFORCE-11	83.0	80.0	28.0	24.5	10.0	11.9	21.0	10.0	0	14.0	0	4.7	0	0	0	0	0	0
AUTHZFORCE-1	0	0	0	0	0	0	73.7	100.0	0	0	0	0	0	0	0	0	0	0
AUTHZFORCE-27	40.0	50.0	75.0	90.0	75.0	75.0	70.0	25.0	0	90.0	75.0	75.0	0	0	0	0	0	0
AUTHZFORCE-32	49.0	63.0	54.0	31.1	20.0	23.0	26.6	20.0	0	13.3	0	7.6	0	0	0	0	0	0
AUTHZFORCE-33	0	0	0	76.5	100.0	90.9	74.8	94.4	0	37.2	55.5	57.5	0	0	0	0	0	0
AUTHZFORCE-48	0	0	0	90.0	63.6	88.2	89.4	62.8	0	0	0	0	0	0	0	0	0	0
AUTHZFORCE-52	0	0	0	62.0	91.6	100.0	62.0	83.3	0	62.0	91.6	100.0	0	0	0	0	0	0
AUTHZFORCE-5	0	0	0	51.1	59.0	60.7	43.1	53.7	0	22.3	20.4	27.3	0	0	0	0	0	0
AUTHZFORCE-63	0	0	0	50.0	35.0	38.8	42.5	30.0	0	27.5	10.0	5.5	0	0	0	0	0	0
AUTHZFORCE-65	0	0	0	60.0	41.6	0	60.0	41.6	0	0	0	0	0	0	0	0	0	0
DUBBO-2	38.0	32.0	42.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FASTJSON-1	8.0	6.0	5.0	.7	.6	.4	.6	.5	.4	0	0	0	0	0	0	0	0	0
FASTJSON-3	56.0	49.0	22.0	19.1	11.6	11.1	10.4	5.0	6.3	0	0	0	0	0	0	0	0	0
FESCAR-12	0	0	0	73.0	12.5	50.0	26.9	0	0	73.0	12.5	50.0	0	0	0	3.2	0	0
FESCAR-18	0	0	0	78.2	71.4	84.0	32.6	35.7	0	72.4	61.9	81.1	0	0	0	18.4	3.5	15.2
FESCAR-1	0	0	0	99.1	97.9	97.7	79.7	60.4	0	99.1	97.9	97.7	0	0	0	0	0	0
FESCAR-23	0	0	0	73.0	63.0	78.5	38.0	28.5	0	73.0	63.0	78.5	0	0	0	0	0	0
FESCAR-25	0	0	0	46.8	37.5	22.2	46.8	37.5	0	46.3	37.5	22.2	0	0	0	0	0	0
FESCAR-36	0	0	0	100.0	100.0	100.0	29.0	14.7	0	100.0	100.0	100.0	0	0	0	2.9	.6	1.8
FESCAR-37	0	0	0	69.3	54.4	71.8	51.0	35.2	0	69.3	52.9	71.8	0	0	0	0	0	0
FESCAR-41	0	0	0	1.7	2.0	2.3	.4	0	0	0	0	0	0	0	0	0	0	0
FESCAR-42	0	0	0	27.9	7.1	0	27.9	7.1	0	0	0	0	0	0	0	0	0	0
FESCAR-7	0	0	0	39.2	40.1	38.2	19.2	18.7	0	37.9	37.7	37.9	0	0	0	0	0	0
OKIO-1	83.0	76.0	3.0	69.3	56.3	6.8	32.5	22.9	8.4	26.0	18.2	8.8	0	0	0	0	0	0
OKIO-4	90.0	73.0	27.0	22.3	19.2	13.0	1.4	.6	.9	22.3	19.2	13.0	0	0	0	0	0	0
SPOON-105	84.0	79.0	71.0	17.9	17.6	23.5	17.8	17.3	0	0	0	0	0	0	0	0	0	0
SPOON-155	98.0	93.0	100.0	25.1	14.0	19.6	19.4	6.2	0	24.3	11.9	19.0	0	0	0	0	0	0
SPOON-169	97.0	89.0	68.0	11.6	6.9	10.2	5.0	3.2	0	0	0	0	0	0	0	0	0	0
SPOON-16	96.0	84.0	87.0	47.8	33.9	45.4	16.0	7.8	0	47.8	33.9	45.4	0	0	0	0	0	0
SPOON-20	91.0	70.0	75.0	49.2	33.3	40.2	42.8	20.0	0	49.2	33.3	40.2	0	0	0	0	0	0
SPOON-211	98.0	94.0	86.0	23.7	16.2	34.7	20.3	15.3	0	19.4	8.1	27.7	0	0	0	0	0	0
SPOON-253	97.0	87.0	94.0	62.2	57.8	68.1	60.5	54.6	0	31.5	29.6	34.8	0	0	0	0	0	0
SPOON-25	60.0	63.0	57.0	28.7	21.4	23.6	28.5	21.2	0	0	0	0	0	0	0	0	0	0
SPOON-32	88.0	87.0	92.0	20.8	5.2	5.5	30.0	12.5	0	5.0	2.0	1.1	0	0	0	0	0	0
SPOON-65	99.0	99.0	100.0	40.6	50.5	1.7	40.6	50.0	0	3.0	1.5	.1	0	0	0	0	0	0
WEBMAGIC-1	0	0	0	39.2	27.7	50.1	0	0	0	32.8	23.3	42.0	0	0	0	0	0	0
WEBMAGIC-4	0	0	0	69.2	9.0	59.1	12.6	2.0	9.7	0	0	0	0	0	0	0	0	0
ZXING-10	91.0	83.0	21.0	90.5	77.4	23.1	66.7	62.0	33.8	88.3	70.1	62.9	0	0	0	0	0	0

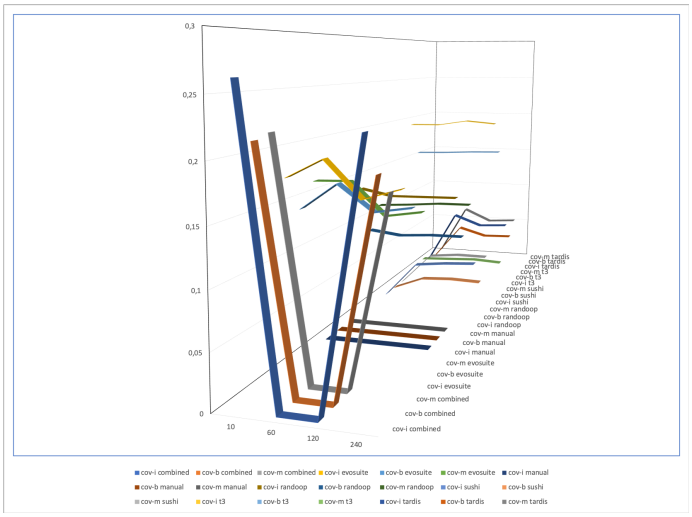

Fig. 1. Performance of tools, manual suites, and combined suites

VI. Concluding remarks

This year's contest was successful in that four tools actively participated, and `evosuite` and `t3` where improved, with respect to the previous edition, by their respective authors. The two new tools, `tardsi` and `sushi` brought in diversity as they rely on symbolic execution. `Randoop` was also updated to its latest version.

Two of the projects form which CUTs were extracted (*Antlr4* and *Imixs-Workflow*) proved to be problematic during coverage computation, despite upgrading JaCoCo. To minimize such issues, it would help if future editions could start promoting the contest early and collect participating tools; this would give enough time to handle potential issues that could arise. We also observed some discrepancies between the coverage results reported by our contest infrastructure and results reported by the coverage tools when executed independently, on the same test suite. Issues were reported by the author of `t3` in which for some of the CUTs, given the same test suite generated during the contest, different coverage results were obtained from the contest infrastructure and independent measurement by the author using the same library (JaCoCo/PIT). Such issues could arise from the interaction between the contest infrastructure and the coverage libraries. Using data from this year's edition those issues will be investigated and resolved by the organizers of upcoming editions and other external contributors.

Similarly to the previous edition, the combined analysis showed better results. This year we executed the combined analysis for two time budgets (i.e., 10 and 240 sec), next editions of the contest could target all time budgets.

Another interesting observation is that `Randoop`, which we included as a baseline, showed quite good performance and resilience to the difficulties that seem to have affected the other tools.

The fact that the contest infrastructure is dockerized proved to be quite effective in reducing the effort required to set up and execute a given tool in the contest setting. We could not, however, run the full experiments using docker due to the fact that docker is not available in the cluster environment where we run all the experiments. This created extra effort in maintaining the two versions synchronized. In future editions, if docker could be used also for the full experiments, it could save valuable time for the organizers.

Acknowledgments

Special thanks to Matteo Biagiola who helped in porting the contest infrastructure to a docker image. This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO) under the project DataME (TIN2016-80811-P); by the Italian Ministry of Education, University, and Research (MIUR) with the PRIN project GAUSS (grant n. 2015KWREMX); and by the EU Project STAMP ICT-16-10 No.731529 and the NIRICT 3TU.BSR (Big Software on the Run) project.

References

[1] A. Arcuri, J. Campos, and G. Fraser. Unit test generation during software development: Evosuite plugins for Maven, IntelliJ and Jenkins. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 401–408. IEEE Computer Society, 2016.

[2] P. Braione. Tardis concolic test case generator, 2019. Available at: https://github.com/pietrobraione/tardis.

[3] P. Braione, G. Denaro, A. Mattavelli, and M. Pezzè. Combining symbolic execution and search-based testing for programs with complex heap inputs. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2017, pages 90–101, New York, NY, USA, 2017. ACM.

[4] P. Braione, G. Denaro, A. Mattavelli, and M. Pezzè. SUSHI: A test generator for programs with complex structured inputs. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ICSE '18, pages 21–24, New York, NY, USA, 2018. ACM.

[5] G. Fraser and A. Arcuri. A large scale evaluation of automated unit test generation using evosuite. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 24(2):8, 2014.

[6] F. Kifetew, X. Devroey, and U. R. Molina. Java unit testing tool competition - seventh round. Technical report, 2019. Available at: https://github.com/PROSRESEARCHCENTER/junitcontest/blob/master/publications/SBSTcontest2019_detailed_results.pdf.

[7] C. Pacheco and M. D. Ernst. 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, pages 815–816, New York, NY, USA, 2007. ACM.

[8] A. Panichella, F. M. Kifetew, and P. Tonella. Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. *IEEE Transactions on Software Engineering*, 44(2):122–158, Feb 2018.

[9] I. Prasetya. T3, a combinator-based random testing tool for Java: Benchmarking. *Int. Workshop Future Internet Testing, Lecture Notes in Computer Science*, 8432, 2014.

[10] I. Prasetya. T3i: A tool for generating and querying test suites for java. In *10th Joint Meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*. ACM, 2015.

[11] U. Rueda Molina, F. Kifetew, and A. Panichella. Java unit testing tool competition: Sixth round. In *Proceedings of the 11th International Workshop on Search-Based Software Testing*, SBST '18, pages 22–29, New York, NY, USA, 2018. ACM.