EvoSuite at the SBST 2020 Tool Competition

Annibale Panichella
Delft University of Technology
The Netherlands
a.panichella@tudelft.nl

José Campos
LASIGE, Faculdade de Ciências
Universidade de Lisboa
Lisboa, Portugal
jcmcampos@fc.ul.pt

Gordon Fraser
Chair of Software Engineering II,
University of Passau
Passau, Germany
gordon.fraser@uni-passau.de

ABSTRACT

EvoSuite is a search-based tool that automatically generates executable unit tests for Java code (JUnit tests). This paper summarizes the results and experiences of EvoSuite's participation at the eighth unit testing competition at SBST 2020, where EvoSuite achieved the highest overall score (406.14 points) for the seventh time in eight editions of the competition.

ACM Reference Format:
Annibale Panichella, José Campos, and Gordon Fraser. 2018. EvoSuite at the SBST 2020 Tool Competition. In *SBST'20: IEEE/ACM 13th International Workshop on Search-Based Software Testing, October 2020, Seoul, South Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Automated unit test generation can support developers and testers, produce regression test suites, and is an enabler for dynamic program analyses. The annual unit test generation aims to foster research and development of automated unit test generators. This paper describes the results obtained by the EvoSuite test generation tool [7] in this competition. EvoSuite uses meta-heuristic search to evolve unit test suites with high coverage, and automatically produces regression oracles in the form of test assertions. In the 8th instance of the competition at the International Workshop on Search-Based Software Testing (SBST) 2020, EvoSuite achieved an overall score of 406.14, which was the highest among the competing and baseline tools. This paper describes the results obtained by the EvoSuite test generation tool in this competition.

2 EVOSUITE

EvoSuite [7] is a search-based unit test generation tool [11]. Table 1 summarizes the features of EvoSuite in the standard format of the SBST unit testing competition: Given just the Java classpath containing all compiled dependencies and the name of a class under test, EvoSuite automatically generates a set of JUnit test cases aimed at maximizing code coverage. EvoSuite can be used on the command line, as a Maven plugin, or using plugins for the Eclipse and IntelliJ development environments [2].

Table 1: 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 4 test cases
Operation	
Interaction	Through the command line, and plugins for IntelliJ, Maven and Eclipse
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 / many-objective optimization
Obtaining the tool and information	
License	Lesser GPL V.3
Cost	Open source
Support	None
Does there exist empirical evidence about	
Effectiveness and Scalability	See [11, 12]

Throughout its development, several different search algorithms have been evaluated, starting with a basic genetic algorithm. EvoSuite's encoding of test cases for the evolutionary search (i.e., its chromosomes) consists of variable-length sequences of Java statements (e.g., primitive statements and calls on the class under test). The usual search operators used in evolutionary search (e.g., selection, crossover, mutation) are adapted for this particular representation. In the original approach, individuals of the genetic algorithm were whole test suites, with the optimization goal of finding a test suite that maximizes code coverage. This was later improved by adding an archive of solutions [26] to keep the search focused on uncovered goals, iteratively discarding covered goals and storing the tests that covered them.

SBST '20, October 2020, Seoul, South Korea

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

Recently, the *Dynamic Many-Objective Sorting Algorithm* (DynaMOSA) search algorithm [21–23] has been shown to be the most effective approach of all the algorithms evaluated so far. DynaMOSA is a many-objective algorithm, where individuals of the search population are test cases, rather than test suites, and the optimization is driven by a collection of individual fitness functions, one for each coverage objective (e.g., line or branch).

The fitness functions in EvoSuite are based on traditional heuristics for code coverage, such as the branch distance and the approach level (see [11] for more details). EvoSuite supports multiple different coverage criteria, which can be optimized at the same time. The default configuration combines branch coverage with mutation testing [13] and other basic criteria [24]. To cope with the potentially large number of coverage objectives, DynaMOSA prioritizes them during the search according to their structural dependencies in the control dependency graph. Initially, the search focuses on coverage objectives positioned higher in the hierarchy, and the remaining objectives are incrementally inserted in later generations when their dominator requirements are covered.

After the search has used up the available search budget (or alternatively, has achieved 100% coverage of all coverage objectives), EvoSuite applies various post-search optimizations aimed to improve the readability of the generated tests [7, 10], such as minimization and addition of test assertions using mutation analysis [17]. It also checks all generated tests for compile errors (which may be the result of bugs in EvoSuite) or flakiness caused by nondeterminism in the class under test, not covered by EvoSuite's extensive instrumentation.

EvoSuite has been evaluated on open source as well as industrial software in terms of code coverage [6, 12, 21, 26], fault-finding effectiveness [1, 28], and effects on developer productivity [16, 25] and software maintenance [29]. EvoSuite has a longstanding record of success at the unit testing tool competition, having ranked second in the third edition of the competition [14] and first in all the other editions [8, 9, 15, 18, 19]. In the 2019 SBST contest, EvoSuite achieved the highest overall score of all participating tools [20], although some bugs inhibited its performance.

3 TOOL SETUP

Similar to the previous years, the configuration of EvoSuite for the 2020 competition is largely based on its default values since these have been tuned extensively [3]. The search algorithm used was DynaMOSA [21], optimizing for the default set of coverage criteria [24] (i.e., line coverage, branch coverage, branch coverage by direct method invocations, weak mutation testing, output coverage, exception coverage). Other features enabled by default include the use of frequency-based weighted constants for seeding [27] as well as support for Java Enterprise Edition features [4]. In the case of difficult dependencies and branches that cannot be covered, EvoSuite can start using mock objects once a certain percentage of the search budget has passed [5] .

No major new features have been introduced in EvoSuite since the 2019 competition, but several bugs that affected EvoSuite's performance during the previous competition have been fixed. In particular, there were several problematic cases when EvoSuite was run using very small search budgets.

The test minimization step used by EvoSuite is computationally expensive and sometimes omitted for empirical studies; in the competition, we always enable the post-processing step of test minimization, because minimized tests are less likely to break or expose flakiness. However, we aimed to reduce the post-processing time by including all regression assertions rather than filtering them with mutation analysis [17]. While this makes test cases less readable and potentially more brittle with respect to future changes in the software under test, neither of these aspects is evaluated as part of the SBST contest.

EvoSuite uses different phases (e.g., initialization, search, minimization, assertion generation, compilation check, removal of flaky tests). Like in previous competitions (e.g., [15]), we allocated 50% of the overall time set by the competition organizers for the search, and distributed the other 50% equally to the remaining phases.

4 BENCHMARK RESULTS

Table 2 summarizes the results achieved by EvoSuite on the competition classes and search budgets. In general, the performance of EvoSuite was in line with previous results. As usual, there are some notable cases where EvoSuite did not perform well. In the following, we discuss these cases, such that future work can address open problems in automated unit test generation.

FESCAR-15. According to the results provided by the competition organizers, it seems that the generated test cases do cover some lines of code but no branches/conditions. For further investigation, we ran EvoSuite as a stand-alone tool (i.e., not using the benchmark infrastructure) but it was still not capable of covering any line. Through manual investigation, we notice that the CUT manages objects of the class `ScheduledExecutorService` and *threads*. Generating tests for CUTs that produce or manage threads is challenging and a well-known open problem in the literature [10]. EvoSuite does not prevent the CUT from spawning new threads, but it (1) forces the use of a wrapper class that leads to deterministic stack traces and thread names, and (2) any spawned threads are joined and removed after a test execution to ensure a clean state for successive test executions, which may consume a substantial amount of time.

FESCAR-2. This CUT contains only a few branches (i.e., 14) and lines (i.e., 19). EvoSuite was only able to generate test cases that cover around 20%-25% of lines but 0% of condition coverage. The generated tests cover branchless methods and the root branch of the method `lookup`, which is the only method with some conditions. It is worth noticing that EvoSuite could not complete a single generation, even with a search budget of 180s. The CUT is particularly expensive, and each generated test requires seconds for its execution. Low search budgets are thus not feasible nor recommended for this CUT.

FESCAR-41, FESCAR-6. EvoSuite reached a very low line and branch coverage for these two CUTs, independently from the search budget. These CUT also manage threads, and more precisely objects of the classes `ThreadPoolExecutor` and `NamedThreadFactory`.

Table 2: Detailed results of EvoSuite on the SBST benchmark classes.

Benchmark	Java Class	Line Coverage		Branch Coverage		Mutation Score	
		60s	180s	60s	180s	60s	180s
FESCAR-10	com.alibaba.fescar.core.model.BranchType	80.0%	90.0%	80.0%	90.0%	80.0%	90.0%
FESCAR-12	com.alibaba.fescar.core.rpc.netty.RpcServerHandler	100.0%	100.0%	87.5%	87.5%	100.0%	100.0%
FESCAR-13	com.alibaba.fescar.core.exception.TransactionExceptionCode	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
FESCAR-15	com.alibaba.fescar.core.rpc.netty.RpcServer	0.8%	0.7%	0.0%	0.0%	0.0%	0.0%
FESCAR-17	com.alibaba.fescar.core.protocol.transaction.GlobalBeginResponse	99.4%	99.4%	100.0%	100.0%	90.0%	90.0%
FESCAR-2	com.alibaba.fescar.core.service.ServiceManagerStaticConfigImpl	20.5%	25.8%	0.0%	0.0%	0.0%	0.0%
FESCAR-23	com.alibaba.fescar.core.protocol.MergeResultMessage	90.5%	60.5%	76.4%	50.0%	0.0%	0.0%
FESCAR-25	com.alibaba.fescar.core.rpc.netty.RmMessageListener	46.9%	37.5%	62.5%	48.8%	22.2%	17.8%
FESCAR-28	com.alibaba.fescar.core.rpc.ClientType	90.0%	100.0%	90.0%	100.0%	90.0%	100.0%
FESCAR-32	com.alibaba.fescar.core.protocol.transaction.BranchRegisterRequest	97.7%	89.2%	94.4%	87.5%	95.2%	78.3%
FESCAR-33	com.alibaba.fescar.core.model.GlobalStatus	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
FESCAR-34	com.alibaba.fescar.core.protocol.ResultCode	90.0%	100.0%	90.0%	100.0%	90.0%	100.0%
FESCAR-37	com.alibaba.fescar.core.rpc.RpcContext	92.4%	94.6%	86.8%	91.2%	0.0%	0.0%
FESCAR-41	com.alibaba.fescar.core.rpc.netty.RmRpcClient	1.7%	1.7%	2.0%	2.0%	0.0%	2.4%
FESCAR-42	com.alibaba.fescar.core.rpc.DefaultServerMessageListenerImpl	24.3%	42.6%	11.8%	27.1%	12.1%	25.4%
FESCAR-5	com.alibaba.fescar.core.protocol.MessageFuture	98.6%	99.1%	96.0%	98.0%	99.2%	100.0%
FESCAR-6	com.alibaba.fescar.core.rpc.netty.TmRpcClient	3.4%	3.4%	2.7%	2.7%	0.0%	2.7%
FESCAR-7	com.alibaba.fescar.core.rpc.netty.MessageCodecHandler	76.1%	78.2%	73.3%	77.2%	0.0%	0.0%
FESCAR-8	com.alibaba.fescar.core.rpc.netty.NettyPoolableFactory	57.3%	62.0%	50.8%	57.5%	0.0%	0.0%
FESCAR-9	com.alibaba.fescar.core.protocol.transaction.GlobalBeginRequest	99.0%	98.3%	100.0%	100.0%	99.1%	98.2%
GUAVA-102	com.google.common.collect.LinkedListMultimap	29.4%	32.3%	12.9%	11.6%	19.2%	14.8%
GUAVA-110	com.google.common.collect.LexicographicalOrdering	3.0%	22.2%	0.0%	7.5%	0.6%	15.0%
GUAVA-128	com.google.common.base.Throwables	75.1%	25.0%	75.8%	25.3%	81.0%	26.8%
GUAVA-129	com.google.common.collect.SparseImmutableTable	31.9%	35.8%	37.5%	42.5%	35.0%	43.8%
GUAVA-159	com.google.common.primitives.ParseRequest	100.0%	100.0%	100.0%	100.0%	50.0%	50.0%
GUAVA-169	com.google.common.math.LongMath	96.2%	86.7%	94.2%	85.3%	99.2%	89.3%
GUAVA-177	com.google.common.primitives.Doubles	98.7%	98.5%	99.3%	99.3%	100.0%	100.0%
GUAVA-181	com.google.common.primitives.SignedBytes	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
GUAVA-184	com.google.thirdparty.publicsuffix.PublicSuffixType	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
GUAVA-196	com.google.common.io.Closeables	71.5%	70.0%	77.5%	75.0%	88.0%	88.0%
GUAVA-2	com.google.common.collect.MinMaxPriorityQueue	13.9%	22.5%	6.4%	11.1%	16.5%	19.2%
GUAVA-206	com.google.common.collect.ImmutableEnumSet	25.4%	26.1%	23.6%	24.5%	7.1%	7.6%
GUAVA-212	com.google.common.net.MediaType	92.6%	94.3%	77.6%	83.0%	0.0%	0.0%
GUAVA-22	com.google.common.graph.Graphs	53.9%	49.7%	51.8%	47.3%	0.0%	0.0%
GUAVA-224	com.google.common.primitives.UnsignedLongs	99.3%	89.6%	100.0%	90.0%	100.0%	90.0%
GUAVA-240	com.google.common.collect.FilteredMultimapValues	12.3%	22.7%	0.0%	5.0%	0.0%	0.0%
GUAVA-39	com.google.common.collect.TreeMultiset	30.2%	43.1%	18.6%	27.9%	19.5%	31.3%
GUAVA-47	com.google.common.collect.FilteredEntryMultimap	2.6%	11.3%	0.0%	0.7%	0.0%	0.4%
GUAVA-90	com.google.common.io.FileBackedOutputStream	98.9%	89.6%	98.1%	90.0%	98.0%	89.3%
GUAVA-95	com.google.common.collect.ComparatorOrdering	27.5%	51.7%	12.5%	30.0%	18.8%	31.2%
PDFBOX-117	org.apache.pdfbox.filter.Predictor	89.0%	93.5%	83.9%	91.0%	0.0%	28.6%
PDFBOX-127	org.apache.pdfbox.pdfparser.PDFObjectStreamParser	57.5%	65.6%	37.1%	43.6%	44.4%	50.6%
PDFBOX-130	org.apache.pdfbox.pdmodel.interactive.digitalsignature.visible.PDVisibleSignDesigner	7.1%	14.3%	1.7%	1.7%	1.5%	2.5%
PDFBOX-157	org.apache.pdfbox.pdmodel.font.PDType1Font	2.1%	0.0%	0.4%	0.0%	0.0%	0.0%
PDFBOX-198	org.apache.pdfbox.pdmodel.fdf.FDFAnnotationLine	66.4%	66.5%	32.4%	32.7%	5.5%	0.0%
PDFBOX-214	org.apache.pdfbox.pdfparser.EndstreamOutputStream	99.5%	90.0%	99.2%	90.0%	48.0%	40.0%
PDFBOX-22	org.apache.pdfbox.pdmodel.fdf.FDFAnnotationCaret	63.9%	63.9%	64.3%	64.3%	10.5%	31.4%
PDFBOX-220	org.apache.pdfbox.filter.JPXFilter	32.7%	32.7%	7.7%	7.3%	0.0%	0.0%
PDFBOX-229	org.apache.pdfbox.util.XMLUtil	62.4%	69.6%	52.5%	60.0%	10.7%	13.6%
PDFBOX-234	org.apache.pdfbox.pdmodel.interactive.action.PDActionSound	97.7%	96.7%	88.9%	87.8%	0.0%	20.0%
PDFBOX-235	org.apache.pdfbox.pdmodel.font.PDTrueTypeFontEmbedder	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
PDFBOX-26	org.apache.pdfbox.pdmodel.encryption.SecurityProvider	55.8%	56.8%	100.0%	100.0%	100.0%	90.0%
PDFBOX-265	org.apache.pdfbox.pdmodel.font.PDType3Font	62.4%	70.2%	42.3%	52.0%	0.0%	0.0%
PDFBOX-278	org.apache.pdfbox.pdfwriter.ContentStreamWriter	96.8%	98.3%	96.7%	96.3%	0.0%	0.0%
PDFBOX-285	org.apache.pdfbox.pdmodel.interactive.digitalsignature.PDSignature	98.9%	99.7%	89.5%	95.5%	0.0%	0.0%
PDFBOX-40	org.apache.pdfbox.pdmodel.font.PDCIDFontType2	57.2%	54.9%	45.1%	46.6%	0.0%	0.0%
PDFBOX-62	org.apache.pdfbox.rendering.PageDrawer	2.3%	6.8%	1.2%	4.2%	0.0%	0.0%
PDFBOX-8	org.apache.pdfbox.pdmodel.font.FileSystemFontProvider	45.2%	48.4%	34.2%	35.8%	41.9%	52.2%
PDFBOX-83	org.apache.pdfbox.contentstream.operator.text.SetTextRenderingMode	89.3%	85.7%	92.5%	100.0%	82.5%	87.5%
PDFBOX-91	org.apache.pdfbox.pdmodel.documentinterchange.taggedpdf.PDArtifactMarkedContent	91.6%	97.9%	71.2%	92.5%	0.0%	0.0%
SPOON-105	spoon.support.compiler.jdt.PositionBuilder	9.6%	5.5%	7.8%	3.9%	0.0%	0.0%
SPOON-155	spoon.reflect.visitor.filter.AllMethodsSameSignatureFunction	13.0%	12.7%	0.0%	1.2%	0.7%	3.2%
SPOON-16	spoon.reflect.path.CtElementPathBuilder	15.9%	16.1%	8.0%	9.0%	10.3%	6.4%
SPOON-169	spoon.reflect.visitor.ImportScannerImpl	1.2%	10.6%	0.1%	4.7%	0.0%	1.3%
SPOON-20	spoon.support.reflect.reference.CtLocalVariableReferenceImpl	30.0%	38.6%	14.0%	18.0%	3.3%	13.3%
SPOON-211	spoon.reflect.path.impl.CtRolePathElement	16.3%	18.3%	6.2%	10.3%	6.2%	11.2%
SPOON-25	spoon.pattern.internal.ValueConvertorImpl	3.0%	7.1%	1.2%	3.1%	0.7%	4.3%
SPOON-253	spoon.pattern.internal.parameter.MapParameterInfo	76.8%	73.9%	72.5%	73.8%	0.0%	0.0%
SPOON-32	spoon.MavenLauncher	27.0%	30.0%	11.2%	12.5%	6.0%	6.7%
SPOON-65	spoon.support.DefaultCoreFactory	10.7%	9.7%	5.9%	8.9%	0.1%	0.0%
Average		55.9%	57.0%	50.8%	51.7%	32.6%	33.8%

GUAVA-110, GUAVA-240. Both classes make use of the annotation `@Nullable`. GUAVA-110 uses the annotation for the input parameter of the method `equals(...)`, while GUAVA-240 uses it for the methods `contains(...)` and `remove(...)`. The static analyzer implemented in EvoSuite does not handle the annotation `@Nullable` and triggers the error *"Cannot find symbol"*. This, in retrospect, would have been easy to avoid.

GUAVA-47. For this class, EvoSuite reached very low coverage (both line and branch) and mutation score. To have a better understanding of the underlying issue, we ran EvoSuite stand-alone. We notice that our tool could not generate any single test cases, never completing the initialization process (and the generation of the initial population, in particular) even with a search budget of 180s. A more in-depth analysis is needed to discover the root cause of the issue.

PDFBOX-157. EvoSuite crashed in 9 out ten times with a search budget of 60s, and in 10 out of 10 runs for larger budgets. The exact reasons are not yet clear, and will require further debugging.

FESCAR-23, FESCAR-37, FESCAR-7, FESCAR-8, GUAVA-212, GUAVA-22, PDFBOX-265, PDFBOX-278, PDFBOX-285, PDFBOX-40, PDFBOX-91, PDFBOX-62, SPOON-105, SPOON-253. For all these classes the mutation analysis failed due to a `java.util.concurrent.ExecutionException` thrown by the experimental infrastructure, rather than by EvoSuite.

PDFBOX-235, SPOON-155. In these two cases, EvoSuite crashed while attempting to set up mock objects; these are bugs in EvoSuite.

5 CONCLUSIONS

This paper reports on the participation of the EvoSuite test generation tool in the 8th SBST Java Unit Testing Tool Contest. With an overall score of 406.14 points, EvoSuite achieved the highest score of all tools in the competition. Despite the many years of development, the benchmark used in the competition points out several opportunities for improvement, which we discussed in this paper.

To learn more about EvoSuite, visit our Web site:

`http://www.evosuite.org`

Acknowledgments: Many thanks to all the contributors to EvoSuite. This project has been funded by the EPSRC project "GREATEST" (EP/N023978/2).

REFERENCES

[1] M. Moein Almasi, Hadi Hemmati, Gordon Fraser, Andrea Arcuri, and Jānis Benefelds. 2017. An Industrial Evaluation of Unit Test Generation: Finding Real Faults in a Financial Application. In *ACM/IEEE Int. Conference on Software Engineering (ICSE)*. IEEE, 263–272.

[2] A. Arcuri, J. Campos, and G. Fraser. 2016. Unit Test Generation During Software Development: EvoSuite Plugins for Maven, IntelliJ and Jenkins. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE Computer Society, 401–408.

[3] Andrea Arcuri and Gordon Fraser. 2013. Parameter tuning or default values? An empirical investigation in search-based software engineering. *Empirical Software Engineering (EMSE)* (2013), 1–30. https://doi.org/DOI:10.1007/s10664-013-9249-9

[4] Andrea Arcuri and Gordon Fraser. 2016. Java Enterprise Edition Support in Search-Based JUnit Test Generation. In *Int. Symposium on Search Based Software Engineering*. Springer, 3–17.

[5] A. Arcuri, G. Fraser, and R. Just. 2017. Private API Access and Functional Mocking in Automated Unit Test Generation. In *IEEE Int. Conference on Software Testing, Verification and Validation (ICST)*. 126–137.

[6] José Campos, Yan Ge, Nasser Albunian, Gordon Fraser, Marcelo Eler, and Andrea Arcuri. 2018. An empirical evaluation of evolutionary algorithms for unit test suite generation. *Information and Software Technology* 104 (2018), 207 – 235. https://doi.org/10.1016/j.infsof.2018.08.010

[7] G. Fraser and A. Arcuri. 2011. EvoSuite: Automatic Test Suite Generation for Object-Oriented Software.. In *ACM Symposium on the Foundations of Software Engineering (FSE)*. 416–419.

[8] Gordon Fraser and Andrea Arcuri. 2013. EvoSuite at the SBST 2013 Tool Competition. In *Int. Workshop on Search-Based Software Testing (SBST)*. 406–409.

[9] Gordon Fraser and Andrea Arcuri. 2013. EvoSuite at the Second Unit Testing Tool Competition.. In *Fittest Workshop*. Springer, 95–100.

[10] G. Fraser and A. Arcuri. 2013. 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)*. 362–369.

[11] Gordon Fraser and Andrea Arcuri. 2013. Whole Test Suite Generation. *IEEE Transactions on Software Engineering* 39, 2 (2013), 276–291.

[12] Gordon Fraser and Andrea Arcuri. 2014. A large-scale evaluation of automated unit test generation using EvoSuite. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24, 2 (2014), 8:1–8:42.

[13] Gordon Fraser and Andrea Arcuri. 2015. Achieving Scalable Mutation-based Generation of Whole Test Suites. *Empirical Software Engineering (EMSE)* 20, 3 (2015), 783–812.

[14] Gordon Fraser and Andrea Arcuri. 2015. EvoSuite at the SBST 2015 Tool Competition. In *Int. Workshop on Search-Based Software Testing (SBST)*. IEEE Press, 25–27.

[15] Gordon Fraser and Andrea Arcuri. 2016. EvoSuite at the SBST 2016 Tool Competition. In *Int. Workshop on Search-Based Software Testing (SBST)*. ACM, 33–36.

[16] Gordon Fraser, Matt Staats, Phil McMinn, Andrea Arcuri, and Frank Padberg. 2015. Does Automated Unit Test Generation Really Help Software Testers? A Controlled Empirical Study. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24, 4 (2015), 23:1–23:49.

[17] Gordon Fraser and Andreas Zeller. 2012. Mutation-Driven Generation of Unit Tests and Oracles. *IEEE Transactions on Software Engineering (TSE)* 28, 2 (2012), 278–292.

[18] José Campos Gordon Fraser, José Miguel Rojas and Andrea Arcuri. 2017. EvoSuite at the SBST 2017 Tool Competition. In *Int. Workshop on Search-Based Software Testing (SBST)*. IEEE Press, 39–41.

[19] José Miguel Rojas Gordon Fraser and Andrea Arcuri. 2018. EvoSuite at the SBST 2018 Tool Competition. In *Int. Workshop on Search-Based Software Testing (SBST)*. IEEE Press, 34–37.

[20] José Miguel Rojas Gordon Fraser and Andrea Arcuri. 2019. EvoSuite at the SBST 2019 Tool Competition. In *2019 IEEE/ACM 12th International Workshop on Search-Based Software Testing (SBST)*. IEEE, 29–32.

[21] A. Panichella, F. M. Kifetew, and P. 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 (Feb 2018), 122–158. https://doi.org/10.1109/TSE.2017.2663435

[22] A. Panichella, F. M. Kifetew, and P. Tonella. 2018. Incremental Control Dependency Frontier Exploration for Many-Criteria Test Case Generation. In *International Symposium on Search Based Software Engineering*. Springer, 309–324.

[23] A. Panichella, F. M. Kifetew, and P. Tonella. 2018. A large scale empirical comparison of state-of-the-art search-based test case generators. *Information and Software Technology* 104 (2018), 236–256.

[24] José Miguel Rojas, José Campos, Mattia Vivanti, Gordon Fraser, and Andrea Arcuri. 2015. Combining Multiple Coverage Criteria in Search-Based Unit Test Generation. In *Search-Based Software Engineering*. Springer, 93–108.

[25] José Miguel Rojas, Gordon Fraser, and Andrea Arcuri. 2015. Automated Unit Test Generation during Software Development: A Controlled Experiment and Think-Aloud Observations. In *ACM Int. Symposium on Software Testing and Analysis (ISSTA)*. ACM, 338–349.

[26] José Miguel Rojas, Mattia Vivanti, Andrea Arcuri, and Gordon Fraser. 2016. A Detailed Investigation of the Effectiveness of Whole Test Suite Generation. *Empirical Software Engineering (EMSE)* 22, 2 (2016), 852–893. https://doi.org/10.1007/s10664-015-9424-2

[27] Abdelilah Sakti, Gilles Pesant, and Yann-Gaël Guéhéneuc. 2015. Instance generator and problem representation to improve object oriented code coverage. *IEEE Transactions on Software Engineering* 41, 3 (2015), 294–313.

[28] Sina Shamshiri, Rene Just, Jose Miguel Rojas, Gordon Fraser, Phil McMinn, and Andrea Arcuri. 2015. Do Automatically Generated Unit Tests Find Real Faults? An Empirical Study of Effectiveness and Challenges. In *IEEE/ACM Int. Conference on Automated Software Engineering (ASE)*. IEEE, 201–211.

[29] Sina Shamshiri, José Miguel Rojas, Juan Pablo Galeotti, Neil Walkinshaw, and Gordon Fraser. 2018. How Do Automatically Generated Unit Tests Influence Software Maintenance? In *IEEE Int. Conference on Software Testing, Verification and Validation (ICST)*. 250–261.