JTExpert at the Fourth Unit Testing Tool Competition

Abdelilah Sakti
United Technologies Research Center
Cork, Co. Cork, Ireland
saktia@utrc.utc.com

Gilles Pesant
Department of Computer and Software Engineering
École Polytechnique de Montréal
Montréal, Quebéc, Canada
gilles.pesant@polymtl.ca

Yann-Gaël Guéhéneuc
Department of Computer and Software Engineering
École Polytechnique de Montréal
Montréal, Quebéc, Canada
yann-gael.gueheneuc@polymtl.ca

ABSTRACT

JTExpert is a software testing tool that automatically generates a whole test suite to satisfy the branch-coverage criterion. It takes as inputs a Java source code and its dependencies and automatically produces a test-case suite in JUnit format. In this paper, we summarize our results for the Unit Testing Tool Competition held at the fourth SBST Contest, where JTExpert received 931 points and was ranked third. We also analyze our tool's performance.

Keywords

—Test-case generation; classes testing; unit testing; random testing; static analysis.

1. INTRODUCTION

This paper describes and discusses the results obtained by applying the test-case generation tool JTExpert [3] on the benchmarks used to compare tools participating in the unit-testing competition held as part of the International Workshop on Search Based Software Testing (SBST) held in Austin, TX, on May 16-17. More details on the competition and the benchmarks can be found elsewhere [2].

In this competition, JTExpert received a total score equal to 931 points and was ranked third. The total score sums up the scores of four experiments evaluating the participating tools using a given time budget: the 1^{st} uses 60 seconds, the 2^{nd} uses 120 seconds, the 3^{rd} uses 240 seconds, and the 4^{th} uses 480 seconds. JTExpert received 179.24 in the 1^{st}, 231.11 in the 2^{nd}, 250.92 in the 3^{rd}, and 269.73 in the 4^{th}.

2. JTEXPERT

JTExpert is a software testing tool that has been developed to automatically generate a whole test suite that satisfies the branch coverage criterion on a given Java source code [3]. Table 1 summarizes the main features of JTExpert. JTExpert automatically generates a JUnit test suite for the Class Under Test (CUT). It can be used through a command line interface. It takes as inputs a Java file (.java) and its dependencies and automatically produces a test-case suite in JUnit format. JTExpert is available as an executable Jar file. It is based on four main components: a source code analyzer, a test-case candidates builder, an instances generator, and a random search strategy.

Table 1: Features of the tool JTExpert

Prerequisites	
Static or dynamic	Dynamic testing
Software Type	Java source code (.java)
Lifecycle phase	Unit testing for Java projects
Environment	Java
Knowledge required	JUnit
Experience required	Unit-testing knowledge
Input and Output of the Tool	
Input	A Java source code and its dependencies
Output	A test-case suite in JUnit 4 format
Operation	
Interaction	Through the command line
Source of information	https://sites.google.com/site/saktiabdel/JTExpert
Maturity	Still under development
Technology behind the tool	Random testing guided by static analyses
Obtaining the tool and information	
License	
Cost	Free
Support	None
Empirical evidence about the Tool	
Effectiveness	See [3]
Efficiency	See [3]
Scalability	See [3]

2.1 Source Code Analyzer

JTExpert uses a Source Code Analyzer (SCA) to determine the set of methods that are likely to change the state of a data member of the CUT and the set of methods that may reach a given branch. The SCA analyzes the source code to collect constants and path information about all the branches of all methods. SCA provides JTExpert's other components with information to guide them throughout the process of test-case generation.

2.2 Test Case Candidates Builder

JTExpert uses the Test Case Candidates Builder (TDCB) to explore only relevant sequences of method calls. Using the collected information by SCA, the test-case generation problem is represented by a vector composed of means-of-

ICSE '16 May 16-17 2016, Austin, TX, USA

ACM ISBN 978-1-4503-4166-0/16/05.

DOI: http://dx.doi.org/10.1145/2897010.2897021

instantiation of the CUT, methods that are likely to change the object state by changing a data member, and methods that may reach the branch target. Thus, JTExpert represents a test-case candidate by: (1) a means-of-instantiation of the class under test (i.e., a constructor, a method factory, a data field, or method external from the CUT); (2) a sequence of method calls whose length (i.e., number of method calls) is bounded by the number of declared data members in the CUT, each method in a sequence being called in the hope to put a given data member in a relevant state; (3) a method call that is likely to reach the test target; (4) a means-of-instantiation for each argument of the method.

The TDCB is a key novelty of JTExpert compared to other tools because it prevents JTExpert exploring useless sequences and thus to generate test cases faster without compromising coverage.

2.3 Instances Generator

JTExpert uses a customized instances generator based on a seeding strategy and a dynamic strategy to diversify generated instances of classes. The seeding strategy gets collected constants for each primitive data type or string and seeds them while generating instances. It defines a seeding probability of each data type according to the number of collected constants. Also, it seeds the null value with a constant probability while generating instances of classes. The diversification strategy generates different instances by using different means-of-instantiation (e.g., constructors, factory methods, subclasses).

The instances generator improves JTExpert exploration of the search space, reaching more branches, and thus increasing code coverage for a given time.

2.4 Random Search Strategy

JTExpert uses a random search that targets every uncovered branches at the same time: it does not focus on only one branch, instead it generates a test-case candidate uniformly at random for every uncovered branches. This strategy allows JTExpert to reach a good branch coverage quickly because it does not waste efforts on unreachable branches and it benefits from the significant number of branches that may be covered accidentally.

3. BENCHMARK RESULTS

Table 3 presents the results of JTExpert aggregated per benchmark. On average, JTExpert achieved 56.34% instructions coverage, 49.09% branch coverage, and 35.51% mutation coverage. These results are in line with our expectations except for classes where JTExpert gets 0% mutation coverage. In the following subsections, we highlight where our tool performed more poorly and provide possible explanations.

3.1 Compilation Errors

During the competition, JTExpert produced many uncompilable test-case files that significantly affected its performance. In all the experiments, JTExpert generated 37 uncompilable test-case files distributed as follow: 8 files during the first experiment; 7 files during the second experiment; 11 files during the third experiment, and 11 files during the fourth experiment. Each uncompilable test-case file received a score of 0 and -2 points as penalty. This problem specially affected four benchmarks: `Chart-1`, `Chart-4`, `Chart-26`,and `Math-44`.

We analyzed these classes and observed that the first four classes are abstract and the problem relates to the way JTExpert generates test cases for an abstract class. Actually, JTExpert instantiates an abstract class by using a possible stub and calls some methods from that stub. The problem was that, during the generation of source-code, we omitted to cast some variable from the abstract-class to the stub type.

3.2 Time Management

In the results file, we observed that for many CUTs JTExpert did not generate a test-case file. It could not generate 33 test-case files: 11 in the first experiment, nine in the second experiment, seven in the third experiment, and six in the fourth experiment. In general, JTExpert always generates a test-case file albeit an empty test-case file. We think that JTExpert generated test-case files for these CUTs but perhaps after the allowed time budget.

3.3 Low Branch Coverage

JTExpert achieved a weak branch coverage on the 16 benchmarks from the library Closure Compiler, `com.google.javascript.jscomp`. We analyzed the results of all tools and observed that this seems a general trend because all the participating tools have poor performances on this set of classes. The Closure Compiler is a JavaScript compiler and its classes includes a parser, lexical analyzer, and syntactical analyzer. In general, to instantiate such a type of class, a test-case generation tool must generate some strings that respect certain regular expressions and syntax. Because randomly generating such strings is almost impossible, a tool will fail to instantiate the CUT or the classes required to call its methods. Therefore, the low coverage achieved on classes from this library represents only the exceptions raised in constructors and public methods.

3.4 Mutation Coverage Measure

For many CUTs, the mutation coverage is very low compared to the code coverage. JTExpert generated a test-case file that reached more that 90% code coverage but, with 0% mutation coverage. For example, in the benchmark `Lang-37` at the third experimentation (240s) and run number 5, the line coverage is equals to 94.67% with 0% mutation coverage. In Table 3, this problem is clear in the benchmarks `Chart-1`, `Chart-12`, and `Chart-23` because it happened in all their test-case files. It also partially affected other benchmarks like `Lang-36`, `Lang-41`, and `Lang-58` but it is not clear in the aggregated results. This problem affects 244 test-case suites generated by JTExpert.

During the set-up, we observed and mentioned a similar problem in Defect4j [1], the tool that measured the mutation coverage. Actually, if Defect4j cannot measure the mutation coverage for a test-case set, then it assigns a score of 0% of mutation coverage. We think that this measurement problem is at the root of some large differences between the code coverage and mutation coverage. Indeed, the organizers confirmed that this problem exists and affected 38 classes and around $1,024$ test-case sets generated by different tools.

4. ANALYSIS AND DISCUSSIONS

Because of the abstract-classes bug, JTExpert produced 37 uncompilable test-case files that represent almost 2.5%

Table 2: DETAILED RESULTS OF JTExpert ON THE SBST-CONTEST BENCHMARKS WITH A TIME BUDGET EQUAL TO 240 SECONDS

Benchmark	Class Name	Score	FailTests	Coverage			Total		
				Mutation	Branch	Line	Mutant	Branch	Line
Lang-63	org.apache.commons.lang.time.DurationFormatUtils	37.83	2.33	0.73	0.96	0.99	329	133	229
Lang-41	org.apache.commons.lang.ClassUtils	32.74	4.62	0.56	0.86	0.91	346	216	268
Lang-33	org.apache.commons.lang3.ClassUtils	32.71	1.08	0.56	0.85	0.91	349	218	268
Time-5	org.joda.time.Period	28.85	0.67	0.67	0.93	0.97	467	64	288
Math-103	org.apache.commons.math.distribution.NormalDistributionImpl	28.56	2.42	0.46	0.67	0.76	108	18	42
Chart-11	org.jfree.chart.util.ShapeUtilities	28.33	3.5	0.39	0.68	0.80	418	116	193
Math-56	org.apache.commons.math.util.MultidimensionalCounter	27.43	1.38	0.48	0.83	0.95	138	32	70
Chart-16	org.jfree.data.category.DefaultIntervalCategoryDataset	27.34	9.62	0.30	0.75	0.80	273	128	186
Math-2	org.apache.commons.math3.distribution.HypergeometricDistribution	26.77	0.17	0.79	0.94	0.99	175	26	66
Lang-57	org.apache.commons.lang.LocaleUtils	26.22	0.75	0.55	0.81	0.95	177	64	76
Time-13	org.joda.time.format.PeriodFormatterBuilder	26.04	4.29	0.50	0.83	0.93	1139	458	665
Lang-36	org.apache.commons.lang3.math.NumberUtils	25.19	0.62	0.43	0.85	0.96	812	352	373
Lang-43	org.apache.commons.lang.text.ExtendedMessageFormat	24.73	3.12	0.20	0.43	0.55	184	99	163
Lang-47	org.apache.commons.lang.text.StrBuilder	24.58	2.12	0.56	0.91	0.96	1642	496	700
Math-106	org.apache.commons.math.fraction.ProperFractionFormat	23.96	0.67	0.30	0.66	0.78	46	19	63
Lang-58	org.apache.commons.lang.math.NumberUtils	22.79	0.42	0.49	0.84	0.94	977	420	433
Time-4	org.joda.time.Partial	22.20	0.29	0.59	0.92	0.94	336	106	249
Lang-65	org.apache.commons.lang.time.DateUtils	21.57	0.67	0.42	0.78	0.91	406	175	208
Chart-7	org.jfree.data.time.TimePeriodValues	21.20	0.08	0.68	0.82	0.87	254	54	149
Math-91	org.apache.commons.math.fraction.Fraction	20.96	0.00	0.67	0.83	0.94	360	90	149
Time-10	org.joda.time.base.BaseSingleFieldPeriod	20.21	0.00	0.69	0.76	0.84	118	46	70
Chart-24	org.jfree.chart.renderer.GrayPaintScale	19.93	0.29	0.38	0.88	0.84	37	10	25
Lang-50	org.apache.commons.lang.time.FastDateFormat	19.54	0.00	0.68	0.83	0.93	689	228	455
Lang-59	org.apache.commons.lang.text.StrBuilder	19.38	0.46	0.58	0.92	0.97	1584	464	656
Chart-17	org.jfree.data.time.TimeSeries	17.42	0.92	0.25	0.55	0.61	428	138	280
Chart-20	org.jfree.chart.plot.ValueMarker	16.66	0.00	0.45	0.75	0.91	11	8	22
Lang-60	org.apache.commons.lang.text.StrBuilder	16.55	0.21	0.56	0.91	0.96	1584	464	656
Lang-37	org.apache.commons.lang3.ArrayUtils	15.79	0.00	0.63	0.81	0.94	2015	806	957
Math-52	org.apache.commons.math.geometry.euclidean.threed.Rotation	15.06	0.00	0.48	0.71	0.77	2706	122	327
Time-3	org.joda.time.MutableDateTime	14.89	0.12	0.23	0.87	0.86	90	63	247
Time-8	org.joda.time.DateTimeZone	13.83	0.08	0.38	0.62	0.75	439	182	354
Time-23	org.joda.time.DateTimeZone	13.38	0.00	0.41	0.65	0.77	353	162	338
Time-7	org.joda.time.format.DateTimeFormatter	13.25	0.00	0.39	0.72	0.88	176	88	208
Math-88	org.apache.commons.math.optimization.linear.SimplexTableau	12.85	0.04	0.32	0.57	0.66	392	90	131
Math-21	org.apache.commons.math3.linear.RectangularCholeskyDecomposition	12.55	0.00	0.24	0.80	0.83	102	28	55
Math-93	org.apache.commons.math.util.MathUtils	12.20	0.04	0.47	0.64	0.68	892	223	227
Time-20	org.joda.time.format.DateTimeFormatterBuilder	11.98	0.00	0.43	0.70	0.80	1743	595	1034
Chart-23	org.jfree.chart.renderer.category.MinMaxCategoryRenderer	11.12	1.42	0.09	0.31	0.42	168	56	149
Math-67	org.apache.commons.math.optimization.MultiStartUnivariateRealOptimizer	10.91	0.25	0.23	0.43	0.61	149	34	90
Chart-3	org.jfree.data.time.TimeSeries	10.77	0.00	0.40	0.66	0.76	490	194	371
Chart-2	org.jfree.data.general.DatasetUtilities	10.16	0.00	0.28	0.51	0.58	794	548	808
Chart-6	org.jfree.chart.util.ShapeList	8.82	0.00	0.18	0.50	0.50	28	16	34
Chart-1	org.jfree.chart.renderer.category.AbstractCategoryItemRenderer	8.52	1.33	0.07	0.25	0.33	246	242	519
Chart-9	org.jfree.data.time.TimeSeries	8.47	0.04	0.26	0.53	0.61	435	140	280
Math-18	org.apache.commons.math3.optimization.direct.CMAESOptimizer	7.73	0.00	0.14	0.50	0.65	1613	276	467
Math-20	org.apache.commons.math3.optimization.direct.CMAESOptimizer	7.63	0.00	0.13	0.48	0.65	1601	272	458
Lang-28	org.apache.commons.lang3.text.translate.NumericEntityUnescaper	6.36	0.00	0.08	0.40	0.46	80	16	25
Closure-98	com.google.javascript.jscomp.ReferenceCollectingCallback	6.33	0.00	0.17	0.27	0.57	248	143	167
Time-11	org.joda.time.tz.ZoneInfoCompiler	5.51	0.00	0.14	0.32	0.36	419	193	385
Chart-12	org.jfree.chart.plot.MultiplePiePlot	4.57	0.04	0.10	0.41	0.52	164	84	187
Closure-100	com.google.javascript.jscomp.CheckGlobalThis	4.34	0.00	0.08	0.18	0.43	98	56	45
Closure-99	com.google.javascript.jscomp.CheckGlobalThis	4.04	0.00	0.07	0.17	0.42	112	62	45
Closure-14	com.google.javascript.jscomp.ControlFlowAnalysis	3.76	0.00	0.10	0.15	0.25	237	276	382
Closure-16	com.google.javascript.jscomp.ScopedAliases	3.69	0.00	0.10	0.13	0.25	166	124	194
Closure-132	com.google.javascript.jscomp.PeepholeSubstituteAlternateSyntax	2.33	0.00	0.07	0.10	0.14	674	536	687
Closure-20	com.google.javascript.jscomp.PeepholeSubstituteAlternateSyntax	2.26	0.00	0.07	0.10	0.14	662	530	686
Math-39	org.apache.commons.math.ode.nonstiff.EmbeddedRungeKuttaIntegrator	1.28	0.00	0.01	0.00	0.30	279	54	108
Chart-25	org.jfree.chart.renderer.category.StatisticalBarRenderer	1.05	0.00	0.01	0.07	0.11	372	90	181
Closure-74	com.google.javascript.jscomp.PeepholeFoldConstants	0.71	0.00	0.01	0.03	0.07	880	624	765
Math-7	org.apache.commons.math3.ode.AbstractIntegrator	0.69	0.00	0.06	0.05	0.22	82	58	127
Closure-130	com.google.javascript.jscomp.CollapseProperties	0.52	0.00	0.00	0.03	0.07	358	266	330
Closure-124	com.google.javascript.jscomp.ExploitAssigns	0.48	0.00	0.01	0.01	0.04	73	68	74
Math-64	org.apache.commons.math.optimization.general.LevenbergMarquardtOptimizer	0.20	0.00	0.00	0.00	0.05	1083	204	351
Math-44	org.apache.commons.math.ode.AbstractIntegrator	0.10	0.00	0.07	0.05	0.23	72	46	120
Closure-68	com.google.javascript.jscomp.parsing.JsDocInfoParser	0.00	0.50	0.00	0.00	0.00	482	589	936
Closure-46	com.google.javascript.rhino.jstype.RecordType	0.00	1.00	0.00	0.00	0.00	58	72	88
Chart-4	org.jfree.chart.plot.XYPlot	-2.17	0.12	0.01	0.03	0.05	1369	966	1749
Chart-26	org.jfree.chart.axis.Axis	-5.38	0.00	0.00	0.00	0.00	378	136	329
Total/Average		**931.01**		**35.51%**	**49.09%**	**56.34%**	**78,664**	**29,172**	**47,608**

Table 3: Synthesis: the Results of all Tools

Metrics		JTExpert	Evosuite	T3	Randoop
Line	#	88,068			
	Coverage	56.34 %	60.79 %	48.74 %	47.92 %
	Rank	2	**1**	3	4
Branch	#	55,888			
	Coverage	49.09 %	48.51 %	41.98 %	38.49 %
	Rank	**1**	2	3	4
Mutant	#	144,660			
	Coverage	35.51 %	33.50 %	37.56 %	25.99 %
	Rank	2	3	**1**	4
Test Cases	#	**76,407**	88,531	194,193	14,013,583
	Rank	**1**	2	3	4
Uncompilable	#	37	13	0	5728
	Rank	3	2	**1**	4
Broken test	#	3,220	224	8,010	190,493
	Rank	2	**1**	3	4
Fail test	#	404	451	482	359
	Rank	3	2	**1**	4
Time	# (h)	104.14	96.63	63.54	126.98
	Rank	3	2	**1**	4
Score Details	Line	166.43	181.54	128.74	129.85
	Branch	284.31	309.37	224.81	201.65
	Mutant	334.08	360.86	319.94	248.92
	Fail	269.33	300.67	321.33	239.33
	Penalty	-123.15	-25.79	-17.26	-72.81
	Total	931.01	1126.65	977.57	746.95
	Rank	3	**1**	2	4

of the CUTs, 4% of the lines, 3% of the branches, and 2.5% of the mutants. Also, because of the time-management bug, JTExpert missed to generate 33 test-case files that represent 2% of the CUTs, 6% of the lines, 5.38% of the branches, and 2.7% of the mutants. Hat with standing, these bugs and with the smallest number of test cases, Table 3 shows that JTExpert outperformed the other tools in terms of branch-coverage criterion and it is ranked second in terms of line-coverage and mutation-coverage.

Table 3 presents a detailed comparison of the results of all participant tools. JTExpert performed better than Evosuite in terms of branch-coverage and mutation-coverage, whereas Evosuite got better scores than JTExpert in terms of these two criteria. This discrepancy could be explained by the way the score is computed. Indeed, the score assigned to a given class does not take in consideration the size of this class, i.e, a small class has the same weight as a large class. If we ignore the negative effects of the two bugs in JTExpert, then we can conclude that JTExpert outperforms Evosuite in large-size classes and Evosuite outperforms JTExpert in small-size classes.

JTExpert outperformed T3 in terms of line and branch coverage whereas T3 is better than JTExpert in terms of mutation coverage which could be explained by the number of test cases: JTExpert generated less than 40% of the number of test-cases generated by T3. It is well known that the number of test cases can significantly affects mutation coverage. The partial scores Line, Branch, and Mutant are in line with the code coverage, JTExpert outperformed T3, whereas T3 outperformed JTExpert in terms of Fail and Penalty metrics which could be explained by the time-management bug in JTExpert: the negative scores in penalty mainly come from the time required for test-cases generation. The metric Fail summarizes the number of actual errors each tool has revealed. To better understand the difference in terms of this metric between T3 and JTExpert, we analyzed in-depth this metric. Our analysis started at Table 3, where we observed that the benchmarks `Closure-46` and `Closure-68` have a value different to 0 in the FailTests column whereas the line coverage equals to 0. This incompatible information leads us to ask, how does a test-case set that does not cover any line can reveal the actual errors? For those benchmarks, we found that JTExpert generated many empty test-case files (i.e., without any test case). Because an empty file systematically generates execution error, Defect4j wrongly considers that this file reveals an actual error. We think that this is a bug in Defect4j that could affect also others classes. Actually, Defect4j considers a test case to reveal an actual error if its execution fails on a buggy version of the CUT regardless the reason why it failed. Therefore, we conclude that any test case fails on the CUT may wrongly be considered as revealing actual errors. To confirm this conclusion, we randomly selected three test-case sets from Randoop results, one of them was generated for the benchmark `Lang-59` and contains three test cases that fail on the CUT (i.e., `toolName=randoop, timeBudget=60, benchmarkName=Lang-59, runId=1`). Indeed, in the result log files, we found that instead of discarding these three test cases, Defect4j considered them able to reveal the actual errors in the buggy version of the CUT and assigned them 4 additional points in the scores. We reported this bug to the organizer who will consider it in next workshops.

This analysis showed that a bug in Defcet4j has affected the scores in terms of the Fail metric. We did not have enough time to analyze the other metrics, but we believe that the bug detected in Defect4j could also affect the mutation scores.

5. CONCLUSION

In this paper, we reported and analyzed the results obtained by JTExpert in the SBST Contest 2016. JTExpert performed well compared to its results in the SBST Contest 2015. However, the SBST Contest 2016 showed us different bugs and faults in JTExpert that must be tackled before the next SBST Contest. Also, our analysis of the results detected a bug in the contest platform that has affected the scores of the participant tools.

Actually, the SBST Contest 2016 offered a good opportunity to test some ideas that we partially implemented in JTExpert. We also learned, that the current version of JTExpert still needs more efforts to become a mature and robust software-testing tool.

We thank the SBST Contest organizers for helping in enhancing our tools and identifying new research directions that will make JTExpert better.

6. REFERENCES

[1] R. Just, D. Jalali, and M. D. Ernst. Defects4j: A database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 437–440. ACM, 2014.

[2] U. R. Molina, R. Just, J. Galeotti, T. Vos, and . Unit testing tool competition : Round four. In *Software Engineering, Search Based Software Testing Workshops (ICSEW), 2016 IEEE 38th International Conference on*, MAY 2016.

[3] A. Sakti, G. Pesant, and Y.-G. Guéhéneuc. Instance generator and problem representation to improve object oriented code coverage. *IEEE Transactions on Software Engineering*, pages 1–1, To appear, 2015.