JTeXpert at the SBST 2017 Tool Competition

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

Gilles Pesant, Yann-Gaël Guéhéneuc
Department of Computer and Software Engineering
École Polytechnique de Montréal
Montréal, Quebéc, Canada
Email: {gilles.pesant, 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 fifth SBST Contest, where JTeXpert received 849 points and was ranked second. We also analyze our tool's performance.**

***Keywords*-Test-case generation; classes testing; unit testing; random testing; static analysis;**

I. Introduction

This paper describes and discusses the results obtained by applying the test-case generation tool JTeXpert [1] 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 Buenos Aires, Argentina, on May 20-28. More details on the competition and the benchmarks can be found elsewhere [2].

In this competition, JTeXpert received a total score equal to 849 points and was ranked second. The total score sums up the scores of seven experiments evaluating the participating tools using a given time budget: the 1^{st} uses 10 seconds, the 2^{nd} uses 30 seconds, the 3^{rd} uses 60 seconds, the 4^{th} uses 120 seconds, the 5^{th} uses 240 seconds, the 6^{th} uses 300 seconds, and the 7^{th} uses 480 seconds. JTeXpert received 61.86 in the 1^{st}, 106.72 in the 2^{nd}, 125.52 in the 3^{rd}, 136.47 in the 4^{th}, 141.83 in the 5^{th}, 137.77 in the 6^{th}, and 138.37 in the 7^{th}.

II. About 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 [1]. Table I 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 publicly available as an executable Jar file. It is based on five main components: a source code analyzer, a test-case candidates builder, an instances generator, a random search strategy, and an oracle builder.

Table I
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 [1]
Efficiency	See [1]
Scalability	See [1]

A. 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.

B. 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-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.

C. 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.

D. 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.

III. Benchmark results

Table III presents the results of JTeXpert aggregated per benchmark. On average, JTeXpert achieved 33.71% instructions coverage, 28.22% branch coverage, and 28.93% mutation coverage. These results are in line with our expectations except for classes where JTeXpert gets score 0, i.e., 28 classes out of 69 or 40% of the competition score. In the following subsections, we highlight where our tool performed more poorly and provide possible explanations.

A. Compilation Errors

During the contest, JTeXpert produced many uncompilable test-case files that significantly affected its performance. In all the experiments, JTeXpert generated 15 uncompilable test-case files distributed as follow: 4 files during the last experiment; 3 files during the second and 5^{th} experiments; 1 file during the 1^{st}, 3^{rd}, and 4^{th} experiments. Each uncompilable test-case file received a score of 0 and -2 points as penalty. This problem appeared in different benchmarks, especially those form the library `LA4J`: `LA4J-1`, `LA4J-4`, `LA4J-5`, `LA4J-7`, `RE2J-2`, `FREEHEP-4`, `BCEL-6`, and `JXPATH-7`.

We analyzed these classes and observed a bug in JTeXpert at the last stage of source-code generation, more precisely, in the assertions generation. This bug appears when JTeXpert puts in the source code a constant string with a length greater than 4000. Actually, during the assertions generation, JTeXpert takes a value returned by a method call and uses it as an oracle. In addition, JTeXpert does not check the size of a constant string before inserting it in the source-code. Therefore, if a returned value is a string with length greater than 4000 this type of bug will emerge.

IV. Analysis and Discussions

JTeXpert did not generate any test-case file for 415 out of 1450 runs, which represents 27% of the competition runs. We randomly selected 10 of the CUTs affected and ran JTeXpert on them on the competition platform. JTeXpert performed well on all the selected CUTs and we have not observed in any run that JTeXpert failed to generate test suite. At the beginning, we thought this could be a bug in JTeXpert. To refute this hypothesis, we analyzed the JTeXpert error-log files that keep track of all the exceptions raised during test-case generation. We did not find any exception that could stop the execution of JTeXpert before writing the test-case file. There are two other possible components could be behind this problem: (1) the communication protocol between JTeXpert and the contest platform or (2) the contest platform itself. We closely inspected the source code of the simple communication protocol (runJTeXpert). runJTeXpert only builds the JTeXpert command line and receives/sends simple messages from/to the contest platform. There is nothing special or complex in this protocol and we used the same version before in the last two SBST contests. We also analyzed the log files produced by the contest platform but we have not found any relevant information to understand this problem. So far, the mysterious problem that prevents JTeXpert to generate test suites for 415 CUTs remains undetermined. We continue our investigation and hope identifying the problem before presenting this paper at the workshop.

Overall, we believe that JTeXpert got a fair rank but with an unfair score. Evosuit [3] deserves the first rank because its team did not stop improving it whereas our engagements and

Table II
AVERAGE COVERAGE AND TOTAL SCORE ACHIEVED BY JTEXPERT ON THE SBST-CONTEST-2017 BENCHMARKS

Benchmark	Class Name	Score	Coverage			Total		
			Mutation	Branch	Line	Mutant	Branch	Line
BCEL-1	org.apache.bcel.classfile.Utility	0	0	0	0	0	344	501
BCEL-10	org.apache.bcel.verifier.structurals.Subroutines	24.99	0.55	0.6	0.68	75	72	119
BCEL-2	org.apache.bcel.verifier.structurals.InstConstraintVisitor	0	0	0.02	0.05	243	776	1076
BCEL-3	org.apache.bcel.generic.ConstantPoolGen	31.49	0.69	0.72	0.85	163	140	280
BCEL-4	org.apache.bcel.generic.InstructionList	0	0	0	0	0	219	431
BCEL-5	org.apache.bcel.verifier.statics.Pass3aVerifier	0	0.01	0	0.04	69	72	120
BCEL-6	org.apache.bcel.verifier.structurals.LocalVariables	28.93	0.66	0.54	0.72	50	54	62
BCEL-7	org.apache.bcel.util.Class2HTML	0	0	0	0	0	45	95
BCEL-8	org.apache.bcel.generic.BranchInstruction	34.23	0.74	0.78	0.8	32	24	61
BCEL-9	org.apache.bcel.classfile.StackMapEntry	24.29	0.51	0.64	0.73	114	196	189
FREEHEP-1	org.freehep.math.minuit.MnPlot	2.42	0.02	0.09	0.13	110.48	116	203
FREEHEP-10	org.freehep.math.minuit.MnUserTransformation	13.98	0.33	0.51	0.56	112	72	169
FREEHEP-2	org.freehep.math.minuit.MnLineSearch	0	0	0.03	0.08	120	94	104
FREEHEP-3	org.freehep.math.minuit.MnFunctionCross	0	0	0.02	0.09	119.43	160	220
FREEHEP-4	org.freehep.math.minuit.MnAlgebraicSymMatrix	6.1	0.08	0.25	0.24	108	126	237
FREEHEP-5	org.freehep.math.minuit.MnUserParameterState	11.07	0.27	0.28	0.49	128.57	110	262
FREEHEP-6	org.freehep.math.minuit.SimplexBuilder	0	0	0.01	0.05	95.24	54	97
FREEHEP-7	org.freehep.math.minuit.MnHesse	0	0.02	0.01	0.1	114.9	48	126
FREEHEP-8	org.freehep.math.minuit.MnPrint	14.39	0.27	0.38	0.44	161.9	72	210
FREEHEP-9	org.freehep.math.minuit.MnMinos	0	0	0.04	0.11	78	74	111
GSON-1	com.google.gson.internal.bind.ReflectiveTypeAdapterFactory	0	0	0	0	0	36	63
GSON-10	com.google.gson.internal.Excluder	0	0	0	0	0	82	89
GSON-2	com.google.gson.internal.LinkedHashTreeMap	0	0	0	0	0	170	234
GSON-3	com.google.gson.JsonPrimitive	0	0	0	0	0	80	82
GSON-4	com.google.gson.stream.JsonReader	0	0	0	0	0	468	656
GSON-5	com.google.gson.internal.LinkedTreeMap	0	0	0	0	0	150	190
GSON-6	com.google.gson.internal.bind.JsonTreeReader	0	0	0	0	0	86	165
GSON-7	com.google.gson.GsonBuilder	0	0	0	0	0	38	94
GSON-9	com.google.gson.reflect.TypeToken	0	0	0	0	0	60	93
IMAGE-1	org.apache.commons.imaging.formats.tiff.write.TiffImageWriterBase	15.24	0.33	0.34	0.5	107	180	304
IMAGE-2	org.apache.commons.imaging.common.RationalNumber	31.4	0.59	0.81	0.89	91.43	60	76
IMAGE-3	org.apache.commons.imaging.formats.bmp.BmpImageParser	0	0	0	0	0	193	380
IMAGE-4	org.apache.commons.imaging.formats.tiff.TiffField	33.18	0.75	0.72	0.77	184	134	253
JXPATH-1	org.apache.commons.jxpath.util.BasicTypeConverter	0	0	0	0	0	298	233
JXPATH-10	org.apache.commons.jxpath.ri.axes.SimplePathInterpreter	0	0	0	0	197	184	271
JXPATH-2	org.apache.commons.jxpath.ri.compiler.Path	8.5	0.2	0.17	0.21	88	86	111
JXPATH-3	org.apache.commons.jxpath.ri.compiler.CoreOperationCompare	24.71	0.52	0.52	0.56	49	64	62
JXPATH-4	org.apache.commons.jxpath.util.MethodLookupUtils	30.47	0.62	0.68	0.71	87.62	102	139
JXPATH-5	org.apache.commons.jxpath.ri.compiler.Step	14.73	0.33	0.28	0.28	42	50	54
JXPATH-6	org.apache.commons.jxpath.JXPathContext	29.97	0.67	0.73	0.81	50	46	96
JXPATH-7	org.apache.commons.jxpath.ri.parser.XPathParserTokenManager	9.3	0.18	0.36	0.49	352	872	1029
JXPATH-8	org.apache.commons.jxpath.util.ValueUtils	18.81	0.46	0.56	0.5	152	150	246
JXPATH-9	org.apache.commons.jxpath.ri.model.beans.PropertyIterator	0	0	0	0	122	98	154
LA4J-1	org.la4j.vector.sparse.CompressedVector	25.44	0.58	0.74	0.78	97.14	126	198
LA4J-10	org.la4j.linear.GaussianSolver	33.61	0.72	0.87	0.91	53	22	37
LA4J-2	org.la4j.decomposition.EigenDecompositor	15.21	0.12	0.58	0.63	241	222	429
LA4J-3	org.la4j.matrix.sparse.CRSMatrix	0.91	0.03	0.05	0.06	18.76	210	339
LA4J-4	org.la4j.matrix.dense.Basic1DMatrix	12.1	0.24	0.36	0.52	112.86	66	116
LA4J-5	org.la4j.Matrix	4.77	0.11	0.32	0.35	124.29	280	520
LA4J-6	org.la4j.linear.ForwardBackSubstitutionSolver	32.46	0.7	0.77	0.87	36	20	26
LA4J-7	org.la4j.matrix.sparse.CCSMatrix	1.56	0.05	0.08	0.09	37.52	210	339
LA4J-8	org.la4j.matrix.dense.Basic2DMatrix	14.55	0.34	0.36	0.51	73.33	68	105
LA4J-9	org.la4j.decomposition.SingularValueDecompositor	20.21	0.11	0.89	0.91	161	175	265
OKHTTP-1	okhttp3.Cookie	0	0.01	0.02	0.05	108	208	236
OKHTTP-2	okhttp3.internal.platform.AndroidPlatform	0	0	0	0	0	30	88
OKHTTP-3	okhttp3.ConnectionSpec	26.88	0.65	0.53	0.66	71	66	82
OKHTTP-4	okhttp3.internal.http.HttpHeaders	22.3	0.5	0.47	0.55	71	62	83
OKHTTP-5	okhttp3.internal.tls.DistinguishedNameParser	7.66	0.16	0.19	0.3	200	168	156
OKHTTP-6	okhttp3.CacheControl	33.79	0.78	0.72	0.84	59	70	128
OKHTTP-7	okhttp3.internal.tls.OkHostnameVerifier	0	0	0	0.05	63	64	80
OKHTTP-8	okhttp3.HttpUrl	0	0	0	0	0	183	221
RE2J-1	com.google.re2j.Parser	14.49	0.12	0.62	0.7	183.67	538.1	723.81
RE2J-2	com.google.re2j.CharClass	33.89	0.71	0.86	0.86	102	112	176
RE2J-3	com.google.re2j.Simplify	14.95	0.11	0.58	0.6	58	56	64
RE2J-4	com.google.re2j.Utils	39.33	0.77	0.93	0.97	117	96	86
RE2J-5	com.google.re2j.Compiler	25.61	0.29	0.85	0.96	103	85	117
RE2J-6	com.google.re2j.Machine	26.67	0.41	0.8	0.91	104	121	159
RE2J-7	com.google.re2j.Regexp	7.47	0.17	0.26	0.35	108	119	164
RE2J-8	com.google.re2j.RE2	26.47	0.34	0.93	0.96	167	98	204
Total/Average		**849**	**28.93%**	**28.22%**	**33.71%**	**123,609**	**210,632**	**312,665**

involvements in other projects have prevented us to maintain JTeXpert during the last two years. We believe that our score could be match better, we spent few days analyzing the results to understand why JTeXpert could not generate any test suite for more than 400 runs. We found that few of them are uncompilable due to a bug in source-code generation but for the vast majority there is nothing in JTeXpert can explain this big number of failures. Many times, we used a manual command line to run JTeXpert on different classes on the contest platform but we could not reproduce the 0% coverage which JTeXpert systematically got during the contest. When we used the contest-platform scripts, we could reproduce the same results, 0% coverage. We believe there is something wrong with the scripts/programs, e.g., in certain conditions a script may remove the test-cases or a bug may prevent a script to continue its execution. It is very frustrating to see this problem unsolved and have not the access to the platform scripts/source to understand and identify the root of this mysterious problem.

V. Conclusion

In this paper, we reported and analyzed the results obtained by JTeXpert in the SBST Contest 2017. JTeXpert performed well compared to its results in the two previous SBST Contests 2016 and 2015. However, the SBST Contest 2017 showed us new bugs in JTeXpert that should be tackled before the next SBST Contest.

Actually, the SBST Contest 2017 offered a new opportunity to test some ideas that we partially implemented in JTeXpert. We also learned, that the current version of JTeXpert still needs further improvements to become a mature and robust software-testing tool.

ACKNOWLEDGEMENT

We would like to thank the SBST Contest organizers, Annibale Panichella and Urko Rueda, for their continuous support in improving our testing tool and identifying new research directions that may make JTeXpert better.

References

[1] Sakti, A., Pesant, G., Guéhéneuc, Y.G.: Instance generator and problem representation to improve object oriented code coverage. IEEE Transactions on Software Engineering **41** (2015) 294–313

[2] Panichella, A., Rueda, U.: Java unit testing tool competition: Fifth round (2017)

[3] Fraser, G., Arcuri, A.: Evosuite: automatic test suite generation for object-oriented software. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ACM (2011) 416–419