2014 9th International Conference on the Quality of Information and Communications Technology

On the Evaluation of Software Maintainability Using Automatic Test Case Generation

Ana Filipa Nogueira
CISUC, University of Coimbra
Pólo II - Pinhal de Marrocos
3030–340 Coimbra, Portugal
afnog@dei.uc.pt

José C. B. Ribeiro
Polytechnic Institute of Leiria
Morro do Lena – Alto do Vieiro
2411–901 Leiria, Portugal
jose.ribeiro@ipleiria.pt

Mário A. Zenha-Rela
CISUC, University of Coimbra
Pólo II - Pinhal de Marrocos
3030–340 Coimbra, Portugal
mzrela@dei.uc.pt

***Abstract*—The measurement of external software attributes and the analysis of how those attributes have evolved through the software's releases are challenging activities. This is particularly evident when we discuss the maintainability of Object-Oriented (OO) systems which, due to their specific characteristics, hide information that cannot be gathered through static analysis. As maintainability can be defined as the "speed and ease with which a program can be corrected or changed", we believe that test data are reflective of the changes performed during maintenance. Moreover, empirical observations allow to speculate about the relationships between maintainability and the behaviour of the software when executed by test cases (e.g., coverage values) and the test cases' characteristics (e.g., generation time). Our aim is to complement the state-of-the art by proposing a new approach for understanding and characterizing the maintainability of OO systems, which makes use of test data and of the information gathered from the tests' execution.**

***Keywords*—Software Maintenance, Software Quality, Software Testing**

I. Introduction

The body of knowledge of Software Measurement encompasses a wide variety of software metrics, frameworks and tools that intend to evaluate different software attributes based on different software artefacts, including: requirements, design artefacts and the source code. External attributes, such as the ones related to quality are of great interest to the people responsible for management activities, who expect to understand the status of a project and also use that information to support their managerial decisions. El Emam [1] states that the estimation of quality attributes means either estimating reliability or maintainability. Moreover, best practices on software development suggest that we should produce code that is simpler, more readable and easier to maintain and modify [2].

According to the SQuaRE standard (ISO/IEC 25000) [3], *maintainability* can be defined as "the degree to which the software product can be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications". This standard also defines that maintainability is characterized by the following sub-characteristics: (i) *modularity* – the degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components; (ii) *reusability* – the degree to which an asset can be used in more than one software system, or in building other assets; (iii) *analysability* – the degree to which the software product can be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified; (iv) *changeability* – the degree to which the software product enables a specified modification to be implemented; the ease with which a software product can be modified; (v) *modification stability* – the degree to which the software product can avoid unexpected effects from modifications of the software; (vi) *testability* – the degree to which the software product enables modified software to be validated; and (vii) *maintainability compliance* – the degree to which the software product adheres to standards or conventions relating to maintainability. Therefore, the improvement of maintainability is inherently related to these sub-characteristics, which should be considered during the software development activities, and taken into account when making important architectural decisions. These sub-characteristics evolve, positively or negatively, throughout the software development cycle.

In our perspective, utilizing test data produced for a software product under test possesses a clear potential for assisting the process of characterizing the maintainability of a software product, as it allows observing the software's behaviour when executed and mirrors the impact of the changes performed to it during the development process. Moreover, we have empirically observed that, typically, the more complex the software is the bigger the test suite is [4]. These premisses entailed the formulation of our main research question: "*is it possible to infer Maintainability properties for a software product with basis on the profile of automatically generated test cases and on the behaviour exhibited by the software product when the test cases are executed?*". In order to answer this question we need to analyse the test case profile of the software product under test.

A *test suite profile* encompasses a set of properties and observations collected by analysing and executing the unit test cases generated for a specific software product; it includes both static and dynamic information that is expected to allow characterizing a software product in terms of its maintainability properties. According to the classification presented in [5], the main techniques for generating unit tests that are considered in our research fall into the *code-based* category, more specifically, techniques that employ criteria based on *control-flow*. Additionally, a random testing tool (a *specification-based* technique) will be used. We intend to analyse the information

978-1-4799-6133-7/14 $31.00 © 2014 IEEE
DOI 10.1109/QUATIC.2014.48

provided by the test suite profiling activity from two different perspectives, namely: (i) a static analysis of the test data's properties in which static metrics obtained from the generated test data will be considered, e.g., number of test cases, number of Lines Of Code (LOC); and (ii) a dynamic analysis of the software product's behaviour which involves the analysis of the behaviour exhibited by the software product during the execution of the test data (e.g., structural coverage).

This paper is organised as follows. Section II overviews the state of the art. Section III presents the set of research objectives and the methodological approach. Section IV discusses the work done so far, while Section V pinpoints the future work. Finally, Section VI concludes this paper.

II. State-of-the-art

Quality models that are estimators or predictors of the quality of software systems with basis on static analysis (static metrics) are widely used and accepted, and in the industry they provide means for quality assessment which, in turn, can support management decisions [6]. From these studies, several focus on the maintainability [7] of the software systems, which is the scope of our research. Nevertheless, the relationships between static metrics gathered from the source code (software under test) and metrics collected from the test data – *test-related* – is still an area with plenty of potential for research. Typically, existing studies focus on the effort required for testing – the software's level of *testability* – which is one of the maintainability sub-characteristics. Despite that, none of them try to get insight on the possible relationship between the software's behaviour exhibited during test data execution, and the software's maintainability properties. Furthermore, most of the existing studies are not interested in using test data as source of information for the characterization of the maintainability evolution of a software product.

Harrison and Samaraweera [8] have investigated whether there is a correlation between the number of test cases, which is a design metric, and the quality of programs, either functional or object-oriented, and the amount of time required to produce them. Bruntink and van Deursen's research tried to investigate factors that can impact the testability of Object-Oriented (OO) systems [9], specifically the testability from the perspective of unit testing of the classes of a Java system. Their research made use of the JUnit framework [1] and their main goals were to determine: (i) if the values of the OO metrics for a class are associated with the required testing effort for that class; and (2) if the values of the OO metrics for a class are associated with the size of the corresponding test suite. The authors found a significant correlation between class-level metrics and test-level metrics. They also concluded that in each group of metrics, the metrics are related to each other. For instance, the larger the test class is, the more assert methods will exist; and the higher the value of the LOC per Class (LOCC) metric, the higher the number of existing methods Number Of Methods (NOM).

Another study that is based on test cases created through the JUnit framework was proposed by Nagappan in his doctoral dissertation [10]. He proposed a suite of nine metrics called Software Testing and Reliability Early Warning (STREW) that evaluate and correlate properties of the code under test, as well as the code generated by the JUnit framework. The STREW metrics were used to build regression models to estimate the post-release field quality using the metric Trouble Reports (TRs)/KLOC from object-oriented projects. Thus, Nagappan's work provides a developer with indications of changes and additions to their automated unit test suite and code for added confidence that product's quality will be high.

Shrivastava and Jain considered the metrics proposed by [11] and defined in [12] an index of testability for a class. Their purpose was to investigate metrics from the perspective related to the design of unit test cases. The result was a set of design metrics, called as Automated Test Case for Unit Testing (ATCUT), designed in order to predict the effort required for unit testing. Lammermann *et al.* [13] have investigated the suitability of structure-based complexity measures for the assessment of evolutionary testability, i.e., the suitability of an object to be subjected to evolutionary testing. Finally, in [14], a technique for predicting the coverage achieved by automatic structural testing was proposed. It uses structural metrics and coverage to train decision tree classifiers. Their study provides an interesting means to better understand how the structure of the program under test can adversely affect the accuracy or performance of the automated testing technique.

Our research is expected to complement the state of the art on correlating test data to software attributes, and to provide a novel approach, by using test-related metrics, to support the analysis of software's maintainability.

III. Research objectives and methodological approach

A. Research Objectives

A set of objectives was defined in order to answer the main question that summarizes our research: "*Is it possible to infer maintainability properties for a software product with basis on the profile of automatically generated test cases and on the behaviour exhibited by the software product when the test cases are executed?*". The list of objectives includes:

- The study of possible metrics to establish the profile of the test cases generated for a specific software product, and of how this profile can be related to software maintainability. Examples of metrics include: number of statements, number of distinct classes instantiated, number of method invocations.
- Determining which software behaviours and properties can be observed by analysing the test code characteristics and the results of its execution.
- Creating a maintainability model based on the outcome of the test data's analysis and execution, based on the maintainability's sub-characteristics specified in the SQuaRE standard [3].
- Defining an ontology that incorporates our maintainability model and specifies how it can be used, in combination with testing tools, to collect maintainability information.
- Implementing a framework that implements the proposed ontology, and validating the framework by

[1] http://www.junit.org/

applying it to different Java projects that are accessible from the open source community (with distinct goals and dimensions), in order to provide empirical evidence for our research.

- Exploring the extent to which the information provided by the test suite profiles and the information collected during the tests' execution can support architectural and product life-cycle decisions. This will be done by comparing our results with historical data collected from maintenance tasks (e.g., refactoring logs, bugs removal).

B. Methodological Approach

The methodological approach starts with a first experimental phase that includes experiments with the tool Evosuite [15], which has been able to generate satisfactory test suites. Currently, test data are being generated and analysed according to the two aforementioned perspectives: i) a static analysis of the test data's properties (e.g., # of test cases, # of LOC); and ii) a dynamic analysis of the software product's behaviour (e.g., branch coverage). After the test data generation process is completed, we will collect all the test-related measures, and determine which properties of the software can be observed by the tests' execution. In turn, that data will be used as input for certain Artificial Intelligence Machine Learning techniques [16] in order to discover patterns among data and determine the relationships between the attributes analysed. Examples of those techniques include clustering, decision trees and classification algorithms.

As the test data generation, execution and parsing processes are conducted, the list of test-related metrics that can compose the test suite profile is built and refined. These activities are part of the next phase "*Definition of the test suite metrics profile*" which includes: i) the identification of the test-related metrics to be considered in the test suite profiling activities; ii) the analysis of which software behaviour is exhibited only during test cases' execution; and iii) the specification of the metrics and corresponding thresholds. The metrics' thresholds represent the upper and lower limits for each metric in order to ensure that the measurements are representative for the maintainability analysis. So far, three categories of test-related metrics were identified (Section IV). In order to compare results from different testing tools, other experimental phases will be conducted, and the test data generated by eCrash [17] and Randoop [18] are going to be compared to the data previously collected and, whenever possible, to the manual test suite provided in each distribution. Experiments were already conducted with the purpose of understanding how these three automated tools behave when generating test data for the same software product [19].

The next main phase in our methodological approach can be described as "*Definition of a maintainability model based on the SQuaRE standard*" and its outcome is a maintainability model that maps the maintainability's sub-characteristics to the metrics included in the test suite profile. This phase includes also the model validation, i.e., it will be verified whether the test-related metrics follow the fundamental rules of measurement theory. The following phase "*Definition of an ontology for maintainability*" aims to create, and to validate, an ontology that specifies not only the concepts and attributes, but also how those concepts and attributes are correlated and how developers can utilize them in their development environments. Accordingly, the next phase "*Development of the tool for maintainability*" consists on: i) developing a tool that implements the proposed ontology; ii) the integration of the maintainability characterization tool with the eCrash test data generation tool; and iii) testing activities. An important final stage of our research comprises the validation of our model, ontology and tool that will require the generation of more test data.

IV. Past work and preliminary results

This section presents a first exploratory study that intended to verify, from a static perspective, if it was possible to identify associations between test-related metrics and static metrics. Thus, this section will briefly describe the steps performed in that exploratory study.

A. Experiment Setup and Data Generation Process

The experiment was conducted on 1434 classes from 26 open source projects from the Apache Software Foundation[2], implemented in the Java programming language. The instruments utilised include: *Evosuite* – a code-based automated testing tool that was responsible for generating the test data; and *CKJM tool* [20] – an open source tool for collecting the static metrics from the source code. Evosuite was the selected testing tool for our initial experiments due to the satisfactory results achieved while generating test data for several projects [19], [21], [22]. The CKJM tool was selected because it has been used to support research related to software quality, namely: it was originally developed by Diomidis Spinellis to support his work on the evaluation of the quality of open source systems [23]; and it is used by other quality-related projects (for instance, it is part of the Isotrol Metric Analytics plugin for the Sonar open source quality management platform [3]).

For each project, the test suites generated for its concrete classes were analysed; the definition of the sample used in our experiment – i.e., the selection of the classes to study – respected the following two requirements: (i) only non-abstract classes can be included; and (ii) classes that were not successfully tested must be excluded (the remaining classes will be addressed in future work). For each class, 10 runs were performed by the testing tool; this number of runs is justified by the need to avoid bias from the randomized component of the tool [24].

B. Hypothesis formulation

The research question that motivated this exploratory study was:

Question: "Is there an association between each pair of metrics <*ck-metric*$_i$, *t-metric*$_j$>measured for a class, in which *ck-metric*$_i$ is a static metric collected from the source code, and *t-metric*$_j$ is a test-related metric collected from the test suite generated by the Evosuite tool? ".

[2]http://www.apache.org/foundation/

[3]http://www.sonarqube.org/

TABLE I. CK METRICS SUITE (*adapted from* [11]).

Metric	Description
WMC	is the sum of the complexity of the methods of a class. The tool utilised in this experiment uses the unity approach – each method has the complexity weight "1" – making it equivalent to NOM.
DIT	the maximum length from the node to the root of the hierarchy tree.
NOC	number of immediate subclasses subordinated to a class in the class hierarchy.
CBO	is a count of the number of other classes to which it is coupled.
RFC	is the cardinality of the set of all methods that can be invoked in response to a message to an object of the class or by some method in the class.
LCOM	is the number of pairs of methods in a class which do not reference a common attribute.

To address the aforementioned question, the following hypotheses were derived:

H0 (i,j) There is *no* association between the CK metric *ck-metric*$_i$ and the test-related metric *t-metric*$_j$.

H1 (i,j) There is an association between the CK metric *ck-metric*$_i$ and the test-related metric *t-metric*$_j$ where *i* can be any of the metrics from the CK suite, and *j* refers to each test-related metric.

Both metric suites will be presented in the following subsection.

C. CK Metrics and Test-Related Metrics

The Chidamber & Kemerer (CK) suite includes six metrics that measure the cohesion, complexity, coupling and inheritance of OO systems (Table I), namely: Weighted Methods Per Class (WMC), Depth of inheritance (DIT), Number of Children (NOC), Coupling Between Objects (CBO), Response For Class (RFC) and Lack of Cohesion in Methods (LCOM).

Currently, the proposed set of test-related metrics[4], which refer to each class under test, encompasses a total of 9 metrics, classified according to three categories (Table II): *Coverage-based metrics* – includes metrics related to the degree of coverage achieved by the test code; *Time-based metrics* – defines metrics related to the amount of time required to reach an acceptable test suite for the class that is being tested; and *Size-based metrics* – includes metrics related to the test suite's size, i.e., the number of test methods included into the test class, and the number of LOC generated[5], both before and after the minimisation process.

D. Results and Statistical Analysis

The investigation of the hypotheses proposed earlier requires the choice of the method for statistical analysis. As both metric suites comprise metrics that are either of the types: interval or ratio; and as is not possible to assume any type of data distribution, a non-parametric technique was selected – the Spearman's correlation coefficient [26]. The Spearman's rank-order correlation coefficient (r_s) measures the degree and strength of a relationship between two or more variables. The numerical coefficient ranges between -1 and 1, in which: $r_s > 0$ implies positive association; $r_s < 0$ implies negative association (or association in the reverse direction); and $r_s = 0$ implies no association. The higher the r_s (the absolute value), the stronger the relationship between the two variables; in this paper, the strength of the relationship (with basis on the absolute value) was interpreted according to the following categories [26]: ***very weak*** (from 0.000 to 0.200), ***weak*** (from 0.201 to 0.400), ***moderate*** (from 0.401 to 0.600), ***strong*** (from 0.601 to 0.800) and ***very strong*** (from 0.801 to 1.000).

TABLE II. TEST-RELATED METRICS.

Group	Metric	Description
Coverage	BrCv	Branch Coverage – percentage of branches covered by the test code.
	MtCv	Method Coverage – percentage of methods covered by the test code.
	Cv	Coverage – percentage of the class coverage achieved by the test code.
Time	GnT	Generation Time – the amount of time required to generate a test suite.
	MnT	Minimisation Time – the amount of time required to minimise the test suite; the minimisation process includes the removal of redundant test code and the creation of test code that is more readable [25].
Size	RsLen	Result Length – number of LOC for the test suite.
	RsSize	Result Size – number of test methods included in the test suite.
	MnRsLen	Minimised Result Size – number of LOC for the minimised test suite.
	MnRsSize	Minimised Result Length – number of test methods included in the minimised the test suite.

Table III depicts the values for the Spearman's coefficient correlation *rs* for each pair of metrics: <*ck-metric*$_i$, *t-metric*$_j$ >; the significant associations are identified by "**" and the significance level *p* considered was the 0.01[6]. For the significant associations we can reject the H_0(i,j) and accept H_1(i,j) – there is some association between the CK metric *i* and the test-related metric *j*; the strength of these associations is discussed next. On the other hand, 7 pairs of metrics, among a total of 54, have no significant association between them, and have the *null* hypothesis accepted.

E. Discussion

The Spearman's correlation determines a coefficient of correlation between variables; meaning that, one variable is associated to another, and that association has a strength that can be used by researchers to study certain software properties. However, the association does not represent a *causation-effect* relationship, and the conclusions that can be drawn from these associations must be made carefully. Hereupon, we observed some "strong" and "moderate" associations between static metrics and test-related metrics; however, we cannot make statements such as: *classes with higher WMC achieve best values for branch coverage*. We adopted a different perspective: *the associations allow us to conclude that is possible to observe certain maintainability indicators (with basis on the static metrics) through the data collected from the test data*

[4]These metrics were collected from the statistics provided by the automated testing tool.

[5]The number of LOC does not include code related to: assertions and exceptions' handling.

[6]The *p* provides a certain level of confidence when rejecting the *null* hypothesis H_0 and accepting H_1.

TABLE III. RESULTS FOR THE SPEARMAN'S CORRELATION COEFFICIENT.

	t-metric$_j$/ck-metric$_i$	WMC	DIT	NOC	CBO	RFC	LCOM
Coverage	BrCv	.343**	-.254**	.065	-.118**	.314**	.166**
	MtCv	-.241**	-.024	-.038	-.174**	-.300**	-.173**
	Cv	-.355**	.084**	-.042	-.183**	-.531**	-.251**
Time	GnT	.584**	-.123**	.097**	.220**	.478**	.387**
	MnT	.346**	.018	.083**	.331**	.513**	.283**
Size	RsSize	.600**	-.010	.122**	.219**	.538**	.468**
	RsLen	.377**	.080**	.058	.225**	.270**	.295**
	MnRsSize	.787**	-.253**	.160**	.135**	.659**	.545**
	MnRsLen	.730**	-.182**	.164**	.190**	.646**	.485**

Note: ** Correlation is significant at the 0.01 level (2-tailed).

generation and execution. These findings are more evident for the static metrics WMC, RFC and LCOM. The results for CBO did not meet our expectations, i.e., strong associations were not found; and the associations with inheritance-based metrics proved to be "weak" and "very weak".

In this experiment, the WMC is equivalent to NOM, meaning that methods' complexity was not really associated to the test-related metrics; although the current findings are still relevant, we believe that a more interesting approach comprises the utilisation of weights based on the effective method's complexity. RFC and LCOM were strongly criticised [27]: (i) the definition of RFC is not clear and originates different implementations and interpretations, which makes the metric unreliable for comparison; and (ii) LCOM has different versions – some were intended to correct and/or replace previous faulty versions, whereas others follow different approaches. Nonetheless, the CKJM tool performs the measurements according to the principles described in [11]; thus, we can claim that our results are in conformance with the original metrics definition.

According to [27], CBO presents some issues related to its ability to accurately predict the actual amount of coupling; its argued that static analysis does not capture certain OO specificities – e.g., polymorphism, dynamic binding; the evaluation of these features is more precise during *runtime* which requires the utilisation of dynamic coupling metrics; therefore, the analysis of these metrics provide strong evidences on how coupling affects software quality.

The associations between DIT and test-related metrics need to be further investigated in order to better understand the impact of two aspects: (i) the type of inheritance (*extends* and *implements*), and (ii) the origin of the classes' ancestors (user-defined, environment-dependent, and third-party components). In our opinion, the type of inheritance and the ancestors' provenance can provide insightful information about the issues associated to the classes that inherited them, and one good way of revealing those issues is by analysing the coverage values achieved during the software testing activities. Also, the analysis of the impact of inherited and overridden methods on the achievement of the test goals (e.g., branch coverage) is imperative. Contrary to DIT, the scenarios in which NOC is associated to test-related metrics are not trivial.

V. FUTURE WORK AND EXPECTED RESULTS

Future work includes four major milestones:

- Milestone 1: Test Suite Profile Specification – publication of a specification that includes a set of test-related metrics that are collected from the test procedures and from the test cases.
- Milestone 2: Maintainability Model – definition and validation of a maintainability model that maps the test suite profile to maintainability properties of the software under analysis.
- Milestone 3: Ontology – definition and validation of an ontology that specifies the concepts and rules that should be followed by developers that desire to use the proposed maintainability model.
- Milestone 4: Tool deployment – development of a tool or plug-in that implements the proposed ontology, and that is integrated with the eCrash testing tool.

Future work includes the implementation of the aforementioned milestones, and also addresses some issues that represent threats to validity, and which were identified during the experiments described in Section IV. For instance, threats on *conclusion validity* may be related to the statistical methods utilised, and the size of the samples used. Threats to *internal validity* might come from the testing tools utilised, which typically include a random component that affects the algorithm used to generate the test data. Threats to *external validity* are common in this type of research and are related to the ability to generalise the results to other environments; this is also applicable to our research: although we have selected 26 projects – and studied a total of 1434 classes – the results cannot be generalised to the reality of the open source community. At least, we can try to find common patterns among the projects deployed in the Apache Software Foundation. Moreover, the results may only reflect associations that occur in code developed according to the OO paradigm and implemented in the Java programming language.

VI. CONCLUSION

Our research focuses on identifying maintainability issues – related to OO systems – that are not always obvious through static analysis and that are only noticeable through the observation of the system's behaviour. Thus, the use of metrics related to unit tests – both from their generation and their execution – was the natural choice, in terms of the dynamic analysis technique that would allow us to observe the desired behaviours.

Currently, there are studies that correlate software attributes to test data (Section II), however, they typically rely on manual testing and on trouble reports. Our approach differs from others because it uses information gathered from the test data

generation and execution processes. The test-related metrics proposed in our experiment are not limited to the number of LOC and to the number of asserts, they belong to different categories, e.g.: coverage, time and size.

In terms of the impact of our results (Section IV), we think that our findings should be regarded as exploratory and indicative rather than conclusive, since they show that is possible to find significant associations between source code (through some static metrics commonly used to evaluate maintainability) and the corresponding test code, but still they are not generalisable. Nevertheless, the main contribution of those results is on the fact that we were able to prove that tests are capable of providing information (even if indicative) about maintainability.

Future work includes (among other objectives): the utilisation of different automated testing tools and the inclusion of more classes into the dataset; the analysis of the classes that were not successfully tested in order to understand its characteristics; the specification of additional test-related metrics; the analysis of more maintainability-related metrics (i.e., other static metrics); the application of different statistical methods; the definition and validation of a maintainability model; the definition and validation of an ontology for our research; and finally, the implementation of a prototype for validation of the results.

ACKNOWLEDGMENT

This material is based upon work supported by the QREN 'Programa Operacional Regional do Centro' under Grant CENTRO-07-ST24-FEDER-002003, project ICIS - Intelligent Computing in the Internet of Services.

REFERENCES

[1] K. E. Emam, "A primer on object-oriented measurement," in *Proc. of the 7th Int. Symposium on Software Metrics*, ser. METRICS '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 185–187. [Online]. Available: http://dl.acm.org/citation.cfm?id=823456.824020

[2] Abran, A., Bourque, P., Dupuis, R., Moore, J. W., and Tripp, L. L., *Guide to the Software Engineering Body of Knowledge - SWEBOK*. Piscataway, NJ, USA: IEEE Press, 2004.

[3] ISO/IEC, "ISO/IEC 25000 - Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE," ISO/IEC, Tech. Rep., 2005.

[4] J. C. B. Ribeiro, M. A. Zenha-Rela, and F. Fernández de Vega, "Test case evaluation and input domain reduction strategies for the evolutionary testing of object-oriented software," *Inf. Softw. Technol.*, vol. 51, no. 11, pp. 1534–1548, Nov. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2009.06.009

[5] S. Vegas, N. Juristo, and V. R. Basili, "Maturing software engineering knowledge through classifications: A case study on unit testing techniques," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 551–565, 2009.

[6] R. Lincke, T. Gutzmann, and W. Löwe, "Software quality prediction models compared," in *Proc. of the 10th International Conference on Quality Software*, ser. QSIC '10. IEEE Computer Society, 2010, pp. 82–91.

[7] M. Riaz, E. Mendes, and E. Tempero, "A systematic review of software maintainability prediction and metrics," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 367–377. [Online]. Available: http://dx.doi.org/10.1109/ESEM.2009.5314233

[8] R. Harrison and L. G. Samaraweera, "Using test case metrics to predict code quality and effort," *SIGSOFT Softw. Eng. Notes*, vol. 21, no. 5, pp. 78–88, Sep. 1996.

[9] M. Bruntink and A. van Deursen, "An empirical study into class testability," *J. Syst. Softw.*, vol. 79, no. 9, pp. 1219–1232, Sep. 2006.

[10] N. Nagappan, "A software testing and reliability early warning (strew) metric suite," Ph.D. dissertation, North Carolina State University, 2005.

[11] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, Jun. 1994. [Online]. Available: http://dx.doi.org/10.1109/32.295895

[12] D. P. Shrivastava and R. Jain, "Metrics for Test Case Design in Test Driven Development," *Int. Journal of Computer Theory and Engineering*, vol. 2, no. 6, pp. 952–956, 2010.

[13] F. Lammermann, A. Baresel, and J. Wegener, "Evaluating evolutionary testability for structure-oriented testing with software measurements," *Appl. Soft Comput.*, vol. 8, no. 2, pp. 1018–1028, Mar. 2008.

[14] B. Daniel and M. Boshernitsan, "Predicting effectiveness of automatic testing tools," in *Proc. of the 2008 23rd IEEE/ACM Int. Conference on Automated Software Engineering*, ser. ASE '08. IEEE Computer Society, 2008, pp. 363–366.

[15] G. Fraser and A. Arcuri. (2011) Evosuite – automatic test suite generation for java. [Online]. Available: http://www.evosuite.org/

[16] C. McDonald, "Machine learning: a survey of current techniques," *Artificial Intelligence Review*, vol. 3, no. 4, pp. 243–280, 1989. [Online]. Available: http://dx.doi.org/10.1007/BF00141197

[17] J. C. B. Ribeiro, "Search-based test case generation for object-oriented java software using strongly-typed genetic programming," in *Proceedings of the 10th Annual Conference Companion on Genetic and Evolutionary Computation*, ser. GECCO '08. New York, NY, USA: ACM, 2008, pp. 1819–1822. [Online]. Available: http://doi.acm.org/10.1145/1388969.1388979

[18] 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*, ser. OOPSLA '07. New York, NY, USA: ACM, 2007, pp. 815–816. [Online]. Available: http://doi.acm.org/10.1145/1297846.1297902

[19] A. F. Nogueira, J. C. B. Ribeiro, F. F. de Vega, and M. A. Zenha-Rela, "ecrash: An empirical study on the apache ant project," in *Proc. of the 5th Int. Symposium on Search Based Software Engineering (SSBSE '13)*, vol. 8084. St. Petersburg, Russia: Springer, 24-26 August 2013.

[20] D. Spinellis. (2005) ckjm ck java metrics. Accessed Nov. 25, 2013. [Online]. Available: http://www.spinellis.gr/sw/ckjm/

[21] G. Fraser and A. Arcuri, "Evosuite at the sbst 2013 tool competition," in *6th International Workshop on Search-Based Software Testing (SBST'13) at ICST'13*, 2013.

[22] ——, "Sound empirical evidence in software testing," in *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*. IEEE, 2012, pp. 178–188.

[23] D. Spinellis, *Code Quality: The Open Source Perspective (Effective Software Development Series)*. Addison-Wesley Professional, 2006.

[24] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proc. of the 33rd Int. Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 1–10. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985795

[25] G. Fraser and A. Arcuri, "Evosuite: On the challenges of test case generation in the real world," *Software Testing, Verification, and Validation, 2008 International Conference on*, vol. 0, pp. 362–369, 2013.

[26] E. Christmann and J. Badgett, *Interpreting Assessment Data: Statistical Techniques You Can Use*. NSTA Press, 2009. [Online]. Available: http://books.google.pt/books?id=3lUyYvGQsdYC

[27] F. Balmas, A. Bergel, S. Denier, S. Ducasse, J. Laval, K. Mordal-Manet, H. Abdeen, and F. Bellingard. (2009) Software metrics for java and c++ practices. [Online]. Available: http://www.squale.org/quality-models-site/research-deliverables/WP1.1_Software-metrics-for-Java-and-Cpp-practices_v1.pdf