Using predictive models to evaluate the quality of a test suite at class and method level.

Keslley Silva (Student)[1], **Érika Cota (Advisor)**[1]

[1]Master's in Computer Science
Postgraduate Program in Computing (PPGC) – Federal University of Rio Grande do Sul (UFRGS) – Porto Alegre – RS – Brazil
Entry Year: 03/2019 – Qualification: 11/2019 – Expected conclusion: 02/2021

{keslley.silva, erika.cota}@inf.ufrgs.br

Abstract. *Testing is an indispensable part of the software development process and is a continuous process during the development life cycle. In this context, examining the behavior of software systems to reveal potential problems is a crucial task. To this end, the test suites usually are utilized to examine the software quality. However, test suite quality control is hard for the tester, especially in an evolving system. Such control is needed to assure and improve the test suite's quality and the application as a consequence. Currently, test coverage criteria are used as a mechanism to assist the tester in analyzing the test suite (e.g., find the weaknesses, and add a new test case or test inputs). However, more strong coverage criteria (potentially showing less glaring weaknesses) are challenging to assess. In this work, we propose a different approach to support the developer in evaluating the test suite quality based on more powerful test coverage criteria. We will follow the Knowledge Discovery in Database process using machine learning algorithms to estimate the prime path coverage at the method and class level. For this purpose, we will create two large datasets consisting of source code metrics and test case metrics from 12 open-source Java projects, and these datasets will be used in the training process to build the predictive models. Using the built models, we expected to predict the prime path coverage at the method and class level with a reliable prediction performance.*

Keywords: *Software testing, Coverage prediction, Code coverage criteria.*

CBSoft Events: *SBES and SBCARS.*

1. Problem Characterization

Software is developed and maintained to be used in a wide diversity of circumstances and different configurations. Consequently, the software's size and complexity increased, and software development became more valuable in providing useful solutions. In this context, software testing activities play an indispensable role in identifying problems and ensuring the software's quality and acceptability [Jorgensen 2013]. However, the tester needs to know the test suites' power, since a strong-power suite can detect more bugs than a weak-power one [Zhang et al. 2019]. In the literature, test coverage criteria became widely adopted as a representative of this power.

A test coverage criterion is a rule or group of rules that describes the test requirements (TR); each requirement is a specific software element that a test case must satisfy or cover [Ammann and Offutt 2016]. The widely used criteria are node and edge coverage, which are graph-based, and many current testing tools measure it. These two criteria generate the minimum TR for a test suite. In contrast, other criteria can explore more sophisticated uses of the software under test (SUT), e.g., criteria that cover paths.

Test coverage criteria can help the team improve the test suite's power through rules for when to stop testing and support to find weaknesses and redundancies. However, there are difficulties related to the estimation of more powerful criteria because current tools do not support them. This automation has been restricted to node and edge coverage. Besides, the manual execution of this process is typically unworkable. To give an example, to measure the graph-based coverage of a test suite, one needs to get the graph that describes the SUT, later derive the test requirements for a specific criterion, and then trace the execution paths on this graph given by each test case. Finally, the TR covered by the traced execution paths are counted, and the coverage for that criterion can be calculated. Although the procedure described above can be implemented without significant technical challenges, technological difficulties hinder the creation and maintenance of these tools.

In order to mitigate the challenges of technology-dependent tools and help the tester improving test suites, recent works have proposed the estimation of the coverage value rather than its precise calculation. With a similar purpose, the focus of this Master's project is to predict the prime path coverage (PPC) of a test suite at class and method level using regression analysis and the knowledge discovery in database (KDD) process.

We selected this criterion because 1) it subsumes most graph-based criteria (including data-flow ones) [Ammann and Offutt 2016]. Indeed, in practical scenarios, it shows more effective when confronted with edge coverage (EC) and edge-pair coverage (EPC), especially in programs that have complicated control flows [Durelli et al. 2018]. Consequently, this criterion supports the tester to examine the power of the test suite with more convinced; 2) nowadays, there are no working tools to support the tester to evaluate the test suite based on this criterion.

2. Background

2.1. Test coverage criteria

The literature introduces diverse testing coverage criteria, but graph-based ones are assuredly the most popular. Graph-based coverage criteria are subdivided into control

flow and data flow. As mentioned, this work focuses on prime path coverage, a control flow coverage criterion. The PPC criterion is based on the definition of a simple path. A simple path is a path in which no node appears more than once (i.e., no internal loops) and only starting and ending nodes can appear more than once [Ammann and Offutt 2016]. A prime path is a simple path that is not a sub-path of any other simple path [Li et al. 2009]. Consequently, the set of test requirements for PPC is composed of all prime paths of the directed graph. When applied to source-code, the directed graph is called Control Flow Graph (CFG), where a node represents an integral piece of code (basic block), and an edge represents a possible control flow between two nodes.

The test requirements generated by the PPC criterion represent less trivial uses of the SUT and are mainly attractive in high complexity SUTs. According to [Ammann and Offutt 2016], in practical situations, PPC criterion subsumes node and edge coverage criteria and data-flow criteria. It is subsumed only by the complete path coverage (CPC) criterion, which is the strongest in graph coverage and defines all possible paths of the graph as test requirements, being impossible to achieve in most cases. An important consideration is that the PPC does not subsume edge-pair coverage (EPC) due to the case when a node has a self-loop.

2.2. Knowledge Discovery in Database

Knowledge Discovery in Database (KDD) techniques can be used to support exploring and analyzing new data and for analyzing old types of data in novel forms. The KDD is a process of converting raw data into useful information, divided into three indispensable phases: data pre-processing, data mining, and post-processing [Tan et al. 2005]. The data pre-processing is an indispensable activity that will help improve the input data's quality and reliability and get the mining results. For this purpose, this first step focuses on structure data into an appropriate format by gathering information from the different accessible sources (raw input data). This step is possibly the most challenging and time-consuming in the overall KDD process [Tan et al. 2005].

As a multidisciplinary field, the data mining phase uses many areas; one of these is machine learning (ML), which refers to the automated detection of meaningful patterns in data [Kirk 2014]. A primary notion in ML is the difference between supervised and unsupervised learning. There is no distinction between training and test data in unsupervised learning, whereas, in supervised learning, there are examples for the learner (training process). Data mining tasks are divided into predictive tasks and descriptive tasks. The goal of the predictive task is to use supervised learning to predict a value (dependent variable) based on the values of other attributes (independent variables). In descriptive tasks, the goal is to use unsupervised learning to derive data patterns, such as clusters, anomalies, and relationships.

The classification and regression make up the predictive tasks, in which the first is applied to predict categorical labels, and the second is widely performed for numeric prediction. We conduct a regression analysis in this Master's project because we want to predict a test suite's PPC value based on a set of numeric metrics. Finally, the post-processing phase ensures that only legitimate and useful results are incorporated and used to present knowledge to users. This step's challenge is enabling anyone to absorb large amounts of visual information and find patterns in it.

3. Methodology

This research supports the tester to evaluate the quality of a test suite. We base our solution on machine learning algorithms, using regression analysis algorithms to build a predictive model based on source code metrics and basic test criteria. To evaluate the power of a test suite, we used the PPC criterion, as justified in Section 1. For a clear understanding, we summarized the phases to develop this study in Figure 1.

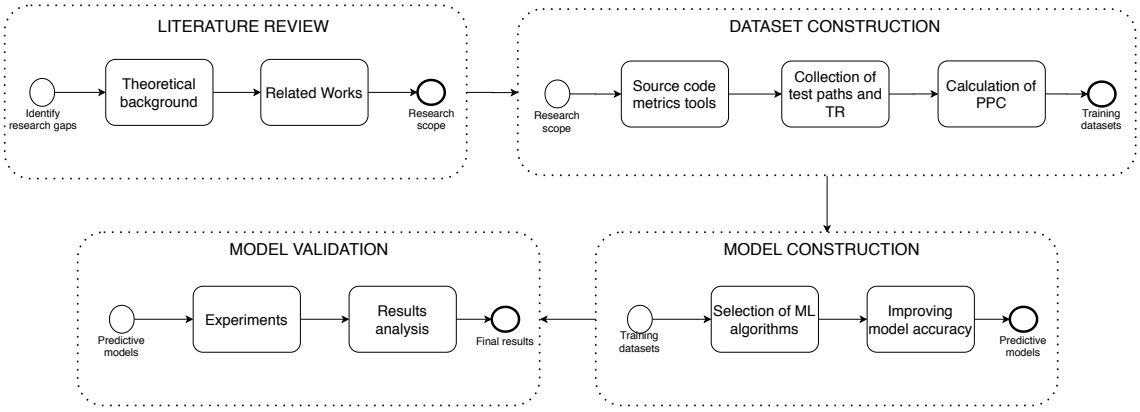

Figure 1. Methodology overview.

Literature Review. This step aims to identify the gaps in the literature. First, we searched for studies that proposed a solution concerning software testing and machine learning. This phase's findings allowed us to collect related works and construct motivation for this project. After searching for works that apply machine learning to support the tester in evaluating or improving the test suite's quality, we inferred that there was an opportunity to try to predict the value of more robust test coverage criteria and support the tester. We looked for tools that generated the exact value of more strong criteria, but we just found PESTT [Gameiro and Martins 2012], which is currently not working.

As a result of the literature review efforts, we defined as our primary motivation to investigate the use of more powerful criteria (in this work, PPC) because few studies propose solutions to predict or calculate these criteria. We believe there are two reasons for that: 1) few studies show practical evidence about the advantages of using strong criteria and 2) testers assume weaker criteria (the ones supported by available tools) are enough and, therefore, the culture for its usage is not created.

Dataset Construction. Supervised learning algorithms are applied to predict a specific output (dependent variable) given inputs (independent variables), learning from input/output pairs. In our approach, we want to explore the relationship of a single dependent variable, the PPC, to several source code metric and edge coverage value. To select the independent variables, we adopted three criteria: 1) should be obtained from real-world projects; 2) must be fairly easy to extract; and 3) must represent factors that describe the code at the method level.

At first, we selected working tools that generated metrics from the source code at the class or method level. To this end, we conducted our search based on the systematic mapping study conducted by Nunuez-Varela et al. in [Varela et al. 2017], in which they

present almost 300 source code metrics and various tools, collected from 226 studies. Based on these tools, we determine the set of metrics that will be used to create the datasets.

For the method level, 23 source code metrics were selected and collected using the Understand tool[1] for both test and application methods. Concerning the class-level metrics, we selected 62 metrics that may have a relationship with the PPC of a test suite. To collect these metrics, we applied the Understand tool and Jnose [Virgínio et al. 2019]. We also selected the edge coverage as a basic test coverage metric for both datasets due to the vast majority of available tools that calculated it. All the defined metrics in class-level and method-level were obtained from 12 open-source Java projects of different sizes and domains. Table 1 summarizes the necessary information about the used projects.

Table 1. Summary of selected open-source Java projects.

Project	Version	LOC	# Class	# Method
Apache Dubbo	2.7.8	179.495	3.110	20.223
Jfreechart	1.5.0	134.540	1.039	11.146
Apache Commons Math	3.6.1	220.851	2.631	16.696
Apache Kylin	3.1.0	262.596	3.293	24.005
Apache Commons Lang	3.11	88.485	955	9.529
Biojava	5.4.0	184.935	1.657	14.519
Apache ServiceComb Java Chassis	2.1.1	163.933	3.703	17.178
Apache Commons Text	1.9	26.466	259	2.638
Apache Commons IO	2.6	32.345	312	2.741
Apache Submarine	0.4.0	51.971	510	3.256
Apache Commons Collections	4.4	66.910	907	7.384
Guava	29.0	511.607	11.887	63.142

To generate the test requirements, we preferred to use the web application implemented by Wuzhi Xu at el. called Graph Coverage[2] because it is widely applied in the literature. However, to collect the test paths and calculate the PPC value, we will need to develop an in-house tool that automates these activities, making it possible to collect many instances for the dataset. Finally, we will carry out the KDD process's pre-processing data phase to generate the training data.

Model Construction. Based on our resulting dataset, we will study and apply different regression analysis algorithms to generate the best predictive model. Combined with the selection of models, if necessary, we will look for ways to improve the model's prediction performance, conducting activities such as hyper-parameter optimization. As a result of this phase, we expect to generate the best model to predict PPC at the method level and another best model to predict PPC at the class level.

Model Validation. To measure the predictive model performance, we will apply the K-fold cross-validation [Kohavi 1995]. To measure the error, we will use well-known metrics, such as Root-Mean-Square Error (RMSE), Mean Absolute Error (MAE), and R-

[1] `https://scitools.com/`

[2] `https://cs.gmu.edu:8443/offutt/coverage/GraphCoverage`

Squared. Second, we will examine and explain the relationship between the dependent and independent variables. After that, we will conduct experiments to simulate how the regression model performs in predicting PPC values under a practical scenario, i.e., predicting the PPC of the test suite from unknown systems. Based on the results of the experiments, we will be able to examine the feasibility of using our predictive models as a tool to assist the tester in assessing the quality of a test suite in class and method level.

4. Contributions

This project aims at contributing to the software testing community in the following ways:

- Support the tester to evaluate the power of a test suite by using a prediction model and a more strong criterion;
- Understand the relationship between source code metrics and PPC of a test suite at the class and method level;
- Validate the approach by performing experiments to evaluate the predictive model in a realistic scenario;
- Provide a public dataset that involves PPC criterion in open source projects, making it available online.

5. Related Work

Durelli et al. [Durelli et al. 2019] gives a systematic mapping of the use of machine learning in software testing and report that the most investigated topics are on test case design, test oracle construction, test case evaluation, and test case refinement. A few authors have proposed the use of ML to estimate the mutation score of a test suite using different source data. Jalbert and Bradbury [Jalbert and Bradbury 2012] present an approach to predict the mutation score of a unit under test based on a combination of source code, test suite metrics, and node coverage information. Their model can only predict mutation scores in a cross-version scenario (new release of the same project used during the training phase).

With a similar goal, Zhang et al. [Zhang et al. 2019] introduce a proposal called Predictive Mutation Testing (PMT), which uses the Random Forest algorithm to build a classification model. The model can predict whether any new mutant is killed or not based on the same set of features used in the training phase. PMT is also used to predict the mutation score of the project based on the proportion of the mutants it predicts will be killed. The work most similar to the one we are developing is Grano et al. [Grano et al. 2019], they proposed the use of regression analysis to predict the branch (edge) coverage of an automatically generated test set at the class level.

Acknowledgements

This work is partially supported by CNPq, Brazil. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

References

Ammann, P. and Offutt, J. (2016). *Introduction to Software Testing*. Cambridge University Press, 2 edition.

Durelli, V. H., Delamaro, M. E., and Offutt, J. (2018). An experimental comparison of edge, edge-pair, and prime path criteria. *Science of Computer Programming*, 152:99 – 115.

Durelli, V. H. S., Durelli, R. S., Borges, S. S., Endo, A. T., Eler, M. M., Dias, D. R. C., and Guimarães, M. P. (2019). Machine learning applied to software testing: A systematic mapping study. *IEEE Transactions on Reliability*, 68(3):1189–1212.

Gameiro, R. and Martins, F. (2012). Pestt educational software testing tool. Master's thesis, LaSIGE & University of Lisbon, Faculty of Sciences.

Grano, G., Titov, T. V., Panichella, S., and Gall, H. C. (2019). Branch coverage prediction in automated testing. *Journal of Software: Evolution and Process*.

Jalbert, K. and Bradbury, J. S. (2012). Predicting mutation score using source code and test suite metrics. In *Proceedings of the First International Workshop on Realizing AI Synergies in Software Engineering*, RAISE '12, pages 42–46, Piscataway, NJ, USA. IEEE Press.

Jorgensen, P. C. (2013). *Software Testing: A Craftsman's Approach, Fourth Edition*. Auerbach Publications, fourth edition.

Kirk, M. (2014). *Thoughtful machine learning: A test-driven approach*. " O'Reilly Media, Inc.".

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Li, N., Praphamontripong, U., and Offutt, J. (2009). An experimental comparison of four unit test criteria: Mutation, edge-pair, all-uses and prime path coverage. In *2009 International Conference on Software Testing, Verification, and Validation Workshops*, pages 220–229.

Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Varela, A., Perez-Gonzalez, H., Martinez, F., and Soubervielle-Montalvo, C. (2017). Source code metrics: A systematic mapping study. *Journal of Systems and Software*, 128.

Virgínio, T., Santana, R., Martins, L. A., Soares, L. R., Costa, H., and Machado, I. (2019). On the influence of test smells on test coverage. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, SBES 2019, page 467–471, New York, NY, USA. Association for Computing Machinery.

Zhang, J., Zhang, L., Harman, M., Hao, D., Jia, Y., and Zhang, L. (2019). Predictive mutation testing. *IEEE Transactions on Software Engineering*, 45(9):898–918.