VITRuM - A Plug-In for the Visualization of Test-Related Metrics

Fabiano Pecorelli
SeSa Lab - University of Salerno, Italy
fpecorelli@unisa.it

Gianluca di Lillo
SeSa Lab - University of Salerno, Italy
g.dilillo1@studenti.unisa.it

Fabio Palomba
SeSa Lab - University of Salerno, Italy
fpalomba@unisa.it

Andrea De Lucia
SeSa Lab - University of Salerno, Italy
adelucia@unisa.it

ABSTRACT

Software testing is the first weapon against software faults, used by developers to preventively locate implementation errors in the exercised production code that may cause critical failures to the inner-working of software systems. According to recent findings, the effectiveness of testing might be not only due to its ability to cover the production code but also to some other properties, like code quality. Among other aspects, the literature reported that an advanced visualization of test-related metrics, e.g., test code coverage on production code, result to be a key strength for developers when dealing with software faults. In this paper, we propose VITRuM (**VI**sualization of **T**est-**R**elated **M**etrics), an *IntelliJ* plug-in able to provide developers with an advanced visual interface of both static and dynamic test-related metrics that has the potential of making them more able to diagnose production code faults. The plug-in is available in the official JetBrains Plugins Repository. A video showing the tool in action is available at https://youtu.be/kFE81eYPgUg.

KEYWORDS

Software Testing; Advanced Visual Interfaces; Test Code Quality.

ACM Reference Format:
Fabiano Pecorelli, Gianluca di Lillo, Fabio Palomba, and Andrea De Lucia. 2020. VITRuM - A Plug-In for the Visualization of Test-Related Metrics. In *International Conference on Advanced Visual Interfaces (AVI '20), September 28-October 2, 2020, Salerno, Italy*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3399715.3399954

1 INTRODUCTION

Pressure, continuous changes, strict deadlines: these reasons often enforce developers to deliver low-quality software and to rely on testing to verify the compliance of software to predefined requirements [12]. While testing has been often identified as a way to improve software quality and reliability [3, 8, 14], researchers have pointed out that its effectiveness heavily depends on how test code-related metrics are made actionable to developers [4, 6, 7]. Particularly, Jones et al. [7] reported that visualization techniques can effectively display to developers a large amounts of data that can assist debugging activities; later on, Jones [6] and Cornelissen et al. [4] discovered that a proper visualization of source code coverage information, i.e., the amount of lines of production code exercised by a test, helps developers with understanding production code and finding critical faults. More recently, researchers highlighted the relevance of test code quality, i.e., test code metrics [9] and smells [11], on the ability of tests to properly discover faults in production code, suggesting that, when dealing with faults, developers should not only rely on basic information on test coverage, but also on indicators able to characterize the quality of a test suite.

In this paper, we aim at putting previous knowledge into action by proposing a tool coined VITRuM (**VI**zualization of **T**est-**R**elated **M**etrics), an *IntelliJ* plug-in able to provide developers with an advanced visual interface of a number of test code-related factors, ranging from statically computable indicators (like quality metrics and smells [9, 11]) to dynamic measures such as code coverage indicators [5, 13]. We designed VITRuM to support developers with both the immediate analysis of source code as well as the evolutionary investigation of the capabilities of software tests, which can be useful to delineate whether and which tests would deserve maintenance operations [2].

2 METRICS EXTRACTED BY VITRUM

VITRuM is available in the *JetBrains Plug-in Repository* [1] and can be run on any Java project. It allows the analysis of three families of metrics. It is worth remarking that all the metrics can be included/excluded in the analysis by VITRuM users through a configuration panel.

Structural Metrics. Visualizing and monitoring structural aspects of test cases can help developers to assess the overall quality of test suites and the extent to which they are in line with the good practice of the object-oriented paradigm.

VITRuM includes the calculation of structural metrics related to several aspects, such as size, cohesion, coupling, and complexity.

In addition to CK Metrics, the plug-in also computes the Assertion Density, defined as the percentage of assert statements with respect to the total number of statements in a test class.

Test Smells. Defined as bad design or implementation choices applied by developers that could have a negative impact on understandability, maintainability, and effectiveness.

VITRuM includes the automatic identification of seven types of test smells, based on the detection mechanism defined by Palomba, et al. [10]. Details about the test smells are described in Table 1.

Dynamic Metrics. These metrics provide information about the effectiveness of tests. The current version of the plug-in allows the

[1] https://plugins.jetbrains.com/plugin/14160-vitrum

AVI '20, September 28-October 2, 2020, Salerno, Italy

ACM ISBN 978-1-4503-7535-1/20/09.
https://doi.org/10.1145/3399715.3399954

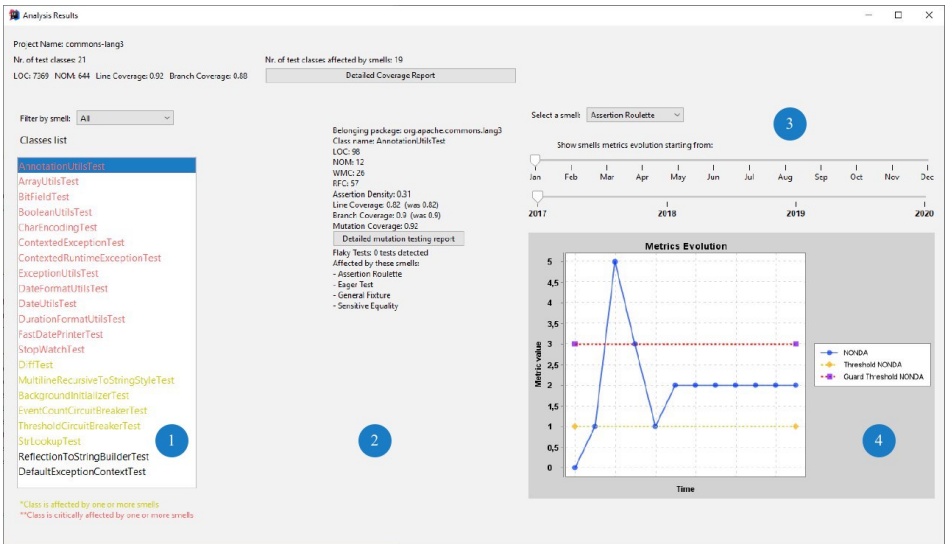

Figure 1: VITRuM main panel

Test smell	Description
Assertion Roulette	A test method having multiple non-documented assertions
Eager Test	A test method excercising more than one method of the production class
General Fixture	A test class having a too general setup (different tests only access part of it)
Mystery Guest	A testing accessing external resources like files or a database records
Resource Optimism	A test that makes optimistic assumptions about the existence of external resources
Sensitive Equality	A test using the *toString* method in assert statements
Indirect Testing	A test exercising different classes with respect to the production class corresponding to the test class

Table 1: List of test smells considered by VITRuM.

calculation of line and branch coverage, calculated as the percentage of lines/branches exercised by the test with respect to the total number of lines/branches in the production class. Moreover, it also allows the calculation of mutation coverage, defined as the percentage of mutated statements in the production class that is covered by the test [1]. Note that we used JaCoCo [2] to calculate line and branch coverage, and pitest [3] for the mutation coverage.

Finally, VITRuM also computes the analysis of flaky tests. Flaky tests are tests exhibiting a non-deterministic behavior (i.e., they can pass or fail with the same input). In order to detect the presence of flaky tests, VITRuM executes each test 10 times: if the output is different in at least one case than the test is flaky. Note that the number of independent executions can be customized in the configuration panel.

[2]https://github.com/jacoco/jacoco
[3]http://pitest.org

3 VITRUM VISUALIZATION

Figure 1 depicts the report shown after the computation of the metrics on the example project Apache commons-lang [4]. The window is composed of panels highlighted in the figure.

Panel ① lists all the test classes of the project under analysis. The classes are ordered by the criticality of the test smells affecting them. The list shows in black the classes that are not affected by any test smell, in yellow the classes affected by at least one test smell, and in red the classes critically affected by test smells, i.e., whose intensity is above a given critical threshold. On the top of the list, the plug-in presents a filter to narrow the search based on the selected test smell. Clicking on the name of one of the test classes in the list, all the other panels are shown. Panel ② reports the value for all the metrics calculated for the selected test class. Panel ④ contains a plot that allows the analysis of the metrics over time starting from the first execution of the plug-in on the subject project. The plot also contains two dotted lines that represent the thresholds for test smells analysis. In detail, the yellow line represents the threshold to determine whether the class is affected by the test smell, while the red line represents the criticality threshold. The slider on the top of the plot (panel ③) allows users to dynamically change the starting date to consider for the analysis

4 CONCLUSION

This paper presents VITRuM, a plug-in for the IntelliJ IDE that automatically compute a number of test-related factors and displays the results through a easy-to-use visual interface. Future work includes the integration of other metrics related to test code quality and the addition of features to export data and plots.

ACKNOWLEDGMENTS

Palomba gratefully acknowledge the support of the Swiss National Science Foundation through the SNF Project No. PP00P2_170529.

[4]https://commons.apache.org/proper/commons-lang/

REFERENCES

[1] James H Andrews, Lionel C Briand, Yvan Labiche, and Akbar Siami Namin. 2006. Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE Transactions on Software Engineering* 32, 8 (2006), 608–624.

[2] Paolo Benedusi, A Cmitile, and Ugo De Carlini. 1988. Post-maintenance testing based on path change analysis. In *Proceedings. Conference on Software Maintenance, 1988.* IEEE, 352–361.

[3] M-H Chen, Michael R Lyu, and W Eric Wong. 2001. Effect of code coverage on software reliability measurement. *IEEE Transactions on reliability* 50, 2 (2001), 165–170.

[4] Bas Cornelissen, Andy Zaidman, Arie Van Deursen, and Bart Van Rompaey. 2009. Trace visualization for program comprehension: A controlled experiment. In *2009 IEEE 17th International Conference on Program Comprehension.* IEEE, 100–109.

[5] Giovanni Grano, Fabio Palomba, and Harald C Gall. 2019. Lightweight assessment of test-case effectiveness using source-code-quality indicators. *IEEE Transactions on Software Engineering* (2019).

[6] James A Jones. 2004. Fault localization using visualization of test information. In *Proceedings. 26th International Conference on Software Engineering.* IEEE, 54–56.

[7] James A Jones, Mary Jean Harrold, and John Stasko. 2002. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002.* IEEE, 467–477.

[8] Pavneet Singh Kochhar, David Lo, Julia Lawall, and Nachiappan Nagappan. 2017. Code coverage and postrelease defects: A large-scale study on open source projects. *IEEE Transactions on Reliability* 66, 4 (2017), 1213–1228.

[9] Fabio Palomba, Annibale Panichella, Andy Zaidman, Rocco Oliveto, and Andrea De Lucia. 2016. Automatic test case generation: What if test code quality matters?. In *Proceedings of the 25th International Symposium on Software Testing and Analysis.* 130–141.

[10] Fabio Palomba, Andy Zaidman, and Andrea De Lucia. 2018. Automatic test smell detection using information retrieval techniques. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 311–322.

[11] Davide Spadini, Fabio Palomba, Andy Zaidman, Magiel Bruntink, and Alberto Bacchelli. 2018. On the relation of test smells to software code quality. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 1–12.

[12] James A Whittaker. 2000. What is software testing? And why is it so hard? *IEEE software* 17, 1 (2000), 70–79.

[13] W Eric Wong, Yu Qi, Lei Zhao, and Kai-Yuan Cai. 2007. Effective fault localization using code coverage. In *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, Vol. 1. IEEE, 449–456.

[14] Shigeru Yamada, Hiroshi Ohtera, and Hiroyuki Narihisa. 1986. Software reliability growth models with testing-effort. *IEEE Transactions on Reliability* 35, 1 (1986), 19–23.