Visualization of test-to-code relations to detect problems of unit tests

Nadera Aljawabrah, Tamás Gergely

Abstract: Visualization of test information can be a means of providing an understanding of test-code relations which, in turn, is essential for other activities in SDLC such as: maintenance, software evolution, refactoring, etc. To our best knowledge, less research have specifically addressed the visualization of test-code relation and its importance in many tasks during software development. This paper is a modest contribution to draw attention to visualization of test-code relations and in particular visualizing of test-to-code traceability links. From the outcome of our investigation it has been demonstrated that visualizing of test-related metrics has been received rather interest of researchers; while in turn, little attention has been paid to visualizing of test to code traceability links. However, several practical questions emerged which can be investigated as the next step of our research , thus further research of visualization of test-to-code traceability links area is highly needed to address these questions.

Keywords: Software testing; traceability recovery; test-to-code traceability; visualization

Introduction

Testing is considered an important phase in software development life cycle (SDLC) in which tests play a significant role in software evolution and maintenance, and helps building a better quality product. To achieve this, it is important to understand the tests and how they are related to the production code.This knowledge is essential for several activities such as: change impact analysis, refactoring, re-engineering, etc.

Visualization in general is widely and effectively used in software development for different purposes as it helps understanding. It can also be a very effective method to support testers to quickly understand the goal and properties of each test, to overview the structure of the test suite and to recognize the relation between test and code items. Many publications have been investigated in the domain of software visualization, source-code related visualization and test related visualization. This work considers visualization of relations between test and code from two aspects: visualization of test information and visualization of test-to-code traceability links.

Visualization of test information can be a means of providing a valuable information about: the adequacy of code testing [11], visualizing software faults [4], and code coverage of test suites to evaluate their quality [1]. The area of test information visualization has been widely targeted. For example, Cornelissen et al. [3] presented a visualization approach based on UML sequence diagram to visualize test information. While Jones et al. [6] proposed a technique that visualizes all statements that are executed by test suites and facilitates the location of faults in these statements. Quite recently, there has been a growing interest in visualizing test-related metrics which could provide information about the quality and process of the tests and the testing effort. However, to the author's best knowledge, very few publications are available in the kinds of literature that address the issue of visualizing the relationship between tests and code under test in term of test-to-code traceability links.

Our research takes into consideration two test-code relation areas: Test-related metrics and test-to-code traceability links. In this work, we define open questions to be answered in future research. The questions address the visualization of test-to-code traceability links from different aspects, for instance, why to visualize test-to-code links, how we can visualize links, what else could be visualized, and so on. By using visualization we could check, for example, the degree of consistency among traceability links, thus it becomes easier to understand how the test and code are really related and to determine if something is going wrong in this relation.

The remainder of this paper is organized in different sections as follow. Test related visualizations are detailed in Section , followed by open research area in Section , we conclude in Section .

Test-code relations visualization

Test-related metrics has been gaining importance in visualization's realm in recent years. Code coverage metrics is one of the most popular test metrics and is used to measure the percentage of code that

is executed by the test cases. There are several tools have been developed to visualize code coverage. Balogh et al. extended CodeMetropolis to include visualization of test related metrics in the Minecraft world to support developers to better understand the test suites quality and its relation to the production code [2]. Different types of test metrics are determined to show the behavior of test on the code of different units; e.g. code coverage metrics, partition metrics and other specific metrics. In the visualization space, code units are represented as buildings protected by outposts which, in turn, represent test suites. The attributes of the outposts reflect the quality attributes of the tests.

Visualization of test suites is useful to give any reader an obvious view of the testing results as well as it can have a significant effect on the quality of the software by reducing the cost and effort required to locate faults in the source code. On the other hand, establishing links between units under test and its related test suites helps in reducing regression tests and in turn saving much time during software development and maintenance [10]. Traceability links could be identified using: manual, automatic, semi-automatic or explicit methods [7]. Various approaches have been proposed to retrieve the traceability links between the tested units and units under test. Rompaey and Demeyer [12] compared six traceability strategies in terms of the applicability and the accuracy of each approach. The comparison covered only those approaches relating to requirement traceability and test-to-code traceability: Naming convention (NC), Fixture Element Types (FET), Last call before assert (LCBA), Lexical Analysis (LA), Static Call Graph (SCG) and Co-evolution (Co-Ev). According to these approaches, units under test are identified by matching test cases and production code's names and vocabulary (e.g. identifiers, comments), examining method invocations in test cases, looking at the last calls right before assert statement, and capturing changes on test cases and production code in the version control change log [12]. These approaches have some limitations particularly in industrial projects with practical usability or support. For instance, there are no industrial tools available to support automatic test-to-code traceability. In addition, visualization test-code links are not supported by any of above approaches [8] despite of the importance of visualization in the areas of software maintenance, evolution and refactoring.

Most of test-to-code traceability recovery approaches retrieve high-level links, i.e. links between unit tests and classes under test [9]. An automated test-to-code traceability approach has been proposed to recover links between source code and test cases on the method level by identifying Focal method under test [5].

Most of the visualization techniques provide visualization of relationships/links between requirements and other software artifacts (source code, design, test cases), while none of these techniques aim visualization of test-to-code traceability links in specific.

Open Research area

Depending on our investigations it is clear that further research in the area of visualization of test-to-code traceability links is necessary. We provide some open questions that can help to reveal specific topics in this area for further research and try to give some example answers. For ourselves, we feel these questions and directions the most interesting ones.

- *First, what the purpose of visualization can be ?* Visualization must have a purpose. Defining our goal can help in finding proper visualization techniques to be used and appropriate elements to be presented in it. Purpose can be: understand relations, impact analysis, find problems ("bad smells").

- *What is the suitable visualization technique that can be used to display test-to-code traceability relations and their attributes?* There are several possible ways to visualize relations including graphs, matrices, hyperlinks, lists, tree maps, 3D space. Based on the existing visualization techniques addressing traceability links between various software artifacts, graph-based visualization and traceability matrices seem to be the most suitable methods to represent links between code and tests. However, this may depend on the purpose. For example, when one tries to check the relations of an item for impact analysis, graph representation,as well as hyperlinks, seem to be appropriate. On the other hand, if someone needs a broader view to check inconsistencies among the relations, graph representation showing the traceability links inferred using different link-detection techniques in different colors might be a better choice. But a 3D visualization also seems to be appropriate to show attributes of different items and relations.

- *What test and code items, and properties should be objects and attributes in the visualization?* Relations can be visualized directly as objects in the visualization space (e.g. as lines between objects), or we can only map their properties to the attributes of the connected objects.
- *What the criteria should be taken into account to choose visualization technique?* For instance, the size of the program can be taken into account as several visualization methods can often become large and thus hard to read and understand for big projects.
- *What is the best recovery approach usable to retrieve the links between test and code?* There are several techniques can be used to infer these traceability relations, and each technique retrieves a slightly different set. Depending on the purpose of the visualization and the technique we use, either all can be visualized or we should choose and build on one of them. But which one? This is another open question that can be investigated.
- *At what detail level information should be visualized?* In a real system, there are thousands of tests and code items exist. Although it is not impossible to visualize all these at once, it is probably not the best way (although not necessarily impossible). Instead, a selective or hierarchical visualization seems to be a better choice. For example, instead of method level, one can show (test and production) classes, or group items based on their relations or some other purposes and visualize the groups only.

Conclusions

From the research that has been carried out, it is possible to conclude that more research in the visualization of test-to-code traceability is quite necessary. As there are many sources from where the traceability relations can be inferred, one of the most important questions is to decide which source or combination of sources are the best to determine the test-to-code links. It is obvious, that if these sources disagree, this will make it harder to understand what is going on, what was the goal of the developer, how the components are really related, change impact analysis can yield in false results, etc. Fortunately, visualization can aid this task. The goal of using visualization in this case is to "see" the disagreement between traceability links inferred from different sources. This might point out places where something is wrong with the tests and/or the code (at least their relationship) in a specific system. Further research then can aim the statistical comparison of these inference methods. Working on addressing these questions is continuing and will be presented in future papers.

References

[1] Vanessa Peña Araya. Test blueprint: an effective visual support for test coverage. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 1140–1142. ACM, 2011.

[2] Gergo Balogh, Tamás Gergely, Arpád Beszédes, and Tibor Gyimóthy. Using the city metaphor for visualizing test-related metrics. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, volume 2, pages 17–20. IEEE, 2016.

[3] Bas Cornelissen, Arie Van Deursen, Leon Moonen, and Andy Zaidman. Visualizing testsuites to aid in software understanding. In *Software Maintenance and Reengineering, 2007. CSMR'07. 11th European Conference on*, pages 213–222. IEEE, 2007.

[4] Marco D'Ambros, Michele Lanza, and Martin Pinzger. "a bug's life" visualizing a bug database. In *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*, pages 113–120. IEEE, 2007.

[5] Mohammad Ghafari, Carlo Ghezzi, and Konstantin Rubinov. Automatically identifying focal methods under test in unit test cases. In *Source Code Analysis and Manipulation (SCAM), 2015 IEEE 15th International Working Conference on*, pages 61–70. IEEE, 2015.

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

[7] Andrian Marcus, Xinrong Xie, and Denys Poshyvanyk. When and how to visualize traceability links? In *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, pages 56–61. ACM, 2005.

[8] Reza Meimandi Parizi, Asem Kasem, and Azween Abdullah. Towards gamification in software traceability: Between test and code artifacts. In *Software Technologies (ICSOFT), 2015 10th International Joint Conference on*, volume 1, pages 1–8. IEEE, 2015.

[9] Reza Meimandi Parizi, Sai Peck Lee, and Mohammad Dabbagh. Achievements and challenges in state-of-the-art software traceability between test and code artifacts. *IEEE Transactions on Reliability*, 63(4):913–926, 2014.

[10] Abdallah Qusef. Test-to-code traceability: Why and how? In *Applied Electrical Engineering and Computing Technologies (AEECT), 2013 IEEE Jordan Conference on*, pages 1–8. IEEE, 2013.

[11] Thomas Tamisier, Peter Karski, and Fernand Feltz. Visualization of unit and selective regression software tests. In *International Conference on Cooperative Design, Visualization and Engineering*, pages 227–230. Springer, 2013.

[12] Bart Van Rompaey and Serge Demeyer. Establishing traceability links between unit test cases and units under test. In *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on*, pages 209–218. IEEE, 2009.