Essay title: **Test to Code Traceability - Preventing Software Risks Through Advanced Code Monitoring and Risk Evaluation**

your name: **Apurva Ravikiran Shirbhate**

student ID number: **23266290**

email address: **apurva.shirbhate2@mail.dcu.ie**

program of study: **MSc in Computing (Data Analytics)**

module code: **CA640**

date of submission: **20 Oct. 23**

word count. **1803**

I understand that the University regards breaches of academic integrity and plagiarism as grave and serious. I have read and understood the DCU Academic Integrity and Plagiarism Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach this policy. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the sources cited are identified in the assignment references. Any use of generative AI or search will be described in a one-page appendix including prompt queries.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work. By signing this form or by submitting this material online I confirm that this assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. By signing this form or by submitting material for assessment online I confirm that I have read and understood DCU Academic Integrity and Plagiarism Policy.

Signed Name: **Apurva Ravikiran Shirbhate**

Date: **20 Oct. 23**

Test to Code Traceability - Preventing Software Risks Through Advanced Code Monitoring and Risk Evaluation

Apurva Ravikiran Shirbhate
MSc in Computing - Data Analytics
Dublin City University
Dublin, Ireland
apurva.shirbhate2@mail.dcu.ie

October 20, 2023

1 Introduction

In software development, software quality and reliability are Pivotal. The effectiveness of testing is a crucial aspect of software quality, mainly unit testing, which provides quality and reduces maintenance. In addition to identifying and fixing bugs, software testing improves software performance and ensures that the software requirements of stakeholders are met Kicsi et al. (2021). In test suites, unit tests can be used to check the code coverage quality, visualize software errors, and ensure the code testing quality Aljawabrah et al. (2019).

Once mapping between test-to-code is established, it is easier for the developer to analyze the impact of changes to the system. Effective unit testing needs appropriate traceability between tests and code. Without a Traceability link, it is challenging to retain synchronization between test code and code that needs to be tested. Therefore, it is more likely that new bugs may be introduced; impact analyses can be challenging and make it hard to understand the program White et al. (2020).

For mitigating software risks, the establishment and maintenance of traceability links is crucial. This will assure the development of robust software White et al. (2020). Chances of test failure and unnoticeable issues will be minimised if tests and code are kept in sync. Manual linking requires effort and is prone to mistakes, but once you establish traceability, it has several advantages. Reuse, synchronization, code modifications and maintenance, improved program understanding, fewer defects, and lower software risks are some of the benefits White et al. (2020). Traceability links are also useful for identifying and performing tests affected by code changes, improving regression test suites in continuous integration and reducing the cost of unnecessary testing.

In order to reduce the risk associated with software and maximize the effectiveness and execution of software development, I have examined various techniques for test-to-code traceability published in multiple journal articles and research papers. Over a year, a variety of methods have been proposed for implementing traceability links

between tests and code, both at method and class level, which will be examined in the literature review.

2 Literature Review

This subsection is divided into multiple subsections. These subsections demonstrate reviews of various articles and research papers about Visualization, test-to-code traceability links and techniques to achieve them using different approaches. The methods used in each piece of article and paper are discussed below in detail.

2.1 Understanding Test-to-Code Traceability Links: The Need for a Better Visualizing Model

In this paper Aljawabrah et al. (2019), authors conclude that the authors have acknowledged that In software development, the visual representation of the connection between test cases and code is crucial. The relationship between these links can be better interpreted and regulated by visualizing them, which improves software quality, makes it simpler to find faults and increases the efficiency of maintenance.

This paper attempts to showcase the advantages of visualizing test-to-code traceability links and suggests a multiple number of research questions and questions for additional research in this field[0].In addition, Visualization is necessary to evaluate code coverage and to provide information to the tester regarding quality, performance, the test suite's location and its relationship with production code and which parts of the code the test cases have covered.

Numerous Visualization technologies have been developed, which include graphs, UML diagrams, metaphors, matrices, hyperlinks, lists, tree maps and 3D space. In terms of determining the traceability relationship between code and tests, "graph-based Visualization" and "traceability links" are considered to be the most suitable methods among all available methods.

2.2 Automated Recovery and Visualization of Test-to-Code Traceability (TCT) Links:An Evaluation

In this paper Aljawabrah et al. (2021), the authors emphasise the significance of visualizing traceability links between test cases and code classes, particularly in agile development and describe the challenges in developing and visualizing links between code classes and unit tests. It underlines the necessity for the dynamic Visualization tool and draws attention to the drawbacks of the current automated traceability recovery proposed methods.

In this paper, we can observe that the authors have presented TCTrasVis, a Visualization tool that supports adequate Visualization and optimizes the creation of associations between test cases and code classes/methods. The tool aggregates multiple TCT recovery approaches and shows the general structure of the traces, which helps developers and software testers recognize between true links and false positives. These linkages are spontaneously recovered by TCTracVis using a variety of techniques, which include static calls graphs and naming conventions.

The tools' Visualization techniques adopt a hierarchical tree network to demonstrate these interactions, which makes it easier for the developers to understand the relation between test cases, test classes and production code. It endorses both 1:N and N:1 relationships. It includes two different types of graphs, where one indicates dependencies from code to test cases, and the other presents the test-to-code relationship. This tool is a pivotal asset in the software development process because it makes it easy for the developer to recover, navigate and comprehend traceability relationships.

2.3 Establishing Multilevel Test-to-Code Traceability Links

In this research paper White et al. (2020), authors have focused on both class-level and method-level simultaneously, offering a more precise and fine-grained illustration of these relationships. This approach differs from the static method as here, authors have used dynamic information and ranking potential links. TCtracer, the proposed approach integrates various methods by using dynamic cell traces and novel call-based statistical algorithms. It generates a single score that performs better than individual techniques and works at both class and method levels.

The method is tested using the manually chosen dataset from four subject projects, providing better efficacy at both the class and method levels. The key contributions of this work are the combination of various techniques, the progression of these techniques, assessing the advantages of multilevel information and the supply of a ground truth dataset from test-to-code function and test-to-class connections. This paper has discussed multiple techniques used in TCtracer for the development of traceability links in order to overcome the weakness of existing techniques.

The techniques which are used in TCtracer are Naming Conventions(NC), Naming Conventions-Contains(NCC), Term Frequently-Inverse Document Frequency(TFIDF), Last Call Before Assert(LCBA), Name Similarity and Tarantula. The common problems bound with static methods are eliminated as these methods use dynamic trace information.

2.4 Large Scale Evaluation of Natural Language Processing Based Test-to-Code Traceability Approaches

The research presented in this paper Kicsi et al. (2021) aims to enhance traceability in software development by utilizing ML models and exploring the advantages of combining multiple techniques for traceability. The authors have carried out a study on eight different systems, and the outcome of the study indicates that an enhanced version of lexical analysis increases traceability accuracy. The Paper provides meaningful insight into the difficulties encountered during traceability in software development.

A combination of naming conventions, IR methods, and ML methods is suggested as a potential solution. The primary goal of the paper is to examine textual techniques for the enhancement of code-to-test traceability links. In this paper, the author has transformed the Java source file into a m machine-learning-friendly one and applies three text-based techniques to gain insight into how comparable test and code classes are. These techniques include Latent Semantic Indexing (LSI), Term Frequency-Inverse Document Frequency (TF-IDF), and Document vectors (Doc2Vec).

The results are refined to filter the lists produced by the text-based technique by using class information from source files, such as the lists of imported packages and specified methods. By comparing the results from various techniques, additional refinement is carried out to retain only the standard code classes. The study introduces five code representations for input and discusses their applicability to the input of the machine learning algorithm. SRC (Structured text), TYPE (AST node types), IDENT(identifiers), LEAF(AST node types and values) and SIMPLE (just SimpleName AST node values) are some of the representations.

3 Conclusion

From this survey, it can be concluded that traceability between tests and code is essential to enabling numerous development risks associated with agile development. It has been examined that Visualization is a powerful tool for software development jobs and an efficient way to demonstrate traceability relationships. This paper Aljawabrah et al. (2019) concludes that there is a necessity to choose the optimal sources for test-to-code traceability and how Visualization can be helpful in this regard. Future research should concentrate on statistical comparisons of inference techniques and it should work to implement a tool for determining and retaining traceability linkage, making it easier to identify the differences between various sources.

The paper Aljawabrah et al. (2021) offers TCTracVis, a traceability Visualization tool that quickly navigates and illustrates traceability links between test cases and code components. The study has concluded that the traceability Visualization tool has undergone a usability assessment to assess its usability and utility. In the future, it might extend to support various programming languages, more recovery techniques, and broaden project-level Visualization. This paper Kicsi et al. (2021) concludes the investigation to maximize the traceability links between code-to-test by using naming conventions and information retrieval techniques. The identifier-centric representation with abstract syntax trees was found to be most successful for naming standards over other traceable link extraction producers and code representation. Results were considerably improved by utilizing regular expressions to obtain call information, particularly when using Doc2Vec. After combining Doc2Vec with call information, it generated the best possible outcome.

In this paper White et al. (2020), TCtracer, a strategy and development for creating test-to-code traceability linkage at both method level and class level, is examined. In order to enable cross-level information flow, TCtracer integrates multiple traceability link techniques and applies them across these two levels. According to an empirical analysis involving four actual open-source projects, TCtracer is the most successful method for the achievement of test-to-code traceability links. On average, TCtarcer outperforms individual solutions. Automated tests can be linked to changing applications through code monitoring. By establishing this connection, testing flaws can be identified and the threat of untested code alteration can be analyzed. In addition, where the code has been frequently changed, an analytical engine can examine these changes and make suggestions for running specific tests to identify the gap.

References

Aljawabrah, N., Gergely, T. & Kharabsheh, M. (2019), *Understanding Test-to-Code Traceability Links: The Need for a Better Visualizing Model*, pp. 428–441.

Aljawabrah, N., Gergely, T., Misra, S. & Fernandez-Sanz, L. (2021), 'Automated recovery and visualization of test-to-code traceability (tct) links: An evaluation', *IEEE Access* **9**, 40111–40123.

Kicsi, A., Csuvik, V. & Vidács, L. (2021), 'Large scale evaluation of natural language processing based test-to-code traceability approaches', *IEEE Access* **9**, 79089–79104.

White, R., Krinke, J. & Tan, R. (2020), Establishing multilevel test-to-code traceability links, *in* '2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)', pp. 861–872.