AUTOMATING METAMORPHIC TESTING BY PREDICTING METAMORPHIC RELATIONS FOR COMPLEX AND VULNERABILITY-PRONE APPLICATIONS

by

Karishma Rahman

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 2025

©COPYRIGHT

by

Karishma Rahman

2025

All Rights Reserved

DEDICATION

I dedicate my thesis to my family.

ACKNOWLEDGEMENTS

I want to thank my advisor, Dr. Clemente Izurieta, for his unwavering support, motivation, and insightful guidance in helping me reach this significant milestone. I also appreciate his efforts in providing invaluable feedback for this dissertation report. I also thank Dr. John Paxton, Dr. Mary Ann Cummings, and Dr. Binhai Zhu for being my esteemed committee members and their ongoing support.

Among my lab fellows and friends, I feel fortunate to have Musaddeque A. Syed, Prashanta Shaha, and Faqeer ur Rehman in my circle. They are always available to guide and help me stay focused, determined, and motivated. I am grateful to Dr. John Paxton for providing me with the teaching assistantship opportunity throughout the program. I wish to extend my heartfelt thanks to the graduate school for granting me the Ph.D. dissertation completion award and to Donna Negaard for her outstanding support and guidance. Additionally, I am grateful to MSU for providing me with an exceptional educational opportunity that enabled me to fulfill my Ph.D. aspirations at one of the leading institutions in the nation.

I want to thank my parents, AKM Shafiqur Rahman and Nishat Hasan, for encouraging me to succeed. I also want to thank my sister, Kashfia Rahman, for being my strength. Also, I'd like to thank my partner, Md Redwan Ahmad Khan, for inspiring me to complete my journey. Finally, I want to thank my friends in Bozeman for their support, especially Kazi Faria Islam, Farzana Sarker Mahin, Sajia Afrin Glory, and Emtiaz Ahmed.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

TABLE OF CONTENTS – CONTINUED

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

Table Page

LIST OF FIGURES

LIST OF FIGURES – CONTINUED

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

ABSTRACT

Software plays an important role in our daily operations and is a critical component of our infrastructure. Testing is necessary to evaluate the effectiveness of any software. To accomplish this, there is a fundamental need for an oracle, which determines whether the observed behavior of the software accurately reflects its intended functionality. However, software testing can pose significant challenges due to its complexity and the necessity of having a reliable oracle, often referred to as the oracle problem in software testing. Metamorphic Testing (MT) can alleviate the oracle problem because it focuses on evaluating software based on its inherent characteristics or properties. MT is a sophisticated technique that involves generating a variety of inputs for a program, subjecting them to predefined transformations, and subsequently comparing the resulting outputs with the original ones to verify the correctness of the program's behavior. Metamorphic Relations (MRs) are central to MT because they establish the relationships between the inputs and outputs of the system being tested and specify how they should change when the inputs are altered. Typically, identifying MRs is a manual process that often necessitates collaboration with domain experts, especially when testing complicated programs. Consequently, this task can be labor-intensive and prone to errors. Therefore, the development of automated methods for identifying MRs holds the potential to enhance the efficiency and effectiveness of MT, making it a more practical and reliable approach for ensuring the reliability of complex software systems. Hence, I employ MT techniques to analyze the software's behavior and anticipate and define MRs to achieve this goal. By predicting MRs, I streamline the testing process significantly. This entails automating the assessment of software behavior, reducing the reliance on manual testing procedures.

In this thesis, I use machine learning classification models to predict MRs using data from diverse fields to identify faults. This approach predicts MRs for more complicated programs, such as Matrix Calculation Programs. Next, I examine the feasibility of MRpredT, a Text Classification-Based Machine Learning approach to predict MRs using only their program documentation as input. Then, I study the scope of testing applications with security flaws using MT. A systematic mapping study that documents the latest empirical research in web application security vulnerability detection indicates that vulnerability testing also encounters the oracle problem due to the vast range of inputs. Afterward, I introduce new MRs through a case study to test banking functions and demonstrate an MT framework. Finally, I detected vulnerabilities using MT, which led me to build an automated approach for vulnerable programs in online banking applications. It offers a catalog of 8 system-agnostic MRs to automate security testing for detecting these vulnerabilities among the OWASP Top 10. All the study results demonstrate that these approaches are theoretical and practical. It scales effectively, allowing for overnight automated software testing, and positions MT as a valuable and powerful tool for enhancing the correctness of any software system application.

1

INTRODUCTION

Advancements in science and technology have significantly evolved the software industry in recent years [1]. With the growing demand for scientific and E-type software in modern society, most applications involve complex scientific calculations and data processing, requiring software engineers to ensure software meets all reliability [2] criteria. E-type systems refer to real-world systems, and one of Lehman's Laws states that these systems constantly evolve, and their complexity continues to increase unless measures are taken to minimize it [3]. Every software system is a highly reusable E-type system whose functionality directly impacts economic growth [3]. Thus, software development and testing are vital steps in the software life cycle. Moreover, efficient validation and verification methods are crucial for guaranteeing the reusability of these applications [2]. As consumer needs for software have increased, projects have grown more complex, prompting further investigation into different facets of software [4]. Testing software demands high complexity, security, and accuracy, with users expecting them to be secure and accurate [1]. Software includes intricate designs and multi-layered workflows, offering various features and functions while handling sensitive data like personal information [2]. Thus, software testing must be thorough since any gaps in test coverage can lead to data breaches, vulnerabilities, fraud, and other criminal activities [5]. Additionally, automating testing techniques can significantly enhance the software development process. Testing is a mandatory and significant part of the software development life-cycle despite the progress in software implementation technologies and programming languages [2]. Testing bears more than 50% of the total software development costs, as it is an expensive, time-consuming, and complex activity [5]. Creating dependable software systems remains an ongoing challenge, and researchers and

practitioners continuously explore more efficient methods to test software [6].

The process of testing is often prone to human error. During testing, test cases are performed on the system under test. A test oracle, either automated or manual, is then used to determine whether it acted as anticipated [6]. In either case, the actual output is compared with the expected outcome. The challenge of using a test oracle arises when testing complex software [7]. It occurs when it is too difficult to determine whether the program outputs on test cases are correct or not [7]. This is called the oracle problem [7]. For example, banking software often includes functionality that can encounter this problem [7]. For instance, with electronic payment software, there may be situations where the consumer needs clarification on how much is being charged for a given input. The challenge becomes even more pronounced when the payment concerns transfer charges among different bank accounts or currency exchange applications.

Metamorphic Testing (MT) is a technique that has proven helpful in certain circumstances to address the challenge of the oracle problem [8]. The principle behind MT is that it might be easier to analyze the relations between the results of multiple test executions, which are referred to as Metamorphic Relations (MRs), rather than specifying the input-output behavior of a system [8]. MT employs MRs to determine system properties, automatically altering the initial test input into follow-up test input [8]. If the system fails to meet the MRs when tested with the initial and follow-up input, it is inferred that it is defective [8]. A significant number of studies have focused on creating MT methods for specific areas such as computer graphics [9], web services [10], and embedded systems [11]. MRs are crucial for boosting the efficiency of fault detection in testing. Traditionally, pinpointing these relations involves a labor-intensive, error-prone manual process, often requiring input from domain experts, mainly when dealing with intricate programs. Hence, developing techniques to identify MRs is a promising avenue. Such automation has the potential to improve the efficiency and effectiveness of MT significantly, rendering it a more feasible and dependable

method for guaranteeing the reliability of complex software systems.

Contributions of the Dissertation

The overall contributions of this thesis are outlined in this section. They exhibit developing MRs that capture properties (i.e., characteristics that are compromised when the system is at risk) and automate testing using prediction models for scientific and E-type software. Each of these contributions is thoroughly discussed in the respective proceedings cited.

- An improved Graph Kernel-Based Machine Learning method for programs performing matrix calculations, which uses Control Flow Graphs (CFGs) to extract feature information from the programs. Random walk and graphlet kernels are used to extract the features to build the models [12].

- A text mining-based machine learning approach to systematically identify MRs for functions that perform matrix calculations. This method leverages Javadoc, a structured documentation framework, as the primary data source for training the model. By utilizing the Bag of Words (BoW) technique, I build a model to recognize patterns that enhance the accuracy and automation of MR identification, contributing to more efficient and reliable software testing [13].

- A systematic mapping study that develops a classification scheme for systematically categorizing research articles on vulnerability detection. Additionally, it presents a comprehensive systematic mapping study that analyzes 76 relevant studies published over the past two decades (2001–2021). This mapping study provides a structured overview of related research, highlighting trends, gaps, and advancements in the field [14].

- An approach to investigate and uncover the essential points in employing MT to test banking software, presented through a list of new MRs for banking functions. To demonstrate the applicability of the proposed MRs, I conduct a case study on banking software functionalities. The study indicates the significance of using MT for testing banking functions. Additionally, I utilize mutation analysis to evaluate the effectiveness of the MT approach [15].

- A catalog of 8 MRs targeting the well-known top 10 OWASP security vulnerabilities commonly found in online banking applications, along with a framework that automatically predicts MRs for unseen programs susceptible to vulnerabilities, using a Graph-based Convolutional Neural Network (GCNN) model.

Overview of the Dissertation

The dissertation is structured as follows. The Introduction Chapter provides the context for this thesis. Chapter 2 outlines the motivation for the thesis, goals, and problem decomposition. The research goals (RGs), the research questions (RQs), and the research metrics (RMs) are presented in this chapter, along with the proceedings contribution of the thesis. In Chapter 3, I introduce the key concepts relevant to this thesis, including software testing, test oracle, MT, MRs, mutation testing, and prediction models. It also summarizes the pertinent work in the field, positioning our research within the domain of MT. It particularly emphasizes the selection and classification of MRs, reviewing previous contributions and highlighting gaps. Chapter 4 discusses improving the graph kernel-based machine learning method for programs performing matrix calculations. Chapter 5 describes the methodology of the text mining-based machine learning approach to identify MRs for a function. Chapter 6 details a systematic mapping study of related research over the past 20 years (2001-2021) by analyzing 76 articles. Chapter 7 outlines a framework for MT in banking

software and discusses a case study that tests primary banking functions. Chapter 8 offers a catalog of 8 MRs targeting the widely recognized top 10 OWASP security vulnerabilities, along with a framework that automatically predicts MRs for previously unseen programs vulnerable to threats using a Graph-based Convolutional Neural Network (GCNN) model. In Chapter 9, I conclude by summarizing the contributions, threats to validity, and discussing avenues for future research.

6

RESEARCH OBJECTIVES

Motivation

In software engineering, there is often a significant focus on creating systems that exhibit a high level of accuracy, functionality, and innovation. However, ensuring the correctness of these systems is an equally critical aspect. Software testing has become a vital area of research aimed at improving its dependability. This thesis primarily focuses on automating software testing, particularly in adapting and enhancing traditional testing methods to enhance software accuracy, which guarantees its intended functions and operates without serious errors or vulnerabilities. Among the various software testing techniques, I have selected MT as an especially effective method for addressing the oracle problem, which involves the difficulty of verifying whether the actual behavior of software corresponds with its expected behavior. By systematically comparing transformed inputs and outputs, MT shows promise in tackling this challenge and automating this testing process can significantly improve its efficiency. My work is motivated by recognizing specific gaps in the existing research landscape, which present exciting opportunities for exploration and further study. By tackling these gaps, I aim to enhance software testing practices and ultimately improve the software testing process. Below are the motivations for employing MT in software testing and automating the process.

- Principle of Simplicity: The core idea and technical foundation of MT are straightforward and refined. Previous research shows that people with minimal testing experience can understand MT concepts in just a few hours and successfully apply them across various systems [16].

- Easy Implementation: According to the established concept, implementing MT is straightforward. It depends on MRs to create test cases and validate results. Previous

studies on MT suggest that identifying MRs, particularly when many have been discovered, tends to be relatively easy, even if some aspects may be automatable [16]. Additionally, it should be feasible for users to build MT tools customized to their particular domains.

- Automation Feasibility of MRs: In addition to the initial identification of MRs, automating the core phases of MT—like test case generation, execution, and verification—should be relatively simple. Creating specific test cases is a direct process since they can be produced using well-established testing techniques. Following this, new test cases can be developed by applying transformations based on MRs. Executing test cases is generally straightforward, making it one of the most easily automated components in various testing strategies. Additionally, verifying test results in MT can be automated through scripts that cross-reference test outputs with the relevant MRs. The only element in the MT workflow that might not be fully automated is the identification of MRs. Still, this area shows promise for enhancement due to recent studies investigating MR identification techniques [17]. While current tools cover the full MT process for certain application domains, further exploration is necessary to create a comprehensive framework that optimizes automation throughout all MT stages. This ongoing research strives to establish a flexible system that automates as many parts of the MT process as feasible [18].

- Affordable Approach: MT is more cost-effective than traditional testing methods. It focuses on identifying MRs, imposing only slight computational costs for generating and running subsequent test cases and verifying their outcomes. Although manual effort is required for MR identification, leading to some overhead, this is an expected and inherent aspect of the process. However, there is potential for automation through MR prediction [18]. As mentioned, creating follow-up test cases using MR-based

transformations is efficient and usually requires few resources [16]. Moreover, the overhead associated with the test result verification, which consists of comparing outputs with MRs, is considerably lower compared to costs in situations where the oracle problem exists [16].

Goal

The primary objective of my thesis is to automate the process of MT by successful automatic predictions of MRs. As it has already been established that MT is a very effective way of tackling the oracle problem in software testing, automation of MT by predicting MR discussed in this thesis improves testing efficiency, minimizes manual effort, and enhances the dependability of complex systems, especially in areas where conventional testing methods fall short.

Research Approach

This section presents the problem statement and research approach addressed in this thesis. The section also shows how each question is followed up by another. Figure 1 illustrates the research approach. I have followed the *GQM (Goal Question Metric)* method, which is the basis for my research plan and directs the entire thesis. To reach my primary objective, I craft research questions (RQs) that align with specific research goals (RGs) and later develop metrics (RMs) to evaluate these questions quantitatively. The RGs are labeled with numbers from one to six (e.g., RG1 , RG2, etc.). The RQs, and the RMs are numbered, respectively (e.g., RQ1.1, RQ1.2, etc., and, e.g., RM1, RM3, etc.).

MT relies on identifying MRs to validate software behavior, and manually identifying MRs is a time-consuming and labor-intensive process that limits scalability and efficiency. I selected two types of subject programs, numerical programs and vulnerable programs,

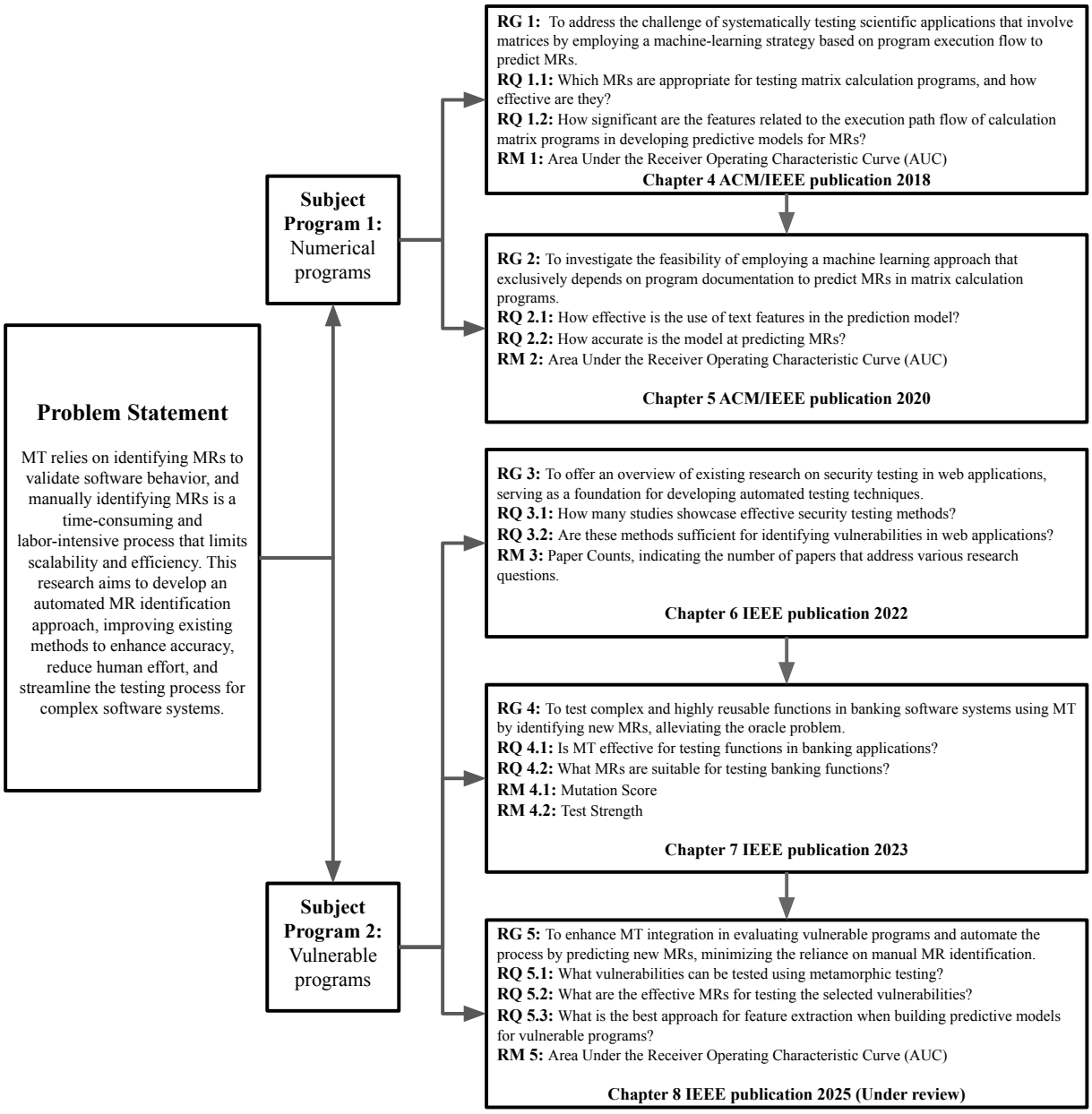

Figure 1: Research Approach Decomposition

to address the problem of identifying an MR. Software applications designed to perform numerical computations, often dealing with mathematical models, scientific simulations, and engineering calculations, are classified as numerical programs [19]. Applications that might have security vulnerabilities, flaws, or weaknesses that can be exploited to compromise the security of that application are classified as vulnerable programs [20].

Research Goal 1: To address the challenge of systematically testing scientific applications that involve matrices by employing a machine-learning strategy based on program execution flow to predict MRs.

Research Question 1.1: Which MRs are appropriate for testing matrix calculation programs, and how effective are they?

Research Question 1.2: How significant are the features related to the execution path flow of calculation matrix programs in developing predictive models for MRs?

Research Metric 1: *Area Under the Receiver Operating Characteristic Curve (AUC)* - AUC quantifies the probability that a randomly selected negative example will have a lower prediction score than a randomly chosen positive example.

Research Goal 2: To investigate the feasibility of employing a machine learning approach that exclusively depends on program documentation to predict MRs in matrix calculation programs.

Research Question 2.1: How effective is the use of text features in the prediction model?

Research Question 2.2: How accurate is the model at predicting MRs?

Research Metric 2: *Area Under the Receiver Operating Characteristic Curve (AUC).*

Research Goal 3: To offer an overview of existing research on security testing in web applications, serving as a foundation for developing automated testing techniques.

Research Question 3.1: How many studies showcase effective security testing methods?

Research Question 3.2: Are these methods sufficient for identifying vulnerabilities in web applications?

Research Metric 3: *Paper Counts*, indicating the number of papers that address various research questions.

Research Goal 4: To test complex and highly reusable functions in banking software systems using MT by identifying new MRs, alleviating the oracle problem.

Research Question 4.1: Is MT effective for testing functions in banking applications?

Research Question 4.2: What MRs are suitable for testing banking functions?

Research Metric 4.1: *Mutation Score*- The MS calculates the percentage of mutants killed out of the total number of mutants. (i.e., excluding the equivalent mutants) created without test coverage.

Research Metric 4.2: *Test Strength*-Test Strength measures the ratio of mutants killed out of all mutants with test coverage. The Test Strength metric does not include mutants that survive due to a lack of coverage.

Research Goal 5: To enhance MT integration in evaluating vulnerable programs and automate the process by predicting new MRs, minimizing the reliance on manual MR identification.

Research Question 5.1: What vulnerabilities can be tested using metamorphic testing?

Research Question 5.2: What are the effective MRs for testing the selected vulnerabilities?

Research Question 5.3: What is the best approach for feature extraction when building predictive models for vulnerable programs?

Research Metric 5: *Area Under the Receiver Operating Characteristic Curve (AUC).*

The following table provides a structured overview of how the thesis chapters and related academic proceedings contribute to addressing the core RQs. Each research question

is mapped to specific thesis chapters that provide theoretical foundations, methodological frameworks, experimental findings, and discussions. Additionally, relevant proceedings are included to highlight peer-reviewed contributions that reinforce and validate the thesis findings. In Table 1, I outline the thesis chapters and the respective proceedings that address each RQ raised.

Table 1: Mapping of Dissertation Chapters and Research Publications to Research Questions (RQs)

Research Question	Chapter No. & Title	Proceedings
RQ1.1	4: Predicting Metamorphic Relations for Matrix Calculation Programs	K. Rahman and U. Kanewala, "Predicting Metamorphic Relations for Matrix Calculation Programs,"2018 IEEE/ACM 3rd International Workshop on Metamorphic Testing (MET), Gothenburg, Sweden, 2018, pp. 10-13.
RQ1.2	4: Predicting Metamorphic Relations for Matrix Calculation Programs	K. Rahman and U. Kanewala, "Predicting Metamorphic Relations for Matrix Calculation Programs,"2018 IEEE/ACM 3rd International Workshop on Metamorphic Testing (MET), Gothenburg, Sweden, 2018, pp. 10-13.

Table 1 – continued from previous page

RQ2.1	5: MRpredT: Using Text Mining for Metamorphic Relation Prediction	K. Rahman, I. Kahanda, and U. Kanewala, "MRpredT: Using Text Mining for Metamorphic Relation Prediction," In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, 2020, pp. 420–424.
RQ2.2	5: MRpredT: Using Text Mining for Metamorphic Relation Prediction	K. Rahman, I. Kahanda, and U. Kanewala, "MRpredT: Using Text Mining for Metamorphic Relation Prediction," In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, 2020, pp. 420–424.
RQ3.1	6: A Mapping Study of Security Vulnerability Detection Approaches for Web Applications	K. Rahman and C. Izurieta, "A Mapping Study of Security Vulnerability Detection Approaches for Web Applications,"2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Gran Canaria, Spain, 2022, pp. 491-494, doi: 10.1109/SEAA56994.2022.00081.

Table 1 – continued from previous page

RQ3.2	6: A Mapping Study of Security Vulnerability Detection Approaches for Web Applications	K. Rahman and C. Izurieta, "A Mapping Study of Security Vulnerability Detection Approaches for Web Applications,"2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Gran Canaria, Spain, 2022, pp. 491-494, doi: 10.1109/SEAA56994.2022.00081.
RQ4.1	7: An Approach to Testing Banking Software Using Metamorphic Relations	K. Rahman and C. Izurieta, "An Approach to Testing Banking Software Using Metamorphic Relations," 2023 IEEE 24th International Conference on Information Reuse and Integration for Data Science (IRI), Bellevue, WA, USA, 2023, pp. 173-178, doi: 10.1109/IRI58017.2023.00036.
RQ4.2	7: An Approach to Testing Banking Software Using Metamorphic Relations	K. Rahman and C. Izurieta, "An Approach to Testing Banking Software Using Metamorphic Relations," 2023 IEEE 24th International Conference on Information Reuse and Integration for Data Science (IRI), Bellevue, WA, USA, 2023, pp. 173-178, doi: 10.1109/IRI58017.2023.00036.

Table 1 – continued from previous page

RQ5.1	8: Metamorphic Relation Prediction for Security Vulnerability Testing of Online Banking Applications	K. Rahman and C. Izurieta, "Metamorphic Relation Prediction for Security Vulnerability Testing of Online Banking Applications," submitted on 2025 IEEE International Conference on Cyber Security and Resilience, Greece, 2025.
RQ5.2	8: Metamorphic Relation Prediction for Security Vulnerability Testing of Online Banking Applications	K. Rahman and C. Izurieta, "Metamorphic Relation Prediction for Security Vulnerability Testing of Online Banking Applications," submitted on 2025 IEEE International Conference on Cyber Security and Resilience, Greece, 2025.
RQ5.3	8: Metamorphic Relation Prediction for Security Vulnerability Testing of Online Banking Applications	K. Rahman and C. Izurieta, "Metamorphic Relation Prediction for Security Vulnerability Testing of Online Banking Applications," submitted on 2025 IEEE International Conference on Cyber Security and Resilience, Greece, 2025.

16

BACKGROUND & RELATED WORK

This chapter offers a comprehensive overview of software testing, specifically for scientific and vulnerable applications, and addresses the related test oracle problem. It explains how MT operates by following the appropriate MRs for the system under test. It also covers mutation testing, which is commonly used to evaluate the effectiveness of MRs. The chapter also describes the machine learning approaches applied in this thesis by reviewing relevant literature on MT and various methods for detecting MRs across different software types. Lastly, it discusses the justification for using classification techniques to uncover MRs.

Software Testing

Software testing is a quality assurance activity that is crucial to software development. It requires conducting tests of the software system with a specified set of test inputs to identify possible failures in them [2]. When the actual output of the system varies from the expected output as outlined in its requirements, a failure is identified [2]. This results in abnormal behavior, system issues, or security vulnerabilities in the system. The primary objective of software testing is to ensure the software meets the requirements and delivers a high quality, reliable and secure product. [21]. A successful case of the testing process requires a clearly specified set of input values that can be systematically processed by the system under test, which covers a wide range of functional, non-functional, and edge-case scenarios [21]. Testers can determine if a test case has passed or failed by implementing a structured process that compares the actual results with the expected outcomes for specific inputs [21]. This systematic approach to software testing allows for early detection of defects during the development life-cycle, reducing the risk of costly failures after the deployment.

As noted by Ammann and Offutt [2], software testing can only uncover existing failures

within a system and does not ensure the total absence of defects. Even with thorough test planning and execution, testing has inherent limitations due to the complexity of contemporary software systems. Executing flawless system performance would require testing the software against every possible input and execution path, which is practically unachievable because of the enormous number of potential input combinations and system states [22]. Rather than exhaustive testing, software testers utilize various methods, such as black-box and white-box testing and automated testing, to improve defect detection effectiveness [2].

Testing Web Applications

Testing techniques in modern web development are often incomplete or flawed, introducing vulnerabilities in web applications. These vulnerabilities enable malicious users to inject harmful artifacts, such as script injections, data flow attacks, and input validation attacks, into web content [4]. Web applications operate with large volumes of user data and provide critical functionalities that make them secure and reliable. The complexity of web applications originates because of their dynamic nature, various user roles, and multiple input interfaces, such as web pages, forms, and cookies. [23]. Given these elements, thorough testing is essential to identify vulnerabilities and ensure that web applications operate securely and efficiently. Functional and security testing are critical in detecting and mitigating risks that could jeopardize application integrity and user data. Frequent changes to configurations and different input parameters make testing more difficult. To ensure web applications are secure, it's important to use both automated tools and manual testing methods. This approach helps identify and fix potential threats effectively.

Test Oracle Problem

A test oracle is important because it can automate software testing and also improve its cost-effectiveness and reliability. A test oracle is a tool that evaluates whether a test case

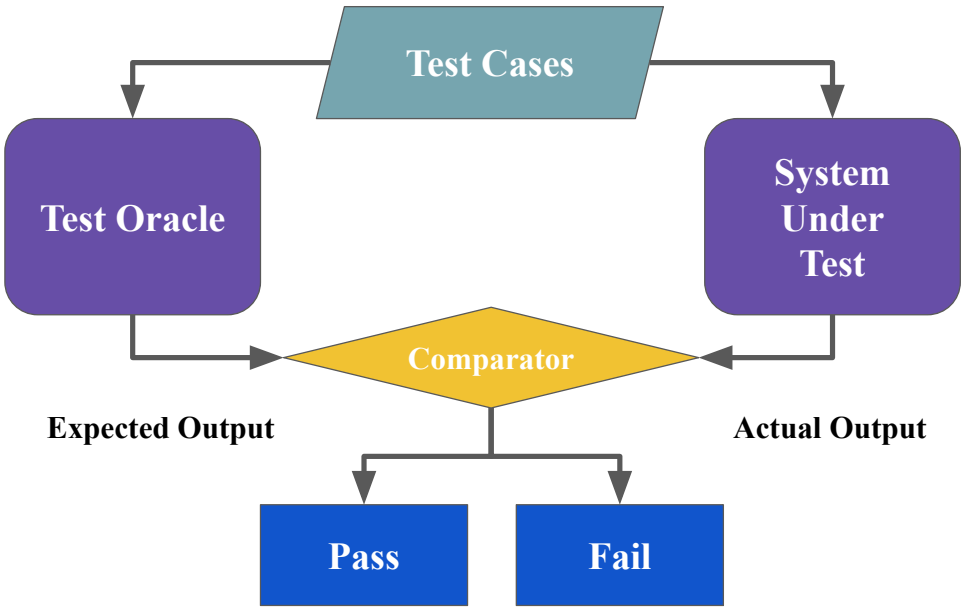

Figure 2: Overview of the mechanism of the test oracle for a program under test

has passed or failed [7]. William E. Howden [24] was the pioneer in introducing test oracles. Most of the time, a tester can function as an oracle, though at times, an oracle may indicate a specification or another application. Figure 2 provides an overview of how a test oracle operates.

One of the most challenging events in software testing is the oracle problem [7], also known as the test oracle problem. This problem mainly refers to the fact that identifying the correct output for a specific input is difficult for a system [7]. Due to its complexity, the oracle problem is very common in scientific and security software. Therefore, verifying the correctness of the output usually depends on manual testing. However, manually testing any system can be prone to errors, may miss subtle faults, and is time-consuming.

Due to the lack of a test oracle in the testing of scientific software and software that can be vulnerable to security threats, it is very difficult to determine whether a test has passed or failed. The lack of the test oracle and its intricate implementation can lead to the test oracle

problem. Test oracle problem complicates the detection of subtle faults and isolated errors in a system [7]. Such complications can especially affect the correctness and resilience of the software system. A domain expert can manually identify whether the output generated by the system under test is accurate for a given set of inputs in such cases[7]. Sanders et al., [6] noted that because of insufficient knowledge of software engineering, scientists often conduct testing inconsistently. Therefore, a test oracle is needed to automate software testing techniques to simplify the testing process.

Metamorphic Testing (MT)

MT is a testing technique that can alleviate the well-known test oracle problem [7]. The test oracle problem arises when determining the correctness of individual program output is too difficult or impossible. MT technique was developed by Chen et al. [8] in a technical paper in 1998. It builds upon specific predefined MRs properties by checking if a program satisfies them. These MRs explain how modifications to a program's input should influence its output [7]. If the output does not function as intended according to these MRs, it may suggest a flaw in the program [7]. MT is especially useful for detecting defects in programs where traditional test oracles are not available or practical [8]. By examining the relationships between inputs and outputs across many executions, MT can identify faults even when the correct result of each execution is not known [7].

The general steps for implementing MT are as follows (Figure 3) [8]:

- **Identify MRs:** Specify a set of MRs that the program under test must satisfy.
- **Create Initial Test Cases:** Create a set of test cases to act as the source inputs.
- **Generate Follow-Up Test Cases:** Apply the input transformations outlined in the MRs to generate follow-up test cases from the initial ones.

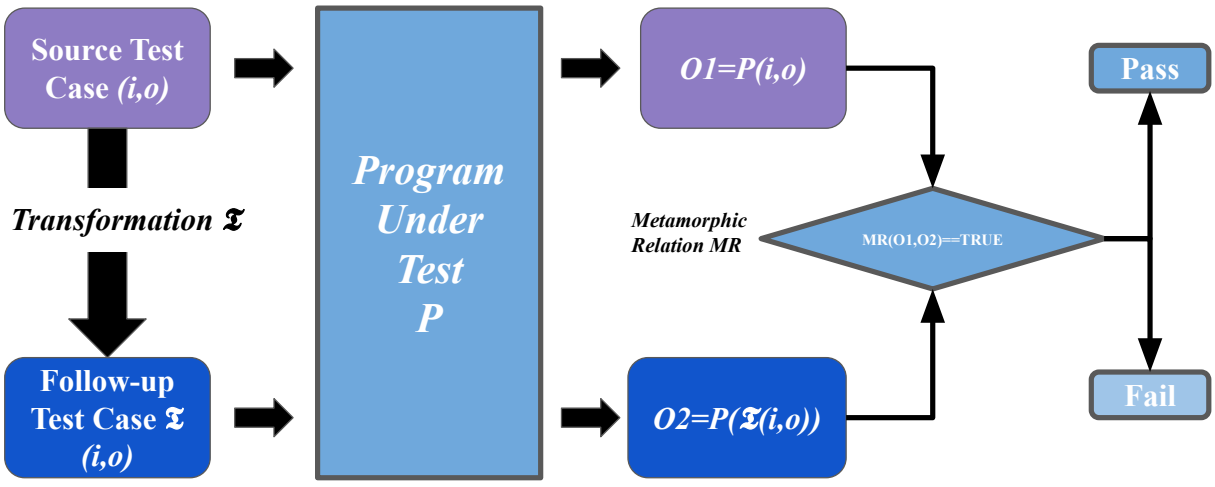

Figure 3: Overview of the Metamorphic Testing (MT) Process

- **Execute and Compare Outputs:** Run the source and follow-up test cases to see if the output changes align with the expected behavior outlined in the MRs. If a runtime violation of an MR occurs, it indicates a fault in the program.

A straightforward and commonly used example of MT is testing the SINE function. $y = \sin(x)$. For any input angle x, according to its property, adding 2π to the input should have an unchanged output. This means $y = \sin(x) = \sin(x + 2\pi)$, making it a valid MR. By utilizing this property, the function can be evaluated as follows:

- Develop a source test case with input x and output $\sin(x)$.
- Create a follow-up test case by implementing the transformation $x' = x + 2\pi$ and calculate $y = \sin(x') = \sin(x + 2\pi)$.
- Evaluate the outputs. If $sin(x') \neq sin(x)$, the MR is violated, indicating a fault in the sine function's implementation.

By systematically implementing these steps, MT offers an effective and efficient alternative for detecting faults in the absence of traditional test oracles [7].

Metamorphic Relations (MRs)

As previously mentioned, MRs establish crucial connections between a program's inputs and outputs. For example, M and N serve as the inputs for the matrix multiplication function, resulting in an initial output of O. If matrix M is multiplied by a positive integer p and matrix M remains unchanged after that, the output alters consequently $O' = (M \times p) \times N = M' \times N = O \times p$. Here M' is the follow-up input. Imagine a simple case of matrix multiplication,

$$M = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, N = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$M \times N = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix} = O$$

A positive integer of 2 is multiplied by the matrix M. After the multiplication of $M' = M \times 2$ and N, the output is O'.

$$M' = M \times 2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times 2 = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}, N = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$M' \times N = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 14 & 20 \\ 30 & 44 \end{bmatrix} = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix} \times 2 = O \times 2 = O'$$

Consequently, this method of multiplying a positive constant can be applied to develop a follow-up test case for each initial test case. Additionally, the output of the follow-up can be anticipated based on changes in the output.

MRs can be derived for vulnerable programs as well. Suppose a web application processes a user login credentials using an SQL query. This vulnerable login function can be written as the following:

```
query = "SELECT * FROM users WHERE username = '" + user_input +
"' AND password = '" + password_input + "';"
```

If the user input is not adequately sanitized, this implementation can be subjected to SQL Injection. Attackers can alter input values to bypass authentication, for example, by entering:

```
SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '';
```

Authentication is bypassed since `'1'='1'` is always true. A metamorphic relation (MR) for SQL Injection vulnerability testing can be identified where the input modification should not alter authentication behavior. Authentication should succeed with a valid username-password pair `(u, p)`. If the username field is changed to attempt SQL injection `(u' = u OR '1'='1')`, but the password remains unchanged, authentication should fail if the system is secure. By employing MT, we create test cases:

- Source test cases: If the input is `(username="admin", password="admin123")`, the expected output will be `Login Successful`.
- Follow-up test cases: Using the transformation specified for the MR, the input is altered to `(username="admin' OR '1'='1", password="admin123")`, if the output is `Login Successful` means the authentication has been passed and the system is not secure. If the MR is violated (e.g., if an unauthorized user is granted a login), the program is flagged as vulnerable to SQL injection.

Thus, if a program's output changes as anticipated after altering the input, it indicates that the function meets the defined metamorphic relation. Accurately identifying these relationships is crucial for effectively implementing MT on the relevant program. However,

testers frequently struggle to compile a set of MRs because they lack sufficient domain knowledge about the programs. Furthermore, identifying these relationships often demands manual effort, rendering the process labor-intensive and time-consuming.

Mutation Testing

Mutation testing is a widely utilized approach for evaluating the effectiveness of testing techniques and the sufficiency of test suites [25]. This method uses mutation operators on the program under test to introduce a variety of faults and create a set of mutant variants. A test case is intended to "kill" a mutant if it causes the mutant to exhibit behavior that differs from that of the original program [25]. The count of killed mutants is utilized to determine the mutation score (MS), indicating the effectiveness of a test suite in eliminating mutants [25]. The MS is calculated using the following formula:

$$MS = \frac{M_k}{M_t - M_e}$$

In the equation, the number of killed mutants is represented as M_k, the total number of mutants is represented as M_t, and the number of equivalent mutants is represented as M_e (i.e., mutants that consistently behave in the same manner). Automatically generated mutants are believed to be more equivalent to real-life faults than manually seeded ones. Thus, the mutation score effectively reflects the effectiveness of the testing technique.

Machine Learning

Machine learning (ML) has recently been applied to automate many software engineering activities, including software testing. ML primarily consists of algorithms that design models and analyze data [26]. Mohri et al. [27] describe ML algorithms as data-driven

methods. Additionally, it integrates fundamental computer science theories with concepts from statistics, probability, and optimization [26]. According to Shalev-Shwartz and Ben-David [28], ML focuses on enabling computers to learn; therefore, algorithmic ideas are crucial. The application of ML to address software testing problems is a relatively new area of research. Numerous researchers in this field have published their findings over the past two decades [23].

An ML classification algorithm, when an adequate amount of data is provided, builds a model and makes decisions based on that model. If more data are added, the machine recognizes patterns more accurately. Thus, the machine learning method teaches computer programs to make better decisions based on experience [26]. The examples used by a machine learning algorithm are divided into two groups: a *training set* and a *test set* [26]. The training set is used to create a predictive model using ML algorithms, while the test set assesses the performance of that predictive model [26].

Supervised Learning

Supervised learning is a machine learning approach that utilizes a set of examples to recognize a pattern [26]. This pattern is then employed to connect unseen inputs to the desired outputs. More specifically, a supervised learning algorithm analyzes the training data and generates a targeted function, which is used to assign labels to the unseen test data [26].

Support Vector Machine (SVM)

SVM algorithms are commonly utilized in practical applications for classification and regression tasks. The core concept of SVMs involves linear classification by creating a hyperplane within a high-dimensional space [29]. This hyperplane can distinguish examples of different classes in the training sets based on the information associated with their class

labels [29]. Numerous kernel functions, including the linear kernel, polynomial kernel, Gaussian kernel, and sigmoid kernel, can map original low-dimensional data sets into a higher-dimensional space [29]. SVMs identify a hyperplane within the data points that separate the different classes of data. A hyperplane maximizes the margin between itself and the support vectors, reducing classification error [29]. SVMs are inductive models, predicting testing sets based on mathematical models derived from the training sets [29]. They develop a classification model, or classifier, using labeled data. This model is then utilized to predict labels for data that has not been seen before and can also handle binary classification.

Neural Networks

A neural network is a machine learning model that makes decisions like those of the human brain. It uses processes that mimic how biological neurons work together to recognize patterns, evaluate choices, and make decisions [30].

Neural networks consist of layers of nodes called artificial neurons. These layers consist of an input layer, several hidden layers, and an output layer [30]. Each node connects to others and has its weight and threshold. When a node's output exceeds its specified threshold, that node is activated and sends data to the next layer in the network [30]. No data is transmitted to the next layer if the output falls below the threshold [30].

Neural networks depend on training data to learn and enhance their accuracy over time. Once fine-tuned, they become powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data quickly [30].

Graph Convolutional Neural Networks (GCNN)

Graph Neural Networks (GNNs) are among the most fascinating and rapidly developing architectures in the deep learning field. Designed to process data structured as graphs, GNNs

provide exceptional versatility and robust learning capabilities [31].

Among the various types of GNNs, Graph Convolutional Networks (GCNs) have become the most widely used and implemented models [31]. A Graph Convolutional Neural Network (GCNN) is a deep learning model tailored for graph-structured data, where nodes and edges symbolize complex relationships [31]. Unlike traditional CNNs, GCNNs gather information from neighboring nodes, allowing for the extraction of local and global structural features [31]. Key components include graph representation, where nodes with feature vectors connect through edges; graph convolutional layers, which iteratively update node embeddings by aggregating information from neighboring nodes; activation and normalization; and pooling and fully connected layers, which accommodate varying graph sizes and process final representations for classification [31].

Related Works

A crucial element in effectively implementing MT is having a set of MRs that streamlines the creation of test cases and validates test outputs. In the first ten years following the implementation of MT, most related studies relied on MRs identified in an ad-hoc manner [17]. The ad hoc identification process has its inherent limitations; however, these MRs showed a strong capability for fault detection. Because this identification mainly depends on the knowledge, skills, and experience of the tester, constructing enough high-quality MRs can be challenging. This challenge has led to numerous investigations into the systematic generation of MRs. MR generation has recently emerged as one of the most significant and widely discussed topics in MT research. The significant rise in publications about MR generation in the past decade clearly indicates this. This trend has been particularly pronounced in the last five years [17]. A total of 81 papers have investigated the systematic generation of MRs, with 63 published between 2019 and 2024 [17].

Several previous studies have examined the automatic generation and prediction of

MRs. Liu *et al.* introduced a novel method called *Composition of Metamorphic Relation (CMR)*, wherein the creation of new MRs is achieved by combining existing ones [32]. Similarly, Dong *et al.* conducted a related study in which *Compositional MR* was produced based on the speculative law of propositional logic [33]. Their empirical studies showed that the fault-detection effectiveness of composite MRs depended on their corresponding individual component MRs and the order of composition [33]. Additionally, it was observed that a composite MR appeared to outperform its individual component MRs, as the former could detect faults collectively [33]. Zhang *et al.* suggested a technique in which an algorithm searches for MRs represented as linear or quadratic equations [34]. Su *et al.* [35] introduced KABU, which is developed to automatically identify MRs by generating new inputs for the program being tested and then inferring relationships in a rule-based manner. Chen *et al.* presented a technique *DESSERT* where the DividE-and-conquer method was utilized to determine the categorieS, choiceS, and choicE MRs for generating test cases [36]. Afterward, Chen *et al.* again introduced another mechanism *METRIC*, where MRs were determined with category-choice framework [37].

Kanewala *et al.* [38, 39] conducted a study where machine learning methods could predict MRs in previously unseen programs. They have used features from the control flow graphs (CFGs) of the functions and then used them to create a predictive model [39]. Kanewala et al. [38] introduced *MRpred*, a method that is developed using a graph kernel-based machine learning approach to predicting MRs for programs that perform numerical calculations. The first step in this approach is to transform a function into its graphical representation, which uses the control flow and data dependency information of the program [38]. They then use a graph kernel function to compute a similarity score between graph representations of programs. The graph kernel values are provided to a support vector machines (SVMs) classification algorithm to build the predictive model [38]. They assessed the effectiveness of their proposed methods using a code corpus that includes 100 functions

taking numerical inputs and producing numerical outputs [38]. Six types of MRs were identified: Permutative, Additive, Multiplicative, Invertive, Inclusive, and Exclusive. Their findings indicate that graph kernels enhance the prediction accuracy of MRs compared to explicitly extracted features. Furthermore, their results reveal that control flow information of a program is more effective than data dependency information for predicting MRs. In some cases, both can contribute to increased accuracy.

In 2018, Hardin et al. extended the study for predicting MRs by implementing machine learning models that utilize support vector machines and the label propagation algorithm [40]. Their feature set includes path information across the graph representations of the program being tested. Their results indicate that label propagation outperformed the SVMs for 5 out of 6 MRs [40]. Furthermore, they conclude that unlabeled data enhances the prediction rate of a classifier [40]. These studies motivated the use of various types of applications as the subject programs for the experiments described in this paper. Many studies have used MT to test machine learning models. Moreover, there are few studies done by Faqeer et al. where the author used MT to verify different machine learning models [41, 42], clustering-based anomaly detection systems [43], and neural network-based detection systems [44, 45].

To identify vulnerabilities in web applications and assist with security testing, various techniques are employed, including static application security testing (SAST) [46], dynamic application security testing (DAST) [47], penetration testing, fuzz testing, risk-based testing [48], and manual code reviews [48]. However, these methods frequently face challenges such as limited test coverage, susceptibility to human error, and difficulties in addressing dynamic behaviors [14]. Another significant challenge is the variability in testing tools themselves, as updates and modifications can lead to differing security assessments and scores across versions [49]. MT can also be utilized to detect vulnerabilities and enhance security testing. The application of MT in security testing mainly focuses on the functional

testing of security components, such as encryption programs, code obfuscators, and login interfaces [10]. It also verifies low-level properties impacted by specific security vulnerabilities like the Heartbleed bug [10]. However, current research has been restricted to particular vulnerabilities and lacks automation for creating executable metamorphic test cases with efficiency [17]. Most methods necessitate manually implementing MRs [17] or defining them in imperative programming languages [17], which limits system-level security verification.

Many previous efforts have aimed to establish criteria for identifying appropriate MRs, but they remain predominantly qualitative. Developing further measurable methods is necessary to effectively assist in selecting suitable MRs and addressing the oracle problem. A promising avenue for future research indicates the process of selecting proper MRs with the systematic identification of MRs. This can create an advanced technique that accomplishes MR identification and selection. This method would identify MRs independently and systematically determine a set of MRs that excel in detecting various faults.

PREDICTING METAMORPHIC RELATIONS FOR MATRIX CALCULATION PROGRAMS

Approach

This section explains the enhancement of the graph kernel-based machine learning approach for programs that execute matrix calculations.

Method Overview

This method utilizes kernel-based machine learning algorithms to develop a classifier that predicts metamorphic relations (MRs) for programs not seen before. Figure 4 shows an overview of this method. The process involves two phases. The first phase is the training phase, which begins by creating a graph representation of functions written in Java. Each function is labeled with its respective metamorphic relations, which are identified manually for this experiment. Graph kernels are then employed to determine the similarity score for each graph pair. The result of the graph kernel function is a similarity score matrix, also known as a gram matrix. The final step of this phase involves training a classifier using kernel-based machine learning algorithms. The second phase is the testing phase, which retrieves the graph representation from an unseen function, following the same procedure as the training phase. Subsequently, the graph kernel function calculates the similarity scores between this new graph and all training graphs. Finally, the trained classifier predicts the MRs for the unseen function.

Function Representation - Control Flow Graph (CFG)

The first step of this method is to convert a function into its Control Flow Graph (CFG). This representation is specifically used since it allows the extraction of information about the sequence of operations performed in a control flow path that is directly related to

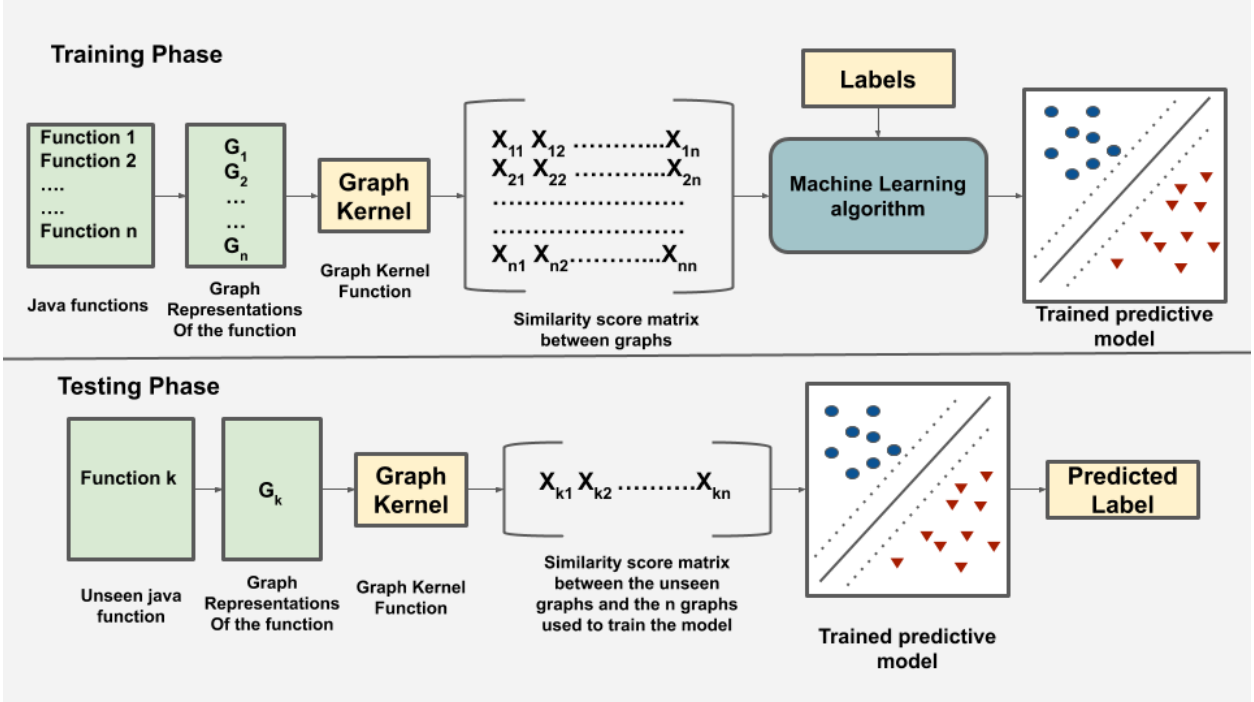

Figure 4: The overview of the graph kernel-based machine learning approach

the MRs satisfied by a given function.

A CFG is a directed graph, denoted as $G_f = (V, E)$, representing a function f. Each statement x in f corresponds to a node $v_x \in V$. The operation associated with each x is indicated by $label(v_x)$. If x and y are statements within f, then upon executing x, y will subsequently execute. Thus, e is defined as an edge where $e = (v_x, v_y) \in E$. The control flow of f is illustrated by all edges, with nodes v_{start} and v_{exit} representing the starting and exiting points, respectively [50].

We utilize the *Soot* [1] framework to generate Control Flow Graphs (CFGs). These CFGs are produced in *Jimple*, which is a typed 3-address intermediate representation of Java code, where each CFG node corresponds to an atomic operation [51]. After generating the CFGs, we label all nodes to indicate the specific operation performed at each one. Furthermore, we annotated each method call node within the CFG with its corresponding return types.

[1] https://www.sable.mcgill.ca/soot/

Figure 5 illustrates a function that calculates the scalar multiplication of a matrix alongside its post-processed CFG representation.

Graph Kernel-Based Method

Given that the suggested approach utilizes kernel-based supervised machine learning algorithms, this description includes explanations of the kernel method, graph kernel, and various machine learning techniques.

Kernel Method: Kernel methods represent a category of machine learning algorithms effective for pattern analysis, enabling tasks such as classification, clustering, and principal component extraction. [52]. The raw data representation can be transformed into a high-dimensional feature space using kernel methods. In this space, machine learning algorithms can identify linear relationships. [52]. Kernel functions facilitate the computation of inner products within this high-dimensional feature space directly from the raw data, eliminating the need to first map the data into the new feature space and measure the coordinates.[52]

Graph Kernel: Kernel-based graph comparison and classification methods have gained significant popularity in various fields, including biology [53], chemistry [54], and social network analysis [55]. Graphs serve as a natural representation for modeling this type of data, primarily focusing on structural characteristics. In 2003, Gartner et al. introduced graph kernels [56], a set of functions designed to compute similarity scores between pairs of graphs by comparing their structures. These precomputed similarity scores can then be utilized by kernel-based machine learning algorithms for classification tasks. Two types of graph kernels are employed in the graph kernel-based machine learning approach [38]. According to previous research by Kanewala et al. [38], a random walk kernel and a graphlet kernel were used to predict MRs, with findings indicating that the random walk kernel is more effective for MR prediction when utilizing CFG representation. This study evaluates

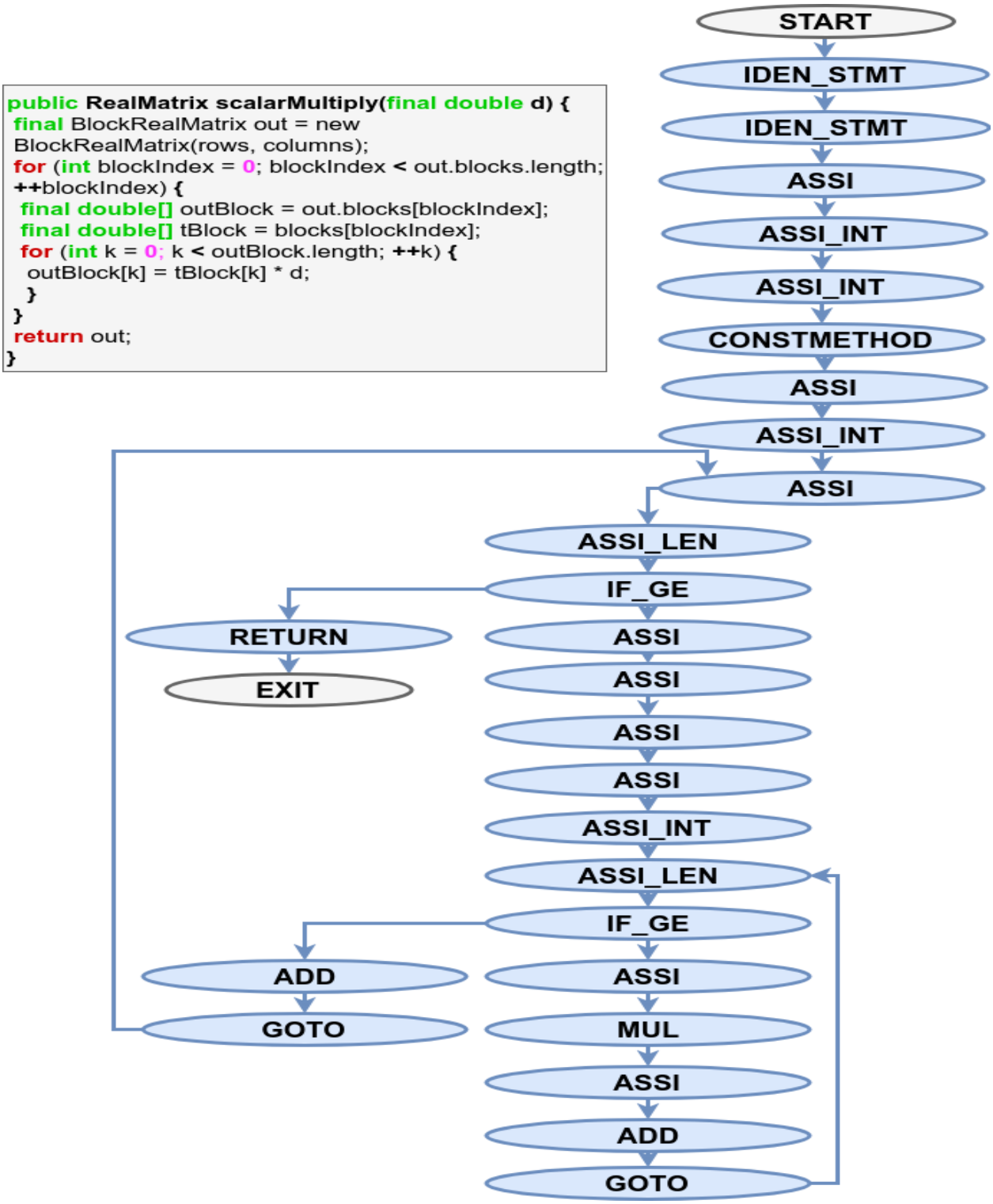

Figure 5: Function that performs scalar multiplication and its post-processed control flow graph (CFG) representation

new function types using both graph kernels.

Random Walk Kernel: Once the function representation of the programs is established, the next phase involves utilizing a graph kernel to determine the similarity between the graphs. The random walk kernel operates on the principle of counting the matching walks found in two graphs [38]. It calculates the similarity score by aggregating the scores of all pairs of walks, achieved by multiplying the similarity scores of their respective step pairs [38]. The similarity score for each pair of steps is initially determined by multiplying the similarity scores of the node and edge pairs that comprise the step [38].

Node labels influence the similarity score for a node pair as follows: if both labels are identical, the pair receives a score of one. If the labels differ, their operations determine the assigned score. When two node labels represent operations with similar characteristics (though not the same), they are given a similarity score of 0.75. For general tasks such as variable assignments or function calls, the score is 0.5. A score of 0.6 applies when the nodes indicate conditions, logic, or basic arithmetic operations. If a node signifies throw statements, it is assigned a score of 0.3, while a score of 0.1 is designated for all other scenarios.

Like node labels, edge labels determine the similarity score associated with a pair of edges. In this study, we utilized a single type of edge that indicates the flow of control among function operations. Therefore, the similarity score for any edge pair is consistently one for this analysis. The random walk kernel definition referenced here comes from prior work by Kanewala et al. [38].

Definition of the random walk kernel Consider the control flow graphs of two programs represented by $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Let's define two walks, $walk_1$ and $walk_2$, for G_1 and G_2 respectively. The walk for G_1 can be expressed as $walk_1 = (v_1^1, v_1^2, \ldots, v_1^{n-1}, v_1^n)$ where each $v_1^i \in V_1$ and $1 \leq i \leq n$. Similarly, the walk for G_2 is defined as $walk_2 =$

$(v_2^1, v_2^2, \ldots, v_2^{n-1}, v_2^n)$ with each $v_2^i \in V_2$ and $1 \leq i \leq n$. The edges of the graphs are represented as $(v_1^i, v_1^{i+1}) \in E_1$ and $(v_2^i, v_2^{i+1}) \in E_2$. Thus, we can now define the kernel value for the two graphs as:

$$k_{rw}(G_1, G_2) = \sum_{walk_1 \in G_1} \sum_{walk_2 \in G_2} k_{walk}(walk_1, walk_2) \tag{1}$$

Now the walk kernel k_{walk} can be defined as follows:

$$k_{walk}(walk_1, walk_2) = \prod_{i=1}^{n-1} k_{step}((v_1^i, v_1^{i+1}), (v_2^i, v_2^{i+1})) \tag{2}$$

In each step, the kernel will be defined by utilizing the two node pairs and edge pair values as follows:

$$k_{step}((v_1^i, v_1^{i+1}), (v_2^i, v_2^{i+1})) = k_{node}(v_1^i, v_2^i) * k_{node}(v_1^{i+1}, v_2^{i+1}) * k_{edge}((v_1^i, v_1^{i+1}), (v_2^i, v_2^{i+1})) \tag{3}$$

Here, k_{node} represents the node kernels that calculate the similarity score between pairs of nodes. It evaluates the similarity of node labels. For assigning the similarity score, we also take into account the grouping of nodes based on their operations. In this study, we categorized the mathematical operations by several properties: commutative and associative operations, labeled as $group_{com,aso}$; conditional statements, referred to as $group_{condition}$;

variable assignments, labeled as $group_{assign}$; and throw statements, denoted as $group_{throw}$.

$$k_{node}(v_i, v_j) = \begin{cases} 1, & \text{if } label(v_i) = label(v_j) \\ 0.75, & \text{if } group_{com,aso}(v_i) = group_{com,aso}(v_j) \text{ and } label(v_i) \neq label(v_j) \\ 0.6, & \text{if } group_{condition}(v_i) = group_{condition}(v_j) \text{ and } label(v_i) \neq label(v_j) \\ 0.5, & \text{if } group_{assign}(v_i) = group_{assign}(v_j) \text{ and } label(v_i) \neq label(v_j) \\ 0.3, & \text{if } group_{throw}(v_i) = group_{throw}(v_j) \text{ and } label(v_i) \neq label(v_j) \\ 0.1, & \text{if } group(v_i) \neq group(v_j) \text{ and } label(v_i) \neq label(v_j) \end{cases} \tag{4}$$

The edge kernel $k_{edge}$is defined as follow:

$$k_{edge}((v_1^i, v_1^{i+1}), (v_2^i, v_2^{i+1})) = \begin{cases} 1, & \text{if } label(v_1^i, v_1^{i+1}) = label(v_2^i, v_2^{i+1}) \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

The direct product graph method, as outlined by Gärtner et al. [57], is employed here. A modification introduced by Borgwardt et al. [58] is utilized for calculating all walks within the two graphs. The direct product graph of $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is defined as $G_1 \times G_2$. The nodes and edges of the direct product graph are defined as follows:

$$V_X(G_1 \times G_2) = \{(v_1, v_2) \in V_1 \times V_2\} \tag{6}$$

$$\begin{aligned} E_X(G_1 \times G_2) = \big\{ \big(\big((v_1^1, v_1^2), (v_2^1, v_2^2) \big) \big) \in V^2(G_1 \times G_2) : \big(v_1^1, v_1^2\big) \in E_1 \\ \wedge \big(v_2^1, v_2^2\big) \in E_2 \wedge label\big(v_1^1, v_1^2\big) = label\big(v_2^1, v_2^2\big) \big\} \end{aligned} \tag{7}$$

Based on the above product graph, the random walk kernel is defined as follows:

$$k_{rw}(G_1, G_2) = \sum_{i,j=1}^{V_X} \left[\sum_{n=0}^{\infty} \lambda^n A_X^n \right]_{ij} \tag{8}$$

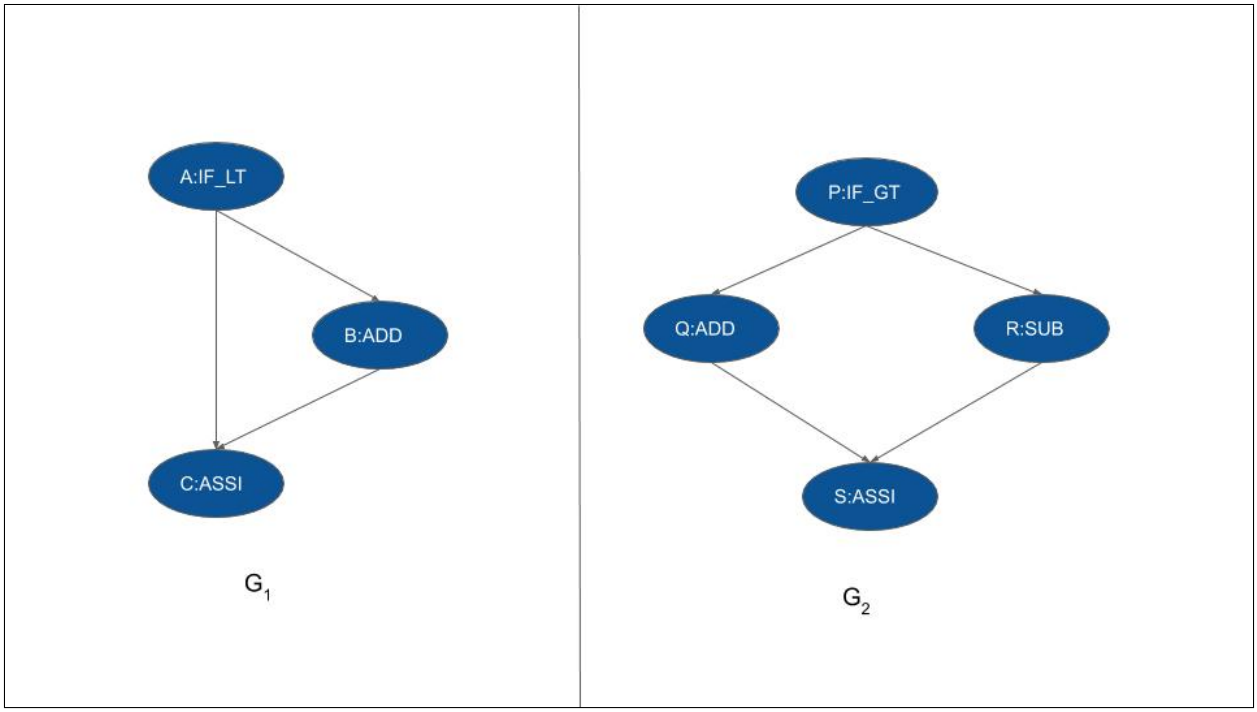

Figure 6: Random walk kernel computation for graph G_1 and G_2

In this context, A_X represents the adjacency matrix of the direct product graph. Here, $1 > \lambda \geq 0$ serves as a weighting factor, while n denotes the path length. The A_X matrix is adjusted as described to incorporate k_{step} defined earlier:

$$[A_X]_{((v_i,w_i),(v_j,w_j))} = \begin{cases} k_{step}((v_i,w_i),(v_j,w_j)), & \text{if } (v_i,w_i),(v_j,w_j) \in E_X \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

In Figure 6, G_1 and G_2 represent two functions in graph form, with the walk lengths limited to two. For G_1, there are walks of lengths one and two:

$$Length\,1 : A \to B, B \to C, A \to C$$

$$Length\,2 : A \to B \to C$$

For G_2, with walks of length one and two are

$$Length\,1 : P \rightarrow Q, P \rightarrow R, Q \rightarrow S, R \rightarrow S$$

$$Length\,2 : P \rightarrow Q \rightarrow S, P \rightarrow R \rightarrow S$$

Then it computes the similarity score between the two walks of the graphs.

$$k_{\text{walk}}(A \rightarrow B, P \rightarrow Q) = k_{\text{step}}((A, B), (P, Q))$$

$$\vdots$$

$$k_{\text{walk}}(A \rightarrow B \rightarrow C, P \rightarrow R, \rightarrow S) = k_{\text{step}}((A, B), (P, R)) \times k_{\text{step}}((B, C), (R, S))$$

Computation of the similarity between two steps are done as stated below-

$$k_{\text{step}}((A, B), (P, Q)) = k_{\text{node}}(A, P) \times k_{\text{node}}(B, Q) \times k_{\text{edge}}((A, B), (P, Q))$$

$$k_{\text{step}}((A, B), (P, R)) = k_{\text{node}}(A, P) \times k_{\text{node}}(B, R) \times k_{\text{edge}}((A, B), (P, R))$$

$$k_{\text{step}}((B, C), (R, S)) = k_{\text{node}}(B, R) \times k_{\text{node}}(C, S) \times k_{\text{edge}}((B, C), (R, S))$$

Similarity score between two nodes and edges are calculated as follows-

$$k_{\text{node}}(A, P) = 0.6 \text{ (two node labels have same conditions)}$$

$$k_{\text{node}}(B, Q) = 1 \text{ (two node labels are identical)}$$

$$k_{\text{node}}(B, R) = 0.6 \text{ (two node labels have basic arithmetic operations)}$$

$$k_{\text{edge}}((A,B),(P,Q)) = 1 \text{ (both edges have the same label)}$$

$$k_{\text{edge}}((A,B),(P,R)) = 1$$

This study employs the concept of the *random walk graph kernel* computation process.

Graphlet Kernel: In this study, we also utilize the Graphlet Kernel, a type of graph kernel method. While a random walk kernel struggles to effectively capture significant structures such as *if conditions*, graphlet kernels excel at this task by analyzing sub-graphs. These sub-graphs, known as graphlets, allow us to calculate the similarity score between pairs of graphs by comparing all graphlet pairs of a defined size [38]. To obtain the similarity score for the two graphs, we sum the similarity scores of all corresponding graphlet pairs present in each graph [38].

The similarity score for two graphlets is determined in the following manner: When the graphlets are isomorphic, the score is computed by multiplying the similarity scores of the corresponding node and edge pairs, similar to the random walk method. If they are not isomorphic, a similarity score of 0.1 is assigned to the pair. This calculation occurs when functions are represented as CFGs.

In this context, a single type of edge illustrates the flow of execution among function operations. Consequently, the similarity score for any pair of edges remains consistently one. The graphlet kernel defined for this study is explained in the earlier work of Kanewala et al. [38].

Definition of the Graphlet Kernel A graph $G = (V, E)$ consists of a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and a set of edges $E \subseteq V \times V$. A graph $H = (V_H, E_H)$ is considered a sub-graph of G if there exists an injective mapping $\alpha : V_H \to V$ such that for any two vertices v and w in V_H, the edge (v, w) is in E_H if and only if the edge $(\alpha(v), \alpha(w))$ is in E.

If H is a sub-graph of G, it is denoted as $H \sqsubseteq G$.

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are considered isomorphic if there exists a bijective mapping $g : V_1 \to V_2$ such that for any vertices u and v in V_1, the edge (u, v) is in E_1 if and only if the edge $(g(u), g(v))$ is in E_2. If G_1 and G_2 are isomorphic, it is denoted as $G_1 \simeq G_2$, and the function g is referred to as the isomorphism function.

Suppose M_1^k and M_2^k are the set of sub-graphs with size k for the graph representations G_1 and G_2. If $S_1 = (V_{s_1}, E_{s_1}) \in M_1^k$ and $S_2 = (V_{s_2}, E_{s_2}) \in M_2$, then the graphlet kernel, $k_{graphlet}(G_1, G_2)$, can be calculated as

$$k_{graphlet}(G_1, G_2) = \sum_{S_1 \in M_1^k} \sum_{S_2 \in M_2^k} \delta(S \simeq S_2) \tag{10}$$

, where

$$\delta(S_1 \simeq S_2) = \begin{cases} 1, & \text{if } S_1 \simeq S_2 \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

To consider the node labels and edge labels, we modified the graphlet kernel equation into

$$k_{graphlet}(G_1, G_2) = \sum_{S_1 \in M_1^k} \sum_{S_2 \in M_2^k} k_{subgraph}(S_1, S_2) \tag{12}$$

, where

$$k_{subgraph}(S_1, S_2) = \begin{cases} \prod_{v \in V_{s_1}} k_{node}(v, g(v)) * \prod_{(v_1, v_j) \in E_{s_1}} k_{edge}((v_i, v_j), (g(v_i), g(v_j))), & \text{if } S_1 \simeq S_2 \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

The graphlet kernel value for two programs, depicted in the graph-based representation outlined in this section, was obtained from equation 12. Like the random walk kernel, k_{node} from equation 13 is derived as in equation 4, and k_{edge} is calculated using equation 5.

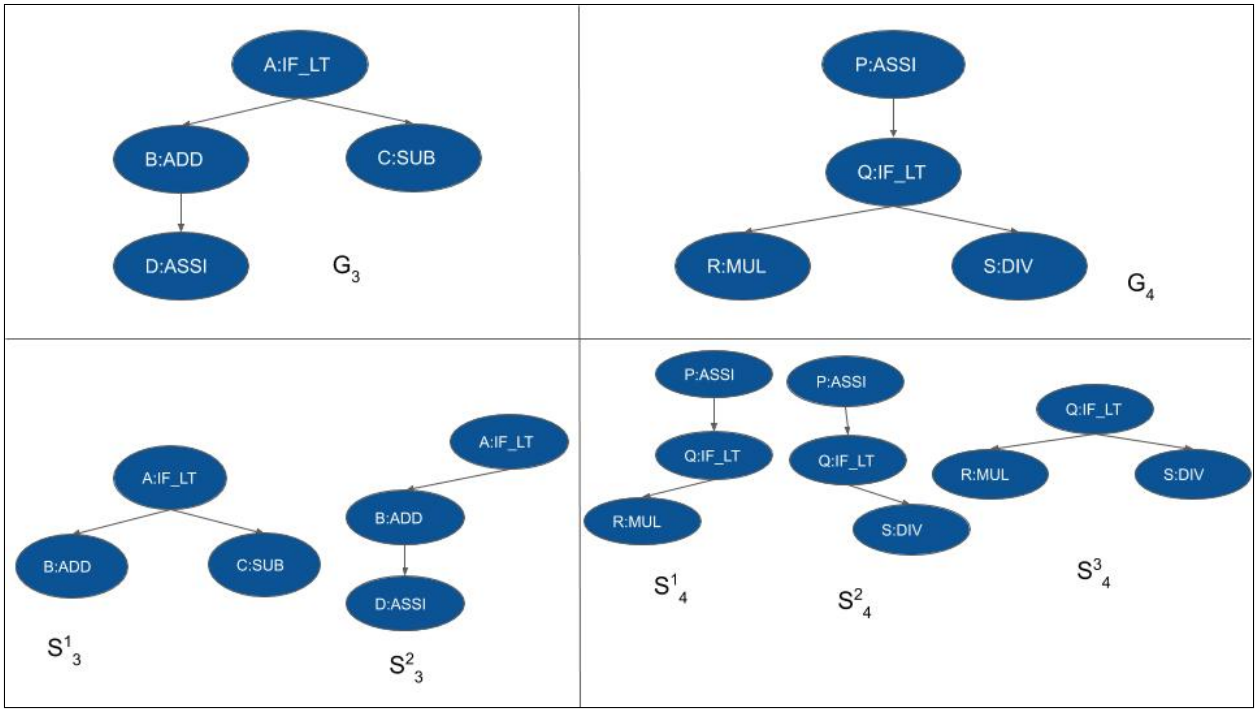

Figure 7: Graphlet kernel computation for graph G_3 and G_4

For example, in Figure 7, the graphs G_3 and G_4 illustrate two different functions. If we limit the graphlet size to 3, then the relevant subgraphs for G_3 and G_4 are S_3^1, S_3^2, S_4^1, S_4^2, and S_4^3.

Then, the similarity score between two graphs is computed using graphlets.

$$k_{\text{graphlet}}(G_3, G_4) = k_{\text{graphlet}}(S_3^1, S_4^1) + k_{\text{graphlet}}(S_3^1, S_4^2) + k_{\text{graphlet}}(S_3^1, S_4^3) + ... + k_{\text{graphlet}}(S_3^2, S_4^2) + k_{\text{graphlet}}(S_3^2, S_4^3)$$

The similarity between two graphlets is computed as follows-

$$k_{\text{graphlet}}(S_3^1, S_4^1) = 0.1$$

$$k_{\text{graphlet}}(S_3^1, S_4^2) = 0.1$$

$$k_{\text{graphlet}}(S_3^1, S_4^3) = k_{\text{node}}(A, Q) \times k_{\text{node}}(B, R) \times k_{node}(C, S) \times k_{\text{edge}}((A, B), (Q, R)) \times k_{\text{edge}}((A, C), (Q, S))$$

$$\vdots$$

$$k_{\text{graphlet}}(S_3^2, S_4^2) = k_{\text{node}}(A, P) \times k_{\text{node}}(B, Q) \times k_{node}(D, S) \times k_{\text{edge}}((A, B), (P, Q)) \times k_{\text{edge}}((B, D), (Q, S))$$

$$k_{\text{graphlet}}(S_3^2, S_4^3) = 0.1$$

Computation of the similarity score between two nodes and edges are shown below -

$$k_{\text{node}}(A, Q) = 1$$

$$k_{\text{node}}(B, R) = 0.6$$

$$k_{\text{node}}(C, S) = 0.6$$

$$\vdots$$

$$k_{\text{node}}(D, S) = 0.1$$

$$k_{\text{edge}}((A, B), (Q, R)) = 1$$

$$\vdots$$

$$k_{\text{edge}}((B, D), (Q, S)) = 1$$

This concept of *graphlet graph kernel* computation is utilized in this thesis.

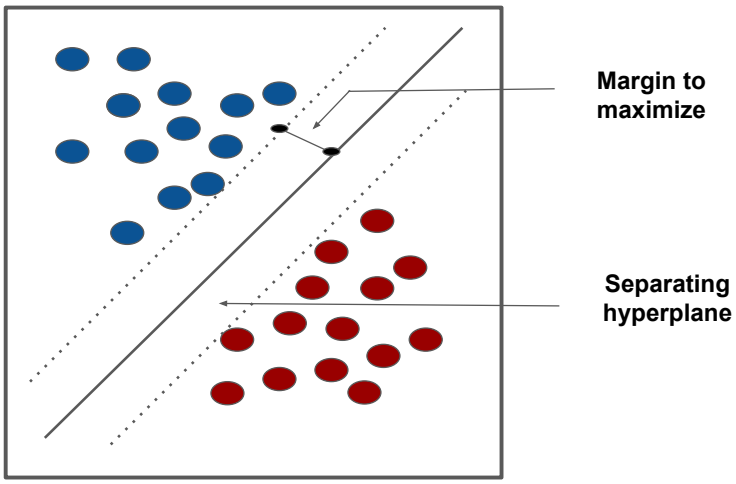

Figure 8: General classification hyperplane representation of SVM algorithm

Predictive Model

As previously mentioned, predicting MRs is formulated as a machine-learning problem. Machine learning algorithms are a well-established method for identifying linear relationships. This section focuses on supervised learning, which employs several commonly used algorithms; this thesis utilizes kernel-based support vector machines.

Support Vector Machines: SVM algorithms are widely utilized in real-world scenarios for classification or regression tasks. The fundamental idea of SVMs involves linear classification by generating a hyperplane within a high dimensional space [29]. This hyperplane can distinguish examples of different classes in the training sets based on the information of their associated class labels [29]. Many accessible kernel functions, such as the linear kernel, polynomial kernel, Gaussian kernel, and sigmoid kernel, can map the original low-dimensional data sets into a higher-dimensional space [29]. As shown in Figure

8, SVMs find a hyperplane amongst the data points that separate the classes of data. A hyperplane maximizes the margin between itself and the support vectors, decreasing the classification error [29]. SVMs are inductive models, as they predict the testing sets based on mathematical models obtained from the training sets [29]. It creates a classification model or a classifier using labeled data. The model is used to predict labels for previously unseen data. It can also be used for binary classification [29].

This thesis focuses on graph kernel functions, detailed in this section. The calculated graph kernel values are fed into SVMs, which receive a binary label indicating whether a specific function meets the designated MR. The SVMs utilize this information to build a model that predicts if a new function will satisfy the given MR. For this study, the SVM implementation from the scikit-learn[2] toolkit was employed. Figure 4 illustrates an overview of how the predictive model is constructed.

Experimental setup

This section outlines the code corpus and the metamorphic relations employed in this study, along with a description of the evaluation procedure.

Code corpus

A total of 93 functions handling matrix calculations are utilized to assess the efficacy of the proposed method for predicting MRs. These functions are sourced from open-source projects, including the Apache Commons Math Library[3], la4j (Linear Algebra for Java)[4], and JAMA (Java Matrix package)[5], They perform a variety of matrix operations such as addition, subtraction, multiplication, and searching (for example, retrieving a column matrix or a row

[2]http://scikit-learn.org/stable/
[3]http://commons.apache.org/proper/commons-math/javadocs/api-3.6/
[4]http://la4j.org/apidocs/
[5]https://math.nist.gov/javanumerics/jama/doc/

matrix). While many functions offer equivalent functionality, their implementations differ. For instance, both the Array2DRowRealMatrix and OpenMapRealMatrix classes include matrix multiplication functions, but they are coded differently. In such instances, both implementations are incorporated into the code corpus.

The suggested strategy utilizes the graph kernel method to compute similarity scores between graph pairs; as a result, control flow graphs (CFGs) for these functions are created using the Soot framework. Each node within the generated CFG is designated with labels to indicate its corresponding role operation.

Metamorphic Relations

In this study, ten MRs have been manually identified that are broadly relevant to matrix calculations. These MRs serve as class labels for the classification model. Below is the list of MRs selected for this research:

1. MR1 - ScalarAddition: When a positive constant is added to the positive source input matrix, resulting in a follow-up input matrix, the total of the elements in the follow-up output matrix will be greater than or equal to that of the source output matrix. For example, A is an input matrix, and b is a positive constant. For MR1, the follow-up input is $A^{'}$, where $\forall i, j, a^{'}_{i,j} = a_{i,j} + b$; therefore, the expected relation among the follow-up and the source output is $\sum_i \sum_j o^{'}_{(i,j)} \geq \sum_i \sum_j o_{(i,j)}$.

2. MR2 - AdditionWithIdentityMatrix: When an identity matrix matching the dimensions of the positive source input matrix is added to create a subsequent input, the total of the elements in the follow-up output matrix should be equal to or exceed the total of the elements in the source output matrix. For example, A is an input matrix, and I is an identity matrix. For MR2, the follow-up input is A', where $\forall i, j, a^{'}_{i,j} = i_{i,j} + a_{i,j}$; therefore, the expected output relation is $\sum_i \sum_j o^{'}_{(i,j)} \geq \sum_i \sum_j o_{(i,j)}$.

3. MR3 - ScalarMultiplication: When a positive constant is multiplied by the positive source input matrix to form a follow-up input matrix, the total of the follow-up output matrix elements must be greater than or equal to that of the source output matrix elements. For example, A is an input matrix, and b is a positive constant. For MR3, the follow-up input is $A^{'}$, where $\forall i, j, a^{'}_{i,j} = b \times a_{i,j}$; therefore, expected output relation is $\sum_i \sum_j o^{'}_{(i,j)} \geq \sum_i \sum_j o_{(i,j)}$.

4. MR4 -MultiplicationWithIdentityMatrix: When an identity matrix matching the dimensions of the positive source input matrix is multiplied element-wise to produce a follow-up input, the sum of the elements in the follow-up output matrix must be less than or equal to the sum of the elements in the source output matrix. For example, A is an input matrix, and I is an identity matrix. For MR4, the follow-up input is A', where $\forall i, j, a^{'}_{i,j} = i_{i,j} \times a_{i,j}$; therefore, the expected output relation is $\sum_i \sum_j o^{'}_{(i,j)} \leq \sum_i \sum_j o_{(i,j)}$.

5. MR5 - Transpose: When a follow-up input is created by transposing the positive source input matrix, the total of the elements in the follow-up output matrix should match the total of the elements in the source output matrix. For example, A is an input matrix. For MR5, the follow-up input is A', where $\forall i, j, a^{'}_{i,j} = a_{j,i}$; therefore, the expected output relation is $\sum_i \sum_j o^{'}_{(i,j)} = \sum_i \sum_j o_{(i,j)}$.

6. MR6 - MatrixAddition: When the positive source input matrix is summed with itself to create the follow-up input, the total of the elements in the follow-up output matrix should be greater than or equal to that of the source output matrix. For example, A is an input matrix. For MR6, the follow-up input is $A' = A + A$; therefore, expected output relation is $\sum_i \sum_j o^{'}_{(i,j)} \geq \sum_i \sum_j o_{(i,j)}$.

7. MR7 - MatrixMultiplication: When the positive source input matrix multiplies itself

to produce the follow-up input, the sum of the elements in the follow-up output matrix must be greater than or equal to the sum of the elements in the source output matrix. For example, A is an input matrix. For MR7, the follow-up input is $A' = A \times A$; therefore, the expected output relation is $\sum_i \sum_j o'_{(i,j)} \geq \sum_i \sum_j o_{(i,j)}$.

8. MR8 - PermuteColumn: When the columns of the positive source input matrix are rearranged to produce the follow-up input, the total of the elements in the follow-up output matrix must equal the total of the elements in the source output matrix. For example, A is a input matrix with $j = 1, 2, 3, .., n$ columns. For MR8, the follow-up test case is A' after permuting the column positions; therefore, the expected output relation is $\sum_i \sum_j o'_{(i,j)} = \sum_i \sum_j o_{(i,j)}$.

9. MR9 - PermuteRow: When the rows of the positive source input matrix are rearranged to form the follow-up input, the total of the elements in the follow-up output matrix must match the total of the elements in the source output matrix. For example, A is a input matrix with $j = 1, 2, 3, .., n$ rows. For MR8, the follow-up test case is A' after permuting the row positions; therefore, the expected output relation is $\sum_i \sum_j o'_{(i,j)} = \sum_i \sum_j o_{(i,j)}$.

10. MR10 - PermuteElement: When the elements of the positive source input matrix are rearranged to form the subsequent input, the total of the follow-up output matrix's elements must equal the total of the source output matrix's elements. For example, A is a input matrix with $i = 1, 2, 3, .., n$ rows and $j = 1, 2, 3, .., n$ columns. For MR10, the follow-up test case is A' after permuting elements. Therefore, the expected output relation is $\sum_i \sum_j o'_{(i,j)} = \sum_i \sum_j o_{(i,j)}$.

We manually assessed whether the programs in our code corpus meet these 10 MRs. Moreover, a lab partner helped me verify the MRs by cross-checking them. Identifying the

Table 2: Number of positive and negative instances for each metamorphic relation

Metamorphic Relation	Positive instances	Negative instances
MR1	77	16
MR2	25	68
MR3	77	16
MR4	39	54
MR5	36	57
MR6	12	81
MR7	20	73
MR8	55	38
MR9	55	38
MR10	55	38

MRs for this study's code corpus was quite time-consuming. Table 2 displays the number of positive and negative instances for each MR; positive indicates a function meets the MR, while negative indicates that it does not.

Evaluation Procedure

Firstly, I standardize each kernel to ensure that every example has a unit norm through the cosine normalization [59]. Below is the equation for the normalization applied in this study.

$$k'_{graph}(G_1, G_2) = \frac{k_{graph}(G_1, G_2)}{\sqrt{k_{graph}(G_1, G_1) k_{graph}(G_2, G_2)}}$$

We implement a stratified *train, validation, and test framework* to assess the effectiveness of MR predictions. As shown in Figure 9 illustrates the procedure involves splitting the data into 10 folds, ensuring that each fold maintains a similar ratio of positive and negative instances as in the original dataset. These folds are further categorized into three subsets: Train data, Test data, and Validation data. The prediction model is developed using the precomputed kernel values from the Train data. The validation dataset was utilized to determine the parameters for the predictive model. The parameters included: (1) the regularization parameter C for the support vector machines (SVMs), (2) the path weighting

Figure 9: Representation of the *stratified train, validation and test* setup for SVMs model

factor λ in the random walk kernel, constrained by $0 \leq \lambda < 1$ (meaning each walk of length n is weighted by λ^n, with n ranging from 1 to 10), and (3) the size of the sub-graphs (2, 3, or 4) used in the graphlet kernel. The parameter values chosen from the validation set were then applied to build the binary predictive model for estimating the MRs in the test data. This train-validation-test process was repeated ten times, allowing each fold to serve as the validation and test set while the remaining folds were used for training the model, thus minimizing biases in the fold divisions. Consequently, each iteration employed eight folds for training, one for validation, and one for testing.

Area Under the Receiver Operating Characteristic Curve

For the evaluation of this study, we used *Area Under the receiver operating characteristic Curve (AUC)* as our measurement metric. AUC indicates the likelihood that the predictive model prioritizes a randomly selected positive example (such as a program with a specific

Table 3: Best C and best λ parameter values for selecting predictive model for each metamorphic relation on validation set

Metamorphic Relation	Best C	Best λ
MR1	0.1	0.5
MR2	10	0.9
MR3	1000	0.7, 0.6
MR4	100, 1000	0.7
MR5	10, 100, 1000	0.8, 0.9
MR6	1	0.8, 0.9
MR7	1000	0.5
MR8	1000	0.3
MR9	1000	0.9, 0.3
MR10	1000	0.7, 0.8

Table 4: Best C and best graphlet size parameter values for selecting a predictive model for each metamorphic relation on validation set

Metamorphic Relation	Best C	Best graphlet size
MR1	100	3
MR2	1	3
MR3	0.1, 1, 10, 100	2
MR4	100, 1000	3
MR5	10	2
MR6	100	2
MR7	100	2
MR8	0.1	3
MR9	1000	2
MR10	100	2

MR label) over a randomly chosen negative example [60]. AUC ranges from 0 to 1, with higher values signifying improved performance; a score of 0.5 indicates a random classifier. AUC is the evaluation metric in this experiment since it is independent of the classifier's discrimination threshold. It is viewed as a more effective measure for assessing learning algorithms than metrics like accuracy. [60].

Results

The random walk kernel and graphlet kernel performance are evaluated with SVMs to develop the predictive model.

Predictive Model Selection

The validation set helps determine the optimal parameter values for identifying the best predictive model. Table 3 and Table 4 show the λ, graphlet size and C values for each MR on the validation set associated with the highest AUC value. For the MRs in Table 10, the chosen value for the parameter C does not significantly impact the prediction accuracy, as most are set to 1000, except for MR1, MR2, and MR3. However, the best value for λ varies among different MRs. Seven out of the ten MRs (i.e., MR2, MR3, MR4, MR5, MR6, MR9, and MR10) have the best λ value greater than 0.5. A higher λ value means the random walk kernel assigns more weight to longer paths, as defined by criterion [38]. Among the MRs (specifically, MR1, MR7, MR8, and MR9), the best λ values range from 0.3 to 0.5. This suggests that for these four MRs, longer paths in the CFGs play a more crucial role in predicting MRs than other paths.

In the graphlet kernel, the chosen value for the parameter C varies across all MRs. The optimal C value of 100 is identified for MR1, MR3, MR4, MR6, MR7, and MR10. Conversely, MR2, MR5, MR8, and MR9 exhibit different optimal C values, suggesting that this parameter C influences prediction accuracy. Additionally, the ideal graphlet size differs among various MRs. Six out of the ten MRs (MR3, MR5, MR6, MR7, MR9, and MR10) have the best graphlet size of 2, while four MRs (MR1, MR2, MR4, and MR8) show a value of 3 for the best graphlet size.

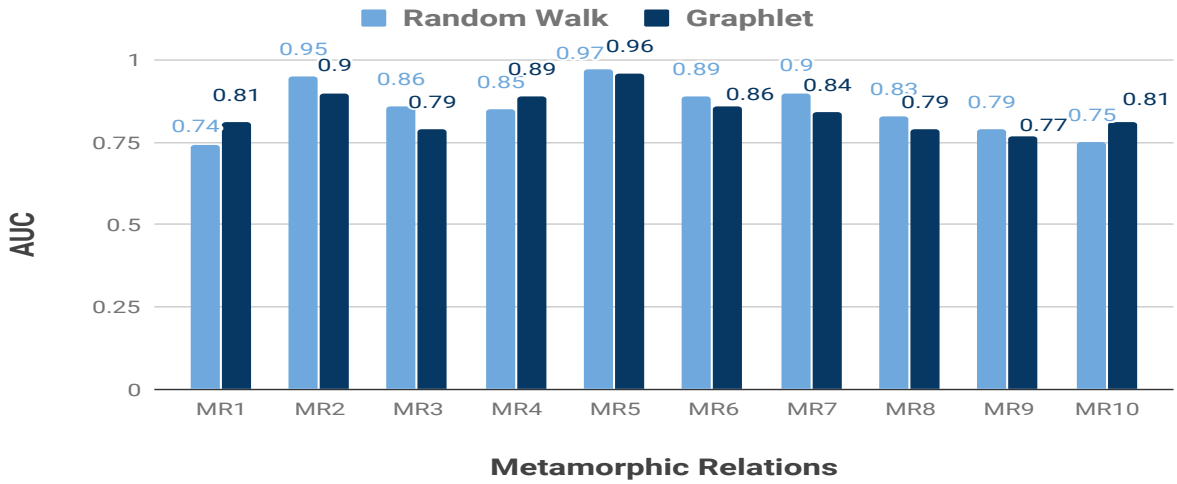

Figure 10: Prediction *AUC* score for *Random walk graph kernel* and *Graphlet kernel* on test set using the predictive model with best parameters

Comparison Between the Graph Kernels

Two types of graph kernels, the random walk kernel and the graphlet kernel, are utilized to train the predictive model. Figure 10 illustrates the AUC scores for the test dataset using both kernels. Among the 10 MRs, 7 (specifically MR2, MR3, MR5, MR6, MR7, MR8, and MR9) perform better with the random walk kernel, while MR1, MR4, and MR10 show improved results with the graphlet kernel. The highest AUC score of 0.97 is obtained for MR5 when using the random walk kernel. Table 2 presents the ratio of positive and negative instances for each MR, indicating that this ratio influences the prediction accuracy. Notably, MR2, MR5, and MR7 exhibit a low number of positive instances and a high number of negative instances, with their AUC scores exceeding 0.9 using the random walk kernel. Other MRs also achieved AUC values above 0.73, suggesting that effective predictive models have been created for all MRs. An MR represents a property that defines the relationship between outputs generated by multiple executions of the function, which may correlate with the MRs. This study concludes that the random walk kernel, which leverages execution traces to compare two functions, outperforms the graphlet kernel.

Conclusion & future work

The metamorphic testing method proves invaluable for evaluating programs lacking a test oracle. Its success largely relies on the selection of the metamorphic relations (MRs) used for testing. However, identifying these MRs is predominantly a manual process. This study builds on prior research, utilizing the random walk kernel to forecast MRs for functions that perform matrix calculations. Our findings indicate that the random walk kernel is effective in predicting MRs for such functions.

Looking ahead, we aim to expand the variety of functions included in this study. Additionally, we are open to introducing new types of MRs, particularly those designed for matrix calculations. We also intend to broaden the scope of MR predictions beyond just the function level.

MRPREDT: USING TEXT MINING FOR METAMORPHIC RELATION PREDICTION

Contribution of Authors and Co-Authors

Manuscript in Chapter 5

Author: Karishma Rahman

Contributions: Problem identification and proposing solution, running experiment, manuscript writing, creating tables and figures. Primary writer.

Co-Author: Indika Kahanda

Contributions: Contribution in manuscript editing/writing, provided feedback, guidance and advice.

Co-Author: Upulee Kanewala

Contributions: Contribution in manuscript editing/writing, provided feedback, guidance and advice.

Manuscript Information

Karishma Rahman, Indika Kahanda and Upulee Kanewala

Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops

Status of Manuscript:
____ Prepared for submission to a peer-reviewed journal
____ Officially submitted to a peer-reviewed journal
____ Accepted by a peer-reviewed journal
_X_ Published in a peer-reviewed journal

IEEE/ACM
2020/6/27
10.1145/3387940.3392250

Abstract

Metamorphic relations (MRs) are an essential component of metamorphic testing (MT) that highly affects its fault detection effectiveness. MRs are usually identified with the help of a domain expert, which is a labor-intensive task. In this work, we explore the feasibility of a text classification-based machine learning approach to predict MRs using their program documentation as the sole input. We compare our method to our previously developed graph kernel-based machine learning approach and demonstrate that textual features extracted from program documentation are highly effective for predicting metamorphic relations for matrix calculation programs.

Introduction

Since its first introduction in 1998, Metamorphic testing (MT) has evolved as an effective testing technique for testing programs that face the *oracle problem.* Further, MT has led to revealing previously unknown faults in diverse applications such as compilers [61], search engines [62], and Google map navigation [63]. At the center of MT are metamorphic relations (MRs), which are necessary properties of the program under test (PUT) and specify relationships between multiple inputs and their corresponding outputs.

For example, consider a program that accepts a list of real numbers and sorts them in ascending order. Imagine that this program is provided with a list of 50,000 numbers. How do you determine the output produced by the sorting program is correct? Even though it is hard to determine whether the produced output is correct in this instance, we can develop relationships between the outputs of some related inputs. For example, from our knowledge about sorting, we know that if we permute the original input and supply it to the sorting program, it should produce the same ordering as before. This property of sorting can be used as an MR to test the sorting program. In MT, the original list of inputs is known

Figure 11: Overview of the approach.

as the *source test case*, and the list obtained by randomly permuting the original list is known as the *follow-up test case*. After executing the source and follow-up test cases on the sorting program, the produced outputs are checked against the MR; in this case, to verify the produced outputs have the same ordering. If they are not, there is a fault in the sorting program.

Often, MR identification is performed manually and requires interaction with domain experts, especially when testing scientific programs. Therefore, this process can be a labor-intensive task that is often error prone [38]. Thus, developing automated methods for identifying MRs can improve the efficiency and effectiveness of MT. To this end, in this paper, we investigate utilizing text mining to extract information from various documentation sources associated with a program and use machine learning techniques to predict MRs for unseen programs automatically.

Text mining techniques are often used to explore and analyze a vast amount of

unstructured text data and identify patterns informative for a given task [64]. This study presents the development of a method for automatically predicting MRs for a given program function through categorizing text data associated with those programs (see Figure 11). In particular, the text data used are collected from the Javadocs of the Java programs. Javadoc contains the application programming interface documentation from the Java source code [65]. We hypothesis that due to the close relationship of the Javadocs contents with the program functionality, textual features extracted from the documentation is highly informative for predicting MRs associated with those programs. Therefore, this study aims to demonstrate the effectiveness of using text mined features generated from program documentation to predict MRs.

This paper is organized as follows: Section 2 provides the background knowledge of the methods and techniques used for the experiment. Section 3 explains the methodology of the Text Mining based machine learning approach to identify MRs for a function. Section 4 discusses the results obtained by using the approach introduced in the study. Section 5 concludes the paper by pointing out future works.

Related Work

Several automated methods have been developed for MR prediction in previous work. Kanewala et al. [38] introduced *MRpred*, a method that uses a graph kernel-based machine learning approach to predict metamorphic relations for programs that perform numerical calculations. The initial step of this approach is to transform a function into its graph representation modeling the control flow and the data dependency information of the program [38]. Then they use a graph kernel function to compute a similarity score between two programs represented in the graph representation mentioned above. The computed graph kernel values are then provided to a support vector machine (SVM) classification algorithm to create the predictive model, which is used for binary classification [38].

```
/**
 * Adds given {@code value} (v) to every element of this matrix (A).
 *
 * @param value the right hand value for addition
 *
 * @return A + v
 */
public Matrix add(double value) {
    MatrixIterator it = iterator();
    Matrix result = blank();

    while (it.hasNext()) {
        double x = it.next();
        int i = it.rowIndex();
        int j = it.columnIndex();
        result.set(i, j, x + value);
    }

    return result;
}
```

Figure 12: Raw data of a program which adds a value to the matrix elements.

They used a code corpus containing 100 functions that take numerical inputs and produce numerical outputs, to evaluate the effectiveness of their proposed methods [38]. Six MRs are identified; Permutative, Additive, Multiplicative, Invertive, Inclusive, and Exclusive. Their results show that graph kernels improve the prediction accuracy of MRs when compared with explicitly extracted features. Their results also show that control flow information of a program is more effective than data dependency information for predicting MRs, but sometimes, both of them can contribute to increasing the accuracy.

In one of our previous studies [12], we applied MRpred for predicting three high-level categories of MRs (i.e., Permutative, Additive, and Multiplicative) for matrix-based programs. Our results show that the random walk kernel can effectively predict these MRs [12]. This study motivated us to use matrix-based programs as the subject programs for the experiments described in this paper.

Further, in several past studies, researches have used different text analysis techniques. They have investigated many ways to improve the software testing process using text analysis techniques. In [66], the authors have conducted a mapping study where they listed the activities of software testing, which are improved by using text analysis techniques. They are static black-box test-case prioritization, robustness testing, test case generation, and test case prioritization [66].

Our work described in this paper also focuses on the automated identification of MRs for programs but using a different source of data, which is the various documentation sources associated with the program. In particular, we use text mining techniques to classify the Javadoc associated with a program and use machine learning to predict MRs for previously unseen programs. To the best of our knowledge, this is the first such study.

Table 5: The Metamorphic Relations used in the study

Metamorphic Relation	**Change made to the input**	**Expected change in the output**
MR1: Permutation of row	Change the row order of the a matrix	Output size will remain same
MR2: Permutation of column	Change the column order of the matrix	Output size will remain same
MR3: Permutation of elements	Change the element position of the matrix	Output size will remain same
MR4: Matrix addition	Adding another matrix to the input matrix	Elements value will increase or remain same.
MR5: Scalar addition	Adding a value to the matrix	Element values will increase or remain same
MR6: Addition with the Identity matrix	Adding Identity matrix to the input matrix	Only diagonal element value will change or output will increase
MR7: Matrix multiplication	Multiplying another matrix to the input matrix	Elements value will increase.
MR8: Scalar multiplication	Multiplying a value to the matrix	Elements value will increase or remain same.
MR9: Element by element multiplication with the Identity matrix	Multiplying Identity matrix element by element to the input matrix	Diagonal element will be same or output will remain same
MR10: Transpose	Transpose the input matrix	Diagonal element will be same or output will remain same

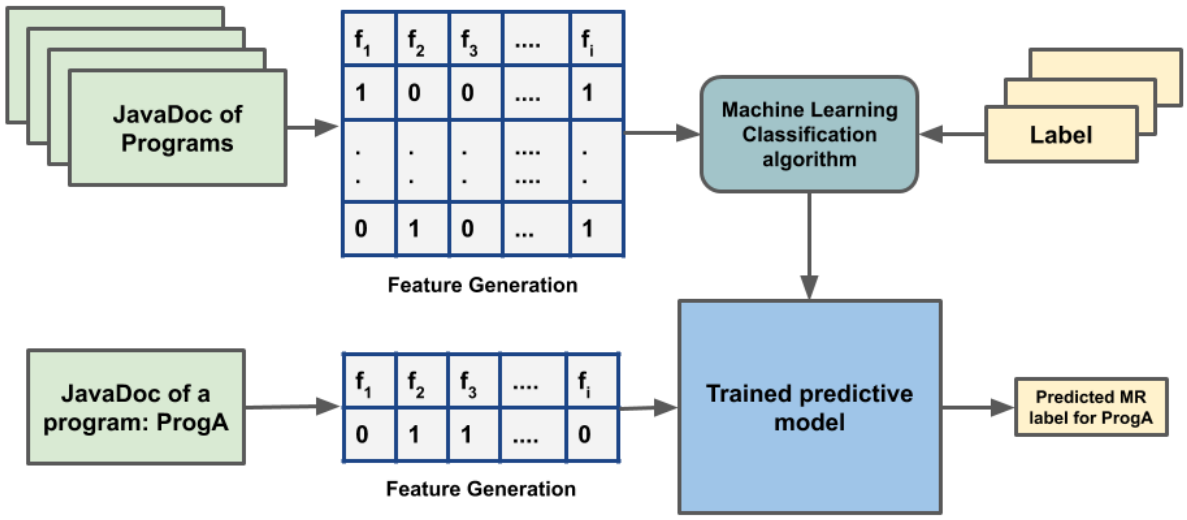

Figure 13: Text classification approach for predicting MRs for Java programs.

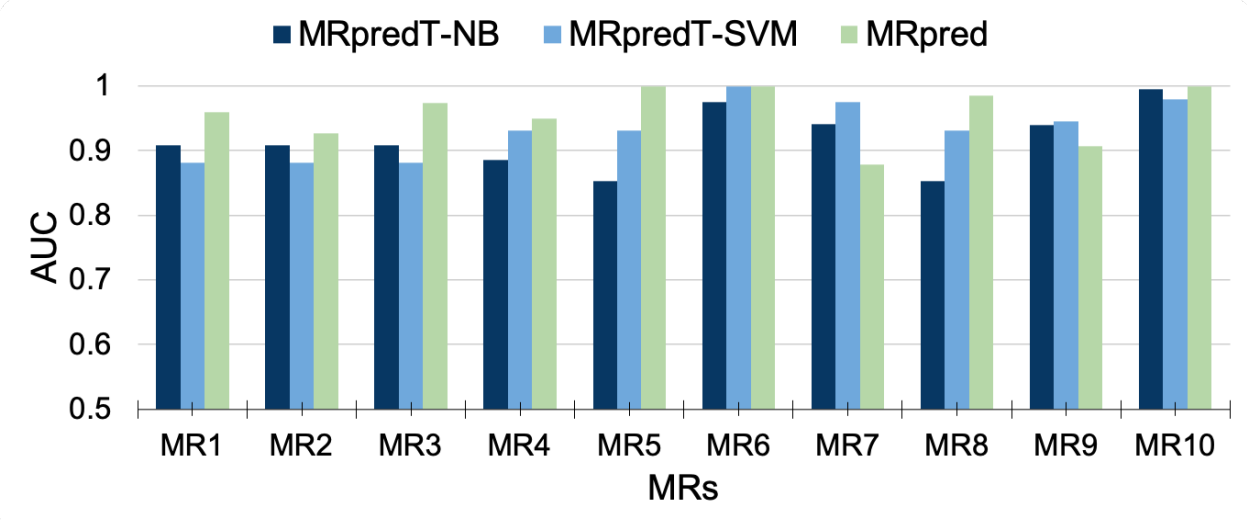

Figure 14: AUC scores of Naive Bayes and SVM models using text classification (MRpredT-NB and MRpredT-SVM), and for the SVM model using Random Walk Graph Kernel (MRpred). AUC: Area Under the Receiver Operating Characteristic Curve. MR: Metamorphic Relations.

Methodology

In this work, we model the task of predicting MRs for a given program as a supervised classification problem. In particular, we treat each MR independently and apply a separate binary classifier for each MR. Binary class labels correspond to the existence of a specific MR, and the input feature representing each program is generated solely using its documentation.

The following subsections describe the text corpus and the MRs used in this study. The overview of the text classification approach for predicting MRs of the dataset is also discussed here. Moreover, the details of the experimental setup are mentioned, as well.

Data

Text Corpus A total of 93 program's Javadocs, which handle matrix operations, are used for this study. They are collected from Apache Commons Math Library3[1], la4j (Linear Algebra for Java)[2], and JAMA (Java Matrix package)[3], which are open-source projects. These programs perform a variety of operations on matrices such as addition, multiplication, subtraction, and searching. Figure 12 shows the raw data example of the Javadocs of a program.

Metamorphic Relations For this study, we identified ten MRs manually that are generally applicable to matrix calculations. These MRs are shown in Table 5 with the change made to the input and their expected output change. These MRs are used as class labels for the supervised classification model.

[1] http://commons.apache.org/proper/commons-math/javadocs/api-3.6/
[2] http://la4j.org/apidocs/
[3] https://math.nist.gov/javanumerics/jama/doc/

Models

Javadoc contains the information of a program's operation, inputs (parameters) and outputs (returns) that are directly related to the MRs satisfied by a given program. Our method solely uses Javadoc information to predict MRs for Java programs. Figure 13 shows an overview of our method. We implemented our models using the scikit-learn[4] python library.

The first step of this method is to extract the Javadoc documentation from the source code using Java Parser[5] and pre-processes them using the lemmatization [67] technique. Then, text feature extraction methods are applied to those pre-processed Javadocs to obtain the feature vectors. Bag of words (BoW) model [67], is used as the feature representation. To make the text learnable by machines, they must be converted to numerical vectors, and the BoW model is a standard technique to obtain such a structure [67]. In this representation, the features (f_i, see Figure 13) are the tokens extracted from the source and feature-values are the frequency of their occurrence within each program documentation. These feature vectors of the programs are then supplied into the machine learning classification algorithm with their associated MR labels. Here, MR labels are identified manually for all the programs, where the label is '1' if an MR is satisfied by the program, and '0' otherwise.

We used two popular machine learning algorithms, Naive Bayes [68] and Support Vector Machines (SVMs) [69], as the underlying classification algorithms. In many other domains, these two algorithms are historically found to be very effective for text classification tasks [70–73]. We implement both using the default parameters available in scikit-learn. The *trained* predictive model is then used to predict the labels (i.e. MR) for the unseen program by supplying the corresponding feature vector generated solely from its Javadoc documentation text, as shown in Figure 13.

[4]https://scikit-learn.org/stable/

[5]`https://javaparser.org/`

Experimental Setup

In this experiment, 10-times stratified 10-fold cross-validation is used to evaluate the effectiveness of the models. It is a cross-validation technique where each fold contains roughly the same percentage of data belonging to each class compared to the full dataset [74]. Also, in the case of prediction problems, the mean response value is maintained relatively equal in all the folds [74]. This process is repeated ten times to negate any selection-bias. This approach is not only useful for the fair assessment of the models but also helps to alleviate over-fitting [74]. We compare our models to MRpred [38].

The evaluation measure used in this study is *Area Under the receiver operating characteristic Curve (AUC)*. AUC measures the probability that a randomly chosen positive example (i.e. a program labeled with a certain MR) will be ranked higher by the predictive model than a randomly chosen negative example [60]. AUC takes values ranged in [0, 1] where higher values indicate better performance, but a value of 0.5 is equivalent to a random classifier. AUC is used as the evaluation metric in this experiment as it does not depend on the discrimination threshold of the classifier and is considered a better measure for analyzing learning algorithms (compared to metrics such as accuracy) [60].

Results and Discussion

In Figure 14, the AUC scores of our metamorphic relation prediction models (MRpredT-NB: Naive Bayes, and MRpredT-SVM: SVM classification model) are compared to MRpred (i.e., classification model using random walk graph kernel with SVMs). For 4/10 MRs (i.e., MR1, MR2, MR3, and MR10), MRpredT-NB outperforms its SVM counterpart (i.e. MRpredT-SVM).

For 6/10 MRs, MRpredT-SVM model performs better than MRpredT-NB. Among the above 6 MRs, the highest possible AUC score (1.0) could be observed when predicting the

addition with Identity matrix (MR6) MR. The other MRs also reported AUC values higher than 0.87, indicating that our text-based approach generated effective predictive models when using SVM for all the MRs. However, the prediction scores achieved by the NB is also promising. The highest score is 0.99 when predicting transpose (MR10) MR.

Furthermore, for 2/10 MRs (MR7 and MR9), one of the two MRpredT models outperform MRpred providing ample evidence for the effectiveness of using text data for MR prediction. Interestingly, for MR7 and MR9, both MRpredT models perform better than MRpred, warranting further investigation. One the other hand, MRpred is still the clear winner for MR1, MR3, MR5, and MR8.

Conclusion and Future work

The metamorphic testing technique is beneficial to test programs that do not have a test oracle. The effectiveness of this technique highly depends on the set of MRs used for testing. But the identification process of MRs is mostly performed manually. This study proposes to use a text classification method to predict MRs for functions that perform matrix calculations using their documentation. The results show that for these types of programs, the text classification-based machine learning approach can be effective in predicting MRs, especially when using the SVM model. However, there are many avenues for future investigation, as described below.

First, we would like to investigate the effectiveness of heterogeneous features by combining MRpredT's text features with the program features used in MRpred. In addition, text features extracted from the source code itself could be a fruitful Addition. Another aspect worth investigating is exploring the text features (i.e., tokens) identified as the most effective for each MR by our MRpredT models. This may provide valuable information for going beyond the standard BoW model and developing domain-specif features.

With the recent popularity of deep learning, we intend to utilize recurrent neural

networks as the underlying machine learning model, which will also negate the need for hand-engineering features. This will also provide the opportunity for exploring the effectiveness of more sophisticated features based on semantic similarity (i.e., word and BERT embeddings). However, this will require obtaining much larger datasets as these models are known to be data-hungry. Considering the highly resource-consuming nature of the manual labeling process, a semi-supervised learning technique that allows the use of unlabeled data may be a viable alternative.

A MAPPING STUDY OF SECURITY VULNERABILITY DETECTION APPROACHES FOR WEB APPLICATIONS

Contribution of Authors and Co-Authors

Manuscript in Chapter 6

Author: Karishma Rahman

Contributions: Problem identification and proposing solution, running experiment, manuscript writing, creating tables and figures. Primary writer.

Co-Author: Clemente Izurieta

Contribution in manuscript editing/writing, provided feedback, guidance and advice.

Manuscript Information

Karishma Rahman and Clemente Izurieta

2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)

Status of Manuscript:
____ Prepared for submission to a peer-reviewed journal
____ Officially submitted to a peer-reviewed journal
____ Accepted by a peer-reviewed journal
_X_ Published in a peer-reviewed journal

IEEE
2022/8/31
10.1109/SEAA56994.2022.00081

Abstract

For the last few decades, the number of security vulnerabilities has been increasing with the development of web applications. The domain of Web Applications is evolving. As a result, many empirical studies have been carried out to address different security vulnerabilities. However, an analysis of existing studies is needed before developing new security vulnerability testing techniques. We perform a systematic mapping study documenting state-of-the-art empirical research in web application security vulnerability detection. The aim is to describe a roadmap for synthesizing the documented empirical research. Existing research and literature have been reviewed using a systematic mapping study. Our study reports on work dating from 2001 to 2021. The initial search retrieved 150 papers from the IEEE Xplore and ACM Digital Libraries, of which 76 were added to the study. A classification scheme is derived based on the primary studies. The study demonstrates that vulnerability detection in web applications is an ongoing field of research and that the number of publications is increasing. Our study helps illuminate research areas that need more consideration.

Introduction

The Web substantially influences all aspects of our everyday social lives nowadays. Billions worldwide use different web applications to get information, play games, communicate, execute financial transactions, and socialize. Thus, they allow people and organizations to communicate utilizing different applications regardless of the potentially substantial geographical distances. Though this technology has brought numerous advantages to our lives, they also come with various challenges. The most significant challenge is the security of web applications [75]. Security in web applications refers to threats because of the unstructured designs of the software, inadequate testing, and poor coding.

Vulnerabilities are manifestations of weaknesses in a system. They can occur accidentally because of the carelessness of the system designer and can cause failures in the security of the system [76]. Over the past few years, vulnerabilities in applications have been continuously increasing, and most of the common vulnerabilities found include SQL injection, cross-site scripting, broken authentication, command-line injection, and identification and authentication failures [76]. Many researchers nowadays investigate these vulnerabilities and develop different automated techniques and tools to overcome these vulnerabilities [77]. Researchers have suggested numerous methods for detecting security vulnerabilities of web applications for the past decade, and paper numbers in this area are increasing. Systematically identifying, interpreting, and classifying the publications is essential to present a summary of the specific domain's trends. Therefore, a systematic mapping study is needed.

Petersen et al. [78] stated that a systematic mapping study is used to examine, categorize and structure articles of particular research areas in software engineering. The objective of the mapping study is to acquire knowledge of a research area through classification. We follow the recommended five steps, which include defining research questions, searching for suitable papers, stating selection criteria, extracting data, and mapping. The main contributions of this paper include *i)* a classification scheme for categorizing the articles, and *ii)* a systematic mapping study that consists of related research over the past 20 years (2001-2021) by analyzing 76 articles.

Background and Related work

Verification techniques in current web development practices are either incomplete or erroneous, which introduces vulnerabilities to web applications. In turn, vulnerabilities allow a malicious user to introduce harmful artifacts (e.g., via script injections, data flow attacks, and input validation attacks) into web content [76]. These harmful artifacts include cross-site

scripting, directory traversal, SQL injection, response splitting, and filename inclusion.

Prior studies have attempted to synthesize web application vulnerability detection. Specifically, Alalfi et al.[79] present a survey that uses 24 different modeling techniques in web verification, validation, and testing. The survey classifies, contrasts, and examines the modeling techniques. Marin et al. [80] provide a brief overview of current testing techniques for web applications. They discuss the limitations of these techniques for testing web applications. Garousi et al. [81] developed a method for classifying papers in the testing of web applications. Their paper is the first systematic mapping study in web application testing. A systematic mapping study of functional testing is conducted, which analyses 79 papers. Rafique et al. [75] synthesize empirical studies in web application vulnerability detection approaches. Their findings correspond with the software development steps, and the vulnerabilities correspond to OWASP's Top 10 security vulnerabilities. Li et al. [82] include static, dynamic, and hybrid analyses in their study. Deepa et al. [83] concentrate on detecting and preventing attacks targeting injection and logic vulnerabilities. Chang et al. [84] describe two web-based malware detection methods, i.e., virtual machine-based and signature-based detection. A comprehensive survey by Gupta et al. [85] describes emerging web application weaknesses, avoidance mechanisms, detection, and attack patterns for all critical web threats in OWASP 2013. A survey by Seng et al. [86] describes web application security scanners and their qualities. Finally, Atashzar et al. [20] survey the web application security features, where features include critical vulnerabilities, hacking tools, and approaches at a high level.

The studies mentioned above have various weaknesses which restrict replication, generalization, and usability. Some studies are conducted without any systematic approach for reviewing the papers. Further, the selection criteria of some studies are not explicitly described, making it impossible to reproduce results. In our mapping study, we mitigate these shortcomings.

Method

This study is conducted following the guidelines for systematic mapping suggested by Petersen et al. [78].

Goal and Research Questions

The study identifies, examines, and synthesizes the research articles published in the last twenty years in web application vulnerability detection. This mapping study addresses the following research questions:

RQ 1– How many papers introduce methods/techniques, tools, models, frameworks, comparison analysis, or processes? The first question identifies the type of contribution made [87].

RQ 2– What are the research methods used in the papers? Petersen et al. propose the following research methods in their systematic mapping guideline- (1) solution proposal, (2) experience papers, (3) evaluation research, (4) validation research, and (5) opinion papers.

RQ 3- What are the testing techniques presented in the papers? The testing techniques include generating test cases, using scanners, injecting faults, etc.

RQ 4– How many approaches are manual versus automated that detect vulnerabilities of web applications?

RQ 5– How many approaches are evaluated on dynamic versus static web applications?

RQ 6– How many papers propose a working detection tool? What are the names, and how many are freely available for use?

RQ 7- What are the common security vulnerabilities in web applications found on those papers?

RQ 8– What is the annual number of publications or the publication rate in this field?

RQ 9– What are the citation rates of the papers in this area?

((web ∨ web application ∨ website) ∧
(vulnerability ∨ vulnerabilities ∨ security ∨ threats) ∧
(testing ∨ assessment ∨ scanning ∨ analyzing ∨ verification ∨ validation))

Figure 15: Formulated search query for the selection of the relevant articles

Table 6: Classification scheme

Attributes	Research question
Contribution type of the paper	RQ 1
Research type of the paper	RQ 2
Type of testing activity/technique	RQ 3
Manual versus automated approach	RQ 4
Static web application versus dynamic web application	RQ 5
Presented tools in the papers	RQ 6
Vulnerability type addressed	RQ 7
Publication year	RQ 8
Number of citations	RQ 9

Paper selection strategy

We mined IEEE Xplore[1] and the ACM Digital Library[2]. Papers published between 2001 and 2021 are included in the pool of papers. Search keywords have been identified using the PICO (Population, Intervention, Comparison, and Outcomes) technique, which is suggested by Kitchenham and Charters [88] to formulate search strings from research questions. The identified keywords are web application, vulnerability, and detecting/testing, which are grouped into sets. We formulate the search string along with their synonyms as shown in Figure 15.

[1]http://ieeexplore.ieee.org
[2]http://dl.acm.org

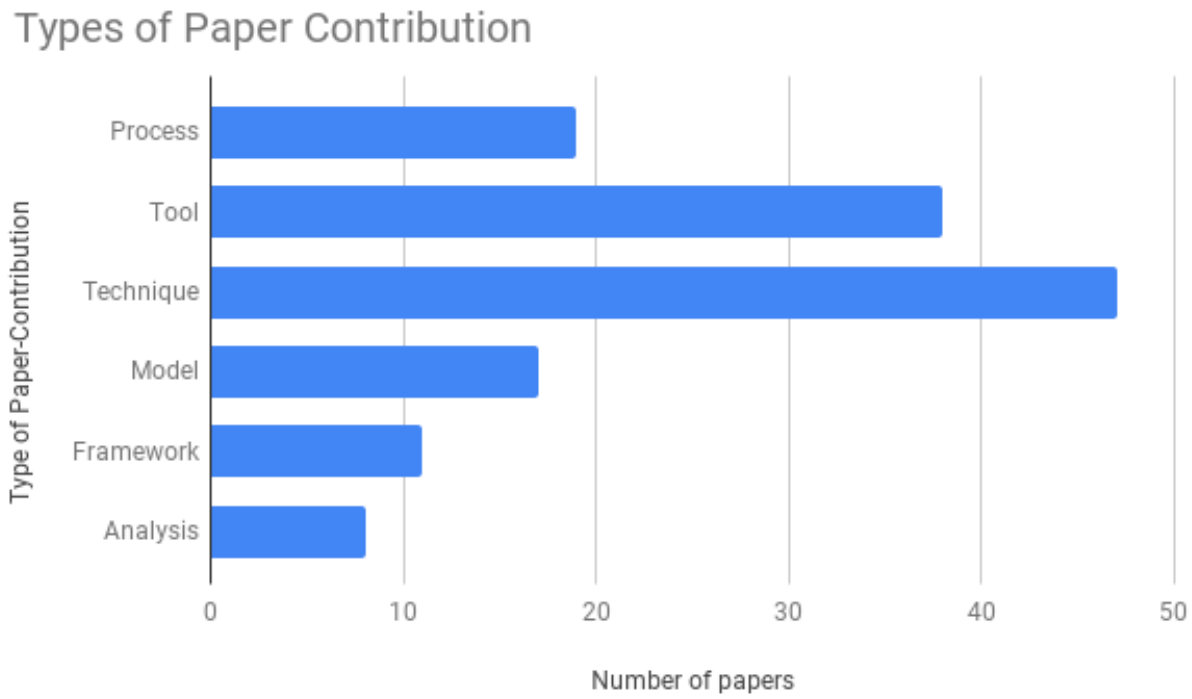

Figure 16: Types of contribution

Exclusion and inclusion criteria

Articles are selected based on the titles and abstracts, keywords, and reading of the evaluation section as suggested in [87]. Both authors reviewed each article to increase reliability. Full-text reading of the paper is taken into account only when in doubt. The inclusion criteria applied to the collection of titles and abstracts required that *i)* research articles were based on empirical evidence related to vulnerability detection methods of web applications, *ii)* that if multiple studies were reported by the same author with the same result, only the latest study was considered, and *iii)* that studies were published from 2001 to 2021. Summaries, editorials, non-peered reviewed studies, studies in other languages, and books and magazines were excluded.

After applying the selection criteria to 150 papers, the collection size decreased to 76. The list of 76 papers can be found in the online repository [89].

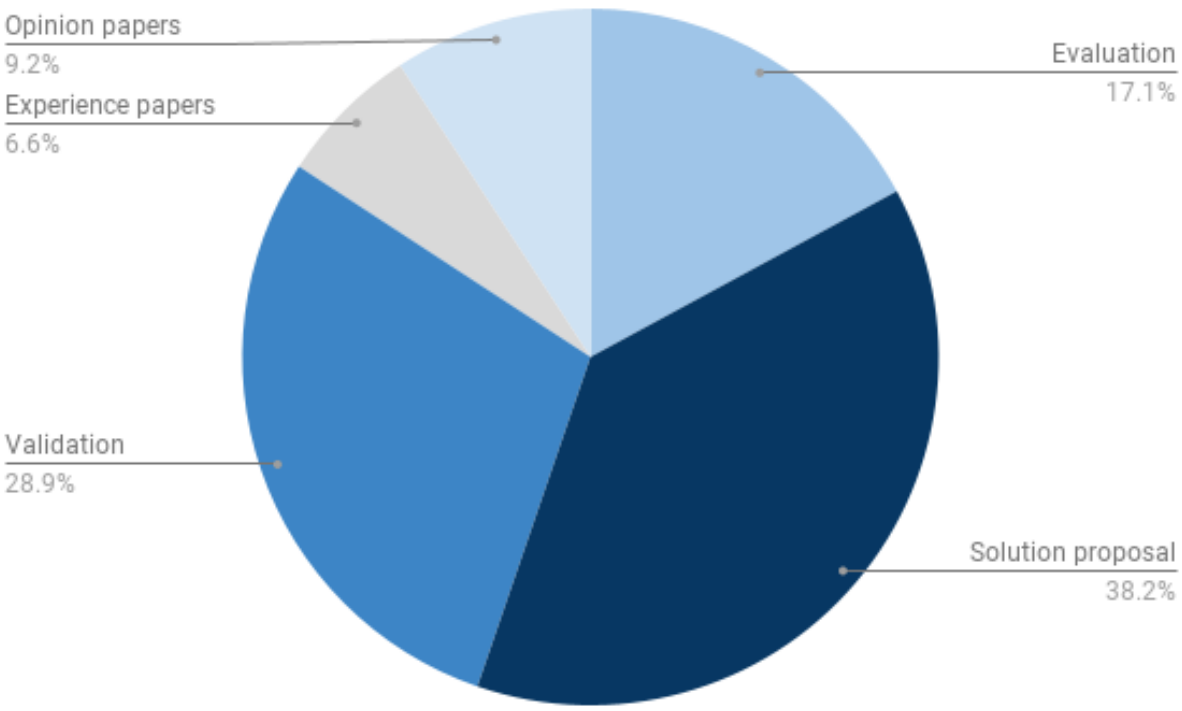

Figure 17: Types of research paper

Classification Scheme and Data Extraction

A classification scheme is also known as a systematic map [78], and Table 6 shows how each attribute maps to a research question. The classification scheme is created iteratively while collecting the data. After developing the classification scheme, the papers are then classified using the scheme. The online repository records the publications numbers in each classification [89].

Results of the Mapping

Herein, we address each research question. RQ 1– Figure 16 shows the distribution of the papers by the type of contributions for the 76 papers in the study. Some papers are classified under more than one type based on their contributions. For example, paper number

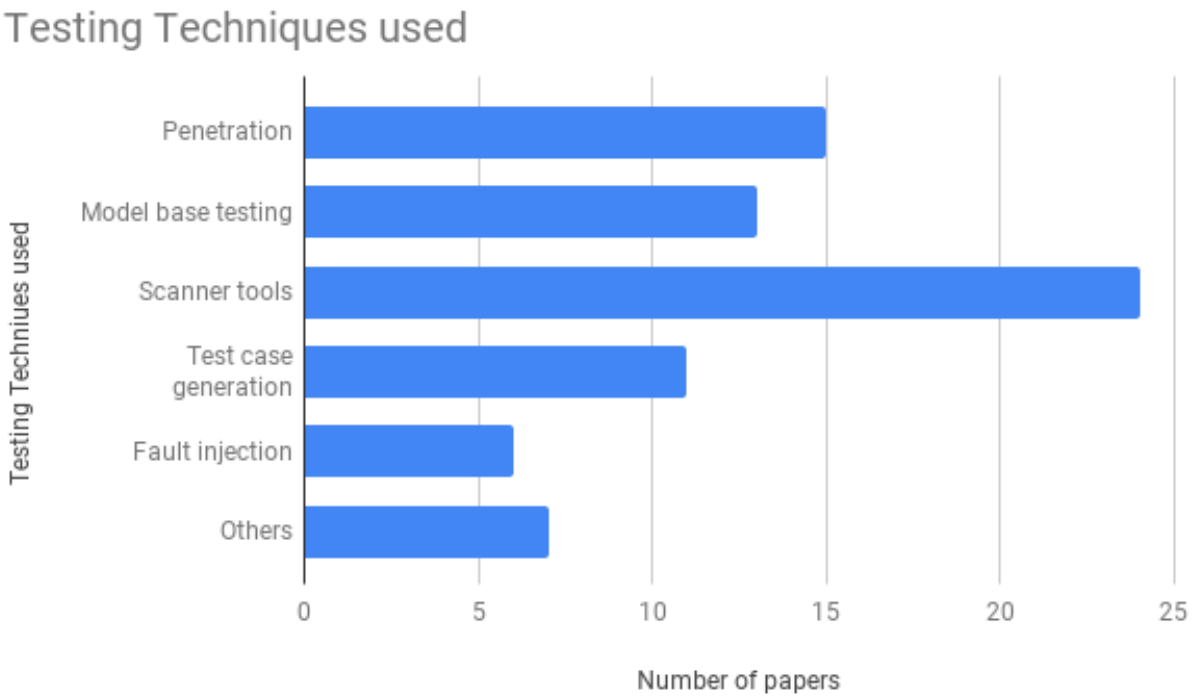

Figure 18: Types of testing techniques

10 [89] made two contributions: (1) a test method (Metamorphic testing based approach), and (2) a test tool called SLMR.

RQ 2-Figure 17 illustrates papers by research facet. The research in web application vulnerability detection is dominated by solution proposals (29 papers: 38.2%) and validation studies (22 papers: 28.9%).

RQ 3- Figure 18 displays the distribution of various testing techniques used in the papers. The ratio of this category is comparatively spread out among the types of techniques.

RQ 4- Testing automation is a known research concern [79]. Forty-one papers describe full automation, while eleven papers are fully manual. The remaining papers use both.

RQ 5- Sixty-seven papers analyze dynamic web applications, and nine analyze static websites. The testing of dynamic web applications is more widespread.

RQ 6- Eighteen papers describe tools. Some tools include SMLR (paper no. 10), Escrow

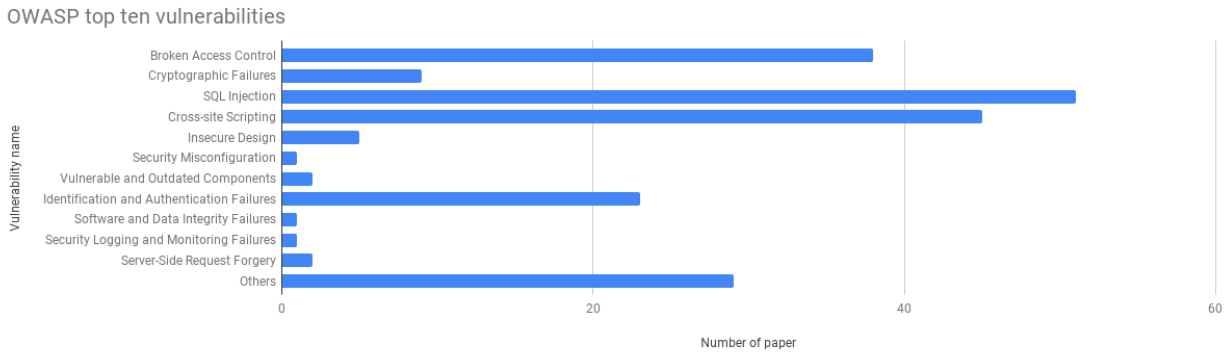

Figure 19: Detection of security vulnerabilities from OWASP top 10

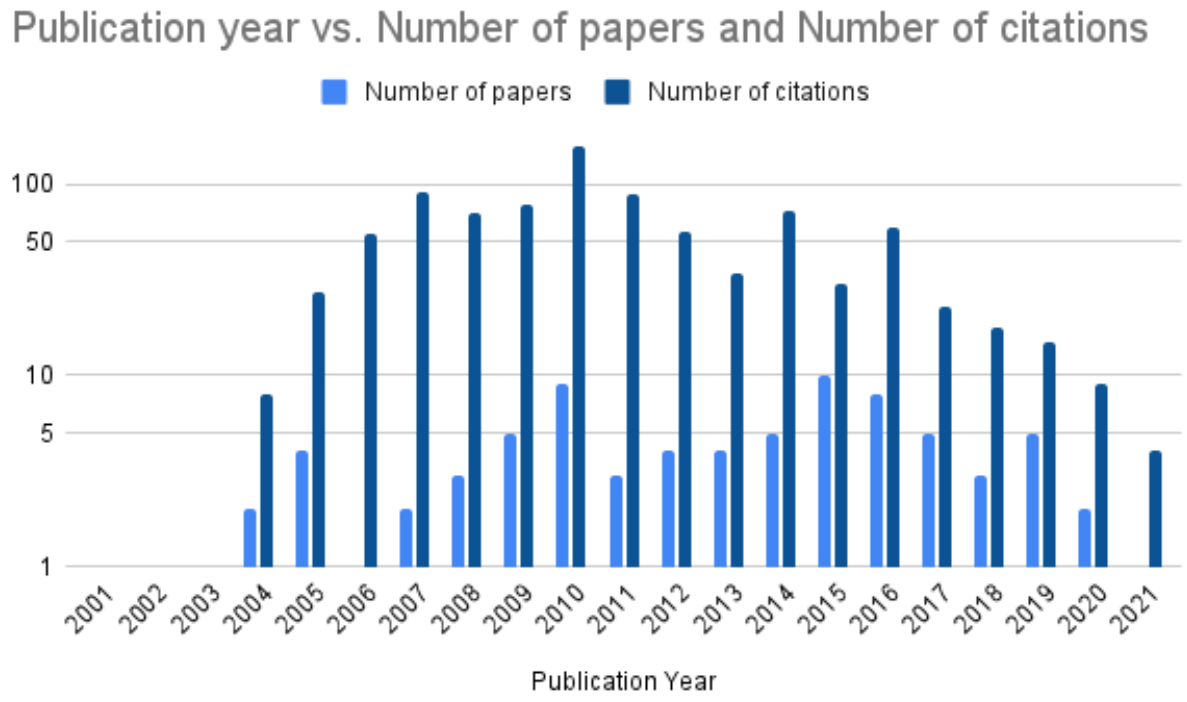

Figure 20: Publication trend per year & citation count vs. publication year.

(paper no. 24), DEKANT (paper no. 60), and MoScan (paper no.74) [89].

RQ 7- We categorized vulnerabilities based on the OWASP Top 10[3] shown in Figure 19. Our results suggest that SQL Injection vulnerability and Cross-site scripting are the most common, with 51 and 45 counts respectively. Besides this, Broken Access Control can be found in 38 papers, and Identification and Authentication Failures can be found in 23 papers. Other vulnerabilities from the OWASP Top 10 can be found in some papers. Others vulnerabilities that are not included in the top 10 list can be found in 29 papers.

RQ 8- Figure 20 shows the publication trend of the studies. We observe that the number of papers is higher in 2010, 2015, and 2016. However, in the years between, paper counts were somewhat lower. A decreasing trend in publications is observed science 2016.

RQ 9- Citation data is extracted from Google Scholar (August 2021). Figure 20 visualizes the counts. We observe that the papers from 2006 to 2016 have more citations than the earlier and later papers. This trend is very common as the papers from 2020-2021 are comparatively newer. This result also helps to reveal the top-cited papers. Paper 46 [89] is the top cited paper with 111 citations.

Discussion, Conclusion and Future Work

Most papers propose new vulnerability detection techniques or improve existing vulnerability detection techniques implemented for web applications. Our study indicates that the most common vulnerabilities in web applications can be found on the OWASP top ten vulnerability list, consistent with expectations. There is an enormous scope for future research in this area, which suggests that the number of papers in this domain will likely increase. Also, although many papers suggest different techniques, only a few tools can be downloaded. This is a scary reality that the research community must address if they want

[3]https://owasp.org/www-project-top-ten/

to influence industry practitioners.

Some testing techniques depend on an implicit test oracle. An implicit oracle depends on implicit knowledge, differentiating between the right and wrong behavior of the system [90]. Despite there being many proposed security testing approaches, the oracle problem is still not appropriately addressed. This opens up various avenues for future studies.

Potential threats to the validity of this study were studied according to Wohlin et al. [91]. The search string poses an internal validity threat; however, this was mitigated by using a known construction technique. Other threats include selection criteria and author judgments which were mitigated by agreements. Conclusions are directly traceable to the data sets associated with each research question.

In conclusion, the web applications vulnerability detection domain has a long history of development and research. This paper delivers a systematic mapping study that shows current trends. Also, it presents potential gaps and suggestions for prospective studies to help bridge this gap. The study focuses on the last 20 years. It analyzes 76 relevant selections while providing a classification scheme by examining the primary studies.

AN APPROACH TO TESTING BANKING SOFTWARE USING METAMORPHIC RELATIONS

Contribution of Authors and Co-Authors

Manuscript in Chapter 7

Author: Karishma Rahman

Contributions: Problem identification and proposing solution, running experiment, manuscript writing, creating tables and figures. Primary writer.

Co-Author: Clemente Izurieta

Contributions: Contribution in manuscript editing/writing, provided feedback, guidance and advice.

Manuscript Information

Karishma Rahman and Clemente Izurieta

2023 IEEE 24th International Conference on Information Reuse and Integration for Data Science

Status of Manuscript:
____ Prepared for submission to a peer-reviewed journal
____ Officially submitted to a peer-reviewed journal
____ Accepted by a peer-reviewed journal
_X_ Published in a peer-reviewed journal

IEEE
2023/8/4
10.1109/IRI58017.2023.00036

Abstract

Software systems used for banking are crucial for daily operations and are considered to be part of critical infrastructure; however, testing the functions of these highly reusable systems can be difficult due to the project's complexity and the absence of a reliable oracle. In software testing, the Oracle problem directs to the difficulty of deciding whether the software's observed behavior is correct. To address this issue, we suggest utilizing metamorphic testing (MT), which tests the banking system's functionalities based on their properties. Metamorphic testing is a software testing technique where multiple inputs are generated for a program, then those inputs are transformed based on a pre-defined set of rules. The resulting outputs are then compared to the original outputs to verify that the program works correctly. Metamorphic relations (MRs) are a fundamental concept in metamorphic testing. They define the relationships between the input and output of a system under test and specify how they should change in response to input transformations. Through a case study, we introduce new metamorphic relations to test banking functions and demonstrate the effectiveness of using these MRs. The study results indicate that this is a feasible and efficient approach using an alternative to a test oracle when testing complex E-type (i.e., real-world) software.

Introduction

Advances in science and technology have led to significant evolution in the software industry in recent years [7]. With the increasing demand for E-type software in modern society, most applications involve complicated scientific calculations and data processing, necessitating software engineers to ensure that they meet all the requirements for reliability [92]. E-type systems refer to real-world systems [92]. One of Lehman's [92] Laws states that E-type systems are constantly evolving, and their complexity is continuously increasing

unless something is purposely done to minimize it. Banking systems are highly reusable E-type systems and their functionality directly affects the growth of the commercial banking industry. Therefore, developing and testing banking software are crucial phases in the software lifecycle for the growth of the industry, and efficient validation and verification techniques are necessary for maintaining the reusability of these banking applications. Consumer demands for banking software have risen, leading to more complex projects and further research into various aspects of banking software [7]. Software testing is a continuous process throughout the banking system project lifecycle and is essential for its progress. The testing of banking systems requires high complexity, security, and accuracy, and customers expect tools for easy transactions and access to financial organizations' services [93]. Banking software has complex designs and multi-layered workflows and offers various features and functions, handling sensitive data like customers' financial and personal information [94]. Therefore, software testing for banking applications must be precise, as any lack of test coverage can lead to data breaches, loss of funds, banking fraud, and other criminal activities [93].

Testing is a crucial aspect of the development process of banking software to ensure that the system behaves correctly. Testing bears more than 50% of the total software development costs, as it is an expensive, time-consuming, and complex activity [95]. The process of testing is often prone to human error. Creating dependable software systems remains an ongoing challenge, and researchers and practitioners continuously explore more efficient methods to test software [7]. Test cases are performed on the system under test during the testing process. A test oracle, either automated or manual, is then used to determine whether it acted as anticipated [7]. In either case, the actual output is compared with the expected outcome.

The challenge of Test Oracle arises when testing complex software. It occurs when it is difficult to determine whether the program outputs on test cases are correct [96]. Banking

software often has intricate functionality, which can exacerbate the Oracle problem in their system. For instance, with an electronic payment service, there may be situations where the consumer needs clarification on how much should be charged for a given input. The challenge becomes even more pronounced when the payment concerns transfer charges among different bank accounts or currency exchange applications.

Metamorphic Testing (MT) is a technique that has proven helpful in certain circumstances to address the challenge of the oracle problem [18]. The principle behind Metamorphic Testing (MT) [8] is that it might be easier to analyze the relations between the results of multiple test executions, which are referred to as metamorphic relations (MRs), rather than specifying the input-output behavior of a system [8]. MT employs MRs to determine system properties, which automatically alter the initial test input into follow-up test input [8]. If the system fails to meet the MRs when tested with the initial and follow-up input, it is inferred that it is defective [97].

A significant amount of study has focused on creating Metamorphic Testing (MT) methods for specific areas such as computer graphics, web services, and embedded systems [98]. Our research aims to apply MT to tackle the test oracle problem in banking software. We aim to systematically define metamorphic relations that capture banking function properties (i.e., characteristics that are compromised when the system is at risk) and automate testing using these metamorphic relations.

We examine how MT applies to banking software testing and convey a case study. This paper delivers the following contributions:

- An approach to investigate and uncover the essential points when utilizing MT to test banking software.
- A list of new MRs for banking functions.
- To show the applicability of the proposed MRs, we conduct a case study on bank

software functionalities. The study indicates the relevance of using MT to test banking functions. We also employed mutation analysis to assess the efficiency of the MT approach.

The rest of the paper is organized as follows. Section 7 introduces underlying concepts related to MT and mutation analysis. Section 7 presents a framework of MT for banking software and reports on a case study where MT tests major banking functions. Section 7 discusses the results of the case study. The next Section 7 discusses the threats to validity of the experiment. Section 7 discusses the related works done in this area. Lastly, Section 7 concludes the paper by pointing out potential future work.

Background

This section introduces relevant literature and concepts related to Banking application testing, MT, and mutation analysis.

Testing Bank Application

The banking industry has changed remarkably due to rapidly growing and innovative technology. Due to the complicated features integrated into banking software, it is regarded as one of the most sophisticated and complex enterprise solutions [93]. The daily transactions carried out through the banking system require accurate data, high scalability, and reliability. Therefore, testing this software under various conditions ensures its efficiency. Moreover, the banking sector requires robust reporting mechanisms to record and instantly monitor transactions and user interactions [94]. Testing is essential to ensure that banking software functions well and effectively. Functional testing of banking software is distinct from standard software testing as these applications handle customers' financial data and money, making it necessary to conduct thorough testing [93]. No critical business scenario should be overlooked during testing.

Metamorphic Testing

Metamorphic testing is a technique that can help solve the well-known test oracle problem. It is developed by Chen et al. [8] to check if a program satisfies a set of previously defined properties known as Metamorphic Relations. It determines how input changes should affect a program's output. If the program fails to meet these expected relations, it could indicate the presence of faults. To use metamorphic testing, a suitable set of MRs should be identified, and a set of initial test cases created. Input changes defined by the MRs are then applied to develop follow-up test cases. The initial and follow-up test cases are then executed, and a fault may exist if the output does not behave according to the predicted MR. Metamorphic testing helps identify defects in programs without test oracles because it examines the input and output relationship between multiple program executions, even when the correct result of each execution is unknown [98]. For example, the SINE function $y = sin(x)$ can be tested using MT by using the property that adding 2π to the input angle does not change the output (i.e., $sin(x) = sin(x + 2\pi)$). If this property is violated, it indicates a failure in the function's implementation.

Mutation Testing

Mutation testing is a typically used practice for evaluating the efficacy of testing strategies and adequacy of test suites [25]. This approach involves applying mutation operators to the tested program, which introduces various faults and generates a set of mutant variants. A test case is considered to "kill" a mutant if it causes the mutant to exhibit behavior different from the original program [25]. The number of killed mutants is used to calculate the mutation score (MS), which measures the thoroughness of a test suite in killing mutants [25]. The MS is calculated using the following formula:

$$MS = \frac{M_k}{M_t - M_e}$$

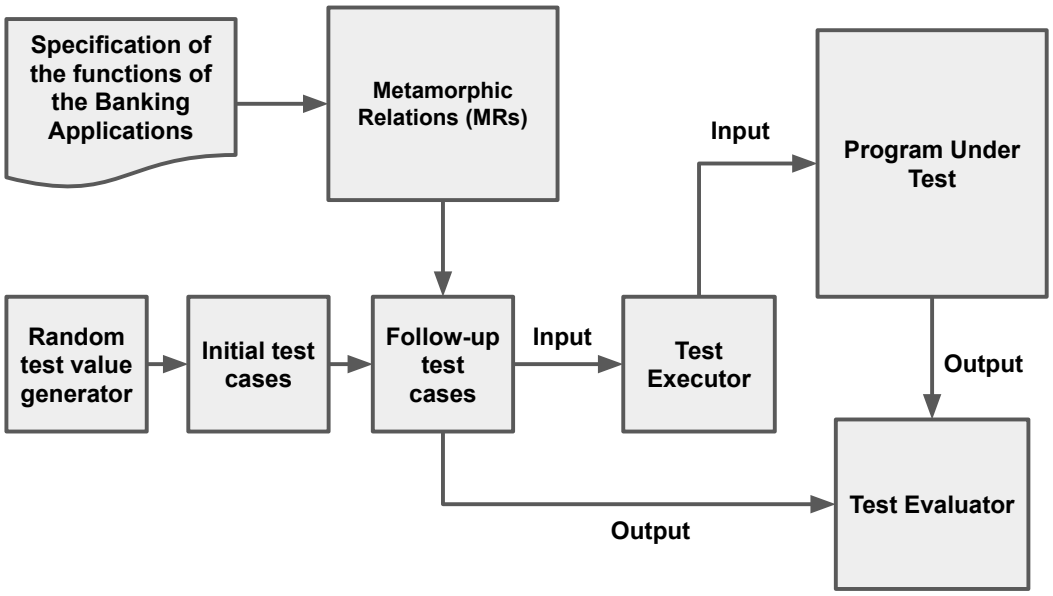

Figure 21: A diagram that portraits the framework of MT to test a system under test (i.e. Banking Software). This framework uses MRs to generate follow-up test cases. MT process is used as the test evaluators to check the output. Here, the arrows represent the flows of information.

where the number of killed mutants is denoted as M_k, the total number of mutants is denoted as M_t, and the number of equivalent mutants is denoted as M_e (i.e., mutants that always behave the same way). Automatically generated mutants are thought to be more similar to real-life faults than manually seeded ones. Therefore, the mutation score effectively indicates the testing technique's effectiveness. In this study, the mutation analysis technique is used to assess the efficacy of our testing method.

Table 7: Metamorphic Relations (MRs) for Banking functions and their associated descriptions

Metamorphic Relations (MRs)	Description

Deposit (Input is the amount to deposit, and the output is the total balance in the account.)	
MR1- Addition	This MR says that for the follow-up input (I_f), if we add a credit (C, where $C > 0$) to the initial input (I_i), i.e., $I_f = I_i + C$, the follow-up output (O_f) will increase accordingly from the initial output (O_i), i.e., $O_f > O_i$.
MR2- Subtraction	This MR says that for the follow-up input (I_f), if we subtract a credit (C, where $0 <= C <= I_i$) from the initial input (I_i), i.e., $I_f = I_i - C$, the follow-up output (O_f) will increase or remain the same accordingly from the initial output (O_i), i.e., $O_f > O_i$ or $O_f = O_i$.
MR3- Multiplication	This MR says that for the follow-up input (I_f), if we multiply a credit (C, where $C > 0$) with the initial input (I_i), i.e., $I_f = I_i * C$, the follow-up output (O_f) will increase or remain same accordingly from the initial output (O_i), i.e., $O_f > O_i$ or $O_f = O_i$.
MR4- Division	This MR says that for the follow-up input (I_f), if we divide a credit (C, where $C > 0$) by the initial input (I_i), i.e., $I_f = I_i/C$, the follow-up output (O_f) will increase or remain same accordingly from the initial output (O_i), i.e., $O_f > O_i$ or $O_f = O_i$.
MR5- Negative	This MR says that for the follow-up input (I_f), if we convert the credit to a negative value (C, where $C < 0$) from the initial input (I_i), i.e., $I_f = -(I_i)$, the follow-up output (O_f) will remain the same as the initial output and an error message will be displayed (O_i), i.e., $O_f = O_i$.

Metamorphic Testing for Banking applications

Approach

In developing software systems for business purposes, the end-users must be confident that the application is performing as expected. However, testing every possible usage scenario can be challenging for developers. As a result, it is necessary to employ a testing technique that allows for the revision of executed tests. MT offers an appropriate solution for testing applications without requiring oracles.

Figure 21 illustrates the MT framework for testing banking application functions. Metamorphic relationships (MRs) are a critical component of the framework since they determine the generation of test cases and the evaluation of results. When used to test banking applications, we first extract metamorphic property specifications from the function's description and identify the MRs. We then use the identified MRs to generate the follow-up test cases from the initial test cases. A test executor executes the follow-up test cases on the system under test. The outputs produced by the program being tested is compared with the follow-up test cases using a test evaluator to establish whether the MR has been satisfied or violated. If any MR is violated, we can say it has detected a fault in the system.

This section outlines the approach employed that can validate the MT framework's effectiveness in testing banking functions. The study focuses on key banking functionalities and identifies corresponding MRs. The effectiveness of MT is assessed through mutation analysis. The findings indicate that MT is capable of detecting approximately 75% of mutants (i.e., faulty programs), thereby demonstrating its efficacy as a testing technique.

Subject Program

This study focuses on testing the functionality of banking software by examining three commonly used features: Deposit, Withdrawal, and Transfer. The Deposit function is relatively simple and involves adding money to an existing account. On the other hand, the Withdrawal function is more complex and consists in withdrawing cash from an existing account. It only performs the withdraw function if the account has enough balance. Lastly, the Transfer function transfers money from the current account to other accounts with a commission fee associated with the transfer type. These functions are generated using the guidelines for general banking software [99] and they are all implemented in Java.

Table 8: Metamorphic Relations (MRs) for Banking functions and their associated descriptions

Metamorphic Relations (MRs)	Description
Withdrawal & Transfer (Input is the amount to withdraw or transfer, and the output is the total balance in the account.)	
MR1- Addition	This MR says that for the follow-up input (I_f), if we add a credit (C, where $0 < C < (total\_balance - I_i)$) to the initial input ($I_i$), i.e., $I_f = I_i + C$, the follow-up output (O_f) will decrease accordingly from the initial output (O_i), i.e., $O_f < O_i$.
MR2- Subtraction	This MR says that for the follow-up input (I_f), if we subtract a credit (C, where $0 <= C <= (total\_balance - I_i)$) from the initial input (I_i), i.e., $I_f = I_i - C$, the follow-up output (O_f) will decrease or remain same with error message accordingly from the initial output (O_i), i.e., $O_f < O_i$ or $O_f = O_i$.

MR3- Multiplication	This MR says that for the follow-up input (I_f), if we multiply a credit (C, where $C > 0$) with the initial input (I_i), i.e., $I_f = I_i * C$, the follow-up output (O_f) will decrease or remain same with an error message accordingly from the initial output (O_i), i.e., $O_f < O_i$ or $O_f = O_i$.
MR4- Division	This MR says that for the follow-up input (I_f), if we divide a credit (C, where $C > 0$) by the initial input (I_i), i.e., $I_f = I_i/C$, the follow-up output (O_f) will decrease or remain same accordingly from the initial output (O_i), i.e., $O_f < O_i$ or $O_f = O_i$.
MR5- Negative	This MR says that for the follow-up input (I_f), if we convert the credit to a negative value (C, where $C < 0$) to the initial input (I_i), i.e., $I_f = -(I_i)$, the follow-up output (O_f) will remain same to the initial output and an error message will be displayed (O_i), i.e., $O_f = O_i$.
MR6- Add insufficient fund	This MR says that for the follow-up input (I_f), if we add a balance greater than the total balance ($total\_balance + C$, where $C >= 0$) to the initial input (I_i), i.e., $I_f = I_i + total\_balance + C$, the follow-up output ($O_f$) will remain same to the initial output and an error message will be displayed (O_i), i.e., $O_f = O_i$.

Metamorphic Relations

Choosing appropriate MRs is crucial when testing an application using MT. There are some guidelines that are available for determining MRs based on the program's specifications. Chen et al. [8] suggested that the MRs affecting the significant functionalities' performance are more effective. They also recommended using MRs that can produce diverse program executions. Based on these guidelines, we identified a set of MRs for each function, which are listed in Table 7 and Table 8. Each function has some MRs derived from the function's

Table 9: Active mutators used for mutation analysis

Active Mutators
BOOLEAN_FALSE_RETURN
BOOLEAN_TRUE_RETURN
CONDITIONALS_BOUNDARY_MUTATOR
EMPTY_RETURN_VALUES
INCREMENTS_MUTATOR
INVERT_NEGS_MUTATOR
MATH_MUTATOR
NEGATE_CONDITIONALS_MUTATOR
NULL_RETURN_VALUES
PRIMITIVE_RETURN_VALS_MUTATOR
VOID_METHOD_CALL_MUTATOR

specification. Tables contain the MR names and the relationship between initial and follow-up test inputs. Here, the initial input is (I_i) and output is (O_i). The follow-up input is (I_f) and output is (O_f). In Table 7, MR1 Addition tests the Deposit function. This MR says that for the follow-up input, if we add a credit C, where the credit is greater than 0 to the initial input, i.e., $I_f = I_i + C$, the follow-up output will increase accordingly from the initial output, i.e., $O_f > O_i$.

Test Case Generation

To carry out MT, test cases are created based on the MRs. Several methods can be used to generate the initial test cases, such as generating specific test values, random test values, or iterative test values. Among these methods, random test value generation is preferred for MT as it is cost-efficient and unbiased [100]. Hence, this study used random test value generation to produce the source test cases. Follow-up test cases are then developed using the MRs described in Table 7.

Table 10: Results of mutation analysis based on mutation score and test strength

MRs	Line Coverage (%)	Mutation Coverage (Killed mutant/Used mutant)	Mutation Score (MS) (%)	Test Strength (Killed mutant/Used mutant)	Test Strength (%)
Deposit					
MR1	93%	3/6	50%	3/6	50%
MR2	93%	3/6	50%	3/6	50%
MR3	93%	3/6	50%	3/6	50%
MR4	93%	3/6	50%	3/6	50%
MR5	64%	2/6	33%	2/5	40%
All	100%	4/6	67%	4/6	67%
Withdrawal					
MR1	82%	4/9	44%	4/9	44%
MR2	82%	4/9	44%	4/9	44%
MR3	82%	4/9	44%	4/9	44%
MR4	82%	4/9	44%	4/9	44%
MR5	53%	2/9	22%	2/5	40%
MR6	71%	3/9	33%	3/8	38%
All	100%	6/9	67%	6/9	67%
Transfer					
MR1	82%	4/9	44%	4/9	50%
MR2	82%	4/9	44%	4/9	50%
MR3	82%	4/9	44%	4/9	50%
MR4	82%	4/9	44%	4/9	50%

Table 10 – continued from previous page

MR5	53%	2/9	22%	2/5	40%
MR6	71%	3/9	33%	3/8	38%
All	100%	6/9	67%	6/9	67%
Total	100%	12/16	75%	12/16	75%

Evaluation

In order to assess the efficacy of MT, we utilize mutation analysis. We create Junit test cases for the functions based on the MRs. Then, we use mutation operators to introduce faults into the implementation of the functions automatically using the PIT mutation testing tool [101]. This resulted in 16 mutants. Usually, equivalent mutants are excluded from experiments. However, this mutation testing tool does not create equivalent mutants. Test suites are generated using the MRs outlined in Table 7, which is then used to test the subject programs. We use the mutation score (MS) metric to measure the effectiveness of MT. When a mutant is killed, it is considered a detected fault.

Results

In this section, we present the results of our experiment, where we utilized mutation analysis to assess the efficacy of our MT framework. We used the PIT mutation tool to generate the mutators listed in Table 9. In Table 10, we summarize the test efficiency of MT mainly by measuring its Mutation Score (MS) and Test Strength. The following describes the information listed in Table 10.

- Line coverage shows the percentage of lines covered by the tests.
- Mutation coverage shows how many mutants are killed from the total mutants.

- The MS measures the percentage of mutants killed out of all mutants (i.e., excluding the equivalent mutants) created without test coverage [101].

- Test Strength measures the ratio of mutants killed out of all mutants with test coverage [101]. The Test Strength metric does not include mutants that survive due to a lack of coverage [101]. This metric is considered a better metric than MS when validating builds. It is also shown as a percentage.

For each function, the overall performance of MT is shown. Moreover, the last row of the table presents the overall performance of MT when we consider the testing results of all functions and all MRs together.

The effectiveness of each MR can be compared using their MS and test strength for each function. Among all the MRs, for the deposit function, MR1, MR2, MR3, and MR4 are more effective than MR5. For example, MR1, MR2, MR3, and MR4 have an MS of 50% with 93% line coverage, while MR5 has an MS of 33% with 64% line coverage. When all 5 MRs are combined, the MS increases to 67% with 100% line coverage. However, when test strength is used, the effectiveness of MR5 increases to 40%, as mutants that survive due to lack of coverage are not counted.

With the withdrawal function and considering all MRs, MR1, MR2, MR3, and MR4 are more effective, while MR5 and MR6 are less effective. This can be seen in the mutation score (MS) of each MR, where MR1, MR2, MR3, and MR4 have an MS of 44% with 82% line coverage, while MR5 has an MS of 22% with 53% line coverage, and MR6 has an MS of 33% with 71% line coverage. When all six MRs are considered, the MS increases to 67% with 100% line coverage. However, when using the test strength metric, it can be seen that MR5 becomes more effective with a score of 40% and MR6 with a score of 38% since the mutants that survive due to lack of coverage are not counted.

Regarding the transfer function and all the generated MRs, MR1, MR2, MR3, and

MR4 are more effective than MR5 and MR6. Specifically, MR1, MR2, MR3, and MR4 have a higher MS of 44% with 82% line coverage compared to MR5's MS of 22% with 53% line coverage. When all 6 MRs are taken into account, the MS increases to 67% with 100% line coverage. However, by using test strength, it becomes apparent that the effectiveness of MR5 increases to 40% and MR6 to 38% since the mutants that survive due to lack of coverage are not taken into consideration.

Finally, if all 17 MRs are utilized collectively, they can eliminate up to 75% of all mutants, and the combined value of both MS and test strength is greater than that of any individual MR. This implies that, as long as there are no concerns about testing expenses, many MRs as possible should be employed to create test suites.

Threats to validity

Possible risks exist when validating the examined MT framework in this study. Two types of threats are discussed: external and internal threats [102]. One of the significant external threats to validation is the ability to generalize the study results to other cases. The study utilized a demo banking application with mathematical functions that perform standard calculations and a relatively small data set. Therefore, the results may not explicitly confirm that this approach will work for industrial-sized software.

The PIT mutation testing tool is used to produce mutation analysis for the experiments. The method utilizes a third-party tool, which may contain potential faults which could threaten the proposed method. This type of threat could occur concerning internal validity.

Related Work

When an oracle is absent, the effectiveness of testing techniques is limited. The use of metamorphic testing (MT) in this paper is effective in conducting testing without requiring

an oracle. MT has been utilized to address the Oracle problem in fault-based testing and symbolic execution. Several studies have examined the oracle problem in various fields. However, there needs to be research on the functional testing of banking software using MT. In one case study, Chen et al. [103] injected a seeded fault into a program that implemented partial differential equations. They compared the efficacy of special test cases versus MT in catching defects. While special test cases missed seeing the fault, MT could identify it by employing only one metamorphic relation. Aruna and Prasad [104] suggested multiple MRs for multi-precision arithmetic software, which is evaluated with four mathematical projects and mutation testing. The banking functions share similarities with these mathematical processes. Therefore, we attempted to assess these functions using MT.

Conclusion and Future Work

This study confirms that the MT framework is suitable for evaluating banking functions and that MT is an efficient and effective testing technique even without an oracle. MT can test a banking application's deposit, withdrawal, and transfer functions without needing oracles, which is a significant advantage for testing complex banking functions. The experimental results demonstrate that MT can detect up to 75% of mutants, indicating its high fault detection capability.

For future work, we plan to extend the testing of the banking software beyond the function level and assess the security aspects of the banking application using MT. We also aim to improve the automation capability of MT by predicting MRs based on program execution flow. This will help the developers so they don't have to define the MRs manually. Additionally, we intend to conduct further empirical studies to evaluate the effectiveness of MT for banking software with more complicated functions and deploy the framework to test various financial and payment applications.

METAMORPHIC RELATION PREDICTION FOR SECURITY VULNERABILITY TESTING OF ONLINE BANKING APPLICATIONS

Contribution of Authors and Co-Authors

Manuscript in Chapter 8

Author: Karishma Rahman

Contributions: Problem identification and proposing solution, running experiment, manuscript writing, creating tables and figures. Primary writer.

Co-Author: Ann Marie Reinhold

Contributions: Contribution in manuscript editing/writing, provided feedback, guidance and advice.

Co-Author: Clemente Izurieta

Contributions: Contribution in manuscript editing/writing, provided feedback, guidance and advice.

Manuscript Information

Karishma Rahman, Ann Marie Reinhold and Clemente Izurieta

2025 IEEE International Conference on Cyber Security and Resilience

Status of Manuscript:
____ Prepared for submission to a peer-reviewed journal
____ Officially submitted to a peer-reviewed conference
_X_ Accepted by a peer-reviewed journal
____ Published in a peer-reviewed journal

IEEE
April 28, 2025

Abstract

Software is the backbone of modern systems and daily life, and the reliability of its testing is paramount. A key challenge in software testing is the Oracle problem, which determines whether software behavior is correct. In modern web systems, such as online banking applications, this often involves verifying outputs generated from a large set of inputs, which is both complex and costly. Full automation is necessary to reduce costs and improve the efficiency of these application development processes. However, achieving such automation requires strategies for automatically generating test inputs and addressing the Oracle problem, including distinguishing between correct and incorrect system behavior. Metamorphic Testing (MT) effectively addresses the Oracle Problem by evaluating software through its core characteristics. This method produces various inputs for a program, applies specific transformations, and compares the outputs to the originals to ensure accuracy. Metamorphic Relations (MRs) are central to MT, which define how outputs should change in response to input modifications. While effective, identifying MRs has traditionally been a manual and time-consuming endeavor that depends on specialized knowledge, leading to potential errors. Automating the MR identification process may enhance the efficiency and reliability of MT. This paper introduces an automated MT approach for programs with vulnerabilities in online banking applications. We identified MRs to test for these vulnerabilities among the OWASP's top 10. We used graph representations of the programs to develop a prediction model to automate MR detection for vulnerable programs. We provide a catalog of 8 system-agnostic MRs to enhance security testing automation. The results show that most MRs have prediction scores of more than 80%. Our results demonstrate that this approach is not just theoretical but practical. It scales effectively, enabling overnight automated security testing and making MT a valuable and powerful tool for improving the security of any online banking application.

Introduction

Recent advances in science and technology have significantly transformed the software industry [92]. As the demand for software increases—especially for complex calculations, data processing, and online functionality—software engineers must ensure these systems are reliable [92]. Everyday software systems, including web applications, are highly reusable E-type systems crucial for economic growth [105]. E-type systems are real-world systems [92]. According to Lehman's Law [92], E-type systems continuously evolve and become more complex. The development and testing of software, especially web applications, constitute vital stages in the software development lifecycle (SDLC) [6]. These stages require effective validation and verification techniques to maintain the system's reliability, security, and reusability. The growing demand for web applications and everyday software has led to more complex projects and an increase in research across various areas of software [6]. Testing software, particularly web applications, is challenging due to high complexity, strict security, and accuracy requirements [6]. Modern software and web applications often involve intricate designs, multi-layered workflows, and various features while processing sensitive data like personal and financial information [6]. As a result, precise testing is crucial because any gaps in test coverage can lead to data breaches, vulnerabilities, fraud, and other serious issues [6, 106]. Additionally, the automation of testing techniques is essential to improve the efficiency and reliability of software and web application development processes.

Web systems are software applications that deliver services via web pages. They are commonly employed for online activities like e-commerce and banking. Since these systems handle critical operations, including credit card processing and protecting sensitive customer information, their security is essential. Security testing aims to discover vulnerabilities and weaknesses that hinder these systems from fulfilling their security requirements [107]. Despite advancements in software development technologies and programming languages,

testing remains an integral fragment of the Software Development Life Cycle (SDLC) [6]. It often accounts for more than 50% of total software development expenditures due to its costly, time-intensive, and complex nature [6]. Creating dependable systems continues to be challenging, and researchers and practitioners actively seek more efficient testing methods [6]. The testing process is often prone to human error. Test cases are executed on the system during testing, and a Test Oracle is used—whether automated or manual—to determine if the system behaves as expected [108]. The actual output is compared to the expected result. The Test Oracle problem occurs in complex systems when assessing whether the program's output for a given test case is correct becomes challenging [7]. This issue is complicated in systems with intricate functionality, such as banking software. For instance, in electronic payment systems, it may be unclear how much should be charged for a given input. The problem becomes even more complicated when dealing with transfer charges between bank accounts or currency exchange applications. Finding vulnerabilities in web systems during testing is further complicated by multiple input interfaces, such as web pages accessed via URL requests. Each interface can handle a wide variety of inputs, such as web forms, cookies, and URL parameters, which may vary depending on the user's role [48]. Vulnerabilities can emerge from specific combinations of user roles, URL requests, and input parameters. Therefore, testing must cover various inputs, including those intentionally crafted to exploit the system. Automated strategies for generating security tests are necessary to address this challenge [48].

Metamorphic Testing (MT) is a technique that has proven useful in certain situations to tackle the challenge of the oracle problem [18]. The principle behind MT [8] is that it may be easier to analyze the relationships between the results of multiple test executions, referred to as Metamorphic Relations (MRs), rather than specifying the input-output behavior of a system [8]. MT utilizes MRs to determine system properties by automatically transforming the initial test input into a subsequent test input [8]. If the system fails to comply with the

MRs when tested with both the initial and subsequent inputs, it is inferred that it is defective [8]. A considerable amount of research has focused on developing MT methods for specific domains such as computer graphics, web services, and embedded systems [9–11, 15], and the corresponding MRs are essential for enhancing the efficiency of fault detection in testing. Traditionally, pinpointing these relations involves a labor-intensive, error-prone manual process, often requiring input from domain experts, mainly when dealing with intricate programs. Hence, developing automated techniques to identify MRs is a promising avenue. Such automation has the potential to significantly improve the efficiency and effectiveness of MT, rendering it a more feasible and dependable method for guaranteeing the reliability of any system.

Our research goal is described as follows:

- **RG**: *To apply MT to tackle the Test Oracle problem in security vulnerability testing of online banking applications.*

We systematically define MRs that capture properties (i.e., characteristics that are compromised when the system is at risk) among the top 10 OWASP vulnerabilities found in online banking applications and automate testing using these MRs for such vulnerable-prone programs. These MRs serve as guidelines or rules governing how the program should change in response to different inputs or variations. They are crafted to capture specific program characteristics that are particularly vulnerable or compromised when the system is at risk. An example of an MR to identify *bypass authorization schema* vulnerabilities is *a web system that should return different responses for two users when the first user requests a URL provided by the GUI* (e.g., in HTML links). In other words, a user should not be able to access URLs that are not provided by the GUI directly. We have developed a technique to automate the testing process by predicting these MRs. Automating the testing process reduces the manual effort and potential errors associated with traditional testing methods.

Here, we present an automated metamorphic testing approach that helps test engineers specify metamorphic relations to capture security requirements of Web systems (i.e., online banking) and automatically detect vulnerabilities (i.e., violations of security requirements) by predicting metamorphic relations. Our approach is constructed on the following novel contributions:

- A catalog of 8 MRs targeting the well-known top 10 OWASP's security vulnerabilities commonly found in online baking applications, and
- A framework that automatically predicts MRs for unseen programs, that are prone to vulnerabilities, using a Graph-based Convolutional Neural Network (GCNN) model.

We applied our approach to 679 vulnerable-prone programs from different sources. The approach automatically detected MRs with prediction scores greater than 80% for most MRs. Considering these results and assessing the effort involved, our approach is practical and effective in addressing the Oracle issue of automatically testing online banking applications.

This paper is organized in the following manner. Section II offers background information on MT and details the testing process for baking applications. In Section III, we outline our approach. Section IV explains the fundamental experimental setup. Section V shares the paper's findings. Finally, Section VI addresses the threats to validity, leading to the conclusion in Section VII.

Background

This section introduces MT and its operation according to the corresponding MRs. It also illustrates the challenges of testing vulnerabilities for banking applications and how MT can alleviate the issue.

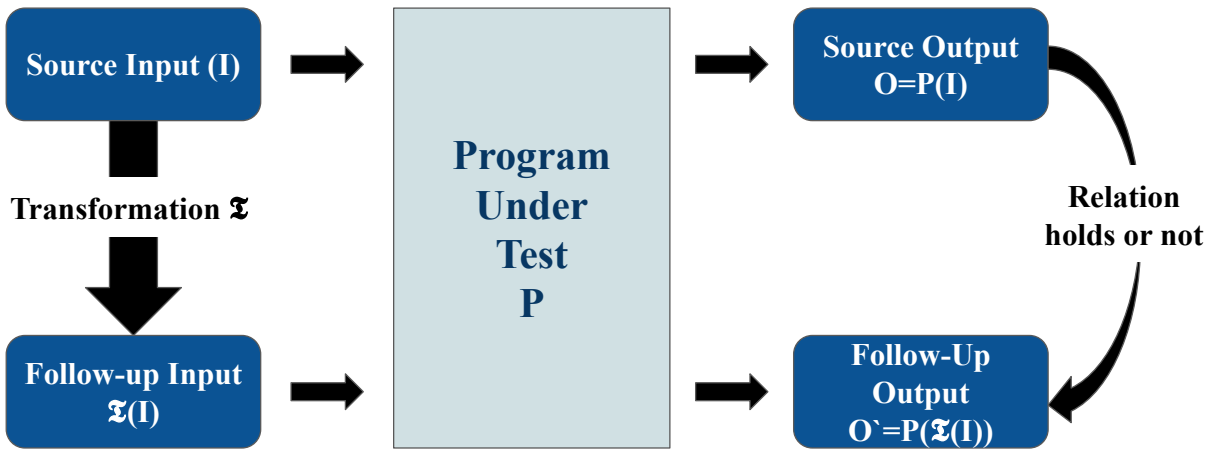

Figure 22: Overview of the Metamorphic Testing (MT) Process

Metamorphic Testing

Metamorphic Testing (MT) is a technique designed to address the well-known Test Oracle problem [18]. This problem arises when it is difficult or impossible to determine the correctness of individual program output. MT, developed by Chen et al. [8], provides a solution by checking if a program satisfies specific predefined properties called Metamorphic Relations (MRs). These MRs describe how changes to a program's input should affect its output [8]. If the output does not behave as expected according to these MRs, it may indicate a fault in the program [8]. MT is particularly valuable for identifying defects in programs where traditional Test Oracles are unavailable or impractical [18]. By examining the relationships between inputs and outputs across multiple executions, MT can detect faults even when the correct result of each execution is unknown [16].

The general steps for implementing MT are as follows (Figure 22) [8]:

- **Identify MRs:** Define a set of MRs that the program under test should satisfy.

- **Create Initial Test Cases:** Develop a set of test cases to serve as the source inputs.
- **Generate Follow-Up Test Cases:** Apply input transformations specified by the MRs to create follow-up test cases from the initial ones.
- **Execute and Compare Outputs:** Run both the initial and follow-up test cases and check if the output changes align with the expected behavior described by the MRs. If a runtime violation of an MR occurs, it indicates a fault in the program.

A simple and widely used example of MT is testing the SINE function $y = \sin(x)$. According to its property, for any input angle x, adding 2π to the input should not change the output. This means $y = \sin(x) = \sin(x+2\pi)$, making it a valid MR. Using this property, the function can be tested as follows:

- Create a source test case with input x and output $\sin(x)$.
- Generate a follow-up test case by applying the transformation $x' = x+2\pi$ and calculate $y = \sin(x') = \sin(x + 2\pi)$.
- Compare the outputs. If $sin(x') \neq sin(x)$, the MR is violated, indicating a fault in the sine function's implementation.

By systematically applying these steps, MT provides an effective and efficient alternative to detect faults in the absence of traditional Test Oracles [18].

Testing Banking Applications

Banking applications handle sensitive financial and personal data, making their security critical. Vulnerabilities, such as weak authentication, injection flaws, insecure communication, and business logic errors, can lead to severe consequences, including financial fraud and reputational damage [48]. These vulnerabilities often arise from the complexity

of banking systems, which include intricate workflows, multiple user roles, and diverse input interfaces such as web pages, forms, and cookies [48]. Common vulnerabilities in banking applications include authentication and authorization issues, injection attacks, cross-site scripting (XSS), business logic flaws, insecure communication, and third-party dependencies [48]. Several testing techniques are commonly employed to detect vulnerabilities in banking applications, including the use of static application security testing (SAST) [46], dynamic application security testing (DAST) [47], penetration testing, fuzz testing, risk-based testing [48], and manual code reviews [48]. These methods often face challenges, such as restricted test coverage, proneness to human error, and difficulties in addressing dynamic behaviors, commonly called the Oracle Problem [14]. Frequent configuration changes and diverse input parameters in banking applications further complicate the testing process.

MT addresses these gaps by leveraging MRs. Unlike traditional methods, MT does not require precise expected results, making it particularly useful in scenarios where Test Oracles are unavailable or impractical [18]. MT resolves the Oracle Problem by focusing on input-output relationships. Additionally, it automates test case generation and validation, reducing human error and systematically covering complex input combinations to improve test coverage [18]. For instance, SQL injection is a common vulnerability in banking applications. Attackers can manipulate input fields, such as login forms or search bars, to execute unauthorized SQL queries. If user inputs are not properly validated, an attacker could bypass authentication by submitting crafted inputs and gaining unauthorized access to sensitive user data. This vulnerability allows attackers to manipulate or delete records, steal financial information, and perform fraudulent transactions. MT can verify the system's behavior by testing transformed inputs to ensure proper handling and validation, mitigating such risks. As banking systems grow in complexity, MT offers a scalable and practical approach to ensure their security and reliability. By addressing the limitations of traditional testing methods, MT enhances the efficiency and effectiveness of vulnerability detection in

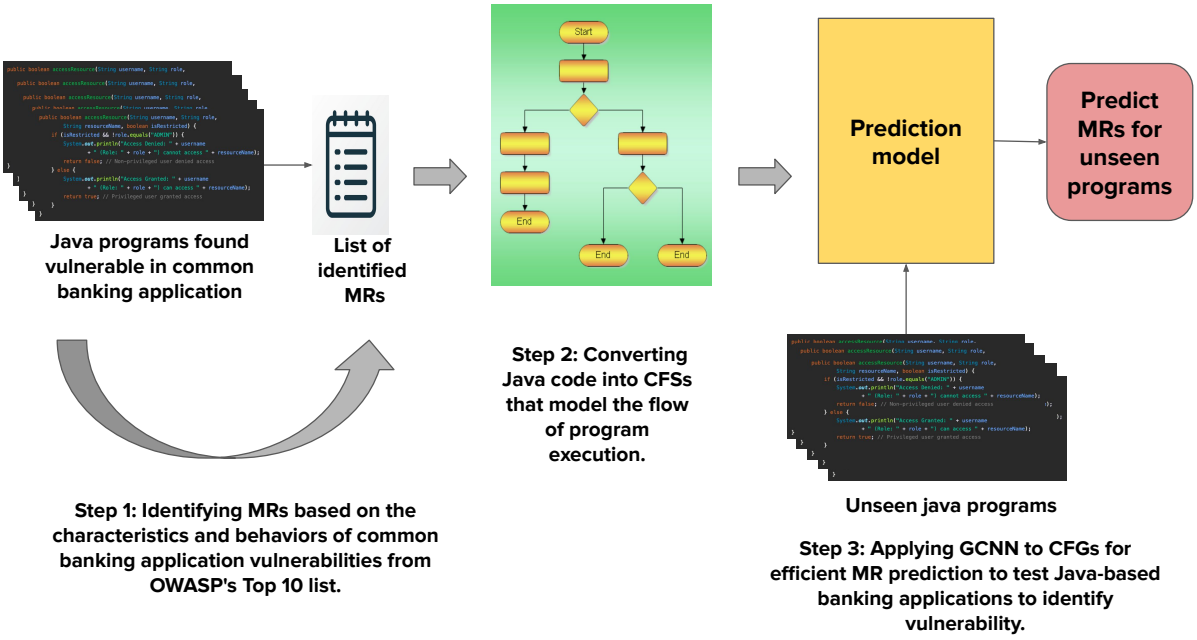

Figure 23: Overview of the proposed approach for automating metamorphic testing (MT) to predict MRs for vulnerabilities in banking applications. The method consists of three main steps: (1) identifying metamorphic relations (MRs) for vulnerabilities from OWASP's Top 10 list related to banking applications, (2) generating control flow graphs (CFGs) from Java source code to capture execution behavior, and (3) applying a graph convolutional neural network (GCNN) to analyze the CFGs for MR prediction.

banking applications.

Approach

In this section, we introduce a new approach to automating the MT technique. We predict MRs for vulnerable-prone programs commonly found in banking applications, which fall under OWASP's Top 10 vulnerabilities. The proposed method, illustrated in Figure 23, consists of three key steps:

1. identifying MRs for vulnerabilities related to banking applications among OWASP's Top 10 vulnerabilities, making our method directly applicable to real-world scenarios,

2. generating control flow graphs that capture the execution behavior of vulnerable-prone Java programs, and

3. applying a graph-based model to analyze CFG datasets for MR prediction, ensuring a comprehensive and accurate assessment.

In the first step, we define MRs based on the characteristics and behaviors of vulnerabilities that can be tested using MT in Java-based banking applications. In the second step, we construct CFG representations from Java source code to model program execution flows. The generated CFGs provide a structured, graphical representation of program behavior, essential for further analysis. In the final step, we employ graph-based classification models to analyze the CFGs. Specifically, we introduce a graph convolutional neural network to automatically learn predictive patterns from CFG data, enabling efficient and accurate vulnerability testing in banking applications by predicting MRs.

Metamorphic Relations (*step 1*)

The first task is to identify MRs. We developed 8 MRs from the OWASP's Top 10 (2021) list. To identify the MRs, we focused on properties related to vulnerabilities found in online banking web applications. We named the MRs by numbering them from 1 to 8 (e.g., MR1, MR2, etc.), and we retained the numbering (e.g., A01, A02, etc.) from the original list in case of vulnerabilities. The descriptions for each MR are as follows:

- **MR 1 (A01: Broken Access Control: Authorization-Based Access Control):** Let U_a represent an authorized user and U_u represent an unauthorized user. Let R denote a restricted resource. If $f(U_a, R) \longrightarrow access\_granted$, then $f(U_u, R) \longrightarrow access\_denied$.

- **MR 2 (A02: Cryptographic Failures: HTTPS vs. HTTP Transmission):** Let P represent plaintext data (e.g., credentials or transaction details) and T denote

the transmission protocol. If $T(HTTPS, P) \longrightarrow encrypted$, then $T(HTTP, P) \neq encrypted$.

- **MR 3 (A03: SQL Injection: Authentication Bypass):** Let I_v represent a valid login input and I_m denote a malicious SQL payload. If $DB\_query(I_v) \longrightarrow valid\_response$, then $DB\_query(I_m) \neq valid\_response$.

- **MR 4 (A04: Insecure Design: Password Reset Token Exposure):** Let T_s represent a securely generated token, and T_u represent an unsecured token. If $Validate(T_s) \longrightarrow valid\_reset$, then $Validate(T_u) \neq valid\_reset$.

- **MR 5 (A05: Security Misconfiguration: Default Credentials Exposure):** Let C_u represent a user-defined credential and C_d denote a default credential. If $Auth(C_u) \longrightarrow secure\_login$, then $Auth(C_d) \neq secure\_login$.

- **MR 6 (A07: Authentication Failures: Session Expire):** Let S_v represent a session within its valid time frame, and S_e denote an expired session. If $Auth(S_v) \longrightarrow access\_granted$, then $Auth(S_e) \longrightarrow access\_denied$.

- **MR 7 (A08: Software and Data Integrity Failures: Checked vs. Unchecked Transactions):** Let T_c denote a transaction that includes checksum validation, while T_u represents a transaction that does not include checksum validation. If $Process(T_c) \longrightarrow valid\_transaction$, then $Process(T_u) \neq valid\_transaction$.

- **MR 8 (A10: Server-Side Request Forgery (SSRF): Allowlisted vs. Denylisted URLs):** Let U_a represent an allowlisted URL and U_d represent a denylisted URL. If $Request(U_a) \longrightarrow allowed$, then $Request(U_d) \longrightarrow blocked$.

Function Representation using Control Flow Graphs (CFG) *(step 2)*

The next step in this approach involves converting a function into its Control Flow Graph (CFG). This representation is chosen because it facilitates the extraction of information regarding the sequence of operations within a control flow path, which directly corresponds to the MRs satisfied by a given function [50].

A CFG is a directed graph, $G_f = (V, E)$, representing a function *f*. Each node $v_x \in V$ corresponds to a statement x in *f*, with the operation performed in x labeled as $label(v_x)$. If x and y are statements in *f*, and y executes immediately after x, an edge $e = (v_x, v_y) \in E$ is established. The control flow of f is defined by all edges in the graph, while the entry and exit points are represented by nodes v_{start} and v_{exit}, respectively [50].

To construct the CFGs, we utilize the *Soot*[1] framework. This tool generates CFGs in *Jimple*, a typed three-address intermediate representation of Java code, where each CFG node represents an atomic operation [51]. After generating the CFGs, we refine them by labeling each node according to the operation it performs. Additionally, we annotate all method call nodes with their return types.

This study aims to develop a technique for identifying MRs from vulnerable characteristics in Java source code. In this context, CFGs produced as intermediate representations of compiled source code serve as inputs for prediction models to detect MRs to test vulnerable programs. Prior research has demonstrated the successful application of CFGs in various fields, including malware analysis [109, 110] and software plagiarism detection [111, 112]. Since semantic errors often become apparent only at runtime, analyzing execution flows can be valuable in distinguishing faulty patterns from correct ones.

[1] `https://www.sable.mcgill.ca/soot/`

Prediction Model *(step 3)*

We used a graph model to predict MRs for vulnerable-prone programs. The Graph Convolutional Network (GCN) is a dynamic graphical model that processes large-scale graphs with intricate structural relationships [31]. Unlike simple node-based representations, a vertex in the GCNN framework is not merely a token but it encapsulates a rich set of features derived from its connectivity within the graph [31]. For instance, in a CFG context, each vertex signifies an operation that may consist of multiple attributes, including the instruction type and operands. In the GCNN model, the first layer is called the embedding layer [31], where each vertex is mapped to a real-valued vector corresponding to its feature representation. Next, we apply two graph convolutional layers [31] that iteratively update the node embeddings by aggregating information from neighboring nodes. These layers capture local graph structures and enhance feature representations at multiple levels. The network has batch normalization layers to stabilize training and improve convergence. A ReLU activation function [31] is applied after each convolutional operation to introduce non-linearity. After the convolutional layers, a global mean pooling layer is used to extract a unified feature representation of the entire graph. Unlike standard CNNs with fixed input dimensions, graphs feature varying sizes, resulting in inconsistencies in feature extraction [31]. Pooling addresses this by consolidating the learned node features into a single vector representation. Ultimately, the feature vector is processed through a fully connected layer and an output layer, where categorical distributions for classification tasks are generated. In this manner, a graph-based model automates vulnerability testing in Java programs by predicting MRs and capturing execution flow characteristics.

Experiments

We experimented with a dataset of programs prone to vulnerabilities commonly found in banking applications. Our aim was to generate a graph-based model to predict MRs for those programs. This section provides a detailed description of the dataset, the experimental setup, and the evaluation metrics used.

Dataset

We constructed a dataset of 679 Java programs prone to vulnerabilities in online banking applications. This dataset is curated based on the OWASP Top 10 vulnerabilities, which represent the most critical security issues for web applications. We built the dataset by collecting Java programs from the following sources:

- **AI-generated vulnerable programs:** Similar code samples were generated using ChatGPT (265) and Meta AI LLaMA (265), focusing on the OWASP Top 10 vulnerabilities.
- **Open-source repositories:** Vulnerable Java applications sourced from open-source projects on GitHub (98).
- **Manually curated examples:** Additional vulnerable samples are chosen to ensure diversity (51).

Each program contains security flaws such as SQL injection, broken access control, authentication failures, and other OWASP vulnerabilities. Figure 24 is an example of a vulnerable-prone Java program generated from ChatGPT. Once the Java programs are collected, they are converted into CFGs representing the program's execution flow. The CFGs are generated using static analysis tools called Soot (a Java optimization framework). It extracts control flow structures from Java bytecode. The generated CFGs represent nodes

as instructions or basic blocks, while edges indicate execution flow between these instructions. Each CFG is stored in DOT file format [113], which provides a graphical representation of program execution.

Experimental Setup

The experiment aimed to analyze and predict MRs for vulnerable-prone Java programs by leveraging CFGs and prediction models. Each node in the CFG is assigned a label based on its operation (e.g., ASSIGNMENT, IF_CONDITIONS). These labels were converted into unique 16-dimensional binary vectors, which serve as node features in the graph representation. For example, all the nodes with the same type of operations have the same 16-dimensional binary vectors. Graph nodes were assigned unique features based on their operations, and edges were structured as adjacency matrices. The dataset was split into training and test sets to evaluate the model's performance. These splits were done based on three categories. They were:

1. train with ChatGPT, Open-source repositories, and manually crafted data. Test with Meta AILLaMa data,

2. train with Meta AILLaMa, Open-source repositories, and manually crafted data. Test with ChatGPT data, and

3. train with Meta AILLaMa and ChatGPT data. Test with Open-source repositories and manually crafted data.

Also, binary class labels were supplied to train the models for each MR.

To ensure a comprehensive evaluation, we compare the effectiveness of Graph Neural Networks (GNNs) against traditional prediction models. A two-layer GCNN model followed by a fully connected layer is used to predict vulnerability classes. We used Support Vector Machine (SVM) approaches with two feature extraction types developed in previous studies

```
public boolean accessResource(String username, String role,
        String resourceName, boolean isRestricted) {
    if (isRestricted && !role.equals("ADMIN")) {
        System.out.println("Access Denied: " + username
                + " (Role: " + role + ") cannot access " + resourceName);
        return false; // Non-privileged user denied access
    } else {
        System.out.println("Access Granted: " + username
                + " (Role: " + role + ") can access " + resourceName);
        return true; // Privileged user granted access
    }
}
```

Figure 24: Simplified Access Control Method Implementing Basic Role-Based Authorization. This method checks if a user with a specific role is allowed to access a restricted resource. Only users with the "ADMIN" role are granted access to restricted resources, while others are denied. The output clearly indicates whether access is granted or denied based on the user's role. Metamorphic Relation 1 (MR1) for Authorization-Based Access Control (RBAC) Violation can be used to test this method, ensuring that privileged users are granted access while non-privileged users are correctly denied.

[12, 13]. One is random kernel-based SVM [12] and SVM with Bag-of-Words (BoW) [13]. We also used the multi-layer perceptron (MLP), which is a fully connected feedforward neural network trained on the same dataset [30]. The results show the average prediction scores for all the categories of train tests split together.

Evaluation Measures

The approaches are evaluated using two widely recognized performance metrics: the area under the receiver operating characteristic (ROC) curve (AUC) and accuracy.

The AUC (Area Under the Curve) provides a more comprehensive evaluation of the model's performance [114]. It assesses the classifier's ability to distinguish between two classes by quantifying the area under the ROC curve, which plots the true positive rate (TPR) against the false positive rate (FPR) at different classification thresholds [114]. The

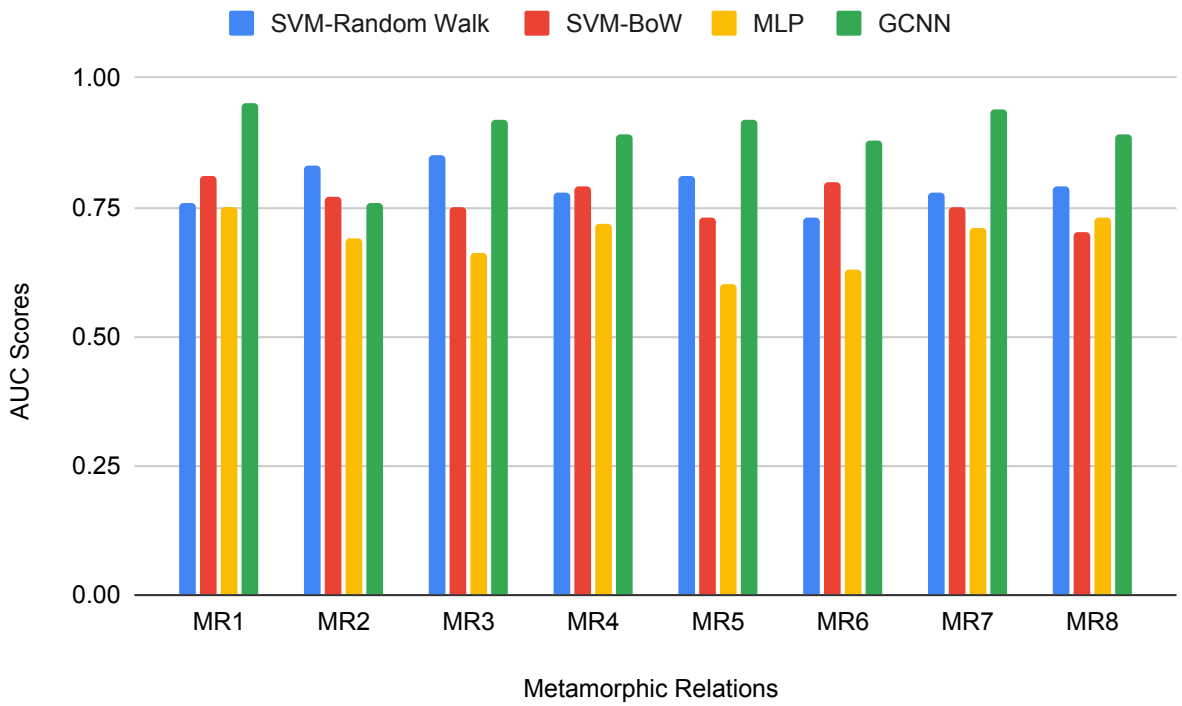

Figure 25: AUC Score Distribution for Metamorphic Relations (MR1–MR8) Using SVM-Random Walk, SVM-BoW, MLP, and GCNN Models.

comprehensive nature of AUC evaluation provides reassurance and confidence in the model's performance, especially when class distributions are imbalanced [114]. A classifier with an AUC of 1.0 perfectly distinguishes between the two classes, while an AUC of 0.5 indicates no better performance than random guessing [114].

Accuracy measures the proportion of correctly classified instances among all predictions and is a fundamental metric for evaluating classification models [115]. Accuracy, while a fundamental metric for evaluating classification models, can be misleading when class imbalances exist. In such cases, where one class has significantly fewer samples than the other, a model biased toward the majority class may achieve high accuracy but perform poorly in detecting minority-class instances. [115].

Results And Discussion

The performance of four classification models—Support Vector Machine (SVM) with Random Walk kernel, SVM with Bag of Words (BoW), Multilayer Perceptron (MLP), and Graph Convolutional Neural Network (GCNN) was assessed using two key performance metrics, the Area Under the Receiver Operating Characteristic Curve (AUC) and accuracy. This evaluation was conducted to determine the effectiveness of the models in predicting MRs for vulnerable-prone Java programs in banking applications.

The GCNN consistently outperformed the other models, achieving the highest AUC scores across most MRs. Figure 25 shows GCNN achieved AUC scores of 0.95 (MR1), 0.92 (MR3), 0.92 (MR5), and 0.94 (MR7). It indicates its capability to capture complex structural patterns from the CFGs of the programs. The SVM with Random Walk kernel also demonstrated strong performance, particularly for MR2 (0.83) and MR3 (0.85). In contrast, the MLP model showed relatively lower AUC scores. Here, the performance declined notably for MR5 (0.6) and MR3 (0.66). It illustrates its limitations in addressing structural graph data compared to GCNN. SVM-BoW showed average performance, with scores ranging from 0.7 to 0.81. It reflects its dependency on text-based feature representation, which may not fully capture the complex program behaviors.

The accuracy results further confirm the AUC findings, emphasizing the robustness of GCNN in predictive performance. Figure 26 shows that GCNN achieved the highest accuracy for MR6 (0.91), MR3 (0.89), and MR8 (0.87), demonstrating its robustness in accurately classifying MRs. SVM with Random Walk kernel upheld competitive accuracy, particularly in MR4 (0.89) and MR5 (0.83). However, its performance slightly varied across different MRs. The MLP model lagged, with the lowest MR2 (0.62) and MR6 (0.63) accuracy, reflecting its struggles with feature extraction from complex graph data. SVM-BoW performed relatively well. It achieved accuracy scores of 0.81 (MR7) and 0.8 (MR8)

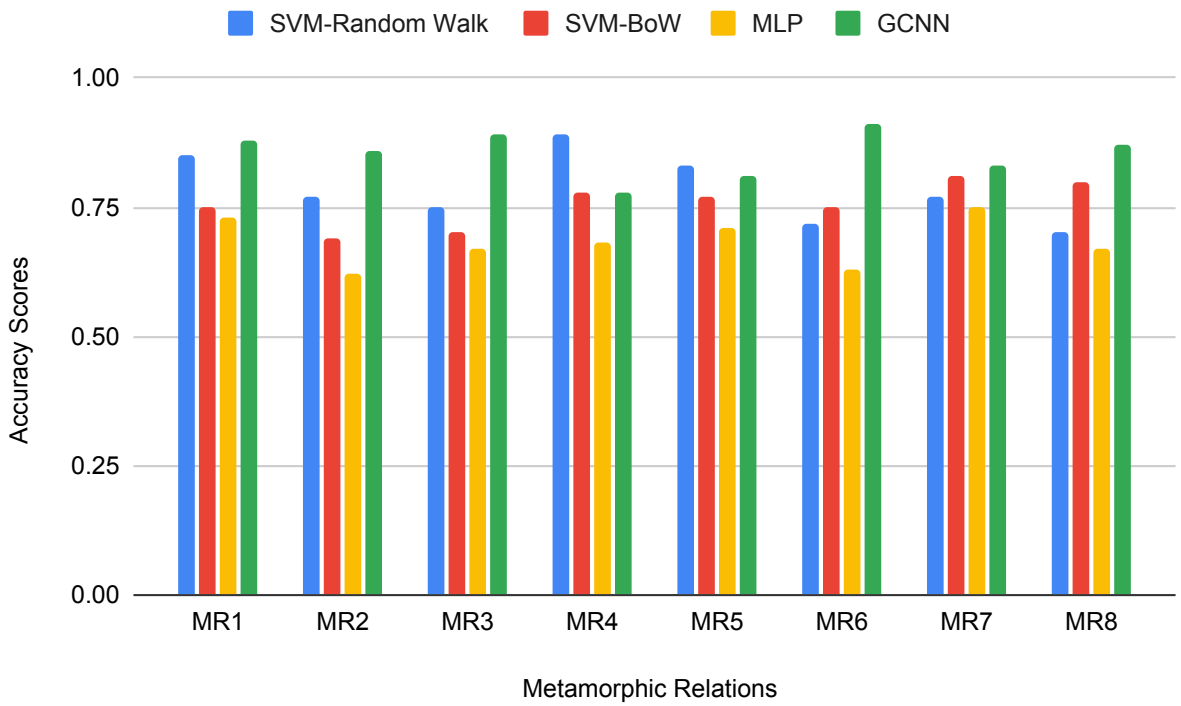

Figure 26: Accuracy Score Distribution for Metamorphic Relations (MR1–MR8) Using SVM-Random Walk, SVM-BoW, MLP, and GCNN Models.

but did not match the consistency observed with GCNN.

The results emphasize the advantage of leveraging graph-based models, particularly GCNN, in analyzing control flow graphs for MR prediction. The AUC and accuracy scores of GCNN can be attributed to its ability to learn hierarchical feature representations from graph-structured data, which traditional models like SVM and MLP cannot fully manage. The relatively strong performance of SVM with Random Walk kernel suggests that kernel-based approaches still hold value, especially when dealing with graph structure data. However, the lower performance of MLP highlights the challenges neural networks face, such as the lack of specialized architectures for graph data. Overall, the findings validate the advantage of leveraging graph-based models for MR prediction for vulnerable-prone Java programs, particularly in domains where program structure plays a crucial role.

Threats to validity

External validity [116] is concerned with the generalizability of the findings beyond the specific dataset and experimental setup used in this study. The study was conducted on a dataset that does not represent all real-world applications. The results may not generalize to datasets with different characteristics, such as varying feature distributions, noise levels, or larger sample sizes. While the models perform well on the given dataset, their scalability to large-scale real-world applications remains uncertain. The effectiveness of GCNNs depends on the graph structure of the data. If the dataset does not naturally fit into a graph representation, the performance of GCNNs might be inferior to traditional models like SVM or MLP. This limits the generalization of GCNNs to various classification tasks.

Internal validity [116], which refers to potential biases, errors, or confounding factors affecting the experimental results, is critical. The study relies on widely used frameworks such as Soot, Scikit-learn, TensorFlow, and Graph Neural Network (GNN) libraries like PyTorch-Geometric. However, while popular, these frameworks may still contain undocumented bugs or implementation inconsistencies that could impact the experimental outcomes. Therefore, further validation on more extensive and diverse datasets is necessary to confirm the robustness of the findings and to ensure the ongoing relevance of this research.

Conclusion

Our research goal was *To apply MT to tackle the Test Oracle problem in security vulnerability testing of online banking applications.*

We evaluate the effectiveness of automatically predicting MRs using graph models for vulnerable-prone Java programs commonly found in banking applications. MT is a testing technique for programs lacking a suitable Test Oracle. Manually identifying such MRs for various applications poses challenges for testers. In prior work, we developed a graph

kernel-based machine learning approach for predicting MRs using supervised learning and control flow graph features tailored for complex scientific programs. Results indicate that utilizing control flow graph features to compute the graph kernel of a testing program for training a machine learning model enhances MR prediction. This research evaluates the effectiveness of using graph representations directly to train graph models of vulnerable-prone Java programs. Graph Convolutional Neural Networks (GCNNs) are evaluated using a control flow graph of vulnerable-prone programs in online banking applications. Eight types of metamorphic relations are manually identified for predicting MRs that cover the properties of vulnerabilities among the top 10 OWASP vulnerabilities. These MRs are predicted on Graph Convolutional Neural Networks (GCNNs), MLP, random kernel-based SVM, and SVM with bag-of-words. The result shows the advantage of leveraging graph-based models, particularly GCNN, in analyzing control flow graphs for MR prediction. GCNN consistently shows the highest AUC scores across most MRs, with values ranging from 0.76 to 0.95, indicating its ability to discriminate between classes.

The proposed method has several potential extensions. Firstly, to enhance external validity, utilizing a larger dataset with various functions can improve accuracy. Currently, the method focuses on predicting a single metamorphic relation, relying on a binary classification system to train predictive models. However, it can be adapted to predict metamorphic relations using multi-class models. Additionally, this approach could be expanded to include datasets from other programming languages, such as Python, C, and Fortran, which would be advantageous for the scientific community.

CONCLUSIONS & FUTURE WORK

Conclusions

This thesis assesses the effectiveness of classification techniques in automatically predicting MRs. This chapter summarizes the overall conclusions of the thesis and suggests potential future research in this area.

The primary goal of this thesis is to automate the MT process by predicting MRs for scientific and vulnerable applications to reduce the manual labor of identifying MRs. MT represents a valuable approach, especially for programs lacking a suitable test oracle. It involves systematically examining a set of properties known as MRs, which describe the relationships between program inputs and their corresponding outputs. The presence of an error in the testing program becomes evident when an MR is violated. One of the challenges faced by testers in the field is the manual identification of MRs for diverse applications. This process can be complex and time-consuming. In response to this challenge, the research presented in this thesis describes the automation of the MT process by leveraging machine learning to predict MRs. By doing so, I streamline and expedite the testing procedure, making it more efficient and less reliant on human-intensive efforts. It is worth noting that prior studies in this area have yielded promising outcomes, demonstrating the feasibility of adopting this automated approach for predicting MRs. In previous work by Kanewala et al. [38], an automated approach is utilized to predict MRs by employing supervised learning for more straightforward programs. Their findings indicate that their approach can effectively train a machine-learning model for predicting MRs. This thesis assesses the effectiveness of explicit graph representation, text-based features, and advanced classification learning algorithms. It evaluates classification methods for programs involved in matrix calculations, banking application functionalities, and those prone to vulnerabilities.

In Chapter 4, ten MRs are manually identified to predict MRs for matrix programs,

and these answer RQ 1.1. These MRs are predicted using a supervised model built using SVMs. The results indicate that the random walk kernel outperformed the graphlet kernel in 7 out of 10 MRs, while the AUC values for the remaining MRs were ≥ 0.74. Generally, an AUC ≥ 0.7 suggests the strong performance of the trained model. The findings indicate that MR2, MR5, and MR7 effectively utilize the graph kernel-based supervised machine learning approach. Each has achieved an AUC score of 0.9 or higher with the random walk kernel. This implies that graph kernels based on walks within the graphs are highly effective for predicting MRs. This answers RQ 1.2 by stating the significance of the execution flow as a feature for building the predictive model. Thus, RG 1 is fulfilled with the conclusion that by utilizing a supervised model, the proposed method represents an effective approach for predicting MRs in programs that execute matrix calculations with minimal human labor.

In Chapter 5, I introduce a method for text classification for predicting MRs for matrix calculation functions based on their documentation. In the results, AUC scores of the MR prediction models—MRpredT-NB (Naive Bayes) and MRpredT-SVM —are compared against MRpred, a random walk graph kernel classification model utilizing SVMs. MRpredT-NB surpasses MRpredT-SVM in 4 out of 10 MRs (MR1, MR2, MR3, MR10), while MRpredT-SVM shows superior performance for 6 out of 10 MRs. Importantly, for 2 out of 10 MRs (MR7 and MR9), one model from MRpredT exceeds MRpred, highlighting the potential of text data for MR prediction, suggesting a need for further exploration. In contrast, MRpred dominates MR1, MR3, MR5, and MR8. These results answer RQ 2.2 by accurately predicting MRs for the text classification model. Answering RQ2.1, the findings suggest that a machine-learning approach based on text classification can effectively predict MRs, especially when using an SVM model, which can reduce human error and, therefore, fulfill RG 2.

In Chapter 6, I present a mapping study on empirical research of web application security vulnerability detection from 2001 to 2021. It examines a total of 150 papers from

IEEE Xplore and ACM Digital Libraries, and of them, 76 are included in the analysis. This number of studies showcasing effective security testing methods presents the answer to RQ 3.1. The classification scheme developed in this mapping study indicates that the most prevalent vulnerabilities correspond with the OWASP top ten list. Additionally, while many techniques have been suggested, there are only a few available tools to download, posing a challenge for the research community in influencing industry practices. Despite various approaches to security testing, the oracle problem is still not sufficiently addressed, indicating opportunities for further investigation and, in turn, answering RQ 3.2. Overall, this meets RG 3 by offering an overview of existing research on security testing in web applications, serving as a foundation for developing automated testing techniques.

In Chapter 7, I examine the use of MT in banking software testing and include a case study. This case study analyzes the use of MT for banking software, focusing on Deposits, Withdrawals, and Transfers functions. It introduces six suitable MRs for Withdrawals and Transfers functions and five for Deposits functions, responding to RQ 4.2. MR1 to MR4 outperforms MR5 for the Deposits function, achieving an MS of 50% with 93% line coverage. For the Withdrawal function, MR1 to MR4 also exhibits higher effectiveness, with an MS of 44% and 82% line coverage compared to MR5 and MR6. In the Transfers function, MR1 to MR4 similarly exceeds MR5 and MR6 with an MS of 44% and 82% line coverage. Considering all MRs together for each function increases the MS to 67 and 100% line coverage. Using all 17 MRs can remove as many as 75% of mutants, showcasing the ability of MT to test the functions of banking applications and answering RQ 4.1. RG 4 is thus fulfilled with the significant testing benefit of using MT, which subdues the need for an oracle.

In Chapter 8, I assess the effectiveness of automatically predicting MRs using graph models for Java programs prone to vulnerabilities commonly found in banking applications. This study explores how effectively graph representations can be used to train models for vulnerable Java programs. The study evaluates GCNNs using a CFG of these vulnerable

programs in online banking applications. It manually identifies eight types of MRs based on the traits of vulnerabilities found in the top 10 OWASP vulnerabilities. This culminates in the answer to RQ 5.2 and RQ 5.1, respectively. The study uses GCNNs, MLP, random kernel-based SVM, and SVM with bag-of-words to build the prediction model. The findings highlight the benefits of using graph-based models, especially GCNNs, for analyzing CFGs in MR prediction, answering RQ 5.3 of identifying the best feature extraction approach. GCNN consistently achieves the highest AUC scores across most MRs, with values ranging from 0.76 to 0.95, indicating its effectiveness in distinguishing between classes. The overall result fulfills RG 5 by enhancing MT integration in evaluating vulnerable programs and automating the process by predicting new MRs, minimizing the reliance on manual MR identification.

This thesis provides comprehensive research on automating MT by predicting MRs for the system under test, which will effectively improve the existing software testing techniques by alleviating the oracle problem.

Threats to Validity

Two types of validity threats are discussed here for each study from Chapters 4 to Chapters 8. One of them is External Validity [102], which focuses on how findings generalize beyond the specific dataset and experimental setup. Another is Internal Validity [102], which addresses biases or confounding factors that impact results.

In Chapter 4 and Chapter 5, the main threat to validity is generalizing study results. These studies use 93 mathematical functions for matrix calculations, a small data set, limiting the ability to confirm scalability to industrial-sized software. The method depends on third-party tools, which may introduce faults, posing risks to internal validity. The study leveraged scikit-learn and Soot, both widely used in the computer science community due to their reliability and few faults.

In Chapter 6, my investigation reveals that the use of search string threatens internal validity, but a known construction technique helps to mitigate this. Other threats, like selection criteria and author judgments, are also addressed through agreements. Conclusions are directly linked to the data sets for each research question.

In Chapter 7, a major external threat to validity is generalizing study results to other cases. The study uses a demo banking application with basic calculations and a small data set, limiting confidence in its applicability to larger software. The PIT mutation testing tool produces mutation analysis, but reliance on a third-party tool introduces potential faults that could undermine the method, affecting internal validity.

In Chapter 8, the effectiveness of GCNNs varies with the structure of the graph data. If the dataset does not align with a graph representation, GCNNs may underperform compared to traditional models like SVM or MLP, limiting their application in various classification tasks. The study uses frameworks like Soot, Scikit-learn, TensorFlow, and GNN libraries such as PyTorch-Geometric, which may have undocumented bugs that could affect outcomes.

Future Work

The proposed approaches mentioned in this thesis can be expanded in various ways. One significant area for improvement involves applying this theory to real-world industrial software systems. By using a larger dataset that includes a broader range of functions, future researchers can identify more MRs, which can be utilized to develop better prediction models. The prediction models can be effectively generalized across different software applications, potentially leading to improved accuracy in predicting MRs. Consequently, a dataset comprising diverse functionalities would enable prediction models to learn various types of behavior associated with specific MRs, enhancing their robustness in predicting them.

Another area for improvement is implementing the classification model as a multi-label

approach. This approach emphasized predicting a single MR using a binary classification approach for training predictive models. However, most programs demonstrate multiple MRs in this research, increasing the complexity of the classification process. This challenge can be tackled by converting the method into a multi-class classification problem, enabling the model to simultaneously predict multiple MRs for a given function. Therefore, a multi-class approach may improve efficiency and more accurately reflect real-world situations, where functions often exhibit multiple metamorphic properties.

This thesis focuses on the programs that are written in Java programming language. Although Java is a popular programming language for software systems, many other languages are also widely utilized nowadays. By expanding, it can be used to support multiple languages, such as Python, C, and Fortran; this method can be utilized across a broader array of software systems, providing advantages to developers and researchers in various fields. This expansion would enable comparisons across languages regarding MRs, providing more significant insights into software behavior and aiding in the development of more reliable and robust applications.

Moreover, empirical research could evaluate the effectiveness of metamorphic testing in real-world software systems, particularly in banking applications that involve complex financial calculations and security-sensitive transactions. By applying the framework to more challenging functions, researchers can examine its scalability and dependability in managing high-stakes software environments. Additionally, implementing this approach in testing various financial and payment applications could enhance software verification processes, ensuring improved accuracy, security, and fault detection in financial systems.

REFERENCES CITED

[1] J. C. Westland, "The cost of errors in software development: Evidence from industry," *J. Syst. Softw.*, vol. 62, p. 1–9, May 2002.

[2] P. Ammann and J. Offutt, *Introduction to Software Testing*. USA: Cambridge University Press, 1 ed., 2008.

[3] M. M. Lehman and L. A. Belady, *Program Evolution: Processes of Software Change*. USA: Academic Press Professional, Inc., 1985.

[4] P. E. Ceruzzi, "Computing: A concise history," 2012.

[5] M. J. Harrold, "Testing: A roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, (New York, NY, USA), p. 61–72, Association for Computing Machinery, 2000.

[6] R. Sanders and D. Kelly, "The challenge of testing scientific software," 01 2008.

[7] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, pp. 507–525, 2015.

[8] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," 01 1998.

[9] R. Guderlei and J. Mayer, "Towards automatic testing of imaging software by means of random and metamorphic testing," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 17, pp. 757–781, 2007.

[10] T. Y. Chen, F.-C. Kuo, W. Ma, W. Susilo, D. Towey, J. Voas, and Z. Q. Zhou, "Metamorphic testing for cybersecurity," *Computer*, vol. 49, no. 6, pp. 48–55, 2016.

[11] F.-C. Kuo, T. Y. Chen, and W. K. Tam, "Testing embedded software by metamorphic testing: A wireless metering system case study," in *Proceedings of the 2011 IEEE 36th Conference on Local Computer Networks*, LCN '11, (USA), p. 291–294, IEEE Computer Society, 2011.

[12] K. Rahman and U. Kanewala, "Predicting metamorphic relations for matrix calculation programs," in *Proceedings of the 3rd International Workshop on Metamorphic Testing*, MET '18, pp. 10–13, ACM, 2018.

[13] K. Rahman, I. Kahanda, and U. Kanewala, "Mrpredt: Using text mining for metamorphic relation prediction," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ICSEW'20, (New York, NY, USA), p. 420–424, Association for Computing Machinery, 2020.

[14] K. Rahman and C. Izurieta, "A mapping study of security vulnerability detection approaches for web applications," in *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 491–494, 2022.

[15] K. Rahman and C. Izurieta, "An approach to testing banking software using metamorphic relations," in *2023 IEEE 24th International Conference on Information Reuse and Integration for Data Science (IRI)*, pp. 173–178, 2023.

[16] S. Segura, G. Fraser, A. B. Sánchez, and A. R. Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, pp. 805–824, 2016.

[17] R. Li, H. Liu, P.-L. Poon, D. Towey, C.-A. Sun, Z. Zheng, Z. Q. Zhou, and T. Y. Chen, "Metamorphic relation generation: State of the art and research directions," *ACM Trans. Softw. Eng. Methodol.*, Dec. 2024. Just Accepted.

[18] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen, "How effectively does metamorphic testing alleviate the oracle problem?," *IEEE Transactions on Software Engineering*, vol. 40, pp. 4–22, Jan. 2014.

[19] W. J. Cody, *Software Manual for the Elementary Functions*. Prentice-Hall Series in Computational Mathematics, USA: Prentice-Hall, Inc., 1980.

[20] H. Atashzar, A. Torkaman, M. Bahrololum, and M. H. Tadayon, "A survey on web application vulnerabilities and countermeasures," in *Proceedings of the 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, pp. 647–652, 2011.

[21] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*. Wiley Publishing, 3rd ed., 2011.

[22] S. Vergilio, J. Maldonado, and M. Jino, "Infeasible paths in the context of data flow based testing criteria: Identification, classification and prediction.," *J. Braz. Comp. Soc.*, vol. 12, pp. 71–86, 02 2006.

[23] V. H. S. Durelli, R. S. Durelli, S. S. O. Borges, A. T. Endo, M. M. Eler, D. R. C. Dias, and M. de Paiva Guimarães, "Machine learning applied to software testing: A systematic mapping study," *IEEE Transactions on Reliability*, vol. 68, pp. 1189–1212, 2019.

[24] W. E. Howden, "Theoretical and empirical studies of program testing," in *Proceedings of the 3rd International Conference on Software Engineering*, ICSE '78, p. 305–311, IEEE Press, 1978.

[25] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011.

[26] P. Louridas and C. Ebert, "Machine learning," *IEEE Software*, vol. 33, pp. 110–115, 09 2016.

[27] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. The MIT Press, 2012.

[28] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. USA: Cambridge University Press, 2014.

[29] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines: And Other Kernel-Based Learning Methods*. USA: Cambridge University Press, 1999.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[31] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017.

[32] H. Liu, X. Liu, and T. Y. Chen, "A new method for constructing metamorphic relations," in *2012 12th International Conference on Quality Software*, pp. 59–68, Aug 2012.

[33] G. Dong, B. Xu, L. Chen, C. Nie, and L. Wang, "Case studies on testing with compositional metamorphic relations," vol. 24, pp. 437–443, 12 2008.

[34] J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, and H. Mei, "Search-based inference of polynomial metamorphic relations," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, (New York, NY, USA), pp. 701–712, ACM, 2014.

[35] F. H. Su, J. Bell, C. Murphy, and G. Kaiser, "Dynamic inference of likely metamorphic properties to support differential testing," in *Proceedings of the 10th International Workshop on Automation of Software Test*, AST '15, (Piscataway, NJ, USA), pp. 55–59, IEEE Press, 2015.

[36] T. Y. Chen, P. L. Poon, S. F. Tang, and T. H. Tse, "Dessert: a divide-and-conquer methodology for identifying categories, choices, and choice relations for test case generation," *IEEE Transactions on Software Engineering*, vol. 38, pp. 794–809, July 2012.

[37] T. Y. Chen, P. L. Poon, and X. Xie, "Metric: Metamorphic relation identification based on the category-choice framework," *The Journal of systems and software*, vol. 116, pp. 177–190, 2016.

[38] U. Kanewala, J. M. Bieman, and A. Ben-Hur, "Predicting metamorphic relations for testing scientific software: A machine learning approach using graph kernels," *Softw. Test. Verif. Reliab.*, vol. 26, p. 245–269, May 2016.

[39] U. Kanewala and J. M. Bieman, "Using machine learning techniques to detect metamorphic relations for programs without test oracles," in *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 1–10, Nov 2013.

[40] B. Hardin and U. Kanewala, "Using semi-supervised learning for predicting metamorphic relations," in *Proceedings of the 3rd International Workshop on Metamorphic Testing*, MET '18, (New York, NY, USA), p. 14–17, Association for Computing Machinery, 2018.

[41] F. U. Rehman and M. Srinivasan, "Metamorphic testing for machine learning: Applicability, challenges, and research opportunities," in *2023 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pp. 34–39, 2023.

[42] F. Ur Rehman and C. Izurieta, "Mt4uml: Metamorphic testing for unsupervised machine learning," in *2022 9th Swiss Conference on Data Science (SDS)*, pp. 26–32, 2022.

[43] F. U. Rehman and C. Izurieta, "An approach for verifying and validating clustering based anomaly detection systems using metamorphic testing," in *2022 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pp. 12–18, 2022.

[44] F. u. Rehman and C. Izurieta, "Statistical metamorphic testing of neural network based intrusion detection systems," in *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, pp. 20–26, 2021.

[45] F. u. Rehman and C. Izurieta, "A hybridized approach for testing neural network based intrusion detection systems," in *2021 International Conference on Smart Applications, Communications and Networking (SmartNets)*, pp. 1–8, 2021.

[46] Z. Wadhams, C. Izurieta, and A. M. Reinhold, "Barriers to using static application security testing (sast) tools: A literature review," in *Proceedings of the Workshop on Human-Centric Software Engineering & Cyber Security (HCSE&CS 2024)*, (Sacramento, CA, USA), Nov. 2024.

[47] V. Somi, "A comparative analysis and benchmarking of dynamic application security testing (dast) tools," *Journal of Engineering and Applied Sciences Technology*, vol. 1, no. 6, pp. 1–6, 2024.

[48] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, "Security testing: A survey," *Advances in Computers*, vol. 101, pp. 1–51, 2016.

[49] A. M. Reinhold, T. Weber, C. Lemak, D. Reimanis, and C. Izurieta, "New version, new answer: Investigating cybersecurity static-analysis tool findings," in *Proceedings of the IEEE International Conference on Cybersecurity and Resilience (CSR)*, (Venice, Italy), IEEE, July 2023.

[50] F. E. Allen, "Control flow analysis," in *Proceedings of a Symposium on Compiler Optimization*, (New York, NY, USA), pp. 1–19, ACM, 1970.

[51] R. Vallee-Rai and L. J. Hendren, "Jimple: Simplifying java bytecode for analyses and transformations," 1998.

[52] T. Hofmann, B. Schölkopf, and A. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, 01 2007.

[53] R. Sharan and T. Ideker, "Modeling cellular machinery through biological network comparison," *Nature Biotechnology*, vol. 24, pp. 427–433, 2006.

[54] H. Hosoya, *Introduction to Graph Theory*, pp. 1–36. Dordrecht: Springer Netherlands, 1994.

[55] R. Kumar, J. Novak, and A. Tomkins, "Structure and evolution of online social networks," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, (New York, NY, USA), p. 611–617, Association for Computing Machinery, 2006.

[56] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," vol. 129-143, pp. 129–143, 01 2003.

[57] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," vol. 129-143, pp. 129–143, 01 2003.

[58] K. Borgwardt, underlineCS, S. Schönauer, S. Vishwanathan, A. Smola, and H. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21 Suppl 1, pp. i47–56, 01 2005.

[59] G. Wiederschain, "Data mining techniques for the life sciences (olivero carugo and frank eisenhaber (eds.), in series "springer protocols. methods in molecular biology", vol. 609, humana press, 2010, 407 p., 110)," *Biochemistry (Moscow)*, vol. 76, 04 2011.

[60] D. J. Hand and C. Anagnostopoulos, "When is the area under the receiver operating characteristic curve an appropriate measure of classifier performance?," *Pattern Recognition Letters*, vol. 34, no. 5, pp. 492–495, 2013.

[61] Q. Tao, W. Wu, C. Zhao, and W. Shen, "An automatic testing approach for compiler based on metamorphic testing technique," in *2010 Asia Pacific Software Engineering Conference*, pp. 270–279, IEEE, 2010.

[62] Z. Q. Zhou, S. Xiang, and T. Y. Chen, "Metamorphic testing for software quality assessment: A study of search engines," *IEEE Transactions on Software Engineering*, vol. 42, no. 3, pp. 264–284, 2015.

[63] Z. Q. Zhou, L. Sun, T. Y. Chen, and D. Towey, "Metamorphic relations for enhancing system understanding and use," *IEEE Transactions on Software Engineering*, 2018.

[64] M. W. Berry, *Survey of Text Mining*. Berlin, Heidelberg: Springer-Verlag, 2003.

[65] Wikipedia contributors, "Javadoc — Wikipedia, the free encyclopedia," 2019. [Online; accessed 10-September-2019].

[66] F. A. Shah and D. Pfahl, "Evaluating and improving software quality using text analysis techniques - a mapping study," 2016.

[67] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge university press, 2008.

[68] P. Kaviani and S. Dhotre, "Short survey on naive bayes algorithm," *International Journal of Advance Research in Computer Science and Management*, vol. 04, 11 2017.

[69] T. Evgeniou and M. Pontil, "Support vector machines: Theory and applications," vol. 2049, pp. 249–257, 01 2001.

[70] M. Rogati and Y. Yang, "High-performing feature selection for text classification," in *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, CIKM '02, (New York, NY, USA), p. 659–661, Association for Computing Machinery, 2002.

[71] S. Hassan, M. Rafi, and M. S. Shaikh, "Comparing svm and naïve bayes classifiers for text categorization with wikitology as knowledge enrichment," in *2011 IEEE 14th International Multitopic Conference*, pp. 31–34, Dec 2011.

[72] X. Li and B. Liu, "Learning to classify texts using positive and unlabeled data.," pp. 587–594, 01 2003.

[73] S. Alsaleem, "Automated arabic text categorization using svm and nb.," *Int. Arab J. e-Technol.*, vol. 2, pp. 124–128, 01 2011.

[74] R. Kohavi, "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*, pp. 1137–1143, 1995.

[75] S. Rafique, M. Humayun, B. Hamid, A. Abbas, M. Akhtar, and K. Iqbal, "Web application security vulnerabilities detection approaches: A systematic mapping study," in *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 1–6, IEEE, 2015.

[76] N. Kaur and P. Kaur, "Input validation vulnerabilities in web applications," *Journal of Software Engineering*, vol. 8, no. 3, pp. 116–126, 2014.

[77] A. Austin, C. Holmgreen, and L. Williams, "A comparison of the efficiency and effectiveness of vulnerability discovery techniques," *Information and Software Technology*, vol. 55, no. 7, pp. 1279–1288, 2013.

[78] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, (Swindon, GBR), p. 68–77, BCS Learning & Development Ltd., 2008.

[79] M. H. Alalfi, J. R. Cordy, and T. R. Dean, "Modelling methods for web application verification and testing: state of the art," *Software Testing, Verification and Reliability*, vol. 19, pp. 265–296, 2009.

[80] B. Marin, T. Vos, G. Giachetti, A. Baars, and P. Tonella, "Towards testing future web applications," in *Proceedings of the 5th International Conference on Research Challenges in Information Science (RCIS)*, pp. 1–12, IEEE, 2011.

[81] V. Garousi, A. Mesbah, A. Betin-Can, and S. Mirshokraie, "A systematic mapping study of web application testing," *Information and Software Technology*, vol. 55, no. 8, pp. 1374–1396, 2013.

[82] X. Li and Y. Xue, "A survey on server-side approaches to securing web applications," *ACM Computing Surveys*, vol. 46, March 2014.

[83] G. Deepa and P. S. Thilagam, "Securing web applications from injection and logic vulnerabilities: Approaches and challenges," *Information and Software Technology*, vol. 74, pp. 160–180, June 2016.

[84] J. Chang, K. K. Venkatasubramanian, A. G. West, and I. Lee, "Analyzing and defending against web-based malware," *ACM Computing Surveys*, vol. 45, pp. 1–35, August 2013.

[85] S. Gupta and B. B. Gupta, "Detection, avoidance, and attack pattern mechanisms in modern web application vulnerabilities: Present and future challenges," *International Journal of Cloud Applications and Computing*, vol. 7, no. 3, pp. 1–43, 2017.

[86] L. K. Seng, N. Ithnin, and S. Z. M. Said, "The approaches to quantify web application security scanner quality: A review," *International Journal of Advanced Computer Research*, vol. 8, no. 38, 2018.

[87] "Guidelines for conducting systematic mapping studies in software engineering," *Inf. Softw. Technol.*, vol. 64, p. 1–18, aug 2015.

[88] S. Keele *et al.*, "Guidelines for performing systematic literature reviews in software engineering," 2007.

[89] K. Rahman, "Paper_selection." Excel spreadsheet. [Revised June. 2022].

[90] P. X. Mai, F. Pastore, A. Goknil, and L. Briand, "Metamorphic security testing for web systems," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pp. 186–197, 2020.

[91] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in software engineering," *IEEE Transactions on Software Engineering*, vol. SE-12, no. 7, pp. 733–743, 1986.

[92] M. M. Lehman and L. A. Belady, *Program Evolution: Processes of Software Change*. USA: Academic Press Professional, Inc., 1985.

[93] F. Molu, "Software testing practices in critical financial systems transformation," in *2013 The International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)*, (Konya, Turkey), pp. 394–399, 2013.

[94] X. Xie, Z. Yang, J. Yu, and W. Zhang, "Design and implementation of bank financial business automation testing framework based on qtp," in *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*, (Changchun, China), pp. 143–147, 2016.

[95] L. Padgham, Z. Zhang, J. Thangarajah, and T. Miller, "Model-based test oracle generation for automated unit testing of agent systems," *IEEE Transactions on Software Engineering*, vol. 39, pp. 1230–1244, Sept. 2013.

[96] E. J. Weyuker, "On Testing Non-Testable Programs," *The Computer Journal*, vol. 25, pp. 465–470, 11 1982.

[97] T. Y. Chen, F.-C. Kuo, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing and beyond," in *Proc. 11th Annu. Int. Workshop Softw. Technol. Eng. Practice*, pp. 94–100, Sep. 2003.

[98] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, pp. 805–824, Sep. 2016.

[99] C. A. Gumussoy, "Usability guideline for banking software design," *Computers in Human Behavior*, vol. 62, pp. 277–285, Sep. 2016.

[100] T. Y. Chen, F. C. Kuo, Y. Liu, and A. Tang, "Metamorphic testing and testing with special values," in *Proceedings of SNPD2004*, pp. 128–134, 2004.

[101] H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque, "Pit: a practical mutation testing tool for java (demo)," in *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016)*, (New York, NY, USA), pp. 449–452, Association for Computing Machinery, 2016.

[102] X. Zhou, Y. Jin, H. Zhang, S. Li, and X. Huang, "A map of threats to validity of systematic literature reviews in software engineering," in *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, (Hamilton, New Zealand), pp. 153–160, 2016.

[103] T. Y. Chen, J. Feng, and T. H. Tse, "Metamorphic testing of programs on partial differential equations: A case study," in *Proceedings of the 26th International Computer Software and Applications Conference (COMPSAC)*, pp. 327–333, 2002.

[104] C. Aruna and R. S. R. Prasad, "Metamorphic relations to improve the test accuracy of multi precision arithmetic software applications," in *Proceedings of the International Conference on Advanced Computing, Communication and Informatics (ICACCI)*, pp. 2244–2248, Sep. 2014.

[105] E., "Type, p-type, s-type systems," n.d. Retrieved May 2, 2023, from `https://denrox.com/post/e-type-p-type-s-type-systems`.

[106] C. Izurieta and M. Prouty, "Leveraging secdevops to tackle the technical debt associated with cybersecurity attack tactics," in *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*, (Montreal, QC, Canada), pp. 33–37, 2019.

[107] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh, "Security requirements engineering: A framework for representation and analysis," *IEEE Transactions on Software Engineering*, vol. 34, pp. 133–153, Jan. 2008.

[108] M. Staats, M. W. Whalen, and M. P. Heimdahl, "Programs, tests, and oracles: The foundations of testing revisited," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*, (Waikiki, Honolulu, HI, USA), pp. 391–400, May 2011.

[109] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," *Journal in Computer Virology*, vol. 7, no. 4, pp. 247–258, 2011.

[110] D. Bruschi, L. Martignoni, and M. Monga, "Detecting self-mutating malware using control-flow graph matching," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 129–143, Springer, 2006.

[111] D. K. Chae, J. Ha, S. W. Kim, B. Kang, and E. G. Im, "Software plagiarism detection: a graph-based approach," in *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, pp. 1577–1580, ACM, 2013.

[112] X. Sun, Y. Zhongyang, Z. Xin, B. Mao, and L. Xie, "Detecting code reuse in android applications using component-based control flow graph," in *IFIP International Information Security Conference*, pp. 142–155, Springer, 2014.

[113] E. R. Gansner and S. C. North, "An open graph visualization system and its applications to software engineering," *Software: Practice and Experience*, vol. 30, no. 11, pp. 1203–1233, 2000.

[114] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, pp. 861–874, Jun. 2006.

[115] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Artificial Intelligence Review*, vol. 26, pp. 159–190, Nov. 2007.

[116] W. R. Shadish, T. D. Cook, and D. T. Campbell, *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Boston, MA, USA: Houghton Mifflin, 2002.