Contents

1 Introduction **4**
- 1.1 Motivation for the Research 4
- 1.2 Problem Statement 4
- 1.3 Goals and Objectives 4
- 1.4 Research Questions 5
- 1.5 Significance of the Research 5
- 1.6 Structure of the Thesis 6

2 Background and Related Work **7**
- 2.1 Traditional Software Testing Methods 8
- 2.2 Challenges in Modern Software Testing 9
- 2.3 Overview of Machine Learning in Software Engineering 11
- 2.4 Existing Approaches to Bug Prediction 11
- 2.5 Summary of Related Literature 11

3 Machine Learning for Software Testing **12**
- 3.1 Types of Machine Learning 12
- 3.2 Feature Extraction in Software Testing 12
- 3.3 Commonly Used Models for Bug Prediction 12
- 3.4 Evaluation Metrics 12

4 Data Collection and Preprocessing **13**
- 4.1 Sources of Data 13
- 4.2 Data Cleaning and Preprocessing Steps 13
- 4.3 Labeling Data for Supervised Learning 13
- 4.4 Challenges in Data Preparation 13

5 Proposed Methodology **14**
- 5.1 Selection of Machine Learning Models 14
- 5.2 Feature Selection and Engineering 14
- 5.3 Model Training and Hyperparameter Tuning 14
- 5.4 Experimental Setup 14

6 Implementation **15**
- 6.1 Tools and Technologies Used 15
- 6.2 Implementation of the Selected Models 15
- 6.3 Training Pipeline and Testing Framework 15

7 Results and Evaluation **16**
- 7.1 Performance of Models on the Dataset 16
- 7.2 Comparison with Traditional Testing Methods 16
- 7.3 Discussion of Findings 16

8 Discussion and Limitations **17**
- 8.1 Interpretation of Results 17
- 8.2 Limitations of the Study 17
- 8.3 Potential Improvements 17

9 Conclusion and Future Work **18**
9.1 Summary of Findings 18
9.2 Practical Implications 18
9.3 Future Research Directions 18

10 Appendices **19**
10.1 Additional Figures 19
10.2 Datasets or Code Snippets 19

List of abbreviations and symbols

AI	umelá inteligencia
API	aplikačné programové rozhranie
B2B	obchodná interakcia medzi 2 spoločnosťami
DPM	proces poskytovania a údržby dát
DSP	proces sťahovania a spracovania dát
EDI	elektronická výmena údajov
GPOS	operačný systém určený na všeobecné účely
GUI	grafické užívateľské rozhranie
IDP	inteligentné spracovanie dokumentov
ML	strojové učenie
OOP	objektovo-orientované programovanie
OS	operačný systém
RPA	automatizácia podnikových procesov
RTOS	operačný systém založený na reálnom čase
UNSPNC	kódex štandardných produktov a služieb OSN v elektronickom obchode
URL	jednotný vyhľadávač prostriedku

1 Introduction

1.1 Motivation for the Research

Software testing is an essential part of the software development process. It ensures that the software meets the required quality standards and performs as expected. However, traditional software testing methods are time-consuming and labor-intensive, especially for large-scale projects. With the increasing complexity of modern software systems and the pressure to deliver products faster, there is a need to improve the efficiency and effectiveness of software testing.

Machine learning has the potential to revolutionize software testing by automating the bug detection process and predicting potential bug locations. By analyzing historical data and identifying patterns indicative of bugs, machine learning algorithms can help testers prioritize their testing efforts and focus on areas of the software that are most likely to contain bugs. This can significantly reduce the time and resources required for testing while improving the overall quality of the software.

1.2 Problem Statement

One of the key challenges in software testing is identifying potential areas of the software that are most likely to contain bugs. This requires a deep understanding of the software codebase and the ability to predict where bugs are likely to occur. Machine learning can significantly improve the efficiency of software testing by predicting possible bug locations. This allows resources to be focused on the most vulnerable areas of the system, reducing overall testing costs and increasing its accuracy.

Another challenge in software testing is the lack of automated tools to assist testers in identifying bugs. Traditional testing methods rely on manual inspection and testing, which can be time-consuming and error-prone. Machine learning can automate the bug detection process by analyzing historical data and identifying patterns that are indicative of bugs. This can help testers prioritize their testing efforts and focus on areas of the software that are most likely to contain bugs.

1.3 Goals and Objectives

The goal of this research is to analyze modern machine learning methods applied in software testing. Specifically, the algorithms or structures will be divided according to known techniques applied in machine learning. The most practical methods will be evaluated by examining existing solutions for predicting software bugs identified from the literature.

The main objectives of this research are:

- To systematically review and analyze different machine learning approaches used in software testing, with a focus on bug prediction and detection
- To evaluate the effectiveness of various machine learning models in identifying potential software bugs
- To compare and contrast traditional testing methods with machine learning-based approaches

- To implement and evaluate selected machine learning models on real-world software testing datasets
- To develop recommendations for implementing machine learning in software testing processes
- To identify challenges and limitations in applying machine learning to software testing
- To propose future research directions and opportunities for further exploration in this field

1.4 Research Questions

To achieve the stated objectives, the following research questions will be addressed:

- What are the different machine learning techniques used in software testing, and how do they compare to traditional testing methods?
- Which machine learning models are most effective in predicting software bugs, and what are the key factors that influence their performance?
- How do machine learning-based approaches improve the efficiency and effectiveness of software testing compared to traditional methods?
- What are the practical implications of implementing machine learning in software testing, and what are the potential challenges and limitations?
- What are the future research directions and opportunities for further exploration in this field?
- How do the results of this research contribute to the existing body of knowledge in software testing and machine learning?

1.5 Significance of the Research

This research is significant for several reasons. First, it addresses a critical need in the software development industry by exploring innovative approaches to software testing that can improve the efficiency and effectiveness of the testing process. By leveraging machine learning techniques, testers can identify potential bug locations more accurately and prioritize their testing efforts accordingly.

Second, this research contributes to the growing body of knowledge in the field of software testing and machine learning. By systematically reviewing and analyzing existing literature on bug prediction and detection, this research provides valuable insights into the current state of the art and identifies opportunities for further research and development.

Finally, this research has practical implications for software development organizations looking to enhance their testing processes. By implementing machine learning-based approaches, companies can reduce the time and resources required for testing, improve the quality of their software, and deliver products faster to market. This research provides recommendations and guidelines for organizations seeking to adopt machine learning in their testing processes.

1.6 Structure of the Thesis

This thesis is organized into eight main chapters. Following this introduction, Chapter 2 provides background information and reviews related work in the field of software testing and machine learning. Chapter 3 explores the fundamental concepts of machine learning as applied to software testing. Chapter 4 details the data collection and preprocessing methodologies used in this research. Chapter 5 presents the proposed methodology for implementing machine learning in software testing. Chapter 6 describes the practical implementation of the selected models. Chapter 7 presents and analyzes the results of the experiments. Chapter 8 discusses the findings, limitations, and potential improvements. Finally, Chapter 9 concludes the thesis and suggests directions for future research.

2 Background and Related Work

Before diving into the details of machine learning in software testing, it is important to understand its context. This section will briefly introduce the reader to the background and terminology of the software testing, as well as the challenges that regularly arise in the testing.

Software testing is an essential part of the software development process. It helps to identify defects, flaws[1] or errors in the application code that must be fixed, which in turn can reduce a considerable amount of time. Proper testing also prevents any post-production flaws or maintenance expenses, which enhances customer satisfaction. From the other point of view, the main goals of testing is to ensure quality of the software but also to idenfity its correctness and completeness.[4, 7]

To test software, testers can utilize a particular format called **test case** which is a set of steps, conditions, and actions to verify the functionality of the software. It ultimately leads to uncovering issues and ensuring the software behaves as expected.[2, 6]

The flow of a test case is shown in the figure below:

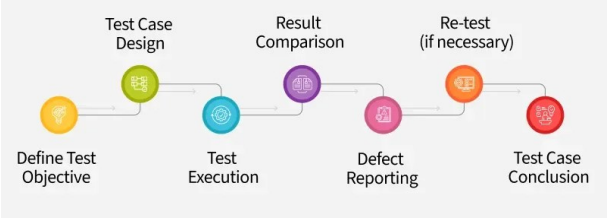

Figure 1: Flow of a test case[2]

Any software testing process should be based on the following approach:

1. **Verifying the behavior of the software** - ensuring it behaves as specified in the requirements and design documents.
2. **Detecting errors** - finding bugs or defects in the software that need to be fixed.
3. **Reflecting user needs** - assuring the software meets the needs and expectations of the end users.[7]

Testing is carried out on every stage of software life cycle but at different levels it has different objectives and range.[5]

1. **Unit Testing** - focuses on testing individual units or components of the software. It is usually performed by the developers themselves.[5]
2. **Integration Testing** - focuses on testing the interactions between different units or components of the software. It is performed after unit testing.[5]

[1] A bug or error is a manifestation of error in code.[3]

3. **System Testing** - focuses on testing the entire system as a whole (end-to-end quality) and is based on the requirement specification. It is performed after integration testing.[5]

4. **Acceptance Testing** - focuses on testing the software from the end user's perspective (software is often handed over from developers to customers). It is performed after system testing.[5]

2.1 Traditional Software Testing Methods

During testing process, certain testing cases must be identified and executed. To define test cases, testers can use various testing methods. A testing method can be thought of as the specification of strategy used in testing to select input cases and analyze the output.[5] The combination of multiple testing methods ensures higher effectiveness of the testing process. By involving all important software aspects, this structured approach ensures more thorough and complete software validation.[3]

Another advantage of combining different testing methods is that it allows testers to focus on test cases that are most likely to reveal defects. Instead of guessing, which test cases to use, testers can rely on these methods to identify scenarios where errors are most probable.[3]

To sum up, combining different testing methods allows testers to achieve better results than using only one method.[3] Currently, there several types of most commonly-used testing methods:

- **Black-box testing**
- **White-box testing**
- **Grey-box testing**

White-box testing (also known as **structural testing**[5]) is a detailed investigation of internal logic and structure of the code. It is also known as white box analysis, clear box testing or clear box analysis. This technique expects tester to have knowledge of the source code.[3][4] In some context it is also called security testing.[3] The results of this testing are evaluated based on a set of coverage criteria. These may include path coverage, branch coverage, and data-flow coverage.[5]

Black-box testing (also known as **functional testing**[5]), on the other hand, is a software testing method that examines the functionality of an application without peering into its internal structures or workings. A black box device[2] is a piece of equipment that performs its function based on the input it receives and produces an output without any knowledge of its internal workings. Black box testing is often used for validation and verification testing to ensure that the software meets the requirements specified by the end user.[3][4] The results of black-box testing can include requirement/design specifications, hand calculated values, and simulated results.[5]

Grey-box testing is a software testing technique that combines elements of both black box testing and white box testing. In grey box testing, the tester

[2]In the context of data mining, a black box device is an algorithm without any clear explanation of how it works.[3]

has limited knowledge of the internal workings of the software being tested, but they understand at least the fundamental aspects. This testing method is often applied in integration testing between modules, to ensure they interact correctly. Another advantage is quite low bias of the tester, as they do not have full access to the source code.[3][4]

2.2 Challenges in Modern Software Testing

In this section we will discuss some of the challenges faced by software testers in the modern software development environment. They will serve as a motivation for the research and provide a context for the proposed solution.

Complexity of Applications

Nowadays, software applications are becoming increasingly complex, with intricate structures, multiple layers of code and dependencies. Testers need to pay meticulous attention to detail and ensure that they understand the overall architecture of the application.[8]

Cem Karner, a well-known software testing researcher, also states that **complete testing** is impossible due to the complexity of modern software applications and vast number of inputs, states and interactions in software.[1]

This complexity can be managed by using one or more effective approaches, like **modular testing**. Basically, modular testing is a software testing technique that divides the application into smaller, more manageable modules, which are then tested individually. This approach helps to identify bugs and defects early in the development process and ensures that the application functions correctly as a whole.[8]

Lack of Documentation

One of the biggest challenges in software testing is the lack of proper documentation. Without clear and detailed documentation, testers may struggle to understand the requirements of the software and the expected behavior of the application. It is therefore developers' responsibility to provide comprehensive documentation that outlines the functionality of the software and the test cases that need to be executed.[8]

It is also important to bridge informational gaps between developers and testers thorough regular interactions, like meetings or workshops. This can help to clarify any ambiguities in the requirements and ensure that the testing process is aligned with the development goals.[8]

Evolving Requirements

During the software development process, requirements may change or evolve, leading to modifications in the codebase. This happens mainly when applying **agile methodologies**[3], where the development process is iterative and requirements are refined over time. It is thus essential for testers to be constantly

[3]Agile: `https://www.researchgate.net/publication/261017281_Agile_Methodology_Adoption_Benefits_and_Constraints`

adaptable and vigilant, ensuring that the testing process remains aligned with the changing requirements of the software.[8]

As was the case with the previous problem, regular communication between developers and testers is critical. This ensures that testers are regularly updated on any changes to the requirements and can adjust their testing strategies accordingly. Another solution is **iterative testing**, where the testing process is carried out in multiple iterations, allowing testers to adapt to changing requirements and ensure that the software meets the evolving needs of the users.[8]

Misallocation of Resouces

Software testing is a time-consuming process that requires careful planning and execution. Testers need to allocate sufficient time for each testing phase, including test case design, execution, and reporting. However, in practice, time constraints are a common challenge in software testing, with testers often facing tight deadlines and pressure to deliver results quickly. This often stems from the lack of communication between team members which ultimately results in delayed detection of bugs and postponement of project deadlines.[8, 7]

Other than time constraints, testers also face the challenge of allocating financial resources. Software testing is especially expensive, making up to 40% of the total development cost. Furthermore, it is often difficult for upper management to persuade clients about this fact.[7]

Testers also fall in into process myths, leading to misallocation of time and resources.[8, 1]

To address these challenges, testers can use **risk-based testing**, where test cases are prioritized based on their impact on the software and the likelihood of failure. This allows testers to focus on critical areas of the application and ensure that the most important test cases are executed first.[8]

Non-reproducible Bugs

One of the most frustrating challenges in software testing is dealing with non-reproducible bugs. These are bugs that occur sporadically and are difficult to reproduce consistently. Testers may spend hours or even days trying to reproduce the bug, only to find that it disappears or cannot be replicated.[8]

This challenge can be addressed by maintaining **exhaustive logs** of the testing process, including the steps taken, the inputs provided, and the expected and actual outcomes. By keeping detailed records of the testing process, testers can identify patterns in the occurrence of non-reproducible bugs and develop strategies to reproduce them consistently.[8]

Lack of Skilled Testers

c A shortage of talented and skilled testers is a common challenge in software testing. Many teams lack skilled programming testers and clear vision for project that would keep them engaged and challenged.[1] Testers need to have a deep understanding of the software development process, as well as the ability to identify bugs and defects effectively.[8]

Organizations can address this challenge by investing in training and development programs for their testers, ensuring that they have the necessary

skills and knowledge to perform their roles effectively. Testers can also benefit from continuous learning and professional development opportunities, such as certifications, workshops, and conferences.[8]

Choosing the Right Testing Tools

A lot of testing tools are available in the market, but choosing the right one can be a challenge. Testers need to evaluate the features, functionality, and cost of each tool to determine which one best meets their testing requirements.[8]

A possible solution to this challenge is to perform an extensive search and **pilot testing** of different testing tools before making a final decision. Testers can also seek recommendations from other professionals in the field and read reviews and case studies to gain insights into the effectiveness of different testing tools.[8]

The rest of the challenges of software testing are concerned with security, integration, automation, testing in different environments, and performance and load testing.[8]

2.3 Overview of Machine Learning in Software Engineering

2.4 Existing Approaches to Bug Prediction

2.5 Summary of Related Literature

3 Machine Learning for Software Testing

3.1 Types of Machine Learning

3.2 Feature Extraction in Software Testing

3.3 Commonly Used Models for Bug Prediction

3.4 Evaluation Metrics

4 Data Collection and Preprocessing

4.1 Sources of Data

4.2 Data Cleaning and Preprocessing Steps

4.3 Labeling Data for Supervised Learning

4.4 Challenges in Data Preparation

5 Proposed Methodology

5.1 Selection of Machine Learning Models

5.2 Feature Selection and Engineering

5.3 Model Training and Hyperparameter Tuning

5.4 Experimental Setup

6 Implementation

6.1 Tools and Technologies Used

6.2 Implementation of the Selected Models

6.3 Training Pipeline and Testing Framework

7 Results and Evaluation

7.1 Performance of Models on the Dataset

7.2 Comparison with Traditional Testing Methods

7.3 Discussion of Findings

8 Discussion and Limitations

8.1 Interpretation of Results

8.2 Limitations of the Study

8.3 Potential Improvements

9 Conclusion and Future Work

9.1 Summary of Findings

9.2 Practical Implications

9.3 Future Research Directions

10 Appendices

10.1 Additional Figures

10.2 Datasets or Code Snippets

References

[1] Vahid Garousi, Michael Felderer, Marco Kuhrmann, Kadir Herkiloğlu, and Sigrid Eldh. Exploring the industry's challenges in software testing: An empirical study. *Journal of Software: Evolution and Process*, 32(8):e2251, 2020. e2251 JSME-18-0181.R1.

[2] GeeksForGeeks. How to write test cases – software testing. 03 2025.

[3] Irena Jovanović. Software testing methods and techniques. *The IPSI BgD transactions on internet research*, 30, 2006.

[4] Mohd Ehmer Khan and Farmeena Khan. A comparative study of white box, black box and grey box testing techniques. *International Journal of Advanced Computer Science and Applications*, 3(6), 2012.

[5] Lu Luo. Software testing techniques. *Institute for software research international Carnegie mellon university Pittsburgh, PA*, 15232(1-19):19, 2001.

[6] Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Psychology and Economics of Software Testing*, chapter 2, pages 5–18. John Wiley and Sons, Ltd, 2012.

[7] Soukaina Najihi, Sakina Elhadi, Rachida Ait Abdelouahid, and Abdelaziz Marzak. Software testing from an agile and traditional view. *Procedia Computer Science*, 203:775–782, 2022. 17th International Conference on Future Networks and Communications / 19th International Conference on Mobile Systems and Pervasive Computing / 12th International Conference on Sustainable Energy Information Technology (FNC/MobiSPC/SEIT 2022), August 9-11, 2022, Niagara Falls, Ontario, Canada.

[8] Rahnuma Tasnim. Challenges in software testing: How to handle the testing challenges. 2023.