Contents

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.[10, 14]

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.[5, 13]

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

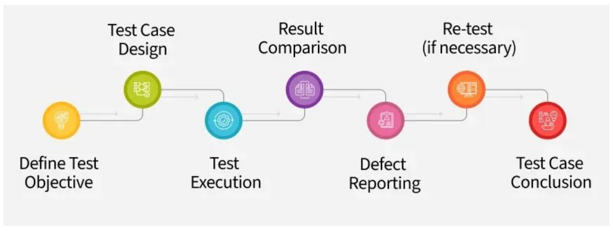

Figure 1: Flow of a test case[5]

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.[14]

Testing is carried out on every stage of software life cycle but at different levels it has different objectives and range.[12]

1. **Unit Testing** - focuses on testing individual units or components of the software. It is usually performed by the developers themselves.[12]
2. **Integration Testing** - focuses on testing the interactions between different units or components of the software. It is performed after unit testing.[12]

[1] A bug or error is a manifestation of error in code.[8]

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.[12]

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.[12]

5. **Regression Testing** - focueses on retesting the software that has been modified or extended to ensure that the changes have not introduced new defects or caused existing functionality to break.[11]

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.[12] 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.[8]

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.[8]

To sum up, combining different testing methods allows testers to achieve better results than using only one method.[8] 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**[12]) 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.[8][10] In some context it is also called security testing.[8] 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.[12]

Black-box testing (also known as **functional testing**[12]), 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.[8][10] The

[2]In the context of data mining, a black box device is an algorithm without any clear explanation of how it works.[8]

results of black-box testing can include requirement/design specifications, hand calculated values, and simulated results.[12]

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 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.[8][10]

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.[16]

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.[4]

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.[16]

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.[16]

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.[16]

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 adaptable and vigilant, ensuring that the testing process remains aligned with the changing requirements of the software.[16]

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.[16]

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.[16, 14]

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.[14]

Testers also fall in into process myths, leading to misallocation of time and resources.[16, 4]

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.[16]

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.[16]

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.[16]

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.[4] Testers need to

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

have a deep understanding of the software development process, as well as the ability to identify bugs and defects effectively.[16]

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.[16]

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.[16]

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.[16]

The rest of the challenges of software testing are concerned with security, integration, automation, testing in different environments, and performance and load testing.[16]

2.3 Overview of Machine Learning in Software Testing

In previous sections we have discussed the general aspects of sofware testing and its challenges. In this section we will bridge the gap between software testing and machine learning, which is the main focus of this thesis.

Machine learning is a subfield of artificial intelligence involves building computer programs that improve their performance on a specific task through past experience. It is based on the idea that systems can learn from data, identify patterns, and make decisions with minimal human intervention.[17][15]

2.3.1 Key Application Areas

Some of the key application areas of machine learning in software engineering include:

1. **Bug Prediction -** Software Bug Prediction is an essential process during software developement as predicting software faults earlier can improve software quality, reliability and reducing maintenance costs. It uses historical data from previous program versions to identify parts which are most likely to contain bugs.[6]

2. **Defect Classification -** Once a bug is identified, it is important to categorize it into specific types. Having done that, each of the type of bug can be assigned its own priority and impact on software quality. Classifying defects can also help us to discover hidden patterns (trends) in data and ultimately reduce their occurrence.[9]

3. **Effort Estimation -** Effort is one of the most important factors in software development and essentially represents the cost of project. At early

stages of software development, it must be estimated to devise a planned schedule and budget. ML models can be used to predict the effort required for a project based on historical data and other relevant factors.[3]

4. **Test Case Prioritization** - It is not feasible to test the whole functionality during regression testing with limited time and budget. Instead, test cases that are most likely to reveal defects should be prioritized.[11]

5. **Code Smell Prediction** – Code smells are indicators of poor design or implementation choices in source code that may not cause immediate issues but tend to increase the risk of defects and the likelihood of future code changes.[2]

6. **Security Vulnerability Detection** - Hidden flaws in software can lead to security vulnerabilities. Attackers can exploit these vulnerabilities to gain unauthorized access to systems, steal sensitive data, or disrupt services. With the recent rise of publicly available open source repositories, it is now possible to utilize data-driven approaches to discover patterns in this domain too.[7]

2.3.2 Common Techniques and Algorithms

In this section we will briefly introduce some of the most commonly used machine learning techniques and algorithms in software testing. We will discuss them in more detail in chapter 3.

We can categorize machine learning techniques into three main categories:

1. **Supervised Learning** - In supervised learning, the algorithm is trained on a labeled dataset, where the input data is paired with the corresponding output labels. The goal is to learn a mapping from inputs to outputs, allowing the model to make predictions on unseen data. Common algorithms include decision trees, support vector machines (SVM), and neural networks.[1]

2. **Unsupervised Learning** - In unsupervised learning, the algorithm is trained on an unlabeled dataset, where the input data does not have corresponding output labels. The goal is to discover patterns or structures within the data. Common algorithms include clustering techniques (e.g., k-means, hierarchical clustering) and dimensionality reduction techniques (e.g., PCA).[1]

3. **Reinforcement Learning** - In reinforcement learning, an agent learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards or penalties based on its actions, allowing it to learn optimal strategies over time. This approach is often used in scenarios where the environment is dynamic and uncertain.[1]

4. **Hybrid Learning** - To overcome the limitations of individual machine learning (ML) methods, it is essential to integrate them into a unified approach that enhances overall efficiency. Hybrid learning methods are commonly based on the practice of combining two or more approaches, a strategy widely adopted by researchers. These methods are highly adaptable and can be applied to a broad range of problem domains.[1]

The summary of the most commonly used machine learning techniques in software testing is shown in the figure below:

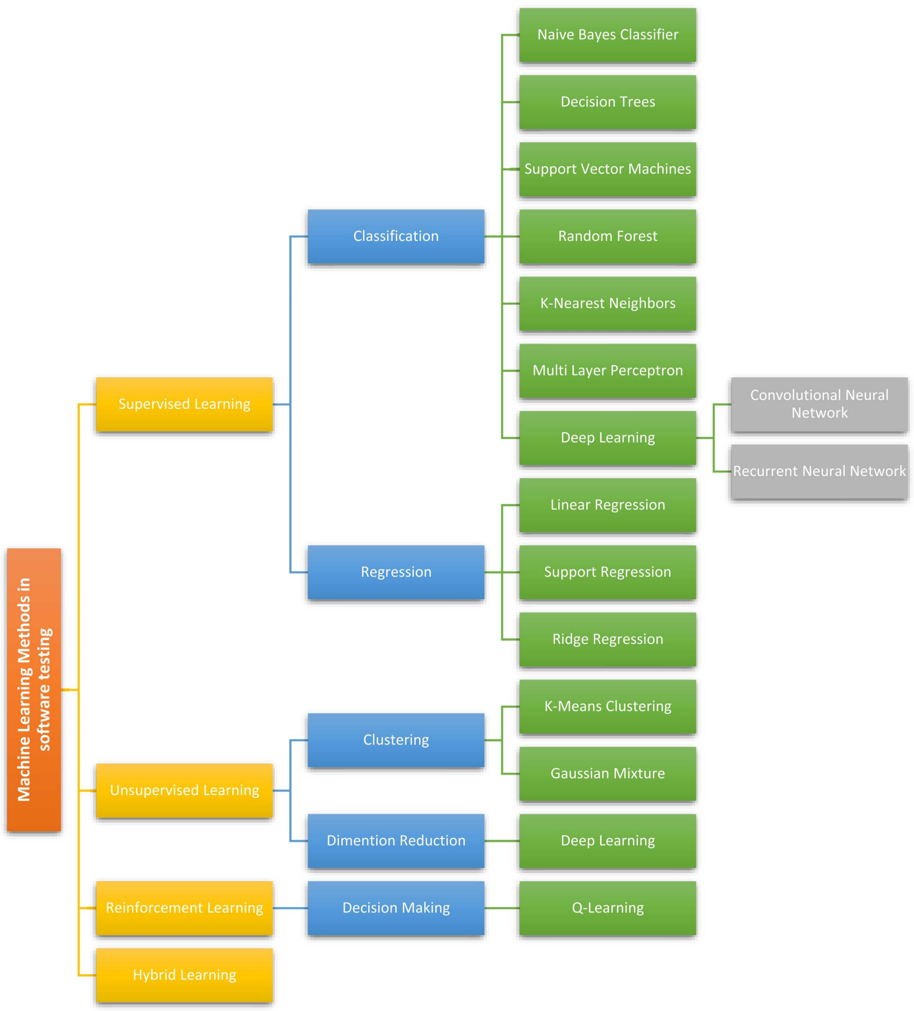

Figure 2: Commonly used machine learning techniques in software testing[1]

2.4 Existing ML Approaches to Bug Prediction

In this section we will briefly introduce some of the existing machine learning approaches to bug prediction. They have been selected based on their relevance to the topic and their potential impact on the field of software testing. They are also publicly available in the available literature.

Security vulnerabilities

The first paper proposed a novel approach to security vulnerabilities detection in C and C++ software using machine learning techniques. To make this possible, the authors costructed a large-scale dataset composed of hundreds of thousands of functions from open-source projects, labeling them using the output of a static analysis tool. Various models were trained and evaluated, including Deep neural networks, random forests, and a hybrid approach combining both. The results showed that the **hybrid model outperformed the others**, achieving the highest accuracy. Also, **models trained directly on source code performed better than those trained on build-derived data**.[7]

Test case prioritization for Regression testing

The second paper designed a novel technique for test case prioritization for manual system-level regression testing based on supervised machine learning. To achieve this, meta-data (like **execution history**) and **natural language descriptions of test cases** were used. The authors applied **SVM Rank** to rank the test cases based on their likelihood of revealing defects. The results showed that the proposed technique outperformed traditional test case prioritization techniques, such as random prioritization and expert-defined test ordering. Additionally, **natural language data** improved the overall effectiveness of failure detection.[11]

Software Bug Prediction using supervised learning classifiers

The third paper introduces a new SBP model based on supervised learning classifiers, more specifically, **Naive Bayes**, **Decision Tree** and **artificial neural networks**. The evalation showed **high accuracy in predicting future software faults**. Finally, the **solution was compared with other existing techniques and was found to be more effective in terms of accuracy and efficiency**.[6]

Summary. Overall, the reviewed studies highlight the increasing applicability and effectiveness of machine learning in various areas of software testing and maintenance, including vulnerability detection, test case prioritization, and bug prediction. Each paper applied supervised learning methods to different aspects of the software quality assurance process—ranging from using deep models to detect security flaws in source code, to prioritizing regression test cases based on metadata and natural language, to predicting future faults with classical classifiers. Across all studies, machine learning approaches consistently outperformed traditional techniques such as static heuristics, random ordering, and expert-defined prioritizations. These results underscore the potential of data-driven models to significantly improve the reliability, efficiency, and scalability of software testing practices.

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] Sedighe Ajorloo, Amirhossein Jamarani, Mehdi Kashfi, Mostafa Haghi Kashani, and Abbas Najafizadeh. A systematic review of machine learning methods in software testing. *Applied Soft Computing*, 162:111805, 2024.

[2] Muhammad Ilyas Azeem, Fabio Palomba, Lin Shi, and Qing Wang. Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. *Information and Software Technology*, 108:115–138, 2019.

[3] Bilge Baskeles, Burak Turhan, and Ayse Bener. Software effort estimation using machine learning methods. In *2007 22nd international symposium on computer and information sciences*, pages 1–6, 2007.

[4] 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.

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

[6] Awni Hammouri, Mustafa Hammad, Mohammad Alnabhan, and Fatima Alsarayrah. Software bug prediction using machine learning approach. *International journal of advanced computer science and applications*, 9(2), 2018.

[7] Jacob A. Harer, Louis Y. Kim, Rebecca L. Russell, Onur Ozdemir, Leonard R. Kosta, Akshay Rangamani, Lei H. Hamilton, Gabriel I. Centeno, Jonathan R. Key, Paul M. Ellingwood, Erik Antelman, Alan Mackay, Marc W. McConley, Jeffrey M. Opper, Peter Chin, and Tomo Lazovich. Automated software vulnerability detection with machine learning, 2018.

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

[9] Stewart Kaplan. Understanding how defects are classified in software testing [boost your qa skills]. Website, `https://enjoymachinelearning.com/blog/how-defects-are-classified-in-software-testing/`, 12 2024.

[10] 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.

[11] Remo Lachmann, Sandro Schulze, Manuel Nieke, Christoph Seidl, and Ina Schaefer. System-level test case prioritization using machine learning. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 361–368, 2016.

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

[13] 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.

[14] 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.

[15] Coursesa Stuff. What is machine learning? definition, types, and examples. Website, `https://www.coursera.org/articles/what-is-machine-learning?msockid=134a9c53edef6543091f88dbecad645c`, 02 2025.

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

[17] Du Zhang and Jeffrey J.P. Tsai. Machine learning and software engineering. *Software Quality Journal*, 11(2):87–119, June 2003.