AN INTELLIGENT APPROACH TO CODE-DRIVEN TEST EXECUTION

Rohit Khankhoje

Independent Researcher, Avon, Indiana, USA

ABSTRACT

In the constantly evolving world of software development, it is crucial to have effective testing methodologies in order to ensure the strength and reliability of applications. This scholarly article presents a new and intelligent approach to test execution that is driven by code and utilizes machine learning to greatly improve adaptability and accuracy in testing processes. Traditional testing methods often struggle to handle changes in code, resulting in less than optimal test execution. Our proposed method utilizes machine learning techniques to predict the impact of code modifications on test results, allowing for a more precise test execution strategy. We have demonstrated significant improvements in test execution efficiency, reducing unnecessary tests and speeding up feedback cycles. The following discussion examines these findings, addresses potential limitations, and suggests future areas for improvement and expansion. Notably, our methodology explains how Git commits aid in updating features, and how the machine learning model predicts the updated feature names. This predicted feature name is then integrated into Behavior-Driven Development (BDD) test selection and execution using standard BDD frameworks. By seamlessly incorporating machine learning into the testing process, developers can achieve greater precision and effectiveness, making significant progress in overcoming challenges posed by changes in code in modern development environments.

KEYWORDS

Test Automaton, Machine learning, Software testing, Automation Framework,Intelligent Test Strategy

1. BACKGROUND

The conventional model of executing tests driven by code struggles to keep up with the iterative nature of modern software development. As developers frequently modify code to implement new features or address issues, the need for an intelligent paradigm for testing becomes evident. Traditional testing often involves executing an exhaustive set of tests, which leads to redundancy and prolonged cycles for receiving feedback[1]. This research aims to revolutionize this process by integrating machine learning, allowing for a more intelligent and targeted selection of tests based on code changes.

1.1. Git Repository

The Git repository is a system for controlling versions that oversees and monitors alterations in source code throughout the process of software development. It promotes cooperation among developers by retaining a record of code modifications, thus facilitating teamwork and ensuring the integrity of projects. Developers can acquire a local copy of the repository through cloning, enabling them to make alterations and subsequently push these modifications back to the central repository. Git guarantees efficient collaboration, code integrity, and the ability to revert to previous states if necessary. It has emerged as an essential tool in contemporary software

David C. Wyld et al. (Eds): ICCSEA, NLAI, SCAI, CSIA, IBCOM, SEMIT, NECO, SPPR, MLDS -2023
pp. 109-117, 2023. CS & IT - CSCP 2023 DOI: 10.5121/csit.2023.132409

development, delivering a distributed and robust platform for managing code and controlling versions.

1.2. Machine learning model

A machine learning model is an algorithmic or statistical model that has been designed for the purpose of detecting patterns and making predictions or decisions based on data. It acquires knowledge through the process of identifying underlying patterns within training data and then utilizes this knowledge to generate predictions on new, previously unseen data. The complexity of these models can range from simple linear regressions to intricate neural networks, depending on the specific task at hand[2]. They are trained using algorithms that optimize their parameters in order to enhance their performance. Machine learning models find application in a multitude of fields, such as image recognition, natural language processing, and predictive analytics, thereby contributing to the development of automation and intelligent decision-making systems.

1.3. BDD Test Script

Behavior-Driven Development (BDD) scripts are composed in a language that is comprehensible by both technical and non-technical stakeholders. BDD places a strong emphasis on collaboration among developers, quality assurance (QA) teams, and business stakeholders in order to ensure that software development is aligned with business goals. The scripts are commonly written in a natural language format, such as Gherkin, which employs Given-When-Then statements to depict behaviors and anticipated outcomes. These scripts function as executable specifications, guiding the development process and establishing the foundation for automated tests. BDD scripts facilitate effective communication, diminish misunderstandings, and promote transparency throughout the software development lifecycle. Prominent BDD tools encompass Cucumber, SpecFlow, and Behave.

2. INTRODUCTION

In the ever-changing realm of software development, the effectiveness of testing methodologies plays a crucial role in ensuring the dependability and functionality of applications. Traditional approaches to testing often encounter difficulties in adapting to the rapid pace at which code changes occur, resulting in inefficiencies and suboptimal testing outcomes. (Pan et al., 2021) This study tackles this pressing issue by introducing an intelligent method for executing tests driven by code, utilizing machine learning to enhance adaptability and precision in the testing process. Challenges Associated with Conventional Approaches to Test Execution:

2.1. Execution of a Thorough Test Suite

Conventional testing approaches typically entail the execution of a comprehensive suite of tests for every alteration made to the code, irrespective of the specific areas affected. This exhaustive execution can result in protracted testing cycles, thereby delaying feedback to developers and impeding the agility of the development process.

2.2. Redundant Test Runs

In the absence of a mechanism to selectively execute tests based on code changes, developers may inadvertently run redundant tests that do not contribute to the validation of modified code. This redundancy consumes valuable resources and prolongs the duration of testing.

2.3. Increased Overhead in Testing

As the codebase expands, so does the size of the test suite. Executing the entire suite for each code change incurs increased testing overhead, thereby consuming additional time and computational resources. This becomes particularly burdensome in large-scale projects with extensive test coverage[8].

2.4. Delayed Feedback

The time-intensive nature of executing a comprehensive test suite can lead to delayed feedback on the impact of code changes. Quick and actionable feedback is critical for developers to promptly identify and address issues, especially in agile and continuous integration environments.

2.5. Resource Intensiveness

Running all tests indiscriminately can strain the testing infrastructure and resources, resulting in longer build and test execution times. This resource intensiveness can hinder the scalability and efficiency of the testing process.

2.6. Inefficiency in Continuous Integration

In continuous integration workflows, where frequent code changes trigger automated builds and tests, executing the entire test suite for each commit can lead to inefficiency. Rapid feedback is a fundamental aspect of continuous integration, and inefficiencies in test execution can compromise this principle.

2.7. Difficulty in Identifying Impactful Tests

Conventional test execution strategies often lack the ability to precisely identify and execute only those tests that are affected by specific code changes. This lack of precision makes it challenging for developers to concentrate testing efforts on the relevant portions of the codebase.

2.8. Limited Scalability

As the codebase expands, concerns arise regarding the scalability of conventional test execution strategies. The sheer volume of tests and the time required for execution may reach a point where maintaining an acceptable testing cadence becomes challenging[2].

Addressing these challenges necessitates the adoption of more intelligent and selective test execution strategies that align with the dynamic nature of modern software development. Introducing mechanisms to execute tests based on code changes is crucial for optimizing testing efforts and facilitating a more agile and responsive development process[3].

The issue of carrying out tests in accordance with alterations in code is situated at the point where the flexible nature of software development meets the necessity for rigorous testing procedures. As software developers progressively alter and upgrade code to introduce novel functionalities, rectify errors, or optimize performance, it becomes essential to guarantee that the existing collection of tests accurately reflects the present condition of the codebase[3]. The difficulty lies in efficiently identifying and executing solely those tests that are relevant to the recent code modifications. This research is significant in its potential to reshape the landscape of executing tests driven by code. By incorporating intelligence through machine learning, it promises to

enhance the precision, efficiency, and adaptability of testing processes, addressing a crucial gap in current methodologies.

The primary aim of this study is to develop and implement an intelligent method for executing tests driven by code, one that effectively tackles the challenges posed by dynamic code changes. Specific goals include:

A. Introducing a Machine Learning Model: Propose a machine learning model capable of intelligently analyzing code modifications and predicting their impacted features and need specific tests that feature only.
B. Optimizing Test Execution: Implementing the model to selectively execute tests based on the identified impact of code changes, thereby reducing redundancy and expediting feedback cycles.

3. LITERATURE REVIEW

The evolution of software development methodologies has highlighted the crucial role that testing processes play in ensuring the quality and reliability of applications. While traditional methods of test execution that are driven by code are fundamental, they are currently facing challenges due to the rapid evolution of codebases in modern agile and iterative development environments[4]. This review of existing literature examines the current state of code-driven test execution and identifies gaps in traditional methodologies, laying the foundation for an intelligent approach that utilizes machine learning.

Table-1 Most recent studies

Study Title	Year	Limitations
Machine Learning to Uncover Correlations Between Software Code Changes and Test Results	2017	No access to detailed data on code changes and test structure Possibility of unsatisfactory results using available data
System-Level Test Case Prioritization Using Machine Learning	2016	The paper does not specifically mention machine learning in test case execution based on code change.
A classification of code changes and test types dependencies for improving machine learning based test selection	2021	Survey conducted with limited number of companies Taxonomies in software engineering are evaluated via utility demonstration
DeepOrder: Deep Learning for Test Case Prioritization in Continuous Integration Testing.	2021	Existing techniques cannot handle large test histories. Existing techniques are optimized for limited historical test cycles.
Test Case Selection Based on Code Changes	2018	The paper does not mention anything about machine learning in test case execution based on code change.

However, a comprehensive synthesis of these studies reveals a gap in the literature regarding the development and implementation of machine learning models specifically designed for code-driven test execution. This research aims to fill this gap by proposing a novel machine learning approach that can intelligently analyze code changes and optimize the selection of tests for execution[5]. In conclusion, the literature review establishes the context by examining the limitations of traditional code-driven test execution methods and the potential benefits offered by machine learning. Building upon the existing body of knowledge, this paper introduces an intelligent approach that addresses the identified gaps, with the potential to revolutionize the landscape of code-driven test execution in contemporary software development.

4. METHODOLOGY

In the realm of software development, it is of utmost importance to furnish thorough and comprehensive information within code commit messages. The following table-2 represents the specific data that developers must input when committing code. This information will then be utilized to determine which feature tests should be executed in order to test the application. The machine learning model will utilize the same git log provided to make predictions.

Once the model acquires training data for the purpose of prediction, it utilizes the classification technique to predict the name of the feature in which the code has been altered and identify the feature that requires testing. Fig-1 explaining the overall flow of the proposed method.

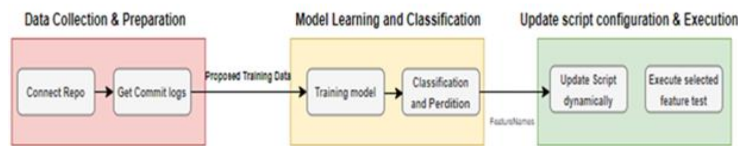

Fig-1 Solution Design

Table-2 Developer commit required details

File	Type Of Change	Feature Name	Story ID
Update README.md and src/main/java/FeatureController.java	Bug/New Feature/Performance	User Auth	XXX

Firstly, the File or paths of the files that have been modified or added in the commit must be explicitly mentioned. This information serves to provide clarity with regards to the exact code components that have been affected by the commit. For the commit message itself, it is imperative to clearly specify the scope or nature of the change. This entails indicating whether it is a bug fix, addition of a new feature, performance enhancement, or any other relevant categorization. This information aids in comprehending the purpose behind the commit.

Furthermore, it is essential to include the name or identifier of the feature associated with the code changes. This inclusion provides context regarding the user story or feature that is being addressed in the commit. If the development process employs feature or user story IDs, it is recommended to include this identifier in the commit message. This establishes a direct link between the code change and the associated feature or user story. Fig-2 the present study aims to demonstrate the correlation between the anticipated nomenclature of a characteristic and the corresponding nomenclature of a test script in order to execute a particular examination.

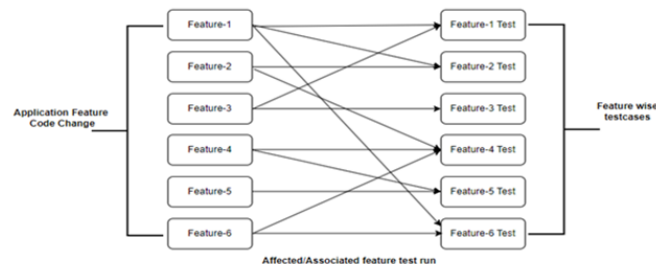

Fig-2 Feature testcase association with application feature

By incorporating this information into commit messages, the Git log transforms into a valuable source of data. This data is utilized by machine learning models for predictive analysis. a machine learning model can be trained to identify patterns in commit messages and extract pertinent information such as feature names or associated IDs. This ultimately contributes to the automation of predicting feature names based on code changes. Consequently, this enhances the efficiency and accuracy in managing and comprehending the evolution of code over time.

Integrating machine learning prognostications of characteristic appellations with automated examination frameworks, such as Cucumber BDD (Behavior-Driven Development), can substantially augment the efficacy of the examination process.

Here is an elucidation of how one can employ machine learning-predicted characteristic appellations to elect and implement automated examinations within a Cucumber BDD framework:

Tagging Cucumber Scenarios with Feature Names

After obtaining the anticipated attribute derived from a machine learning model, it is imperative to utilize the same attribute nomenclature in order to execute a Cucumber Behavior-Driven Development (BDD) framework test run. In the realm of BDD, test scenarios are commonly denoted by particular features. It is essential to establish a correlation between each scenario or feature file and a label that aligns with the prognostic characteristic appellation.

For instance, if a Cucumber scenario is related to a feature denominated "User Authentication," label it with @User Authentication.

```
Feature: User Authentication
 @User Authentication
 Scenario: User logs in with valid credentials
  Given the user is on the login page
  When the user enters valid credentials
  Then the user is logged in successfully
```

Automation Test Selection Based on Predicted Feature Name:

In order to execute BDD tests dynamically, it is necessary to establish the feature name within the runner class in the BDD framework, After the machine learning model prognosticates the feature name for a specific code alteration, utilize this prognosticate feature name to dynamically select the pertinent automation examinations.

```
@Run With (Cucumber. class)
@Cucumber Options( features = "src/test/resources/features",
   glue = "step_ definitions",
   tags = "@User Authentication" // Dynamically set based on machine learning
prediction )
public class Test Runner { }
```

Run Only Selected Scenarios:

Configure your test runner to execute solely the scenarios labeled with the prognosticate feature name. This assures that solely pertinent examinations associated with the prognosticate feature are implemented. Conduct the automated examinations using the configured Cucumber runner. This approach facilitates selective execution of examinations based on the prognostic feature name, permitting a targeted and efficient testing process.

By integrating machine learning prognostications with Cucumber BDD and labeling scenarios with feature names, one establishes a dynamic and adaptive testing framework. This approach optimizes the test suite by selectively executing solely the pertinent examinations associated with the prognostic feature, diminishing test execution time and resource utilization while upholding comprehensive test coverage for the identified feature.

5. DISCUSSION

Despite the promising findings, it is imperative to recognize the constraints associated with this approach. The accuracy of feature predictions heavily relies on the quality and diversity of historical data utilized to train the machine learning model. In situations where there is limited historical data or when codebases evolve rapidly, the predictive accuracy of the model may be compromised. To overcome this limitation, continuous refinement of the model and exploration of more advanced machine learning techniques are necessary[6].

Another aspect to consider is the necessity for robust error handling mechanisms in the event of inaccurate predictions. Further investigation should be conducted to devise strategies for handling unforeseen scenarios and false positives/negatives. There are potential areas for improvement, including:

5.1. Dynamic Test Suite Generation

The exploration of methods that dynamically generate test suites based on predicted features is essential. This allows for a more precise control over the scope of testing.

5.2. Integration with CI/CD Pipelines

Enhancing the integration with continuous integration and continuous delivery (CI/CD) pipelines is crucial. This will facilitate the seamless incorporation of the intelligent test execution approach into the development workflow[7].

5.3. Real-time Feedback Mechanisms

The implementation of real-time feedback mechanisms is necessary to update the machine learning model based on the outcomes of executed tests. This ensures continuous learning and adaptation[2].

5.4. Collaborative Testing

The exploration of collaborative testing practices, where developers actively contribute to the testing process by validating and refining predictions, is recommended.

The intelligent approach to code-driven test execution introduces a paradigm shift in testing practices, leveraging machine learning to tailor testing efforts to the ever-changing software landscape. While acknowledging limitations, the approach demonstrates substantial advantages over traditional methods and establishes the foundation for future innovations in the field of intelligent software testing.

6. CONCLUSION

In this research, we have introduced a groundbreaking and sophisticated strategy for executing tests based on code, harnessing the power of machine learning to anticipate the consequences of

alterations in code on particular functionalities. Our discoveries and contributions emphasize the importance of this strategy in revolutionizing customary testing methodologies and enhancing the efficiency of the testing procedure in dynamic software development settings.

In contrast to conventional methodologies that implement a fixed array of examinations for every alteration in code, the intelligent approach offers numerous advantages. Targeted Testing, for instance, focuses solely on the relevant examinations that pertain to the specific attributes affected by the alterations in code. This guarantees that the exertion put forth in testing is efficiently guided towards areas that have potential risks. Resource Optimization is another advantage, as the approach avoids conducting unnecessary examinations, which in turn optimizes the utilization of resources and leads to quicker building and execution of tests. Adaptability is yet another advantage of this approach, as it adjusts itself to the ever-changing codebase. This makes it particularly suitable for agile development environments where frequent changes in code occur. Enhanced Developer Productivity is yet another benefit, as it facilitates faster feedback on code changes, thus reducing waiting times for test results and enabling quicker identification and resolution of issues. Additionally, it also reduces the overall testing overhead by selectively executing tests, resulting in a more scalable and efficient testing process. To conclude, our investigation introduces a fundamental alteration in the execution of tests guided by code, thereby providing a more sophisticated and adaptable approach to testing. The results demonstrate the practicality and advantages of leveraging machine learning in the testing procedure, which in turn opens up possibilities for enhancements in the efficiency of testing, the productivity of developers, and the overall quality of software. The proposed intelligent approach not only tackles existing challenges but also establishes a foundation for future innovations in the constantly evolving domain of software development and testing.

REFERENCES

[1] Al-Sabbagh, K., Staron, M., Hebig, R., & Gomes, F. (2021). A classification of code changes and test types dependencies for improving machine learning based test selection. 10.1145/3475960.3475987

[2] Khankhoje, R. (2023). Quality Assurance in the Age of Machine Learning. Quality Assurance in the Age of Machine Learning, 13(10). 10.29322/IJSRP.13.10.2023.p14226

[3] Lachmann, R., Sandro, S., & Nieke, M. (2016). System-Level Test Case Prioritization Using Machine Learning. 10.1109/ICMLA.2016.0065

[4] Marijan, D. (2022). Comparative Study of Machine Learning Test Case Prioritization for Continuous Integration Testing. 10.48550/arxiv.2204.10899

[5] Martins, R., Rui Abreu, & Lopes, M. (2021). Supervised Learning for Test Suit Selection in Continuous Integration. 10.1109/ICSTW52544.2021.00048

[6] Mochamad, M. M., & Tetsuro, T. (2020). Code Coverage Similarity Measurement Using Machine Learning for Test Cases Minimization. 10.1109/GCCE50665.2020.9291990

[7] Negar, N. (2017). Machine Learning to Uncover Correlations Between Software Code Changes and Test Results.

[8] Pan, C., & Pradel, M. (2021). Continuous test suite failure prediction. 10.1145/3460319.3464840

[9] Pan, R., Bagherzadeh, M., & Ghaleb, T. (2021). Test Case Selection and Prioritization Using Machine Learning: A Systematic Literature Review, (arXiv: Software Engineering).

AUTHOR

I am Rohit Khankhoje, a Software Test Lead with over 15+ years of experience in software quality assurance and test automation. With a passion for ensuring the delivery of high-quality software products, I am at the forefront of harnessing cutting-edge technologies to streamline and enhance the testing process. I am dedicated to advancing the automation testing field and continue to inspire colleagues and peers.

© 2023 By AIRCC Publishing Corporation. This article is published under the Creative Commons Attribution (CC BY) license.