KARYA ILHAM PUBLISHING
CONNECTING MINDS, BRIDGING IDEAS, INSPIRING READERS

Journal of Advanced Research in Computing and Applications

Journal homepage:
https://karyailham.com.my/index.php/arca
2462-1927

Continuous Integration and Continuous Delivery Based on ARM Cortex-M4 Embedded Software Automation Testing

Yingbei Niu[1], Soo See Chai[1,*], Kok Luong Goh[2], Kim On Chin[3], Emily Sing Kiang Siew[2]

[1] Faculty of Computer Science and Information Technology Universiti Malaysia Sarawak (UNIMAS) 94300 Kota Samarahan Sarawak, Malaysia
[2] Faculty of Computing and Software Engineering, i-CATS University College, Jalan Stampin Timur, 93350 Kuching, Sarawak, Malaysia
[3] Faculty of Computing & Informatics, University Malaysia Sabah, Kota Kinabalu, 88400, Malaysia

ARTICLE INFO

Article history:
Received 15 July 2025
Received in revised form 20 August 2025
Accepted 26 August 2025
Available online 4 September 2025

Keywords:
Embedded software; automated testing; continuous integration; continuous delivery

ABSTRACT

This paper explores how automated testing can be effectively integrated into the continuous integration and continuous delivery (CI/CD) process to enhance development efficiency and software quality in embedded systems. Despite the increasing complexity of embedded software, traditional testing methods often struggle with long development cycles and poor cross-team collaboration. To address these challenges, this study proposes a novel strategy that incorporates automated GPU-based testing pipelines into CI/CD workflows. The approach involves designing and embedding automated testing modules within existing build and deployment stages using standardized testing frameworks and monitoring tools.The proposed strategy was validated through real-world embedded system projects. Experimental results demonstrated significant reductions in testing time, improved software reliability, and enhanced team coordination. Furthermore, example validations confirmed the consistency and effectiveness of the approach across different system architectures. The discussion analyzes the broader impact of automated testing within CI/CD pipelines, highlighting both the performance benefits and implementation challenges. This research contributes to the ongoing innovation in embedded systems development by offering a practical solution to common testing bottlenecks. It encourages the wider adoption of automated testing to meet dynamic market demands, improve software quality, and increase overall customer satisfaction. Future work will explore further optimization of feedback mechanisms and integration tools to support more scalable and efficient testing architectures.

1. Introduction

In today's software development field, continuous integration (Continuous Integration, CI) and continuous delivery (Continuous Delivery, CD) have become indispensable practices in the development of high-quality and efficient software. These two concepts represent a revolutionary change in the software development process, by automating development, testing, and deployment, enabling faster iteration cycles, higher quality standards, and more flexible software delivery.

* *Corresponding author.*
E-mail address: sschai@unimas.my

https://doi.org/10.37934/arca.39.1.146163

With the extensive application of embedded system in modern life, the development of embedded software is also facing higher and higher requirements. Embedded systems must not only have a high degree of reliability and security, but also meet the evolving market needs. In this context, it becomes critical to introduce the concept of continuous integration and continuous delivery into embedded software development. However, the special nature of embedded systems and the strict constraints make implementing CI / CD even more complicated and challenging.

The purpose of this paper is to investigate how automated testing can be integrated into continuous integration and continuous delivery processes of embedded systems to achieve faster development cycles and higher quality standards. This is not only important for embedded software developers and teams, but also has a profound impact on the reliability and security of embedded systems.

A series of key issues arise in the CI / CD process of introducing automated testing into embedded systems; How to select the automated test tools and frameworks suitable for embedded systems? How are test cases effectively designed and managed to meet the testing requirements of embedded systems? How do you build a reliable, continuous integration environment and ensure automated testing and validation for each code submission? How to monitor and analyze the results of automated tests, and find and solve problems in time? How to ensure the stability and security of the embedded system during continuous delivery?

Through in-depth study and exploration of these issues, this study aims to provide practical guidelines and best practices for the embedded software development field to help development teams better apply the concept of continuous integration and continuous delivery and improve development efficiency and software quality.

2. Methodology

2.1 Evolution of Embedded System Testing

The testing of embedded systems has always been an important issue in the field of software engineering [1-3]. With the continuous development of technology, embedded system testing has also undergone many evolution. Early embedded system testing was mainly focused on functional testing [4-6], namely, ensuring that the software functions properly under certain conditions. However, with the increasing complexity of embedded systems, relying solely on functional testing is no longer sufficient to meet the quality and reliability requirements. Therefore, more detailed test methods such as performance tests [7-9], safety tests and reliability tests [10,11] are gradually introduced. Recently, the concept of continuous integration and continuous delivery has begun to be widely used in embedded system testing to accelerate the development cycle and improve software quality.

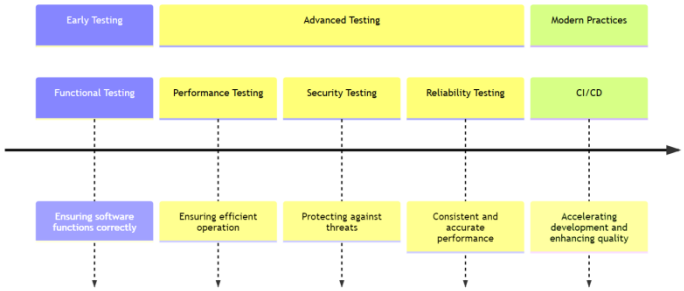

Fig. 1. Evolution of the embedded system tests

Section 1: Early Testing

In the initial phase of embedded system testing, functional testing served as the primary focus. This stage was dedicated to verifying whether the software functioned correctly under specific, predefined conditions. The primary goal was to validate that the embedded system could accurately perform the tasks for which it was designed [12-14]. By ensuring the basic operational correctness, functional testing laid the essential foundation for more advanced testing methods.

Section 2: Advanced Testing

As embedded systems increased in complexity, more rigorous testing approaches became necessary. Performance testing emerged to ensure that systems operated efficiently and within acceptable performance parameters [15-17]. This type of testing focuses on validating the system's speed, responsiveness, and stability when subjected to particular workloads. Besides that, security testing gained significant attention as protecting embedded systems from potential threats and vulnerabilities became crucial [18,19]. Security testing involves identifying and addressing vulnerabilities, safeguarding data, and ensuring compliance with relevant security regulations. In addition, reliability testing was introduced to verify that the system could perform consistently and accurately over a specified period. This testing aims to uncover hidden issues that may affect the system's long-term durability and dependability. Together, these advanced testing practices addressed the growing demands for high performance, security, and reliability in embedded systems.

Section 3: Modern Practices

In recent years, Continuous Integration and Continuous Delivery (CI/CD) practices have become widely adopted in embedded system testing [20]. CI/CD integrates automated testing processes into the development lifecycle, enabling rapid feedback and facilitating more frequent software releases. Therefore, this approach accelerates development cycles while significantly improving software quality through continuous validation.

The progression of testing methodologies for embedded systems is visually represented by the timeline, which reflects how testing practices have evolved in response to increasing system complexity and higher performance expectations. Each phase introduces new methodologies that either build upon or resolve the limitations of the previous stages. Thus, embedded system testing has developed into a more comprehensive and adaptive discipline, capable of addressing the dynamic and stringent requirements of modern embedded applications.

2.2 Principles and Advantages of Continuous Integration and Continuous Delivery

Continuous integration (CI) and continuous delivery (CD) are the core principles in modern software development [21]. CI means frequent integration of code into a shared repository and automated testing to ensure that the new code does not disrupt existing functionality. CD further extends the concept of CI to the automated deployment and delivery phase, enabling faster software delivery to end-users.

The adoption of continuous integration and continuous delivery (CI/CD) brings several significant advantages to embedded system development. One key benefit is the ability to achieve a faster development cycle. By automating testing and deployment processes, CI/CD minimises manual intervention and greatly improves development efficiency. This acceleration enables teams to

release updates and new features more frequently, which is essential in fast-paced development environments.

In addition, CI/CD contributes to maintaining higher quality standards. Automated testing enhances test coverage, reduces the likelihood of human error, and ensures the stability and reliability of the software. Furthermore, this approach supports continuous validation throughout the development process, which helps to identify defects at an early stage.

Another important advantage is the improved ability to detect and resolve problems quickly. The CI/CD process provides real-time feedback, allowing issues to be rapidly identified and addressed. This immediate visibility of defects helps prevent the accumulation of unresolved problems and significantly shortens the time required for troubleshooting and repair.

Besides that, CI/CD fosters better team collaboration. It encourages developers to integrate and deliver code frequently, promoting closer cooperation and more effective communication among team members. This collaborative environment supports smoother workflows and enhances overall project coordination. Therefore, the integration of CI/CD not only improves technical outcomes but also strengthens team dynamics and development efficiency.

2.3 Role of Automated Testing in Software Development

Automated testing plays an important role in software development [22]. It can automatically perform the test cases, generate detailed test reports, and improve the efficiency and consistency of the tests. Automated testing can be applied to different levels of testing, including unit, integration, functional, and performance testing. It can not only detect and report problems, but also speed up the process of identifying and solving problems.

In the context of continuous integration and continuous delivery, automated testing becomes particularly important. It allows to automatically run tests after each code submission, quickly find problems and notify the development team in a timely manner. This rapid feedback helps to reduce the accumulation of problems, thereby improving the quality and maintainability of the software.

Through these literature reviews, we can see that embedded system testing has evolved into a more complex and automated process, with continuous integration and continuous delivery becoming an effective means to accelerate the development cycle and improve software quality, in which automated testing plays a key role. This study aims to further explore how automated testing can be integrated into the CI / CD process of embedded systems to achieve more efficient development and higher quality software delivery.

3. Methods and Strategies

A structured development process that emphasizes the importance of automation, continuous integration and deployment, and quality of code. Such processes are designed to ensure code robustness, security, and efficiency. Figure 1 is the automated test business scenario.

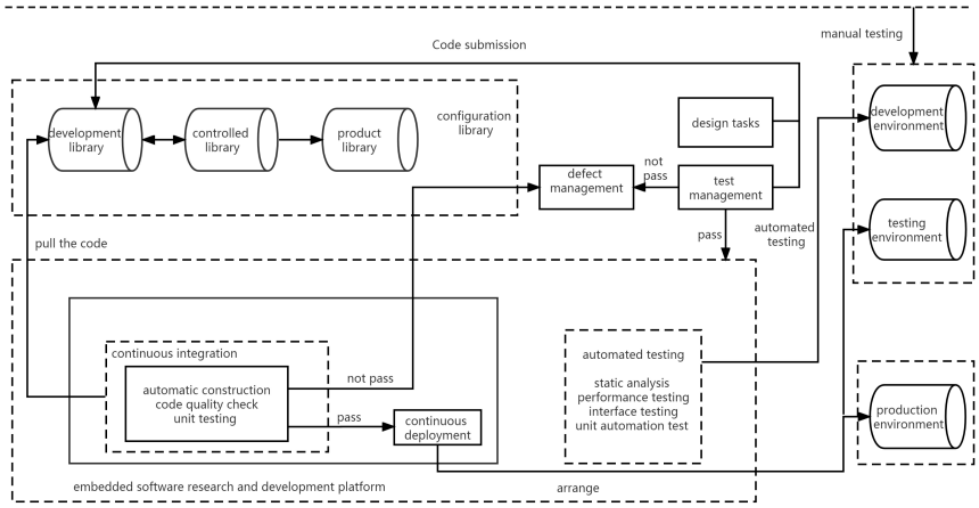

Fig. 2. Automatic test business scenario diagram

The diagram illustrates a typical business scenario in embedded software development. The development library serves as the workspace where developers carry out coding activities. Upon completion, the developed code is submitted to a central library through the code submission process. This step ensures that all code changes are systematically collected for further processing.

Following submission, the code undergoes review. Once it is deemed reliable and meets predefined quality standards, it is moved into the controlled library. This controlled repository functions as a stable version for further verification. Subsequently, the validated code is transferred to the product library, where it becomes ready for formal release or deployment.

The configuration library plays a supporting role by storing configuration profiles and related settings essential for the development and deployment processes. Alongside development, design tasks–such as interface design and functional planning–are systematically undertaken to guide and structure the software's architecture [23].

Defect management is integral to the process, providing a systematic approach to track, analyse, and resolve any defects identified during code review or testing. This defect tracking ensures that issues are promptly addressed and do not propagate to later stages.

A fully automated testing framework is implemented, encompassing static analysis, performance testing, interface testing, and unit automation testing. These automated processes significantly contribute to the reliability and robustness of the software [24]. In parallel, manual testing remains necessary for certain cases that cannot be fully automated or require human judgement [25].

The workflow incorporates continuous integration (CI), an automated pipeline that includes code compilation, quality checks, and unit testing. Codes that fail these stages are blocked from progressing, thus maintaining high standards at every step. Upon passing the CI process, the software enters continuous deployment (CD), which automatically deploys the code to the appropriate environment without manual intervention.

The development process transitions through multiple environments. The development environment is where the initial coding and basic testing occur. The test environment supports comprehensive and often automated testing, ensuring that all functional and performance aspects are validated. Finally, the production environment hosts the deployed software for end-user access and operational use.

An embedded software development platform underpins the entire process, integrating the necessary tools and technologies to facilitate streamlined development, testing, and deployment.

The CD deploys the integrated code to the real operating environment (production environment) on the basis of continuous integration. After completing unit testing, you can deploy code to more testing in the Staging environment connected to the database. If the code is fine, you can continue to manually deploy to the production environment. The figure below reflects the working mode of CI / CD.

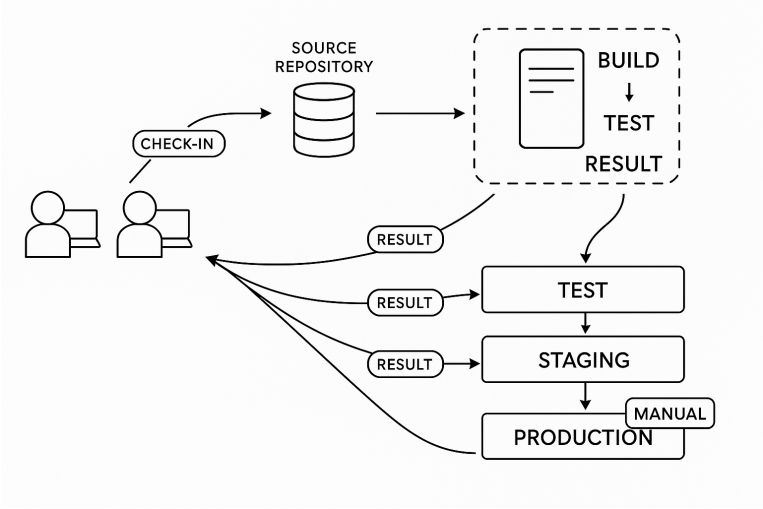

Fig. 3. The CI / CD working mode

The diagram presents the process of Continuous Integration (CI) and Continuous Deployment (CD) in detail. It systematically illustrates each key step from code development to production deployment.

Firstly, the developer is represented by two icons featuring "10101", symbolising programmers who write or modify code within their local development environment. Upon completing or updating the code, developers proceed to check-in, a process where the newly written or modified code is submitted to a central code repository. This check-in operation is essential for tracking all code changes and maintaining version control.

The source repository typically refers to a system such as Git or SVN, which centrally manages all submitted code. Once new code enters the repository, the CI server automatically initiates the continuous integration workflow. The server begins with the build phase, where the code is compiled or transformed into executable formats or deployable packages.

Following the build process, the CI server performs automated testing to verify that the recent code changes do not introduce new defects or disrupt existing functionality. Upon completion of these tests, the result is generated, indicating whether the build has passed and detailing any issues encountered. These results are promptly fed back to the developer for immediate attention, allowing either rapid bug fixes or continued development.

The deployment process then proceeds through three distinct stages: Test, Staging, and Production. In the Test phase, the software is deployed to a test environment that closely simulates real-world operating conditions. Next, the Staging phase serves as a pre-production environment where the software undergoes further validation to confirm its readiness. Finally, in the Production phase, the validated software is released to the live environment for use by end users.

Importantly, a manual step is typically required before the software transitions from staging to production. This manual review or audit is critical to ensure that all testing criteria have been met and that the software is fully prepared for deployment in the production environment.

Overall, the diagram comprehensively demonstrates the seamless transition from code development to production release. It underscores the pivotal role of automated testing and deployment in ensuring software quality and accelerating delivery cycles. Additionally, it highlights the necessity of manual validation at critical checkpoints to safeguard the reliability of the final product.

3.1 Method to Integrate Automated Testing into the CI / CD Process

Integrating automated testing into the CI / CD process requires a clear set of steps and strategies:

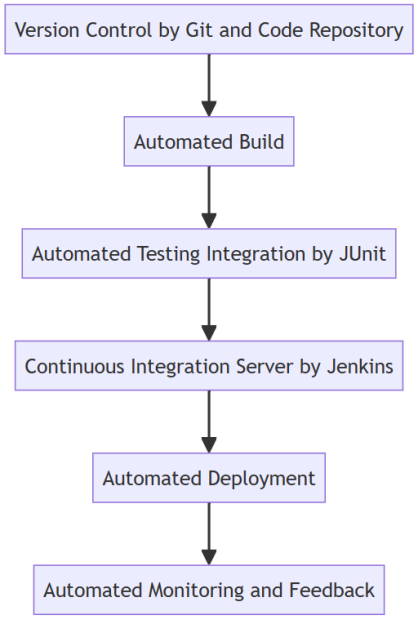

Fig 4. CI/CD flow chart

Version Control and Code Repository Management is a fundamental aspect of modern software engineering. Utilising version control tools such as Git ensures that all source code is securely stored within a central repository. This setup allows team members to collaborate efficiently, manage code branches, and track changes with precision. The central repository facilitates seamless collaboration and maintains a complete history of code evolution.

Automated Build processes are essential for maintaining consistency and reducing human error. The system is configured to automatically trigger builds whenever new code is submitted to the repository. This build process typically involves code compilation, packaging, and initial deployment steps. Automation at this stage ensures that the software is consistently built under controlled, repeatable conditions.

Automated Test Integration is a critical part of the continuous delivery pipeline. Various types of automated tests–such as unit tests, integration tests, and functional tests–are integrated directly into the build process [26]. By employing appropriate testing frameworks and tools, these tests can be executed automatically with each build, ensuring that new code does not compromise existing functionality. Detailed test reports are automatically generated to provide rapid feedback on software quality.

A Continuous Integration (CI) Server is deployed to orchestrate these automated activities. The CI server constantly monitors the version control repository for changes. Upon detecting new code

submissions, it automatically triggers the build and test processes. Widely used CI servers, such as Jenkins, play a pivotal role in enabling reliable, repeatable integration workflows.

Automated Deployment further enhances the efficiency of the software delivery process. Once the code passes all automated testing phases, the system automatically deploys the tested software to the appropriate environments. This may include the development, testing, and production environments [27]. Automated deployment minimises manual intervention and reduces the likelihood of configuration errors.

Finally, Automated Monitoring and Feedback mechanisms are established to maintain high system observability. Continuous monitoring tools and alerting systems are integrated to promptly notify the team of potential issues. Automated test reports are generated and shared with relevant stakeholders, ensuring timely feedback and promoting rapid issue resolution.

3.2 Automated Tools and Framework for Test Case Design and Execution

The diagram describes a framework or architecture for a software test, especially for embedded software. Figure 5 provides a detailed depiction of a data-driven embedded software testing framework, with each component playing a specific role in ensuring comprehensive test coverage and process automation.

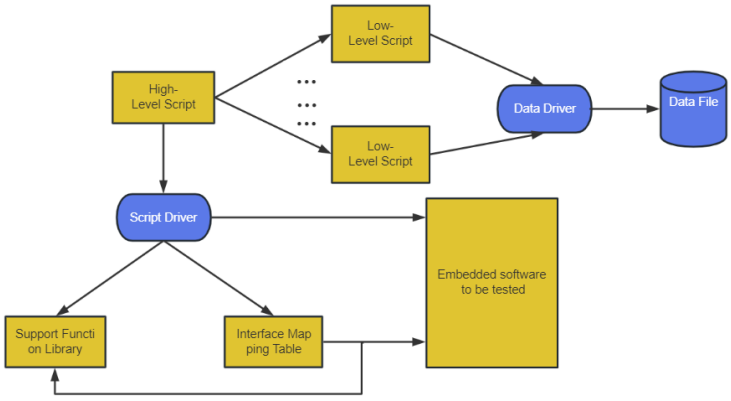

Fig. 5. Automated test framework

The High-Level Script represents an abstract and structured description of test scenarios. It typically outlines the objectives, key actions, and expected outcomes of the test. Beneath this layer, Low-Level Scripts contain the specific test commands and detailed execution steps required to implement the high-level scenarios. These low-level scripts directly interact with the system under test.

The Script Driver serves as the execution engine within the framework. It orchestrates both high-level and low-level scripts, facilitating their execution and potentially invoking auxiliary functions. It also manages interactions between scripts and other components of the system.

The Support Function Library provides reusable auxiliary functions that can be called by the script driver or individual test scripts during execution. These supporting utilities enhance testing efficiency by offering common operations, reducing redundancy across scripts.

The Interface Mapping Table plays a critical role in maintaining the correct associations between test scripts and system interfaces. It ensures that the script driver correctly identifies and interacts with the intended functions or modules during testing.

The Data Driver is responsible for managing test data. It retrieves input datasets from Data Files and supplies them to low-level scripts or directly to the embedded software under test. These Data Files contain structured input data for various test scenarios and conditions, supporting the data-driven nature of the framework.

At the core of this structure is the Embedded Software to be Tested. It interacts with all the above components and undergoes rigorous evaluation across diverse test cases to verify its correctness, robustness, and performance [28].

Overall, the figure illustrates an integrated, automated framework that supports in-depth, data-driven testing of embedded software. By combining layered scripting, automated data management, and reusable support tools, this approach enables scalable and efficient testing processes.

3.3 Continuous Integration of The Environment Building and Configuration

For successful continuous integration, an appropriate development and testing environment needs to be established. Here are some key steps:

i. Environment Configuration Management: Use tools such as Docker, Vagrant, etc. to ensure consistency in the development and test environment to avoid problems in different environments.
ii. Automate deployment scripts to create automated deployment scripts to enable rapid and reproducible deployment of applications to different environments.
iii. Continuous integration tool configuration, configure the continuous integration server, ensure that it can automate build, testing, and deployment tasks and generate corresponding reports and logs.

These methods and strategies will help to integrate automated testing into the CI / CD process of embedded systems, improving development efficiency and software quality. In the subsequent experiments and results section, you can detail how to implement these strategies and their impact on the project.

4. Experimental Verification

4.1 Experimental Design and Execution

In this study, a development board based on the widely used 32-bit microcontroller ARM Cortex-M4 processor was used as a test object. The ARM Cortex-M4 microcontroller has powerful processing power, low power consumption and efficient function, and is very popular in embedded system development. The selection of ARM Cortex-M4 microcontroller as a test object improves the utility and reliability assessment of the embedded software under developed. This selection ensures that the testing process is very similar to the real-world conditions, thus allowing for a more accurate assessment of the software's performance, functionality, and overall robustness. This study aims to develop stable and reliable embedded software to meet the needs of practical application. The embedded software allows for data acquisition, processing, storage, and control of signal output[29]. To ensure optimal performance, the software is designed to provide real-time responsiveness, to enable a low-power operation, maintain stability, display fault tolerance, and provide adequate security measures. The development board shall connect and respond to a variety of sensor data and execute different instructions, including data acquisition, processing, storage, and control signal

output. Development boards typically integrate multiple hardware modules and interfaces so that developers can create and test different hardware and software applications.

32-bit microcontroller ARM Cortex-M4 development board, integrated development environment Keil, version control tool Git. Automated test tools and framework Selenium and JUnit. A computer or server is used to set up the CI / CD environment.

4.1.1 Requirements analysis and project establishment

Prior to the experiment, the requirements analysis of the embedded system was conducted to test the module input, output and interaction functions of the development board, including but not limited to sensor data response, instruction execution and control signal output, GPIO module, ADC module, DAC module, PWM module, timer module, UART module, SPI module, I2C module, USB module, Ethernet module, temperature sensor, light sensor, acceleration sensor. To meet the requirements of real-time performance, the time threshold for the data acquisition and control algorithm execution was set at 5 ms, ensuring that various parameters and signals of the peripheral system can be monitored and responded in real time. Create a new project Project, and set up the development environment to support the ARM Cortex-M4 development board.

4.1.2 Version control and version warehouse

Using Git as a version control tool, a version warehouse was created for the project, submitted at least once daily, ensuring real-time backup of the code and version management. To achieve high reliability, a dual-mode redundancy system was designed and the error detection frequency was set to 1Hz.

4.1.3 Code-based and test-case design

During the coding phase, tests for environmental adaptation were conducted, including a temperature range of-40℃ to 85℃, a humidity range of 5% to 95%, and vibration and electromagnetic interference to stabilize the system under various environmental conditions. Start writing the code, and simultaneously design the corresponding automated test cases. The test case shall cover both functional tests and unit tests, ensuring that all aspects of the project are covered.

For each module and function point of the system, detailed test cases including normal process, abnormal process and boundary conditions are designed to ensure the robustness and stability of the code. In order to cover all possible scenarios, work closely with the development team and business teams to continuously improve and update the test case library.

4.1.4 CI / CD environment settings

The CI / CD environment is configured on a dedicated computer and uses the Jenkins as the CI / CD tool. At least 1000 tests of communication interfaces were conducted, including various peripheral, interfaces to ensure stable communication of the development board with the peripheral sensors.

4.1.5 Automated test and integration

Integrate the automated tests into the CI / CD process. Ensure that the execution of the test cases is automatically triggered after each code submission. Functional testing using Selenium to simulate user operation and verify that the functionality of the system is as expected. The Selenium scripts are regularly reviewed and updated to accommodate system changes and new requirements. Unit testing using JUnit, unit tests were written for each function to verify the logical correctness of the code and exception handling power. Code coverage is tracked, ensuring that both important code paths and logical branches are test-covered. Through Mock object and dependency injection, external dependencies can be isolated for accurate unit testing. Integrate the automated testing into the CI / CD process, conduct an hourly system health check, and automatically trigger the testing after each code submission, and carry out at least one comprehensive test per day to realize the function of fault detection and diagnosis.

4.1.6 Monitoring and feedback settings

In order to meet the requirements of low power consumption, the system monitors the energy consumption in real time, the data is recorded every minute, and the warning threshold is set at 50W. A real-time feedback mechanism was set up to notify the development team once a problem is found. Generate detailed test reports, including test coverage, execution time and test results, failed test cases, error logs, etc., to facilitate team analysis and improvement[30]. And provide it to the team members [31].

After new feature development and bug repair, regression tests are performed to ensure that the modifications do not introduce new errors. Performance tests were also performed to simulate scenarios with high concurrency and large data volume to ensure the stability and response speed of the system under pressure. Through these measures, the CI / CD process greatly improves the efficiency of development and the quality of the software, ensuring that every release is rigorously tested and validated.

4.1.7 Measurement of reduced development cycles

Standard development cycles were recorded in days before the start of the experiment. Then, after applying the CI / CD process, the new development cycle is recorded. Calculate the percentage reduction in the development cycle. Record the time of each development cycle, and conduct the safety assessment once weekly and conduct the deep safety test once a month to ensure the safety of the system.

By comparing the Gantt chart, it is obvious that after the introduction of the CI / CD process, especially in the "environmental configuration" and "test integration" stages, the development cycle of the project is significantly shortened, thus improving the development efficiency. The Gantt chart clearly shows the various phases of the project and its duration, which helps the team members to understand the project progress and immediate tasks, and ensure the orderly progress of the project.

The project team strengthened internal collaboration and communication with customers, and was able to timely solve problems and demand changes in the development process, ensuring the smooth progress and successful delivery of the project. The project team regularly conducts safety assessments and in-depth tests to ensure the safety of the project, which is an important guarantee for the success of the project. By recording and analyzing the time of each development cycle, the project team continuously optimizes the development process and improves the work efficiency, which contributes to the smooth progress of the project.

i. Development efficiency and time management
In the case of CI / CD, each stage was continuous and compact, with a total development period of 35 days. In the absence of CI / CD, the total development cycle increased to 50 days due to the environmental configuration and the test integration phase of 10 days respectively, 15 days more than CI / CD, nearly 43% compared to the use of CI / CD.

ii. Project stage management
Whether using CI / CD or without CI / CD, projects were divided into seven major phases, each with clear start and end times, which helped the team to better manage the project schedule and resources.

iii. Team cooperation and communication
From the orderly progress of the project, we can see that the cooperation and communication between the teams is smooth.

iv. Safety management
Security management is a part of the project [32].

v. Customer feedback and demand change
The figure shows the various stages and the schedule of the project.

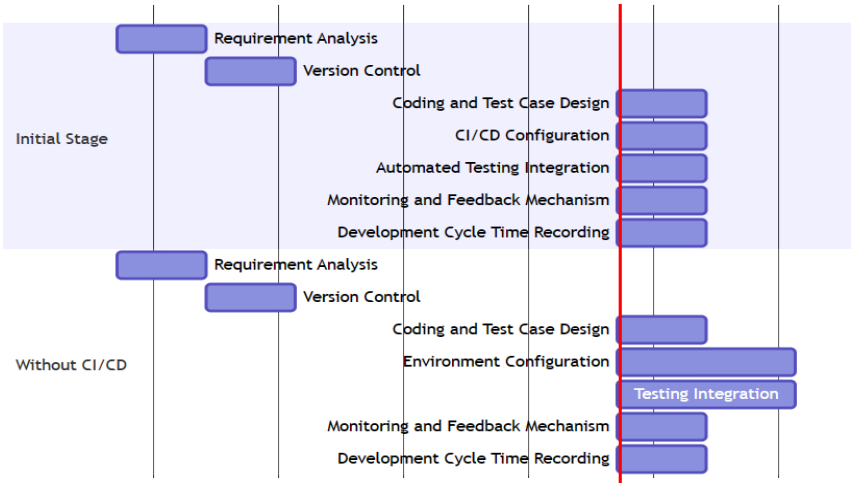

Fig. 6. Project development cycle

In conclusion, the initial development cycle is 50 days and the development cycle after CI / CD, percentage reduction: (50-35) / 50 * 100% = 30%. By introducing CI / CD, the project achieves efficient and professional project management, strengthening team collaboration and customer communication, the emphasis on security management, and the optimization of time management.

4.1.8 Measurement of the test coverage rate

Use of the automated test tools and framework to measure test coverage. Tocan reports tool be used to get the exact value. Test coverage was measured with an initial coverage of 80% and the goal to improve to 97%. To meet the firmware update requirements, the firmware update test is conducted monthly to ensure that the firmware update function is supported. Ensure that the test coverage rate exceeds 95%.

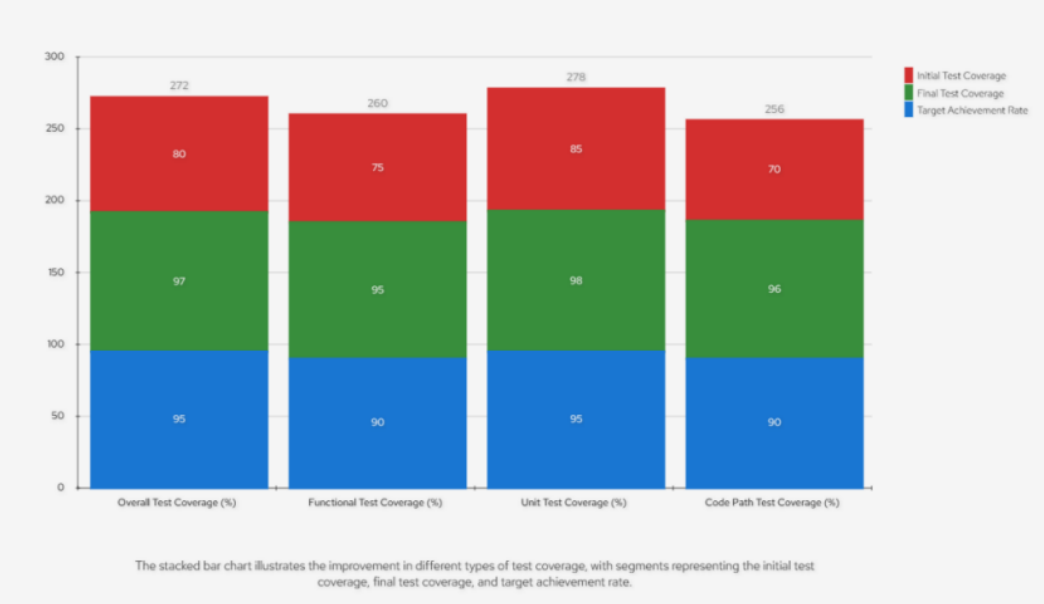

Fig. 7. Test coverage improvement

The improvement in test coverage can be clearly seen through the figure above and to ensure that the indicators meet the predetermined goals. This helps the project team to better understand the status of test coverage and to take the necessary steps to improve coverage. Increase in test coverage:

Significant coverage for all types of tests. Overall test coverage improved from an initial 80% to 97%, exceeding the set target of 95%.

Functional test coverage, unit test coverage, and code path coverage also exceeded their respective target achievement rates.

Measurement of test coverage using automated test tools and frameworks facilitates a more precise and efficient assessment of the quality and robustness of the code. Through the practice of continuous integration / continuous deployment (CI / CD), the team is able to ensure that code changes do not reduce test coverage, but rather help improve coverage. The project team successfully increased the test coverage to over 95%, which helped to reduce the defects in the software and improve the product quality and reliability. With the tool generated reports, the team can obtain the exact value of test coverage to better understand the status of the project and take the measures necessary steps to maintain or improve the test coverage.

While test coverage has reached high levels, the team continues to ensure that new code submissions do not reduce coverage. The team can further explore more advanced testing strategies and tools to continuously optimize the testing process and ensure the quality of the software. In short, the project has performed well in the test coverage and successfully achieved the set goals, but still remain vigilant and continuously optimize the testing process.

4.1.9 Measurement of the defect resolution speed

Record the time of defect detection and resolution. Compare the speed of defect resolution before and after using the CI / CD process and calculate the percentage improvement in the CD speed. In the experiment, the detection and resolution time of each defect were recorded as shown in the figure.

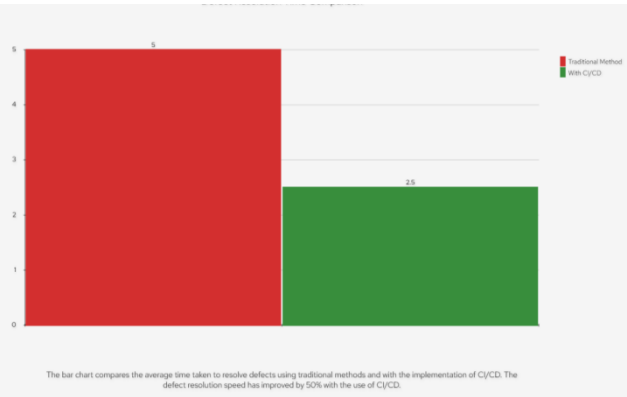

Fig. 8. Defect resolution speed measurement

The mean time to resolve defects using conventional methods was 5 days, and that after using CI / CD was 2.5 days,

Percentage improvement in defect resolution speed: (5-2.5) / 5 * 100% = 50%

4.1.10 Experimental summary and conclusion

Based on the experimental results, the effects of the development board using the 32-bit microcontroller ARM Cortex-M4 combined with CI / CD and automated testing were summarized. Emphasis on the advantages of improved development efficiency, quality standards and team collaboration.

4.2 Efficiency and Effectiveness of Continuous Integration and Continuous Delivery Processes

The following effects and improvements were observed:

i. Faster development cycles, and CI / CD processes have significantly reduced development cycles. Developers are able to submit code more frequently without having to wait for a long manual testing and deployment process.
ii. Higher quality standards, automated testing, and continuous integration ensure higher software quality. Problems can be found and solved in the early stage, thus reducing the workload of later maintenance.
iii. Better teamwork, CI / CD facilitated collaboration of development teams. Tests are run automatically with each code submission, making the responsibility for the problem clearer and the team members to resolve it quickly.

5. Discussion

5.1 Significance and Impact of The Experimental Results

The experimental results reveal the significance and impact of integrating automated tests into the CI / CD process of embedded systems. First, a significant reduction in the development cycle was observed, which is essential to meet market demand and changes in competitors. Faster development cycles allow products to come to market faster, thus increasing competitiveness.

Secondly, the high-quality standards are the key elements of the embedded systems. Automated testing and continuous integration ensure improved software quality, reduced accumulation of errors, lower maintenance costs, and enhanced reliability and stability of the system. This is particularly important for embedded system applications, such as medical, automotive, industrial control, etc.

Most importantly, the experimental results indicate an improvement in teamwork. The CI / CD process encourages close collaboration between developers, testers, and operations personnel, making the responsibility of issues clearer. This helps to find and solve problems faster, improving the efficiency and satisfaction of the whole team.

5.2 Advantages and Challenges of Automated Testing in CI / CD

Although automated testing brings many advantages in CI / CD, there are some challenges. During the discussion, these aspects need to be analyzed.

The advantages are as follows:

i. Efficient testing: Automated tests can perform test cases quickly, provide quick feedback, and help quickly detect problems.
ii. Improve quality: Through automated testing, increased test coverage, reduced human error, and enhanced software quality.
iii. Automated Deployment: Part of the CI / CD process is automated deployment, which ensures that every code submission can be quickly deployed to a different environment.

The challenges are as follows:

i. Maintenance of automated test cases: As the code changes, test cases need to be updated and maintained, or test cases fail[33].
ii. Resource requirements: Building and maintaining the CI / CD environment requires certain resources and investment.
iii. Learning Curve: Team members may need time to adapt to the CI / CD process and automated testing tools.

5.3 Future Research Direction and Improvement Strategies

In terms of future research direction and improvement strategies:

i. Continuous improvement process, Further improved CI / CD processes, including automated testing, deployment, and monitoring, to improve efficiency and stability.
ii. Research on test automation tools, Continue to study and evaluate new test automation tools and frameworks to meet the needs of embedded system development.
iii. Integrated security testing, Automatic safety testing is introduced to ensure the security and reliability of the embedded systems.
iv. Machine learning and artificial intelligence applications, Explore how machine learning and artificial intelligence can be applied to automated testing to improve test coverage and the ability to detect hidden flaws.
v. Cross-teamwork, Cross-teamwork research involving development, testing, operations, and security teams to achieve a more comprehensive CI / CD process.

Through continued research in these directions and continuous improvement of CI / CD processes and automated testing strategies, the efficiency and quality of embedded system development can be further improved against evolving market demands and technical challenges.

6. Conclusion

By studying in detail the methods and effects of integrating automated testing into the continuous integration and continuous delivery (CI / CD) process of embedded systems, drawing the following important conclusions:

First, we demonstrate that automated testing and continuous integration have great implications for embedded system development. Through the experimental results, a significant reduction in the development cycle, higher software quality standards, and better team collaboration were clearly observed. These results demonstrate that integrating automated testing into embedded system development is not just a trend but an essential strategy to significantly improve the efficiency and quality of software development.

Secondly, it highlights the advantages of automated testing in CI / CD, including efficient testing, quality standard improvement, and automated deployment. However, also recognized possible challenges in the automated testing process such as maintenance of test cases and resource requirements. Nonetheless, these challenges can all be addressed with appropriate strategies and tools, and the advantages far outweigh the challenges.

Most importantly, the embedded system development industry is encouraged to adopt this strategy. Continuous integration and continuous delivery and automated testing are not only for traditional software development but also for embedded system development. By adopting these best practices, embedded system developers can better meet changing market needs, improve software quality, and ultimately achieve higher customer satisfaction.

In the future, further research and improvement of automated testing and CI / CD processes are encouraged, especially for applications in the field of embedded systems. Some future research directions are also proposed, including research on test automation tools, integration of security testing, machine learning and application of artificial intelligence. Through continuous efforts, the efficiency and quality of embedded system development can be further improved to meet future challenges and opportunities.

In conclusion, this paper highlights the importance of integrating automated testing into the development of embedded systems and encourages the industry to adopt this strategy to improve software quality and development efficiency. This strategy will be widely used in the field of embedded system development, to provide higher quality software for future embedded systems.

Acknowledgement

This research was not funded by any grant.

References

[1] Zheng, Xiaoxia, and Fang Dong. "[Retracted] Application of Virtual Instrument Technology in the Teaching of Embedded System Course." *International Transactions on Electrical Energy Systems* 2022, no. 1 (2022): 6721450. https://doi.org/10.1155/2022/6721450

[2] Sozzo, Emanuele Del, Davide Conficconi, Alberto Zeni, Mirko Salaris, Donatella Sciuto, and Marco D. Santambrogio. "Pushing the level of abstraction of digital system design: A survey on how to program fpgas." *ACM Computing Surveys* 55, no. 5 (2022): 1-48. https://doi.org/10.1145/3532989

[3] Ren, Xinguang, and Yongmin Cui. "Embedded system intelligent platform design based on digital multimedia artistic design." *Wireless Communications and Mobile Computing* 2021, no. 1 (2021): 3959199. https://doi.org/10.1155/2021/3959199

[4] Zhang, Tian. "Application of AI-based real-time gesture recognition and embedded system in the design of English major teaching." *Wireless Networks* (2021): 1-13. https://doi.org/10.1007/s11276-021-02693-0

[5] Zhong, Cao, Li Wen-Feng, Liu Chen, Peng Yu-Yu, Huang Ying, and Xiao Zhong-Liang. "Design and fabrication of embedded wireless monitoring system based on Bluetooth low energy transmission for potentiometric sensors." *Chinese Journal of Analytical Chemistry* 47, no. 2 (2019): 229-236.

[6] Belhamel, Loubna, Arturo Buscarino, Antonio Cucuccio, Luigi Fortuna, and Gaetano Rasconà. "Model-based design streamlines for STM32 motor control embedded software system." In *2020 7th International Conference on Control, Decision and Information Technologies (CoDIT)*, vol. 1, pp. 223-228. IEEE, 2020. https://doi.org/10.1109/CoDIT49905.2020.9263910

[7] Kouki, Rihab, Alexandre Boe, Thomas Vantroys, and Faouzi Bouani. "Autonomous Internet of Things predictive control application based on wireless networked multi-agent topology and embedded operating system." *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 234, no. 5 (2020): 577-595. https://doi.org/10.1177/0959651819870340

[8] Aparo, Carmelo, Cinzia Bernardeschi, Giuseppe Lettieri, Fabio Lucattini, and Salvatore Montanarella. "An analysis system to test security of software on continuous integration-continuous delivery pipeline." In *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 58-67. IEEE, 2023. https://doi.org/10.1109/EuroSPW59978.2023.00012

[9] Bhadra, Sushovan. "A stochastic Petri net model of continuous integration and continuous delivery." In *2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 114-117. IEEE, 2022. https://doi.org/10.1109/ISSREW55968.2022.00050

[10] Parama, Restu Agung, Hudan Studiawan, and Rizky Januar Akbar. "Implementasi Continuous Integration dan Continuous Delivery Pada Aplikasi myITS Single Sign On." *Jurnal Teknik ITS* 11, no. 3 (2022): A264-A269. https://doi.org/10.12962/j23373539.v11i3.99436

[11] Singh, Amarjeet, Vinay Singh, Alok Aggarwal, and Shalini Aggarwal. "Improving Business deliveries using Continuous Integration and Continuous Delivery using Jenkins and an Advanced Version control system for Microservices-based system." In *2022 5th International Conference on Multimedia, Signal Processing and Communication Technologies (IMPACT)*, pp. 1-4. IEEE, 2022. https://doi.org/10.1109/IMPACT55510.2022.10029149

[12] Merino, Pablo, Gabriel Mujica, Jaime Señor, and Jorge Portilla. "A modular IoT hardware platform for distributed and secured extreme edge computing." *Electronics* 9, no. 3 (2020): 538. https://doi.org/10.3390/electronics9030538

[13] Mårtensson, Torvald, Daniel Ståhl, and Jan Bosch. "Test activities in the continuous integration and delivery pipeline." *Journal of Software: Evolution and Process* 31, no. 4 (2019): e2153. https://doi.org/10.1002/smr.2153

[14] Khodos, Daniel R., David I. Adegbesan, Oliver Khairallah, and Shouling He. "Team Cleaning Robots." In *2018 ASEE Annual Conference & Exposition*. 2018.

[15] Castillo-Martínez, Diego Hilario, Adolfo Josué Rodríguez-Rodríguez, Adrian Soto, Alberto Berrueta, David Tomás Vargas-Requena, Ignacio R. Matias, Pablo Sanchis, Alfredo Ursúa, and Wenceslao Eduardo Rodríguez-Rodríguez. "Design and on-field validation of an embedded system for monitoring second-life electric vehicle lithium-ion batteries." *Sensors* 22, no. 17 (2022): 6376. https://doi.org/10.3390/s22176376

[16] Ren, Xinguang, and Yongmin Cui. "Embedded system intelligent platform design based on digital multimedia artistic design." *Wireless Communications and Mobile Computing* 2021, no. 1 (2021): 3959199. https://doi.org/10.1155/2021/3959199

[17] Ibrahim, Bishar R., Farhad M. Khalifa, Subhi RM Zeebaree, Nashwan A. Othman, Ahmed Alkhayyat, Rizgar R. Zebari, and Mohammed AM Sadeeq. "Embedded system for eye blink detection using machine learning technique." In *2021 1st Babylon International Conference on Information Technology and Science (BICITS)*, pp. 58-62. IEEE, 2021. https://doi.org/10.1109/BICITS51482.2021.9509908

[18] Kissich, Meinhard, Klaus Weinbauer, and Marcel Baunach. "ATTEST: Automated and Thorough Testing of Embedded Software in Teaching." In *Proceedings of the 5th European Conference on Software Engineering Education*, pp. 199-203. 2023. https://doi.org/10.1145/3593663.3593678

[19] Pham, Khang, Vu Nguyen, and Tien Nguyen. "Application of natural language processing towards autonomous software testing." In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1-4. 2022. https://doi.org/10.1145/3551349.3563241

[20] Foss, Kyle, Ivo Couckuyt, Adrian Baruta, and Corentin Mossoux. "Automated software defect detection and identification in vehicular embedded systems." *IEEE Transactions on Intelligent Transportation Systems* 23, no. 7 (2021): 6963-6973. https://doi.org/10.1109/TITS.2021.3065940

[21] Strandberg, Per Erik. *Automated system-level software testing of industrial networked embedded systems*. Malardalen University (Sweden), 2021.

[22] Xie, Xinqiang, Xiaochun Yang, Bin Wang, and Qiang He. "DevRec: Multi-relationship embedded software developer recommendation." *IEEE Transactions on Software Engineering* 48, no. 11 (2021): 4357-4379. https://doi.org/10.1109/TSE.2021.3117590

[23] Lemes, Marcelo JR, and Adriano Sarmento. "Qualifying People for Embedded Software Development and Data Science: An Experience on University-Industry Cooperation." In *Congresso Brasileiro de Software: Teoria e Prática (CBSoft)*, pp. 1-4. SBC, 2022. https://doi.org/10.5753/cbsoft_estendido.2022.226175

[24] Cai, Jing. "Research on embedded software based on adaptive filtering algorithm." In *2022 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*, pp. 973-979. IEEE, 2022. https://doi.org/10.1109/AEECA55500.2022.9918946

[25] Wang, Boxiang, Rui Chen, Chao Li, Tingting Yu, Dongdong Gao, and Mengfei Yang. "SpecChecker-ISA: a data sharing analyzer for interrupt-driven embedded software." In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 801-804. 2022. https://doi.org/10.1145/3533767.3543295

[26] Zhang, Weiyan, Mehran Goli, and Rolf Drechsler. "Early performance estimation of embedded software on risc-v processor using linear regression." In *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pp. 20-25. IEEE, 2022. https://doi.org/10.1109/DDECS54261.2022.9770144

[27] Su, Zhuo, Dongyan Wang, Yixiao Yang, Yu Jiang, Wanli Chang, Liming Fang, Wen Li, and Jiaguang Sun. "Code synthesis for dataflow-based embedded software design." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, no. 1 (2021): 49-61. https://doi.org/10.1109/TCAD.2021.3055487

[28] Mittal, Rakshit, Dominique Blouin, and Soumyadip Bandyopadhyay. "PNPEq: Verification of Scheduled Conditional Behavior in Embedded Software using Petri Nets." In *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 509-514. IEEE, 2021. https://doi.org/10.1109/APSEC53868.2021.00059

[29] Mahdian, Navid, Seyed-Hosein Attarzadeh-Niaki, and Armin Salimi-Badr. "A systematic embedded software design flow for robotic applications." In *2021 11th International Conference on Computer Engineering and Knowledge (ICCKE)*, pp. 217-222. IEEE, 2021. https://doi.org/10.1109/ICCKE54056.2021.9721465

[30] Hierons, Robert M., and Tao Xie. "Adaptive or embedded software testing and mutation testing." *Software Testing: Verification & Reliability* 31, no. 7 (2021). https://doi.org/10.1002/stvr.1798

[31] Liu, Yuchu, David Issa Mattos, Jan Bosch, Helena Holmström Olsson, and Jonn Lantz. "Bayesian propensity score matching in automotive embedded software engineering." In *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 233-242. IEEE, 2021. https://doi.org/10.1109/APSEC53868.2021.00031

[32] Liu, Yuchu, David Issa Mattos, Jan Bosch, Helena Holmström Olsson, and Jonn Lantz. "Size matters? Or not: A/B testing with limited sample in automotive embedded software." In *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 300-307. IEEE, 2021. https://doi.org/10.1109/SEAA53835.2021.00046

[33] Liu, Yuchu, Jan Bosch, Helena Holmström Olsson, and Jonn Lantz. "An architecture for enabling A/B experiments in automotive embedded software." In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 992-997. IEEE, 2021. https://doi.org/10.1109/COMPSAC51774.2021.00134