Biola Balogun

Testing Embedded Software for an IoT-Enabled Acoustic Imaging Camera

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master's Thesis
11th of August 2025

PREFACE

This thesis is a trip to the world of Embedded software applications. The trip has challenged me, shaped me, and widened my understanding of both manual and automated testing strategies. I was driven by documenting and teaching my four years of testing and validating embedded software applications in a device that listens to low-frequency acoustics events. I wanted to demonstrate the benefits and the drawbacks of both manual and embedded test scripts. What started as a single letter and a line of code gradually developed into a full diary of knowledge and testing practice that anyone can implement. The IoT-enabled acoustics imaging camera can detect a low whisper of fault, an invisible event, but critical to industrial maintenance. To my supervisor, Dr. Antti Piironen, whose guidance helped me stay focused while working on this project, and Esko Sivunen, my supervisor at Teledyne-NLAcoustics, thank you. To the people who beautify my world, Dr. Abiodun Balogun, Ayomide, Ayomikun, Iredamola, Olamide, and Oluwagbemiga, and friends (Seun & Ladoke), thank you for believing that I will complete whatever I have started. To my mom, you taught me persistence, patience, and the satisfaction of solving problems quietly. This project is dedicated to all of us who develop, test, and improve systems that keep our world running and safe.

Espoo, Finland, 11th of August 2025

Biola Balogun

Abstract

Author:	Biola Balogun
Title:	“Testing Embedded Software for an IoT-Enabled Acoustic Imaging Camera”
Number of Pages:	34 + 1 appendix
Date:	11th of August 2025
Degree:	Master of Engineering
Degree Programme:	Information Technology
Professional Major:	Networking and Services
Supervisors:	Antti Piironen, Project Manager
	Ville Jääskeläinen, Principal Lecturer

This thesis enumerates the verification and validation of an embedded software application for an IoT-enabled acoustic imaging camera. The device is used to detect abnormal events, partial discharge, and air leaks of low frequency sound. A testing framework was developed that combined manual and automated test script methodologies, and the methods were evaluated based on four known criteria, namely performance, scalability, reliability, and cost. The result indicated that automated testing improves the testing efficiency of embedded software applications, reduces time, and feedback time. Offering scalable testing practice for future IoT-based devices.

Keywords: Appium, Python Client, Test-Double, Testing, Automated Test Script, Manual Test Environment, Partial Discharge and Air Leak

Table of Contents

List of Abbreviations

IoT	Internet of Things
PD	Partial Discharge
LD	Leak Discharge (Air Leak)
R&D	Research and Development
OS	Operating System
SWD	Serial Wire Debug
JTAG	Join Test Action Group
QA	Quality Assurance
HIL	Hardware-in-the-Loop
OTA	Over-the-air
IDE	Integrated Development Environment
UI	User Interface
CI/CD	Continuous Integration and Continuous Development
IP	Internet Protocol
OTA	Over the Air
USB	Universal Serial Bus
FAT	Factory Acceptance Testing
CPU	Central Processing Unit
EMI	Electromagnetic Interference

1 Introduction

Sound detection technology can visualize sound sources such as compressed Air leak (LD), Partial discharge (PD) in electrical transmission and distribution lines, and bearing failure on a conveyor belt. The need for low-frequency sound detection devices has increased over the last decade for industrial and maintenance applications. The technology allows the detection of leaks in the form of sounds at a low frequency that cannot be detected by the human ear. By accurately identifying these sound sources and estimating the leak cost in the case of pressurized air leak and estimating the severity of damage in the case of Partial Discharge (PD), preventive action can be taken by the maintenance teams, and energy loss, hazards, and system failure can be averted.

This introductory chapter explains the focus of the thesis, “Testing the embedded software of an acoustic imaging camera”. The connectivity of the Internet of Things and the acoustic imaging camera are two capabilities employed by the technology. The subsequent sections in this chapter explain the motivation for this thesis and express the problem statement.

The motivation behind writing this thesis was conceived and actualized by the need to share my experience as a Software Testing Engineer. I have used several testing tools for testing and validation on the web, Windows, and cloud-based applications, but testing and validation of an Internet of Things (IoT) enabled acoustics device was particularly a learning curve for me. IoT is used in the manufacturing and industrial equipment and monitoring sectors. The ability to detect leaks at an early stage, which can prevent costly downtime, provide safety, and improve the system's overall performance (Zhang et. Al., 2020). The embedded software that drives the acoustics imaging camera, such as Flir Si2 serial, must be carefully tested to ensure the reliability of the estimates it provides and mic accuracy to avoid internal and environmental noise, which may interfere with the measurement (Lin et al., 2019).

Embedded systems play a critical role in IoT-enabled devices, providing the needed hardware and software ability to perform complex tasks in real time.

However, the performance and reliability of embedded software remain a challenge, as the system often operates in a changing and unpredictable environment (Norris, 2018).

The aim and objective of this thesis are to demonstrate testing methodologies, for the embedded software of an IoT-enabled acoustics imaging camera. This testing project is driven by the necessity to develop effective automated test and validation strategies that will yield high-quality software, which can give reliable leak estimates and prevent late detection of critical issues in an industrial setting. Appropriate testing methodologies and practices suitable for embedded software, tailored for IoT devices. This thesis addresses the drawbacks in the current testing methods and provides recommendations that will improve testing Speed, Reliability, Cost, and Flexibility.

This document, “Testing Embedded Software for an Acoustics IoT-Enabled Imaging Camera,” is composed of six chapters, and each chapter builds upon the next, forming the entire structure of this material. Chapter 2 outlines the methodology and materials that explain the detailed approach employed in testing the embedded application and the drawbacks encountered in each phase of testing. Chapter 3 provides the background information on the IoT-enabled imaging camera, which is designed to detect low-frequency Partial Discharge (PD) and Leak Detection (LD). It also describes the testing framework, tools, and the testing management tool. Chapter 4 explained the detailed approach for running the test cases using an automated testing environment instead of manual testing. Chapter 5 describes the test results and the performance analysis, which also includes finding and identifying areas for improvement. Chapter 6, as the final section of this thesis, outlines the conclusion and provides recommendations for future improvements needed in the testing of embedded software.

2 Methods and Materials

The thesis employs a comprehensive review of the current state of testing methodologies involved in testing embedded software applications, with particular focus on methods used in acoustic imaging devices. Automated testing techniques with the use of Appium will be employed based on its applicability to Flir Si2 series of imaging cameras. The choice of Appium as an automation testing framework to execute the test cases is because it allows custom drivers to be designed and built specifically for custom use. Appium allows end-users to design and build custom Appium drivers based on individual use-cases. The primary material for this thesis will include the Flir Si2 documentation, the user's general manual that are publicly available to end-users, and freely downloadable on the internet, these manual outlines the technical specifications and operational capabilities of the Si2 Pro, Si2 PD, Si2 LD models, and ATEX cameras. These documents provide insights into the software UI/UX design and the functionalities that will guide the testing process and techniques. Secondly, books and articles that are relevant for testing IoT-enabled acoustics devices will be consulted to provide a rich theoretical background for the development of a testing strategy. Key text materials will include beginner books on software testing and validation (M. Binder, 2023) and articles focusing on the intricacies of testing embedded software (I. M. B. Singh, 2022; S. B. Choudhary et al., 2021). Finally, testing will be done with the aid of Flir Si2-Pro device to assess its performance in an actual scenario, the focus will be on sensitivity to various frequencies of sounds. The results from this automated testing procedure will contribute to building a comprehensive software validation in the field of acoustic imaging. Ultimately, the end goal is to achieve a product with enhanced product reliability and user satisfaction, thereby addressing the actual need for comprehensive automated testing of an embedded device.

2.1 Current Testing State

The lifecycle of embedded software involves design, development, and testing of the firmware for an IoT-enabled imaging camera. The development is done by

specialized teams working in a collaborative effort to deploy quality software to production. Each team contributes to the testing lifecycle at different stages of development, from unit testing to the final verification and validation stage. This chapter enumerates the current testing methodologies employed by the Research and Development (R&D) team. Quality Assurance (QA) team and the importance of system testing. The Research and Development (R&D) team has ownership responsibility to design, develop, test, and deploy the embedded application for the IoT-enabled camera. The R&D team consists of highly skilled individuals, each with their own area of focus, area of focus could be image processing, hardware integration, or firmware development. Their individual responsibility is interrelated to ensure that the software is reliable and performs well. At every development stage, each team member carries out different types of testing to catch bugs early enough and maintain a certain quality standard. While the QA team runs system tests independently to verify and validate the designed functionalities of each feature, and or integrated features (when two units feature are integrated together). The QA team ensures that the entire system functions together as a whole and as intended before the product is released to production.

2.2 Embedded Team

The Embedded Team is responsible for developing the Operating System (OS) and the firmware that controls the device's main functionalities, such as the microphone array, Raspberry Pi (also known as Microprocessor), Internal battery, data acquisition, and video streaming. The team testing efforts include. The testing process involves Unit Testing, where each individual feature is tested as it is being built. This is done using testing frameworks such as CppUTest and Google Test (memon, 2020). They also perform static Analysis. This method involves examining the vulnerability in the code without executing the code, and in the context of Embedded software a specialised tools such as SonarQube and Coverity are used (Chess & West, 2007). For issues that occur at the hardware level, the team uses the Hardware Debugging technique to verify and resolve issues at the firmware execution at the hardware level. Specialised tools such as JTAG and SWD interfaces are employed for real-time investigation on ARM

Cortex-M processors. However, the R&D team has a unique challenge in the simulation of real industrial operational conditions during testing, because of this challenge, most tests are conducted in a lab environment, which may represent the complexity of the real operational environment.

2.3 Analytic team

This team is primarily responsible for testing and refining algorithms, which ensures that the analytic module performs accurately and efficiently before it can be integrated into the system. The testing methodologies in this team include. Algorithm Validation, where they check the correctness, performance, and reliability of computational methods, using tools like MATLAB and Python-based simulation. This method allows the team to see potential bugs early enough and be able to fine-tune the algorithm before final implementation.

Another technique used is Edge Case Testing. This is a testing technique that focuses on examining the system's reaction when unusual or data that are at the edge of expectation are entered, such as a Synthetic dataset with an extreme noise level. Performance Benchmarking: a testing technique that measures accuracy and latency against an industrial standard (FLIR, 2023). The notable limitation with this team is the over-dependence on simulated data instead of real industrial acoustics data, which may not be able to simulate the entire industrial scenario.

2.4 Hardware Team

The design of the device’s electronic components is the responsibility of the hardware team, and that team also ensures that software and hardware work seamlessly together without any issues. The team employed Hardware in the Loop (HIL), a Testing methodology that ensures real-time communication between software and hardware. This technique is used to verify and validate system behaviour using NI LabVIEW (Jin et al., 2021). Another important technique is Environmental Stress Testing, where device performance is validated under extreme temperature conditions, such as high and low

temperatures, testing device reliability outside operating conditions, such as thermal, electromagnetic interference (EMI) testing device durability and physical vibration testing, to assess the device durability.

Also, perform Power Consumption Analysis by measuring how efficiently the device uses energy at different operating conditions. EMI also aids in optimizing battery lifespan and helps the device perform well in practical scenarios. One of the main challenges to the team is accurately simulating a real-world industrial scenario in a controlled environment. This limitation might potentially lead to an undetected bug or issues that only appear in production.

2.5 Operation/Cloud Team

The cloud team is responsible for designing, building, and managing cloud-based applications. Their primary role is ensuring the seamless deployment and maintenance of software application systems. Beyond their primary functions, the team also ensures that updates via IP are possible. This update method is known as OTA deployment and patch updates, which are necessary for software maintenance and for keeping the acoustic device operationally safe and running smoothly. Part of the team’s ownership responsibility is to roll out updates without any issues and seamlessly without any disruption to the device's operational performance in different environments. Testing procedures involve a good number of critical steps. The Network Security Test involves the process of scanning for security vulnerabilities and reliability of the connected IoT device, reducing the attack surface, and ensuring that the device can protect itself against potential security threats. Cloud End-to-End testing verifies the consistency and reliability of data after the synchronization process with the cloud user interface (UI) or a cloud platform. Reliability Testing is done to verify and validate that the patch and update do not break the device functionality during and after operation. Meanwhile, the maintenance of seamless interactions among different IoT protocols continues to be a challenge to the team.

2.6 Quality Team (System Testing)

The Quality Assurance (QA) team has the ownership responsibility to validate end-to-end system testing before production release. Their testing activities involve. Functional testing: To ensure that all designed features function as specified and approved in the system requirement specifications document. Regression test ensures that batch releases and patch updates are verified without a break to the existing functionalities. Presently, these test suites are manually executed. User Scenario Testing: Device usage in an industrial environment, which represents real-world usage. The activity is aimed at uncovering usage issues. Despite rigorous testing rounds during end-to-end system testing, the QA team still faces challenges such as Automation Testing: 100% of the executed test rounds are done manually, manual tests are prone to error and are time-consuming. Firmware-Hardware Dependencies: Bugs are sometimes found under specific hardware conditions. It is more difficult to scale test across different software configurations, and the scalability issue increases testing difficulties.

2.7 The Summary

The present testing methodologies for IoT-enabled imaging devices employ a multi-team approach, whereby each team contributes specialized testing methods and expertise. However, improvements in the testing techniques and methodologies are highly desired now. Comprehensive Unit ensures that individual modules work as specified, and Integration Testing improves interaction within the modules. HIL and Environmental Testing verify and validate that hardware and software interactions are smooth. The current techniques also come with it inherent weaknesses, such as the over-reliance on Simulated Data in testing. Therefore, the Industrial acoustic data are not fully replicated. Secondly, the lack of Automated Testing Suites. Manual testing is labour-intensive and time-consuming. Low testing Resources are another form of weakness of the current practice. The QA team requires more resources to keep up with release cycles. Finally, the limited Fault Injection Test and rare edge cases may also go unnoticed. Some forms of improvement are required to

improve the current testing techniques, such as enhancing the Field Test and incorporating industrial acoustic data. Introduce the Automation Testing Framework. The Advanced Fault Injection Test aids in finding and strengthening the weak parts in the software application. This is done by reducing the threat surface. This testing technique helps to scale testing processes, ensure reliability, and increase testing efficiency, which can ultimately improve the quality of the software in production. The techniques, such as Fuss testing, are used to uncover rare and difficult-to-detect software issues.

3 Background Information

The development of embedded and validation of embedded applications for an acoustic imaging camera requires a good knowledge and understanding of various aspects of technical, methods, and organizational aspects. This chapter explains the foundational background on the major aspects involved in the process, including literature materials on performing system tests on embedded applications, methodologies in application development, testing management tools, and testing frameworks. An understanding of these elements is critical in the design and implementation of an effective automated testing environment. This chapter starts with a literature review to establish foundational knowledge, followed by an exploration of the process of application development. It then explores the testing framework, testing and bug management tools, and the supporting technologies such as drivers and IDEs (Integrated Development Environments).

3.1 Literature review

The testing of embedded software applications has been extensively studied in the industrial sector and academia. According to Ammann and Offutt (2016), a specialized testing approach is required for testing embedded systems because of real-time limitations and hardware dependencies. Whittaker (2019) argues that the importance of automated testing improving efficiency and reducing human error is important. In addition, IoT has inherent challenges such as security vulnerability and interoperability, these challenges are also emphasized by Zheng et al. (2018). FLIR's documentation (2023) gives insight into the testing requirements and functionalities of acoustic imaging devices. Research on fault injection methods (Sutton et al., 2007) and Hardware-In-the-loop (HIL) testing techniques (Jin et al., 2021) further includes best practices in embedded testing. This literature review forms the basis for developing a comprehensive testing framework.

3.2 Application development

The software development lifecycle (SDLC) of embedded applications for IoT-enabled devices follows an iterative process and mostly incorporates DevOps or Agile methodologies. Using Agile methodology in software development will change the traditional linear practices into more iterative and collaborative practices. Agile ensures customer feedback, quick delivery, and flexibility. The embedded firmware is written in C/C++, while other higher functionality could be written in Node.js or Python. Continuous Integration and Continuous Development (CI/CD) process allows early bug detection and fast feedback from the QA team and ensures frequent code injection (Chess and West, 2007)

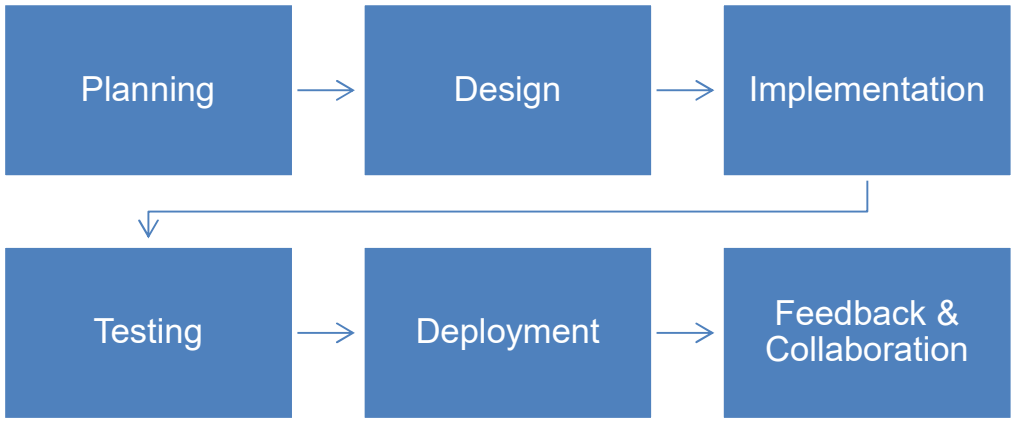


Figure 1. SDLC and Agile Methodology.

Cross-functional cooperation between teams is very important; the analytic team defines and refines algorithms, and the hardware team provides hardware compatibility. BitBucket is a version control system that facilitates seamless collaboration within teams. It is a git-based system, whose functionalities are like those of the GitHub system.

Cross-functional cooperation between teams is very important; the analytic team defines and refines algorithms, and the hardware team provides hardware

compatibility. Bitbucket is a version control system that facilitates seamless collaboration within teams. It is a git-based system, whose functionalities are like those of the GitHub system. Bitbucket makes collaborations possible among developers and testers belonging to the same organization and supports collaboration with third-party organizations/ external teams for software development. It's a CI/CD integrated environment which automates, build and test process and it improves efficiency within teams. Bitbucket is mostly integrated with an IDE. Integrated Development Environment (IDE) such as VScode is a lightweight, powerful development environments, which support embedded applications development with extensions for code debugging and refactoring.

3.3 Testing Frameworks and Test Management Tools

The choice of testing framework is largely based on functional requirements. CppUTest and Google Test are generally used for unit testing by the developers (Memon, 2020), while Appium has support for user interface (UI) automation. Fuzz testing tools such as AFL++ help to detect edge case vulnerabilities (Sutton et al.,2007). Tools such as NI LabVIEW for HIL testing allow testing the interaction between software and hardware (Jin et al., 2021). The choice of testing frameworks depends on the number of requirements, such as security requirements and performance in real-time. Efficient and effective test management methods ensure traceability matrix and bug management. Platforms such as Jira and tools such as X-ray provide an environment for test case tracking and bug lifecycle, and reporting. Integrating these structures with CI/CD pipelines ensures automated test execution and report analysis. Aids early detection of bugs and maintains software quality consistency throughout the SDLC

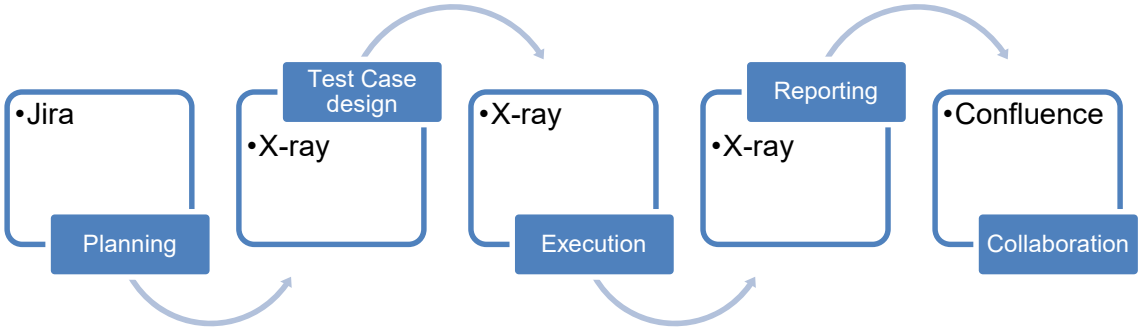


Figure 2. Test Management Tool Ecosystem

Project management workflow, such as Jira, is widely used because it supports Agile methodology in practice on it kanban board. Team uses test management tools because its workflow supports a traceability matrix of task tickets, bugs, and test coverage. The integration workflow with Bitbucket ensures collaboration, continuous feedback in the entire development process, and Confluence supports a centralised documentation platform, which aids clear communications and document sharing among teams. X-ray is a Jira paid plugin built for test case management. It has features that support both automated and manual test case/suite execution, with support for bug management and a requirement traceability matrix, and reporting. The lifecycle depicts how bugs travel from their source of discovery to final resolution and aiding visibility of bugs and related issues throughout the software development.

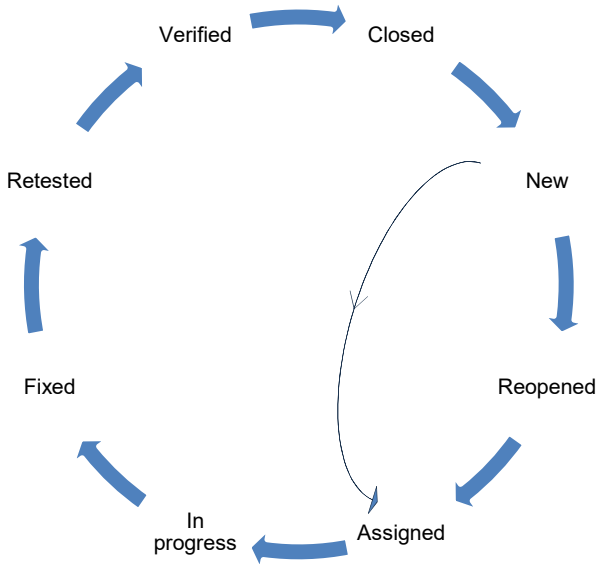


Figure 3. Bug Lifecycle

Confluence functions as a knowledge database, with features that function as a medium for storing documents such as meeting minutes, test plans, technical specifications, and it supports audit planning. It also provides a feature for reporting.

3.4 Test Double for Embedded App

Test double for testing embedded software, it supports interaction between IoT-enabled devices and Windows applications, and the interaction occurs via an IP (Internet Protocol). It enables software testing of embedded applications. According to (Ammann and Offutt, 2016), test doubles simulate hardware interaction without a physical device, ensuring early software testing feedback. Appium is an open-source framework for UI automation. It is widely used for both mobile and embedded applications, and it can equally be adopted for testing IoT-enabled devices with a custom driver. It features cross-platform compatibility. Appium has support for Python client, Java, Ruby, and C#. It also supports other clients such as WebDriver IO, Nightwatch.js, both for Node.js, Robotframework, etc. The Python programming language for testing automation, otherwise known as Python client, uses an extensive library and its flexibility to automate testing tasks, such as unit and system testing.Node.js facilitates server-side scripting and

is used as an IoT communication protocols, which enable real-time data processing. Drivers, such as Appium drivers, ensure communication between hardware and software. Custom drivers are mostly employed where specialized requirements are needed, such as microphones and sensors for an acoustics imaging camera. Custom drivers are designed and developed in-house in compliance with proprietary hardware components to ensure higher testing performance.

3.5 Application deployment

Application deployment includes over-the-air update (OTA), USB deployment, and integration to ensure seamless software distribution (FLIR, 2023). A fleet management system facilitates deployment monitoring.

Fleet management system monitors deployment process, patch update process, update status, and update statistics, ensuring remote access and diagnostics. Customer support team uses tools such as a fleet management system to provide diagnostics services and logs to resolve end-user complaints and ensure long-term reliability.

4 Implementation

4.1 Test Environment Setup

The first thing to do is build a test environment, where the testing of the embedded applications will be tested. This will include installing the necessary testing tools and their dependencies, network and hardware settings to aid reliable and accurate test executions. The hardware was based on the Flir Si2-PD platform, which was made up of a microphone array and a Raspberry Pi that had the capability to handle the processing of acoustic data in real-time. The acoustic chamber was set up to simulate real-world industrial PD conditions, which also includes internal noise and a controlled PD signal from a Sonic tester.

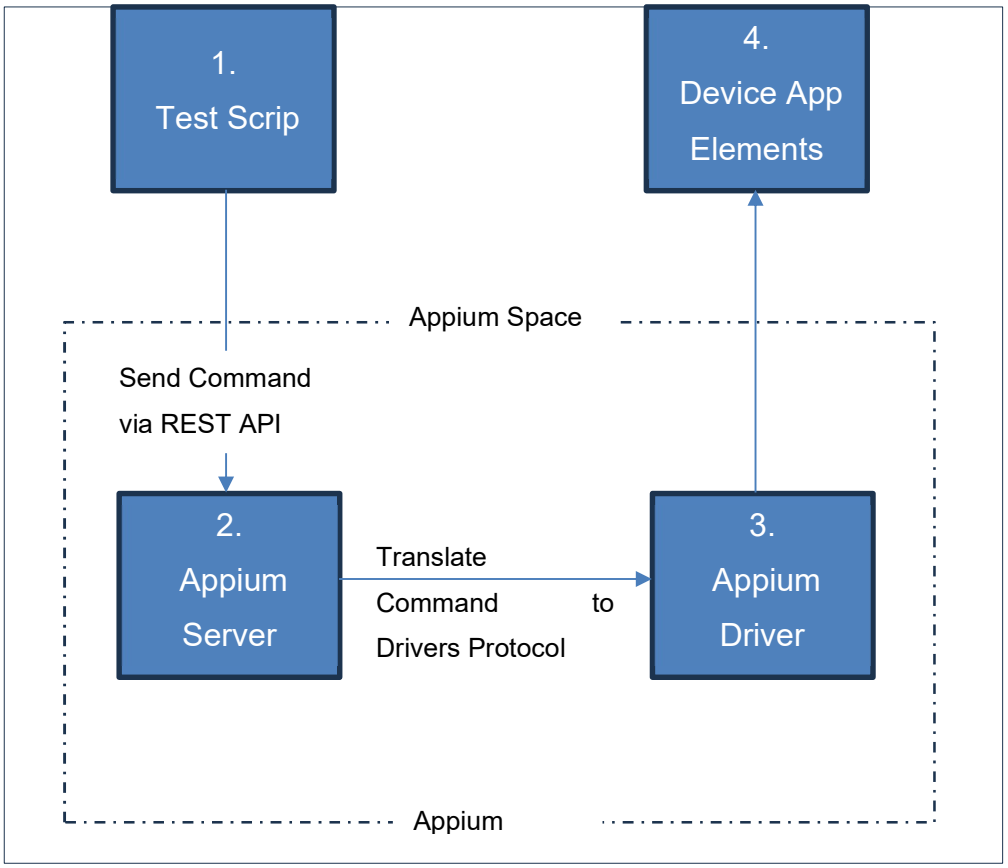


Figure 4. Environmental Setup and How Appium Works

The Test Script creates the test that outlines the actions performed by the device Application. The Appium Server is responsible for receiving commands from the test script; simply put, it is the bridge between the test script and the device Application. Whenever the Appium server receives a communication signal from the test script, the server translates the signal into the WebDriver protocol. The Appium driver then uses the translated protocol to communicate with the device UI elements, such as entering text, clicking buttons, navigating through UI windows, checking and unchecking instructions through screen navigation, illustrating real user scenarios. The software environment is set up in such a way that it includes a lightweight embedded OS, the test software, and a custom Appium test driver. A combination of Python, Selenium library, and Appium library to perform data collection and automated test execution. The acoustic PD signal was generated using a Sonic tester that was turned to PD. The setup helps to separate software anomalies from hardware behaviour, and the procedure ensures repeatability of test execution.

4.2 Test case design

The focus of test case design is to cover both non-functional and functional sections of the software. Each embedded software component was divided into testable units, such as signal acquisition, feature extraction, data preprocessing, and communication. For instance, data preprocessing was tested for its ability to filter out unwanted noise, and the signal acquisition module was tested for sampling accuracy. A stress test was also included to validate the system's behaviour in different weather conditions. Automated cases are written using a mix of white and black box techniques. The black box technique helps to validate system functionality from a user's point of view, while the white box technique aids developers to validate the logic in the code. Each test case is designed with a clear test objective, input conditions, steps, data, expected output, and pass and fail criteria. The stated design made it possible to automate test cases and track the output.

4.3 Test Plan

A test plan aims to lay out the testing strategies involved for testing the embedded software. It describes the testing scope, objectives in the test, schedule, and the resources allocated for every testing phase, and an efficient method to execute the testing lifecycle. Priority was given to the critical aspects of the software, such as signal processing and classification, because failure in these important modules will directly impact the camera's ability to detect either PD or LD signals. The plan also includes the progress made in the system-level test.

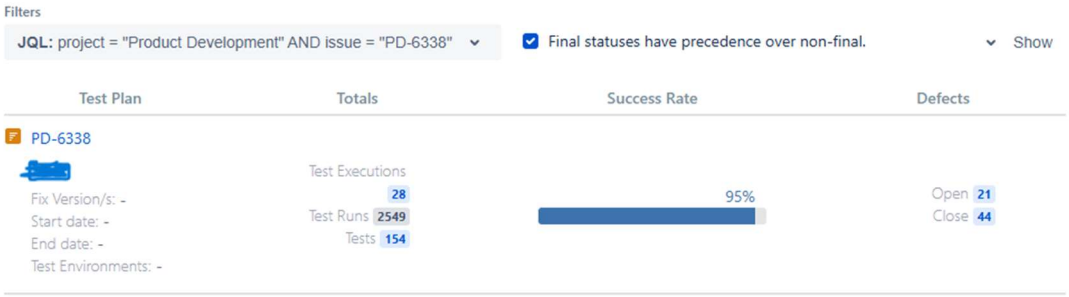


Figure 55. Test Plan

The testing phase was divided into three phases, namely development, integration, and field testing. The development phase focuses on validating how the individual modules are performing in isolation, the integration phase validates how the modules function together, and field-testing checks how the system is performing in real-life scenarios. Each phase has its exit criteria, and the deliverables are reviewed at every stage to ensure that the quality targets stay on course.

4.4 Test Case

The test case was written in a structured format, which includes test case Identification (ID), precondition, description, test steps, expected results, and the actual result. For instance, a test case can be said to verify that the device can detect a PD signal at a frequency of 10kHz within 100 milliseconds, and another

test case could also verify whether the device can ignore unwanted internal and external noise below a specified threshold. The two cases are important to validate both system performance and accuracy.

Key	Summary	Test Type	Status	Status
PD-6528	DEVICE REGISTRATION SHOULD BE PREVENTED IF NO...	Manual	TO DO	PASSED
PD-6526	Verify That The Device Wait for response about pairing...	Manual	TO DO	PASSED
PD-6525	'M' FILTER FREQUENCY LIMITS PERSIST ACCROSS REB...	Manual	TO DO	PASSED
PD-6472	Test manual frequency selection in cloud for PD snaps...	Manual	TO DO	TO DO
PD-6449	VERIFY THE FREQUENCY RANGES FOR 'M' FILTER IN T...	Manual	TO DO	PASSED
PD-6446	VERIFY THAT MANAUAL DISTANCE IS ADJUSTABLE	Manual	TO DO	PASSED
PD-6443	VERIFY THAT PD SOURCE WITH CORRECT CLASSIFICAT...	Manual	TO DO	FAILED
PD-6442	VERIFY SUPPORT FOR WiFi-INFORMATION ON QR_CO...	Manual	TO DO	PASSED
PD-6441	VERIFY A KNOWN SIGNAL FREQUENCY SPECTRUM O...	Manual	TO DO	PASSED
PD-6440	VERIFY THAT MANUAL FREQUENCY BANDPASS IS ADJ...	Manual	TO DO	PASSED

Prev 1 ... 6 7 8 ... 34 Next Total 335 issues

Figure 6. Test Cases

Test cases are stored in a shared repository, which ensures easy access for the team members and enables easy test case review and updates. The automated test script pulled the newest test case and ran it on the embedded device. The passed and failed test status was determined from the test execution logs. The setup ensures early feedback as regressions are caught as early as possible, which ensures consistency of each test cycle.

4.5 Test Execution

The test execution consistency was sustained by executing test cases in a controlled environment, ensuring that the test execution results were repeatable and that external interference that could impact the outcome was run in batches of test suites, by grouping the batches by their assigned priority, starting with high, medium, and low priority. The critical features that constitute the core functionality of the device are given high priority. Each test suite was run multiple times to verify reliability and stability, ensuring consistent performance throughout runs. Actual test results cases were logged automatically, and the system was reset in every test run to avoid contamination.

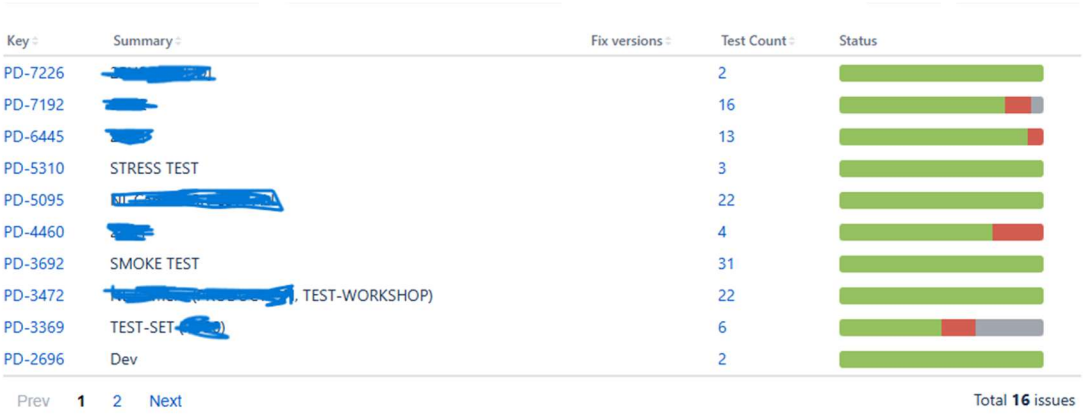


Figure 7.6 Test Executions

During test runs, reports of CPU usage, signal processing latency, and memory consumption were monitored. These data provided valuable insight that helped to determine potential bottlenecks, aid in improving the system efficiency, and ensure smooth operation in different scenarios. A spike in the CPU usage was recorded during the test stress test execution, which led to the optimization of the algorithm. The test objective was not only to pass the test but also to ensure that the system performed well under stress conditions.

4.6 Passed or Failed

Test cases were marked as passed or failed based on predetermined conditions, determined by objective evaluation, and expected results throughout the test run. A test case is marked as passed when the expected result matches the actual result within the acceptable threshold. A test case is marked failed if the expected result differs from the result. To maintain the test objective, only an automated test script can handle the evaluation, and manual judgment was not considered. The failed test case was flagged, and a bug ticket was raised for immediate review and possible fix. Jira dashboard displayed the pass and fail test status of the test executions and trends over time. Trends of failed test cases help to identify recurring bugs and prioritize fixes. There was steady improvement in the pass rate as the code was refined and bugs fixed.

4.7 Retest and Regression Test

After the bugs are fixed, the affected module is retested to confirm that the identified issue has been resolved. The regression test suite was also executed to ensure that the fixes did not break another associated module. It was particularly very important to carry out regression tests on tightly coupled modules such as signal processing and classification, because small changes could have a very high impact on the entire system functionality. Regression test cases include all previously passed test cases to ensure that the system maintains stability as more features are being added. The approach is a proof of system confidence that the software was evolving without introducing new issues while old issues are being fixed and new ones added. t introducing new issues while old issues are being fixed and new ones added.

4.8 End-to-End

Testing and validating the entire system from signal input at a predetermined frequency to the data analysis on the cloud as output. End-to-end testing injects a real acoustic signal into the microphone array, and the PD severity or estimated leak cost estimate is displayed on the cloud dashboard. The objective was to simulate the actual industrial usage scenario and confirm that the entire system responded correctly.

The end-to-end test is carried out by detecting a PD event, classifying it into different PD classifications, such as surface or internal corona, and displaying the severity level. Based on the severity level, the system will display recommendations for maintenance action to avoid downtime of the equipment. End-to-end usability was also performed on the device, which validated user interaction, such as starting a scan and viewing logs. The scenario gave a complete view of the device's performance in real-world conditions.

4.9 Factory Acceptance Testing

The final testing step before the system deployment. A predefined set of sequential tests was executed in the presence of the stakeholders to validate that the system meets all the requirements. Factory acceptance testing (FAT) covered functional, performance, and reliability testing. FAT sequential checklist covered boot-up time, signal calibration, and microphone response to signal detection accuracy. The FAT results were documented, and the passed results were stored. The passed results from FAT mean that the system is ready for production and field deployment.

4.10 Bug Ticket

Each bug that is identified during the testing phase is written as a ticket in the Jira tracking system. Each ticket includes steps to reproduce the bug, description, severity level, attached files such as pictures and or video showing the bug in action, and is assigned to the appropriate developer. The process made bug management easy, aided the debugging process, and ensured accountability.

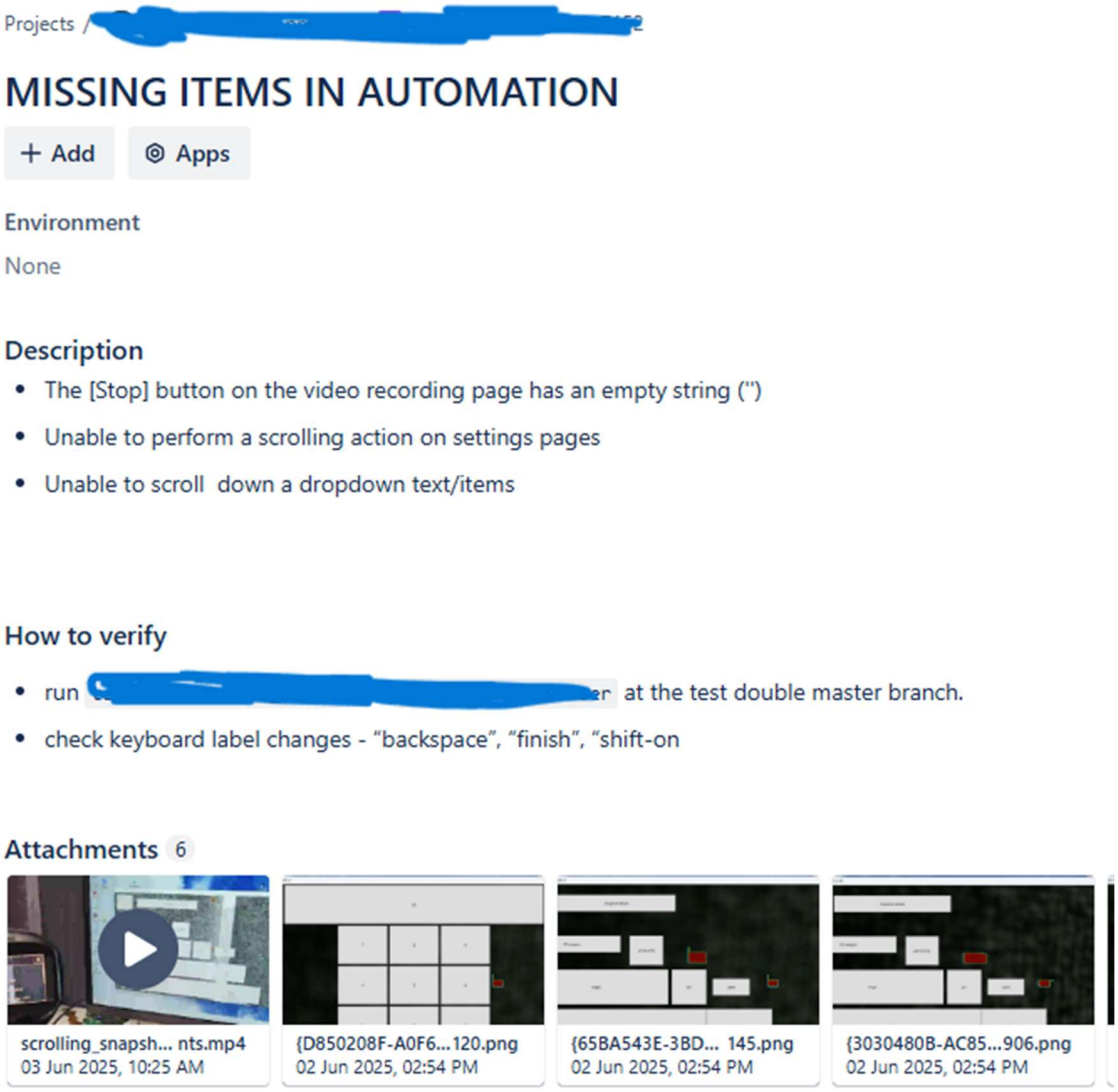


Figure 8. 7 Bug Ticket

Bugs are categorized based on their impact on the entire system. Bugs with high impact were classified as critical; critical bugs were fixed immediately, while medium impacts are also fixed based on priority, and low impact bugs were scheduled for later. Every fix was linked to an associated test case, and fixed bugs were verified during the retesting phase. The entire testing process kept the developmental process organized and enforced accountability within the team.

4.11 Bug tracking

Bug tracking was handled using a lightweight issue tracker, Jira, which was integrated with the IDE and the version control system Bitbucket. It helped to link code commits directly to bug tickets. The system enables visibility into which developer fixed what, and who created which bug ticket, and when it was assigned to whom, and how long the ticket stayed open. The entire traceability matrix of the bug life cycle is known to every team member.
The bug tracker generated a report displaying open, closed, and reopened bugs. The report helped the team to visualise the pattern and used the knowledge to improve the testing strategy. For instance, if a certain module continues to record too many bugs, then it is an indication that such a module requires more attention in the next sprint.

4.12 Test Report

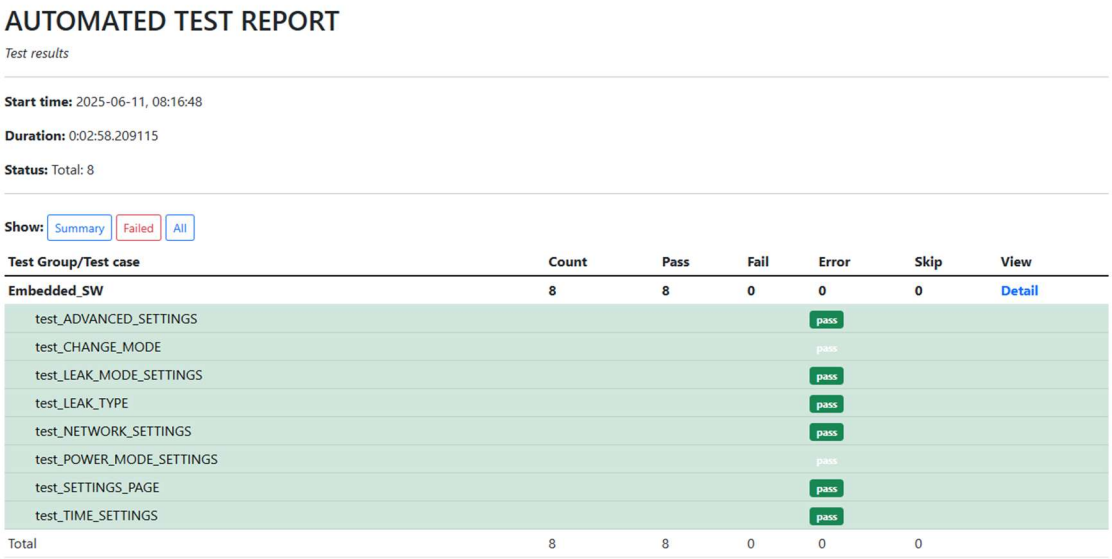
The test report is the last scheme in the testing phase; the detailed test report was compiled, and it contained the entire overview of the test plan, executed results, bug statistics, environmental impact, and lessons learned. The report also contains a list of fixed bugs and recommendations for future improvements.

The report is shared with the necessary stakeholders using Confluence, such as developers, other testers, and the project manager. The most important function of a report is that it shows that the embedded software is ready for deployment in production. It is the records of what was tested, what passed, and what needs further improvement.

5 Results and Analysis

5.1 Test results and performance analysis.

The system testing phase of the embedded software application indicated that the software performs under different test conditions and in different testing environments. The test implementation indicated that Automated test script execution returns with a faster feedback loop than manual test executions. Automated test executions are configured and integrated into the Bitbucket pipeline in such a way that the Test cases run at every code change, without any delay or waiting time. The fast feedback detects Bugs as they are introduced, while manual test execution takes hours or days.



AUTOMATED TEST REPORT

Test results

Start time: 2025-06-11, 08:16:48

Duration: 0:02:58.209115

Status: Total: 8

Show: Summary | Failed | All

Test Group/Test case	Count	Pass	Fail	Error	Skip	View
Embedded_SW	**8**	**8**	**0**	**0**	**0**	**Detail**
test_ADVANCED_SETTINGS				pass		
test_CHANGE_MODE				pass		
test_LEAK_MODE_SETTINGS				pass		
test_LEAK_TYPE				pass		
test_NETWORK_SETTINGS				pass		
test_POWER_MODE_SETTINGS				pass		
test_SETTINGS_PAGE				pass		
test_TIME_SETTINGS				pass		
Total	8	8	0	0	0	

Figure 9. 8 Automated Test Result

During the implementation phase, performance metric was taken, using four key areas or categories. Such as speed, reliability, affordability, and accuracy. Automated test execution shows higher performance in all four categories. For instance, automated test execution runs eight Test Cases in three minutes, while manual testing took 25 minutes for the same number of Test Cases, for the same coverage. Automated test script shown improved reliability of test execution, because script runs are the same in every test execution and test steps are not skipped. The test script runs the same way every time, edge cases are easily

detected than in manual execution testing. The system performance was tracked during the testing implementation. The embedded software maintained stable CPU performance and memory utilization. The device detects around 95% of LD and PD issues, and software crashes were not recorded during the stress test. Stress test indicated that the device was both functional and reliable in a 4-hour stress test.

FACTORS	AUTOMATED TESTING	MANUAL TESTING
Accuracy	Good accuracy for repetitive testing.	Excellent for UX and exploratory testing.
Affordability	High initial cost	Lower initial cost
Reliability	More reliable	Prone to human errors
Speed	Faster	Slower

Figure 10. Manual & Automated Test Script

5.2 Findings: The areas for improvement.

The most significant findings are the time saved in automated test script execution. Manual testing can be advantageous during early testing phases, but it is difficult to scale as the number of test cases increases and the codebase grows. Manual testing can also become a bottleneck to software development as the number of test cases increases exponentially. Automated testing frees up more time for bug fixing for the developers. The automated testing also has some drawbacks. Because some bugs can only be detected during exploratory manual testing, bugs such as UI glitches and latency in the action button or weird sequence of events cannot be detected by automated testing. Small percentages of the test cases are kept as a manual test suite, which can test for the identified unusual behaviours. The weird behaviours can be tested by adding smarter validation logic in the cases and by increasing the tolerance logic. Finally, an automated test script required an initial time investment in setting up the environment and in test script writing.

6 Conclusion and Recommendations

6.1 The Summary of key findings and contributions of the thesis.

This Thesis intends to verify and validate the designed specification of an embedded software application for an IoT-enabled imaging camera. The core designed functionality of the camera detects low-frequency acoustic Partial Discharge (PD) and Air leaks (LD). Through the implementation of different stages of testing strategies, such as unit, integration, and system testing, it was established that the embedded software meets both functional and performance design requirements.

FINAL VERDICT	
Affordability	Manual is cheaper short-term, while Automated saves money long-term.
Flexibility	Manual wins here for exploratory and adaptive testing.
Reliability	Automated Wins due to fewer human errors.
Speed & Accuracy	Automated still wins here due to the repetitive task.

Figure 11. The Final Verdict

The implementation of an automated test script provided a faster feedback loop, better reliability, and reduced cost and time that are associated with manual test execution. Automated test script provided testing repeatability, scalability as the codebase grew, and early bug detection. The development of a HIL (hardware-in-the-loop) testing framework that is tailored to embedded software testing is the second major contribution to this thesis. HIL simulated real real-world industrial acoustics event, which helped to validate the system behaviour under realistic conditions. The thesis revealed the difference between two testing methodologies in terms of speed, cost, reliability, and practical scalability to automated testing methodology.

The recommendations for future work and best practices.

The future testing practices should focus on improving automated test implementation and expanding the testing script to cover edge cases and complex test scenarios, such as the edge case of an acoustics pattern and the end user's device interaction. While automated testing performed better with functional validation, exploratory manual testing played an important role in detecting the system's unexpected behaviour and UI glitches. Integrating artificial intelligence and machine learning to detect abnormal system behaviour, improve system stability, to detect acoustic events more accurately.

WHEN TO USE EITHER	
Automated Test Run	Best for Repetitive, Performance, Large-scale and Regression Testing.
Manual Test Run	Best for Short-term projects, Ad-hoc, Exploratory, and Usability Testing.

Figure 12. When to Use Manual and Automated Test Runs

To the team(s) working or would be working on similar embedded software development and testing, start with test script automation as soon as possible. Automating basic scripts can save hours of manual testing, automated script will also detect regression bugs early in the development. Secondly, a good version control should be maintained with a simple commit phrase, and test cases should be documented. These best practices would make it easier to onboard new testers and developers and maintain consistency within the team. Finally, testing should be treated as a continuous practice, and not as a standalone event during software development. The mind shifts to testing as an integral part of software development can make a big difference in software quality assurance.

References

1. Ammann, P., & Offutt, J. (2016). Introduction to Software Testing.
2. Binder, M. (2023). *Software Testing: Concepts and Techniques*. Springer.
3. Chess, B., & West, J. (2007). Secure Programming with Static Analysis. Addison-Wesley.
4. Choudhary, S. B., Sinha, A., & Gupta, R. (2021). "Challenges in Testing Embedded Software in IoT Devices." *International Journal of Embedded Systems*, 13(3), 281-290.
5. FLIR. (2023). Acoustic Imaging Camera Development Guidelines.
6. Jin, Y., et al. (2021). Hardware-in-the-Loop Testing for Embedded Systems. IEEE.
7. Lin, T., Zhang, Y., & Wang, J. (2019). "Acoustic Imaging and Its Applications." *Journal of Acoustics*, 55(2), 145-160.
8. Memon, A. (2020). Automated Software Testing. Springer
9. Norris, J. (2018). *Testing Embedded Software*. Wiley.
10. Singh, I. M. B. (2022). "Innovative Approaches to Embedded Systems Testing." *Journal of Software Engineering and Applications*, 15(5), 230-245.
11. Sutton, M., et al. (2007). Fuzzing: Brute Force Vulnerability Discovery.
12. Whittaker, J. (2019). How to Break Software.
13. Zhang, L., Chen, X., & Yu, D. (2020). "Applications of IoT in Industrial Monitoring: Problems and Prospects." *IEEE Internet of Things Journal*, 7(3), 1410-1420.
14. Zheng, P., et al. (2018). IoT Security: Challenges and Solutions.

FLIR Si2 Series Documentation:

1. FLIR Systems, Inc. FLIR Si2 User Manual.
2. FLIR Systems, Inc. FLIR Si2-PD User Manual.
3. FLIR Systems, Inc. FLIR Si2-LD User Manual.

Appendix 1 Automated Test Script Environmental Setup