SE 4485: Software Engineering Projects

Spring 2025

Test Plan

| Group Number | 9 |
| --- | --- |
| Project Title | QNX and Python Implementation |
| Sponsoring Company | Communications and Power Industries |
| Sponsor(s) | Luke Carr |
| Students | 1. Muhammad Ali sxa220115@utdallas.edu  2. Nicholas Anderson nba220001@utdallas.edu  3. Tabark Abaid tya210000@utdallas.edu  4. Diego Ibarra dxi200004@utdallas.edu  5. Saghar Abdi sxa190148@utdallas.edu  6. Khaled Elkhaled kye210001@utdallas.edu |

## QNX and Python Implementation Test Plan

#### ABSTRACT

This document presents the test plan for the QNX and Python Implementation project. It outlines the system-level test cases, testing techniques, and criteria for test quality, including black-box and white-box testing. The document provides a mapping of test cases to use cases to ensure all requirements are tested. It also confirms that the test plan is managed under configuration control and adheres to relevant engineering standards. The purpose of this test plan is to establish a structured approach for validating the system’s functionality and performance.

**TABLE OF CONTENTS**

[**QNX and Python Implementation Test Plan 2**](#_tmjilbreipjf)

[ABSTRACT 2](#_mqdxf1nq3ipx)

[LIST OF FIGURES 3](#_8uietdnzkqw7)

[LIST OF TABLES 3](#_nkp8n2naetay)

[INTRODUCTION 4](#_809nn868lr3c)

[REQUIREMENTS/SPECIFICATIONS-BASED SYSTEM LEVEL TEST CASES 4](#_bd23fhsuup63)

[TECHNIQUES FOR TEST GENERATION 4](#_pgks5hv6i3aw)

[TRACEABILITY OF TEST CASES TO USE CASES 6](#_agt2kp7hgrik)

[EVIDENCE THE TEST PLAN HAS BEEN PLACED UNDER CONFIGURATION MANAGEMENT 7](#_mwijqdyvxai)

[ENGINEERING STANDARDS AND MULTIPLE CONSTRAINTS 8](#_w0kpkvur8eaq)

[ADDITIONAL REFERENCES 9](#_ouxdrsqrunjy)

#### LIST OF FIGURES

\*No figures are included in this document.\*

#### LIST OF TABLES

Table 1 Requirements/Specifications-Based System-Level Test Cases 4

Table 2 Traceability of Test Cases to Use Cases 7

#### INTRODUCTION

This document defines the test plan for the QNX and Python Implementation project. The purpose of this test plan is to outline the methodology, approach, and processes for testing the system to ensure it meets all defined requirements and functions correctly. The scope of the plan includes identifying system-level test cases, specifying testing techniques such as black-box and white-box testing, and establishing criteria for evaluating test quality. The document begins with an overview of the system-level test cases derived from project requirements, followed by a description of the testing techniques used. It also covers the traceability of test cases to use cases to ensure comprehensive requirement coverage. Additionally, the plan addresses configuration management to ensure the test plan is maintained throughout the project lifecycle.

#### REQUIREMENTS/SPECIFICATIONS-BASED SYSTEM LEVEL TEST CASES

| Test Case ID | Requirement ID | Description | Input | Expected Output | Pass/Fail Criteria |
| --- | --- | --- | --- | --- | --- |
| TC-001 | RQ-001 | Verify API returns JWT on valid login | POST /login with valid username/password | Response includes a JWT token | JWT token present in response body |
| TC-002 | RQ-001 | Verify API denies access without authentication | GET /device/status without JWT token | 401 Unauthorized | Response code is 401 |
| TC-003 | RQ-002 | Verify Python CLI can send INIT command to C function | `python3 cli.py --init` | C function `init()` is triggered, and returns “Initialized” | Console prints confirmation of successful init |
| TC-004 | RQ-003 | Verify Python CLI can send START command to device | `python3 cli.py --start` | C function `start()` is triggered, and returns “Started” | Console prints confirmation of successful start |
| TC-005 | RQ-004 | Verify C code executes embedded Python script | Run C program calling embedded script that prints output | Output from Python appears in terminal | Printed Python output is correct and visible |
| TC-006 | RQ-005 | Verify REST API accepts START command via POST | POST /device/start with valid JWT token | Device state changes to “running” | JSON response confirms new state |
| TC-007 | RQ-006 | Verify expired JWT is rejected | POST /device/start with expired token | 401 Unauthorized | Response code is 401 |
| TC-008 | RQ-007 | Verify system recovers after Python script crashes | Simulate Python script crash (e.g., exception or segfault) | System logs crash and restarts the script | Script restarts automatically; log shows recovery |
| TC-009 | RQ-008 | Verify throttle value is accepted via REST API | POST /device/throttle with `{"value": 75}` and JWT | Device receives throttle value and adjusts | Response confirms new throttle; device reflects it |
| TC-010 | RQ-009 | Verify GET /device/status returns correct status | GET /device/status with valid JWT | JSON object includes `status`, `uptime`, `last\_command` fields | Response matches current internal device state |
| TC-011 | RQ-010 | Verify memory usage stays under 100MB during runtime | Run full system including C + Python for 10 minutes | Max memory usage ≤ 100MB | Use monitoring tool like `valgrind` or `top`; logs show limit |
| TC-012 | RQ-012 | Verify system logs error when invalid command is sent | Send malformed JSON or invalid CLI command | Log entry records error with timestamp and context | Log file includes entry about the invalid request |

#### TECHNIQUES FOR TEST GENERATION

To create effective test cases on the QNX and Python Integration System, both black-box and white-box testing methods will be applied. As such, these methods would give us an opportunity to analyze the system from two different points of view: functionality (black-box) and internal workings (white-box).

**Black Box Testing:**

* Black box testing is looking at the functionality of the system without knowledge of internal workings. It tests the system as a whole and based only on requirements and specifications. It validates the behavior of the system against user expectations and functional requirements.

1. Test Generation Approach:

* Functional Tests: These will be based on what is functionally important according to the Requirements Documentation. Examples include:
* Verify that the system generates and returns the correct JWT token upon successful authentication (RQ-001).
* Verify that the system handles real-time data processing correctly (RQ-002).
* Verify that the REST API returns correct responses for various HTTP methods (RQ-005).

1. User Scenarios: Use cases and user stories will guide the creation of test scenarios, ensuring that every feature described in the requirements is validated by the tests. For example, we will simulate a user accessing the system via the REST API and check the authentication and data retrieval processes.
2. Error Handling: Test cases will generate to check the graceful handling of all mentioned errors by the system. For example:

* Using invalid input and checking that appropriate messages are provided.
* Recovery mechanisms will be tested for system failure situations, like having the Python script crash.

1. Advantages:

* From a user perspective focusing on system behavior.
* No knowledge is needed about the internal code structure of a system.
* Its case is excellent in verifying E2E scenario functions.

1. Disadvantages:

* Cannot help in identifying issues related to the internal structure of the system, such as logic or inefficiency in the code.

**White-Box Testing**

White-box testing is testing the internal logic and structure of the system. It basically focuses on the code and its paths, by examining whether all internal functions are implemented properly and whether the system behaves according to expectations on a detailed level.

Test Generation Method:

1. Code Coverage: We will ensure that all branches, conditions, and paths of the code of the system are tested using code coverage tools. This includes testing the Python-C interoperability functions, making sure that Python functions are called from C correctly and vice versa.

* For example, PyObject\_CallObject() and PySys\_SetArgv() should be tested under the following conditions.

1. Path Testing: Tests will be designed to ascertain that every execution path truly is executed including critical paths that involve all the respective areas of system interaction between Python, C, and QNX.

* This means that data and flow must be tested between python and c functions to verify that the information has been passed and returned correctly.

1. Unit Testing: Tests concentrating on individual components or parts of the system. For example, unit tests will be produced for certain Python functions related to data processing or for the functions interfacing QNX hardware.
2. Error Handling Tests: Identify how the system deals with faults at the code level and whether recovery mechanisms are available in the event of unforeseen failures such as in Python scripts or API calls.
3. Advantages:

* Provides insight into how its internals and weaknesses potentially behave.
* Performance bottlenecks, security loopholes, or logical flaws in code can be detected.
* It ensures that all code paths are exercised and checked.

1. Disadvantages:

* Requires extensive knowledge in the code base.
* Might not be possible to cover every user scenario through such testing or fully capture the usage scenario across an end-to-end system.

**Criteria for Measuring the Quality of Tests**

To ensure the quality of our test cases, we will use the following criteria:

1. Coverage:

* Code Coverage: Coverage of the code in overall tests (functions branches loops).
* Requirement Coverage: There will be test cases for every requirement so that each requirement in the documentation is well documented.

1. Defect Detection Rate: The ratio of defects found in testing to those found later in post-release. Higher is better in regard to the test quality.
2. Test Case Effectiveness:

* True Positive Rate: Defect detection
* False Positive Rate: Reporting of failure which is incorrect.

1. Efficiency:

* Test execution time: Testing effectively gives maximum coverage in the least possible time.
* Resource Utilization: Not all tests should be resource consumptive.

1. Maintainability:

* Low Modification Cost: All these tests could change as the system changes.
* Reusability: These tests should be reused over different modules.

1. Traceability:

* Ensuring each test case is traceable to a specific requirement.

1. Robustness:

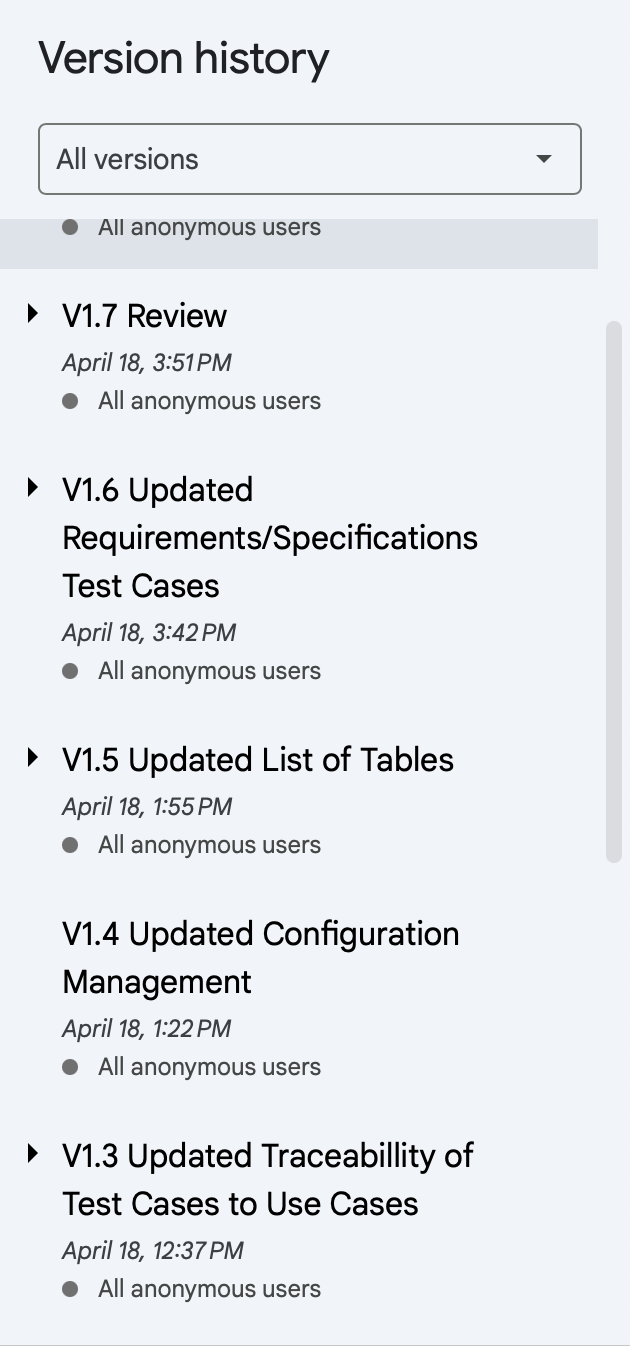
* The tests should check for proper input error handling and ensure the system recovers from unexpected inputs.

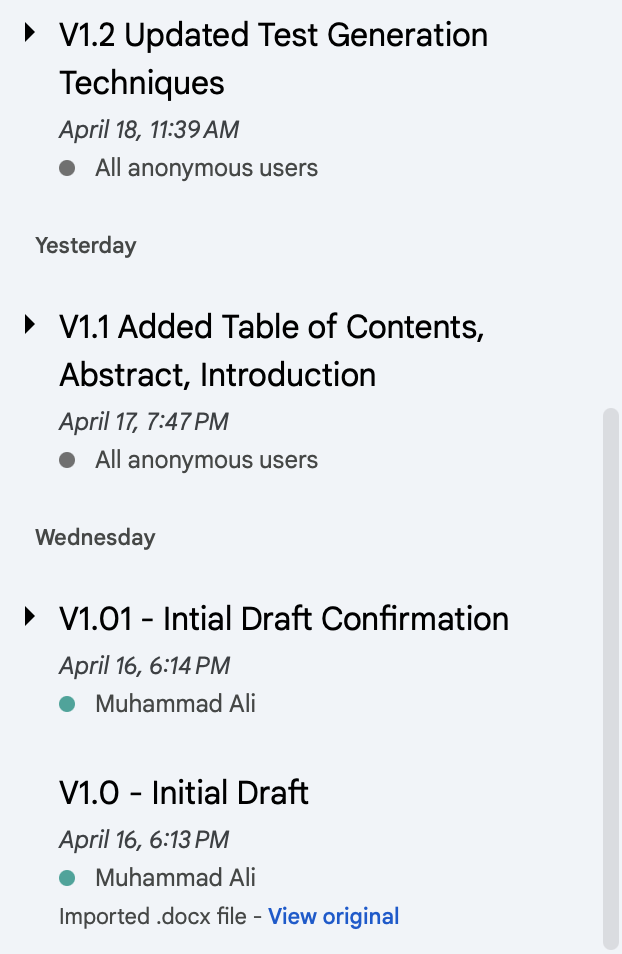
#### TRACEABILITY OF TEST CASES TO USE CASES

* provide a mapping between test cases and use cases
* clearly describe how each requirement in the *Requirements Documentation* is captured in testing

| **Use Case** | **Requirement ID** | **Description** | **Related Test Cases** |
| --- | --- | --- | --- |
| Use Case 1: Execute Python Script from C | RQ-004 | C code must execute a Python script via the Python C API | TC-005 |
| Use Case 2: Control Device from Python via C | RQ-002, RQ-003 | Python CLI sends commands like INIT, START, THROTTLE to control the device by calling C functions | TC-003, TC-004 |
| Use Case 3: REST API Receives Device Control Requests | RQ-005, RQ-008, RQ-009 | REST API endpoints allow clients to control the device remotely | TC-006, TC-009, TC-010 |
| Use Case 4: Authentication via JWT | RQ-001, RQ-006 | API must return a JWT token on successful login and restrict access for unauthenticated users | TC-001, TC-002, TC-007 |
| Use Case 5: Diagnostics and Error Reporting | RQ-005, RQ-012 | System provides diagnostics and logs or reports any device errors | TC-010, TC-012 |
| Use Case 6: Resilience to Python Crashes | RQ-007 | If a Python script crashes during runtime, the system should automatically recover or restart the process | TC-008 |
| Use Case 7: Resource Usage Constraint | RQ-010 | Python runtime must stay under 100MB of memory usage to ensure performance on embedded hardware | TC-011 |

#### EVIDENCE THE TEST PLAN HAS BEEN PLACED UNDER CONFIGURATION MANAGEMENT





| Version | Date | Author | Changes Made | Previous Versions | Reviewers | Ship-it |
| --- | --- | --- | --- | --- | --- | --- |
| v1.7 | 04/18/2025 | Diego Ibarra, Muhammad Ali, Nicholas Anderson | Review | v1.6 | Tabark Abaid, Saghar Abdi, Khaled Elkhaled | Approved |
| v1.6 | 04/18/2025 | Khaled Elkhaled, Nicholas Anderson | Updated Requirements/Specifications Test Cases | v1.5 | Tabark Abaid, Saghar Abdi | Approved |
| v1.5 | 04/18/2025 | Diego Ibarra, Tabark Abaid | Updated Tables | v1.4 | Khaled Elkhaled, Muhammad Ali, Saghar Abdi | Approved |
| v1.4 | 04/18/2025 | Muhammad Ali, Khaled Elkhaled | Updated Configuration Management | v1.3 | Nicholas Anderson, Diego Ibarra | Approved |
| v1.3 | 04/18/2025 | Saghar Abdi, Diego Ibarra, Muhammad Ali | Updated Traceability of Test Cases to Use Cases | v1.2 | Tabark Abaid, Nicholas Anderson | Approved |
| v1.2 | 04/18/2025 | Tabark Abaid, Nicholas Anderson | Updated Test Generation Techniques | v.1.1 | Khaled Elkhaled, Muhammad Ali, Saghar Abdi | Approved |
| v1.1 | 04/17/2025 | Khaled Elkhaled, Saghar Abdi | Added Table of Contents, Abstract, Introduction | v1.01 | Muhammad Ali, Diego Ibarra | Approved |
| v1.01 | 04/16/2025 | Muhammad  Ali | Draft Confirmation | - | - | - |
| v1.0 | 04/16/2025 | Muhammad Ali | Initial Draft | - | - | - |

#### ENGINEERING STANDARDS AND MULTIPLE CONSTRAINTS

* + IEEE Std 829-1983: Software Testing [[pdf](https://course.techconf.org/se4485/IEEE/IEEE%20Std%20829-1983-Software-Testing.pdf)]
  + ISO/IEC/IEEE Std 29119-1-(Revision-2022): Part 1 - Software Testing General Concepts [[pdf](https://course.techconf.org/se4485/IEEE/IEEE-Std-29119-1-(Revision-2022)-Software-Testing-General-Concepts.pdf)]
  + ISO/IEC/IEEE Std 29119-2-(Revision-2021): Part 2 - Test Process [[pdf](https://course.techconf.org/se4485/IEEE/IEEE-Std-29119-2-(Revision-2021)-Test-Process.pdf)]
  + ISO/IEC/IEEE Std 29119-3-(Revision-2021): Part 3 - Test Documentation [[pdf](https://course.techconf.org/se4485/IEEE/IEEE-Std-29119-3-(Revision-2021)-Test-Documentation.pdf)]
  + ISO/IEC/IEEE Std 29119-4-(Revision-2021): Part 4 - Test Techniques [[pdf](https://course.techconf.org/se4485/IEEE/IEEE%2029119.4%20(2021)%20-%20Test%20Techniques.pdf)]
  + Additional standards suggested by the sponsor(s)

#### ADDITIONAL REFERENCES

* + Jorgensen, P.C., 2013. *Software Testing: A Craftsman's Approach.* Auerbach Publications
  + Mathur, A.P., 2013. *Foundations of Software Testing, 2/e.* Pearson Education
  + Additional references suggested by the sponsor(s)