Practical Assessment Task

Phase 4.2 – Testing Document

Igor Karbowy

Table of Contents

4.2.1 — Evaluation of the Programmed Solution

In general, the programmed solution meets the requirements set out in section 1.3. All the functionality outlined in that section was successfully implemented in *PingPal*. This is not to say that the program is perfect—a lot of the functions could be slightly improved.

The first function outlined in 1.3 was subnet scanning. This function was fully met. The project includes the *SubnetScan* class that takes a network range, scans all IPs in the subnet concurrently, collects reachable IP addresses into *SubnetScanResult* objects, and displays results in the UI table while updating a progress bar. However, this functionality could be improved by adding optional reverse DNS resolution (hostname), and an optional “device info” mode (e.g., basic service/banner checks) so users see more than just IP addresses.

The second function outlined in 1.3 was device pinging. This function was fully met. The program contains the *DevicePing* class, which repeatedly pings a target IP at a user-configurable interval and count (or continuously), records round-trip times and successful/failed pings, computes packet loss and summary statistics, and presents both per-ping rows and aggregated summary tables in the UI. However, this functionality could be improved by providing the user with richer statistics (median, standard deviation, percentiles) and an option to export intermediate results so long runs can be resumed or recovered.

The next function outlined in 1.3 was port scanning. This function was fully met. The program implements the *PortScan* class, which scans a user-specified port range against a target IP, detects open ports, maps open ports to protocol names (via the *Protocols* class), stores results as *PortScanResult* objects, and updates the UI table and progress bar. However, this functionality could be improved by creating additional functionality which allows for optional banner-grab or service probing for discovered ports.

The next function outlined in 1.3 was TCP message functionality. This function was fully met. The two TCP messaging classes—the *TCPMessageListen* class which opens a server socket on a user-chosen port, then waits for a client; and the *TCPMessageConnect* class connects to a user-specified IP and port—ensure that *PingPal* meets this requirement. Both classes enable for the receiving and displaying of incoming messages in the UI and allowing for the sending of replies, with formatted time/host display, and a means to stop the session. However, this functionality could be improved by adding multi-client support or a client list UI for broader use cases and/or an option to save chat sessions automatically.

The second to last function outlined in 1.3 was exporting information to and importing information from JSON files. This function was fully met. The *ExportResults* class can export *SubnetScan*, *DevicePing*, and *PortScan* data to JSON files with appropriate structures, and writes TCP message sessions to plain text files for archival use. The *ImportResults* class detects whether an uploaded JSON file contains subnet scan, device ping, or port scan results, validates required fields and types, parses data into result objects, and notifies the UI to display the imported results. However, this functionality could be improved by including simple metadata in exported files (export timestamp, app version, file schema version) to make future imports and provenance tracking easier.

The last function outlined in 1.3 was error handling. This function was fully met. The project includes a *ValidationUtils* class used across the UI and import routines to perform presence, format, type, range, and logic checks. UI components show field errors and dialogues to provide the user with feedback on invalid input or file problems. However, this functionality could be improved by adding a simple centralised log file for validation and runtime errors to aid troubleshooting and to capture any bugs that may reveal themselves in the future.

In conclusion, all functions listed in section 1.3 are implemented and demonstrably met by the program components and UI flows described above. The core features (subnet scanning, device pinging, port scanning, two-way TCP messaging, JSON export/import, and robust validation) are present. Suggested enhancements are mostly user-orientated—richer device metadata (hostnames, banners), improved export metadata/versioning, optional extended statistics, and multi-client/more robust connection behaviours. These will increase usability and diagnostic value without changing the core functionality the program already meets.

4.2.2 — Functional Testing

PingPal was tested by the same individual on two separate occasions. This was done to mark the progress of the development of the application. The results of the two test are found below. Each result provides: (a) the name of the tester; (b) the date of the test; (c) the results from the test—

A. **Test One—**

  a. Neo Jordaan

  b. 4 May 2025

  c. The following functions were tested—

    i. **Subnet scanning—**

    The test performed was to start a subnet scan from the UI and watch UI responsiveness while the scan runs. The expected result was for the scan to run while the UI remained responsive, the progress updates to be shown, and for the discovered IPs to appear incrementally.

    However, the actual result presented that the executed, but the UI froze while the scan ran. There was no live feedback while work proceeded. This indicates that the program currently does not fulfil the subnet scanning functionality.

    ii. **Device pinging—**

    The test performed was to run a ping test for a target with a ping interval and count, and to then observe the UI and results. The expected result was for repeated pings to be performed, per-ping results appear to appear as the test is running, and for the packet loss and summaries to be calculated and displayed at the end.

However, the actual result showed that the pinging executed, but the UI froze during the test (same symptom as subnet scan). This indicates that the program currently does not fulfil the device pinging functionality.

iii. **Port scanning—**

The test performed was to start a port scan across a small range of ports and then to observe the progress and results. The expected results were for the open ports appear incrementally, the progress bar to advance as the scan occurs, and for the UI to stay interactive.

However, the actual result presented that the port scan ran, but it caused the application to freeze, and no live progress was shown during the duration of the scan. This indicates that the program currently does not fulfil the port scanning functionality.

iv. **TCP message listening—**

This feature has not been implemented at the time of testing. Therefore, it automatically means that the program currently does not fulfil the TCP message listening functionality.

v. **TCP message connection—**

This feature has not been implemented at the time of testing. Therefore, it automatically means that the program currently does not fulfil the TCP message connection functionality.

vi. **Exporting results—**

The test performed was to export sample results from a scan and chat to file via the UI. The expected results were for the file chooser to work, and for the exported data to be written and saved successfully to a JSON/text file.

In that sense, the export functionality worked in the early test. The expected results were achieved. This indicates that the program currently does fulfil the export results functionality.

vii. **Importing results—**

The test performed was to import a prepared JSON file into the application and to observe the validation process and whether the results are displayed to the tables. The expected result was that the imported file should be validated and safely presented, and that any invalid files should be rejected with clear error.

The actual result showed that the importing functionality worked, and the data was loaded. However, there was no validation, so malformed data could crash the app. This indicates that the program currently only partially fulfils the import scanning functionality.

viii. **Error handling—**

The test performed was to enter an invalid IP, blank fields and invalid ranges in the UI. The expected result was for the UI to prevent any invalid inputs, to highlight problems and to avoid crashes.

However, the actual results showed that many of the validations are not in place or are inconsistent. In some instances, the wrong dialogues appeared at the wrong times. This indicates that the program currently does not fulfil the error functionality.

Overall, at the initial testing stage, the application implemented the core functionality (scans, pings, port checks, messaging, import/export) but several critical issues prevented the program from delivering a proper user experience. The main functional failure was that long-running network tasks were executed on the UI thread, causing the application to freeze while scans and ping operations ran. The import functionality technically worked, but lacked input validation, risking crashes when malformed JSON was provided. Additionally, there were several usability and correctness problems such as wrong error dialogues and minor UI behaviours (e.g., text fields not clearing). These issues explain why, although the core features existed, the program did not meet the functional expectations for responsiveness, safe import, and consistent user experience at this early stage.

B. **Test Two—**

a. Neo Jordaan

b. 16 August 2025

c. The following functions were tested—

i. **Subnet scanning—**

The test performed was to start a subnet scan from the UI and watch UI responsiveness while the scan runs. The expected result was for the scan to run while the UI remained responsive, the progress updates to be shown, and for the discovered IPs to appear incrementally.

The actual result showed that the subnet scan runs on a background thread. This resulted in the UI staying responsive, the IP addresses appearing in the table progressively, and the progress bar advancing. This indicates that the program finally does fulfil the subnet scanning functionality.

ii. **Device pinging—**

The test performed was to run a ping test for a target with a ping interval and count, and to then observe the UI and results. The expected result was for repeated pings to be performed, per-ping results appear to appear as the test is running, and for the packet loss and summaries to be calculated and displayed at the end.

The actual results showed that the ping loop is executed on a background thread. This meant that the per-ping rows and summary tables were being updated correctly. The continuous mode and finite mode behaved as expected. This indicates that the program finally does fulfil the device pinging functionality.

iii. **Port scanning—**

The test performed was to start a port scan across a small range of ports and then to observe the progress and results. The expected results were for the open ports appear incrementally, the progress bar to update as the scan occurs, and for the UI to stay interactive.

The actual results showed that the port scanning performed concurrently on a separate thread to the UI thread. This meant that the open ports were being displayed live. The protocols were resolved to the corresponding port numbers, and the table updated in real time. This indicates that the program finally does fulfil the port scanning functionality.

iv. **TCP message listening—**

The test performed was to start a server listen, accept any client that is trying to connect, exchange messages, and verify UI behaviour. The expected results were for the server to listen off the UI thread, for the messages to be displayed with a timestamp and host tags, and for the input message to be cleared after the send button is pressed.

The actual results showed that the server accepted the client, and ran the messaging loop off the UI thread. The messages were formatted and displayed correctly. The input cleared reliably, and there were no UI freezes. This indicates that the program finally does fulfil the TCP message listening functionality.

v. **TCP message connection—**

The test performed was to connect to server, exchange messages, and verify UI behaviour. The expected results were for the client to connect off of the UI thread, for the messages to be displayed correctly, and for the UI to be responsive.

The actual results showed that the client connection was performed on background thread. The messages were received and displayed correctly, and the connection errors were reported clearly. This indicates that the program finally does fulfil the TCP message connection functionality.

vi. **Exporting results—**

The test performed was to export sample results from a scan and chat to file via the UI. The expected results were for the file chooser to work, and for the exported data to be written and saved successfully to a JSON/text file.

The actual results showed that the export functionality worked for saving the subnet scan, device ping, port scan to JSON files and the TCP message chats to text files. Success dialogues were shown, and error dialogues were correctly shown when an error occurred. This indicates that the program still does fulfil the export results functionality.

vii. **Importing results—**

The test performed was to import a prepared JSON file into the application and to observe the validation process and whether the results are displayed to the tables. The expected result was that the imported file should be validated and safely presented, and that any invalid files should be rejected with clear error.

The actual results showed that the import results functionality validates the scan type, scan format, variable presence, types, and format, as well as, variable logic before importing. Valid files are loaded into correct tabs, and the UI is correctly populated. Invalid files are rejected with specific errors. This indicates that the program finally does fulfil the export results functionality.

viii. **Error handling—**

The test performed was to enter an invalid IP, blank fields and invalid ranges in the UI. The expected result was for the UI to prevent any invalid inputs, to highlight problems and to avoid crashes.

The actual test showed that the validation utilities were applied consistently across the UI. In the case of errors, user inputs were highlighted and error dialogues were shown. There were no unhandled exceptions from bad input. This indicates that the program finally does fulfil the error handling functionality.

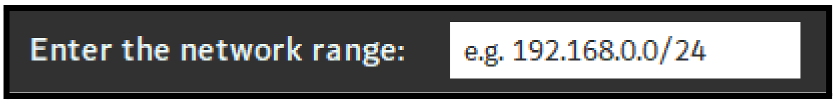
Overall, all program functions listed in section 1.3 have been implemented and validated in the final test. The scans (subnet, ping, port) run concurrently without blocking the UI. The TCP messaging supports both listen and connect modes, with formatted chat display. The import/export functionality works with correct validation and clear feedback, and input validation prevents invalid user actions while preserving stability.

4.2.3 — Test Plan and Results for Input Variables

The two input fields selected for the purpose of the data validation testing are—

A. **txfNetworkRange—**

The following shows the field before any test—

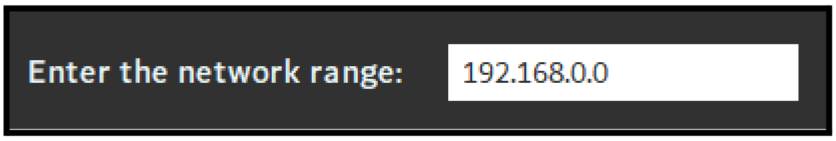

Enter the network range: e.g. 192.168.0.0/24

Example data is displayed in the text field.

The following shows the results of the data validation tests regarding a—

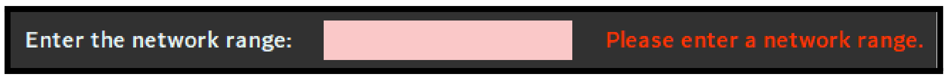
a. **Presence Check—**

i. <u>Normal</u>


Enter the network range: 192.168.0.0

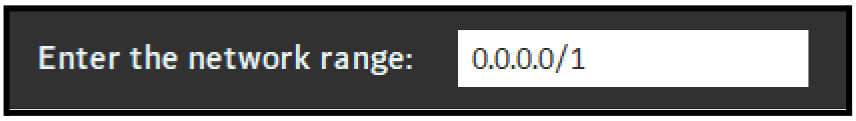
No error message is displayed.

ii. <u>Abnormal</u>


Enter the network range: Please enter a network range.

The following error message is displayed when the field is left empty or contains a white space(s), asking the user to enter a network range.

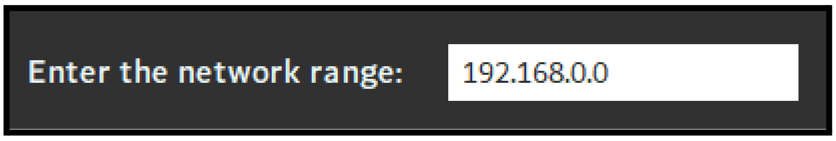
iii. <u>Extreme</u>


Enter the network range: 0.0.0.0/1

No error message is displayed.

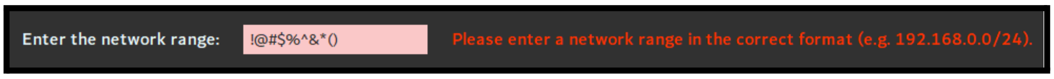
b. **Format Check—**

i. <u>Normal</u>


Enter the network range: 192.168.0.0

No error message is displayed.

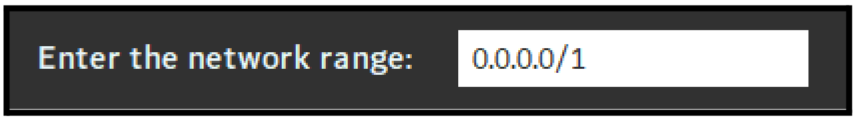
ii. Abnormal



Enter the network range: !@#$%^&*() Please enter a network range in the correct format (e.g. 192.168.0.0/24).

The following error message is displayed when a value does not match the set format, asking the user to enter a network range following the correct format.

iii. Extreme

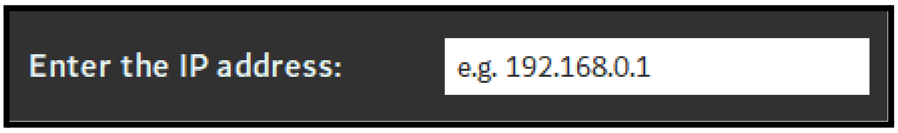


Enter the network range: 0.0.0.0/1

No error message is displayed.

B. **txfIPAddressPortScan—**

The following shows the field before any test—



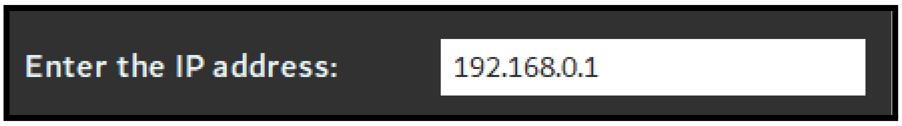
Enter the IP address: e.g. 192.168.0.1

Example data is displayed in the text field.

The following shows the results of the data validation tests regarding a—

a. **Presence Check—**

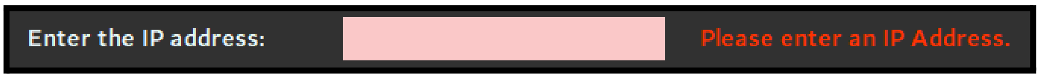
i. Normal



Enter the IP address: 192.168.0.1

No error message is displayed.

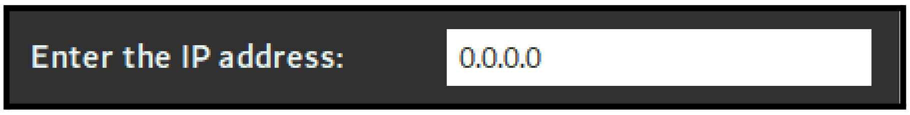
ii. Abnormal



Enter the IP address: Please enter an IP Address.

The following error message is displayed when the field is left empty or contains a white space(s), asking the user to enter an IP address.

iii. Extreme

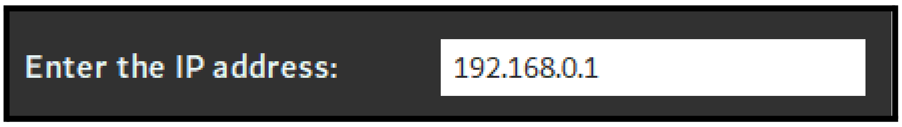


Enter the IP address: 0.0.0.0

No error message is displayed.

c. **Format Check—**

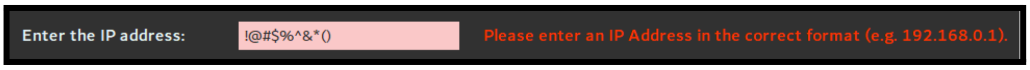
i. Normal



Enter the IP address: 192.168.0.1

No error message is displayed.

ii. Abnormal

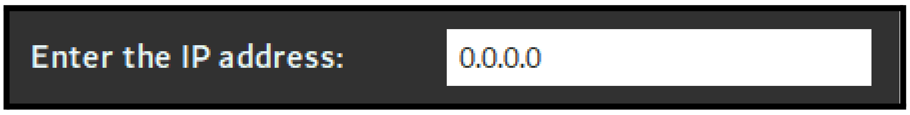


Enter the IP address: !@#$%^&*()

Please enter an IP Address in the correct format (e.g. 192.168.0.1).

The following error message is displayed when a value does not match the set format, asking the user to enter a network range following the correct format.

iii. Extreme



Enter the IP address: 0.0.0.0

No error message is displayed.