



THE UNIVERSITY OF QUEENSLAND

Enhanced Rail Test Automation

Student Name: Isha, JOSHI

Course Code: ENGG7290

Supervisor: Graeme Smith – Associate Professor
School of Information Technology and Electrical Engineering

Submission date: 25th June 2020

EXECUTIVE SUMMARY

Hitachi Rail STS (Hitachi) has been contracted to deliver and maintain an Automated Train Operation (ATO) system for Rio Tinto Iron Ore as a part of their AutoHaul® project. This ATO system facilitates the driverless movement of trains in a railway network in Western Australia. Because of its safety critical nature, any modifications made to the AutoHaul® system require extensive testing before they can be rolled out. Presently, this testing is performed manually and uses a lot of testing time and resources. Hitachi has commissioned the design and development of an automated testing tool which can be used to reduce the testing time and improve testing efficiency. The testing would focus on the communication interfaces between various AutoHaul® sub-systems.

The primary aim of this project was to design and develop an integrated testing framework to support the automated testing of the AutoHaul® project. The testing focused on testing the behaviour of the Fiber Interface Processor (FIP) and the Train Control Sub-system (TCS) by testing the communication interfaces between these sub-systems and other AutoHaul® sub-systems.

This report provides an overview of the AutoHaul® system and details about the specific interfaces that testing is being automated for. A review of literature relevant to designing an automated testing framework is also present.

This report also delivers a summary of the design and outcomes of the tool used to test the FIP's behaviour. This tool, called the FIP Tester, simulates the behaviour of the Interlocking (IXL) devices that connect to the FIP. It verifies that the FIP is responding correctly to messages sent by the IXL simulators in order to ensure that the FIP is behaving correctly. Furthermore, the report provides a summary of the testing activities conducted to validate the behaviour of the FIP Tester.

Similarly, this report provides a summary of the design and project outcomes of enhancing a pre-existing tool, called the TCS Testing Suite. Prior to the project, this tool was developed at Hitachi to begin to automate the TCS testing. It contains the testing framework required to automate the testing procedure that is conducted by Hitachi.

During this project, the TCS Testing Suite's capabilities were enhanced to make it capable of running more complex tests. Additionally, a Log Parser which iterates through various log files and consolidates all of the errors into a single file was developed to aid users in troubleshooting while using the TCS Testing Suite. The report also provides a summary of the testing activities performed to validate the behaviour of the TCS Testing Suite enhancements and the Log Parser.

Overall, the developed FIP Tester and the enhanced TCS Testing Suite can be reliably used to replace a lot of the manual testing activities conducted at Hitachi.

ACKNOWLEDGEMENTS

I would like to sincerely thank a number of people who have helped me throughout this project.

Firstly, I would like to thank my supervisor Associated Professor Graeme Smith for the guidance and advice that was provided throughout the semester which helped shape this project.

I would also like to thank the EAIT Employability team and Dr Christopher Leonardi at the University of Queensland for all of their help in organising my placement at Hitachi Rail STS.

I would also like to thank Hitachi Rail STS, the host company, for allowing me the opportunity to work on this project and learn about the railway industry and the AutoHaul® project. In particular I would like to express my gratitude to my supervisory team at Hitachi, Dr Anthony MacDonald, Lionel Van Den Berg and most of all Michal Cedrych for all of the support and advice I was given throughout the course of the project. I would also like to thank Ujas Soni, Andrew Mijat and Benjamin Mountford at Hitachi for all of the technical advice and support they gave me throughout the project.

TABLE OF CONTENTS

Executive Summary.....	III
Acknowledgements.....	IV
List Of Figures.....	VII
List Of Tables.....	VIII
1.0 Introduction	1
1.1 Placement Context.....	1
1.2 Placement Purpose	1
2.0 Technical Background	2
2.1 The AutoHaul® Project.....	2
2.2 Interface Details And Current Testing Methods	5
2.2.1 Interlocking – FIP Interface	5
2.2.2 TCS Interfaces.....	7
2.3 Literature Review	10
2.3.1 Designing a Test Automation Framework.....	11
2.3.2 Evaluating Testing Tools.....	12
2.3.3 Designing Test Cases	12
2.3.4 Summary of Literature Review	13
3.0 Project Description.....	14
3.1 Aims And Objectives	14
3.2 Scope.....	14
3.3 Project Deliverables	16
3.4 Project Management	16
4 Design.....	17
4.1 FIP Tester	17
4.1.1 Test Framework Design.....	17
4.1.2 Test Case Design	19
4.1.3 GUI Design.....	23
4.1.4 Performance Assessment Design.....	24
4.1.5 Documentation	24
4.2 TCS Testing Suite	25
4.2.1 Increasing Testing Capabilities.....	26
4.2.2 Log Parser.....	27
4.2.3 Performance Assessment.....	27
4.2.4 Documentation	28

5	Project Outcomes.....	29
5.1	FIP Tester	29
5.1.1	The Testing Tool	29
5.1.2	The GUI.....	30
5.1.3	Test Cases.....	30
5.1.4	Performance Assessment.....	30
5.1.5	Documentation	34
5.2	TCS Testing Suite	34
5.2.1	The Enhanced TCS Testing Suite	34
5.2.2	The Log Parser.....	36
5.2.3	Performance Assessment.....	36
6	Conclusion And Recommendations	39
6.1	FIP Tester	39
6.2	TCS Testing Suite	39
6.3	Project Improvements	40
7	References.....	41
8	Appendix A: Project Management Summary.....	44
8.1	Project Timeline And Resources	44
8.1.1	Background Research Stage.....	44
8.1.2	Design Stage.....	44
8.1.3	Implementation Stage.....	44
8.1.4	Documentation Stage	45
8.1.5	Project Timeline	45
8.2	Risk Analysis	47
8.2.1	Operational Health And Safety Risks	47
8.2.2	Project Scheduling Risks.....	48
8.2.3	Risk Matrix	49
8.3	Project Deliverables Outcomes.....	50
8.4	Opportunities	50
9	Appendix B: FIP Messages.....	51
10	Appendix C: TCS Message Protocols	52
10.1	TCS – ATOC Communication Protocol.....	52
10.2	TCS – RES Communication Protocol.....	52
10.3	TCS – VSS Communication Protocol.....	53
11	Appendix D: Test Bench Set-Up	54
12	Appendix E: FIP Tester Workflow Diagrams.....	55

13	Appendix F: FIP Tester Tests	57
14	Appendix G: TCS Testing Suite Architecture	58

LIST OF FIGURES

Figure 1	AutoHaul® system breakdown.....	1
Figure 2	High level system architecture diagram of relevant sub-systems and interfaces	3
Figure 3	FIP to IXL connections in the field.....	5
Figure 4	FIP to IXL connections in a test environment	6
Figure 5	Example test from TCS integration test specification document [19]	8
Figure 6	TCS Testing Suite behaviour.....	9
Figure 7	Ranking of test case criteria as determined by Adlemo et al. [29]	13
Figure 8	High level overview of FIP Tester tool	19
Figure 9	Generic test case workflow diagram.....	20
Figure 10	Flowchart of process to test new configurations management – See Figure 32 for the start-up sequence testing.....	20
Figure 11	GUI design sketch.....	23
Figure 12	High level TCS Testing Suite test engine design.....	25
Figure 13	Code showing a train being created before the enhancements	26
Figure 14	High level log parser design	27
Figure 15	System architecture of FIP Tester	29
Figure 16	High level overview of designed FIP Tester	29
Figure 17	GUI for the FIP Tester.....	30
Figure 18	Confirmation pop-up for the FIP Tester.....	30
Figure 19	'tcpdump' output	31
Figure 20	Linux command 'netcat' output.....	31
Figure 21	Python terminal output of test case 1 (Start-up sequence) with correct replies as seen by the FIP Simulator	32
Figure 22	Python terminal output of test case 1 (Start-up sequence) with incorrect replies as seen by the FIP Simulator.....	32
Figure 23	FIP Tester log file when incorrect message is received	32
Figure 24	FIP Tester log showing a passing test.....	33
Figure 25	FIP Tester log showing a failing test.....	33
Figure 26	Screenshot from user guide	34
Figure 27	Code showing the helper function implemented to create multiple trains	35
Figure 28	Screenshot of the collated errors file	36

Figure 29 Example of a test case used to verify test behaviour with pass/fail comments.....	37
Figure 30 Project timeline.....	46
Figure 31 Test bench set-up in Brisbane.....	54
Figure 32 Flowchart of process to test initial testing sequences.....	55
Figure 33 Flowchart of process to test site-like transmission errors.....	56
Figure 34 Start-up test code snippet	57
Figure 35 TCS Testing Suite architecture diagram [32].....	58

LIST OF TABLES

Table 1 Table of abbreviations.....	IX
Table 2 Description of sub-systems	4
Table 3 Polling cycle for FIP and IXL communications [15].....	6
Table 4 TCS interface details.....	7
Table 5 Types of tests to implement autonomously	14
Table 6 Interfaces covered in the scope	15
Table 7 Key deliverables list.....	16
Table 8 Summary of project stages.....	16
Table 9 System requirements for the FIP Tester	18
Table 10 Test specifications for FIP test 1: Testing the initial start-up sequence	21
Table 11 Test specifications for FIP test 2: Testing with site-like conditions.....	21
Table 12 Test specifications for FIP test 3: Testing how new configurations are handled.....	23
Table 13 Tests to verify the FIP Tester's behaviour.....	24
Table 14 Results from manually checking the logs in Figure 24	33
Table 15 Results from manually checking the logs In Figure 25	34
Table 16 Results of asking a tester to debug using the Log Parser.....	38
Table 17 OH&S risks summary.....	47
Table 18 Scheduling risks summary	48
Table 19 Risk matrix	49
Table 20 Outcomes of key project deliverables.....	50
Table 21 Project opportunities	50
Table 22 FIP – IXL Message types and corresponding control characters [15]	51
Table 23 TCS – ATOC message protocol [4]	52
Table 24 TCS – VSS message protocol [12]	53

Table 1 Table of abbreviations

Acronym	Long Term
AMMI	Automation Man Machine Interface
AS	Automation Server
aTest	A proprietary tool used at Hitachi to simulate ATOC connections
ATO	Automatic Train Operation
ATOC	Automation Train Operation Controller
CTC	Central Train Control
FIP	Field Interface Processor
GUI	Graphical User Interface
HMI	Human Machine Interface
IXL	Interlocking
MISS	Microlok Interlocking Simulation System
RTIO	Rio Tinto Iron Ore
RES	RTIO External Servers
SIL	Safety Integrity Level
TCS	Train Control Sub-system
UDP	User Datagram Protocol
UI	User Interface
VM	Virtual Machine
VSS	Vital Safety Server

1.0 INTRODUCTION

1.1 PLACEMENT CONTEXT

Rio Tinto Iron Ore (RTIO) operates a heavy-haul freight railway network in Western Australia [1]. This network moves iron ore from mines in the Pilbara region to coastal ports for shipping overseas. The AutoHaul® project provides an Automated Train Operation (ATO) system to facilitate the driverless movement of the trains on the mainline in RTIO's network [2] [3]. As shown in Figure 1, AutoHaul® uses three main sub-systems to control and monitor locomotives and ensure their safe movement in the network. These sub-systems are the Trainborne System, the Control Centre and the Wayside Systems. More details about these sub-systems are given in Section 2.0. Hitachi Rail STS (Hitachi) was contracted to deliver and maintain this ATO system.

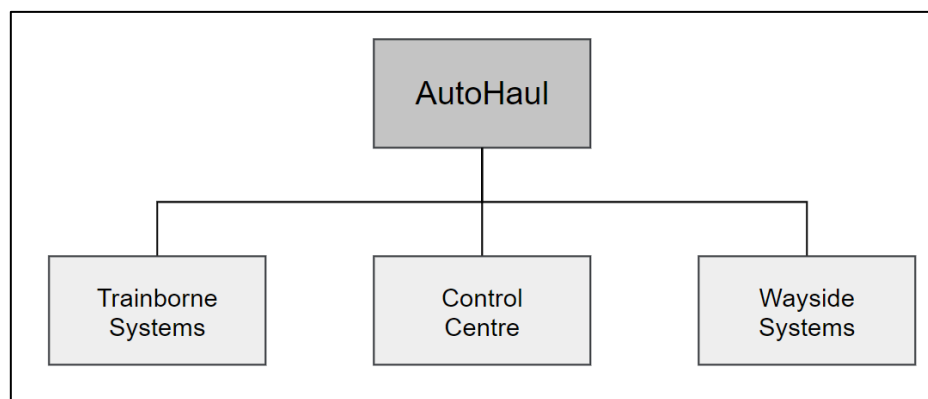


Figure 1 AutoHaul® system breakdown

Because the operations of trains in the AutoHaul® network are safety critical, any modifications made to the AutoHaul® system require extensive testing before they are rolled out. Hitachi's integration team is responsible for performing system-wide testing. This includes testing the system as a whole to ensure that all of the sub-systems are integrating properly with each other and all of the communication interfaces between different sub-systems work as required. Presently, the integration team uses manual testing for this purpose.

Due to the AutoHaul® project's complexity, manual testing is a time consuming and repetitive process that does not always identify the "edge case" issues in the system. As such, there is a potential to automate these tests.

1.2 PLACEMENT PURPOSE

Because Hitachi regularly roll out upgrades and modifications for the AutoHaul® project, they are investing in methods to reduce testing time by automating tests. As such, Hitachi wish to develop an automated testing tool that can be used to:

- Automate time-consuming and repetitive tests,
- Free up the tester's time and allow them to perform different tests and therefore improve the thoroughness of the testing and
- Reduce the products costs earlier in the testing stage of the product's lifecycle.

2.0 TECHNICAL BACKGROUND

2.1 THE AUTOHAUL® PROJECT

The AutoHaul® project introduced an Automated Train Operation (ATO) system so that trains are able to move autonomously on the mainline of RTIO's railway network in Western Australia [3].

As seen in Figure 1, the three sub-systems AutoHaul® uses to ensure the safe movement of trains are the Trainborne System, the Control Centre and the Wayside Systems [3].

The Trainborne System is installed in each AutoHaul® locomotive and uses the Automatic Train Operations Controller (ATOC) as its primary control system. The ATOC can be thought of as the train's driver. It interfaces to the locomotive's equipment to perform tasks such as braking, accelerating and collision detection. It also communicates with the Control Centre to receive instructions and transfer relevant data [4].

Located in Perth, the Control Centre uses the Train Control Sub-system (TCS) to control the majority of the AutoHaul® network [3]. It manages the train routing, mission planning and provides a user interface for operators to use.

The Wayside Systems contains a range of devices that are placed on the side of the tracks at various points in the network. Collectively, this system performs functions such as train tracking, interlocking and controlling intersections on the track [5].

A high level system architecture containing the sub-systems and communication interfaces relevant for this project is provided in Figure 2. A description of the sub-systems is provided in Table 2 and more details are given about these interfaces in Section 2.2.

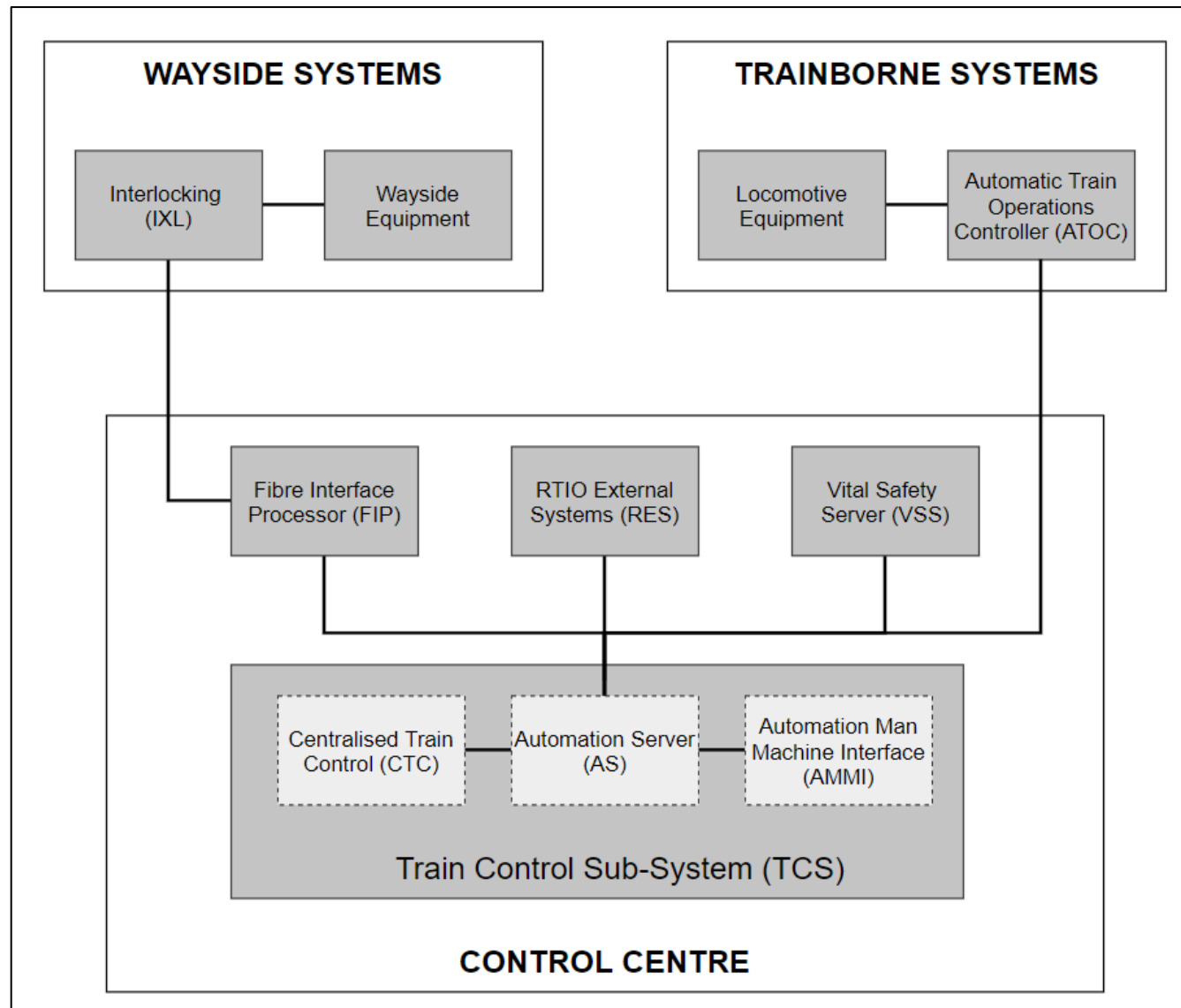


Figure 2 High level system architecture diagram of relevant sub-systems and interfaces

Table 2 Description of sub-systems

Sub-system	Functionality
WAYSIDE SYSTEMS	
Wayside Equipment	Equipment which is placed on the trackside and is used to monitor the health and status of track-based assets [6].
Interlocking (IXL)	<p>This component of signaling systems ensures that the railway behaves in a safe manner and is fail-safe [7] [8]. IXL devices do this by:</p> <ul style="list-style-type: none"> • Performing vital functions such as route setting, • Sending signaling information to the TCS and • Receiving commands, such as clearing signals, from the TCS. <p>Hitachi's Microlok II is used as the IXL devices.</p>
TRAINBORNE SYSTEM	
Locomotive Equipment	A collection of equipment and computer systems that are used to monitor and perform train operations such as interfacing with the ATOC and operating the throttle and brakes [3].
Automatic Train Operations Controller (ATOC)	The primary control system of the train. It communicates with other sub-systems and controls all of the train's operations [4].
CONTROL CENTRE	
Train Control Sub-system (TCS)	The system used to monitor and control the railway network. It is used to set routes, track train movement, manage alarms and perform monitoring actions that were previously undertaken by drivers [9].
Centralised Train Control (CTC)	A train control system that provides the network overview, shows indications and allows route setting and train sheet management [10].
Automation Man Machine Interface (AMMI)	A user interface to all the trains and locomotives in the AutoHaul® system. It allows users to access the CTC [11].
Automation Server (AS)	A messaging service that acts as a gateway for the TCS and the rest of the systems in the AutoHaul® system [5].
Vital Safety Server (VSS)	Provides movement authorities to the train based on data from the interlocking and level crossings [12]. The VSS also acts as a user interface and allows users to set commands which are relayed to the rest of the system.
Field Interface Processor (FIP)	A device which connects to all of the IXLs in the field and facilitates the exchange of information between the IXLs and the TCS [13].
RTIO External Systems	A TIBCO Enterprise Message Service which acts as the interface between the AutoHaul® and RTIO's other systems [14]. RTIO External Systems are used for functions such as producing an electronic train graph.

2.2 INTERFACE DETAILS AND CURRENT TESTING METHODS

2.2.1 Interlocking – FIP Interface

2.2.1.1 FIP Functionality

Each IXL device acts as a User Datagram Protocol (UDP) server which has an open socket on a specific IP address and port. The FIP runs multiple UDP clients which connect to IXLs on the given IP addresses and ports to enable bi-directional communication as shown in Figure 3.

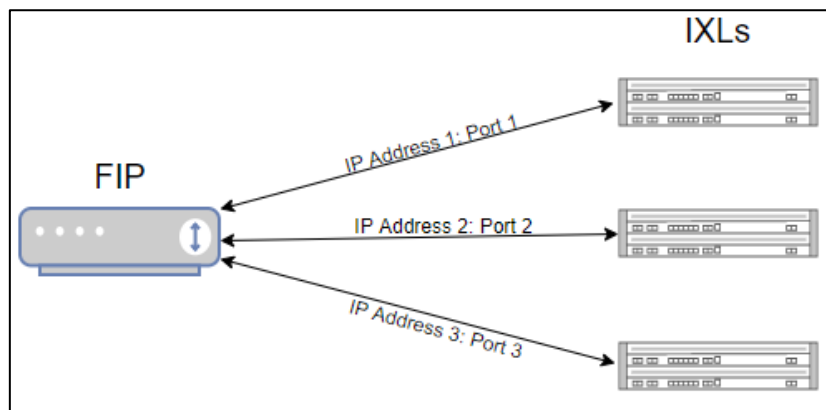


Figure 3 FIP to IXL connections in the field

Please note that only 3 connections are shown in Figure 3, the AutoHaul® project actually uses many more IXLs.

2.2.1.2 Data Contents

The data sent between the FIP and the IXLs uses the Genisys protocol [15]. Each message contains a control character, the address of the recipient/sender IXL, the interlocking data bytes, a security checksum and a termination character. More details about the packet structure are provided in Appendix B: FIP Messages.

The interlocking data bytes coming out of the Microlok IIs are at a Safety Integrity Level (SIL) 4 [16], and are therefore considered to be highly reliable. The FIP is only responsible for collating this data and passing it along to the TCS, where it is validated. As such, validating the interlocking data is outside the scope of this project. Instead, the main focus of the testing is validating the FIP's behaviour.

To request for data, the FIP uses a pre-defined polling cycle [15] as shown in Table 3. There are five types of messages that can be sent from the FIP to the IXL. These are poll messages, control messages, recall messages, execute messages and master acknowledge messages. Furthermore, there are three types of messages that can be sent back to the FIP. These are indication messages, control check back messages and slave acknowledgement messages. More details about the messages are provided in Appendix B: FIP Messages.

Table 3 Polling cycle for FIP and IXL communications [15]

FIP to IXL Message			Expected Reply
FIP to IXL1	FIP to IXL2	FIP to IXL3	IXL to FIP
Recall message	Recall message	Recall message	Indication message from each IXL.
Control message	Control message	Control message	Indication or slave acknowledge message from each IXL.
Poll message	Poll message	Poll message	Indication or slave acknowledge message from each IXL.
Recall message	Poll message	Poll message	Indication message from IXL1 and slave acknowledge messages from other IXLs.
Poll message	Recall message	Poll message	Indication message from IXL2 and slave acknowledge messages from other IXLs.
Poll message	Poll message	Recall message	Indication message from IXL3 and slave acknowledge messages from other IXLs.
Recall message	Poll message	Poll message	Indication message from IXL1 and slave acknowledge messages from other IXLs.

2.2.1.3 Current Testing Methodology

In a test environment, setting up the physical connections between the IXLs and the FIP is not feasible as there are too many hardware requirements. As such, a combination of a Virtual Serial Port Emulator (VSPE) and a Microlok Interlocking Simulation System (MISS) is used. The FIP requires the incoming data to come from UDP connections. However, due to being limited by its legacy software, MISS can only communicate using Transmission Control Protocol (TCP) communications. Therefore, a Socket Cat (SoCat) router is used to convert messages between TCP and UDP protocols. The testing set-up is illustrated in Figure 4.

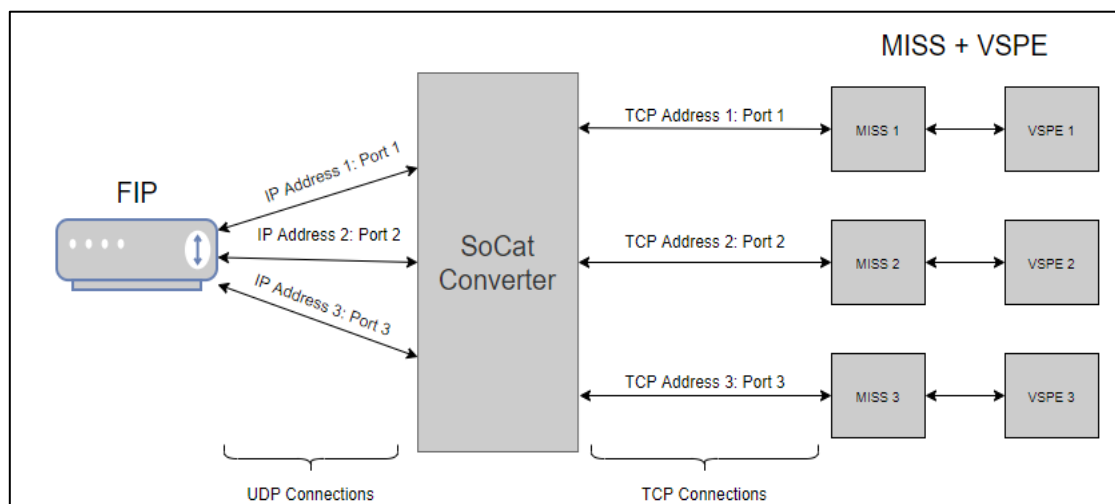


Figure 4 FIP to IXL connections in a test environment

Currently the FIP behaviour is tested using two main methods. These are:

- 1) Verifying that the IXL data received at the TCS is correct. It is assumed that if the data is correct when it arrives at the TCS then the FIP must be behaving correctly.
- 2) Manually checking the log files that are stored in the FIP to verify the messages. This testing is only conducted when an end-to-end change, such as adding a new IXL device, is made [17].

This project will focus on automating the manual checks that are conducted of the FIP's behaviour.

2.2.2 TCS Interfaces

As described in Section 2.1, the TCS is the main control system of the AutoHaul® network and is therefore tested frequently.

2.2.2.1 Relevant Interfaces

There are three communication interfaces within the scope of this project. These are the TCS – ATOC interface, the TCS – RES interface and TCS – VSS interface. The key elements of these interfaces are summarised in Table 4. Additional details about the specific packet structures and message types of each communication interface are provided in Appendix C: TCS Message Protocols.

Table 4 TCS interface details

Interface	Details
TCS – ATOC	<ul style="list-style-type: none">• The TCS – ATOC interface is a collection of communication channels between the TCS and each ATOC system that is active in the AutoHaul® network.• Locomotives use this interface to send locomotive status information to the TCS and receive instructions and software updates from the TCS.• The interface uses the TCS – ATOC Protocol [4] as the messaging protocol.• A propriety tool, aTest, is used to simulate messages that the ATOC would usually send autonomously. Users can send these messages via a User Interface (UI) or through commands set in a Python script.
TCS – RES	<ul style="list-style-type: none">• This interface allows the RES to send planned train sheets and timetables to the TCS. It also allows the TCS to send data regarding the actual train movement back to the RES.• Communication occurs via a TIBCO Message Service.• In the existing test environment, ActiveMQ is used as the messaging service and Python scripts can be used to simulate RTIO messages.• The interface uses the TCS – RTIO Protocol [14] as the messaging protocol.
TCS – VSS	<ul style="list-style-type: none">• This is a bi-directional communication channel that allows the TCS to send locomotive supervision data to the VSS and receive track information back from the VSS.• The interface uses the TCS – VSS Protocol [12] as the messaging protocol.• A VM running an instance of a VSS is running on the test bench.

2.2.2.2 Testing Methodology

Currently, two methods are used to test the TCS interfaces. These are manual testing and the TCS Testing Suite.

Manual Testing

This is the most commonly used form of testing at Hitachi. Testing the TCS involves manually performing sequences, such as creating a train sheet and verifying that all of the sub-systems behave as expected. This testing is done at a test bench which runs either a version, or a simulation of all sub-systems deployed in the AutoHaul® project. Appendix D: Test Bench Set-Up shows an image of this test bench.

The expected behaviour of the system is based on the TCS's functional and behavioural expectations [18] and is tested with tests laid out in the TCS Integration Test Specification document [19]. Figure 5 shows one test from this document. The test outlines activities that must be done before the test can begin (preconditions), which are the sequences to perform during the test and the expected results to each step.

3.33 TCS_EMS_FAUL_006: Display Open Faults			
Description: Display open faults to the Authorised User.			
Precondition: There exist several open faults belonging to different trains and being claimed by different users.			
Trace: TCS_683			
Step No	Description	Expected Result	Pass/Fail
1.	Log on to the CTC HMI with the role of an Asset Health Evaluator. Click on the Faults button on the Automation Toolbar.	'Fault Management' window opens.	
2.	Select 'Manage Faults' tab. Ensure Start Date and End Date in the Filter Options section covers sufficient time window to display existing faults. Enter the following search criteria: Fault Status = OPEN Click 'Request' button.	List of open faults pertaining to the search criteria is displayed, showing for each fault, as a minimum, train ID, summary of the fault, and the owner. Note: Empty owner field denotes that the fault is currently unclaimed.	

Figure 5 Example test from TCS integration test specification document [19]

All of the tests must be run independently to each other. For example, step 1 of Test 3.33 in Figure 5 requires the tester to log into the CTC's Human Machine Interface (HMI) as the Asset Health Evaluator (AHE). At the end of Test 3.33 the user must log out as the AHE even if the next test requires them to login as the AHE again. This adds a lot of time to the testing procedure as the testers must set-up and remove the test environment for each test.

Furthermore, the testing methodology is very structured and does not leave much time for "creative testing". Therefore, the testing does not cover all of the edge cases and unexpected situations that might arise in the field.

An experienced tester at Hitachi will require approximately an hour to complete the integration testing. This time estimate assumes that no bugs are found. In reality, testers find that they get half way through the set of tests before a test fails. After fixing the cause of the failure, they must restart the testing from the beginning.

TCS Testing Suite

A test environment has been created to run some of the more basic tests autonomously. The test environment is installed on a VM and uses Python scripts to perform the actions listed below.

- Initialise connections with local versions of AutoHaul® servers such as the RES and VSS,
- Connect to the ATOC emulator, aTest and emulate messages sending from ATOC,
- Connect to Squish, a tool created by Froglogic to automate UI testing [20],
- Use Squish to mimic interacting with the UIs and verify that the system is behaving as expected,
- Use a library of helper functions to facilitate communication between the TCS and all other AutoHaul® servers, ATOC and Squish and
- Run tests and log the activities in various log files depending on the action being performed.

This behaviour is summarised in Figure 6. Additionally, the file structure and architecture of the TCS Testing Suite is provided in Appendix G: TCS Testing Suite Architecture.

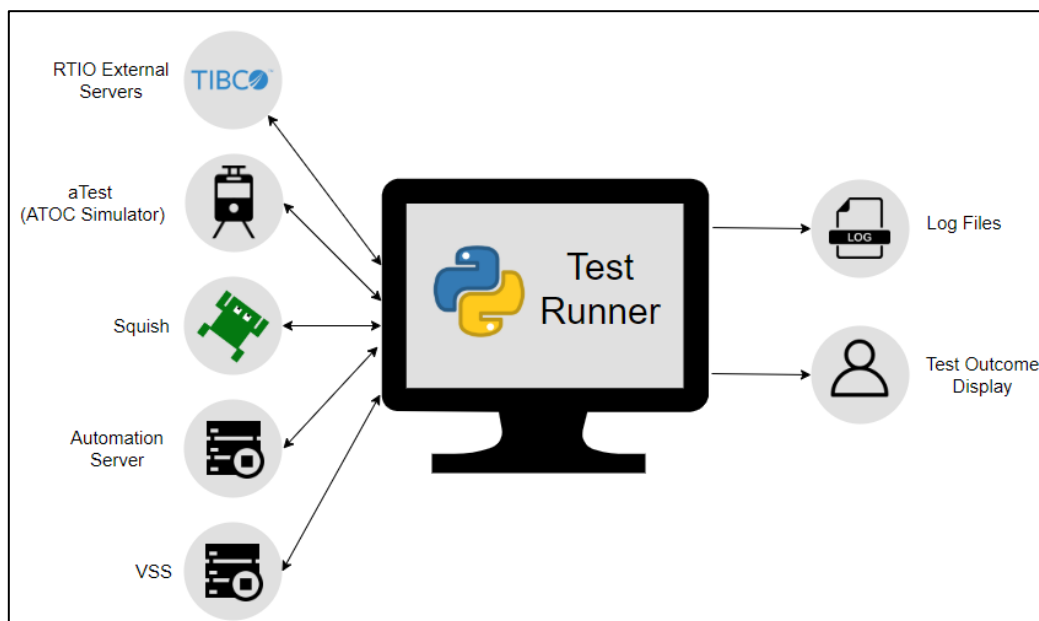


Figure 6 TCS Testing Suite behaviour

To use the test suite, a user must run a particular test script in a PyCharm environment. Once a test script is run it behaves as follows:

1. The AutoHaul® servers and simulators are started. This includes, beginning connections to the Automation Server, VSS, and RTIO External Servers.
2. A start-up function is implemented. The start-up function sets up all of the pre-conditions (such as creating and registering a train) that are required for the tests.
3. The tests are run. Generally, between 1 and 5 tests are contained within each script. The tests are typically implemented by:
 - a. Making an event occur in the system. For example, creating and registering a train.
 - b. Verifying that all of the messages related to that event have been received at their destination. For example, verify that the CTC has received the details for a new train.

- c. Using Squish to verify that the relevant user interface has been updated. For example, making sure that the new train ID is visible to the user.
 - d. A shut down function is implemented. This function resets the testing environment so that it is back to its original status. For example, if a train has been registered during the tests, then the train is deleted.
 - e. The servers and simulators are shut down. This includes closing all active connections.
4. A Python console displays the outcome of each test to the user.

Like the manual testing, the test suite also uses the TCS Integration Test Specifications documentation [19] for the test case design. However, not all of the required tests from the document have been implemented.

The current implementation only includes tests which check the functionality of a particular sub-system and that sub-system's User Interface (UI) currently exist. For example, a test can be run to check that a train sheet is created and registered in the correct user's view. But it cannot perform more complicated tests such as performing load testing to ensure multiple trains have been registered.

Furthermore, the test suite is a complex system that requires many active connections and uses many log files to keep track of the events in a test. Therefore debugging a test to determine the root cause of failure can be a time consuming task. For example, as a part of registering a new train, a TCS message is sent to the RES and the RES needs to send back a confirmation. If there is an error in connecting to the RES and therefore no confirmation is received by the TCS, then the following messages would be recorded in the most relevant log files.

- The TCS and CTC logs would record that the train could not be registered,
- The Automation Server logs would show that there was a timeout in waiting for the confirmation message and
- The Automation Server logs would record that a connection could not be established with the RES. However, this message would have been recorded at the start of the start-up activities and would have been hidden by the newer activities.

It would be up to the user to recognise that the error is due to connection issues, not problems with the AutoHaul's ® code or the test's logic.

This project focused on enhancing the capabilities of the TCS Testing Suite so that more complicated tests could run. It also provided a method for users to quickly determine the causes of failure in a system.

2.3 LITERATURE REVIEW

An increasing amount of businesses have begun to use automated testing tools to improve the efficiency of their testing and lower their product life cycle costs. The existing literature that is relevant to the project covers the design and performance testing of automated testing tools and frameworks.

2.3.1 Designing a Test Automation Framework

The framework required for automated testing involves defining support structure of the automation testing suite and the logical interactions between components within the testing suite. The project will require an automated testing framework to be developed from scratch for testing the FIP – IXL interface.

A paper by Bajaj [21] determines that a well-designed framework is essential for the automated testing suite to be reusable, maintainable and of high quality. The author categorises the types of frameworks for test automation under four main categories; modular frameworks, data-driven frameworks, hybrid frameworks and key-word driven frameworks. These categories are also supported by Umar and Zhanfang in their paper [22].

Méndez-Porras et al. [23] recommend using a top-down approach to designing an automated testing framework. The authors recommend that initially the testing framework's objectives and implementation requirements should be identified. Then a design should be created that uses these requirements. The design should initially be a high-levelled overview and then provide more details about each element in it. The authors found that creating detailed descriptions for the framework and its elements allowed them to minimise the number of test cases needed to identify all the errors in the system they were testing.

Additionally Wang et al. [24], hypothesises that because modern train control systems are becoming more complicated, a purely model based testing approach will not sufficiently test the system. Instead, they propose using a hybrid framework that is created in a virtual environment. The authors built a testing platform using an online Model-Based Testing (MBT) platform and a railway simulator. Their testing platform automatically generated and executed test cases.

To evaluate the testing platform, a case study was built using a real Communication Based Train Controller (CBTC) system and testing was conducted for 12 hours. It was found that in most cases, the hybrid MBT could detect errors more efficiently and in less time than traditional testing methods. However, because the hybrid MBT created and executed test cases simultaneously, it could not detect some errors such as minor delay errors.

User Interface Design

For this project a UI was developed to increase the useability of the testing tools. The UI was designed using the approach recommended by Borisov et al. [25]. The authors present an approach to designing a Test and Diagnosis User Interface (TDUI) which can be used to test ambient intelligent systems in production environments. Their UI design process focused on ensuring that the UI met the requirements needed for a safety-critical system, such as task conformance, providing feedback to users based on user actions and reporting on process status. The authors designed a TDUI using a two layer structure. The first layer was focused on the outward appearance of the UI and used strategies such as creating guideline resources, style guides and having a good page layout to reduce errors in interactions. The second layer was the main Interaction Logic Layer (ILL) used to connect the Graphical User Interface (GUI) to the rest of the test environment. The paper found that using the two layer approach, allowed for the designed TDUI to be adaptive to new changes and meet individual user requirements.

Relevant Standards

Because the railway industry is highly regulated the relevant standards must be followed to implement a testing framework. As the AutoHaul® project is based on the European Train Control System (ETCS), it uses standards set by the European Committee for Electrotechnical Standardization (CENELEC).

For this project, the CENELEC standard EN50128:2011 for “Railway applications – Communication, signalling and process systems – Software for railway control and protection systems” [18] is applicable. Based on the information in the standard, the project must include the development of test specifications, a test procedure, and the tests must produce a test report.

2.3.2 Evaluating Testing Tools

The literature surrounding evaluating the performance of testing tools is primarily focused on evaluating tools which are commercially available and may be used in various applications. Although this project requires the evaluation of a tool developed for a specific purpose, the general criteria that other papers have used to evaluate tools will still be relevant to the project.

As per Bajaj [21], the effectiveness of a testing tool or testing framework for a project is dependent on the technology stack which is being tested, the testing requirements, the skill sets of the users and the project’s testing budget.

Bajaj provides a summarised view of different testing tools and how they meet the above listed selection criteria. The paper recommends that the chosen testing tool should be tested with a proof of concept as soon as it is adopted. This allows the users to test the compatibility of the testing tools with the existing systems and ultimately allows the users to be more confident when choosing the final testing tool.

Similarly, in a paper by Polo et al. [26], various testing tools are compared based on their maturity levels and the expertise level required to use them. The authors provide a comparison of different testing tools and provide recommendations for tools based on the intended use. Overall, Polo et al. concluded that the best testing tool for a system is the one which has the highest maturity level and requires the lowest expertise level.

In his paper Jönsson [27], ranks the performance of automated GUI testing tools based on their defect detection, repeatability of tests and time requirements from users. Jönsson compares Squish, TestComplete and manual testing. The paper focused on comparing Squish, a GUI testing tool developed by Froglogic [20], TestComplete, a GUI testing tool by Smart Bear [28] and manual testing. Jönsson found that all three methods identified the same amount of defects, but Squish performed better in repeatability testing and required less time to set-up initially than TestComplete. This is useful to the project as the current TCS Testing Suite utilises Squish to automate GUI tasks.

2.3.3 Designing Test Cases

A large part of evaluating how effective a test tool is in uncovering a system’s flaws includes developing high quality test cases. The literature surround developing test cases is relevant to developing tests for the FIP – IXL interface.

In a preliminary investigation created for the Ontology-based Software Test cAse Generation (OSTAG), Adlema et al. consolidate and rank 15 criteria for “good” test case performance [29]. The authors use existing literature to identify important criteria for evaluating test case performance and then ask 13 software experts to rank the criteria on a scale of 0 (not relevant at all) to 10 (highly relevant). All of these software experts were from low to medium sized Swedish companies. The resulting rankings are shown in Figure 7. Based on the findings of Adlema et al. the designed test cases should be repeatable, accurate and correct.

Rank	Criteria	Mean
1	Repeatable	9.2
2	Accurate	8.4
3	Correct	8.1
4	Powerful	7.5
5	Maintainable	7.4
6	Complete	7.2
7	Traceable	7.2
8	Consistent	6.8
9	Reusable	6.5
10	Simple	6.3

Figure 7 Ranking of test case criteria as determined by Adlema et al. [29]

In their paper, Freudenstein et al. [30] develop a tool to partially automate the requirement based testing process [30]. Their tool, Specmate, captures testing requirements and uses them to generate test procedures or test-scripts for use in Allianz Deutschland. Their algorithm for creating test procedures is a three-step process. Firstly, Cause-Effect-Graphs (CEGs) model logical statements and their relationships. Secondly, test specifications are developed from the CEG outputs. Finally, the test procedure is implemented from the test specifications. The authors tested Specmate in Alliance Deutschland and found that it helped reduce the efforts in the creation of test-procedures but ultimately did not save time as the system took too long to setup. Because the testing requirements for the FIP – IXL interface will not change, an automated test case generator, as suggested by Freudenstein et al. [30] is not necessary. Instead, the three step approach recommended by the authors can be used manually to derive a single set of test cases in this project.

2.3.4 Summary of Literature Review

This review addressed the existing literature used to design a testing framework, evaluate testing tools and develop test cases. Based on the findings of the literature review, the following decisions were made for the development of the project:

- Developing an automated testing framework using the top-down method recommended by Méndez-Porras et al. [23],
- Using a two layer structure, as recommended by Borisov et al. [25], when developing a GUI,
- Ensuring that the project complies with the CENELEC standards EN50128:2011 [18],
- Continuing to use Squish for TCS testing due to the findings of by Jönsson [27],
- Utilising Freudenstein et al.’s three-step method to designing test cases [30] and
- Ensuring the test cases meet the criteria’s for good test cases as described by Adlema et al. [29].

3.0 PROJECT DESCRIPTION

3.1 AIMS AND OBJECTIVES

The overarching purpose of the project was to design and develop an integrated testing framework to support the automated testing of the AutoHaul® project. The testing concentrated on expanding the current testing framework to include the tests described in Table 5.

Table 5 Types of tests to implement autonomously

Testing Type	Description
Site-like behavior testing	Testing with inputs that the AutoHaul® system received from the site.
Load testing	Testing the behavior of the AutoHaul® system under normal and extreme loads.
Capacity testing	Testing to validate that the AutoHaul® system can handle the amount of traffic that it was designed to handle.
Automated testing of new configurations	Tests which can be used to see how the AutoHaul® system will behave when new configurations are programmed into it.

As such, the main goals were:

- Reviewing the existing testing methods at Hitachi,
- Successfully developing a testing tool that can perform the tests detailed in Table 5 and
- Creating documentation that allows the designed testing tool to be used easily.

3.2 SCOPE

The scope of the project focused on the capabilities of the testing tool. Items and tasks that were considered within the scope were:

- Investigating and choosing tools to automate the testing of systems via testing four specific communication interfaces. The interfaces, and a brief description of their purpose are given in Table 6.
- Developing a test engine with the chosen testing tools that can perform all required tests.
- Providing the documentation necessary for the tests to be used at Hitachi.

The tasks that were considered out of scope for this project were:

- Validating the payload of the interlocking data that is sent to the Fiber Interface Processor (FIP) from the Microlok Interlocking System (IXL).
- Testing the communication mediums (such as wireless or fibre optics) that are used in the communication interfaces.

The IXL – FIP interface was included in the scope as there was no testing tool used at Hitachi to test behaviour of this interface. All previous testing was focused on validating IXL data at the TCS and assuming that the FIP behaviour was correct. As such, testing the FIP's behaviour using the FIP – IXL interface was the project's main priority.

The three TCS interfaces were included in the scope as the TCS is the main system in the AutoHaul® project and any changes made to the system generally impact it. Therefore, this project tested the TCS's behaviour by using the ATOC, RES and VSS interfaces. Only these three interfaces have been chosen instead of other TCS interfaces, such as the TCS – FIP interface, because the other interfaces either have testing tools that have been specifically designed for them already or have no supporting functions in the TCS Testing Suite. As such, automating the testing for those interfaces would take more time than was available for this project.

More about these interfaces and how they fit into the AutoHaul® project are provided in Section 2.0.

Table 6 Interfaces covered in the scope

Interface	Description
Train Control Sub-system (TCS) – Automatic Train Operation Controller (ATOC)	<ul style="list-style-type: none"> • The TCS is a part of the Control Centre and is responsible for controlling most of the operations of the railway network. • ATOC is a part of the Trainborne System that acts as the main control system inside each train. • The TCS – ATOC interface is responsible for transmitting messages between these systems and is vital for the safe movements of trains in the network.
TCS – RTIO External Systems (RES)	<ul style="list-style-type: none"> • The RES is a server which acts as a gateway between the TCS and other RTIO's operations. • The TCS – RES interface is responsible for transmitting messages between these systems.
TCS – Vital Safety Servers (VSS)	<ul style="list-style-type: none"> • The VSS utilises data from the network to determine if and where it is safe for trains to move. • The TCS – VSS interface is responsible transmitting messages between these systems.
Microlok Interlocking System (IXL) – Fiber Interface Processor (FIP)	<ul style="list-style-type: none"> • Various IXLs are placed next to tracks and perform vital functions such as route settings. The FIP is a device which polls all of the IXLs, collates their responses and sends the data onto the TCS. • The FIP – IXL interface is responsible for transmitting messages between these systems.

3.3 PROJECT DELIVERABLES

The key deliverables for the project are outlined in Table 7.

Table 7 Key deliverables list

Deliverable	Description
1. Software Solution	A software package that meets the goals detailed in Section 3.1.
2. Architecture Design	The design and architecture of the software solution.
3. Test Procedure	The procedure used to validate the design of the software solution.
4. User Manual	A guide which steps the user through how to configure and run tests and interpret the output.
5. Reflective Journals	5 journals which reflect on key learnings relating to Engineers Australia stage 1 competencies during the placement.
6. Project Proposal	A document which outlines the context, project plan and risk assessment of the project.
7. Interim Report	A report which presents the project's progress and includes a discussion of the results obtained.
8. Oral Presentation	A presentation which summaries the findings and recommendations of the project.
9. Final Report	A report which details the findings and recommendations of the project.

3.4 PROJECT MANAGEMENT

The project was split into four stages. A description of each stage is provided in Table 8. More details about the stages and the project's timeline are provided in Appendix A: Project Management Summary.

Table 8 Summary of project stages

Stage	Description	Time spent on stage
Research	This stage was used to conduct research in order to understand the full context of the tests being performed and the interfaces being tested.	6 weeks
Design	In this stage, the testing tools that would be used for testing were chosen. A test engine was also designed.	4 weeks
Implementation	During this stage, functionality was programmed so that the test engine could perform the required testing.	11 weeks
Documentation	In this stage, user guides and test specifications were written to document the work done on the project and ensure that Hitachi could continue to access it after the completion of the project.	6 weeks

4 DESIGN

This section describes the technical approach used to develop testing tools for the FIP – IXL interface and the TCS interfaces. As the two testing tools were different, two separate design strategies were applied and are described below.

4.1 FIP TESTER

The aim of testing the FIP – IXL interface was to check the behaviour of the FIP by ensuring that its responses to IXL messages are correct. Hence, it was decided that a FIP testing tool (FIP Tester) would be developed. This tool would simulate IXL devices in a fashion similar to how aTest (Hitachi's proprietary ATOC simulation tool) operates in the TCS testing.

4.1.1 Test Framework Design

The design process used to design this testing tool was based on the top-down methodology recommended by Méndez-Porras et al. [31] in their paper. As recommended by Méndez-Porras, the system requirements were defined, then a high level system was designed.

System Requirements

The requirements for the FIP Tester are provided in Table 9.

Table 9 System requirements for the FIP Tester

Requirement	
Implementation Requirements	
1	All IXL simulations shall connect to the FIP in the same way that IXL devices do.
2	IXL packets shall use the Genisys protocol. Instead of actual interlocking data, the IXL simulator shall send "0101".
3	No changes shall be made to the FIP's software to use this testing tool.
4	The FIP Tester shall integrate with Hitachi's existing systems.
Test Case Requirements	
5	In order to run capacity and load testing, the FIP Tester shall simulate multiple IXL devices at the same time.
6	The FIP Tester shall test the behavior of the FIP in response to site-like transmission errors. These errors include: <ul style="list-style-type: none"> • Partially received messages, • Corrupted messages, • Duplicated messages, • Lost messages and • Messages time-out.
7	The FIP Tester shall test the behaviour of the FIP in response to a new IXL connecting to it after it has already been running.
Software Requirements	
8	The FIP Tester shall only use open source software libraries.
Documentation Requirements	
9	A document outlining the architecture of the software solution and the test procedure used to validate its behavior shall be provided.
10	A user guide which provides instructions shall be provided. The user guide shall provide instructions on: <ul style="list-style-type: none"> • Setting up the FIP Tester, • Running tests and • Making modifications to the system
Output Requirements	
11	The user shall be presented with a pass/fail report at the end of the tests. If a test fails, the report should include the cause of failure.
12	The FIP Tester shall log the testing activities to make it easier to debug issues.

High Level Design

As per Méndez-Porras et al. [31], the next step in designing the framework was to develop a high level system overview and then breakdown the functions of each element in the framework. As summarised in Figure 8, the designed behaviour of the FIP tester is as follows.

1. A GUI allows the user to enter test parameters. Example parameters include specifying the number of IXLs to create.
2. The test engine runs the tests to check the functionality of the FIP. This test engine should be able to perform the following tasks.
 - a. Log all testing activities.
 - b. Initialise IXL simulators and connect them to the FIP based on user specified IP addresses and port numbers.
 - c. Run tests to check that all the IXLs are connected and that communications are following the polling cycle described in Table 3.

- d. Test how the FIP behaves in each of the test cases.
- e. Shut down the test environment to prevent it from affecting future tests.
3. A pass/fail report is generated to inform the user about the results of the tests. In case a test fails, the report will include a reason for the failure.

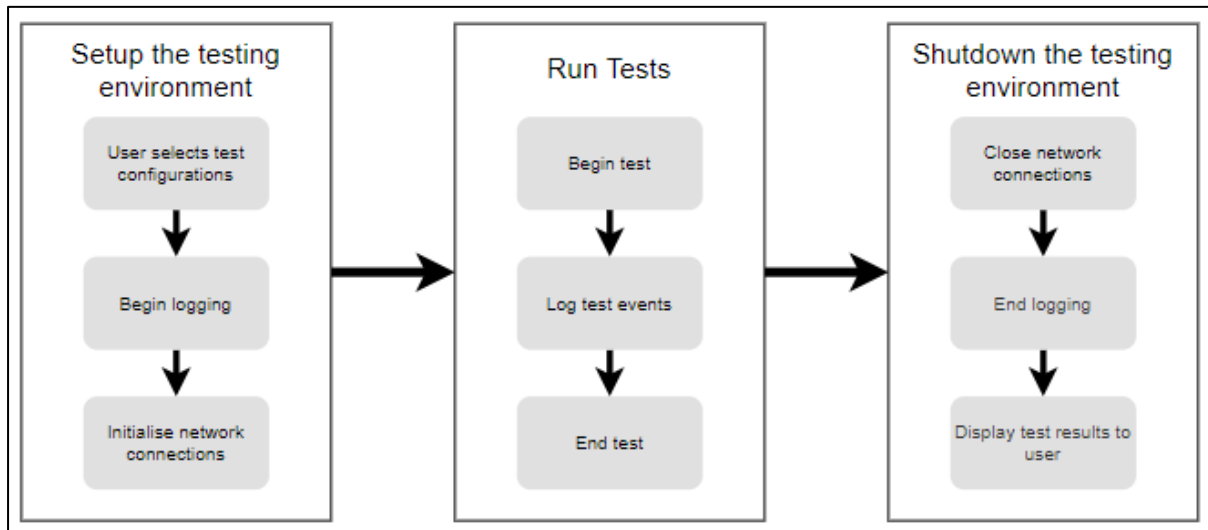


Figure 8 High level overview of FIP Tester tool

Key decisions made when designing this framework include:

- Using Python as the programming language because of the availability of multiple open-source libraries that will be useful to the project,
- Implementing the program in a Linux VM to increase adoptability in Hitachi.

4.1.2 Test Case Design

There were three test cases for which tests were designed. These were:

- 1) Testing the initial start-up sequence,
- 2) Testing with site-like conditions and
- 3) Testing how new configurations are handled.

The generic process used for tests (1) and (2) are shown in Figure 9. Detailed workflow diagrams for each test are provided in Appendix E: FIP Tester Workflow Diagrams. For testing site-like conditions, the 6 types of communication errors that were simulated and their expected response are:

- 25% packet loss,
- 50% packet loss,
- 75% packet loss,
- Duplicate packets being sent,
- Corrupted packets being sent and
- 100% packet loss.

In all of these cases, the FIP is expected to disregard the response that is received from the IXL and resend the request for the response.

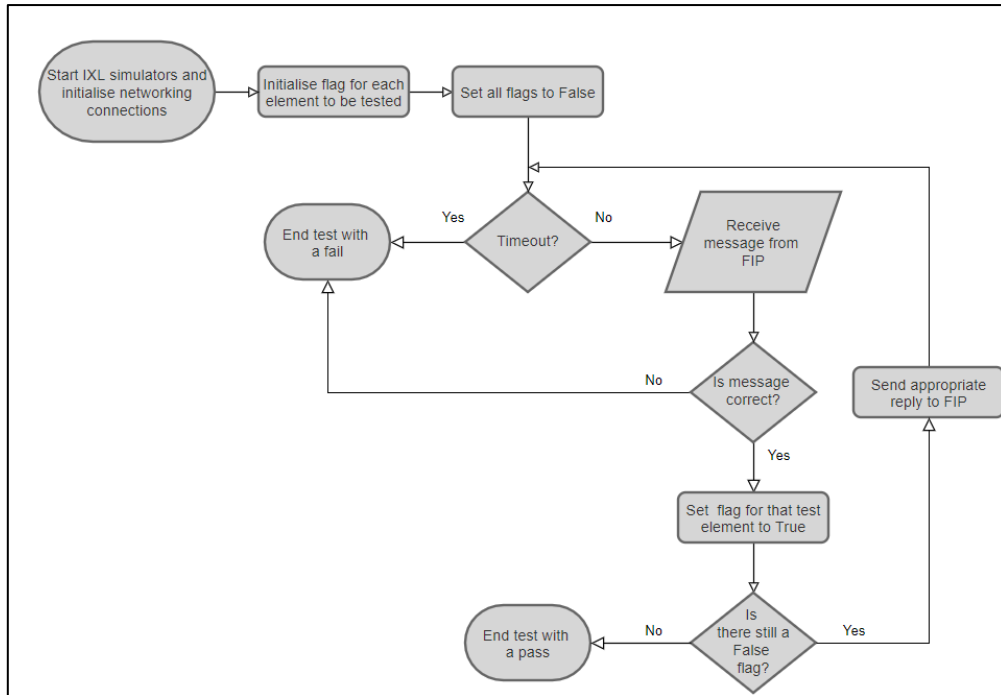


Figure 9 Generic test case workflow diagram

The generic process used for test (3) is provided in Figure 10. For each of the three tests, a test specification was created and is detailed in Table 10 to Table 12.

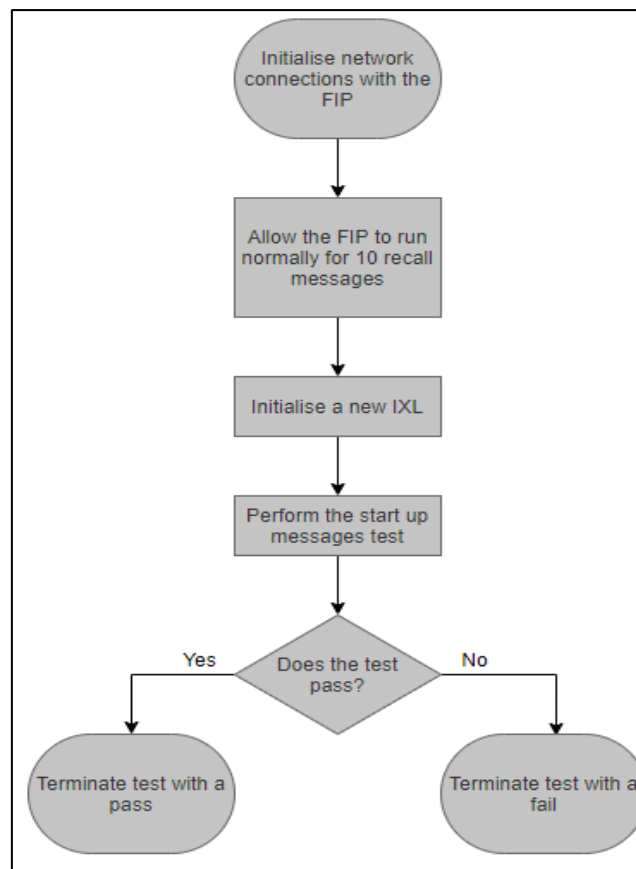


Figure 10 Flowchart of process to test new configurations management – See Figure 32 for the start-up sequence testing

FIP Test 1: Testing the initial start-up sequence

Description: Ensures that the FIP sends a recall, control, and poll message as per its polling cycle.

Pre-test actions:

- 1) Restart the FIP.
- 2) Ensure the VM hosting the FIP Tester has the correct networking settings.

Table 10 Test specifications for FIP test 1: Testing the initial start-up sequence

Step No	Description	Expected Result
1	FIP Tester begins running	A GUI opens that asks the user to enter test configurations.
2	A recall message is received at each valid IXL	The FIP Tester logs show the received message and display a message that shows that the recall test is passed.
3	An indication message is sent by each IXL. The FIP should receive this message and send a control message back to the IXL	The FIP Tester logs show the received message and display a message that shows that the control test is passed.
4	A slave acknowledge message is sent by each IXL. The FIP should receive this message and send a poll message back to the IXL	The FIP Tester logs show the received message and display a message that shows that the poll test is passed.
5	An indication message is sent by each IXL. The FIP should receive this message and send a recall message back to the IXL	The FIP Tester logs show the received message and display a message that shows that the test is complete. Test ends.

FIP Test 2: Testing with site-like conditions

Description: Ensures that the FIP can respond appropriately when there are communications errors.

Pre-test actions:

- 1) Restart the FIP.
- 2) Ensure the VM hosting the FIP Tester has the correct networking settings.
- 3)

Table 11 Test specifications for FIP test 2: Testing with site-like conditions

Step No	Description	Expected Result
1	FIP Tester begins running	A GUI opens that asks the user to enter test configurations.
2	A recall message is received at each valid IXL.	The FIP Tester logs show the received message and display a message saying the first recall message has been received.
3	An indication message is created and 25% of the message is sent by each IXL to the FIP. The FIP should receive this message, realise that it is invalid and resend the recall message.	The FIP Tester logs show the recall message arriving and displays a message that shows that the 25% packet loss recall test has passed.
4	A recall message is received at each valid IXL.	The FIP Tester logs show the received message.

5	An indication message is created and 50% of the message is sent by each IXL to the FIP. The FIP should receive this message, realise that it is invalid and resend the recall message.	The FIP Tester logs show the recall message arriving and displays a message that shows that the 50% packet loss recall test has passed.
6	A recall message is received at each valid IXL.	The FIP Tester logs show the received message.
7	An indication message is created and 75% of the message is sent by each IXL to the FIP. The FIP should receive this message, realise that it is invalid and resend the recall message.	The FIP Tester logs show the recall message arriving and displays a message that shows that the 75% packet loss recall test has passed.
8	A recall message is received at each valid IXL.	The FIP Tester logs show the received message.
9	An indication message is created. It is duplicated and sent by each IXL to the FIP. The FIP should receive this message, realise that it is invalid and resend the recall message.	The FIP Tester logs show the recall message arriving again received message and display a message that shows that the duplicate packet recall test has passed.
10	A recall message is received at each valid IXL.	The FIP Tester logs show the received message.
11	An indication message is created. Its CRC is changed to "0000" and it sent by each IXL to the FIP. The FIP should receive this message, realise that it is invalid and resend the recall message.	The FIP Tester logs show the recall message arriving again received message and display a message that shows that the corrupted packet recall test has passed.
12	A recall message is received at each valid IXL.	The FIP Tester logs show the received message.
13	No reply is sent	The FIP will timeout while waiting for a reply and resent a recall message.
14	A recall message is received at each valid IXL.	The FIP Tester logs show the received message. The display will show that all control tests have passed.
15	Steps 2 – 14 are repeated except a control message is received instead of a recall message.	The FIP Tester logs show the received message and displays a message saying that part of the testing is complete. The display will show that all control tests have passed.
16	Steps 2 – 14 are repeated except a poll message is received instead of a recall message.	The FIP Tester logs show the received message and display a message. The display will show that all poll tests have passed.
17	A recall message is received at each valid IXL.	The FIP Tester logs show the received message and displays a message that shows that the test is complete. Test ends.

FIP Test 3: Testing how new configurations are handled

Description: Ensures that the FIP behaves correctly when a new IXL is initialised after the FIP is already running.

Pre-test actions:

- 1) Restart the FIP.
- 2) Ensure the VM hosting the FIP Tester has the correct networking settings.
- 3) Run FIP Test 1 with 5 IXLs initialised.

Table 12 Test specifications for FIP test 3: Testing how new configurations are handled

Step No	Description	Expected Result
1	After FIP Test 1 has completed, allow time for 10 recall messages to be received at each IXL.	The logs show that FIP Test 1 has completed and 10 recall messages have been sent and replied to appropriately.
2	A new IXL is initialised.	The start-up testing is repeated for all of the connected IXLs.
3	A recall message is received at each valid IXL.	The logs show the received message and display a message that shows that the test is complete. Test ends.

4.1.3 GUI Design

In order to make the system easier to use, a GUI was designed using a two layer structure as recommended by Borisov et al. [25]. A sketch of the design is shown in Figure 11.

The GUI design sketch for the FIP Tester shows a window with a title bar. The main content area is titled "Section to Select Test Configurations". It contains a grid of input fields and buttons. The first row has "Number Of Connections To Initialise" and "User Types In Number". The second row has "Test Start-Up Sequence" and a "Drop Down Menu" button. The third row has "Test On-Site Conditions" and another "Drop Down Menu" button. The fourth row has "Test New Configurations Management" and a third "Drop Down Menu" button. The fifth row has "Configurations File" and a "File Selector" button with a file icon. The sixth row has "Hosts File" and a "File Selection" button with a file icon. At the bottom of the window, there are three buttons: "Confirm", "Cancel", and "Help".

Figure 11 GUI design sketch

Key design decisions made based on the two-layer GUI design strategy [25] were:

- Having a simple layout,
- Incorporating a “Help” button which opens the user guide,
- Including a pop-up dialogue box which asks the user to confirm all the settings,
- Allowing the user to control the complexity and type of testing that would run and
- Mimicking the configurations requirements (Hosts file and configurations file) to the FIP’s configuration requirements in order to minimise tester effort.

4.1.4 Performance Assessment Design

Table 13 describes the tests that were performed to ensure that the FIP Tester was working as expected.

Table 13 Tests to verify the FIP Tester's behaviour

Test Number	Test Method	Description	Purpose
1	'netcat' and 'tcpdump' at the FIP	<p>The Linux commands 'netcat' was used to view the incoming and outgoing traffic on a particular IP and port in the FIP and the FIP Tester.</p> <p>Additionally, the command 'tcpdump' was used to ensure that all incoming and outgoing packet contents were correct</p>	Ensured that the networking setup is accurate.
2	FIP Simulator	<p>Created a UDP client that connects to the FIP Tester on a particular IP address and port that would otherwise have been used to communicate with the FIP.</p> <p>A user then created messages in the Python terminal and sent them to the FIP Tester as if the FIP was sending them. For the test cases outlined in Table 10 to Table 12, the FIP's expected replies were sent via the FIP Simulator.</p> <p>In addition to those test cases, the user also introduced errors to the system and ensured that the FIP Tester responds correctly to those errors.</p>	Ensured that the FIP Tester's behavior is accurate.
3	Manually check FIP Tester Logs	<p>All of the FIP Tester's activities during tests were logged in the log files.</p> <p>A user manually checked the log files and ensure that the correct sequence of events, as defined by test cases shown in Table 10 to Table 12, are correct.</p>	<p>Ensured that the tests were repeatable and correct.</p> <p>It was also used to load test the FIP Tester.</p>

4.1.5 Documentation

The documentation provided for the FIP – IXL Interface Testing Tool was based on the documentation required by the CENELEC standard EN 50128:2011 [18]. It includes a user guide which provides details for initiating and running tests and a set of test specifications.

4.2 TCS TESTING SUITE

In order to achieve this project's goals, it was decided that the work done on the existing TCS Testing Suite would be built upon in the project. In terms of testing the TCS interfaces, the TCS Testing Suite had two main limitations.

These were:

- The existing tests and helper functions did not have the capabilities necessary to run complex tests.
- The entire Testing Suite was complex and not user friendly. Therefore, it was difficult for a user to traverse through log files and identify a cause of failure.

Therefore, the enhancements made to the TCS Testing Suite were focused on these two limitations.

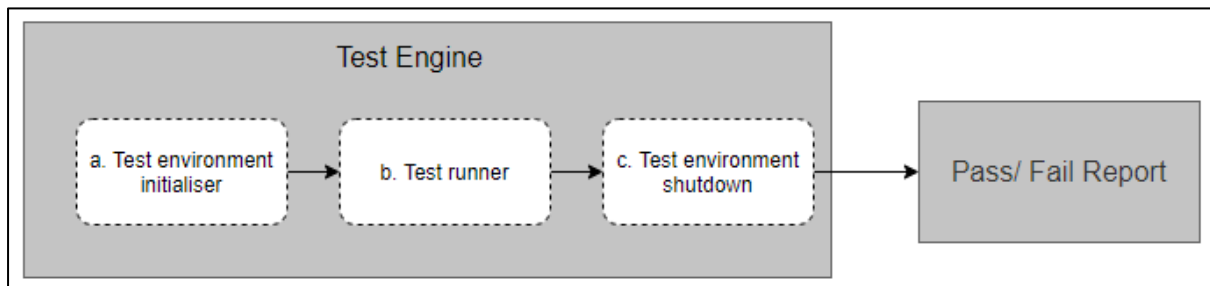


Figure 12 High level TCS Testing Suite test engine design

Figure 12 shows a high level design of the existing test engine used in the TCS Testing Suite.

Originally, the testing tool design involved modifying the test engine so that it uses a hybrid framework as recommended by Wang et al. [24]. This test tool would iterate through log files to automatically generate test cases and then run the tests using the Test Engine shown in Figure 12.

This test engine would have combined a Log Parser with the test engine, such that the overall testing tool could automatically identify the causes of error and then execute the relevant tests from the TCS Testing Suite. It would also have replaced Squish with an open-source GUI automation testing tool to save future licencing costs. However, this design decision was rejected for the following reasons.

- Developing the tool would require more time than was available in the project,
- Consultation with engineers in the testing team revealed that the TCS Testing Suite would be easier to maintain if the log parser and test engine were separate,
- The functionality required from the GUI automation testing tool was already implemented with Squish and
- Background research by Jönsson found that Squish is an effective tool for the required functionality [27].

Instead of modifying the test engine it was decided that all work done to enhance the TCS Testing Suite would remain compatible with the existing test engine. Additionally, a Log Parser was designed to reduce user effort when debugging tests. This section outlines the design process for both these activities.

4.2.1 Increasing Testing Capabilities

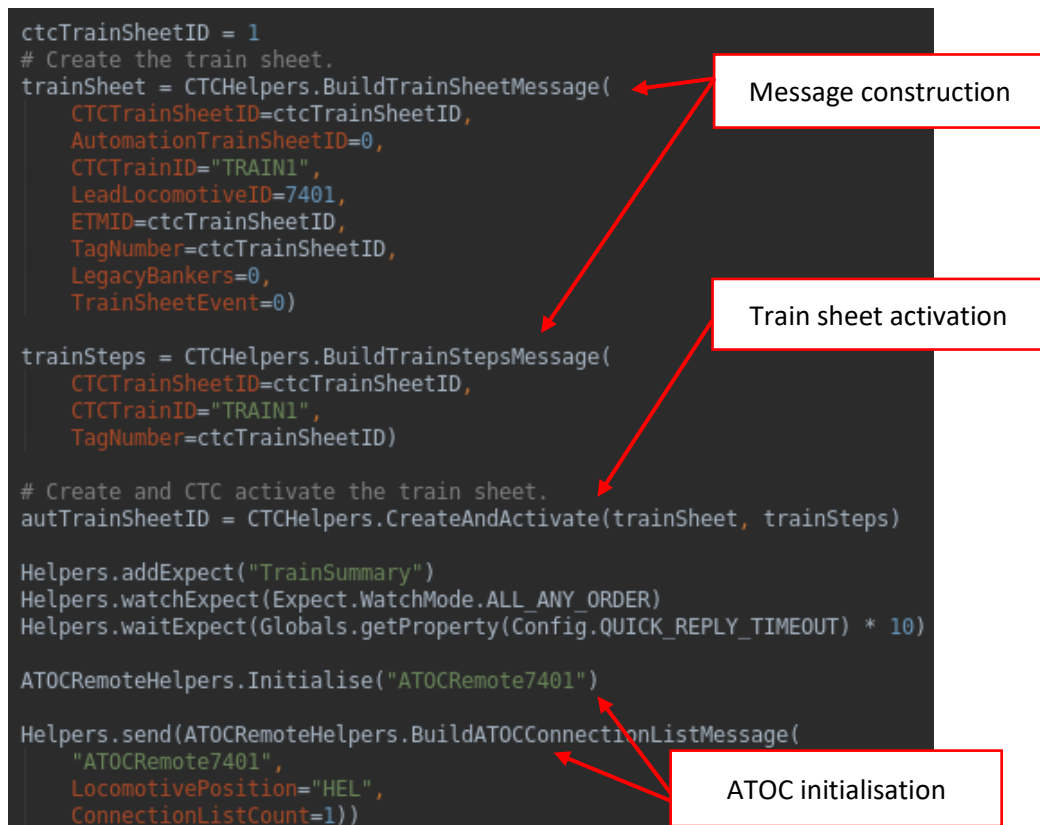
To increase the testing capabilities, two main activities were conducted. These were to increase the capabilities of the helper functions and existing tests and to add more tests.

Increasing Capabilities of Helper Function and Existing Tests

The TCS Testing Suite contains a library of helper functions and configurations that could be called during tests. But this library was only set-up to allow testing with certain configurations. For example, only a train with a specific Train ID could be created during tests because the configurations file and helper functions were hard coded to implement functionality for that Train ID. Therefore, the functions and tests were modified so that they could refer to a larger amount of trains.

As an example, Figure 14 shows a screenshot of the existing code used to register a train. This code would generally be called at the start of each test to create and register a train in the system. It works by:

1. Constructing the train sheet and train steps messages required to register a train. The helper functions used to construct this message contain a list of parameters and corresponding values. If a user provides a value for a parameter, then the default value is overwritten by the user generated value.
2. Creating and activating the train by using the "CreateAndActivate()" helper function. This function sends the train sheet and train steps message and then checks the Automation Server logs to ensure that the TCS has been provided with updated information. For example, it will ensure that a message has arrived that says that the train is active.
3. Initialising the ATOC for that train.



```
ctcTrainSheetID = 1
# Create the train sheet.
trainSheet = CTCHelpers.BuildTrainSheetMessage(
    CTCTrainSheetID=ctcTrainSheetID,
    AutomationTrainSheetID=0,
    CTCTrainID="TRAIN1",
    LeadLocomotiveID=7401,
    ETMID=ctcTrainSheetID,
    TagNumber=ctcTrainSheetID,
    LegacyBankers=0,
    TrainSheetEvent=0)

trainSteps = CTCHelpers.BuildTrainStepsMessage(
    CTCTrainSheetID=ctcTrainSheetID,
    CTCTrainID="TRAIN1",
    TagNumber=ctcTrainSheetID)

# Create and CTC activate the train sheet.
autTrainSheetID = CTCHelpers.CreateAndActivate(trainSheet, trainSteps)

Helpers.addExpect("TrainSummary")
Helpers.watchExpect(Expect.WatchMode.ALL_ANY_ORDER)
Helpers.waitExpect(Globals.getProperty(Config.QUICK_REPLY_TIMEOUT) * 10)

ATOCRemoteHelpers.Initialise("ATOCRemote7401")

Helpers.send(ATOCRemoteHelpers.BuildATOCConnectionListMessage(
    "ATOCRemote7401",
    LocomotivePosition="HEL",
    ConnectionListCount=1))
```

The image shows a screenshot of code with three red arrows pointing to specific parts of the code, each with a label in a red-bordered box:

- An arrow points to the `BuildTrainSheetMessage` function call, labeled "Message construction".
- An arrow points to the `CreateAndActivate` function call, labeled "Train sheet activation".
- An arrow points to the `BuildATOCConnectionListMessage` function call, labeled "ATOC initialisation".

Figure 13 Code showing a train being created before the enhancements

In order to enhance this code so that it can be used to create multiple locomotives, or different locomotives, the following actions were taken.

- Modifying the helper functions to remove references to specific configurations and
- Creating a new helper function which can be called during tests.

Adding More Tests

Several tests from the TCS Integration Test Specifications document [19] had not been included in the Testing Suite because more functionality needed to be added to the Testing Suite's helper functions before they could be run.

Therefore, as a part of increasing the testing capabilities, several tests relevant to the TCS – ATOC, TCS – VSS and TCS – RES interfaces were added to the system.

4.2.2 Log Parser

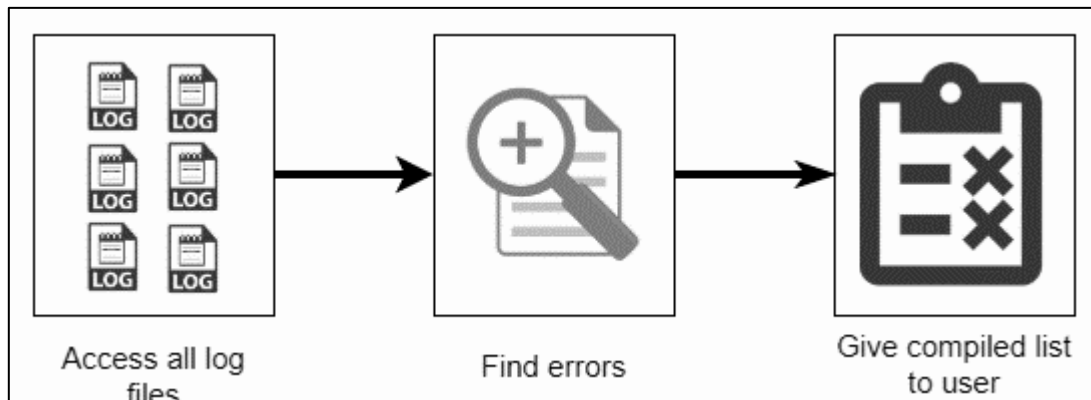


Figure 14 High level log parser design

The Log Parser was designed to help users identify errors in the system. It was developed in Python so that it can be included in the TCS Testing Suite's existing libraries package. As outlined in Figure 14, it does this by:

1. Taking in a file which contains the address of all relevant log files. The log files may be from the actual AutoHaul® system or the testing system.
2. Collating the critical, fatal and connection errors present in each of the log files.
3. Outputting this information to the user in a compiled list.

Users can then use this information to identify sections of the source code which must be changed to fix the errors.

4.2.3 Performance Assessment

Testing Enhancements

The testing conducted for TCS interfaces must follow the guidelines set in the TCS Test Specifications document [19]. Therefore, to verify that the enhancements made to the TCS system were working properly, the implemented tests were run. The user was able to visually verify that all of the steps and the expected outcomes of those steps were being performed.

Log Parser

Because the Log Parser only collates the errors already present in log files to save the user time while debugging, the aim of the performance assessment was to ensure that it did not miss any errors and that it was actually useful to the testers at Hitachi.

To perform this assessment, a simple test to check whether a train had been properly registered in the CTC was run multiple times. Each time the test was run, a different error was introduced to the system. A user then manually verified that all of the expected critical, fatal and connection errors produced by the system in response to these errors were present in the log files. The errors introduced to the system were:

- Disabling TCS communications by disconnecting the Automation Server,
- Attempting to create a train with invalid parameters and
- Commenting out configurations required to register a train.

These errors were chosen because they resulted in multiple error messages appearing in multiple log files as described in Section 0. They were also chosen because the changes made to the system could be rectified easily and would not break the functionality of the overall TCS Testing Suite.

Additionally, a user who regularly uses the TCS Testing Suite for testing was given the compiled error list from the three scenarios listed above. The user was then asked to identify the cause of failure based on the error messages.

4.2.4 Documentation

Hitachi engineers are already familiar with the TCS Testing Suite and they have already undergone a vigorous process to ensure that the tests in the TCS Integration Test Specifications document [19] comply with CENELEC standards [18]. Therefore, the documentation provided for the TCS Testing Suite was focused on ensuring that engineers know what functionality has been added to the TCS Testing Suite and how they can use the Log Analyser.

5 PROJECT OUTCOMES

5.1 FIP TESTER

5.1.1 The Testing Tool

The designed FIP Tester has been implemented with the architecture shown in Figure 15. The architecture was modelled after the TCS Testing Suite's architecture, which is provided in Appendix G: TCS Testing Suite Architecture. It was intentionally made to be simple as this allowed it to be easy to navigate and maintain for users who are not familiar with the Python programming.

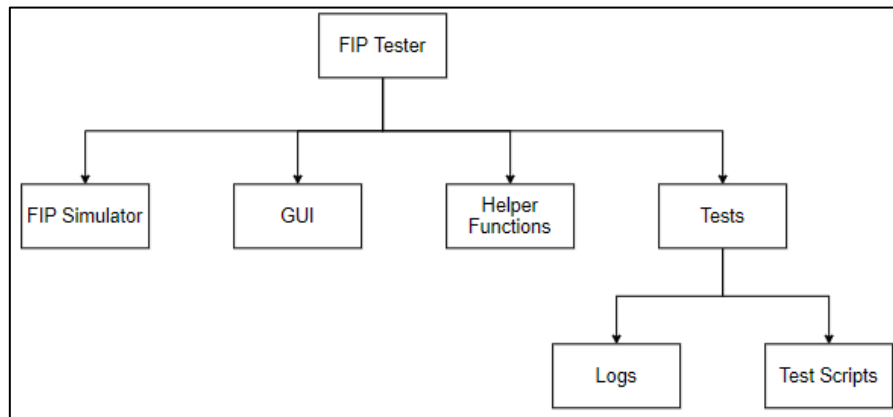


Figure 15 System architecture of FIP Tester

As illustrated in Figure 16 the FIP Tester is primarily controlled by a Test Controller. This controller is responsible for:

- Initialising the GUI to get test configurations from the user,
- Initialising the IXL simulators which connect to the FIP,
- Performing all logging functions,
- Running and ending tests and
- Displaying the test outcomes to the user.

In order to increase easy of maintenance and accessibility, the FIP Tester has been hosted on a VM and uses the same networking set-up that exists between the SoCat and the FIP that is shown in Figure 4. Therefore, the FIP's software does not need to be changed in order to use the FIP Tester.

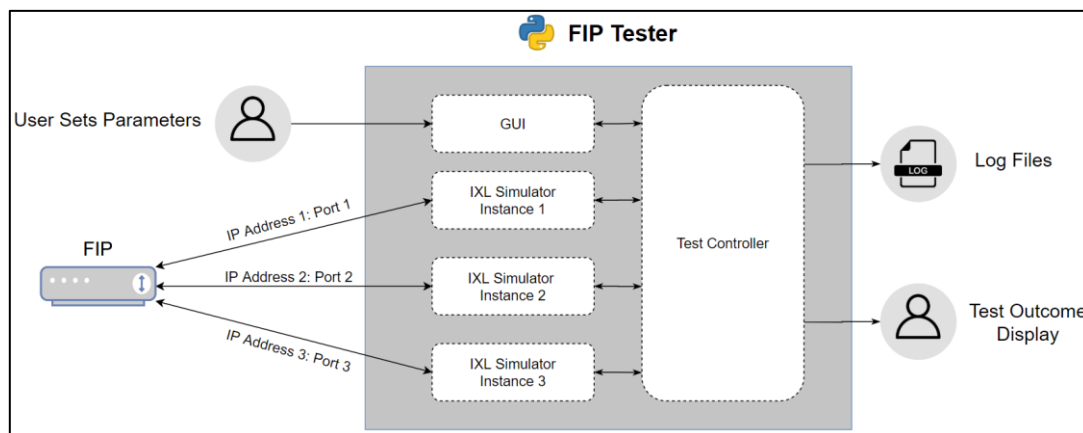


Figure 16 High level overview of designed FIP Tester

5.1.2 The GUI

The GUI implemented to take user inputs and set test configurations is shown in Figure 17. The GUI requires the user to enter the number of IXLs that the simulator will create, a text file containing the IP addresses, a text file containing the port numbers and the tests which they want to run.

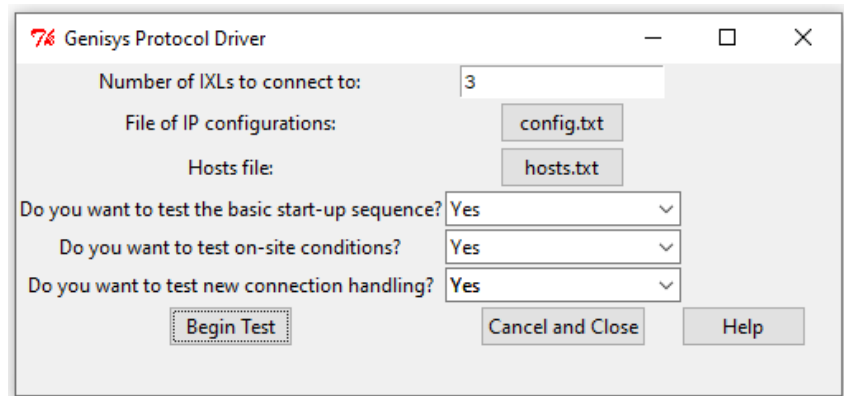


Figure 17 GUI for the FIP Tester

To reduce errors in interactions with users, a confirmation pop-up dialogue is created once the user selects “Begin Test” This pop-up is shown in Figure 18.

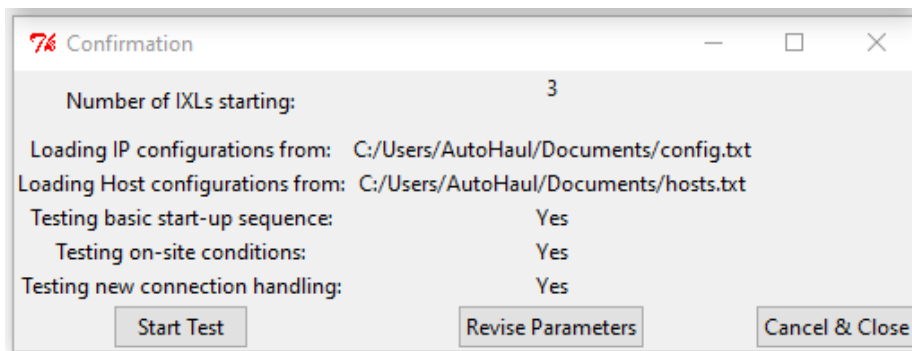


Figure 18 Confirmation pop-up for the FIP Tester

5.1.3 Test Cases

Code was written to run through the three tests outlined in in Table 10, Table 11 and Table 12. A code snippet from the start-up testing procedure is provided in Appendix F: FIP Tester Tests.

The main limitation of the implemented code is that because it was written to be easy to use and maintain for users not familiar with Python programming, it is not memory efficient.

5.1.4 Performance Assessment

Outcomes for Test 1 (Using the Linux commands ‘tcpdump’ and ‘netcat’ in the FIP)

The Linux command ‘tcpdump’ was used to verify that the packet contents were correct. The output from one of the connections is shown in Figure 19.

```

20:21:36.328240 IP fips.signet.hi.cra.com.au.hydap > CL-1.7.hydap: UDP, length 5
[root@fips fip]# /usr/sbin/tcpdump -x -t port 15000
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
IP fips.signet.hi.cra.com.au.hydap > CL-1.7.hydap: UDP, length 5
  0x0000: 4500 0021 0000 4000 4011 2b4d cb03 2b66
  0x0010: 0ad4 0e42 3a98 3a98 000d 0f9e fd2a c14f
  0x0020: f6
IP CL-1.7.hydap > fips.signet.hi.cra.com.au.hydap: UDP, length 7
  0x0000: 4500 0023 aeb6 4000 4011 7c94 0ad4 0e42
  0x0010: cb03 2b66 3a98 3a98 000f bed2 f22a 0101
  0x0020: d320 f600 0000 0000 0000 0000 0000

```

Recall message "fd 2a c1 4f f6" being sent to IXL

Indication message "f2 2a 01 01 d3 20 f6" being sent to FIP

Figure 19 'tcpdump' output

One of the key problems solved by using 'tcpdump' was that the FIP logs and documentation implied that individual elements of messages were surrounded with square brackets. For example, an acknowledge message would be "[f1][29][f6]". However, using 'tcpdump' made it clear that this was not the case and the message was actually "f129f6".

Furthermore, the Linux command 'netcat' was used to verify that the messages were arriving at and leaving from the correct IP address and port number. The output from one of these connections is shown in Figure 20. This figure shows the initial recall message ("fd 2a c1 4f f6") being sent from the FIP to the IXL. Because no message is being sent back to the FIP, the recall message is being sent repeatedly.

```

[autohaul@localhost ~]$ nc -luvn 15000 | hexdump -C
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Listening on :::15000
Ncat: Listening on 0.0.0.0:15000
Ncat: Connection from 203.3.43.102.
00000000 fd 2a c1 4f f6 fd 2a c1 4f f6 fd 2a c1 4f f6 fd
00000010 2a c1 4f f6 fd 2a c1 4f f6 fd 2a c1 4f f6 fd 2a

```

Figure 20 Linux command 'netcat' output

Using 'netcat' was highly effective in ensuring that the initial networking set-up was correct. A major problem encountered early in the project was that messages would be seen arriving correctly by 'tcpdump' but would not register as having arrived in the FIP software and therefore not show up in the FIP's logs. Using 'netcat' it was found that the firewalls were preventing the data from actually "reaching" its destination. The firewall rules were subsequently modified to allow the data to be transmitted correctly.

Outcomes for Test 2 (Using a FIP Simulator)

A FIP Simulator was used to manually send messages that met the expected behaviour of the test cases outlined in Table 10, Table 11 and Table 12. Figure 21 shows the replies sent by the FIP Tester when all of the correct messages are sent from the FIP Simulator in the start-up testing sequence. The logs in the FIP Tester look exactly like Figure 24.

```

FIPClient x TestController x
/home/autohaul/PycharmProjects/FIPTestSuite/venv/bin/python /home/autohaul/PycharmProjects/FIPTestSuite/scripts/FIPSimulator/FIPClient.py
> f22ac14ff6
Received data: f22a010d320f6
> f22ac14ff6
Received data: f12af6
> f22ac14ff6
Received data: f22a010d320f6
> |

```

Figure 21 Python terminal output of test case 1 (Start-up sequence) with correct replies as seen by the FIP Simulator

Figure 22 shows the replies from the FIP Tester when an incorrect message is sent from the FIP Simulator. As seen in the Python terminal output, the reply sent by the FIP Simulator after receiving a control message is incorrect. Therefore, the FIP Tester resends the control message. The corresponding log file in the FIP Tester is shown in Figure 23.

```

FIPClient x TestController x
/home/autohaul/PycharmProjects/FIPTestSuite/venv/bin/python /home/autohaul/PycharmProjects/FIPTestSuite/scripts/FIPSimulator/FIPClient.py
> f22ac14ff6
Received data: f22a010d320f6
> f22a010d320f6
Received data: f22a010d320f6
> |

```

Figure 22 Python terminal output of test case 1 (Start-up sequence) with incorrect replies as seen by the FIP Simulator

```

2020-06-15 22:27:19,755 INFO Logging started for CL-1.7
2020-06-15 22:27:19,755 INFO Attempting to connect to on IP:Port
2020-06-15 22:27:19,756 INFO Connection to FIP established
2020-06-15 22:27:29,486 INFO RECEIVED -> fd2ac14ff6
2020-06-15 22:27:29,487 INFO Recall Test Passed
2020-06-15 22:27:29,488 INFO Sent f22a010d320f6
2020-06-15 22:27:38,505 INFO RECEIVED -> fa000000f6
2020-06-15 22:27:38,506 INFO Invalid message received. Resending indication message
2020-06-15 22:27:38,506 INFO Sent f22a010d320f6
-----

```

Figure 23 FIP Tester log file when incorrect message is received

Please note that for confidentiality purposes, the IP address and port number that are present in the log file have been overwritten by the phrase “IP:Port” which is highlighted in blue. This also applies to other figures in this report.

Outcomes for Test 3 (Manually checking FIP Tester Logs)

The logs for the FIP Tester were checked manually to ensure that the system’s behaviour matched the expected behaviour that is outlined in Table 10, Table 11 and Table 12. An example of a passing test is shown in Figure 24. This is a test to verify the start-up message sequence. The corresponding results table is shown in Table 14.


```

2020-06-15 20:59:20,393 INFO Logging started for CL-1.7
2020-06-15 20:59:20,394 INFO Attempting to connect to on IP:Port
2020-06-15 20:59:20,394 INFO Connection to FIP established
2020-06-15 20:59:21,471 INFO RECEIVED -> fd2ac14ff6
2020-06-15 20:59:21,472 INFO Recall Test Passed
2020-06-15 20:59:21,473 INFO Sent f22a0101d320f6
2020-06-15 20:59:21,500 INFO RECEIVED -> fa2ac37ff6
2020-06-15 20:59:21,501 INFO Control Test Passed
2020-06-15 20:59:21,501 INFO Sent f12af6
2020-06-15 20:59:21,837 INFO RECEIVED -> fb2af6
2020-06-15 20:59:21,838 INFO Poll Test Passed
2020-06-15 20:59:21,838 INFO Sent f22a0101d320f6
2020-06-15 20:59:21,838 INFO Start up tests passed for CL-1.7.

```

Figure 24 FIP Tester log showing a passing test

Table 14 Results from manually checking the logs in Figure 24

Step No	Description	Expected Result	Pass/Fail
1	FIP Tester begins running.	A GUI opens that asks the user to enter test configurations.	NA
2	A recall message is received at each valid IXL.	The logs show the received message and display a message that shows that the recall test is passed.	Pass
3	An indication message is sent by each IXL. The FIP should receive this message and send a control message back to each IXL.	The logs show the received message and display a message that shows that the control test is passed.	Pass
4	A slave acknowledge message is sent by each IXL. The FIP should receive this message and send a poll message back to the IXL.	The logs show the received message and display a message that shows that the poll test is passed.	Pass
5	An indication message is sent by each IXL. The FIP should receive this message and send a recall message back to each IXL.	The logs show the received message and display a message that shows that the test is complete. Test ends.	Pass
Overall Outcome: Pass			

Similarly, Figure 25 and Table 15 show a failing test. In this test a connection to the IP address and port cannot be established and therefore no recall message is received by FIP Tester.

```

2020-06-15 21:47:06,331 INFO Logging started for CL-1.7
2020-06-15 21:47:06,331 INFO Attempting to connect to on IP:Port
2020-06-15 21:47:06,331 ERROR Connection to FIP could not be established.
2020-06-15 21:47:06,334 INFO Test environment shutdown.

```

Figure 25 FIP Tester log showing a failing test

Table 15 Results from manually checking the logs In Figure 25

Step No	Description	Expected Result	Pass/Fail
1	FIP Tester begins running	A GUI opens that asks the user to enter test configurations.	NA
2	A recall message is received at each valid IXL.	The logs show the received message and display a message that shows that the recall test is passed.	Fail
Overall Outcome: Fail because connection with FIP could not be established			

5.1.5 Documentation

```
#####
#
#                               USER GUIDE                               #
#
#####

SOFTWARE REQUIREMENTS

In the FIP Tester VM
  • Python 3.6
  • (Optional) PyCharm editor (Version 2020.1.1 Community Edition was used)

Non-Standard Python Packages
  • crccheck0.6 (https://pypi.org/project/crccheck/#description)
  • binascii
#####

NETWORKING SET-UP

Host Setup
  1) Create a host-only network (VMnet1)
     - Use the subnet IP <IP Address> and subnet mask:<mask details>
  2) Ensure both the FIP VM and the FIP Tester VM have a Network Adapter for
     this VM

In the FIP Tester VM
  1) Ensure that a network adapter for VMnet1 is setup
  2) Inside the FIP VM, the hosts file contains the IP addresses of all of
     the Interlocking devices that the FIP connects to. Assign all of the IP
     address the FIP Tester VM by going to the network settings and adding the IPs.
  3) Create a route for the FIP Tester to talk to the FIP by using the following
     command in the terminal.
     su
```

Figure 26 Screenshot from user guide

Figure 26 shows a screenshot from the user guide. The user guide provides instructions for:

- Installing the FIP Tester,
- Setting up the networking between the FIP Tester and the FIP and
- Making modifications to the tests.

5.2 TCS TESTING SUITE

5.2.1 The Enhanced TCS Testing Suite

As described in Section 4.2.1, two main activities were conducted to enhance the TCS Testing Suite. These were to increase the capabilities of the helper functions and existing tests and to implement more tests from the TCS Test Specifications document [19]. This section describes the product from performing these activities.

Increasing the Capabilities of Helper Functions and Tests

A number of changes were made to helper functions and tests in order to increase their capabilities and make them more useful for testing. These changes included:

- Modifying all helper functions which were hardcoded to be specific to one Train ID to be compatible with multiple Train IDs and
- Modifying tests to run with any train configurations.

As an example of these modifications, Figure 27 shows the enhanced version of code written during the project to create and register multiple trains when given an initial Train ID and the number of trains to create. It is based on the code shown in Figure 13. The changes made to the original code were:

- Moving the train creation functionality into a helper function. This allows users to modify parameters easily.
- Modifying the default parameters in the “BuildTrainSheetMessage()” and “BuildTrainStepsMessage()” so that factors which limit the train configurations to one train (such as train ID or train location) are no longer limiting.
- Creating ATOC helper functions, so that ATOC messages with non-default configurations can be sent.

Figure 27 has been annotated to show examples of these modifications.

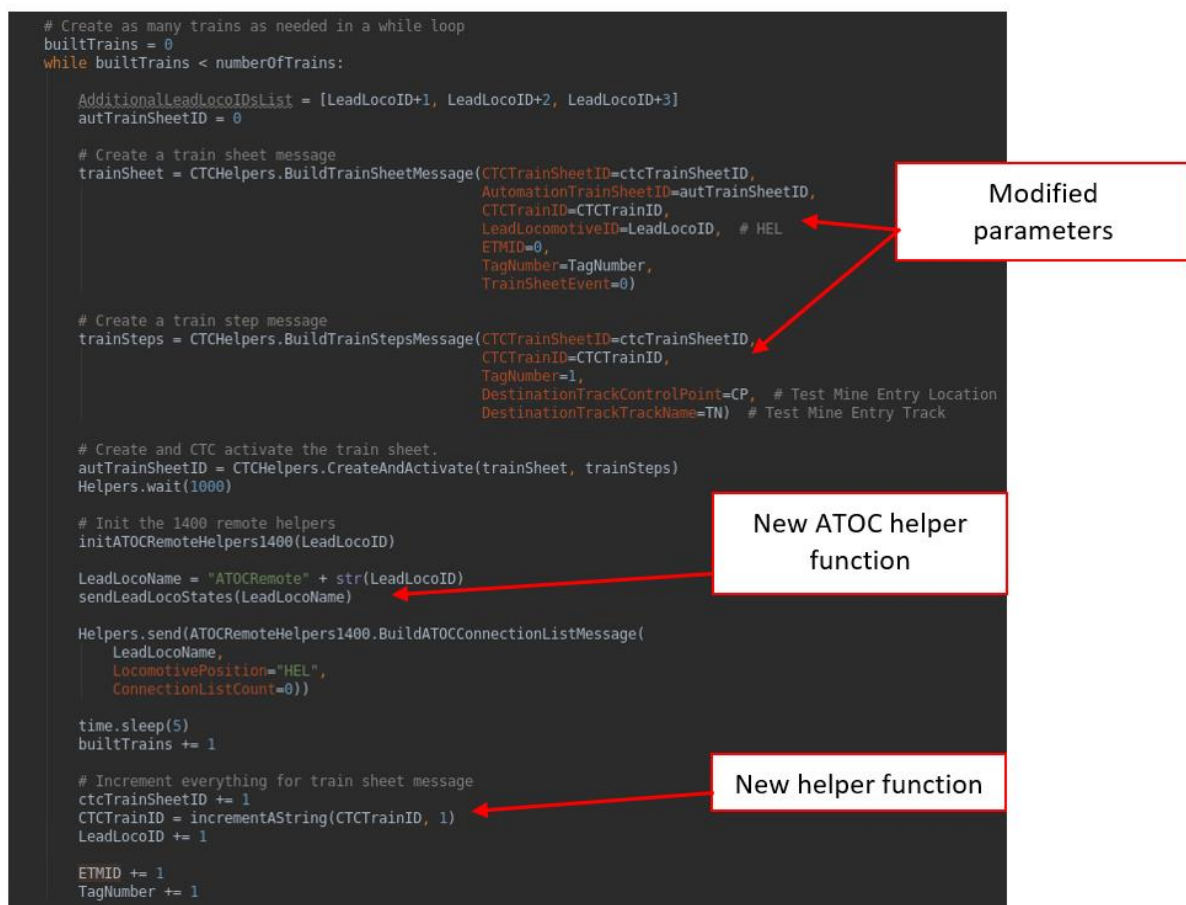


Figure 27 Code showing the helper function implemented to create multiple trains

5.2.2 The Log Parser

In the TCS Testing Suite, each AutoHaul® sub-system has a folder which contains its log files. Each time a test is run, the test runner migrates the most recent log to an archives folder and then creates a new log file for the sub-system.

As its input, the Log Parser takes a file containing the addresses of the folders which contain these logs. Each time the Log Parser is run, it iterates through the most recent log file for each sub-system and finds the critical, fatal and connections errors present in the file.

The Log Parser then collates the error messages into a single file. The errors are sorted by their file of origin and then by their time stamp. An example output is shown in Figure 28.

```
Errors in ATOCAdaptorLog:
2020-06-10 15:16:04,603 - ERROR - ThreadID: 140180158343360; Class: Routes; Method: AddRoute; Line: 72; Error: Routes already contains a route named GA_10_EM_16
2020-06-10 15:16:04,604 - ERROR - ThreadID: 140180158343360; Class: Routes; Method: AddRoute; Line: 72; Error: Routes already contains a route named GE_10_94_4E
2020-06-10 15:16:04,604 - ERROR - ThreadID: 140180158343360; Class: Routes; Method: AddRoute; Line: 72; Error: Routes already contains a route named GE_9_GU_5

Errors in RTIOAdaptorLog:
2020-06-10 15:16:07,617 - ERROR - ThreadID: 13970622721216; Class: AMQILayerAPI; Method: IL_Publish; Line: 363; Error: Cannot send message, no connection to broker
2020-06-10 15:16:08,632 - ERROR - ThreadID: 139705404340160; Class: RioExternalCommunications; Method: _logTibemsError; Line: 751; Error: Stack trace:
_tibemsClientLinkTcp_createSocket

Errors in AutomationServerLog:
2020-06-10 15:16:07,924 - ERROR - ThreadID: 140442927790016; Class: Routes; Method: AddRoute; Line: 74; Error: Routes already contains a route named GA_10_EM_16
2020-06-10 15:16:07,924 - ERROR - ThreadID: 140442927790016; Class: Routes; Method: AddRoute; Line: 74; Error: Routes already contains a route named GE_10_94_4E
2020-06-10 15:16:07,924 - ERROR - ThreadID: 140442927790016; Class: Routes; Method: AddRoute; Line: 74; Error: Routes already contains a route named GE_9_GU_5
2020-06-10 15:16:08,037 - ERROR - ThreadID: 140442927790016; Class: Routes; Method: AddRoute; Line: 74; Error: Routes already contains a route named JD_16_JD_6
```

Figure 28 Screenshot of the collated errors file

To save memory, the compiled errors file gets overwritten each time the Log Parser is run. The user has the option of modifying a parameter to save the previous version of the file if required.

A GUI was not designed for the Log Parser as the code is intended to be used by Hitachi testers who are already familiar with the TCS Testing Suite and are proficient with Python coding. Hence, adding a GUI would increase the time needed to use the Log Parser and therefore slow the testing efficiency.

A major limitation of the Log Parser is that it only searches for three types of errors. However, a lot of errors, particularly those seen during deployment at site, are not registered as errors by the AutoHaul® system and will not show up in the logs. Therefore, this tool is only useful as a quick debugging aid to narrow down the causes of error.

5.2.3 Performance Assessment

TCS Testing Enhancements

To verify that the tests written to enhance the testing capabilities of the TCS Testing Suite were working correctly, the user ran tests and visually observed the screen to ensure that the expected events were occurring as per the TCS Integration Test Specification document [19].

Where events occurred in the background and could not be visually verified, print statements were used to print the status of elements that needed to be checked. Figure 29 shows an example of a test that was verified using this method. Testing outcomes and notes have been written in the pass/fail column.

5.10 TCS_VSS_TCD_002: VSS Train Composition Data Request Scenario 2

Description: Request for Train composition data from VSS for a CTC activated train sheet.

Trace: TCS_457

Step No	Type	Description	Expected Result	Pass/Fail
1	Action	On CTC HMI, create a train sheet. CTC sends a CTC_TRAIN_STEPS message to Automation Server for this train sheet so that the train sheet gets activated.	There now exists an activated train sheet.	Pass
2	Action	From VSS send to Automation Server the message VSS_TRAIN_COMPOSITION_DATA_REQUEST for that HEL locomotive.	Automation Server receives the request for train composition from VSS.	Pass - Used print statement to see the message
3	Check	VSS receives the message TCS_TRAIN_COMPOSITION_DATA from Automation Server.	Automation Server sends the active train details to VSS.	Pass

Figure 29 Example of a test case used to verify test behaviour with pass/fail comments

TCS – RES Interface

Due to uncontrollable factors, the server hosting the RES was disabled in May 2020. Therefore, the TCS Testing Suite was not able to communicate with the RES and the code implemented was not able to be verified. Once RES functionalities resume, Hitachi engineers will be able to test the code written in this project.

Log Parser

To test the log parser, errors were intentionally introduced to the system and the combined log file was checked to ensure that all the errors appeared in it.

A tester at Hitachi, who had written several of the tests in the TCS Testing Suite, was asked to identify the cause of error based on the compiled error reports. Table 16 shows the results of this testing.

Table 16 Results of asking a tester to debug using the Log Parser

Test Type	Was The Error Identified?	Tester Comments
Disconnected the Automation Server	Yes	<p>Other connection errors, such as time synchronization issues between 2 sub-systems or incorrect port configurations also produce the same error message as a disconnected system in the logs.</p> <p>As such, the tester identified a disconnected Automation Server as the most likely cause of the error but indicated that they would have to do more testing to rule out other potential causes.</p>
Invalid train parameters	Yes	
Removing steps required to create a train	Yes	Identifying the error required the tester to go through the code.

6 CONCLUSION AND RECOMMENDATIONS

In conclusion, this project designed and developed an integrated testing framework for automating the testing of four communication interfaces used within the AutoHaul® project. The conclusions of the project are outlined below.

6.1 FIP TESTER

A testing tool called the FIP Tester was designed to simulate IXL and send IXL messages to the FIP in order to test that the FIP behaves correctly when responding to these IXL messages. This testing tool was designed to replace the manual testing done when an end-to-end change, such as adding or removing an IXL device, was made to the AutoHaul® system. The tool is effective in performing this task.

However, because the FIP Tester does not test the FIP's behaviour as a response to TCS requests, it has limited functionality at Hitachi. Therefore, future work could be done to create a TCS Simulator within the FIP Tester. This could be used to verify that the FIP responds to TCS messages correctly.

Furthermore, the scope of the project did not include sending correct interlocking data to the FIP. Therefore, the developed IXL Simulators inside the FIP Tester only send "dummy" interlocking data to the FIP. In order to make the testing more useful for Hitachi, the existing Microlok Interlocking Simulation System (MISS) testing tool could be integrated with the FIP Tester. Combined with a TCS Simulator and integration with MISS, the FIP Tester would allow Hitachi to test the entire journey of the data from the IXL devices to the FIP and finally to the TCS.

6.2 TCS TESTING SUITE

The project covered the testing for three TCS communication interfaces. These were the TCS – ATOC interface, the TCS – RES interface and the TCS – VSS interface.

Prior to the project, a TCS Testing Suite had been developed in Hitachi to automate basic tests. During this project, the TCS Testing Suite's functionality was enhanced so that it could autonomously perform tests for these three interfaces. This was done by:

- Modifying the helper functions to allow more complex test scenarios to be created,
- Modifying the existing tests to allow them to perform more complex tests and
- Writing more tests based on Hitachi's existing test specifications.

The TCS Testing Suite is a complex software tool that is difficult to navigate and debugging. To address this issue, a Log Parser was developed which scanned through various log files generated by a test and collated the errors into a single file. This made it easier for the user to debug the issue.

The main limitation of the TCS Testing Suite is that both the helper functions and the tests must be constantly updated as the AutoHaul® requirements change. Therefore, it requires a lot of maintenance in-order for it to remain useful at Hitachi. To address this issue, work can be done to make the tests and the helper functions more modular. This would significantly reduce the workload that arises when the system has to be updated as only the helper function would need to be updated.

Additionally, future work could also include further enhancing the TCS Testing Suite so that it can perform automated testing for all of the TCS interfaces that were excluded from the scope in this project.

6.3 PROJECT IMPROVEMENTS

This project could have been improved by better documentation and project management strategies.

In response to the COVID-19 pandemic, the order in which project activities were conducted was changed in March 2020. At that stage, work had primarily focused on enhancing the TCS Testing Suite and very little progress had been made on developing a FIP Tester. Due to the seemingly high likelihood of server access being lost, the project's focus was switched to the FIP Tester, as that was the project's priority. However, progress made on the TCS Testing Suite was not properly documented when the project's priority changed. Therefore, when the project switched back to enhancing the TCS Testing Suite weeks later, a lot of knowledge had to be re-learned.

Furthermore, Hitachi had implemented updates to the TCS Testing Suite and included new components. Therefore, a lot of the previously written tests were outdated and work had to be re-implemented.

A lot of time could have been saved if the progress made initially had been documented better.

7 REFERENCES

- [1] K. Smith, "Rise of the machines Rio Tinto breaks new ground with AutoHaul," *International Railway Journal*, vol. 59, no. 8, pp. 14-18, 2019.
- [2] Mining Media International, "Rio Tinto Achieves First Delivery of Iron Ore With World's Largest Robot," *Engineering and Mining Journal*, vol. 219, pp. 4-6, 2018.
- [3] Andrew Stewart, "AutoHaul System Architecture and Design Specification," Hitachi Rail STS, Brisbane, 2016.
- [4] Andrew Stewart, "TCS – ATOC Interface Control Document," Hitachi Rail STS, Perth, 2017.
- [5] Andrew Stewart, "Automation Server Subsystem Requirements Specification," Hitachi Rail STS, Brisbane, 2019.
- [6] Anthony MacDonald, "TCS-Wayside Interface Control Description," Hitachi Rail STS, Brisbane, 2013.
- [7] Graeme Reid, "TCS - PRCCI Interface Control," Hitachi Rail STS, Brisbane, 2020.
- [8] T. Rowbotham, "Interlocking," in *Introduction to Signalling*, London, Institution of Railway Signal Engineers, 2000.
- [9] Lionel Van Den Berg, "TCS Architecture And Design Specifications," Hitachi Rail STS, Brisbane, 2019.
- [10] Andrew Stewart, "CTC – Automation Interface Control Document," Hitachi Rail STS, Perth, 2018.
- [11] Samuel Dekker, "AUTOMATION MMI User Manual For Train Controllers," Hitachi Rail STS, Perth, 2020.
- [12] Andrew Stewart, "TCS - VSS Interface Control Document," Hitachi Rail STS, Perth, 2016.
- [13] Kent Yip, "Fiber Interface Processor," Hitachi Rail STS, Perth, 2008.
- [14] Andrew Stewart, "TCS – RTIO External Systems Interface Control Document," Hitachi Rail STS, Perth, 2017.
- [15] Stephane Joubert, "Genisys Protocol Description," Hitachi Rail STS, Perth, 2013.
- [16] P. Wigger, "Experience with Safety Integrity Level (SIL) Allocation in Railway Applications," Institute for Software, Electronics, Railroad Technology (ISEB), Cologne, 2001.
- [17] U. Silchanka, "FIP Test Manager," Hitachi Rail STS, Perth, 2008.

- [18] Cenelec, "Railway Applications - Communication Signalling and Processing Systems - Software for Railway Control and Protection Systems," Cenelec, Brussels, 2011.
- [19] Michal Cedrych, "TCS Integration Test Specification," Hitachi Rail STS, Brisbane, 2019.
- [20] Froglogic, "Squish," Froglogic, 2020. [Online]. Available: <https://www.froglogic.com/squish/>. [Accessed 28 February 2020].
- [21] H. Bajaj, "Choosing The Right Automation Tool and Framework Is Critical To Project Success," Infosys, Bengaluru, 2018.
- [22] M. A. Umar and C. Zhanfang, "A Study of Automated Software Testing: Automation Tools and Frameworks," *International Journal of Computer Science Engineering (IJCSE)*, vol. 8, pp. 217-225, December 2019.
- [23] A. Méndez-Porras, M. N. Hidalgo, J. M. García-Chamizo, M. Jenkins and A. M. Porras, "A Top-Down Design Approach for an Automated Testing Framework," in *International Conference on Ubiquitous Computing and Ambient Intelligence*, Springer, 2015.
- [24] Y. Wang, L. Chen, D. Kirkwood, P. Fu, J. Lv and C. Roberts, "Hybrid Online Model-Based Testing for Communication-Based Train Control Systems," *IEEE Intelligent Transportation Systems Magazine*, vol. 10, no. 3, pp. 35-47, 2018.
- [25] N. Borisov, A. Kluge, W. Luther and B. Weyers, "User Interface Design for Test and Diagnosis Software in Automotive Production Environments," in *International Conference on Ubiquitous Computing and Ambient Intelligence*, Springer, 2014.
- [26] M. Polo, P. Reales, M. Piattini and C. Ebert, "Test Automation," *IEEE Software*, vol. 30, no. 1, pp. 84-89, Jan 2013.
- [27] T. Jönsson, "Efficiency determination of automated techniques for GUI testing," School of Engineering in Jönköping, Jönköping, 2014.
- [28] Smart Bear, "Test Complete," Smart Bear, 2020. [Online]. Available: <https://smartbear.com/product/testcomplete/overview/>. [Accessed 3 June 2020].
- [29] A. Adlemos, H. Tan and V. Tarasov, "Test case quality as perceived in Sweden," in *2018 ACM/IEEE 5th International Workshop on Requirements Engineering and Testing*, ACM, 2018.
- [30] D. Freudenstein, J. Radduenz, M. Junker and S. Eder, "Automated test-design from requirements: the Specmate tool," in *5th International Workshop on Requirements Engineering and Testin*, ACM, 2018.
- [31] A. Méndez-Porras, M. N. Hidalgo, J. M. García-Chamizo, M. Jenkins and A. M. Porras, "A Top-Down Design Approach for an Automated Testing Framework," in *International Conference on Ubiquitous Computing and Ambient Intelligence*, Cham, 2015.
- [32] B. Mountford, "Testing Procedure For TCS Test Suite," Hitachi Rail STS, Perth, 2020.

- [33] Andrew Stewart, "VICS Interface Control Document," Hitachi Rail STS, Perth, 2016.
- [34] European Committee for Electrotechnical Standardization, "Railway applications – Communication, signalling and process systems – Software for railway control and protection systems," CENELEC, Brussels, 2011.

8 APPENDIX A: PROJECT MANAGEMENT SUMMARY

8.1 PROJECT TIMELINE AND RESOURCES

The overall project has been broken down into four stages – background research, design, implementation and documentation. More details about each stage and the resources required for that stage are provided in Sections 8.1.1 to 8.1.4. The project timeline is provided in Section 8.1.5.

8.1.1 Background Research Stage

Time Period: Week 1 – Week 6 (6 weeks).

In order to understand the full context of the tests that need to be automated, background research was conducted. This research focused on:

- How railway systems in general operate.
- The architecture, design specifications and testing protocols of the AutoHaul® project.

8.1.2 Design Stage

Time Period: Week 5 – Week 8 (4 weeks).

The design stage of the project was completed by using the following steps.

1. Understanding the existing systems and their advantages and weaknesses.
2. Researching alternative technologies to replace or supplement the existing tools. These technologies were required to be open-source and capable of running on an air-gapped system.
3. Designing test engines for testing the TCS interfaces and FIP communications.
4. Reviewing the test engine with the project supervisor.

8.1.3 Implementation Stage

Time Period: Week 8 – Week 19 (11 weeks).

The implementation stage was the longest stage in the project. It focused on implementing a test engine and then verifying that the test engine was reliable. The implementation stage was completed by using the following steps.

1. Obtaining initial test data for the TCS and the FIP.
2. Testing basic scenarios for TCS and the FIP. These scenarios were simple and easy to implement, for example creating a train sheet in the system. The purpose of these tests was to validate the design of the test engine.
3. Revising the test engine as needed.
4. Create a testing protocol for the FIP testing.
5. Coding tests for the FIP. This included verifying that the tests yield correct results.
6. Coding the tests from the pre-existing TCS Integration testing protocol. This included verifying that the coded tests are performing correctly.

Initially, the TCS testing was to be carried out first as it required more input from other engineers to set-up. However, due to the COVID-19 pandemic, the focus shifted to completing the FIP testing as it was the project's priority.

8.1.4 Documentation Stage

Time Period: Intermittent during Week 4 – Week 23 (19 weeks).

The documentation stage focused on presenting the final solution and all of its relevant details. This included creating documentation at Hitachi, to be used by engineers working on the AutoHaul® project and documentation for university. The tasks and the time spent on each documentation activity are shown below.

For Hitachi (Week 19 – Week 20 (2 weeks)):

- The software solution's design specifications (Week 19),
- The software solution's test procedure (Week 19) and
- User guide (Week 20).

Furthermore, for university assessment, the following items are still required to be completed:

- Monthly reflection journals (0.5 days per journal),
- Project Proposal (Week 7 – Week 8),
- Interim report (Week 14 – Week 15),
- Oral presentation (Week 20 – Week 21) and
- Final report (Week 19 – Week 23).

The original timeline that was submitted in the Project Proposal was modified so that the final report was allocated an extra two weeks of time. This was done to ensure that there would be ample time for the document to undergo a review at Hitachi to ensure that no confidential material is being published.

8.1.5 Project Timeline

The expected project timeline is presented below as a Gantt chart in Figure 30. The tasks to be completed are sorted by the stages described in Sections 8.1.1 to 8.1.4. The completion dates for the tasks in the chart acted as milestones for the project.

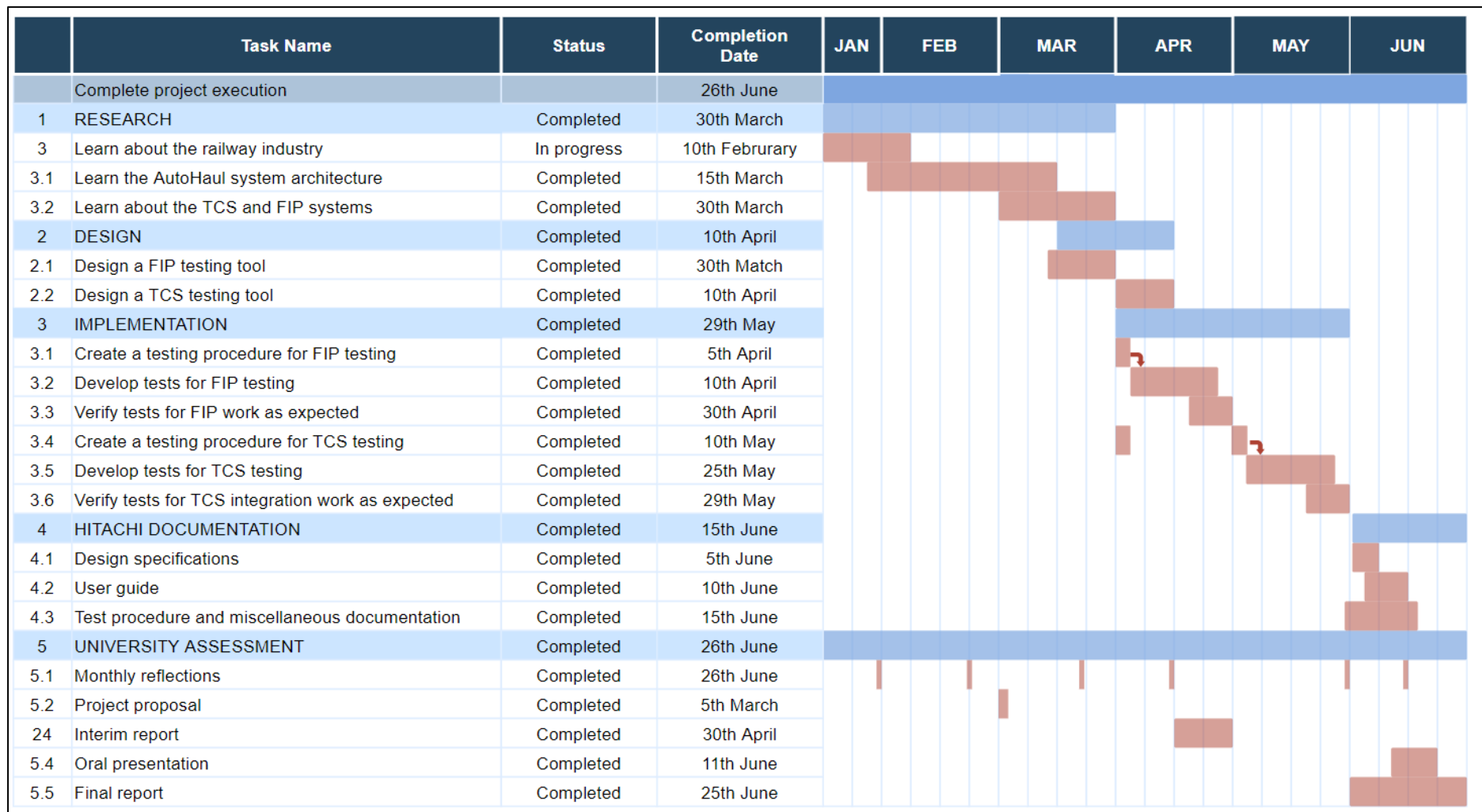


Figure 30 Project timeline

8.2 RISK ANALYSIS

The risks for this project were classified as, Operational Health & Safety (OH&S) risks and project scheduling risks. The risks and relevant mitigation actions are detailed below. The risks have been assigned a residual risk level based on the risk matrix in Section 8.2.3.

Since the Project Proposal, the only major change to the risk assessment was to accommodate for risks posed by the COVID-19 pandemic.

8.2.1 Operational Health And Safety Risks

The work being conducted in this project was done in the Hitachi office which was a low risk environment.

Table 17 OH&S risks summary

Hazardous Activity	Consequences	Residual Risk Level	Preventative Tasks	Mitigation Methods
Potential exposure to COVID-19 and other viruses	Getting sick	High	<ul style="list-style-type: none"> Practice the recommended preventative tasks such as maintaining social distancing and good hygiene practices. Use the same office desk and equipment. 	<ul style="list-style-type: none"> Alert office members immediately. If required, parts of testing the TCS interfaces will be removed from the scope of the project.
Using testing equipment, which has multiple large monitors, for extended periods	Aches and cramps on neck and back muscles	Medium	<ul style="list-style-type: none"> Take regular breaks and perform stretches. Stand where possible rather than tilting neck upwards. 	<ul style="list-style-type: none"> Rest well and perform non-test bed related tasks for a few days.
Computer usage for extended time periods	Eye strain	Low	<ul style="list-style-type: none"> Ensure lighting and screen positioning is optimal. Take regular breaks and perform eye exercises 	<ul style="list-style-type: none"> Use lubricating eye drops daily and take rest.
Typing for extended periods of time	Repetitive strain injuries to wrist	Low	<ul style="list-style-type: none"> Use an ergonomic keyboard and perform wrist exercises at regular intervals during the day. 	<ul style="list-style-type: none"> Rest well and use wrist supporting.
Slips, trips and falls	Skin injuries or broken bones	Low	<ul style="list-style-type: none"> Remain observant to surroundings when moving around the office. 	<ul style="list-style-type: none"> Apply first aid from the office and contact the HSE officer.

8.2.2 Project Scheduling Risks

These are risks which may cause the project to not be delivered in its ideal form.

Table 18 Scheduling risks summary

Hazardous Activity	Consequences	Residual Risk Level	Preventative Tasks	Mitigation Methods
The office is shut down due to COVID-19	Cannot access VMs on servers at Hitachi and therefore sections of work cannot be completed on time or to required standards.	High	<ul style="list-style-type: none"> Practice recommended prevention practices to reduce the spread of COVID-19. 	<ul style="list-style-type: none"> Install VMWare Workstation on a laptop and export VMs from the server onto it. If required, parts of testing the TCS interfaces will be removed from the scope of the project.
Sickness	Delays to the project.	High	<ul style="list-style-type: none"> Practice recommended prevention practices to avoid COVID-19 and other illnesses. 	<ul style="list-style-type: none"> Notify supervisor and work efficiently on return. Determine sections of the project to priorities completing if required.
Delayed access to required resources	Sections of work cannot be started on time.	Medium	<ul style="list-style-type: none"> Plan ahead and request access to resources in advance to requiring to use them. 	<ul style="list-style-type: none"> Focus on different sections of the project while waiting for particular resources.
Data losses	Work will need to be repeated.	Medium	<ul style="list-style-type: none"> Make weekly backup and document progress made. 	<ul style="list-style-type: none"> Revert to the last backup and continue work.
Procrastination or mental slumps	Delays to the project.	Low	<ul style="list-style-type: none"> Set weekly goals and use accountability tools to track progress. 	<ul style="list-style-type: none"> Work efficiently at other times.

8.2.3 Risk Matrix

Table 19 Risk matrix

			CONSEQUENCE				
			Negligible (1)	Minor (2)	Moderate (3)	Significant (4)	Catastrophic (5)
	OH&S		First aid only	Medical treatment required	Prolonged hospitalisation required	Permanent injury	Fatality
	Scheduling		No significant delays to the project	Minor delays with no lasting impact to the project	Delays requiring significant effort to recover	Sections of the project will not be completed	Project cannot be completed
LIKELIHOOD	Certain (5)	Frequent occurrence (once a week)	Medium (11)	High (16)	High (20)	Extreme (23)	Extreme (25)
	Likely (4)	Likely to occur (once a month)	Low (7)	Medium (12)	High (17)	Extreme (21)	Extreme (24)
	Possible (3)	Possible to occur (once every 6 months)	Low (4)	Medium (8)	Medium (13)	High (18)	Extreme (22)
	Unlikely (2)	Unlikely to occur (once a year)	Low (2)	Low (5)	Medium (9)	Medium (14)	High (19)
	Rare (1)	Practically impossible (once in 10 years)	Low (1)	Low (3)	Low (6)	Medium (10)	High (15)
RESIDUAL RISK LEVEL AND LEVEL OF AUTHORITY INVOLVEMENT							
Low Risk		Medium Risk		High Risk		Extreme Risk	
1 to 7		7 to 14		15 to 20		20 to 25	
No authority needed		Team leader		Academic/ Hitachi Supervisor		Course Coordinator	

8.3 PROJECT DELIVERABLES OUTCOMES

As outlined in Table 7, this project had 9 deliverables. The outcomes of these deliverables are outlined in Table 20.

Table 20 Outcomes of key project deliverables

Deliverable	Outcomes
1. Software Solution	Two separate software solutions were developed. The first built on the existing TCS Testing Suite and the second developed a Test Suite for testing the FIP's behavior via the FIP – IXL interface.
2. Architecture Design	The architecture design for both software solutions has been provided in this report.
3. Test Procedure	A document containing the test procedure has been submitted to Hitachi.
4. User Manual	A user manual has been submitted to Hitachi.
5. Reflective Journals 6. Project Proposal 7. Interim Report	These deliverables were submitted to UQ at various stages of the semester. Figure 30 contains the full details of the submission times.
8. Oral Presentation	An oral presentation which summarised the project was delivered on the 11 th of June 2020.
9. Final Report	This report is the final report.

8.4 OPPORTUNITIES

The major project opportunities of this project are provided in Table 21.

Table 21 Project opportunities

Opportunity	Description
Further expanding the TCS Testing Suite	As a part of this project, the existing TCS Testing Suite was modified to make it easier to perform more complicated operations, such as registering multiple locomotives. Therefore, it will be possible to add additional tests which can be used to test other TCS functionality and use other TCS interfaces.
Testing the FIP's behavior via a TCS – FIP interface.	This project focused on testing the FIP's behavior by focusing on the communications between IXL devices. Now that this test suite exists, it is relatively easy to enhance this test suite so that it includes a TCS simulator as well. This will make it easier to verify that the FIP is behaving correctly to TCS requests and allowing the FIP's behavior to be validated from the TCS side of operations.

9 APPENDIX B: FIP MESSAGES

Genisys Protocol Message Structure

Messages sent over the Genisys protocol have five components. All elements are sent as hexadecimal characters. The five components of messages are [15]:

1. Control character – a character which denotes the type of message being sent.
2. Station address – The address of the station where the IXL is located as a hexadecimal value. A message sent from an IXL would contain its own station number and a message sent from the FIP would contain the station number of the target IXL.
3. Data bytes – The interlocking data being sent for the TCS to use. Depending on the message type, data bytes may not be sent.
4. Security checksum – A two byte Cyclic Redundancy Check (CRC) of all message bytes, up to but not including, the security checksum. Depending on the message type, a security checksum may not be sent.
5. Termination character – The character “f6”.

Table 22 provides more details about each message type.

Table 22 FIP – IXL Message types and corresponding control characters [15]

Message Type	Control Character	Description
FIP to IXL Messages		
Master Acknowledge Message	fa	Used to acknowledge a message and to act as a poll message.
Poll Message	fb	Used to ask the IXL if it has any new indications it wishes to report. It receives an indication message if the IXL does wish to send updated data or it receives a slave acknowledgement.
Control Command	fc	Used to send controls.
Recall Indication Command	fd	Requests the IXL to send all its indications to the FIP.
Execute Controls Command	fe	Causes the IXL to write the controls to its database.
IXL to FIP Messages		
Slave Acknowledge Message	f1	Sent as a response when no other response is needed.
Indication Data Response Message	f2	Used to send data to the FIP.
Control Checkback Message	f3	Used to verify the controls from the FIP when in checkback control mode.

As an example, a slave acknowledge message from station 80 would be sent as “f150f6”.

10 APPENDIX C: TCS MESSAGE PROTOCOLS

10.1 TCS – ATOC COMMUNICATION PROTOCOL

The message structure for all TCS – ATOC messages is described in Table 23.

Table 23 TCS – ATOC message protocol [4]

Field	Description
Sequence number	A number used to order the messages
Time stamp	Message time stamp
Protocol version	An identifier for the protocol version
Source ID	An identifier for the sender of the message
Destination ID	An identifier for the receiver of the message
Length	The length of the data being sent. Varies according to the data.
Data	The data being communicated between the sub-systems. Is variable in length.
Cyclic redundancy checks (CRC)	Uses CRC-32

10.2 TCS – RES COMMUNICATION PROTOCOL

TCS – RTIO External Systems messages all comply with the XML schema. The element and attribute names depends on the type of message being sent. In general, the elements include:

- Message type,
- Time stamp and
- Message details.

There are 12 types of messages that can be sent from the TCS to RTIO External Systems. These are:

- Train Sheet,
- Train Step,
- Track Block,
- Fleeting,
- Track Notification,
- Train Notification,
- Train Position,
- HiRail Track Machine Position,
- Point Tag,
- Turnout Status,
- Temporary Speed Restrictions and
- Train Speed Restrictions.

Additionally, there are 3 messages that RTIO External Systems might send to the TCS. These are:

- Refresh,
- Execute Schedule and
- Notification.

10.3 TCS – VSS COMMUNICATION PROTOCOL

The message structure for all TCS – VSS messages is described in Table 24.

Table 24 TCS – VSS message protocol [12]

Field	Description
Source ID	An identifier for the sender of the message.
Destination ID	An identifier for the receiver of the message.
Protocol version	An identifier for the protocol version.
VSS Status	Can be Master or Standby.
Time stamp	Message time stamp.
Length	The length of the data being sent. Varies according to the data.
Data	The data being communicated between the sub-systems. Is variable in length.
Cyclic redundancy checks (CRC)	Uses CRC-32.

11 APPENDIX D: TEST BENCH SET-UP

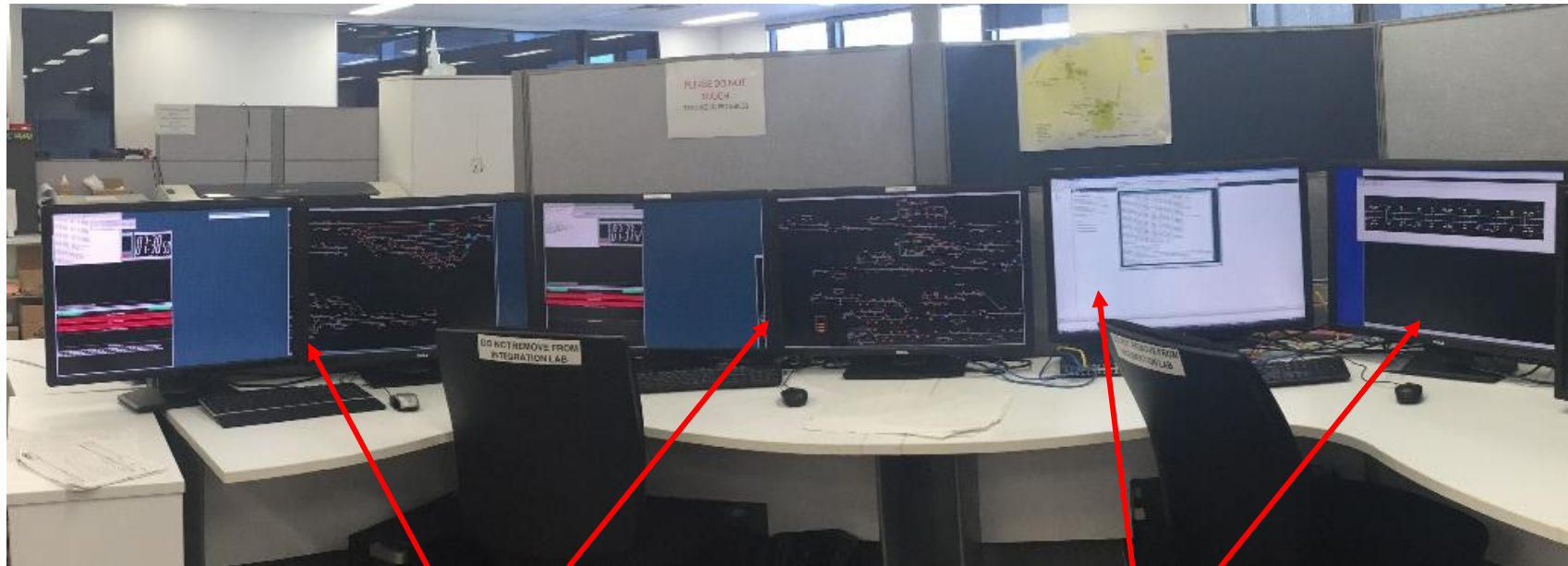


Figure 31 Test bench set-up in Brisbane

Human-Machine
Interfaces

Test Management
Monitors

12 APPENDIX E: FIP TESTER WORKFLOW DIAGRAMS

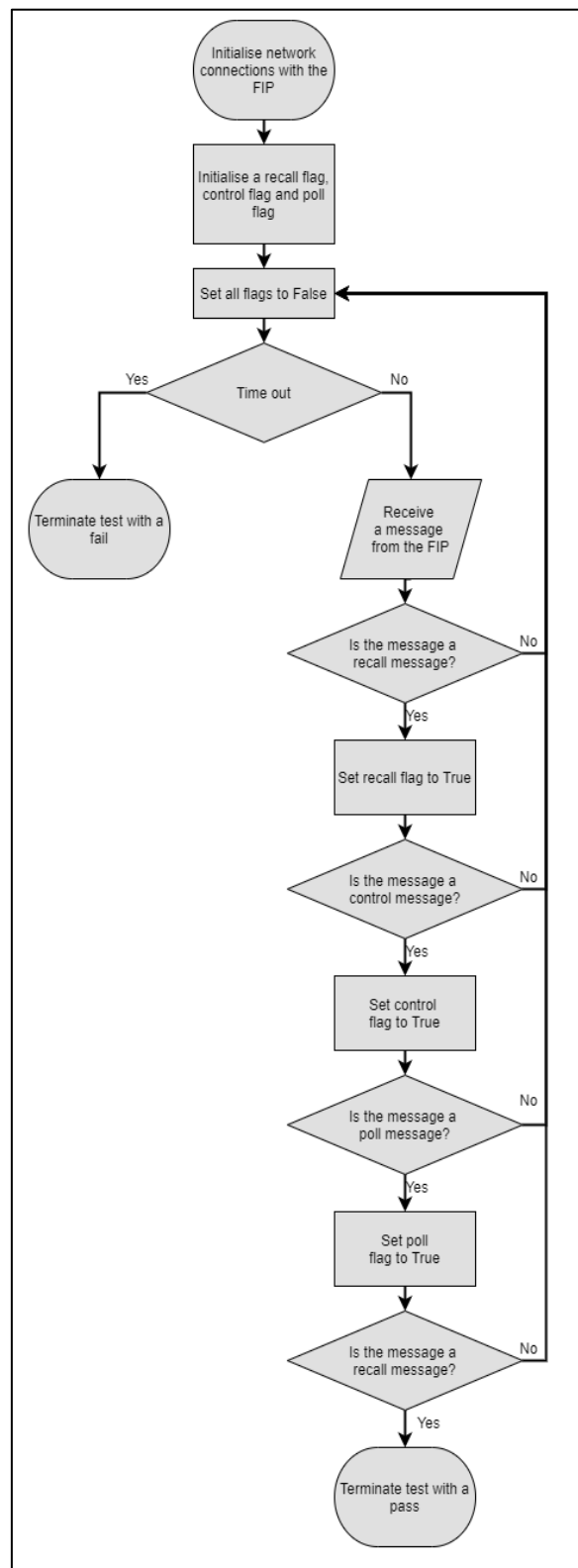


Figure 32 Flowchart of process to test initial testing sequences

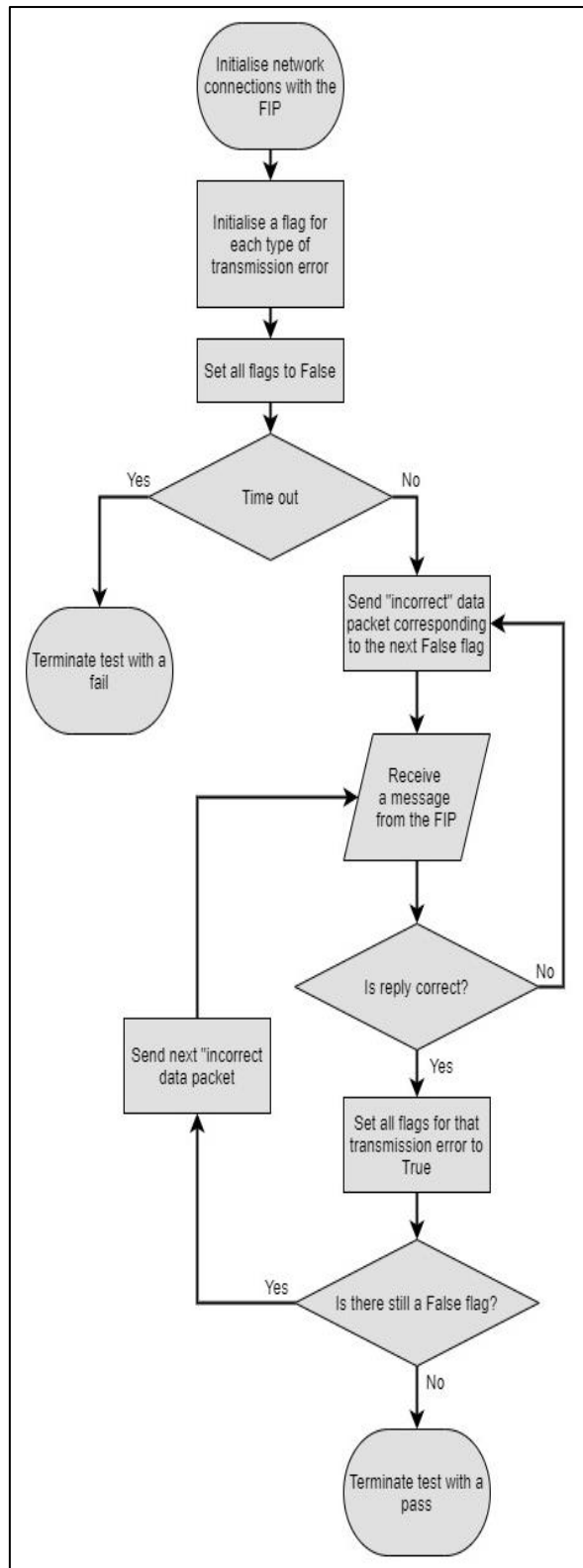


Figure 33 Flowchart of process to test site-like transmission errors

13 APPENDIX F: FIP TESTER TESTS

```
def test_startup_sequence(ixl, testConfigList):

    logger = ixl.get_IXL_log()
    network = ixl.get_IXL_network()
    startupMessageFlags = [False, False, False] # Flags: recall msg, control msg, poll msg

    count = 0
    while True and count < 5:
        try:
            incomingMsg = network.receive_from_fip()

            if incomingMsg:
                startupMessageFlags = handle_incoming_messages(incomingMsg, ixl, network, logger, startupMessageFlags)
                if startupMessageFlags[0] and startupMessageFlags[1] and startupMessageFlags[2]:
                    count = 5
                    break
        except OSError as e:
            logger.error(e)
            break
        # Increase count & resend message
        count += 1

    if not startupMessageFlags[0]:
        logger.error("Error receiving first recall message from the FIP.")
    elif not startupMessageFlags[1]:
        logger.error("Error receiving first recall message from the FIP.")
    elif not startupMessageFlags[2]:
        logger.error("Error receiving first recall message from the FIP.")
    else:
        logMsg = "Start up tests passed for " + ixl.get_IXL_name() + "."
        logger.info(logMsg)
        print("All Startup Tests Passed For", ixl.get_IXL_name())
```

Figure 34 Start-up test code snippet

14 APPENDIX G: TCS TESTING SUITE ARCHITECTURE

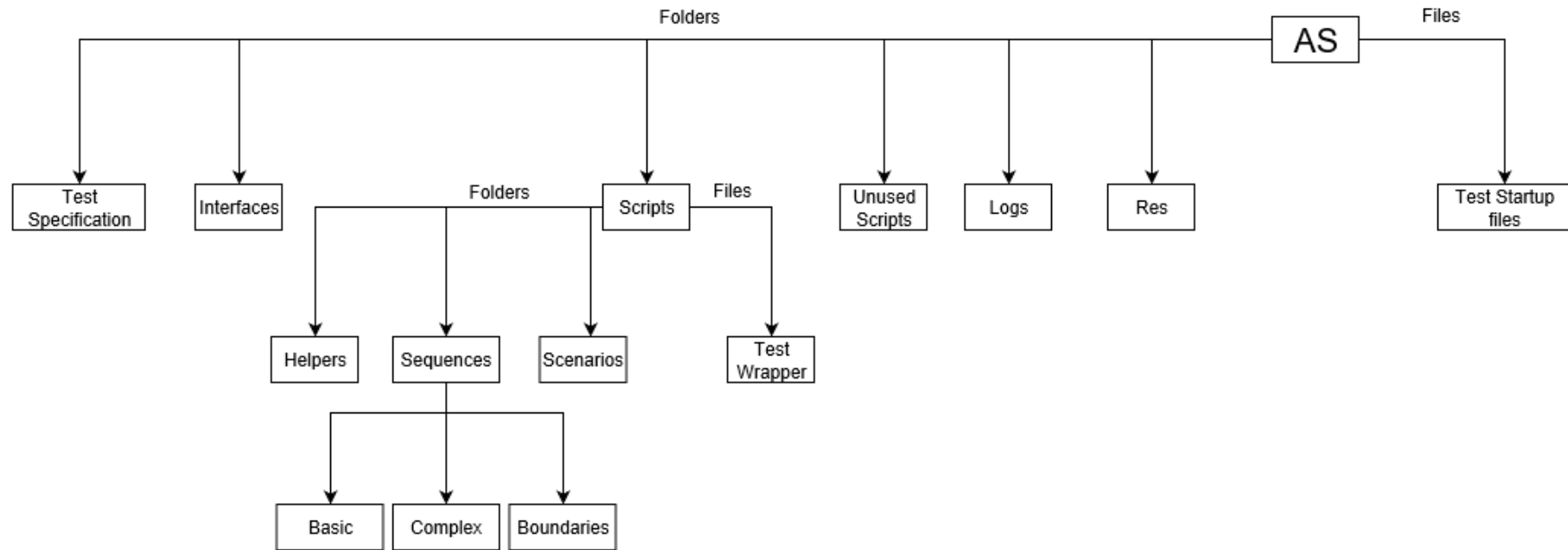


Figure 35 TCS Testing Suite architecture diagram [32]