Programme Area: Smart Systems and Heat

Project: EnergyPath Operations

Title: EnergyPath Operations – EPO Verification and Validation Strategy

Abstract:

This report explains how individual software modules (EPO and model components) are tested against their requirements. This is not a strategy for testing or performing analysis on the integrated model. This is explained in the "EPO Analysis Plan and Results" document.

Context:

DNV GL and a partnership between Hitachi & EDF worked independently on a functional specification to develop the first phase of EnergyPath Operations - a software tool that allows designers to better understand the information and communications technology (ICT) solutions they will need to implement to deliver new home heating solutions.
A first version of this tool is now being developed by DNV GL and the Energy Systems Catapult. EnergyPath Operations will provide knowledge to users on how to design ICT systems, the cost implications of such designs and the viability of various systems.
This project compliments the EnergyPath Networks software modelling tool which will be used in the planning of cost effective local energy systems.

Smart Systems & Heat (WP3)

Future GB Systems Architecture & EnergyPath ® Operations

Verification and Validation Strategy

Sub-deliverable ID WP3-7

Contents

1 Document Control

Type:	Test Plan
Title:	Smart Systems & Heat (WP3)
ESC Project Number:	Future GB Systems Architecture & EnergyPath® Operations Verification and Validation Strategy ESC00050
Version:	ESC00053 1.0
Status*:	Released
Restrictions**:	
Release Date:	May 2018
Filename:	EPO_Testing_Strategy
Review and Authorisation	

	Name	Position
Author	Justin Okoli	Test Engineer
Reviewer	David Wyatt, Daniel Mee	Software lead, Technical Lead
Authoriser	Daniel Mee	Technical Lead

Revision History

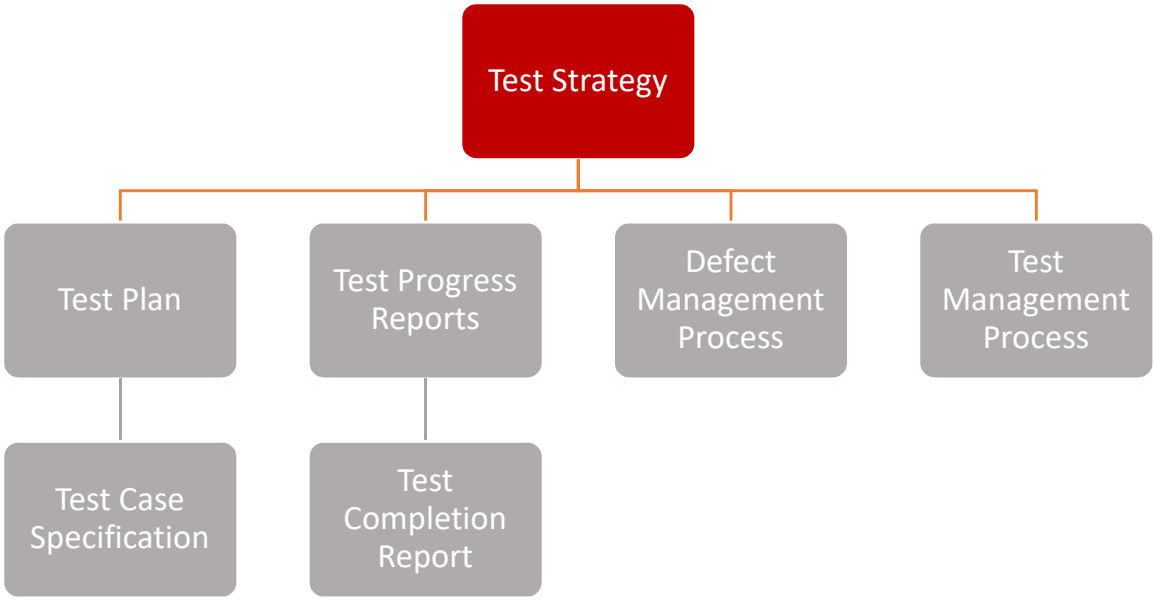
Date	Version	Updates	Comments
June 2018	1.0		

* Status defined as follows – **Draft**: Contains preliminary information only. **Released**: Contains reviewed and approved content.

** Restrictions defined as follows: **Public**: Regarded as "within the public domain"; **Confidential**: Contains confidential information and comprises intellectual property rights, including copyright, belonging to or licensed to other parties; **Confidential (R):** As Confidential, however certain information or data has been removed due to confidentiality, commercial, or license requirements. To request access to the full (Restricted) version, please refer to the document provider Energy Systems Catapult Ltd.; **Restricted**: As Confidential, however additional restrictions apply (as detailed in this chapter) due to confidentiality, commercial, or license requirements.

Document hierarchy

The diagram below illustrates how this document, Test Strategy fits in with the hierarchy of Test documents.



Terminology

This document uses the International Software Testing Qualifications Board (ISTQB) standard glossary of terms used in Software Testing found here:

http://www.istqb.org/downloads/send/20-istqb-glossary/186-glossary-all-terms.html

Glossary

Actor	A general (modelling) term that refers to system of interest. Energy Service Providers, HESG and Consumer are examples of Actors.
Interface	Used in the same context as Actor
Sprint	Iteration of Agile Incremental Development process
Agile	Software development process where solution evolves through collaboration between self-organizing and cross-functional teams.
Feature complete	A feature complete version of a piece of software has all of its planned or primary features implemented but is not yet final due to bugs, performance or stability issues.
Code Complete	A software release is called code complete when the development team agrees that no entirely new source code will be added to the release.
Regression-averse Test Strategy	A Test Strategy whereby the test team applies various techniques to manage the risk of regression such as functional and/or non-functional regression test automation at one or more levels.
White Box Testing	Testing based on an analysis of the internal structure of the component or system. For example, Unit testing code.
Black Box Testing	Testing, either functional or non-functional, without reference to the internal structure of the component or system.
Stub	A skeletal or special-purpose implementation of a software component, used to develop or test a component that calls or is otherwise dependent on it. It replaces a called component.
Mock	An object that is given a specification of the messages that it must receive (or not receive) during the test if the test is to pass.
Waterfall Model	The waterfall model is a sequential (non-iterative) design process, used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of conception, initiation, analysis, design, construction, testing, production/implementation and maintenance.
Jenkins	Continuous integration open source automation server
Jira	Agile project and defect management

Abbreviations

ERS	EPO Requirement Specification
EPO	EnergyPath® Operations, name of the tool been developed
ISTQB	International Software Testing Qualifications Board
ESC	Energy System Catapult.
CI	Continuous Integration Build Process
DevOps	Development Operations
JSON	JavaScript Object Notation
STF	Custom Simulink Test Automation Framework developed internally be the EPO test team.
Shift-Left	Software testing technique where testing is started early in development phase.

Table of Figures

2 Executive Summary

The primary objective of this Test Strategy is to define the test approach and rationale to the intended audience, the Project Delivery Team and Stakeholders; to enable successful delivery and acceptance of the EPO tool.

The Test Strategy will serve to govern how EPO tools requirements will be verified by the various test levels and types by applying Test Design Techniques, descriptions of the test infrastructure required to perform testing and the supporting test procedures.

The software development methodologies adopted to deliver the EPO tool are the V-model and Agile using the Scrum framework. One of the benefits to adopting a hybrid V/Agile model is that there is a cooperative phase where development, modelling and testing work in parallel but together forcing a 'Shift-Left' to testing: i.e. where testing is performed earlier in the development cycle

'Shift-Left' testing is made possible by key enablers as defined in this Test Strategy;

- Early test approach
- Continuous feedback
- Continuous Integration
- Regression-averse strategy
- Test automation approach

3 Test Strategy Objective

This Test Strategy sets out to meet the following objectives:

- Introduce the background to EPO project.
- Introduce the test approaches and demonstrate how the Hybrid Energy System Catapult (ESC) V/Agile model and the 'Shift-Left' model aims to detect defects in earlier phases where the cost to rectify defects is much less than in later phases.

4 EPO Project introduction

4.1 EPO project Background

The EPO tool is a set of software capabilities to enable the design and simulation of new Great Britain energy value propositions to gain insight into the interaction of different actors, businesses and processes that will underpin the new system architecture. It will simulate the key characteristics of different layers of the system, analyse the interaction between the layers and provide answers in quantitative and qualitative terms to shed light on numerous energy system related questions.

The EPO sets of actors and their capabilities are modelled primarily in MATLAB Simulink® with some data (initialisation, parameterisations) pre-processing done via python.

EPO system level requirement gathering and specification are currently ongoing activities, handled by software engineering requirement team working to develop the specifications that govern the sets of capabilities to be developed for the EPO tool. The tool will incrementally evolve by aggregating sub-sets of the identified capabilities from the system level specification, translating them into EPO requirement specification (ERS) and these capabilities will then be developed into a specific EPO tool version using Agile development methodology split over multiple Sprint iterations.

The sets of capabilities in each ERS for a Sprint is chosen to allow consistent and systematic evolution of the EPO tools. This allows incremental retaining of the tools capabilities across various versions. Each version builds on the previous version capabilities.

It is envisaged that this incremental scenario approach will maintain backwards compatibilities between scenarios, but it is recognised that due to changing nature of the system requirements, there may be alterations and changes occurring across EPO versions. These changes will be minimised or even be eliminated over the mid to long term as the EPO tools matures.

4.2 Scope

Verification - Confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled[1].

Validation - Confirmation by examination and through provision of objective evidence that the requirements for a specific intended use or application have been fulfilled[2].

4.2.1 Verification

Requirement based verification methodology will be applied to verify that the Actors/Interface features meets the approved EPO requirement specification (ERS).

The testing processes to be applied will only focus on proving that the functionalities within the actors/models/interfaces/functions are produced to meet the specified requirements as documented in the ERS.

Verification will answer questions like:

- Did we build the systems right?
- Did we build the system according to the requirement specification?

4.2.2 Validation

System level testing based on User level cases and scenarios will be applied to provide evidence on the suitability of the interfaces to its objective.

Validation will answer question like:

- Did we build the right system?
- Did we build the system based on the right requirement specification?

5 Test Concept

5.1 Hybrid V/Agile model

The EPO development process is a hybrid model [Figure 1 Hybrid V-Model/Agile Scrum Framework], combining the ESC software engineering V model requirement gathering to the Agile Scrum framework for the actual development of EPO.

The hybrid V model/Agile model shown below underpins the verification process.

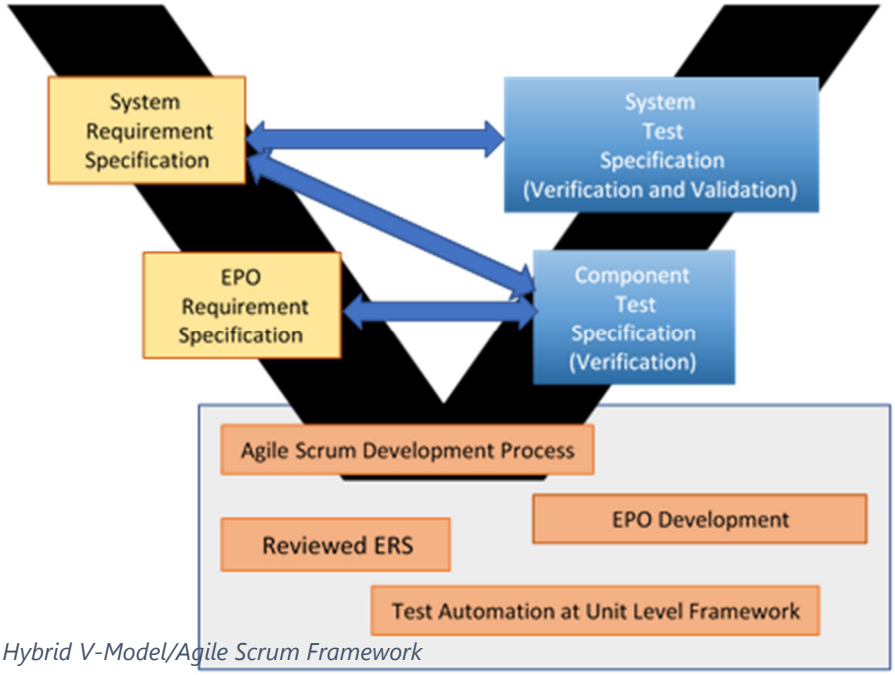


Figure 1 Hybrid V-Model/Agile Scrum Framework

1

System requirements will be elaborated into user scenarios and subsequently captured as EPO requirement specification which is split up into multiple Agile Sprints during Sprint planning. V-model requirements and specifications may be refactored and updated through continuous feedback during the development phase via Agile Sprints. These changes will sequentially flow back into the Agile development process. This increased agility in continuous feedback to V-model requirements is expedited by the hybrid model. The right-hand branch details the corresponding test activities.

During the development phase, user scenarios captured as ERS's are developed into Simulink Models. The approved ERS's provide the test basis for the development of tests.

As ERS development is ongoing, or partially completed, Test activities (review ERS, Test Plan, Test Case development, etc.) and development work can start. Once development work is "code-completed" formal Component Integration Level Testing will be performed. Once development is "feature complete" formal System Testing will be performed.

The benefits realisation of the hybrid V/Agile model is a fundamental 'Shift-Left' to software testing uncovering defects earlier in the specification and development phases when they are less expensive to fix.

5.1.1 Shift-Left Testing

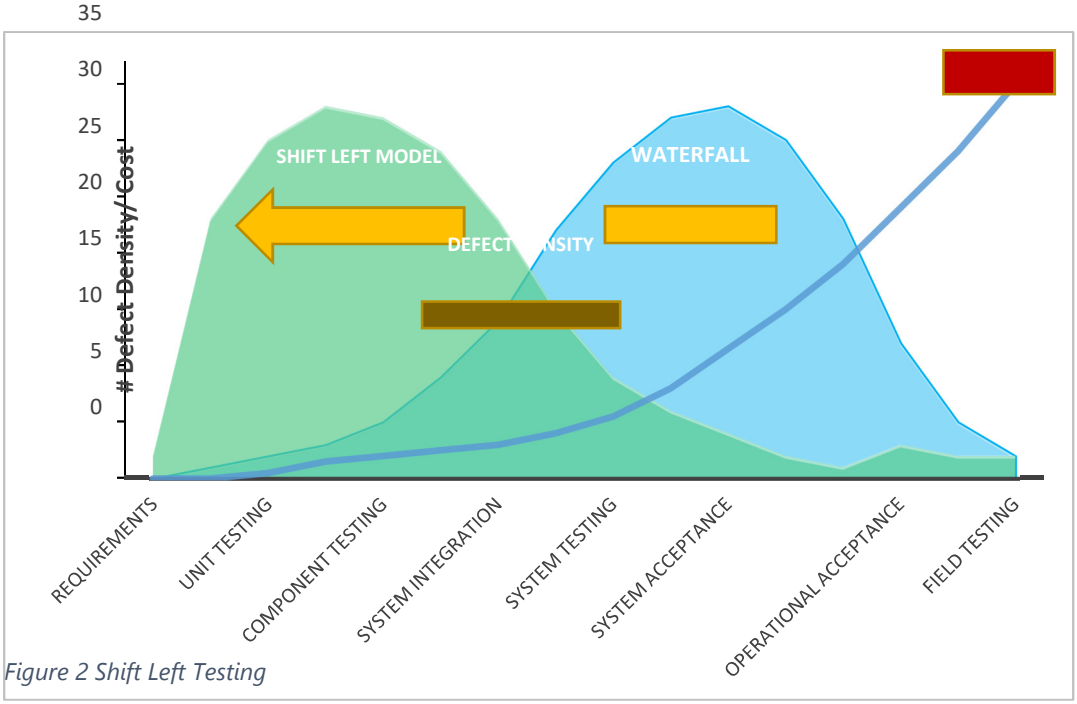


Figure 2 Shift Left Testing

The benefits to 'Shift-Left' testing is illustrated in the area graph of Figure 2 above. Defect detection rates tend to be higher in earlier development phases because of the implementation of the under listed test steps which aligned with the hybrid V/Agile Model, this is made possible by:

- Requirement specification is iteratively and statically reviewed by the testing team, starting off early in the specification cycle and as soon as draft of the requirement specification is available. Test review comments and results are communicated to the requirement team. A follow-on walkthrough may result were the review findings are discussed with the requirement team and decisions taking may be fed back into the requirement specification process.
- Test Cases development starts as soon as sufficient progress is made with the requirement specification, individual requirements are again evaluated to check that sufficient information and data are provided to aid development of appropriate test case. Result of this process is fed back to the requirement team in cases where additional information is needed.

- As development delivers potential reusable code at the end of Sprint iteration, tests are performed to uncover issues occurring at the Unit level and the result of this is fed back to the development team.
- Test Cases developed at the Unit level are deployed in the continuous integration build automation pipeline, during every Sprint commit to the repository, these tests are run as part of the build automation process, serving as regression test suites to detect issues and possible changes occurring across the different versions of the tools.

5.2 Assumptions, risks and constraints

5.2.1 Assumptions

This Test Strategy champions 'early system integration' where possible; however, the overall guidance and detailed approach to be taken to de-risking Component Integration is to be worked out during Sprint planning.

Requirements management, change management and release management are supporting processes which, although mentioned in this document, are not directly under the control of the EPO Test Team. Their operation is aligned with the time lines of this Test Strategy.

5.2.2 Risks

EPO tool is a work in progress research project and not all its future trajectory is currently known, the current Test Strategy is based on the current understanding of capabilities of EPO tool which may be substantially different in the future.

5.2.3 Risk management

Risk based testing is not being employed by this strategy as any risk should be managed and be transparent at an Agile Sprint level to the 'whole team'. These risks should guide testing and enable planning decisions to be made based on evaluating these risks. For example, there may be technical risks raised at Sprint level that may conclude in increased testing in a specific functional area or component. Tests can therefore be tailored to explore and/or confirm that specific risk.

6 Test Level, Phases and Types

Test Levels

A group of test activities that are organized and managed together. A test level is linked to the responsibilities in a project. Examples of test levels are component test, integration test, system test and acceptance test[3].

Test Phases

A distinct set of test activities collected into a manageable phase of a project, e.g., the execution activities of a test level[4].

The test levels detailed below are aligned to the ESC V/Agile model. As components or interfaces are developed, a corresponding level of testing is planned. Tests levels prevent the overlap of tests types; hence the tests are designed specifically for the various test levels. The test levels also promote early testing during development. This early testing is accelerated as described earlier with 'Shift-Left' enablers and testing during Agile Sprints.

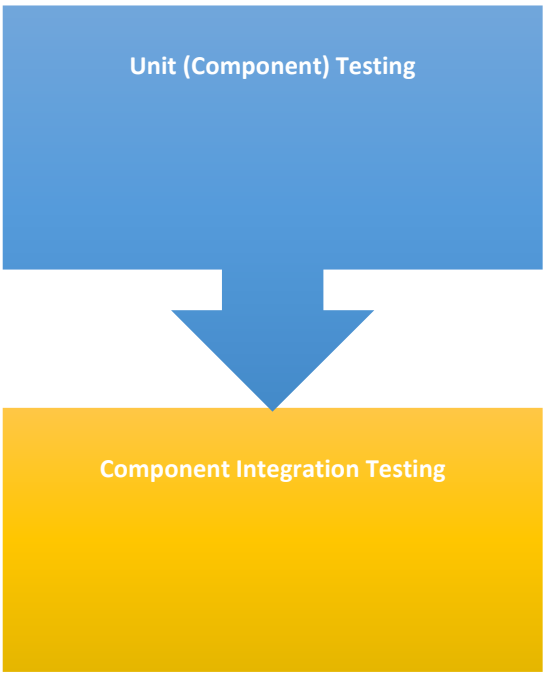


Figure 3 Test Level

These successive test levels also act as logical Quality Gates.

Quality Gate - A special milestone in a project. Quality gates are located between those phases of a project strongly depending on the outcome of a previous phase. A quality gate includes a formal check of the documents of the previous phase[5].

The outcome of testing from any test execution phase is evaluated against Test Exit Criteria (see Section 6.4.2) and can be documented in the form of a test completion report. This test exit criterion does not determine progression to the next test execution phase. The entry criteria to other high-level test execution phases are independent of the outcome of this test phase.

Below is a swim lane illustration of the planned sequential phases with quality gates [Figure 4].

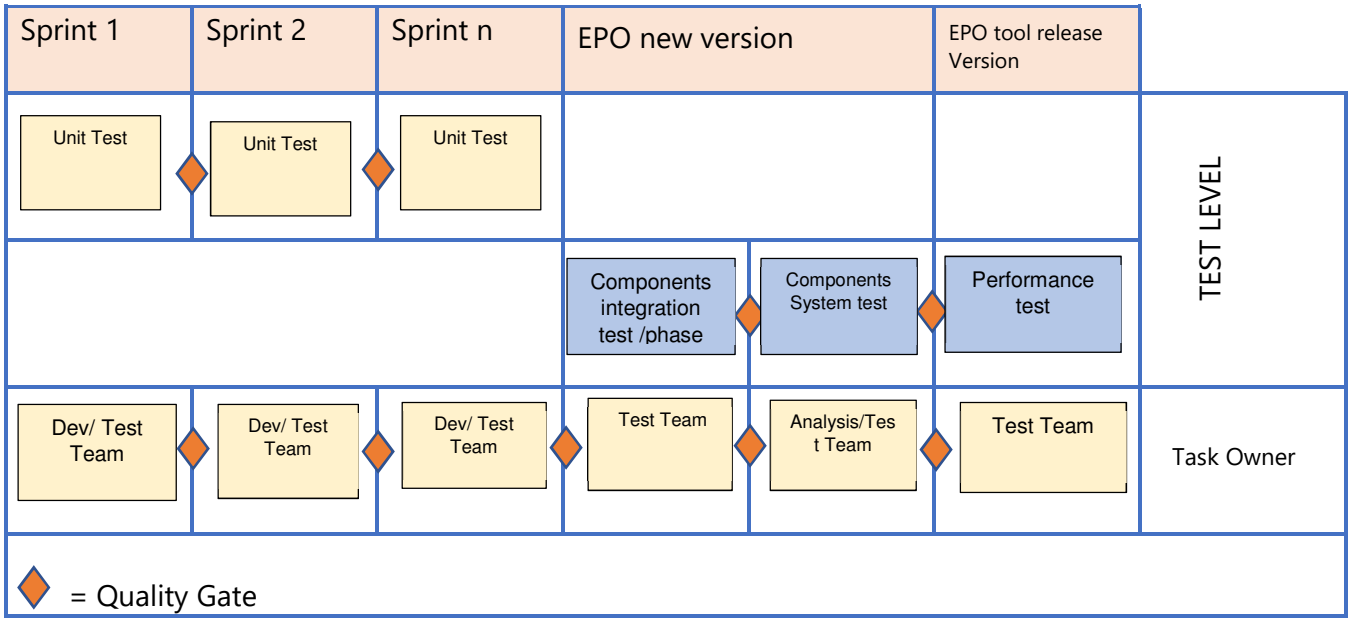


Figure 4 Test level, quality gate and Task Owner

6.1 Test Level

6.1.1 Unit (Component) Testing Level

Synonyms: Unit Testing

The testing of individual software components[6].

Component or Unit tests are 'technology- facing' tests that support programming verifying that there are no errors in the logic. Unit testing is a 'White-Box' testing technique.

When development starts, the items of the requirements are first implement as model function library, they form the basic unit of the EPO tools that can later be integrated together. Each function will have associated test(s) to verify its behaviour, the test coverage and other success criteria are defined during Agile Scrum task planning activity usually done in Sprint 1 when development first starts.

Model based Unit Test Harness are primarily developed in MATLAB® to test the function library. Each Unit Test Harness is associated with a model function library and the Test Harness is an independent replica of the associated model functions containing a separate model workspace and configuration set. However, it persists and is linked with the main function library. Changes made to the main function library are synchronised to the Test Harness.

During development, the Test Harnesses are connected with programmable Simulink test assessment and sequence blocks, these blocks provide the automated programmable interface that allow access to the input and output values of a function library from which the verification logics that governs the function library test are constructed.

Test Harness are configured to synchronise their content at compile time from the main function library. This insures that any changes to the main function library are auto-replicated to the Test Harness during execution and therefore subjecting any version of the function library to the same validation rule that is applied to the Test Harness.

Developed Test Harnesses are integrated into the Continuous Integration (CI) build pipeline as part of Agile Sprint completion deliveries. The CI uses the test automation libraries to provisions the required environment for automated test execution.

The CI integrated tests serve as regression test suites, as the function libraries are developed further and committed to the code repository (Git), they trigger the execution of the corresponding Test Harness by the CI servers.

Test Results are provided after test run, this provides an immediate regression feedback to the development team. Any issue identified can be rectified before resubmitting.

Successful execution of the regression test suites is a basic requirement for Agile Sprint completion deliveries to be approved and allowed to be integrated into the main branch of the source code repository.

Unit Test Workflow Demonstration:

Below is a Test Harness to test a hypothetical function that implements the requirement shown below:

Requirement 1.0.23:

A function shall be implemented to simulate the heat transferred between two thermal masses using the following equation:

E = K*(T1 - T2)

Where:

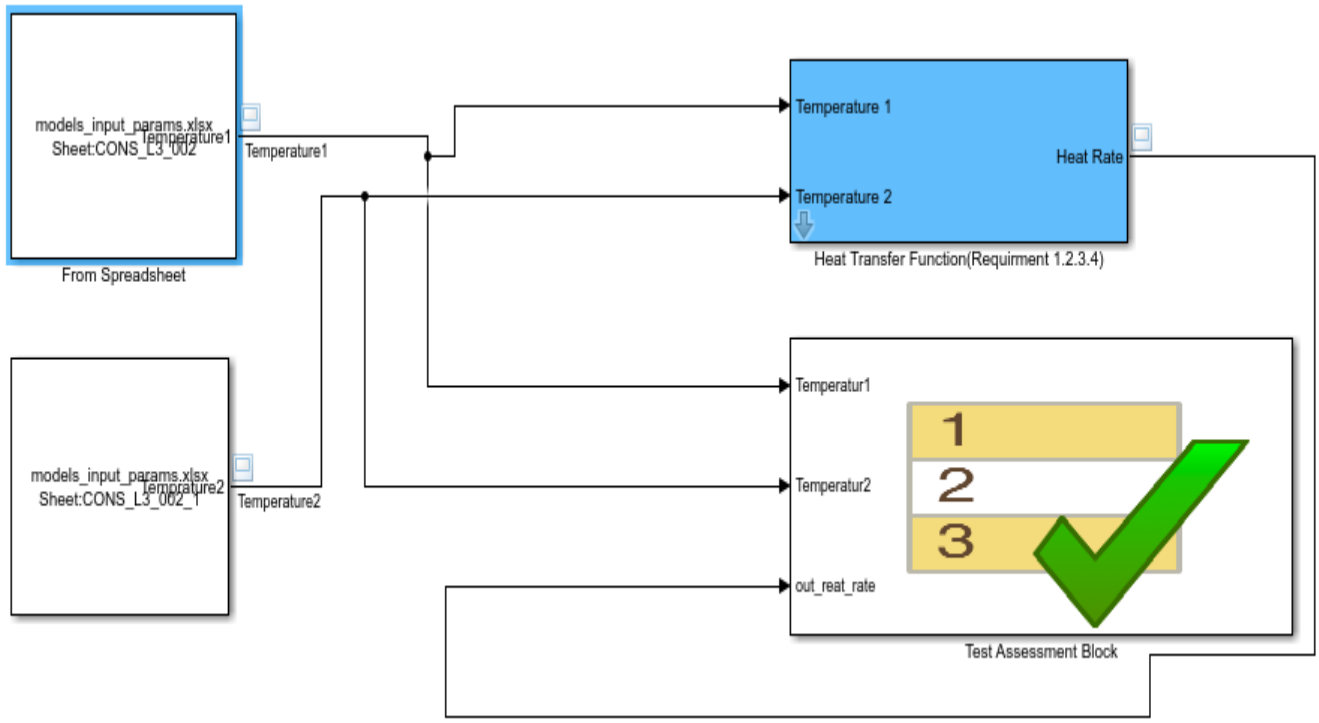
T1 is the temperature of thermal mass 1 (°C)

T2 is the temperature of thermal mass 2 (°C)

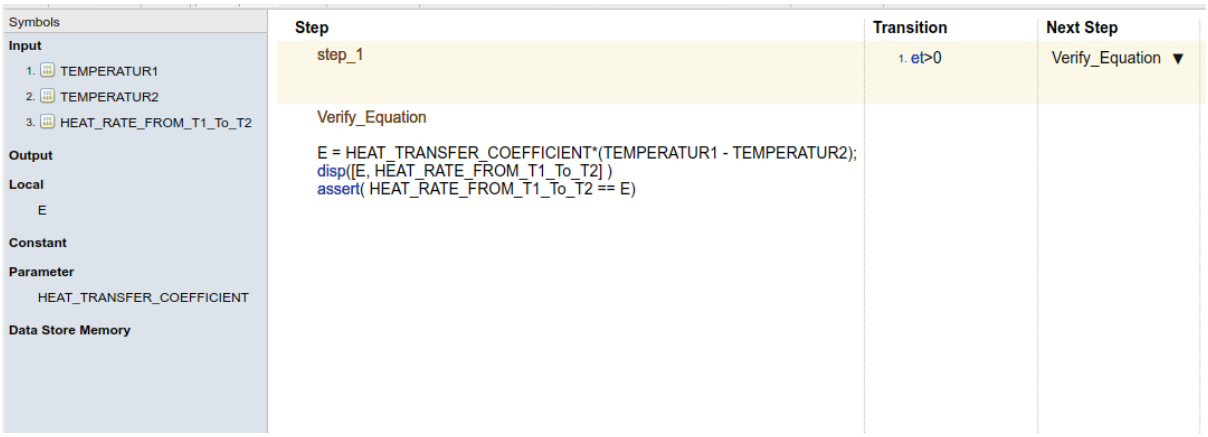
K is the heat transfer coefficient between thermal mass 1 and thermal mass 2 (W/°C)

E is the heat flow from thermal mass 1 to thermal mass 2 (W)"

In Figure 5 the Test Harness is constructed with assessment block connecting the inputs and output signals. Inside the assessment block, the signals properties are programmatically accessible and test verification logic is implemented.



Below [Figure 6] is the Verification logic that underpins the assessment. Signal values of Tempratur1 and Temperatur2 are wired to their corresponding assessment input parameter "TEMPERATUR1" and "TEMPERATUR2" respectively. The run time value of heat transfer coefficient, which is set in the function library workspace, is read into the corresponding assessment block parameter of the same name (HEAT_TRANSFER_COEFFICIENT). Then an assertion is constructed based on the equation giving in the requirement by replacing the temperature of the thermal masses T1 and T2 with the Signal values of TEMPERATUR1 and TEMPERATUR2 respectively.



Using this approach, guarantees that the relationship as defined by the equation will always be verified irrespective of the input source type, values or any change made to the internal logic of this function library.

6.1.2 Component Integration Testing Level

> *Testing performed to expose defects in the interfaces and interaction between integrated components*[7].

As per definition, Component Integration testing builds on and extends Unit testing to test integrated functions forming a component. This is a Black Box Testing technique.

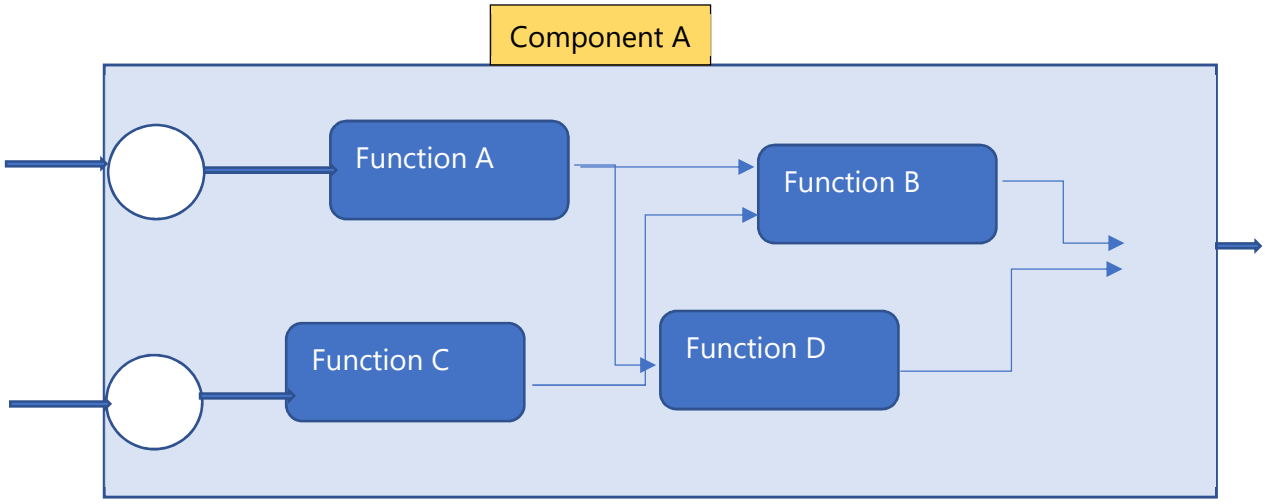


Figure 7 Component Integration Testing

During development, as the library functions implementing different features of a components are continually developed and tested, they will be logically connected to provide the features defined in the requirement specification

After development is code completed and integrated into the CI pipeline, the Component Integration testing is started. Requirement specification based verification is done using Test Cases developed from the component level requirement specification (ERS).

Test Cases for each component under test are developed to demonstrate test steps, test data and the expected result required to verify each item in the requirement specification. See Section 7 for details information on test case development technique that is applied.

Test Harnesses are generated out of every component under test. Hypothetical signals serving as input signals from are connected to the input ports of the component. These hypothetical input signals are generated with the same characteristics as the real signal, which would have otherwise connected the component under test with other interfaces it interacts with during use case analysis. The characteristics of these signals are defined in the Interface Control Documents (ICD), part of the requirement specification documents.

Simulink assessment and sequence block are then connected to the Test Harness to provide a programmable interface to the component's input and output signals. Using the Simulink Test tool blocks the steps of a test case can be programmatically applied and the expected result is verified.

As the Test Cases become too complex to be easily implemented programmatically, they are tested manually. Such Test Cases should be marked as candidates to be implemented in the future.

After the test execution and review process, the automated Test Cases are integrated into the CI pipeline, where they are configured by the CI server to run as regression test.

Component Integration testing is organised with Jira Zephyr in test cycles. During a test cycle are the approved and required Test Cases are aggregated into a named Jira Zephyr test cycle. Using Jira Zephyr cycle enables test execution result, test run history and test artefact to be managed centrally. Results of individual test case execution are stored and can be retrieved.

Defects are opened directly from the test cycle for any test execution failure, this provides traceable link between the defects and the failed steps of the Test Cases. Trace Logs and other evidence are aggregated and attached to the defects.

Component Integration testing will take place in local workstations using MATLAB® and Simulink Libraries and Test Toolbox.

Test Specification and artefacts will be integrated into the CI Pipeline.

6.2 Test Phase

Component Unit Test Level will be done during the implementation phase within the Agile Sprint Iteration. While Component Integration level testing will be done after the code completion.

6.3 Test Types

Test Types

A group of test activities aimed at testing a component or system focused on a specific test objective, i.e. functional test, usability test, regression test etc. A test type may take place on one or more test levels or test phases[8].

Functional Testing: Testing based on an analysis of the specification of the functionality of a component or system[9].

6.3.1 Functional Tests

Functional tests will be performed during the Component Unit and Integration Test Level, the functional verification process will be requirement specification based.

- Component Integration Test Level

Black box methodology will be applied to the component, the components will be tested by feeding them an input and examining the output based on the specifications of the requirement. No additional verification will be performed outside that defined in the ERS (see Section 6.1.2).

- Unit Test Level

White box methodology will be applicable to the Component Unit Level Testing. The components will be tested by feeding them an input and examining the output based on the transfer functions or equation which underpins the functional behaviour. The scope of this test may be limited to component that have function library (see Section 6.1.1).

6.3.2 Non-Functional Tests

Performance Test is a Non-Functional Test. This test will test the performance of the EPO tool against specified performance Requirement. This test will be developed further in the future once the performance requirements are developed.

6.4 Test Phase Entry, Exit and Suspension Criteria

The criteria detailed below act as Quality gates to all test phases. NOTE: The Test Phase Entry and Exit Criteria may be refactored, specific to the test level.

6.4.1 Test Phase Entry Criteria

- Definition of Done met for all In Scope requirements specifications.
- Phase Test Plan in an Approved status.
- Planned Test Cases in an Approved status.
- Test Environment configured with components deployed via CI process.
- CI release meets Quality Criteria and results published.
- Automation Test tools and frameworks configured for environment and ready.
- Test progress reporting format and frequency agreed.
- Tests cases prioritised in accordance with this strategy.

6.4.2 Test Phase Exit Criteria

- 100% of planned tests executed, and any exceptions documented with reasons.
- All defects observed, raised and assigned to a future planned release for retesting.
- Test Results documented and evidence captured.
- Regression testing completed.

6.4.3 Test Suspension and Resumption Criteria

- Test environment is not stable.
- Emergency release or patch is required that includes fixes for previously known High Severity defects. A new cycle of tests to be started to resume testing.
- Significant changes to infrastructure or development during a test cycle automatically suspends testing. A new cycle of tests to be started to resume testing.

7 Test Process

The fundamental test process comprises Test Planning and control, test analysis and design, test implementation and execution, evaluating exit criteria and reporting, and test closure activities.[10]

Figure 8 Test Proces

This systematic process provides consistency in planning, analysis, design and execution at different levels. These stages should be followed in this logical order to maintain a quality approach to testing the EPO tool.

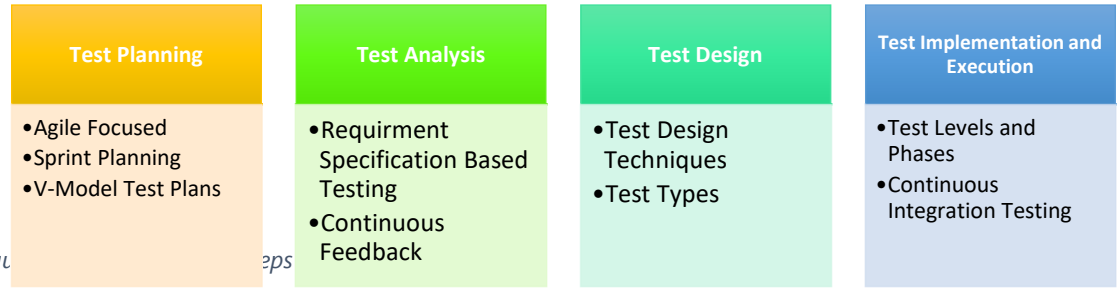
For each test level, identifying the objectives and scope of testing, requirements coverage, risks, approach, infrastructure requirements, schedules for testing, progress reporting, evaluation of results should be defined in a test level Test Plan.

Once each test level Plan is complete, test case development can begin. The context of the Test Cases should reflect the Agile methodology. Jira Zephyr is currently used as test management tool that is integrated with Jira (Agile project and defect management tool). All Test Cases should be traceable to the appropriate level of requirements as shown in the V/Agile model.

Tests cases are then executed in Zephyr test cycle and the results recorded. Any defects found are reported in Jira for rectifying. Finally, once test execution is complete, a test level completion report is generated from Zephyr, documenting the results of testing.

To verify, validate and ensure the EPO tool is delivered to a high quality it is essential that testing at all test levels is effective in discovering possible failures.

The adoption of ISTQB standardised test techniques, types and levels ensures increased test coverage and best practice is applied when planning and developing tests for the EPO tool testing. The illustration below [Figure 9] summarises the test process:



Figu eps

One of the main approaches to Test Planning, analysis and design adopted by this strategy is requirements-based testing.

7.1 Requirements Based Testing

An approach to testing in which Test Cases are designed based on test objectives and test conditions derived from requirements, e.g., tests that exercise specific functions or probe non-functional attributes such as reliability or usability[11]

To verify and validate EPO tools system level functional and non-functional requirements are met, all Test Plans should define which requirements are in scope for testing. Traceability back to requirements will demonstrate which requirements are being satisfied through testing. As the V/Agile model shows there is continuous feedback from the Agile development stages to the V-model requirements. This continuous feedback can result in refactoring of requirements to include clarification and change requests.

Static analysis of requirements through reviews and inspection can be deployed to uncover issues which may lead to defects manifesting themselves in the development phases. The aim is to prevent defects occurring in the first instance. Scenarios to include valid, invalid and edge cases can uncover behavioural or technical issues.

7.1.1 Requirements coverage using traceability matrices

A two-dimensional table, which correlates two entities (e.g., requirements and Test Cases). The table allows tracing back and forth the links of one entity to the other, thus enabling the determination of coverage achieved and the assessment of impact of proposed changes[12]

The EPO tool must completely satisfy the set of requirements. To evaluate the level of coverage, every test case will be traceable to at least one requirement.

Test Cases will be mapped to the specific component requirements specification (ERS) by using a 'Requirements Traceability Matrix'. This is implemented as Jira Zephyr Test Case "Requirement ID" field which should contains the reference ID of the specific requirement verified by the test. All other specifications should also be referenced. The relationship of a test case to requirements can be 'one to one', 'one to many' and vice-versa. An example requirements traceability matrix is shown below.

Test Case	Req #1	Req #2	Req #3	Req #4	Req #5	Req #6
SYSTC#01	✓					
SYSTC#02		✓				
SYSTC#03			✓			
SYSTC#04			✓	✓		
SYSTC#05					✓	
SYSTC#06		✓			✓	✓
SYSTC#07						✓
SYSTC#08	✓	✓	✓			
SYSTC#09				✓	✓	✓

Fig

7.2 Agile Scrum Sprint Testing

During Agile Sprints development, User Scenarios are written to indicate the behaviour which must be met for the User Story to be Done. These behaviours are captured in a Requirement Specification (ERS) which serves as the basis to verify the Sprint deliveries using requirement based testing (see Section 7.1).

Depending on the component or features under development, Unit or Component Integration tests can be created. The appropriate test level is applied to ensure the acceptance criteria has been met. Test Cases will be developed in Jira Zephyr (test case management tool integrated with JIRA) to meet specification Requirement and traceable to the Sprint level ERS.

7.3 Defect Management

Atlassian Jira will be used to manage defects identified during test phases. This section will be updated with more information detailing the Change Management Process to be adopted for defect categorisation and resolution workflow. At the time of writing this strategy, Change Management Process development is still ongoing.

7.4 Test design techniques

Test Design Techniques

Procedure used to derive and/or select Test Cases.[15]

The purpose of a Test Design Technique is to identify test conditions, Test Cases, and test data. Test design techniques can be used to verify for example, data flows and transformation through component interfaces, error handling system exceptions, identifying edge cases and negative tests. The categorisation of techniques and associated techniques are shown below [Figure 11].

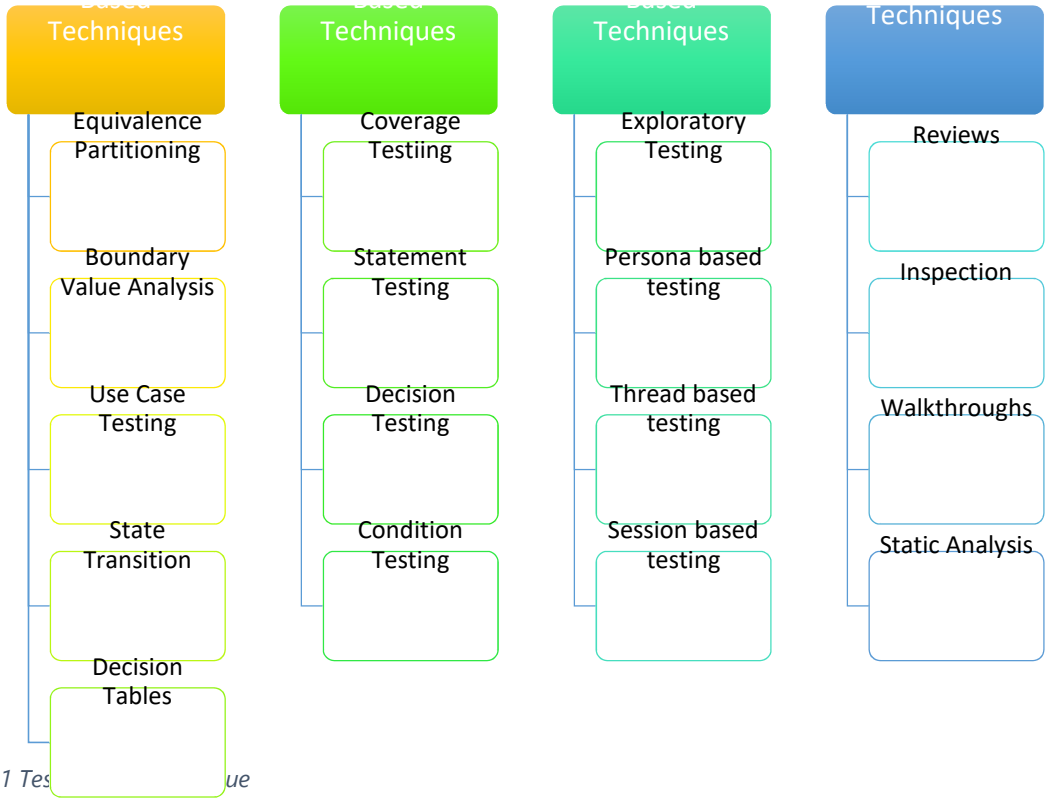


Figure 11 Tes ue

The techniques above are shown here as a quick reference guide to list some of the test techniques (but not limited to) that should be considered in test case analysis and design, test data preparation and coverage techniques. Specification based techniques can be applied to the EPO tool test levels. For example, the boundary value analysis technique can be used to verify data validity across components. As the test techniques are well documented in widely available ISTQB literature, no further descriptions are being provided in this strategy (see for details: https://www.istqb.org/downloads/send/20-istqb-glossary/186-glossary-all-terms.html).

7.5 Re-Testing and Regression Testing

Testing of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software, because of the changes made. It is performed when the software or its environment is changed.[13]

The approach taken to regression testing is regression-averse. Automated regression tests will be created where possible, which will reduce the overall manual testing effort required during test execution cycles. As new features are developed and defects are uncovered, retests should focus on the feature where the defect was found and regression tests should include the surrounding features.

7.5.1 Regression Test Automation Approach

A realization/implementation of a test automation architecture, i.e., a combination of components implementing a specific test automation assignment. The components may include commercial off-the-shelf test tools, test automation frameworks, as well as test hardware[14].

Automated Test Harnesses are developed for the testing of the component in the different levels of testing, these tests are programmatically designed to capture issues which violated the requirement specification in all versions of the EPO where the requirements apply.

After test execution, these automated Test Harnesses are integrated into the source code repository branches and are configured to be executed for any code change occurring within that branch.

These configured test acts as regression test. When the software repository branch changes as result of incremental development of new features or bug fixes applied to the code base, these tests are executed by the CI servers to verify the changes.

As defects are discovered and fixed, additional automated Test Cases are developed and added to the CI pipeline to verify that the fixed defects are not re-introduced into future release.

Regression tests are scheduled to be autonomously, executed either during development phase or post development phase once there is a code change in the appropriate branch of the source code repository where the tests are configured. Test can also be manually triggered for executed or switch off entirely from been executed.

As the number of defects increases during development, the number of regression tests will also increase. To add to this, during Agile development phase, submission of code deliveries at the end of every Agile Sprint will increase the frequency of regression testing. This

facilitates faster regression test feedback in the form of Test Results which will guide the integration level Test Planning.

Regression automation approach ensures that there is accelerated feedback from failed tests in the form of new defects uncovered testing new features and secondly any defects that have regressed.

The various level of automated tests can be illustrated using the automation pyramid below [Figure 12]:

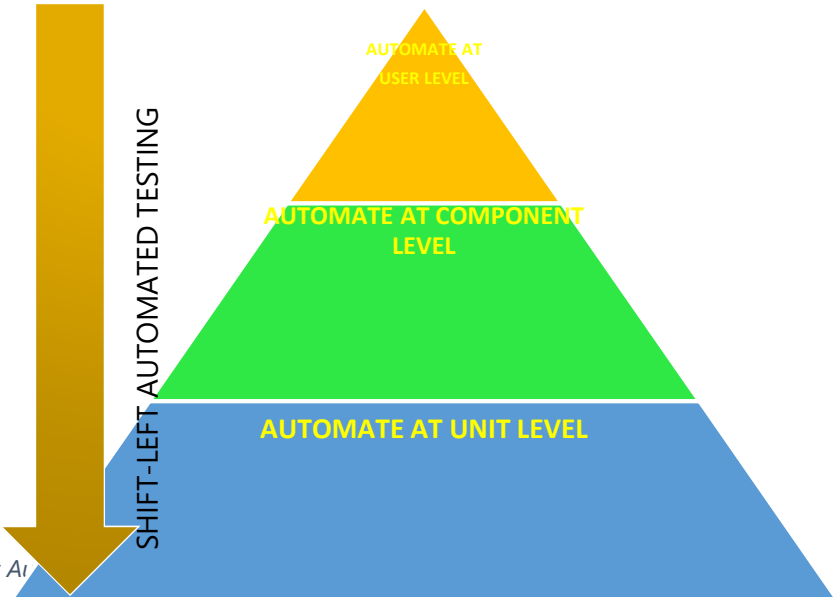


Figure 12 Test Au

As illustrated in the test automation pyramid, Unit level Test Harnesses are at the lowest level, they are the primary regression test to verify codes submitted to the source code management system. They will run greater frequently as codes are continuously submitted to the source code repository during the Agile Sprint iterations.

As component development is code complete, further automated tests will be written to include integration tests at component level. Tests covering different component variants and scenarios are added to the regression test suites. They are then scheduled to be executed in subsequent Sprint iterations whenever changes are applied to the component.

Using this approach enables regression test to be continuously performed at early stage of the development.

A Simulink Test Automation Framework (STF) using MATLAB® python engine is developed and implemented in the CI Pipeline. This provides the capabilities required to achieve this regression approach.

7.5.2 Regression Manual Test approach

Since not all Test Cases are automated, Manual testing is performed in all other cases were automation testing is not possible.

8 Validation Planning

8.1 Types of Validation

The term validation is overloaded, with multiple meanings. Here are the definitions of the term as used on the SSH1 WP3 Project. All forms of validation are different to verification, which is the assessment of an implementation's adherence to a specification.

1. **Requirements Validation**: Assesses if the specified functionality is correct. In other words "have the right requirements been specified and are they correct". On this project, there are two approaches to requirements validation. The first is by exercising peer-review of requirements prior to implementation. The second is through iterative feedback (once a model has been explored, shortcomings, issues and problems are fed back to inform the designer whether the requirements need amending).
2. **Model Validation**: Assesses if the models return representative answers. For example, if an electricity network model returns a value of "volts" at a certain location then is that value within an acceptable tolerance of what that real network would see? On this project, each of the designed models have the validation approach explained within the respective EDD. Some models are harder to validate than others particularly when they are predictive models of the future. For example, the allocation function uses probability distributions to allocate appliances to domestic properties in an attempt to predict future distributions of technologies, specifically plug-in electric vehicles and electrified domestic heating appliances. Since the future isn't known the validity of this distribution cannot be assessed. An additional element of Model Validation involves checking the accuracy of datasets to ensure that they are "valid" i.e. that they aren't excessively full of errors, duplications and omissions. This data validation is checked as part of the pre-processing activities prior to running a simulation.
3. **Product Validation**: Assesses whether the tool and integrated model meets the user's expectations. This is beyond verification, since captured user requirements will have been tested. On this project, product validation is primarily informally captured as feedback from having used the model and/or tool software and communicating updated needs. These desires might either be used to tweak the existing implementation where it's in scope or might result in generating brand new requirements. In the case of the model, the modelling analysts validate the integrated model by performing "by-eye" sanity checks on a reduced sub-set of outputs from the model and checking understanding of why the results look like they do. Where an unusual outcome is discovered then a deep-dive exploration occurs to manually validate why the result is as it is.
4. **System Validation**: Assesses whether the integrated model, constructed from verified components, hangs together and operates as intended. This validation is the check that the system architecture, previously specified, perform as needed and provides the evidence base (or otherwise) to suggest continuing with a given design

of a Future GBES. On this project, System Validation is the output of the analysis function.

8.2 Future Work / Plans

To date requirements validation is being performed as part of the peer review process and it is expected that this will continue.

Though the process for model validation is explained and understood not much validation has taken place as most of the models generated are still at the prototype stage. It is vital that the components are validated to build credibility that results from EnergyPath® Operations can be relied upon. Data validation will continue in the same way as now. In the case of future predictions then there are two main methods of validation. The first is peer review and the EPO project will seek to use external experts to assist in this operation to gain independent endorsement that the models are "sensible". The second is to align with other organisations who have similar, maybe competing implementations, e.g. SmartNet, that share common attributes and then perform comparative studies. Multiple, independently researched and built models which have a high level of correlation in output tend to validate one another.

Systems Validation is, arguably, what EPO exists to do and as the models become more sophisticated and integrated then the candidate system architecture(s) will be validated for what does (and doesn't) work.

Product Validation is also a continuous process and relies on feedback from the users of the product. Depending on the exploitation approach, and whether the tools and models are used by multiple organisations this process will need to be formalised and built in to development road-maps.

9 Test Infrastructure

9.1 Test Environments

Development of test environments is key to the success of the EPO tools. The provision of test environments will need to be planned to include access, representative data loaded to environment, system logging, monitoring and alerting of environments in place. Multiple test environments maybe required to allow parallel testing to occur, also to aid root cause analysis of faults against a release

The DevOps team will be responsible for creating and managing environments. This includes set-up of CI test infrastructure and deployment of releases through the CI Process and creation of multiple CI instances with relevant test data loaded.

9.2 Test Equipment

Standard ESC windows workstation and CI servers.

9.3 Software and test Tools and frameworks

The software and libraries available to support test and test automation are detailed below:

This list may be subject to change:

Tool	
Jenkins	Continuous integration open source automation server
MATLAB® R2017 with State flow	Framework to support Testing of models and state flow based model's logics
Simulink Test Toolbox®	Simulink Testing toolbox, providing programming interface to models harnesses
PyTest /Python 3	Algorithm testing and programming language for test automation framework development
MATLAB® python engine	MATLAB® interface to python objects and environment
Matplotlib	Framework to plot and analyse Signals
Invoke, pandas, xlrd, junit-xml	Test framework dependent python libraries.

Test Automation framework	Custom test automation tool
Jira Zephyr	Test Case Management Tools.

10 References

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 The International Software Testing Qualifications Board (ISTQB) standard glossary of terms.

http://www.istqb.org/downloads/send/20-istqb-glossary/186-glossary-all-terms.html