DIGITAL HEALTH AND CARE WALES

TEST FRAMEWORK

Document Control

Version Control

Version	Status	Primary Author(s)	Description of Version	Date Completed
1.0	Approved	Rachael Wettach		30/04/2009
2.0	Approved	Philip Samuel		01/09/2015
3.0	Approved	Rob Murray		29/03/2019
1.0	N/A	Geoff Norton	Replaced any references and logos to NHS Wales Informatics Service / NWIS with Digital Health and Care Wales / DHCW. Re-titled as *Test Framework* and restarted version numbering	07/07/2021
1.1	N/A	Geoff Norton	Changed the format of the Appendix 4 heading so it's consistent and appears in the Table of Contents Replaced the abbreviation with the full name in the Appendix 1 heading to make the intent clearer.	12/12/2022

Distribution Control

Version	Distributed By	Distributed To	Date Distributed
3.0	Rob Murray	D. Latham, K. Booth, J. Taviner, J. Forster. G Cudmore L. Porter	20/03/19

Reviewers & Authorisation

Type	Name	Company	Role	Date Complete
R	G. Cudmore	NWIS	Lead Test Analyst	29/3/19
R	D. Latham	NWIS	Lead Test Analyst	29/3/19
R	K. Booth	NWIS	Lead Test Analyst	29/3/19
R	J. Taviner	NWIS	Lead Test Analyst	29/3/19
R	J. Forster	NWIS	Lead Test Analyst	29/3/19
R	L. Porter	NWIS	Lead Test Analyst	29/3/19

Contents

Executive Summary

This document lays out the Test Framework and Methodology for Digital Health and Care Wales.

The Test Framework is centred on Risk Based Testing, its alignment with the Clinical Risk Management Process and the DHCS Validation Policy employed at DHCS. Testing is a crucial aspect to minimising risk with risk-based testing seeking to identify areas of an implementation where either failure is most likely or that are critical to the end user of the system and concentrating testing first and foremost on those. Alignment with the seven-step Clinical Risk Management Process is an essential aspect of the Framework and ensures the Framework is relevant, practical and aimed at the overriding principle of patient safety.

To complement a Risk Based Testing approach, where applicable, Requirement Based Testing will be considered to validate that the product meets the functional and non-functional requirements and therefore provides value to the organisation and/or user community.

DHCW's Test Methodology sees testing as a five step process being:

1. Strategising
2. Planning
3. Designing
4. Execution
5. Evaluating

Each step is vital to a successful testing outcome and each one results in documentation to support the test effort and provide stakeholders with the information required to make informed decisions on the state of the application.

In supporting the Test methodology DHCW is using Microsoft Team Foundation Server as its Test Management Tool.

The following templates are available for projects to use when generating test documentation. These templates are located on the DHCW Intranet site under Service Management:

1. Test Strategy
2. Test Plan
3. Test Summary Report
4. Test Certificate
5. Traceability Matrix

Possible test phases and types are presented in the DHCW Test Framework which, as a non-prescriptive framework, does not dictate which of these DHCW projects need to employ. Rather, it is proposed that projects firstly

a. Decide on their individual project delivery method then
b. Decide which test phases and types are relevant based on the delivery method

Essential features of successful testing are management procedures and methods that support testers and support the project and its reliance on accurate, timely information. To this end, the framework presents recommendations on the types of information the test team may produce throughout a project's lifecycle as well as ways in which defects can be logged and managed. In addition, advice on resources and control and release procedures is provided in the document.

It must be reiterated that this document is not prescriptive but provides DHCW projects with a common framework upon which to work, and is aimed to ensure consistency in terms of testing across the programme. As testing matures, this document may be amended to reflect future learning as well as any changes in the way projects and testing are managed. As DHCW continues its roll out of AGILE and the Scrum Methodology as its preferred software development approach so too will this document reflect this.

Introduction

Document Purpose

This document, the Digital Health and Care Wales (DHCW) Test Framework, is centred on the premise that testing should commence as early as possible in a project's life cycle. It is a known fact that as a project progresses, errors become increasingly costly to resolve. That is, an error picked up during a system's testing phase costs, on average, ten times more to fix than one found and fixed during the requirements phase. Finding an issue after production implementation can cost one hundred times more.

Subsequently, the DHCW Test Framework recommends principals, processes and techniques that may be utilised by DHCW to ensure testing commences early and that any deliverable developed or procured is robust, adheres to its functional requirements and is as defect free as possible.

This document is intended to:

1. Act as a guide for DHCW project members on what testing is required to verify and validate a deliverable in conjunction with DHCW's clinical risk management process
2. Act as a guide for any testers involved in the process of validation and verification within DHCW
3. Highlight the Risk Based approach to testing in line with the overall DHCW Clinical Risk Management process
4. Outline the various delivery methods in place at DHCW
5. Provide the reader with an understanding of the overall Test Methodology employed by DHCW providing:
    a. The high level test process
    b. Detail on the test strategy, planning, design, execution and evaluation phases
    c. Information on the various test phases a deliverable should pass through
6. Outline DHCW's control procedures and fault management methods

Terminology

Terminology is a contentious issue within the world of testing. Every industry, company, even individual utilises disparate terms to describe the same testing concept.

This document defines testing terms that are, on the whole, standard however the detailed descriptions may differ from those previously used at DHCW or those utilised by project members within other companies or roles.

It is recommended that a standard, common terminology be adopted within DHCW to avoid confusion between project and project members.

Testing Checklists

The majority of DHCW implementation projects require some form of testing whether it be throughout the development lifecycle, interface testing after the receipt of a bespoke solution from a third party or acceptance testing after the purchase of an off the shelf product.

As the planning and preparation of testing needs to be initiated as early as possible in the project's lifecycle, the PM (Project Manager) Testing Checklist has been provided to assist DHCW project managers in addressing the right test related concerns at the project's commencement phase rather than being left until the closing stages when this would compromise the effectiveness of the test phase(s), increasing the risk of a poorly tested solution.

In addition, a Test Manager/Lead Test Analyst Checklist has been included to assist Test Managers and Lead Test Analysts with understanding the tasks that should be considered to ensure a successful test effort at DHCW.

The Testing Checklists are provided in Appendix 1 and Appendix 2.

Testing Concepts

There are some general testing concepts that are applicable to, and have been referred to in the development of, DHCW's Test Methodology. These include the following:

Agile Testing

Quality Assurance versus Testing

The V-Model of Testing

As they are an adjunct to the main purpose of this document, they have been placed in the Appendices for reference purposes. Please refer to Appendix 3 – General Testing Concepts.

Patient Safety

Validation Process

DHCW is required to validate the Computerised Systems and supporting Infrastructure that is provided for use by NHS Wales Services that operate within a regulatory framework and as such the DHCW Validation Policy has been produced to ensure that regulated Computerised Systems provided by DHCW are implemented, validated, supported and retired in a controlled manner relative to risk.

The DHCW approach to validation follows GAMP5, and can vary depending upon the scope, complexity and categorisation of the system

In determining the test approach to any DHCW product consideration need to be given whether there is a validation requirement and the test approach needs to reflect this.

Clinical Risk Management Process

The Clinical Risk Management Process in place at DHCW focuses on the safety testing of DHCW's products within the design, development and testing phases of an implementation. Within this context, it is DHCW's aim is to "ensure that all DHCW products and services are designed and manufactured in such a way that, when used under the conditions and for the purposes intended, they will not compromise:

- The clinical condition or the safety of a patients, or
- The safety and health of users of the products or
- Those persons who may indirectly be affected by the products or services

and that the process will ensure, where risk still exists, it is at an acceptable level when weighed against the benefits to the patient.

The focus of the DHCW clinical risk management process is the clinical safety assessment of its products from design through to development and final testing stages. It also provides a model for the NHS to apply when developing or procuring health information technology. [1].

As outlined in the risk management process document, one of the specific principles underpinning DHCW's Quality Assurance Process is:

- to establish whether clinical safety has been addressed, testing has been complete and service management arrangements are in place[2]

[1] Clinical Risk Management Process – January 2015
[2] SOP-Clinical Risk Management Process v2.0

One of the test approaches that the DHCW Test Framework is centred on is Risk Based Testing, to supplement this, the Clinical Risk Management Strategy's Risk scoring has been included as an appendix (see Appendix 5).

Patient Safety Alignment

Testing is a crucial aspect to minimising risk. Ensuring that testing is introduced early, occurs throughout the project lifecycle and that a solution is tested to its fullest extent reduces the possibility of errors within the implementation and therefore reduces the likelihood of subsequent clinical risks occurring.

DHCW's Clinical Risk Management Process has seven key stages that align with the DHCW project management methodology. As it is important for project testing activities to support this alignment, the activities outlined in the following section have been mapped against with each Clinical Risk Management and each Project stage. The following steps originate from the Clinical Risk Management Process and have been amended to include the corresponding test activities.

Link to Patient Safety Board / NADB	Product / Project Phase	Additional Patient Safety Activities	Testing Activity
Step 1 - Clinical Risk Management Plan **NADB Stage 1 – Strategic fit**	Scoping and initial design and development of use cases		▪ **Generate the Project Test Strategy including Risk Assessment** ▪ **Include testing in Project Risk Plan**
Step 2 – Clinical Risk Assessment	More detailed Design including the development of use cases	Develop the Risk Register iteratively and address mitigations to key risks	▪ **Complete Test Plans for each identified phase of testing** ▪ **Update Risk Assessment**
Step 3 - Clinical Risk Assessment Report **NADB Stage 2 – Approval of clinical and technical requirement**	Agreed clinical and technical requirement/specification	Fully populated Risk Register used to identify risks and mitigations	▪ **Generate Test Conditions/Cases/Scripts** ▪ **Update Risk Assessment**
Carry out Test Execution as identified in Project Test Strategy – Update Risk Assessment			
Step 4 - Clinical Safety Case Report	Decision to implement (Pilot)		▪ **Complete Test Summary Report including Final Defect Report**
Step 5 – PSAB approval for initial implementation	Authority to implement (Pilot)	Certificated as safe at this point in time	▪ **Test Team representation to discuss test recommendation based on test execution outcome** ▪ **Input - Test/Defect Summary Report** ▪ **Exit Criteria from testing met**
Step 6 – Post Pilot Review Report	Decision to implement (Full Rollout)		
Step 7 – Pre Release Stage Report **NADB Stage 3 - Authority to implement**	Authority to implement (Full Rollout) and handover to Service Management	Handover to Service Management	

Testing Approach

Risk Based Testing

Risk can be considered to be a function of two components: the probability that a defined adverse event will occur and the severity of the consequences of that occurrence. Testing is essentially the measures used in software delivery to reduce risks associated with the introduction of a new system or service. Risk Based Testing as a type of testing, seeks to identify the areas of a delivery where either failure is most likely or that are critical to the end user of the system and concentrates testing first and foremost on those.

To ensure a pragmatic approach to testing within DHCW, it is important to prioritize the features and functions to be tested based on importance and likelihood or impact of failure and focus resources (people, money, systems) on those areas.

Essentially the degree of acceptable risk for a function is directly related to how critical it is to the clinical end user. At DHCW, Test Cases are classified based on risk and critical tests are performed first followed by lower classified tests. This ensures testing and any subsequent fault reporting and resolution is focused initially on critical elements.

Risk Assessment and Scoring

Risk Assessment should form a part of the project Test Strategy phase. Each Test Strategy should include a section on system and testing related risks[3], their scoring and any mitigation activities required to offset the identified risks.

The following are examples of the types of questions that can be asked to try and identify risks within an implementation:

- Which functions and attributes are critical for the success of the product?
- What weaknesses or possible failures are there in a component or function?
- How visible is a problem in a function or attribute for clinical end users?
- How often is a function used?
- Who or what would be impacted by failures and how bad would that impact be?
- What input or situation could occur that might adversely affect or trigger a failure within a vulnerable system component?

Some of the tasks that can assist with assessing risks are

[3] Project related risks should be managed by the project manager within the project risk register.

1. Identify the kinds of problems a product could have. Talk to clinical users and technical resources utilising the questions listed above.
2. Identify which problems matter most. These are then allocated a high priority.
3. Outline how you would be able to detect the problems if they were apparent within the software.
4. Make a list of interesting problems and design tests specifically to reveal them.
5. Consult clinical and technical experts, design documentation, past bug reports.

To ensure a common risk scoring strategy across the DHCW programme, risks identified by any test or project team and subsequently specified in the test strategy should be scored based on the Risk Classification Matrix that forms part of the Clinical Risk Management Strategy. The Risk Classification Matrix, as shown below, provides a measure of safety risk by combining the risk severity and likelihood categories (definitions of the severity and likelihood categories are provided within Appendix 5). This then produces a risk score of Very Low, low, Medium or High. These risk scores should be applied to system and test related risks as identified within the project Test Strategy.

Requirements Based Testing

Requirements Based Testing is a testing approach in which test conditions, cases, scripts and data are derived from requirements defined by a project or programme. These can include functional tests and non-functional tests relating to attributes such as performance, security or usability. Requirements in turn are typically prioritised to define the order in which they should be considered for delivery.

Whilst there are various methods for prioritising requirements, the most commonly used method within DHCW projects is the MoSCoW method, where each requirement will be assigned a priority to indicate if it must be delivered to meet the business needs or whether it could be considered for delivery at a point in the future:

M – Must have this requirement to meet the business needs

S – Should have this requirement if possible, but project success does not rely on it

C – Could have this requirement if it does not affect anything else in the project

W – Would like to have this requirement later, but it won't be delivered at this time

DHCW's Test Methodology

Overview

Due to the disparate nature of projects within the DHCW programme, applying a single prescriptive test methodology to all projects is not possible. As every project differs in its requirements, risks, delivery method and solution design, testing needs to adapt in accordance.

This section of the document provides an introduction to DHCW's high level test process, the detail around how this may be executed in practice and an overview on how this should then applied to DHCW projects based on their delivery method. The methodology should be used as a guideline within DHCW and adapted according to each individual project's needs.

High Level Test Process

No matter the project's delivery type or development methodology, for example Scrum, successful testing progresses through a number of steps. Each of these can be as big or as small as required and often steps are revisited throughout the project's lifecycle as requirements or clinical priorities change.

TFS is used as the DHCW Test Management Tool to store test scripts, control the running of the test scripts and the logging of defects.

The following diagram presents a general view of the lifecycle of testing and how testing normally progresses within a software implementation project.

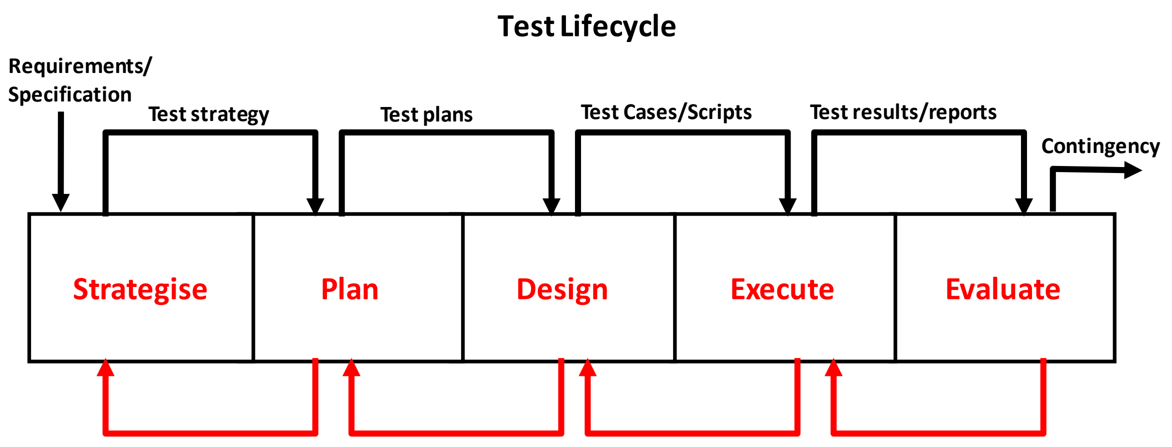

FIGURE 1 – TEST LIFECYCLE

The design of this diagram is not meant as a representation of any specific development methodology. The diagram is included to indicate that certain steps should be present to ensure successful testing, whether test execution appears as multiple phases at the end of developing the complete deliverable, or whether testing occurs within iterations such as the Agile methodology (more detail on this methodology is provided in Appendix 3 – General Testing Concepts), or whether testing only involves an Acceptance or User Acceptance period.

As highlighted by the Figure 2, successful testing is very often iterative with increasing levels of refinement as knowledge becomes available. Within the Agile Development model, the above processes reoccur throughout the development as sprint cycles represent mini-projects and Planning, Design and Execution can all occur simultaneously. It is advocated that, even when an Agile methodology is utilised, an overriding Test Strategy is completed close to the initiation of the project to ensure all project stakeholders fully understand and sign off how the testing process will unfold.

Test Deliverables

The testing deliverables of each DHCW project should include the following:

1. Test Strategy – developed in the Strategy phase
2. Test Plans – developed in the Planning phase
3. Test Conditions, Cases and Scripts – developed in the Design phase
4. Traceability Matrix – developed in the Design phase
5. Test Reports – delivered throughout the Execution phase and at the completion of testing
6. Test Certificate – this can replace a Test Plan and corresponding report on small, low complex test engagements.

For each project a Test Strategy and where applicable a requirements traceability matrix should be created or the information extracted from a Test Management tool. For each test phase such as System Integration Testing or User Acceptance Testing, a Test Plan should be produced. Manual Test Scripts should be developed for each section, module or distinct segment of the application under development.

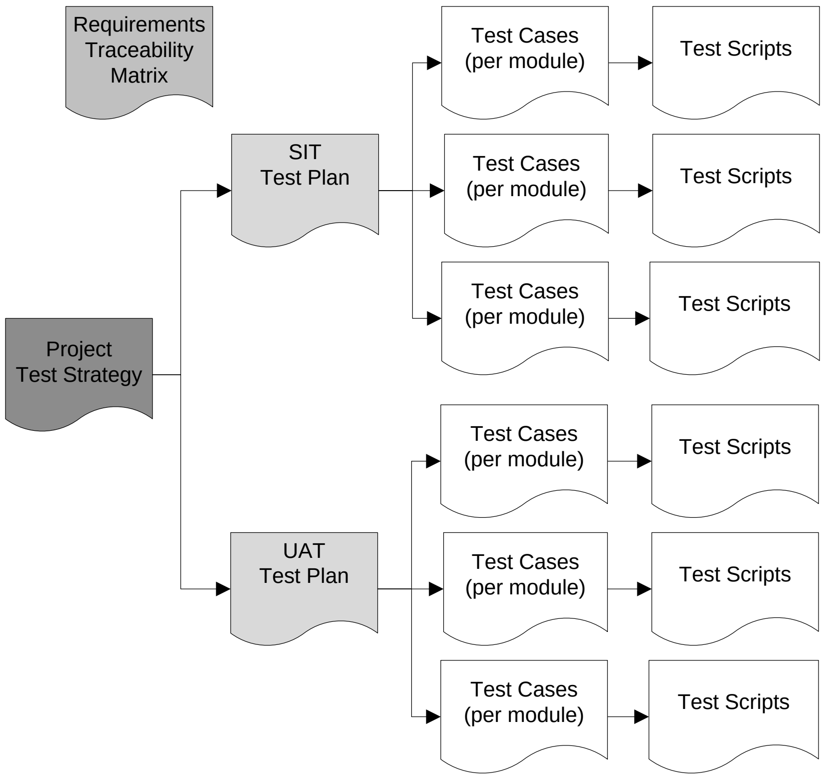

FIGURE 2 – TESTING DELIVERABLES

All test documents are intended to be used for a different purpose as outlined in Figure 3 below.

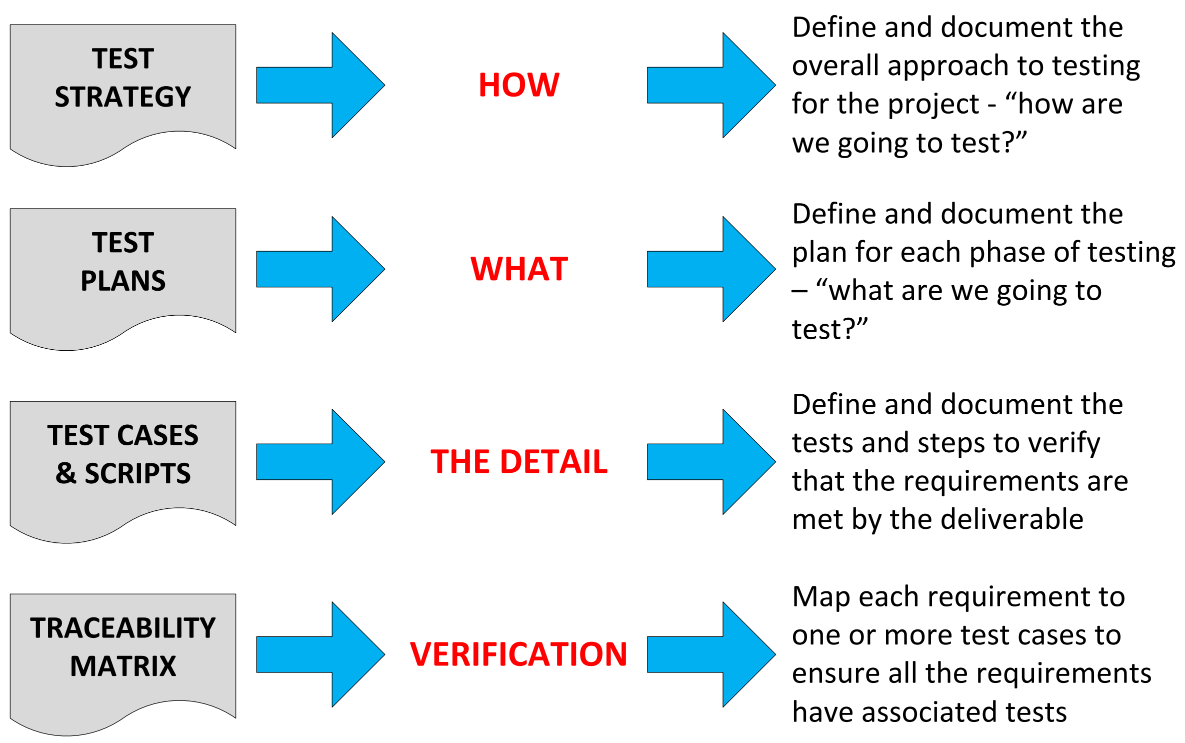
TEST STRATEGY
HOW
Define and document the overall approach to testing for the project - “how are we going to test?”
TEST PLANS
WHAT
Define and document the plan for each phase of testing – “what are we going to test?”
TEST CASES & SCRIPTS
THE DETAIL
Define and document the tests and steps to verify that the requirements are met by the deliverable
TRACEABILITY MATRIX
VERIFICATION
Map each requirement to one or more test cases to ensure all the requirements have associated tests

FIGURE 3 – TEST ARTEFACTS

Test Strategy

The Test Strategy involves defining and documenting the overall approach of the testing requirements for the project. A Test Strategy is a strategic plan providing the overall direction for testing with the main purposes being to:

- communicate the test approach to project stakeholders to gain their agreement
- convey the strategy to the test team to ensure they understand what is required and to enable a coordinated and consistent effort

Commencing as early as the Project Inception phase and continuing through each subsequent phase, the test strategy is readdressed continually as the project lifecycle evolves to ensure any new risks or changes are addressed by testing.

As one of the recommended test deliverables of every DHCW project, strategies provide the reader with a detailed understanding of the approach to the testing that will be performed and it's alignment with the DHCW Clinical Risk Management strategy. Essentially, the test strategy document should outline the following:

- the overall aim and scope of the testing
- the overall test approach
- the risks involved including the relevant scoring and mitigation activities
- any assumptions involved
- the test phases that are required
- how test cases will be classified
- at a high level, what test resources are required including people, tools, environments and data
- the test milestones and the deliverables from each phase
- defect/issue management
- release management

Test Planning

The planning phase of the test process is an essential component to ensure all project participants are in agreement regarding the tasks and responsibilities of the test team. Planning additionally allows the test team to provide milestones and time estimates for inclusion in the overall project plan.

As outlined in Figure 2 above, Test Plans should be completed for each unique test phase such as System Integration Testing (SIT) or User Acceptance Testing (UAT) (these test phases are detailed in the Test Phases section below).

Where an Agile approach is being followed the Test Plan should be written and the release level.

Phase specific test plans provide the reader with a detailed understanding of what will be tested during each test phase and provides more detail on the scope and risks for that phase. Due to the dissimilarities in testing priorities, a SIT plan will differ to a UAT plan (for example). Different risks will be evident, different environments may be used, different resources will be required, different test data needs will be evident. Therefore a test plan is completed for each test phase to ensure project stakeholders know what will and will not be tested during that period.

Essentially, the test plan should include the following:

- the aim and scope of the test phase
- the risks and assumptions involved for that test phase
- the contingencies in place
- the types of testing that will be undertaken during that test phase
- test coverage during that phase outlining what areas of the system will be tested
- the tasks and responsibilities of all parties
- the entry and exit criteria of the test phase
- the test environment and tools required to conduct the testing during the test phase
- the test data required for that phase

- the test resources required for that phase
- the test schedule for that phase
- what communication will be undertaken during the test phase in the form of test summary reports, defect reports and so on

Test Design

Test Design involves the nitty-gritty of testing. This phase takes the higher level test plan and breaks it up and refines it into specific features and testable units. A number of deliverables are produced during the test design phase:

1. Requirements Traceability Matrix
2. Test Conditions, Cases and Scripts

Test Conditions, Cases and Scripts

Test Conditions are items or events of a component or system that could be verified by one or more test cases, e.g. a function, transaction, feature, quality attribute or structural element.

Test Cases are a set of input values, execution preconditions, expected results and execution post conditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement.

Test Scripts are the step-by-step maps that test analysts should follow to verify each high level test case. These plans are very detailed and should provide the steps to follow, the expected results of each step and any data required to verify a test case. Test scripts can exist for manual tests and also automated tests.

Quite often a Test Case and Script are combined.

Requirements Traceability Matrix

A Requirements Traceability Matrix should be developed for each project to provide the following functions:

- enabling each test case to be mapped backwards to a functional requirement or use case
- providing stakeholders with the confidence that all signed off requirements have been tested
- identifying which test cases verify which requirements
- identifying in what phase the requirements will be tested

Traceability matrices can be complex or straightforward but their main purpose is to provide clear evidence that all documented requirements are covered by the expected tests.

The following table provides an example of a basic matrix, you can see that all tests associated with requirement R1.0 have passed testing, therefore the requirement can be deemed to be satisfied:

Requirements	Priority	Test Case	Test Script	Test Status
R1.0	High	TC1.0	1.1	Pass
			1.2	Pass
			1.3	Pass
		TC2.0	2.1	Pass
			2.2	Pass
R2.0	Low	TC3.0	3.1	Not Run
			3.2	Not Run
			3.3	Not Run
R3.0	Medium	TC4.0	4.1	Not Run
			4.2	Not Run
		TC5.0	5.1	Not Run
		TC6.0	6.1	Not Run
			6.2	Not Run
			6.3	Not Run

FIGURE 4 – TRACEABILITY MATRIX

Test Execution

Test Execution is a stage of the testing process and involves the running of tests that have been designed during the Test Design phase. Successful test execution is accomplished when the results of the test are known, whether it is a result that is expected or not.

Test Reporting

The objectives of testing as a formal project activity include:

a) utilising a series of methods and techniques throughout the life cycle of the deliverable to verify and validate that the application adheres to the requirements specified at the commencement of the project
b) provide effective information about the status of the development to ensure decisions can be made in an informed manner

The second objective involves the test team providing information that is relevant, practical and enables stakeholders to make decisions based on the known state of the deliverable. 'Reporting' therefore, is a key activity of any test team.

The number and structure of test related reports should be documented within each project's Test Strategy. However, it is recommended, that during the lifecycle of the implementation, the following reports be considered:

- **Test Progress Report** – A report on the progress of testing including how many test cases exist and their status, how many defect reports have been filed, their status and so on. The frequency of this report should be specified within the project test strategy. For example, if utilising an Agile Methodology, this report may be sent out daily during the test phase of each Sprint session.
- **Test Defect Report** – A report on the defects identified, prioritised by criticality. This report should be easily obtained via the Defect Management tool. The frequency of this report should be specified within the project test strategy. For example, if utilising an Agile Methodology, this report may be sent out daily during the test phase of each Sprint session. There is no reason why the Test Progress Report and the Test Defect Report cannot be combined if required.
- **Test Phase Summary** - A summary report generated at the end of each test phase (i.e. SIT, OAT or UAT). These summaries should include a report at a phase level on tests executed, tests passed, tests failed, tests incomplete, outstanding defects.
- **Test Summary Report** - A summary report generated at the end of a project. These summaries will include a report at on tests executed, tests passed, tests failed and incomplete tests as well as highlight any outstanding defects.
- **Test Certificate** – A 'light touch' summary report generated at the end of a test phase. These certificates should be considered for projects that are of minimal duration and will include details on tests execution status, outstanding defects and a recommendation for the project.

Entry and Exit Criteria

Each project phase should specify entry criteria (the criteria for starting that phase) and exit criteria (the criteria for completing that phase). These criteria are sometimes called quality gates. The reason for entry and exit criteria is to clarify to the project how and when handover can take place and when it should not take place. In a project with several parallel activities a set of exit criteria may feed more than one set of entry criteria.

Entry and Exit Criteria need to be part of the project's initial Test Strategy and should be approved by the stakeholders of each phase. That is, the exit criteria from the unit testing phase should be defined in collaboration with the development team. Entry criteria for the UAT phase should be defined in collaboration with the users who will be accepting the deliverable.

As Entry and Exit requirements are project specific, a generic set of criteria cannot be defined within this document. However, the following table has been provided as an example of how and what sort of criteria can be put in place to ensure a smoother transition from one project phase to the next.

Phase	Entry Criteria	Exit Criteria
Development Testing	▪ Design solution documents signed-off	

Phase	Entry Criteria	Exit Criteria
	▪ Technical Specification completed and approved ▪ Development test plan/checklist has been prepared and viewed	▪ Development test plan/checklist has been completed and signed-off ▪ Peer Review completed ▪ No Critical or High defects outstanding. ▪ All open defects have an owner and resolution timeframe agreed.
SIT Testing	▪ All Development Testing Exit Criteria met ▪ SIT Test Plan completed, reviewed and signed off ▪ SIT Test Cases completed and reviewed ▪ Applications and hardware installed ▪ Test Data anonymised and loaded ▪ Any required tester training completed ▪ Test Environment is ready to use and is as representative of the live environment as feasible ▪ All required resources have access to the applicable defect management tool ▪ Support process in place	▪ All Mandatory and Expected Test Cases executed ▪ Majority of Desired test cases have been executed ▪ All Critical and High issues resolved ▪ All Medium and Low issues accepted and signed-off ▪ Test Phase Summary completed and signed off
Operational Acceptance Testing	▪ All SIT Exit Criteria met ▪ Operational resources available	▪ All Mandatory and Expected Test Cases executed ▪ Majority of Desired test cases have been executed ▪ All Critical and High issues resolved ▪ All Medium and Low issues accepted and signed-off ▪ Test Phase Summary completed and signed off
User Acceptance Testing	▪ All Technical Acceptance Testing Exit Criteria met if applicable ▪ All System Testing Exit Criteria met if applicable ▪ UAT Test Cases completed ▪ Any required User Training	▪ All Mandatory and Expected test cases have been ▪ Majority of Desired test cases have been executed ▪ All Critical and High issues resolved

Phase	Entry Criteria	Exit Criteria
	completed ▪ Clinical Users assigned and available ▪ Test data is anonymised and loaded ▪ UAT Environment is set up and ready to use ▪ Support process in place	▪ All Medium and Low issues accepted and signed-off ▪ Test Phase Summary report completed and signed off. ▪ Project Test Summary report signed off

Test Evaluation

Test Results

As confirmation that test cases have been executed and that the system under test has passed each test step, an audit trail should be retained including the completed execution plans and screen dumps. Screen dumps need not be taken for every screen of a deliverable (as the time involved outweighs the perceived benefit) but should capture vital data and functional areas. These areas should be decided upon prior to test execution and confirmed in the Test Plan document.

Project Evaluation

Post project evaluation sessions should include an item on testing and how the following aspects of testing supported or did not support the overall objective of the project:

- Test Planning – was this relevant, accurate, useful
- Test Resources – was resource estimation accurate
- Time estimation – was time estimation accurate
- Defect tracking and management – were issues managed effectively
- Testing Deliverables – was there too much/too little documentation
- Test Execution – was the overall execution of tests successful
- Alignment with requirements – were there requirements and did testing verify they had been met
- Any areas where testing actively supported the successful outcome
- Any areas the project felt could be improved

Test Phases

An implementation typically passes through a number of test phases from the commencement of development through to the live deployment. This occurs no matter the project delivery method, be it Agile or Waterfall. However the responsibilities of the test phases will be affected depending on whether the deliverable is developed in-house, by a third party or procured as an off-the-shelf product. This concept is described in the Test Phases per Delivery Method section below.

For a successful implementation of a software product, the following diagram outlines at a high level the test phases a product will pass through. These phases are then described in more detail below. Some of these may occur in parallel, or parts of the testing may occur prior to upstream test phases. For example, during an in-house development project, Operational Acceptance Testing may occur in parallel with System Integration Testing.

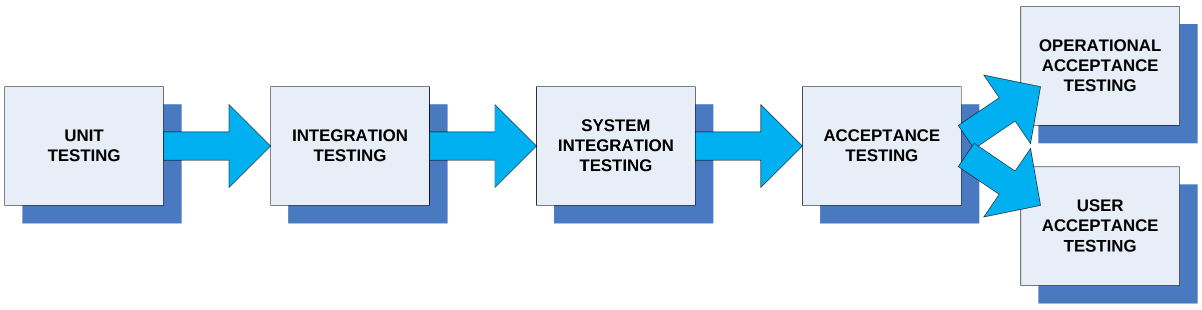

FIGURE 5 – TEST PHASES

Development Testing

Unit Testing

Unit testing is a process by which the smallest testable parts of a system, called units or components, are individually and independently validated to ensure they are working correctly before the units are combined into a working programme. Unit Testing is also known as Component Testing.

Unit testing of an application by developers prior to release to testers for system testing is an important component of the testing process. In some circumstances, it is even advisable for unit tests to be written prior to coding to enable developers to code according to required outcomes. Unit testing should include an integration test where components due for delivery to the test team are integrated and tested to ensure they compile and work together to provide a stable release. Unit Testing should be performed by technical staff within the development team.

Integration Testing

Integration Testing is performed on integrated components or units that have passed Unit Testing. Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system and is ready for System Integration Testing. Integration Testing should be performed by technical staff within the development team.

System Integration Testing

System Integration Testing (SIT) analyses an application based on the requirements specified in the functional specification and user requirements document. Specifically, system testing should be applied to an application to ensure:

- The development adheres to requirements
- The development meets the quality measures (if any) specified at the commencement of the project
- The development is consistent and accurate
- Defects are revealed and resolved before release to the client

SIT comprises of two fundamental test stages namely Sanity Testing and Structure Testing

Sanity Testing

Sanity testing a development build can also be referred to as Build Verification Testing or Build Acceptance Testing. For the purpose of the DHCW Framework, these terms have been rolled into the one expression "Sanity Testing".

Sanity testing is an *unstructured* test of the initial development. It involves a simple walk through of the system to check that the main functionality is performing as specified in the specification documentation. Essentially, Sanity Testing determines whether it is reasonable to proceed with further testing – if the sanity test fails, it is not practical to attempt more rigorous testing.

This type of testing is essential to get the application to a state that is suitable for the commencement of structured testing. Specifically, Sanity testing has the following objectives:

- Catch obvious bugs that hinder the basic operation of the system
- Get the application to a stable state which will support Structured Testing
- Enable the test analyst to become conversant with the system facilitating the generation of test scripts before embarking on the more formal Structure Testing
- Provide a window to make last minute enhancement changes to the application

Structured Testing

Structured Testing is the execution of pre-prepared and well-defined test execution plans. Structured Testing is designed to analyse every area of the application under the greatest range of conditions in an attempt to simulate every situation that may cause the system to fail. Specifically Structured Testing has the following objectives.

- Provision of a documented test analysis structure, which includes clearly defined start points, execution paths and termination points for each test component undertaken.
- Detailed and exhaustive testing of every function and process within the application.
- Detailed and exhaustive testing of the application's interaction with the external technical environment (if applicable)
- A clearly visible audit trail of the analysis undertaken, updates executed and termination of the analysis process.

Structured Testing can be broken down into a number of test types with formal Test Cases and Execution Plans (as detailed above) documenting the tests that will be utilised in the analysis of the deliverable. Test types performed during Structured Testing can include:

1. Functional GUI Testing
2. Functional Data Testing
3. Interface Testing
4. Application Interaction
5. Configuration Testing
6. Non-Functional Testing
7. Regression Testing

For a detailed explanation of these test types, please see **APPENDIX 4** – Test Types
.

System Integration Testing should be undertaken by professional Test Analysts who have knowledge of the product and testing concepts.

Acceptance Testing

Acceptance testing can take many forms whether it is Contract Acceptance, User Acceptance or Operational Acceptance. Essentially this testing takes a pre-defined set of conditions (specified within either a contract or a user requirements document) and tests the deliverables against those criteria.

Contract Acceptance Testing

When procuring an off-the-shelf or bespoke product, it is important to verify that what was requested by the supplier was delivered. This testing can be as simple or as complex as is required to ensure all features, functions, specifications and components have been supplied.

Contract Acceptance Testing should be undertaken by project resources familiar with the contractual requirements, to enable them to verify the deliverable.

Operational Acceptance Testing

Operational Acceptance Testing (OAT) ensures the deliverable will be reliable and maintainable within an operational environment. These tests should also confirm that the infrastructure, components and applications operate in the technical environment as expected according to specified requirements. OAT can be broken down into a number of test types namely:

1. Infrastructure
2. Connectivity
3. Deployment
4. Storage
5. Disaster Recovery
6. Failover
7. Backup/Restore
8. Performance Monitoring tools
9. Security
10. Soak Testing

For a detailed explanation of these test types, please see **APPENDIX 4** – Test Types

Operational Acceptance Testing should be undertaken by technical and support resources.

User Acceptance Testing

Involving users throughout the life-cycle of a project ensures that surprises at the end of the project, when issues are more costly to resolve, are less likely to eventuate. In addition, a formal User Acceptance Testing (UAT) period, where tests are designed by users, performed by users and signed off by users is essential to ensure the application functions as they expect based on the pre-specified user requirements.

UAT should include tests that simulate the daily activities of users and should be performed in an environment that replicates as much as possible, the environment in which the user will be utilising the deliverable.

User Acceptance Testing should be performed by the end users who will be utilising the system in their day-to-day working environment.

Primary Care Supplier Assurance

Primary Care Supplier Assurance ensures that General Practice System Supplier Releases do not have a negative impact on the Welsh specific functionality that is contained within the supplier software releases.

There are two system suppliers for GP clinical applications; the scope for supplier assurance is further defined in the respective assurance strategies

Mobile Assurance

The term Mobile can be interpreted in several ways. Is it assurance of a product accessed via a mobile device or is it the assurance of an application specifically designed for mobile devices. The assurance process is the same as described in earlier sections but with considerations for the different operating systems, different versions of these operating systems as well as the plethora of physical devices and whether emulators of these devices should be used instead of the physical devices.

Project Delivery Methods

Within the NHS Wales Informatics Programme based on the nature of the implementation, there are three main methods or ways in which projects are executed.

In-house Development – involves the internal DHCW development team planning, analysing, developing and testing a software application. Development of an in-house application generally follows the software development lifecycle with DHCW being responsible for all stages of testing from Unit through to Operational Acceptance.

Bespoke Purchase – where a third party develops a customised application based on DHCW's specific requirements. Development of a bespoke application follows the vendor's methodology with the vendor being responsible for unit and system testing. DHCW would then be responsible for System Integration with other DHCW support national products, Operational and User Acceptance testing. The responsibility for each phase of testing should be defined in each project's test strategy as well as in any contract. In addition, DHCW's expectations on the vendor with regards to testing and the verification of tests undertaken, results and unresolved defects should be specified within the project test strategy. This should then be circulated to the vendor.

Off-the-shelf Procurement – where DHCW has purchased a "ready-made" software application which needs integrating within the DHCW set of systems. Off-the-shelf vendors are responsible for their own unit and system testing methodologies with DHCW being responsible for Integration and Operational and User Acceptance testing.

As is highlighted above, the testing responsibilities of DHCW and each third party involved differ according to the software delivery type. In addition, as these methods dictate the phases and types of testing undertaken by DHCW, the Test Phases per Delivery Method section below addresses the testing that is applicable to each delivery type.

Test Phases per Delivery Method

As part of each project's individual test strategy, the specific delivery method should be identified and the phases and types of testing to be employed by DHCW throughout the project defined. It is acknowledged that there will be occasions when projects involve a combination of the abovementioned delivery methods.

The table below provides an outline of how DHCW test phases and test types may be utilised by the different delivery methods.

Delivery Method	Test Phases[4]	Test Types[5]
In-house	Unit Testing	
	Integration Testing	
	System Integration Testing	Functional GUI Testing Functional Data Testing Interface Testing Application Interaction Configuration Testing Non-Functional Testing Regression Testing
	Operational Acceptance Testing	Infrastructure Connectivity Deployment Storage Disaster Recovery Failover Testing Backup/Restore Performance Monitoring Security Soak
	User Acceptance Testing	
Bespoke application by third party	Contract Acceptance Testing	
	System Integration Testing	Interface Testing Application Interaction Configuration Testing Non-Functional Testing Regression Testing
	Operational Acceptance Testing	Infrastructure Connectivity Deployment Storage Disaster Recovery Failover Backup/Restore Performance Monitoring Security Soak
	User Acceptance Testing	
Off-the-shelf procurement	Contract Acceptance Testing	

[4] See the Test Phases section for definitions

	System Integration Testing	Interface Testing Application Interaction Configuration Testing Non-Functional Testing
	Operational Acceptance Testing	Infrastructure Connectivity Deployment Storage Disaster Recovery Failover Testing Backup/Restore Performance Monitoring Security Soak
	User Acceptance Testing	

Vendor Testing

Vendors are a vital aspect of DHCW's test lifecycle. Any project that utilises third parties (as part of a bespoke development or system procurement) will need to ensure each party is aware of DHCW's expectations in terms of testing. The individual project's test strategy should outline the roles and responsibilities of each party and how the vendor's test plans will be evaluated in addition to testing related aspects of the Contract between DHCW and the vendor.

[5] See Appendix 4 – Test Types for definitions

Control Procedures

Test Metrics

Communication to stakeholders is important for any successful testing effort. Metrics, or “standards of measurement”, are used to gauge the effectiveness and efficiency of a particular activity within a project and generally have the following purposes:

- To provide information to project members on the progress of testing
- To provide information to project stakeholders on the state of the implementation
- To provide information for managers on the effectiveness of the project’s process
- To provide information for the development team on defect incidence
- To enable the test team to estimate future testing more accurately

Essentially, metrics should be used to improve project practices – either immediately or in future projects. They can be as detailed or as simple as is necessary to be able to provide objective feedback to the Project Team regarding any of the development processes. In determining what metrics to collect, the following aspects should be considered:

- They should be quantifiable
- They should be easy to collect
- They should be simple but meaningful
- They should be non-threatening

At a very high level, metrics aim to provide information such as:

- Number of overall test cases
- Number of test cases completed
- Number of test cases passed or failed
- Number of test cases re-executed
- Number of outstanding defects
- Percentage of defects resolved
- And so on

The metrics to be used within each project will be defined within each project’s overall Test Strategy.

Defect Management

The effective management of defects is vital to the progression and success of any testing effort. Any defect management tool should be easy to access, easy to use, reliable, secure and provide sufficient reporting tools to enable efficient and concise communication around the status of the system under test.

Defect Identification

Due to the complex and unpredictable nature of software development, errors (defects, issues, faults, bugs) are introduced during the development process. As testing progresses, issues are identified and are considered to be any part or function of the application that deviates from the end user's expectation or the specification. However, there will also be times when issues are identified that have not been explicitly stated in the requirements documentation but rather, are implicitly assumed to be a requirement. An example of this is a GUI spelling mistake. As it is implicit that the application will meet "common" quality criteria such as fundamental Windows standards, these may not be explicitly stated in the requirements documentation.

Once a deviation from expected behaviour or look and feel has been identified, the issue will be logged in the defect management system.

Defect Logging and Tracking

Where possible, each project should utilise Visual Studio Team Foundation Server to log, track and report on defects. Visual Studio Team Foundation Server provides access to bug information in a standardized format and can also be used to record new development items as user stories.

The Visual Studio Team Foundation Server system:

- Provides project teams with a permanent record of what is known about their software projects.
- Allows tracking of the status of bugs and provides notification when that status changes.
- Gives the user quick access to open bugs assigned to them.
- Provides an essential tool for Project Management, Development, Testers and Technical Support to collaborate in prioritizing bugs and in deciding which bugs to fix for a particular release.
- Provides an audit trail of all changes to a bug
- Provides test managers and testers with the ability to report on the status of the project as it relates to the number of open and resolved defects

Defect Progression

Generally, the logging and resolving of defects should follow the below process:

- Defect Originator discovers defect and enters details into the defect tracking tool
- Defect Originator assigns the defect to appropriate project resource for resolution (Defect Owner).
- Defect Owner resolves defect, retests the defect to ensure the defect has been resolved then updates the defect tracking tool with details of the resolution.
- Defect Owner re-assigns defect to the Defect Originator and changes the status to Resolved.
- Defect Originator verifies the defect has been resolved (through re-testing) and Closes the defect.
- If the Defect Owner is not able to resolve the issue they must either:
    - Re-assign the issue to someone who can (this person now becomes the Defect Owner) or;
    - Assign it back to the Defect Originator after adding a comment as to why it cannot be resolved.
- If the Defect Originator does not agree that the issue has been resolved they should speak directly with the party who marked the defect as resolved to clarify resolution requirements. The defect can then be updated and reassigned as necessary.
- Any delays or difficulties in getting resolution must be escalated to the Project Manager.

Severity/Priority

Each defect logged should be assigned a Severity and a Priority rating. The Severity rating depends on the effect the issue would have to the clinical end user. The Priority rating depends on the effect the issue has to the testing effort. Defect ratings are normally defined within a project's test strategy. However, to provide commonality across DHCW projects, it is recommended that the following ratings be utilised, these are consistent with the ratings used in Visual Studio Team Foundation Server.

Severity

Severity	Severity Definition
1 - Critical	The issue relates to a crucial part of the system. Clinical users would be critically impacted if the defect was released into production. No workaround exists.
2 - High	The issue relates to a crucial part of the system. Clinical users would be highly impacted if the defect was released into production. A workaround does exist.
3 - Medium	The issue relates to an essential part of the system. However, clinical users would not be severely impacted if the defect was released into production as there is a workaround.
4 - Low	The issue relates to a non-essential function or a function is working in a restricted manner, or there is a minor problem with the application. Effect on the end user would be minimal.

Priority

Priority	Priority Definition
1	The system cannot be tested until this issue is resolved
2	The defect has a considerable impact on the progress of testing overall
3	The defect has an impact on the progress of testing a particular component
4	No impact to the progress of testing

Test Resources

Each individual DHCW project's Test Strategy should specify the test resources required at a project level, as this will depend on the project delivery type, test phases identified and amount of testing required. This section of the framework provides an indication of the possible roles, environment utilisation and data requirements that may be needed during the test phase of an DHCW project.

People

The following table outlines the possible roles that may be utilised during an DHCW project. This is not intended as a prescriptive list but rather, provides an indication of the test resources that may be required and their corresponding responsibilities. Actual roles and responsibilities should be specified within each project's test strategy.

Role	Responsibility
Programme Test Manager	▪ Manage DHCW's testing department ▪ Train and mentor test team members ▪ Champion DHCW's quality and test standards and procedures ▪ Create and manage programme schedule for testing ▪ Allocate test resources to projects if and when required ▪ Identify and escalate major issues to senior management ▪ Oversee the procurement of test tools ▪ Review weekly Testers' status reports ▪ Work closely and co-ordinate work with other work streams ▪ Attend checkpoint meetings to give testing status to board ▪ Provide technical support to testing team
Lead Test Analyst	▪ Document and implement the project test strategy ▪ Create phase specific test plans ▪ Assist with the creation and execution of test scripts ▪ Manage the project's test environment ▪ If required, manage Test Analysts and Testers in relation to their roles within the project ▪ Provide regular status reports to the Project Manager and Programme Test Manager ▪ Manage the Defect lifecycle
Test Analyst	▪ Create Test Scripts ▪ Execute Test Scripts ▪ Log and retest defects as required ▪ Provide regular status reports to the Lead Test Analyst
Tester	▪ Execute Test Scripts ▪ Log and retest defects as required
Automation Specialist	▪ Provide expertise on testing automation tool

	▪ Define and implement the project's automation strategy ▪ Create and execute automation test scripts ▪ Manage the automation tools and scripts and update as required ▪ Provide meaningful test reports based on the tools results ▪ Train and mentor other test analysts if required
UAT Testers	▪ Assist with the creation of UAT test plans ▪ Create UAT test cases ▪ Assist with the creation of UAT test scripts ▪ Execute UAT test scripts ▪ Provide feedback to projects on the product to be implemented ▪ Sign-off UAT test plans, scripts and reports ▪ Log and retest defects as required

Test Environments

Environment Requirements

The need for test systems that emulate, as much as possible, the systems on which end users will utilise the application on a day to day basis (the 'live environment') is essential. Hardware and software versions, configurations and architecture have a considerable effect on applications. Therefore, if applications are tested within an environment that differs from that on which it will finally be deployed, a substantial risk exists that the application will fail when installed into the live environment.

In addition, dedicated, stable test environments are essential to the successful outcome of the testing phase of a project. "Dedicated" ensures that the environment changes only when expected by testers reducing the logging of false defects. "Dedicated" also ensures that test data remains clean and usable and once again, only changes when expected. Stability is also vital in that any unexpected environment downtime may severely impact testing times.

Environment requirements will be gleaned from project technical resources, architecture diagrams and analysis of the production environment. As each project will differ in its environment requirements based on the underlying architecture, it will be each project's responsibility to specify the test systems required and ensure the environment is set up as needed for testing. The following tasks are recommended to ensure test systems are ready and usable for testing:

1. Identify environment requirements based on architecture diagrams, user requirement documentation, project technical resource knowledge and discussions with Informatics
2. Specify environment requirements within the project Test Strategy

Environment Management

All factors that may affect the outcome of testing should be under the control of the Lead Test Analyst for the designated test period. This includes (but is not limited to) test applications, servers, databases and any other components utilised solely for test purposes. Any changes to be introduced into the test environment (whether a minor or major release[6]) should be undertaken as a formal release with release notes provided to the Lead Test Analyst prior to the deployment. Release notes should contain:

- Code Drop or Deployment Number
- Any new functionality included in the drop if applicable
- Any known issues identified and fixed through unit testing
- Any Defect Fixes - including the defect number - defect description and resolution
- The likely date and time the changes will be released into the test environment.

Change Control

An Environment Change Control Register should be managed throughout the project by the Lead Test Analyst. With each release, the Lead Test Analyst should record the release number and details in the Environment Change Control Register and check there are no conflicts between releases to be introduced. Any database refreshes should be scheduled so as not to conflict with any application code changes whose testing and/or functionality relies on a particular schema or dataset. And it must be ensured that it is possible to roll back from any change implemented.

Test Data

A coherent set of test data that suitably mimics the scenarios under test is vital for a successful test phase. Test data that accurately reflects real world scenarios ensures that tests are undertaken in an environment that replicates the live systems and information utilised by clinical users.

It is felt that with the complexity of NHS patient information, attempting to reproduce scenarios by manually entering test data prior to testing would be time consuming and virtually unfeasible. Therefore, it is recommended that each project arrange for the utilisation, and subsequent anonymisation, of live NHS patient data to better mimic the complicated scenarios that occur in the live environment.

Due to the nature of the live data, test scenarios may be infinite as each patient differs from the next. However, at a minimum, test data should capture the most complex scenarios to ensure these are managed accurately by the system under test. To locate suitable data, it is advised that project's obtain technical support from DHCW's technical teams.

[6] In principle, with software releases, the major number is increased when there are significant jumps in functionality, the minor number is increased only when minor features or fixes have been added

Test Tools

The more complex a project's testing requirements, the greater the need for tools to ensure testing can be properly and effectively managed. The main types of tools include:

- Test Management tools – these tools assist with the recording, storing and execution of test scripts as well as the traceability back to the requirements. DHCW uses Microsoft TFS as its Test Management Tool including defect management.
- Test Automation tools – these tools assist with test execution in replacing manual actions performed by human testers with application assisted actions. Automation tools also include Load and Stress tools that mimic multiple, simultaneous usage on the application. They can also be used to imitate peak utilisation to 'stress' the application to its uttermost capacity.
- Defect Management tools – these tools assist with the logging and management of defects detected throughout the project.

Release Procedures

Release procedures are important for establishing the acceptable criteria with which an implementation version is accepted into the test or production environments. Release procedures may differ between projects so these should also be specified within each project's individual test strategy. However, the below sections provide high level recommendations on the procedures that could be adopted when testing and releasing a deliverable.

To Test

All activities that may affect the outcome of testing should be under the control of the project test manager or lead for the designated test period. Any changes to be introduced into the test environment (whether a minor or major release) should be undertaken as a formal release with release notes provided to the Test Manager prior to the deployment. Release notes should contain:

- Code Drop or Deployment Number
- Any new functionality included in the drop if applicable
- Any known issues identified and fixed through unit testing
- Any Defect Fixes - including the defect number - defect description and resolution
- The likely date and time the changes will be released into the test environment.

With each release, the Project Test Manager/Lead should record the release number and details in the Environment Change Control Register and check there are no conflicts between releases to be introduced. Any database refreshes should be scheduled so as not to conflict with any application code changes the testing and/or functionality of which relies on a particular schema or dataset. It must be ensured that it is possible to roll back from any change implemented.

To Production

Strategy

The testing strategy to ensure that each phase of an DHCW project is in a fit state to handover to any NHS Trust operations or clinical teams should be based on:

- Sign off of test results
- Test Summary Report indicates that the final test phase exit criteria, as defined within the project's test strategy, has been met
- Test Summary Report signed off by all stakeholders

Documentation

Test cases

It is important that any knowledge gained by an DHCW Project be maintained within DHCW for use in future DHCW related projects. Test documentation such as the Project Test Strategy, SIT, OAT and UAT Test plans and all test cases should be handed over to the appropriate project teams at the completion of the phases. These documents should be stored and maintained for future use by the relevant team.

Test Summary Report

The Test Summary Report should contain an overview of the results from testing including how many test cases for each test phase were run from the total number of cases, those test cases that were not run, how many test cases passed and how many failed and details of any outstanding defects.

This report should be circulated to all stakeholders so that the relevant Operational and Clinical Teams can initiate any workarounds if needed and are aware of any risks to the solution being implemented into production. This may include updating any Help, Process, Training or Support documentation depending on the nature of any known changes or issues.

Sign-off

Test documentation should be signed off as defined within the relevant project Test Strategy.

Conclusion

This document has established the Test Framework and Methodology for Digtal Health and Care Wales.

DHCW is dedicated to implementing "products and services that meet the needs of patients and clinicians and provide measurable benefits for the people of Wales" and ensuring the "design development and *testing* of all DHCW software products achieve the maximum level of patient safety and that they adhere to robust clinical risk and safety assessment processes" [7].

This test framework supports that aim by

- Advocating a risk based testing approach
- Aligning the testing methodology to the patient safety Clinical Risk Management Process
- Providing a guide to project and test managers on the testing methods, processes and standards that should be utilised throughout each project
- Ensuring test strategies and test planning conform to DHCW's overall objective
- Ensuring that testing and test documentation is consistent across the programme
- Promoting the need for effective and efficient communication so that stakeholders are informed of the state of any implementation allowing them to make accurate decisions

This methodology should be utilised for the test phases of every DHCW deliverable and be referred to as a basis for generating project test strategies and plans.

[7] Clinical Risk Management Process – January 2015

Appendix 1 – Project Manager Testing Checklist

1.	Have you read the DHCW Test Framework?	☐
2.	Has your project's delivery type been defined? See Page 31 of the DHCW Test Framework	☐
3.	Have the high level test phases been identified based on the delivery type? See Page 32 of the DHCW Framework.	☐
4.	Has each test phase been included within the project plan and have the related testing costs been provided within the project's cost breakdown?	☐
5.	Have the project's test resource requirements, such as environment and data, been defined based on the test phases and time estimation? See Page 39 of the DHCW Test Framework	☐
6.	Have the type of people resources the project requires been considered – Test Manager, Lead Test Analyst, Test Analysts, Testers?	☐
7.	Will the project budget need to cater for added resource costs to cover additional head count?	☐
8.	Has it been agreed who will write the test strategy/plan? Has it been agreed who will execute the testing?	☐
9.	Has the Service Management team been engaged with to ensure they understand the project's test requirements?	☐
10.	Has it been established whether User Acceptance (UAT) test resources are required?	☐
11.	Has it been established where the UAT resources should be drawn from? Has this been discussed with the relevant departments and have they been advised of their responsibilities?	☐
12.	Have you added testing resource requirements into the Resource Management system?	☐
13.	If required, has a booking been made at the Health Informatics Laboratory unit based on the project's estimated testing dates?	☐

14. Will the project need to fund a test environment or added licences for test systems? ☐

15. How will defects and issues be managed during the project? See page 36 of the DHCW Test Framework ☐

Appendix 2 – Test Manager/Lead Test Analyst Checklist

1.	Have you read the DHCW Test Framework?	☐
2.	Have you located the DHCW Test Framework templates?	☐
3.	Have you confirmed the test documentation that will be required by the project?	☐
4.	Have you identified and engaged with the relevant technical, clinical and Service Management teams?	☐
5.	Have you identified what systems/infrastructure/peripherals you will require for testing?	☐
6.	Have you completed a project Test Strategy?	☐
7.	Have you determined how defects and issues will be logged and managed throughout the life of the project?	☐
8.	Do you require any extra testing resources?	☐
9.	Have you engaged with clinical teams around their User Acceptance Testing expectations?	☐
10.	Have you received the requirements document and technical specifications?	☐
11.	Have you undertaken a risk assessment on the risks associated with testing?	☐
12.	Have you had the project Test Strategy reviewed by the Project and Service Management teams?	☐

Appendix 3 – General Testing Concepts

QUALITY ASSURANCE VS TESTING

Software Quality Assurance (SQA) and Testing are considered two distinct paradigms. SQA is regarded as those practices that are aimed at “prevention”. That is, Quality Assurance procedures, checkpoints and measures are employed to ensure problems are found and dealt with early on in the project’s life cycle. SQA is also employed to monitor and improve the System Development Life Cycle (SDLC) process, making sure that any agreed-upon standards and procedures are followed.

Testing however, is viewed as a series of methods and techniques utilised throughout the life cycle of the deliverable to verify and validate that the system documentation and coded application adhere to the measures specified at the commencement of the project. Although testing is primarily aimed at ensuring the application is delivered as defect free as possible testing is not about bug free deliverables. Testing is put in place to provide formal confirmation that the development meets the client’s requirements and information about the status of the development to ensure decisions are made in an informed manner.

Testing on its own, is powerless to create a high quality deliverable and the execution of testing, resulting in finding and fixing defects, does not help if the deliverable is unusable and does not fulfil the clinical users’ needs and expectations. Testing is also generally impossible without the existence of requirements and specifications. The quality of these documents will determine the quality of the testing.

Testing, therefore, is considered a distinct subset of quality management and both these concepts coalesce to verify the process of building the software as well as validate the coded deliverable. That is, they ensure the right processes are utilised to build the application and that the right application is delivered at the conclusion of the process.

This document details the internal testing methodology employed at DHCW. It does not provide details of DHCW’s Quality Assurance procedures.

V-MODEL OF TESTING

The V-Model of testing outlines the relationships between each phase of the development life cycle and its associated phase of testing and provides a benchmark view of the way in which testing typically progresses through the implementation of a deliverable. Although a simplified look at the progression of testing, the V-model is used to determine what specifications and subsequently what testing is required throughout a project.

The V-Model is not considered a testing methodology nor is it only applicable to waterfall type projects. Its main purpose is as a tool to identify each high level test stage, with these then being further disseminated within the project's test strategy.

The below diagram provides a look at a simple V-model and how it is used to relate testing to the solution specifications and previous phases of the project cycle.

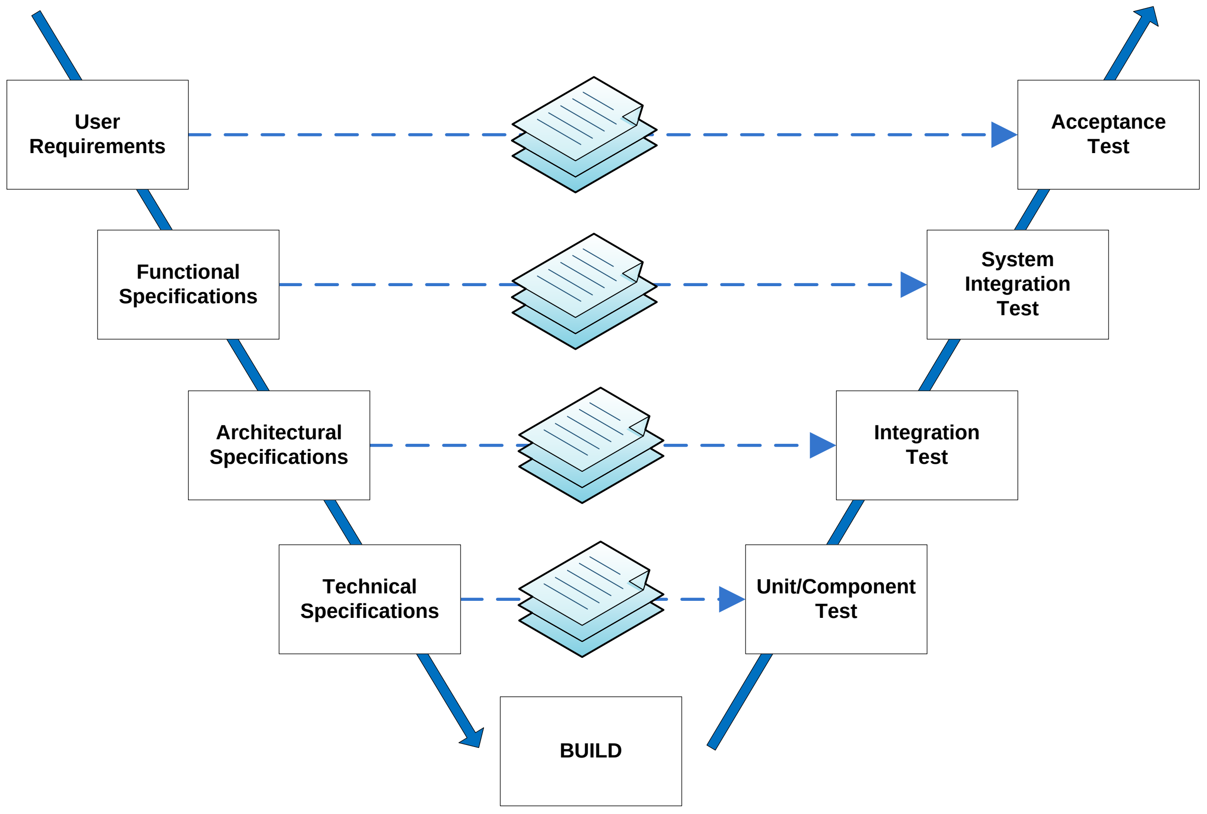

FIGURE 6 – V-MODEL

AGILE TESTING

Agile development encompasses an incremental, iterative and adaptive methodology. Incremental in that the system is built gradually enabling project members to easier demonstrate progress and end users to be involved and see how the system grows. Iterative in that the project involves multiple, smaller releases in an effort to minimise risk. Each release, or iteration, or 'Sprint', resembles a miniature software project and includes all of the tasks necessary to release the mini increment of new functionality: planning, analysis, design, coding and testing. Adaptive in that goals may change with each iteration based on lessons learned, feedback from clinical end users and changing clinical priorities.

Within Agile development, testing is not a phase at the end; it is integrated throughout the entire iteration cycle and is repeated with each cycle. This results in test resources being required for a longer overall time period as test phases are incremental throughout the duration of the project. In addition, the level of rigour attached to documentation can differ from that within a traditional waterfall project, although changes to requirements still need managing in a way that enables testers to know that what they are testing is accurate.

There are recommended practices within Agile Development which can assist testing and the subsequent quality of each release. Some of these include:

- Create unit test cases prior to coding
- Developer Peer Review and Code Inspections
- No new requirements for an iteration once the iteration has commenced
- Programmers and Testers should co-locate and develop a peer to peer relationship, collaborating throughout the sprint cycle
- Pair Programmers with Testers so that developers can gain insight into potential issue areas
- Pair Testers with users or analysts so that testers can better understand the user requirements
- Regression Testing after each iteration is important

Appendix 4 – Test Types

Test Type	Definition
Functional GUI Testing	This test type focuses exclusively on the graphical user interface (GUI) functionality of the client application. This type analyses the operation of each screen function and ensures that the client process flow throughout the system has been implemented correctly. Specifically, this test type checks the following: ▪ Client application look and feel ▪ Cross-system consistency and standardisation ▪ Accurate responses to all GUI actions ▪ GUI input validation ▪ GUI response to illegal or anomalous actions or inputs ▪ System usability ▪ Application help facilities The test scenarios for this type are administered directly onto the client system (which could be a browser/web server package or a windows based application) and can be undertaken manually by the test analyst. As part of the Functional GUI test type, Standard Tests are performed on each screen of the application. Standard Tests should analyse the following: ▪ GUI Window functionality ▪ GUI Formatting including Spelling and Tabbing order ▪ GUI Menus ▪ Keyboard Functions ▪ Grid Controls ▪ Browser navigation
Functional Data Testing	Functional Data Testing focuses on the accuracy of the data between the client application and the underlying database. This type specifically checks the accuracy of the following client functions and processes: ▪ The accuracy of data input ▪ The accuracy of system queries that extract data from the database to display on client screens. ▪ The accuracy of data calculations ▪ The accuracy of data updates initiated manually or automatically (e.g. date/time roll-over, cascading updates & deletes, etc.) ▪ Accuracy of GUI data exports

	There are a number of steps that are required for this test type. These include: ▪ Input of a defined data set that will be utilised in subsequent reconciliations ▪ Initiation of data queries from the client application ▪ Initiation of data updates (this could be manual e.g. user creates an entry, or automatic, e.g. the audit functionality records an event) ▪ Reconciliation of the data posted back to the client GUI with the expected result in the test script ▪ Extraction of the actual values in the data repository via an independent channel (e.g. direct SQL query on a database) and the execution of a reconciliation of the results with the GUI results set and the expected result set
Interface Testing	Interface testing is performed to validate the connections and data movement between interconnected systems to ensure components that are downstream and upstream from the application under test, send and receive data as expected.
Application Interaction	These scenarios are designed to ensure that the client application can operate on a customer's PC without hindering the operation of other applications that may be open at the time and vice versa. These other applications can include: ▪ Common shrink-wrapped, off the shelf applications e.g. Microsoft Office programs. ▪ In-house client applications ▪ Third party bespoke applications This type specifically tests the following: ▪ The interaction of the client application with other systems ▪ The interaction of multiple copies of the client application running simultaneously ▪ The operating footprint of the application and its impact on workstation performance In general, multi-application interaction test scenarios are run in the later stages of system testing and can be undertaken in parallel with the system testing types already discussed above.
Configuration Testing	In most large organisations, such as the NHS, users' computer workstations and the server systems are standardised. Therefore the operation of an application should be consistent across all workstations in an organisation. However, there will always be exceptions to a standardisation initiative, which can have a detrimental effect on any new system. These exceptions include:

	▪ Differing date formats between international operations ▪ Differing regional settings between international centres ▪ Differing versions of operating systems (Windows, Windows NT, Windows XP, Mac OS X) and browsers (Internet Explorer, Netscape) ▪ Differing hardware configurations (e.g. international keyboard differences, architecture, etc.) To catch these issues, the test scenarios for this type are run on a range of workstation and server configurations to ensure cross-platform compatibility. This type is usually run in the later stages of system testing or in parallel with the System Testing types described above.

Non-functional Testing	Non-Functional system testing involves testing everything that doesn't directly relate to the functionality of the system. This testing is more concerned with how well a system performs its function than the functions themselves. Non-Functional Test Types include:
	
	▪ Load Testing – the process of generating demand on a component, system or device and measuring its response. Often this testing involves the simulation of multiple users accessing a system concurrently in order to mimic the expected usage.
	▪ Performance Testing – undertaken to ascertain how fast some aspect of a system performs under a particular workload. It can be used for different purposes: to prove that the system meets pre-defined performance criteria; to compare two systems to determine which performs better; to measure what parts of a system causes it to perform badly (i.e. to locate bottlenecks).
	▪ Stress Testing – used to determine the stability, and understand the uppermost limits, of a system. It involves testing beyond normal operational capacity, often to breaking point, and observing the results to allow mitigation plans to be formulated if required. A subset of stress testing is Data Volume Testing which tests the behaviour of an application with large amounts of data in the system.
	▪ Security Testing – to ensure the deliverable is secure. This testing may include all or parts of the following:
	  o Network Surveying
	  o Port Scanning
	  o Router Testing
	  o Firewall Testing
	  o Intrusion Detection System Testing
	  o Password Cracking
	  o Denial of Service Testing
	  o Database intrusion

Regression Testing	Retesting parts of the system that have changed is implicit. That is, when new parts of system are deployed or a defect resolved, it is obvious that these areas are re-tested. However, it is also important to test those parts of the system

	that have not noticeably changed but may still be affected by the new or updated code. To ensure rework around regression testing is kept to a minimum: ▪ A reusable set of test scripts should be identified and archived to allow for expedient testing ▪ If appropriate, an automated tool can be utilised to perform regression testing. However this is only recommended if the system as a whole is stable enough that the same tests can be repeated without excessive maintenance. ▪ Release management is planned to reduce the regression testing required.
Infrastructure	This testing ensures that the deliverable works as expected in the environment on which it will be finally installed. This testing requires a test environment that simulates the production infrastructure in terms of hardware, networking components, security, software and peripherals.
Automation Testing	The use of software to perform or support test activities, e.g. test management, test design, test execution and results checking
Connectivity	Testing to ensure connection to networks, internet, local connections are correctly configured
Deployment	This testing ensures that deployment procedures have been accurately defined and that technical staff understand the requirements, procedures and stages to correctly deploy the system into the live environment.
Storage	Does the system meet its specified primary and secondary storage requirements? Does the program/component under test store data correctly and does the data store reserve sufficient space to prevent errors resulting from lack of memory or capacity?
Disaster Recovery	The process of regaining access to the data, hardware and software necessary to resume critical clinical operations after a disaster is termed Disaster Recovery. Disaster Recovery Testing therefore, is the testing of those processes that form the Welsh NHS overall DR plan
Failover	If a primary component fails, a secondary system should automatically engage. The primary objective of Failover Testing is to test the system recovery measures in place for the production architecture.
Backup/Restore	Backup testing ensures that the data backup policies and procedures agreed and implemented work as expected. However, backups are only as good as the restore procedures. Restore testing ensures the support team can correctly retrieve and restore any backup.
Performance Monitoring tools	Testing to validate any system alarms that have been established. For example, alarms indicating peak disk usage.
Security	The Process to determine that a system protects data and maintains functionality as intended. The basic security concepts that should be covered by security testing are: confidentiality, integrity, authentication, authorization and availability.

Soak Testing	This testing involves running a system at high levels of load for prolonged periods of time. A soak test would normally execute several times more transactions in an entire day than would be expected in a busy day, to identify any performance problems that appear after a large number of transactions have been executed.

Appendix 5 – Clinical Risk Management Risk Scores

The scoring of clinical risk requires a risk classification scheme in order to assist with prioritising risk reduction activities and subsequent assessment of test issues and service incidents.

Table 3 defines categories of severity associated with system risks. These categories relate to single incidents of failure, which may affect individual patients or several patients at once.

Severity of Risk		
Descriptor	**Score**	**Consequence**
Negligible (minimal)	1	No harm to patients, loss, delay, inconvenience or interruption on services. Minimal injury requiring no/minimal intervention or treatment. Can be easily and quickly remedied. Peripheral element of treatment or service suboptimal.
Minor	2	Short term harm to single patient, loss, delay, inconvenience or interruption on services. Increase in length of hospital stay by 1-3 days. Overall treatment or service suboptimal.
Moderate	3	Harm affecting single patients for up to a year, or minor harm to multiple patients. Increase in length of hospital stay by 4-15 days Medium term effect which may be expensive to recover. Treatment or service has significantly reduced effectiveness
Major	4	Permanent harm to a single patient or moderate harm to multiple patients. A serious impact on quality and/or output and reputation. Mismanagement of patient care with long-term effects. Increase in length of hospital stay by >15 days. Medium to long-term effect and difficult to recover.
Catastrophic	5	Death of a single patient or multiple permanent injuries or irreversible health effect. An event which impacts on a large number of patients. Totally unacceptable level or quality of treatment/service. Huge impact on reputation and/or costs.

Table 3

Table 4 defines ***categories of likelihood*** of the occurrence of system risks.

Categories of likelihood of the occurrence of system risks.		
Descriptor	**Score**	**Probability**
Almost certain	5	It will undoubtedly happen/recur, possibly frequently. Expected to occur at least daily in any area where the product is used. Greater than one in 10 patients per year (10% error rate)
Likely	4	It will probably happen/recur but it is not a persisting issue. Expected to occur at least weekly in any area where the product is used. Harm is unlikely to be prevented by Clinician. One in 10-on in 100 patients per year (10-1%) error rate
Possible	3	It might happen or recur occasionally. Expected to occur at least monthly in any area where the product is used. Harm may be prevented by Clinician. One in 100 – one in 1000 patients per year (1- 0.1% error rate)
Unlikely	2	Do not expect it to happen/recur but it is possible that it may do so. Expected to occur at least annually in any area where the product is used. The risk may be corrected and/or managed. One in 1000- one in 10 000 patients per year (0.1-0.001%) error rate
Rare	1	This will probably never happen/recur (except in very exceptional circumstances) in the life cycle of the product. Greater than on in 10 000 patients per year (> 0.001% error rate)

Table 4

Table 5 below combines the risk severity and likelihood categories to produce a ***Risk Classification Matrix,*** which provides a measure of safety risk. These measures have been grouped into 'Risk Acceptance' categories.

Likelihood	Almost Certain 5	5 (Moderate)	10 (Significant)	15 (High)	20 (High)	25 (High)
	Likely 4	4 (Moderate)	8 (Moderate)	12 (Significant)	16 (High)	20 (High)
	Possible 3	3 (Low)	6 (Moderate)	9 (Significant)	12 (Significant)	15 (High)
	Unlikely 2	2 (Low)	4 (Moderate)	6 (Moderate)	8 (Moderate)	10 (Significant)
	Rare 1	1 (Low)	2 (Low)	3 (Low)	4 (Moderate)	5 (Moderate)
		1 Negligible	2 Minor	3 Moderate	4 Major	5 Catastrophic
		Consequence				

Table 5

Table 6 defines ***Acceptance of Risk Categories*** associated with risks and sets out the required mitigating actions. It also defines the actions required for the acceptance of safety risk associated with in-service incidents.

Table 6

Risk	Acceptance Category	Action required for identified risks
Low (1-3)	Acceptable	A project will be content to carry these risks. No need to consider the risk appetite nor to proceed any further with the assessment but merely record in the risk register that the risk has been identified and that due to its low likelihood or impact no further action will be required. No justification is required unless explicitly requested.
Moderate (4-8)	Justifiable	A project needs to consider mitigation against these risks. Justification for the acceptability of risk controls to be recorded in the Clinical Safety Case Report and supported by evidence.
Significant (9-14)	Intolerable	A project will be concerned about these risks. The risks need to be managed by the project, depending on the impact, and may need ongoing assurance to the Project Board. DHCW/Project Test Leads to be notified of the risk as soon as practicable and appropriate mitigating action agreed. Where agreed mitigation leads to additional functional or non-functional requirements, these will be identified in the Clinical Safety Case Report and evidence for their achievement provided.
High (14-25)	Unacceptable	These risks will be considered as significant and will need proactive review and oversight by the Project Board. DHCW/Project Test Leads to be notified of the risk as soon as practicable. Re-specification or re-engineering is required to eliminate the risk. This risk may only be accepted under exceptional circumstances.