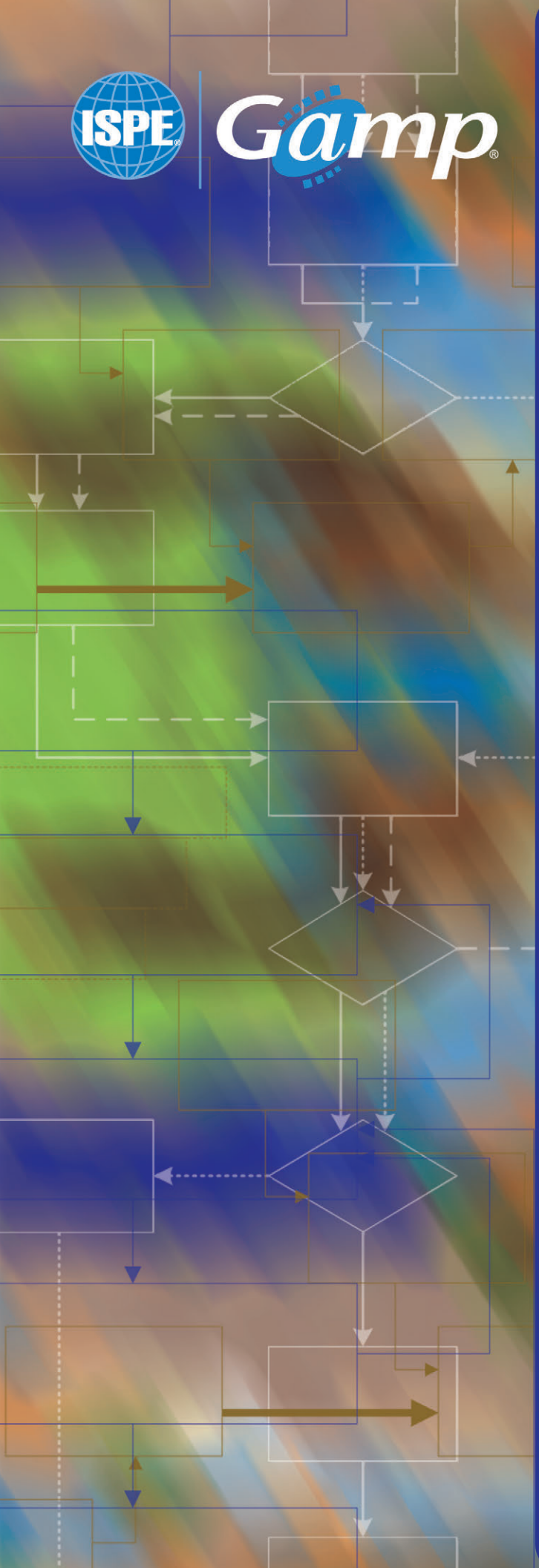


GAMP Good Practice Guide

A Risk-Based Approach to Testing of GxP Systems

Second Edition

What's your approach...

A leader in providing Computer Systems Implementation, Validation and Quality Assurance Services.

CQV (An Azzur Group Company) provides consulting services to the Life Sciences Industry. We offer system integration, compliance and validation services within the Quality Assurance, Laboratory, Clinical, IT/IM, Business Processes, Manufacturing, Operations, and R&D settings to our pharmaceutical, biotechnology and medical device clients.

With a depth of expertise in all facets of the system life cycle we are able to assist our clients within any phase to reach their goals of running a lean, efficient operation.

SERVICES AVAILABLE

- RISK-BASED LAB RELOCATION
- COMPUTER SYSTEMS VALIDATION
- INSTRUMENT IQ/OQ/PQ SERVICES
- LIFE CYCLE DOCUMENTATION
- LIMS SYSTEM QUALIFICATION
- IT INFRASTRUCTURE
- PROJECT MANAGEMENT

"...to achieve success?"

CQV (An Azzur Group Company) takes a holistic approach when working with our clients to provide the best service possible. We work hand in hand with each client to achieve success by ensuring that all goals are met and safe and effective products are delivered.

CQV
An Azzur Group Company

Azzur Group LLC
helping life sciences succeed

Tel: 800.726.0331 Email: info@azzurgroup.com www.azzurgroup.com

GAMP Good Practice Guide

A Risk-Based Approach to Testing of GxP Systems

Second Edition

Disclaimer:
This Guide focuses on functional, structural, and performance testing of GxP systems and is intended to assist regulated organizations and suppliers to work together to ensure sufficient test coverage to guarantee fitness for intended use, while minimizing any duplication of effort. ISPE cannot ensure and does not warrant that a system managed in accordance with this Guide will be acceptable to regulatory authorities. Further, this Guide does not replace the need for hiring professional engineers or technicians.

Limitation of Liability
In no event shall ISPE or any of its affiliates, or the officers, directors, employees, members, or agents of each of them, or the authors, be liable for any damages of any kind, including without limitation any special, incidental, indirect, or consequential damages, whether or not advised of the possibility of such damages, and on any theory of liability whatsoever, arising out of or in connection with the use of this information.

ISBN 978-1-936379-51-4

Preface

This document, the ISPE GAMP® Good Practice Guide: A Risk-Based Approach to Testing of GxP Systems, represents a revision of the first edition, ISPE GAMP® GPG – Testing and is intended as a supplement to ISPE GAMP® 5: A Risk-Based Approach to Compliant GxP Computerized Systems. This Guide focuses on functional, structural, and performance testing. It has been updated to align with the concepts and terminology of GAMP® 5 and regulatory and industry developments which focus attention on patient safety, product quality, and data integrity.

The ISPE GAMP® Community of Practice Document Task Team was asked to revise this Guide. The team consisted of representatives from regulated organizations, contract research organizations, suppliers of pharmaceutical systems and equipment, and consultants.

Acknowledgements

The production of the ISPE GAMP® Good Practice Guide: Risk-Based Approach to Testing of GxP Systems was initiated by the Testing Task Team at the request of the GAMP® Council.

The following members of the GAMP® Community of Practice (COP) Testing Task Team worked on one or more sections of this Guide and volunteered countless hours to attend meetings and review the many drafts. The members committed to, and achieved, a challenging timescale for this substantial revision and their exceptional efforts are much appreciated.

Chairs

Karen Ashworth	Karen Ashworth Consulting	United Kingdom
Charlie Wakeham	Pall Life Sciences	United Kingdom

Task Team Members

Kevin Ashley	Siemens Healthcare Diagnostics	United Kingdom
Karen Ashworth	Karen Ashworth Consulting	United Kingdom
Roger Buchanan	Lilly UK	United Kingdom
Jonathan Davey	Provalidus Ltd.	United Kingdom
Stephen Dawson	Eden Biodesign Ltd.	United Kingdom
Angille K. Heintzman	NHS Blood and Transplant	United Kingdom
Daniel G. Mewborn	Eli Lilly and Company	USA
Daniel Montgomery	Covance, Inc.	USA
Radha Ramesh	TCS	USA
Genni Sanders	Systematicity	United Kingdom
George Smerdon	Industrial Technology Systems Ltd. (ITS)	United Kingdom
Paul Smith	Agilent Technologies	United Kingdom
Kimberly Stanton	QPharma, Inc.	USA
David Stokes	Business & Decision Life Sciences	United Kingdom
Thanabalan Subramanian	GE Healthcare	United Kingdom
Matthew Theobald	Three Circles	United Kingdom
Steve Tinson	Scitech	United Kingdom
Simon Topham	Napp Pharmaceuticals Limited	United Kingdom
Anders Vidstrup	NNIT A/S	Denmark
Linda Waddick	Eli Lilly and Company	USA
Charlie Wakeham	Pall Life Sciences	United Kingdom

Within this highly professional and committed team, special thanks are due to David Stokes, Kimberly Stanton, and Steve Tinson for their expertise in innovative technologies and methodologies, and their dedication in formulating the guidance in these areas.

Contributing Members

The following contributors are members from the GAMP® COP US Automation SIG and GAMP® COP DACH Automation SIG. David Stokes coordinated the work of the Automation SIGs and integrated that work into the Guide as Appendix T11.

Peter Brandstetter	Arcondis GmbH	Austria
Konstantin Clevermann	IDS Scheer Consulting GmbH	Germany
Thierry Dietrich	Arcondis	Germany
Srinivas Gopanapalli	Cognizant	USA

Andreas Hengstberger	Nycomed Austria GmbH	Austria
Bernhard Kausler	ITQ GmbH	Germany
Oliver Kieckhöfel	BTConsult GmbH	Germany
Stefan Münch (DACH Lead)	Rockwell Automation Solutions GmbH	Germany
Radha Ramesh (US Lead)	TCS	USA
Shweta Rangarajan	Celgene Corp.	USA
Ernst-Reiner Schäfer	Abacon GmbH	Switzerland
Jason R. Tepfenhardt	Genilogix	USA

The GAMP® Testing Task Team would like to acknowledge the support of their direct management and companies in this endeavor.

Particular thanks go to the GAMP® COP Editorial Review Board for their review and comment:

Winnie Cappucci	Bayer Healthcare (retired)	USA
Chris Clark (Chair)	Napp Pharmaceuticals Limited	United Kingdom
Gail Evans	ISPE	United Kingdom
Colin Jones	Conformity Limited	United Kingdom
Randy Perez	Novartis	USA
Sion Wyn	Conformity Limited	United Kingdom

The Leaders of the GAMP® Testing Task Team wish to thank GAMP® Council, GAMP® Europe Steering Committee, and the Guidance Documents Executive Committee (GDEC) for their contributions and commitment throughout the production of this Guide.

Connecting a World of Pharmaceutical Knowledge

ISPE Headquarters
600 N. Westshore Blvd., Suite 900, Tampa, Florida 33609 USA
Tel: +1-813-960-2105, Fax: +1-813-264-2816

ISPE Asia Pacific Office
73 Bukit Timah Road, #04-01 Rex House, Singapore 229832
Tel: +65-6496-5502, Fax: +65-6336-6449

ISPE China Office
Suite 2302, Wise Logic International Center
No. 66 North Shan Xi Road, Shanghai, China 200041
Tel +86-21-5116-0265, Fax +86-21-5116-0260

ISPE European Office
Avenue de Tervueren, 300, B-1150 Brussels, Belgium
Tel: +32-2-743-4422, Fax: +32-2-743-1550

www.ISPE.org

Table of Contents

1 Introduction

1.1 Overview

This Guide is a revision of the first edition of the ISPE GAMP® Good Practice Guide (GPG): A Risk-Based Approach to Testing of GxP Systems. It has been updated to align with the concepts and terminology of GAMP® 5 [1], associated GAMP® guidance, and recent regulatory and industry developments. These developments focus attention on patient safety, product quality, and data integrity.

GAMP® 5 and associated GPGs aim to provide guidance to achieve computerized systems that are fit for intended use and meet current GxP regulatory requirements, by building upon existing industry good practice in an efficient and effective manner. This Guide builds on the framework described in GAMP® 5, and provides detailed guidance on testing GxP systems.

GAMP® is an ISPE Community of Practice (COP). For further information, see www.ispe.org [2].

The approach and terminology used in this Guide are generally harmonized with the following industry guidance:

- International Conference on Harmonization (ICH) Guidance including Q8 [3], Q9 [4], and Q10 [5]
- ASTM Standard E2500-07, Standard Guide for Specification, Design, and Verification of Pharmaceutical and Biopharmaceutical Manufacturing Systems and Equipment [6]
- EU GMP EudraLex Volume 4 (Chapter 4) [7] and Annex 11 [8]

This edition of the Guide also has been aligned with advances in industry best practice, including:

- Increased adoption and implementation of Process Analytical Technology (PAT) and Quality by Design (QbD)
- Increased industry focus on risk-based approaches
- Increased use of non-linear development life cycles
- Increased use of automated test tools

Consideration should be given to the existing policies and procedures within a regulated organization and to other external industry standards and practices when applying the principles described in this Guide.

1.2 Purpose

This Guide has been written to provide regulated organizations and suppliers with pragmatic guidance on the testing of computerized and software based systems that impact patient safety, product quality, and data integrity. The key objective of this Guide is to encourage regulated organizations and suppliers to work together to ensure sufficient test coverage to guarantee fitness for intended use, while minimizing any duplication of effort.

The Guide seeks to identify the testing that should be performed and the associated level of documentation. Where suppliers' systems do not meet the expectations of a regulated organization, the Guide identifies suitable risk control strategies. These strategies can include the execution of additional testing, or the selection and use of alternative suppliers or products, by the regulated organization.

This Guide intends to provide pragmatic answers to questions such as:

- Why should I test?
- What should I test?
- How much testing is enough?
- How should I conduct tests?
- How should I document my testing?

Specifically, this Guide is intended to take a risk-based approach to compliance of GxP computerized systems and provide practical advice on the application of this approach in the planning and execution of testing.

The Guide is intended to assist:

- Regulated organizations (the pharmaceutical customer or regulated user organization contracting a supplier to provide a product)
- Suppliers (usually external third parties, but also including “in-house” providers of IT services) including:
  - Suppliers of standard products
  - Systems integrators responsible for configuration and coding of standard product to create a specific application
  - Suppliers of control systems that are packaged with the process equipment
  - Service providers
- Regulatory agencies

The overall computer systems compliance approach recommended for regulated organizations is described in GAMP® 5. Suggested good practices for suppliers of GxP systems and services are described in Supplier Activities (Section 7) of GAMP® 5 [1].

1.3 Scope

The scope of this Guide is the testing of GxP regulated computerized systems, including systems used in regulated activities covered by:

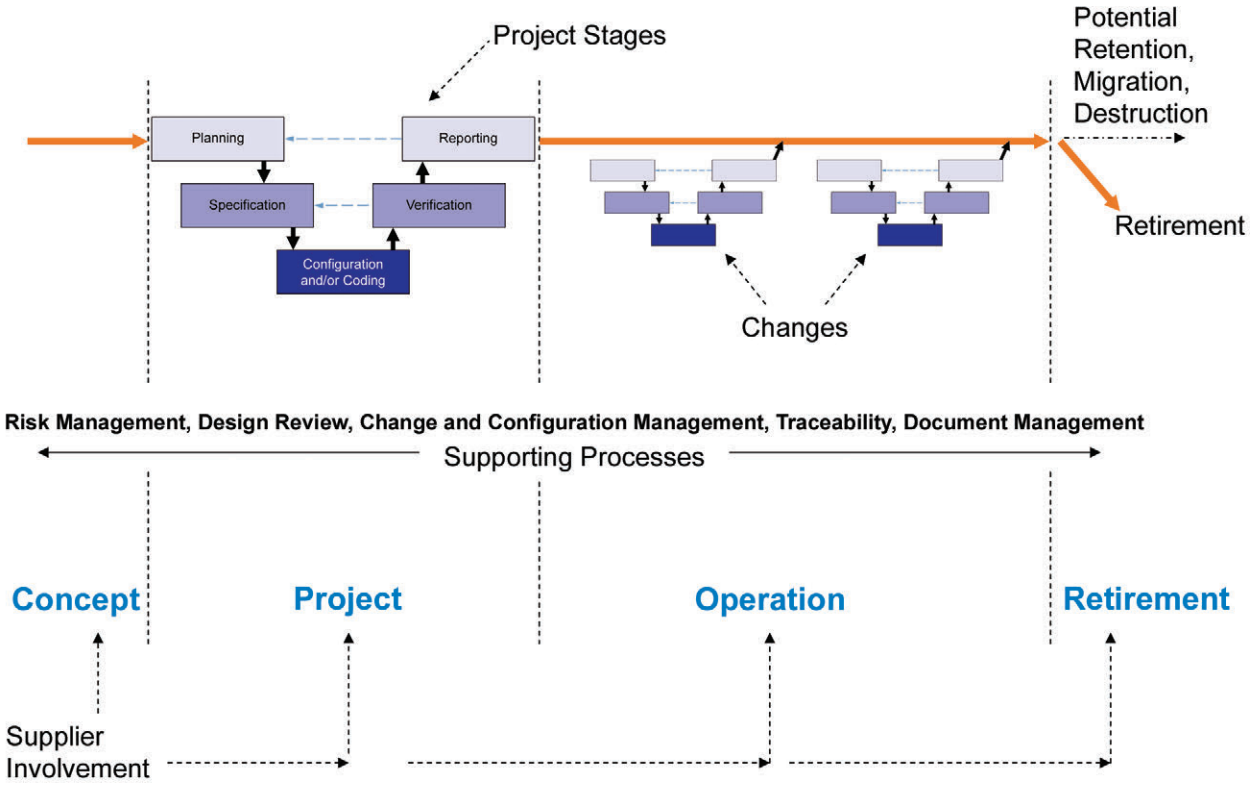
- Good Manufacturing Practice (GMP) (pharmaceutical including Active Pharmaceutical Ingredient (API), veterinary, and blood)
- Good Clinical Practice (GCP)
- Good Laboratory Practice (GLP)
- Good Distribution Practice (GDP)
- Medical Device Regulations (with the exception of software embedded within medical devices)

The Guide focuses on functional, structural, and performance testing. It also discusses other:

- Activities of the verification stage of the project phase of the life cycle
- Verification activities used to support or replace testing activities as a result of a risk assessment

Testing activities within the operation and retirement phases the life cycle are also considered. Other life cycle activities are out of scope.

Figure 1.1: GAMP® 5 Life Cycle



In the project and operation phases, verification should confirm that the regulated organization (or supplier) requirements and project specifications have been met. This can be achieved by applying an appropriate combination of:

- Design reviews
- Checking installation
- Reviewing configuration and data items, e.g., recipes
- Testing against specifications, e.g., module testing, integration testing, and functional testing
- Testing against requirements to ensure intended use will be consistently fulfilled
- Testing to ensure identified risk controls are in place and effective

This Guide will concentrate on the testing activities within verification.

The focus of this Guide is GxP risk (i.e., risk to patient safety, product quality, and data integrity (and data confidentiality)); other aspects of testing (e.g., risks related to health and safety for personnel/caregivers or risks related to the environment) are out of scope of this Guide.

In the preparation of this Guide, the authors have deliberately chosen not to redefine testing good practice used in the wider software development and testing community. Focus is given to the pragmatic application of these practices to the testing of GxP systems and references for further reading are provided where appropriate.

1.4 Benefits

The main benefits from verifying systems include:

- Improved cost effectiveness from moving into the operational environment with systems that are fit for their intended use
- Ease of demonstrating compliance to regulatory requirements
- Improvement in patient safety, product quality, and data integrity as a result of control of identified risks

Insufficient or inappropriate testing may cause problems later in the system life cycle; these problems could be time consuming and troublesome to resolve.

Where there is pressure to implement systems in timescales that are unrealistic, there are several potential consequences:

- Reduction in the effectiveness and efficiency of the system at go live
- Increased maintenance and support costs
- A costly program of corrective actions may need to be implemented to correct faults and meet the original requirements
- A system that does not meet the basic user requirements is released into the operational environment

Effective testing will reduce the overall life cycle costs of implementing and operating the system, and prevent delays to the effective and efficient use of the system.

1.5 Objectives

Testing is concerned with identifying defects so that they can be corrected, as well as demonstrating that a system meets requirements. Testing computerized systems is considered a fundamental verification activity. Appropriate testing is a regulatory expectation. As stated in EU GMP Annex 11 [8]:

> *"Evidence of appropriate test methods and test scenarios should be demonstrated. Particularly, system (process) parameter limits, data limits and error handling should be considered."*

This indicates a focus on careful choice of test strategy, including negative or challenge testing, intended to identify defects.

Verification should confirm that project specifications have been met. This may involve multiple stages of reviews and testing depending on the type of system, the development method applied, and the use of the computerized system. Regulated organizations should be prepared to justify the adequacy of their verification approach.

Appropriate testing should:

- Assure patient safety, product quality, and data integrity (and data confidentiality)
- Improve reliability and reducing system downtime; increasing user confidence in the system
- Reduce the cost of commissioning, start-up, and on-going support
- Protect business reputation and credibility

A science-based quality risk management process (e.g., using ICH Q9 [4]) should be used to determine the appropriate scope and rigor of system verification. Tests should be designed to demonstrate that all required risk controls are in place.

Failure to appropriately test functions which have high impact on patient safety, product quality, and data integrity appropriately may undermine the compliance and fitness for intended use of a system. Inadequate testing also may lead to regulatory citation and possibly further regulatory action.

Any proposed change to a system should trigger an impact assessment to determine the extent of any re-verification, including any regression testing required. Alterations to a system should be made only in accordance with a predefined procedure for change control and change management which should include provision for approving, implementing, verifying, and, if necessary, backing out the change.

1.6 Structure of this Guide

1.6.1 Overview of GAMP® Documentation Structure

This Guide forms part of a family of documents that together provide a powerful and comprehensive body of knowledge covering all aspects of computerized systems good practice and compliance.

Figure 1.2: GAMP® Documentation Structure

1.6.2 Structure of this Guide

This Guide contains the following:

- Main body defining how the GAMP®5 Key Concepts [1] relate specifically to testing and outlining regulated organization and supplier considerations
- A set of “T” appendices giving detailed guidance on test practices, applicable across all phases in the life cycles unless explicitly stated otherwise
- A set of “E” appendices providing examples of special considerations involved in applying the test practices to different system types

Templates and examples are also included in the appendices to assist in the development of appropriate documents, suitable for planning, executing, and reporting on the testing of a GxP system.

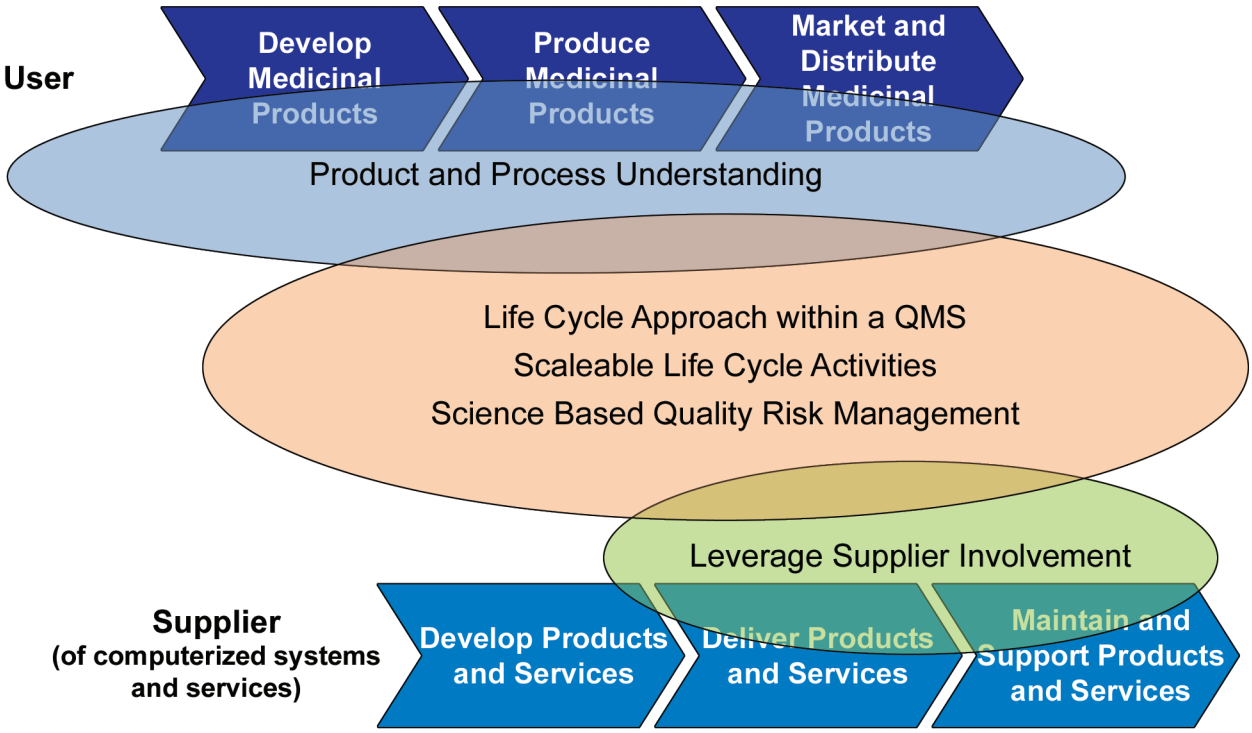
2 Key Concepts

Within this section, the key concepts from GAMP® 5 [1] are shown *italicized* and are followed by a discussion of the application of each concept to the testing of GxP systems.

The following five key concepts are described in GAMP® 5 [1].

- *Product and Process Understanding*
- *Life Cycle Approach within a Quality Management System (QMS)*
- *Scalable Life Cycle Activities*
- *Science-Based Quality Risk Management*
- *Leveraging Supplier Involvement*

Figure 2.1: GAMP® 5 Key Concepts



2.1 Product and Process Understanding

An understanding of the supported process is fundamental to determining system requirements. Product and process understanding is the basis for making science- and risk- based decisions to ensure that the system is fit for its intended use.

Efforts to ensure fitness for intended use should focus on those aspects that are critical to patient safety, product quality, and data integrity. These critical aspects should be identified, specified, and verified.

Systems within the scope of this Guide support a wide range of processes, including clinical trials, toxological studies, API production, formulated product production, warehousing distribution, and pharmacovigilance.

For some manufacturing systems, process requirements depend on a thorough understanding of product characteristics. For these systems, identification of Critical Quality Attributes (CQAs) and related Critical Process Parameters (CPPs) enable process control requirements to be defined.

Specification of requirements should be focused on critical aspects. The extent and detail of requirement specification should be commensurate with associated risk, complexity, and novelty of the system.

Incomplete process understanding hinders effective and efficient compliance, and achievement of business benefit.

The understanding of both product and process (including business processes as well as physical processes) assists in determining appropriate test scope and strategy. It underlies the risk management processes that allow critical aspects to be identified and specified; therefore allowing those critical aspects to be verified as part of ensuring fitness for intended use. Traceability between requirements, identified risks, and test cases is an important part of demonstrating this fitness for intended use.

In the manufacturing arena where systems control critical process parameters, the testing of process controls should challenge both measurement and control systems to ensure that the process remains within acceptable limits. In an environment where process analytical technology is in use, testing also may need to include tests designed to challenge the process model and control model. Appendix E8 covers specific risk scenarios associated with the use of PAT and the ways in which testing can help to control these.

Where the process is not known, e.g., in a system designed to control a multi-purpose facility (such as is often found in contract manufacture or in Research and Development (R&D) pilot plants), testing should adequately verify the designed control capability.

Outside the manufacturing arena, risk management processes allow critical aspects to be identified (e.g., data processing functions that impact on a regulated organization's ability to support pharmacovigilance monitoring or to perform a product recall) so that testing can be appropriately focused to verify or challenge these aspects.

2.2 Life Cycle Approach within a QMS

Adopting a complete computerized system life cycle entails defining activities in a systematic way from system conception to retirement. This enables management control and a consistent approach across systems.

The life cycle should form an intrinsic part of the company's QMS, which should be maintained up to date as new ways of working are developed.

As experience is gained in system use, the QMS should enable continuous process and system improvements based on periodic review and evaluation, operational and performance data, and root-cause analysis of failures. Identified improvements and corrective actions should follow change management.

A suitable life cycle, properly applied, enables the assurance of quality and fitness for intended use, and achieving and maintaining compliance with regulatory requirements. A well managed and understood life cycle facilitates adoption of a QbD approach.

Testing plays a vital role in the project, operational, and retirement phases of a computerized system life cycle:

- During all phases, the main purpose of testing is to discover defects; therefore, avoiding those defects being present in the operational environment

- Within the project phase, testing helps to demonstrate fitness for intended use by verifying the correct operation of GxP critical functions and effective implementation of controls identified during risk assessment

- Within the operational phase, risk-based testing may be required following a system change to demonstrate that:

  - Any new functionality is correct.

  - Remaining original functionality has not been adversely affected.

  - Required risk controls are still in place and effective.

- Within the retirement phase, testing of data migration and/or archive and retrieval methods may be required prior to decommissioning the system.

The adoption of a suitable life cycle means that testing has a context where previous life cycle activities provide inputs to the testing process (e.g., the generation of clear requirements and risk controls that tests can be designed to verify and challenge). Testing feeds into later life cycle activities (including iterative activities such as re-design and re-implementation under change control). An understanding of this context is vital to ensuring that testing is appropriately scaled and is focused on higher risk areas.

Appendix T10 describes how appropriate testing can be applied when using non-linear development methods and models.

2.3 Scalable Life Cycle Activities

Life cycle activities should be scaled according to:

- *System impact on patient safety, product quality, and data integrity (risk assessment)*

- *System complexity and novelty (architecture and categorization of system components)*

- *Outcome of supplier assessment (supplier capability)*

Business impact also may influence the scaling of life cycle activities.

The strategy should be clearly defined in a plan and follow established and approved policies and procedures.

As with other life cycle activities, testing should be scaled according to system impact on patient safety, product quality, and data integrity, system complexity and novelty, and the outcome of supplier assessment. What types of test are required, and the scope and rigor of each test type, are determined in part by risk assessment. Tests may be grouped into test phases in a life cycle to reflect both the overall philosophy of testing and the practicalities of testing the particular system. Some types of test may be included in more than one test phase, e.g., functional tests could be present in both integration testing and user acceptance testing (or equivalent phases with alternative names, such as factory acceptance testing and site acceptance testing).

Scalable elements, based on a documented and justified risk assessment, include the following:

- Rigor of testing:

  - Some higher impact or more novel/complex functions may require structural testing as well as functional testing. Within the structural tests, some functions may demand full coverage of every possible path, while for others, covering each outcome of a decision once is sufficient. Other functions may be appropriately

verified with functional tests alone and still others may be appropriately verified by parameter or configuration checks. Scope and rigor of non-functional testing such as performance testing also can be scaled according to risk. Refer to Appendix T1 for further information on types of testing.

- Extent of testing:
  - Depending on the supplier assessment results, it may be appropriate to leverage the supplier's results entirely, perform sample checks focused on high risk functions, or perform a full, comprehensive set of tests. Refer to Section 3.4 for further information on Supplier Assessment and on optimizing testing done by the regulated organization.
- Test evidence:
  - Depending on the risk associated with the functionality under test, there may be a requirement (or not) for detailed manual recording of values or hard copy evidence. Refer to Appendix T6 for further information on appropriate methods for recording results
- Rigor of test incident management:
  - Rigor of problem resolution and tracking processes and the extent of regression analysis can be scaled based on the risk associated with the failed functionality and the phase of the life cycle. Refer to Appendix T2 for further information on the management of test incidents.
- Test review/approval:
  - Some tests may be reviewed/approved by an appropriate Subject Matter Expert (SME) while others require an additional input from the regulated organization management or quality unit. Refer to Appendix T2 for further information on test roles and responsibilities.

2.4 Science-Based Quality Risk Management

Quality risk management is a systematic process for the assessment, control, communication, and review of risks.

Application of quality risk management enables effort to be focused on critical aspects of a computerized system in a controlled and justified manner.

Quality risk management should be based on clear process understanding and potential impact on patient safety, product quality, and data integrity. For systems controlling or monitoring CPPs, these should be traceable to CQAs, and ultimately back to the relevant regulatory submissions for manufacturing systems.

Qualitative or quantitative techniques may be used to identify and manage risks. Controls are developed to reduce risks to an acceptable level. Implemented controls are monitored during operation to ensure ongoing effectiveness.

Quality risk management should be applied when defining test strategy and designing tests that are intended to verify and challenge a system. The controls identified by the quality risk management process should form the basis for designing tests that verify and challenge the functions within the system; with each control being considered as a starting point for generating test cases. This allows critical aspects of a computerized system to be identified and tested adequately.

Tracing that all controls have been verified as effective is part of effective test reporting and determination of residual risk.

2.5 Leveraging Supplier Involvement

Regulated companies should seek to maximize supplier involvement throughout the system life cycle in order to leverage knowledge, experience, and documentation, subject to satisfactory supplier assessment.

For example, the supplier may assist with requirements gathering, risk assessments, the creation of functional and other specifications, system configuration, testing, support and maintenance.

Planning should determine how best to use supplier documentation, including existing test documentation, to avoid wasted effort and duplication. Justification for the use of supplier documentation should be provided by the satisfactory outcome of supplier assessments, which may include supplier audits.

Documentation should be assessed for suitability, accuracy and completeness. There should be flexibility regarding acceptable format, structure, and documentation practices.

Testing is an area of the system life cycle in which there are potentially large efficiency gains to be made by the appropriate leveraging of supplier involvement; in particular through the:

- Avoidance of duplication in testing
- Leveraging of supplier test activities and evidence to the maximum practical extent whilewhile still ensuring fitness for intended use

Responsibilities should be defined between the regulated organization and the supplier, but the final responsibility for the system's fitness for intended use rests with the regulated organization.

Depending on the scope of the supply (see Section 3.1 of this Guide for types of supplier), testing may be performed by suppliers both during their own product development life cycle (as part of the release of a new standard product or function) and during the life cycle for developing the required end user application. Therefore, the testing of a system is a combination of:

- Testing conducted by the supplier during basic development of the standard product
- Testing conducted by the supplier (or integrator) during application-specific development, i.e., the development of a solution customized or configured to the customer's business process
- Testing conducted by the regulated organization

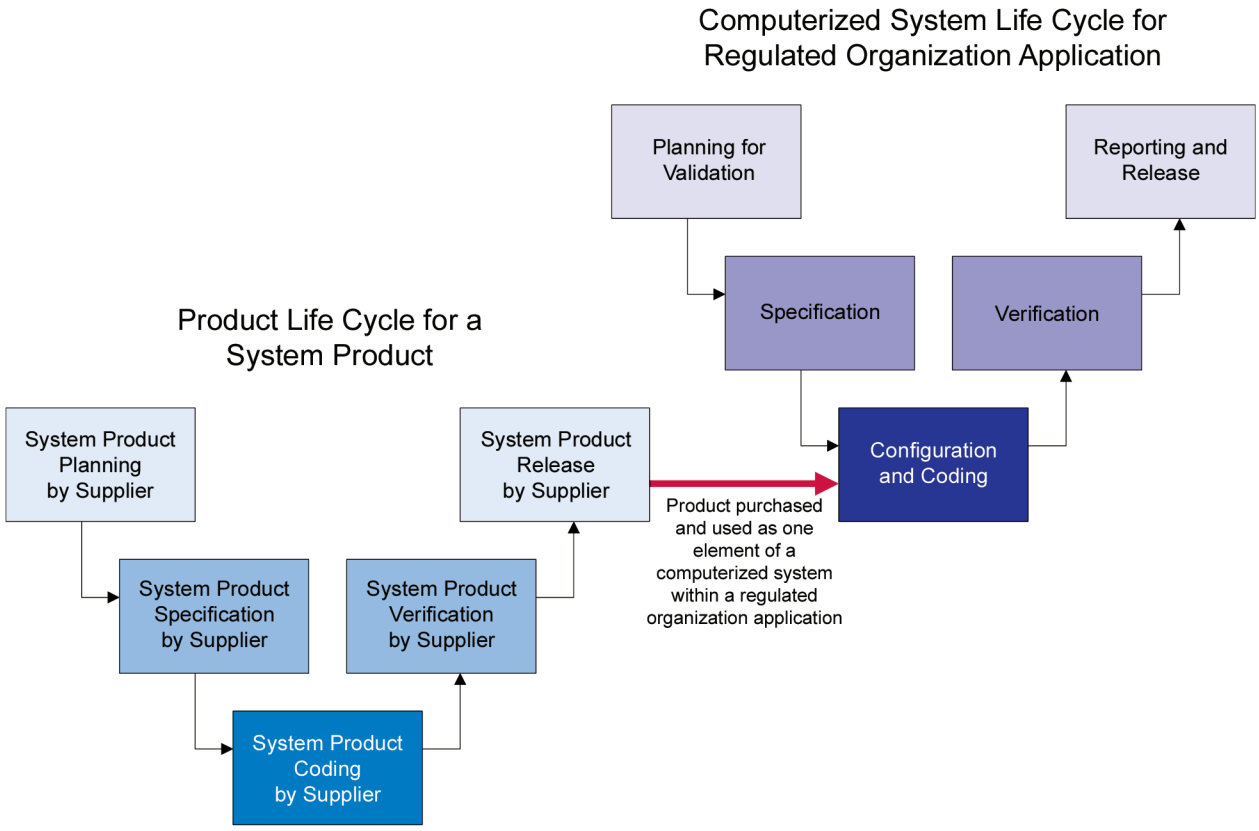
Where supplier(s) have been assessed and their quality management practices found to be appropriate, the regulated organization should seek to minimize the duplication of testing effort by:

- Leveraging testing already performed as part of the development of the standard products being used (e.g., focusing on high risk and customized rather than low risk and standard)
- Leveraging testing already conducted earlier in the development of the specific application (e.g., repeating on site only those tests relating to functions that may have been affected by the move to the operational environment, not those that were successfully passed during factory testing and which are unaffected by the move)
- Leveraging testing previously performed on earlier instances of equipment where business processes allow repeat purchases of that equipment
- Ensuring that test planning does not result in unnecessary duplication between different suppliers (particularly where communication interfaces or interfaces between systems and processes are involved)

The supplier may minimize duplication of testing effort through robust configuration management and re-use of common modules across multiple application-specific developments.

The examples that follow show supplier and regulated organization development and test activities and are based on the specification and verification model described in GAMP® 5 [1]. Many suppliers may use alternative development methods and these may be perfectly acceptable.

Figure 2.2: Leveraging Work Already Done in the Supplier's Product Life Cycle



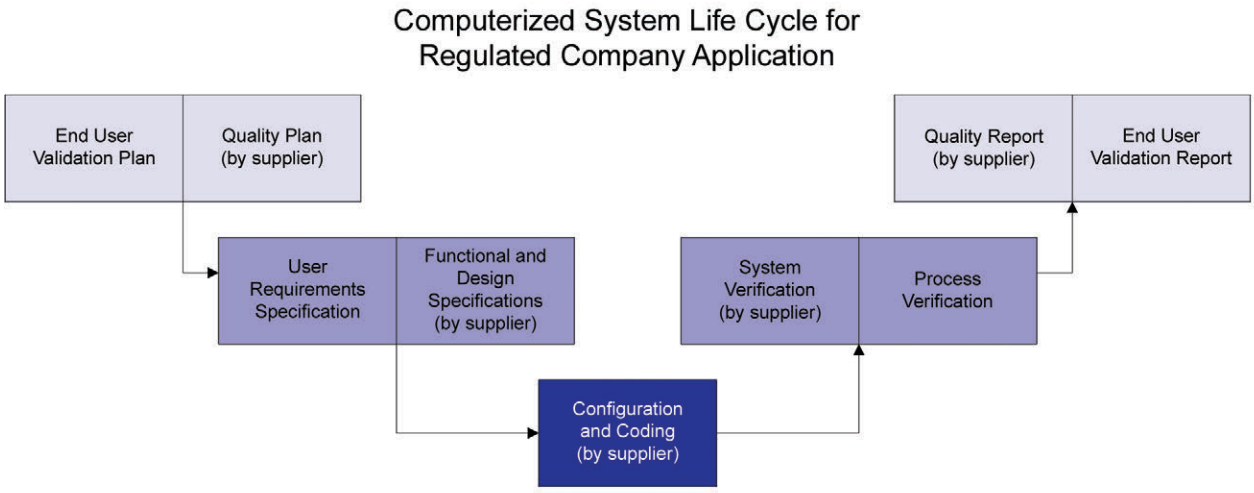
For example, where a non-configured product from an appropriately assessed supplier is used within an application, the verification element of the application life cycle may be limited to:

- Ensuring and recording correct installation (and parameterization if appropriate)
- Demonstrating and documenting fitness for intended use (against user requirements)
- Identifying any further tests required as a result of the quality risk management process

The structural level testing, functional testing, and challenge testing performed by the supplier during product development are leveraged rather than repeated.

Guidance on managing supplier test documentation is given in Table 3.4: Considerations for Determining Appropriate Test Evidence.

Figure 2.3: Leveraging Work by the Supplier during the Application Life Cycle



For example, where an assessed supplier is integrating a system for a regulated organization application, the verification steps should seek to avoid repeating tests relating to system installation and operation, referencing the supplier test results/reports instead.

3 Regulated Organization and Supplier Relationship

3.1 Introduction

Ultimate responsibility for testing and confirming that the system meets its requirements lies with the regulated organization.

Where the regulated organization is not developing the application themselves, it is likely that they will be acquiring their system or service from one of the following sources:

- Product developer: developing a product or a custom system.
- Systems integrator: implementing a system specifically for the regulated organization and this work forms part of the regulated organization's life cycle. Their supply is likely to be based on commercial off-the-shelf products which they have configured for the particular regulated organization application, although there also may be some custom elements within the supply.
- Service provider: whose services require an element of testing, e.g., cloud computing providers, calibration outsourcing, etc.

The term *supplier* in this Guide encompasses all of the above.

In some cases, a supplier may fulfill more than one role, e.g., being both the product developer and system integrator (developing the regulated organization's application using their own organization's products). It also may be that the supplier is a division within the regulated organization.

The effort required by the regulated organization can be reduced if the relationship between it and the supplier allows for a thorough understanding by both parties as to their requirements on both a commercial and technical basis.

The regulated organization should ensure that the supplier fully understands the user requirements for the system and also how the system design, development, and implementation methods can impact the effort required by both parties. They also should ensure that the commercial aspects are clearly defined and that all the deliverables are defined and agreed in the contract for supply of the system.

The supplier should be prepared to work with the regulated organization to ensure that the supplier not only fully understands the user requirements for the system, but also the processes that will be necessary to verify the system for use. By understanding these processes and working with the regulated organization, significant savings may be achieved

A defined and agreed software development approach will help both parties during the development phase particularly with confirming that the system will meet the needs of the regulated organization. It also may enable the regulated organization to leverage the work conducted by the supplier . The effort saved can be particularly noticeable in the testing phase where, if the testing performed by the supplier has been completed to an acceptable standard and the results made available to the regulated organization, it may not be necessary for the regulated organization to retest those aspects of the system's functionality.

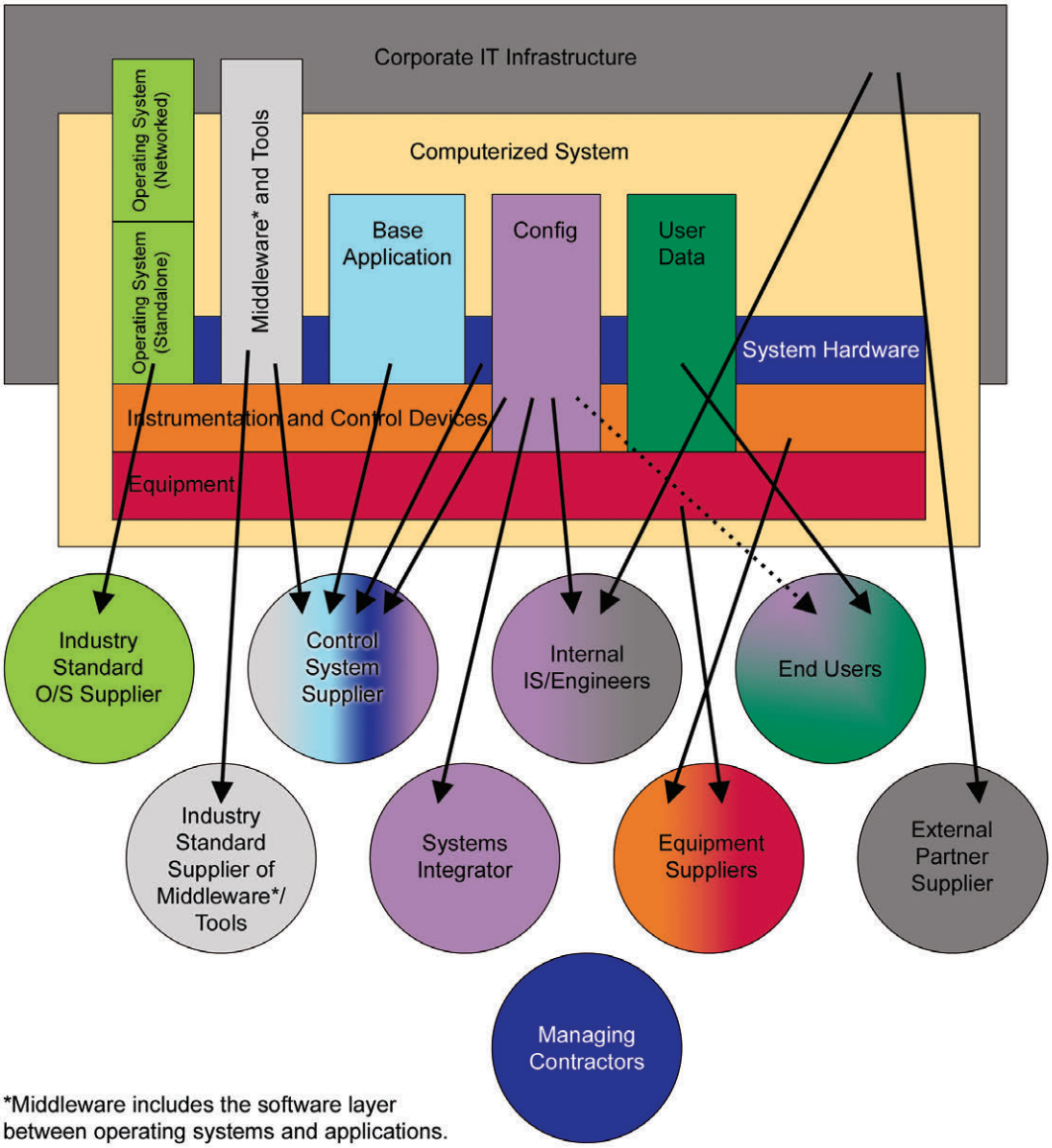
This leveraging of the work done by the supplier can not only save effort, but also can lead to an improved relationship between both parties with potential benefits for both.

The following sections provide guidelines as to what considerations the two parties should give in maximizing the benefits in the area of system testing.

3.2 The Supply Chain

Many suppliers may be involved. Figure 3.1 illustrates examples of the wide variety of potential sources of control system components. Similar supply chain considerations also may apply to other types of systems.

Figure 3.1: Source of Components of a System

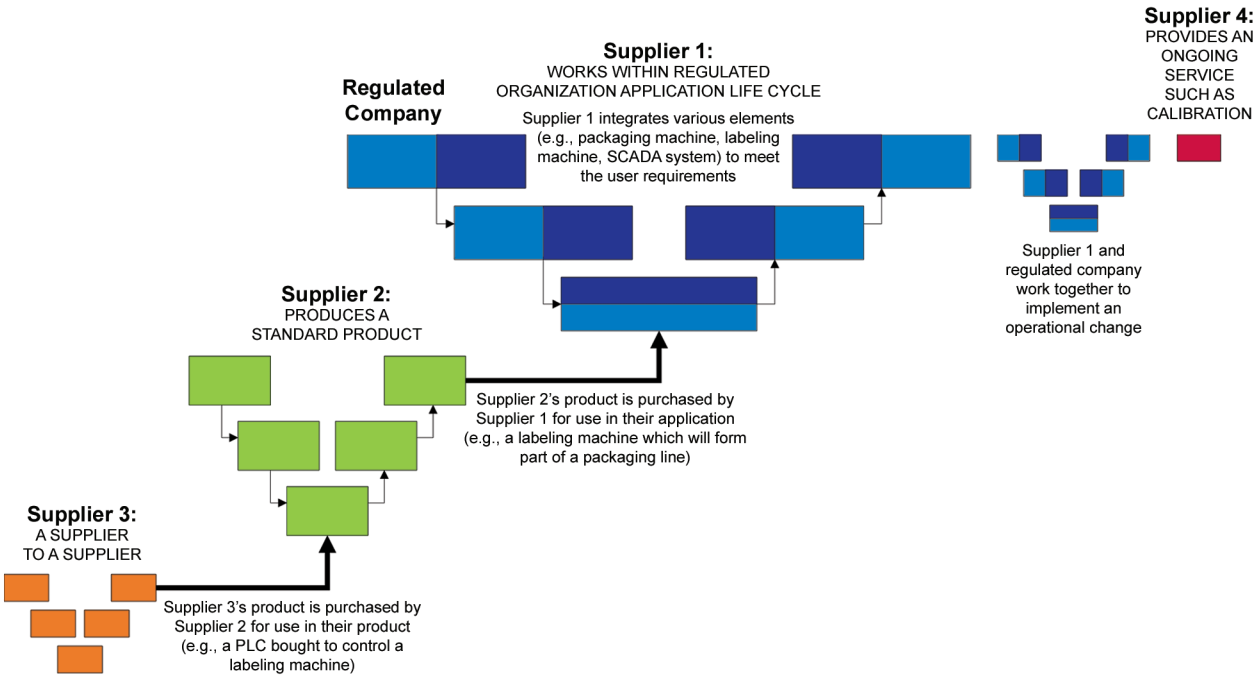


*Middleware includes the software layer between operating systems and applications.

Within the supply chain, there may be suppliers to the regulated organization's supplier. These sub-suppliers should be taken into account when considering the overall supply chain.

Note: that there may be a series of sub-suppliers within the chain.

Figure 3.2: Successive Life Cycle Components of a System



3.3 Quality Risk Management

Table 3.1 includes regulated organization and supplier considerations on quality risk management.

Table 3.1

Regulated Organization Considerations	Supplier Considerations
GAMP® 5, Section 3 Quality Risk Management together with associated Appendices describes a risk management process. The regulated organization should undertake a risk assessment for the overall project. This may highlight some areas within the computer system that need particular attention and this needs to be taken in account by the regulated organization when determining the level of testing required on the computer system.	It is likely that a systems integrator will be involved in regulated organization risk assessment process and in risk-based verification activities within the application life cycle. A supplier of standard product will often take risk-based decisions (e.g., about scope and rigor of verification within own product) based on general marketplace understanding ("typical use") and may need to assume worst case impact for particular functions.

Table 3.1 (continued)

Regulated Organization Considerations	Supplier Considerations
The outcome of risk assessment will determine on which, if any, areas of the system the regulated organization should seek to perform additional testing. These may include those areas of the system that are perceived as: • Being critical to product quality and patient safety • Having a potential impact on data integrity • Complex or having novel features • Having been impacted by project specific configuration • Having a weakness in the supplier development life cycle, e.g., poor testing practices or poor test documentation • Having an unacceptable degree of risk following an assessment of the system and the supplier Having determined these areas, the regulated organization can then plan the testing of the system to focus appropriately.	Suppliers should use their own risk management processes in seeking to leverage the testing already executed by their own supplier(s), or testing conducted by themselves on identical systems or pieces of equipment.
The regulated organization should be wary of relying upon any risk assessment of the level of testing required carried out by the supplier in isolation. It is unlikely that the supplier will fully understand all the regulated organization's processes and all the critical aspects of the system. If the supplier has a thorough knowledge of the regulated organization's processes through a long standing relationship with them and the pharmaceutical market, this may be practical, but care should be exercised by the regulated organization.	Suppliers should be willing to share their "generic marketplace" risk assessment for a product, application, or service so that regulated organizations know the basis from which the assessment results were derived. Any differences between the assumed worst case impact and the actual risk severity associated with the end user application can then be analyzed and controlled.

3.4 Supplier Assessment

In order to decide what level of verification is appropriate, a regulated organization needs to consider the scope and rigor of testing that has already been executed on the functions that are critical to the specific application (and how well it was controlled and documented). This may be achieved through an appropriate type and rigor of supplier assessment. For further information on supplier assessments see Appendix M2 of GAMP® 5 [1].

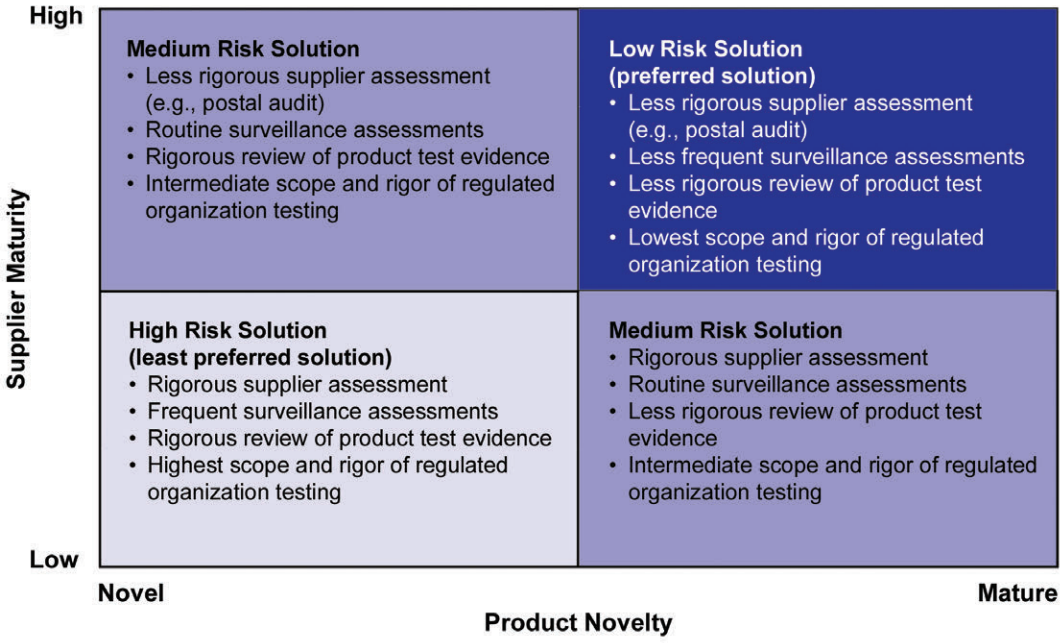
Products that are widely used in pharmaceutical and other highly regulated industries are generally considered to have a lower likelihood of undiscovered software defects than new products or those developed for general markets and used infrequently in such regulated industries.

Suppliers who are experienced in the industry, who implement appropriate quality practices and have appropriate functionality, features, and documentation, will supply products more likely to meet the needs and quality expectations of regulated organizations.

Market review and an initial supplier assessment should establish the relative maturity of a product and its supplier and this should include a consideration of supplier testing. Note this may need to cover more than one product and supplier for a given application and the regulated organization should, if applicable, take note of this throughout the risk management process.

The following diagram gives an example approach to considering the relative risk associated with acquiring a solution (a specific computerized system from a specific supplier), and the level of assessment and review needed to be performed by the regulated organization, based on the maturity of the product and the supplier.

Figure 3.3: Example Supplier and Product Maturity Approach



Supplier maturity refers to regulated industry track record and appropriate quality management system. A mature product will have a history of good product quality with a high level of customer satisfaction in the relevant industry as opposed to novel, relatively unproven product.

Regulated organizations may choose to select products which fall into the High or Medium Risk Solution areas shown in the Figure 3.3, but this may result in an increase in assessment rigor (audit) and/or regulated organization documentation and testing to ensure that the risk is appropriately addressed.

Table 3.2: Specific Considerations for Supplier Assessment

Regulated Organization Considerations	Supplier Considerations
The regulated organization should confirm that the approach to testing being adopted by the supplier is appropriate to the product, application, or service being provided. If the system being supplied comprises a mix of categories, e.g., a largely configurable solution, but having some elements of custom "special to project" software, the regulated organization must take care to ensure that supplier testing is appropriate to the complexity and risk associated with each element and not simply to those categories that form the larger part of the system.	Suppliers should be willing to share the approach used for product development, application development, or service provision, and where necessary, explain how the testing relates to the novelty and complexity of the items being developed.

Table 3.2: Specific Considerations for Supplier Assessment (continued)

Regulated Organization Considerations	Supplier Considerations
The regulated organization should ensure that any sub suppliers to their main supplier have all been considered, risk assessments carried out as necessary, and where required, supplier assessments of appropriate rigor have been conducted. These should have been conducted by the regulated organization's supplier, or in turn their (sub) suppliers with any shortcoming or actions required being clearly documented with appropriate recommendations for closure. If there are any outstanding shortcomings or actions from the risk assessment, these should be minor in nature such that they will easily be closed out by the regulated organization's supplier (or sub supplier). Those with a good QMS are likely to have made this consideration already.	Where the supplier integrates third party software or hardware at any stage in their product development life cycle, they should consider the quality of their own suppliers and their suppliers' products when determining an appropriate level of testing. This Guide provides assistance to regulated organizations in the pharmaceutical industry as to how they should approach the testing of supplied systems. The same approach is recommended for adoption by suppliers when they make use of third party products. Suppliers should be in a position to verify that the products they use have been developed following good practice and that they have taken all possible measures to ensure this. This may involve, but not be limited to: • Assessment of developers of the third party products. This may be restricted to a postal audit, but consideration should be given to carrying out a full audit. • The specific testing of their use of these products, e.g., where specific configurations of automated tools are used, should be tested and documentary evidence provided of fitness for intended use. • Where third party products are considered to be a widely used standard product, suitable evidence should be available to support this statement.

3.5 Leveraging Supplier Testing

The regulated organization should be seeking to leverage supplier documentation and testing. Where the system has been appropriately tested, there is no value in the regulated organization repeating those tests. The need for additional testing should be based on the regulated organization applying a documented quality risk management process to ensure the supplier is adopting good quality practices toward testing.

The following diagram shows the differing levels of testing that the regulated organization should consider undertaking when using products from the preferred and least preferred option from the supplier and product maturity model outlined in the Supplier Assessment section as shown in Figure 3.3.

Figure 3.4: Relative Regulated Organization's Test Burden when using Preferred versus Least Preferred Solutions

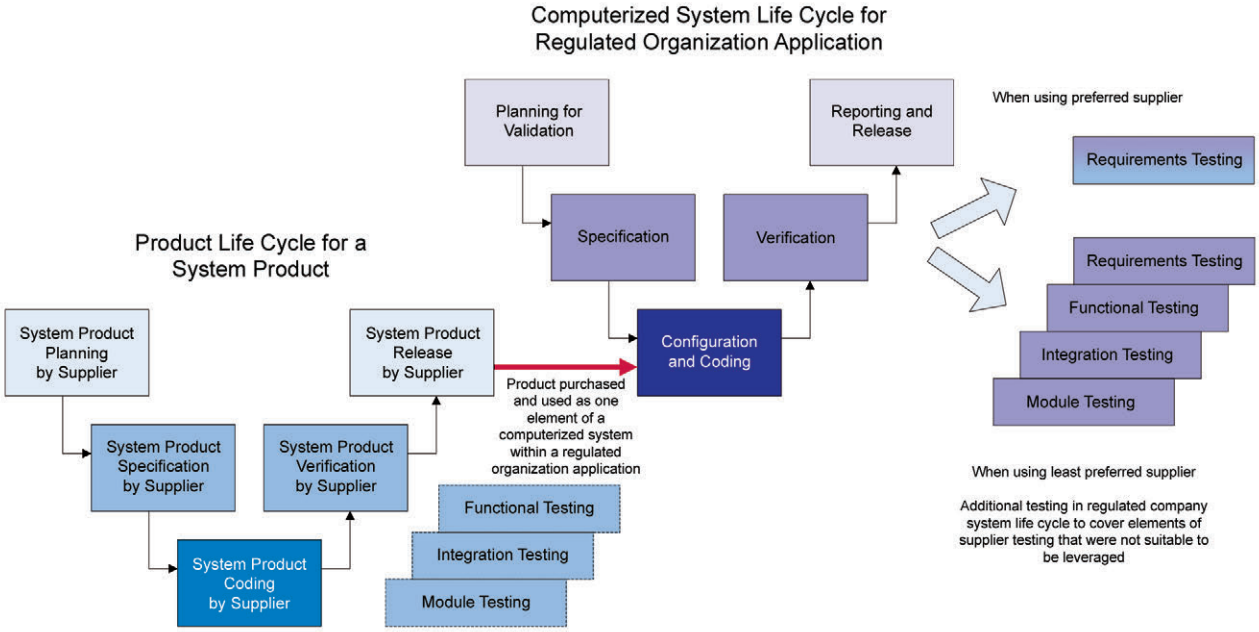


Table 3.3: Considerations for Leveraging Supplier Testing

Regulated Organization Considerations	Supplier Considerations
Supplier testing and associated documentation should be assessed and used if appropriate. The regulated organization can minimize the volume and rigor of testing required by: • Avoiding unnecessary customization, e.g., by modifying the business process, if this is practical, to match an off-the-shelf application • Seeking to leverage the testing already executed by the supplier or possibly by the regulated organization on identical systems or pieces of equipment	A systems integrator will be involved in risk-based verification activities within the application life cycle. A supplier of standard product or services needs to ensure their verification activities are suitable in scope/ rigor and are suitably controlled/documented so they can be leveraged by regulated organization. A supplier of standard product who then also acts as integrator (e.g., configuring each instance of product to meet specific customer requirements) needs to separate out what is "product development" from what is "application" as the verification evidence from the first will need to be made available to all customers (e.g., during supplier audit). The verification results from the second will accompany the delivered end user application.

Table 3.3: Considerations for Leveraging Supplier Testing (continued)

Regulated Organization Considerations	Supplier Considerations
In an ideal situation, regulated organization verification may be optimized at a level which confirms that the system meets their requirements and verifies the adequacy of previous testing. Where the business process and/or functionality are deemed high risk, the regulated organization needs to consider testing additional to that performed by the supplier.	The use of a defined and auditable software development method will potentially allow the regulated organization to make use of the supplier's test results in validating the system for use in operation. This can lead to savings for the regulated organization which may give the supplier a differentiator from other potential system suppliers. Section 7 Supplier Activities of GAMP® 5 [1] gives a good breakdown of what the supplier should be considering when seeking to supply to the life sciences industry. This guide follows the principles given in GAMP® 5 [1], and the methodologies given within this Guide will take a supplier through the testing considerations for systems to be supplied.
Where shortfalls in supplier testing processes are identified, the regulated organization needs to assess the impacts of these gaps and put in place actions to address or control them: • Deficiencies may be addressed in a one-off product or on a project basis as part of a plan of corrective actions agreed between the regulated organization and supplier, and this may include additional testing by the regulated organization. • Regulated organizations should encourage suppliers to address any shortcomings in their testing processes and documentation in a systematic manner. This may be done as part of a program of continuous improvement under the supplier's quality system and may include the introduction of an appropriate Testing Maturity Model. • The regulated organization may consider that where the systems are of a highly critical nature, it may be appropriate that corrective actions to the supplier's quality system have a suitable contractual basis. Where additional regulated organization testing does not appropriately control the risks resulting from inadequate supplier testing, it may be appropriate to consider the selection of alternate products and/or suppliers.	
Where it is not practical to conduct an assessment of the supplier (e.g., open source software or uncooperative suppliers) appropriate risk controls steps will need to be taken. This may include additional detailed testing depending upon the maturity of the product and the supplier or considering the selection of alternative products and/or suppliers.	

3.6 Determining Appropriate Test Evidence

Table 3.4: Considerations for Determining Appropriate Test Evidence

Regulated Organization Considerations	Supplier Considerations
Regulated organizations should define what level of test evidence should be in place before the system can be considered to be fit for intended use.	The supplier should consider what test evidence will be beneficial to the regulated organization's system verification exercise and endeavor to design their testing strategy to assist in this area.
When leveraging test results generated under the supplier's QMS, regulated organizations should ensure that they focus on the coverage and quality of test results rather than on the terminology or formats used. As long as the content and coverage is appropriate and good test documentation practices are followed (detailed elsewhere in this Guide), it is acceptable to leverage test documentation provided in the supplier's standard format.	The supplier may need to consider how these test results are recorded at the various stages of development and at each point of supply of the system. Development testing may be conducted under the supplier's own QMS with its particular methodology while testing at the point of supply, which may be factory and/or site testing, may be under the regulated organization's QMS or the supplier's own. Ensuring that whatever approach is adopted, the test results are of suitable quality for use by the regulated organization can save effort in the verification process.
Regulated organizations should define how long test evidence needs to be available. The value of such test evidence changes over time and the retention period can be determined by risk assessment. For example, the retention of detailed unit test scripts, with the results of the executed scripts, will facilitate subsequent regression testing if the custom software is still subject to change, but once the custom software is no longer subject to modification, the usefulness of retaining the test evidence can be assessed, results documented and appropriate actions taken.	The supplier should consider the period of test result retention and ensure that test results from the product development phase are retained throughout the product's life. A situation may arise when an established product/ supplier is acquired by a new supplier. The new supplier should ensure that relevant test evidence from the previous supplier is secure and available for audit.
Where possible, the regulated organization should build on evidence of testing provided by the supplier and aim not to duplicate test evidence. The ability of regulated organizations to leverage supplier testing may depend on suppliers allowing access to specific test evidence, e.g., during supplier assessment. This may need specific contractual agreement.	Where the supply is within the regulated organization's application life cycle, test evidence will be associated with that life cycle and available to the user as part of the application life cycle documentation. Where the system is to be generic, the supplier should consider what testing they will conduct during product development and what test results they will retain and be able to provide, if required, to the regulated organization as part of the overall supply. Where generic systems may be slightly modified for a given regulated organization, consideration as to how these will be tested and how these test results may be provided to as part of the overall supply will be of assistance to the customer.

3.7 Ongoing Support

Table 3.5: Considerations for Ongoing Support

Regulated Organization Considerations	Supplier Considerations
Service Level Agreements covering not only spares and repairs mechanical aspects, but also backups, system software checks, and performance monitoring will help the regulated organization to ensure that the system remains in good working order.	The supplier should ensure that they have processes in place to provide long term support to the regulated organization. Being able to do this offers the supplier the opportunity to maintain a relationship with the regulated organization with the potential benefits this may bring them.
Service Level Agreements that cover the process for installing upgrades or patches to the system can help define the testing that will be required as part of this process.	Suppliers should consider the need for ongoing support of the system following its delivery and setting to work at the regulated organization's site – including requirements for installation or patches and upgrades and the appropriate level of regression testing.
Patches for fault rectification and/or updates to the system, while they may require some verification activities on the part of the regulated organization, can bring benefits such as improved efficiency, less down time, better user interfaces etc. Having a defined and structured approach to this activity can greatly assist the regulated organization and improve the status of the supplier.	The need for a rationale as to why the patch/update is being provided together with suitable testing evidence appropriate to the risks involved with the change can offer substantial benefits to the regulated organization and help in their decision as to whether to install the change. See Appendix T8 for more detail.

3.8 Commercial Issues

It is important that level of documentation, including design, development, and testing information to be supplied to the regulated organization is agreed at the earliest point in the project as possible, preferably before the contract is finalized.

Testing, particularly where there is any custom development, is often a milestone linked to a stage payment. Key points for success include:

- Agree and document early in the project the stages, scope, and rigor of the testing required – an assessment of the critical functions and risk impact to the regulated organization can assist with this.
- Agree test phasing and acceptance criteria for release, such as allowed levels of residual defects.
- If appropriate, ensure that a non disclosure agreement or similar is in place to enable the supplier to allow access by the regulated organization to sensitive information, test results etc., and vice versa.
- Ensure the test scripts are traceable to the various design specifications particularly those requirements specific to the life sciences market.
- Where appropriate, involve the supplier and regulated organization personnel in the review and approval of test scripts that are relevant to them. This should ensure that all parties understand the test objectives and should limit the effect of subsequent changes.
- Focus on the critical and high risk requirements carrying out full black box and stress testing for these areas and consider structural testing and code reviews if appropriate.

- Reduced testing for the lower risk areas
- Document and agree the configuration management process and regression analysis being followed during the testing activities
- Ensure all equipment, including spare parts required for any disaster recovery testing, are available prior to the commencement of testing
- Ensure that time planned for document reviews factors in the expected size and complexity of the item to be reviewed
- Ensure all personnel are available when required, including system developers in case of a deviation occurring which requires a change to the system
- Prepare contingency and recovery plans

On completion of the testing, agreement on any outstanding actions or deviations should be reached between the supplier and regulated organization in order for the project to progress to the next stage.

Other commercial issues for consideration include:

- The retention period for the test documentation relative to the needs of the system and the business process
- Whether escrow agreements are required and in place for software and documentation
- Whether the regulated organization retains the right to audit supplier either at fixed intervals or in response to particular triggers

4 Appendix T1 – Test Practices

4.1 Introduction

This appendix covers the need for high level test policies within an organization, discusses the different types of verification activity that may be covered by such a policy, and goes on to define how the scope of testing for a particular project may be determined.

4.2 Test Policies

Regulated organizations should have consistent policies for testing computerized systems; particularly for GxP computerized systems. Test policies are usually defined at the corporate level. Policies may be established at site or department level if required by regional needs or to provide applicable procedures for specific types of software or systems. Regulated organizations should define the test policy within the context of their own computerized systems' QMS. The test policy may be included within a specific testing policy document or may be part of a validation policy document.

Test policies, dealing with the high-level approach across all systems, must not be confused with test strategies, which are defined at the system level and consider risks specific to business processes.

Test policies can be influenced by the risk philosophy of an organization, as testing is closely related to the risk management process. For regulated organizations, factors which can influence a corporate risk philosophy include:

- Positive and negative experiences with system suppliers
- The inherent product or process risk (potential for patient harm, poor quality product or loss of data confidentiality and integrity) and the level of product and process understanding within the organization
- Recent and historic regulatory inspection findings
- Corporate reputation in the market

Regulated organizations should seek to maximize supplier involvement throughout the system life cycle in order to leverage knowledge, experience, and documentation, including existing test documentation, to avoid wasted effort and duplication (see GAMP® 5 [1]). However, a risk-averse organization may choose to test lower risk functions with greater rigor, and may be reluctant to leverage supplier testing, preferring to repeat it under their own in-house control.

Key testing principles should be defined in either the test policy or a high level Testing Standard Operating Procedure (SOP). This should provide the flexibility to define how these principles will be applied to different types of system or within different life cycles. Specific SOPs or work instructions (which may be system, project, or model specific) should provide the detail of how these principles are applied. The test types needed will come out of the risk assessment, and what phases they will be done in will fit with where it is most efficient to do them.

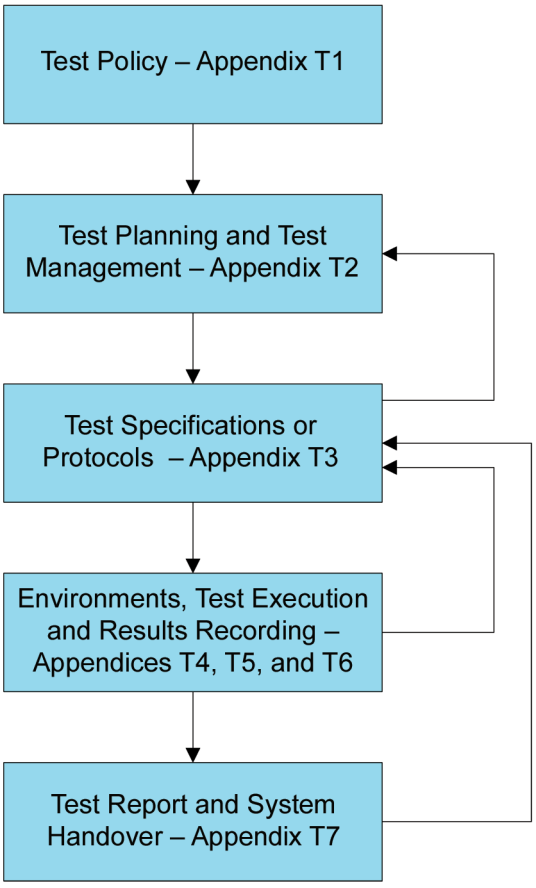
Specified procedures for testing should help to provide consistency during execution of the testing process. These procedures should be based upon a common testing policy (which should apply to both manual and automated testing). A common testing policy may provide benefits, including:

- Common methodologies being adopted
- Common terminology (and less potential for confusion)

- Common test documentation models
- Common risk management and analysis processes and documentation standards
- Use of common test documentation templates
- Provision for the use of computer based testing tools (see Appendix T11)
- Similarity in approach means testers are familiar with the process so reducing the duration of the learning curve
- Consistent testing processes throughout the organization
- Formal and well understood practice for efficiently managing defects, including defect resolution
- Faster and more consistent reviews

The test policy may need to address particular requirements for the situation where a system is being delivered in stages to ensure that changes made as a result of test incidents in one environment are promoted and backtracked as necessary into earlier or later test environments and the operational system.

Figure 4.1: Overview of Testing Process



4.2.1 Additional Considerations for Non-Linear Development Methods

Testing in non-linear development methods is discussed in detail in Appendix T10.

Key differences associated with non-linear development which should be addressed by a test policy include:

- The delivery of the system is usually in more than one stage, referred to as an increment or phase.
- Each iteration of an increment/phase includes cycles of:
  - Design
  - Development
  - Testing
  - Review
- Design and test documentation is not formally approved until the end of an increment/phase.

Where a prototype is to be further developed (an “evolutionary” prototype rather than a “throw away” prototype), the test policy should be clearly defined and should consider how the prototype will be baselined and tested before becoming part of the main development life cycle.

Points to address the risks of non-linear development life cycles for consideration in a test policy include:

- Definition of measures during an iteration to ensure that the resulting application is fit for purpose. These may include:
  - Regular regression testing (the use of automated test tools for this is a considerable advantage – see Appendix T11)
  - Source code reviews
  - Regular reviews of draft test scripts to ensure adequate test coverage
  - Maintenance of traceability
- Definition of periods or milestones when development documentation should be updated to reflect the current use of the system
- Definition of a formal test phase prior to deployment of the iteration which verifies the critical functions by execution of formally approved test scripts against the finalized and approved design
- Requirements added, removed, or modified can cause a need for requirements and impact analysis to ensure requirements are still consistent and do not contain non-conformities.

4.3 Types of Verification

This section discusses the types of tests that can be performed as part of product development and project testing.

The types of test that are required, and the scope and rigor of each type of test, are determined in part by risk assessment.

Some types of test may be included in more than one test phase, e.g., functional tests can occur in both integration testing and user acceptance testing.

During the development of a project/system specific test strategy or test plan, each type of test should be considered in the context of the risks assessed and the test objectives, e.g.:

- Is positive case testing sufficient to demonstrate correct functioning of a low risk Category 3 system?

- Is it necessary to demonstrate the effective mitigation of high risks within a software Category 5 system using challenge, stress, and positive and negative case functional testing?

Each strategy or test plan should clearly define which types of tests are applicable to each test phase.

Tests may be automated using computerized test tools; this can benefit functional testing (especially regression testing) and performance (load) testing.

4.3.1 Static Testing

Static tests can be used to capture early coding errors and issues. Static tests include:

- **Design Reviews** (outside the scope of this Guide)

- **Source Code Review** – this provides a means for documenting the structural verification of a custom coded module. Source code review is normally performed before the start of formal module testing. It should include review against the:

  - Required coding standards

  - Design requirements

- **Static Analysis Tools** – these are commercially available test tools which evaluate a program in a non-runtime environment, e.g., to detect coding errors which may result in defects such as buffer overruns and resource leaks.

- **Configuration Testing** – for a configurable system, testing should verify that a package has been configured in accordance with the specification. This could take the form of inspections or a check of supplier documentation.

4.3.2 Structural Testing

4.3.2.1 Test Objective

The objective of structural testing (or white-box testing) is to ensure that each program statement performs its intended function. Structural testing identifies test cases, based on knowledge of the source code, detailed design specification, and other development documents. These test cases should challenge the control decisions made by the program and the program's data structures, including any configuration settings. Structural testing also can identify dead code that is never executed when a program is run.

Structural testing is recommended for requirements identified as having a high-risk priority (in addition to functional testing) because testing of all functionality defined by the user requirements does not mean that all software code has been tested.

In cases where a system is customized from Commercial-Off-The-Shelf (COTS) software, this type of testing can be employed on customizations. Customizations may include a program written for an interface or code written to follow a different workflow in the system. Structural testing requires careful planning to achieve complete coverage (see Section 4.3.2.2 of this appendix) efficiently; however, once structural tests have been developed, they may be automated for subsequent regression testing.

4.3.2.2 Test Scope

The scope of structural testing should reflect the risk priority associated with a system or function. Structural test coverage may include:

- **Statement Coverage** – this criterion requires sufficient test cases to ensure each program statement is executed at least once; however, its achievement is insufficient to provide confidence in a software product's behavior.

- **Decision (Branch) Coverage** – this criterion requires sufficient test cases to ensure each program decision or branch is executed so that each possible outcome occurs at least once. It is considered to be a minimum level of coverage for most software products, but decision coverage alone is insufficient for high-integrity applications.

- **Condition Coverage** – this criterion requires sufficient test cases to ensure each condition in a program is executed, to test all possible outcomes at least once. It differs from branch coverage only when multiple conditions should be evaluated to reach a decision.

- **Multi-Condition Coverage** – this criterion requires sufficient test cases to exercise all possible combinations of conditions in a program decision.

- **Loop Coverage** – this criterion requires sufficient test cases for all program loops to be executed for zero, one, two, and many iterations, covering initialization, typical running, and termination (boundary) conditions.

- **Path Coverage** – this criterion requires sufficient test cases to ensure that each feasible path, from start to exit of a defined program segment, is executed at least once. As a result of the very large number of possible paths through a software program, complete path coverage is generally not achievable. The scope of path coverage is normally established based on the risk priority of the software under test.

- **Data Flow Coverage** – this criterion requires sufficient test cases to ensure that each feasible data flow is executed at least once. A number of data flow testing strategies are available [9].

The style of structural testing may depend on the type of algorithm/program under test. For example:

- Testing of a module implemented as flowchart style logic may need to focus on coverage of statements that make up each process within the flowchart and coverage of branches created by each decision.

- Testing of a module implemented as a state transition diagram may need to focus on coverage of the statements/ actions which are supposed to happen in each state and of branches created by each possible transition criterion.

- Testing of a recursive or iterative algorithm may need (in addition to checking the logic carried out within each iteration and the correct passing of data from one iteration to the next) to focus on areas such as correct interpretation of the finish condition and exit from the recursion/iteration, correct behavior when a boundary condition is encountered, whether there is a possibility to end up in an endless loop and whether there is the possibility that a local optimal solution may be found and the algorithm exit without finding the correct global solution.

- Testing of a program split into modules or objects may need (in addition to checking the logic carried out within each module) to focus on data and messages passed between these modules or objects.

When executing business process based testing of COTS applications, the principles of structural testing also can be used to ensure that every process path (i.e., normal, alternate, and exception) is tested and that every step in the process (i.e., every system transaction) is tested. Note: if the required functionality matches that provided as standard by the COTS package, this level of coverage may already have been addressed in full by the supplier and it may be possible to leverage the supplier's results rather than repeat test effort.

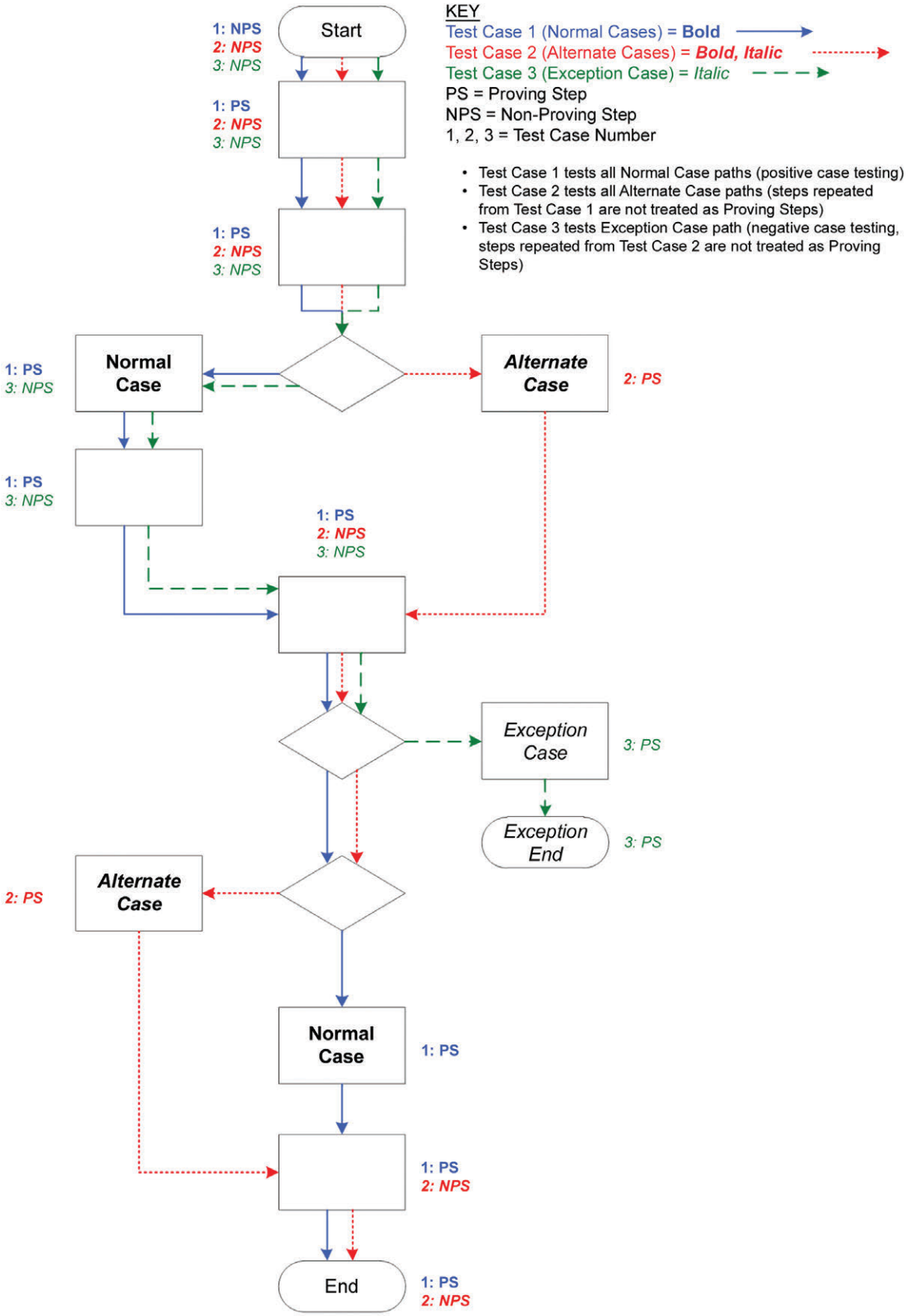
Complete coverage testing requires a careful analysis of the software design to determine for the branches:

- Normal case (the expected path through the code)
- Alternate case (an unusual, but anticipated path through the code)
- Exception case (an exceptional or error condition)

In many cases, careful analysis of the software design means that it is possible to test a number of paths/branches with a limited number of test cases, i.e., multiple paths can be tested by a single test case. In other cases, steps that need to be executed to reach a specific branch can be treated as non-proving steps for that test case if they were treated as proving steps in an earlier test case.

An illustrative example of this approach is shown in Figure 4.2. It applies the example test step classification scheme described in Appendix T1.

Figure 4.2: Branch and Path Coverage by Test Cases



This example shows that:

- Test Case 1 executes all normal case paths. The Start step is a non-proving (set-up) step, but all other steps are proving steps, see Appendix T3. Note that this single test case exercises two normal case paths.

- Test Case 2 executes all alternate paths and the single test case exercises two alternative case paths. The initial steps executed in Test Case 1 should be repeated in order to arrive at the branches at which the alternate cases are tested, but because these initial steps were proved during Test Case 1, they are considered to be non-proving steps for Test Case 2.

- Test Case 3 executes the single exception case path. The initial steps executed in Test Case 1 should be repeated in order to arrive at the branch at which the exception case is tested, but because these initial steps were proved during Test Case 1, they are considered to be non-proving steps for Test Case 3. Only the steps that test the exception case are considered to be proving steps for Test Case 3.

In this example, it is also possible to define the repeated steps from Test Case 1 as test pre-requisites or set-up steps for Test Case 2 and 3 and thereby reduce the size/complexity of these latter test cases.

Using this approach, it is possible to assure complete branch coverage testing with an optimal number of test cases, but this requires careful analysis and detailed test planning.

4.3.2.3 Test Positioning within the Life Cycle

Structural testing is performed primarily within the unit or module test phase.

4.3.3 Functional Testing

4.3.3.1 Test Objective

The objective of functional testing (or black-box testing) is to evaluate the compliance of a system or component with specified functional requirements. Functional testing therefore identifies test cases based on the definition of what the software is intended to do. These test cases challenge the intended use or functionality of a program, and the program's internal and external interfaces.

Functional testing is required in addition to structural testing because testing of all of a program's code does not necessarily mean that all required functionality is present in the program.

Functional testing also requires careful planning to achieve complete coverage efficiently. Significant benefit can be gained by automating functional tests for subsequent regression testing, but not all functional tests lend themselves to automation. The automation of invalid case tests requires specific attention to the design of the automated scripts.

4.3.3.2 Test Scope

Functional testing usually should cover all specified functional requirements. For a particular requirement; however, the number and types of functional tests performed may reflect the risk priority associated with the system or function.

Note: although a project may include a test cycle called functional testing (see E Appendices for examples), from the perspective of test types, functional testing may include:

- Black box testing of individual software objects (e.g., conducted by a regulated organization, following white box structural testing by a supplier)

- Coverage of functional requirements in an appropriate test cycle (e.g., conducted by a supplier and/or regulated organization in test cycles typically called system test, functional test, or operational qualification)

- Coverage of user requirements in an appropriate test cycle (e.g., conducted by the regulated organization in a test cycle typically called user acceptance testing or performance qualification)

Some common types of functional tests include:

- **Normal Case (Positive Case) Testing** – testing to show that a system does what it is supposed to do in response to the usual and expected inputs (e.g., checking that a calculation gives the correct result in response to expected inputs). By itself, normal case testing does not provide sufficient confidence in the dependability of a software product.

- **Invalid Case (Negative Case) Testing** – testing to show that a system does what it is supposed to do in response to specified invalid inputs (e.g., giving the correct error message in response to an out-of-range input).

- **Business Process Testing** – testing, possibly using a test harness, to challenge the different branches and paths of the business workflow.

- **Special Case Testing** – testing to show that a system does what it is supposed to do in response to inputs at the limit of the permitted domain (boundary or limit condition testing) or to inputs which form a special case or exception (e.g., checking that a calculation produces the correct result for the maximum and minimum values of each input, or checking that a zero input is handled without leading to a “divide by zero” error).

- **Output Testing** – choosing test inputs to ensure that all software outputs are generated at least once during testing (and if relevant that the outputs are exercised at the limits of their allowed range).

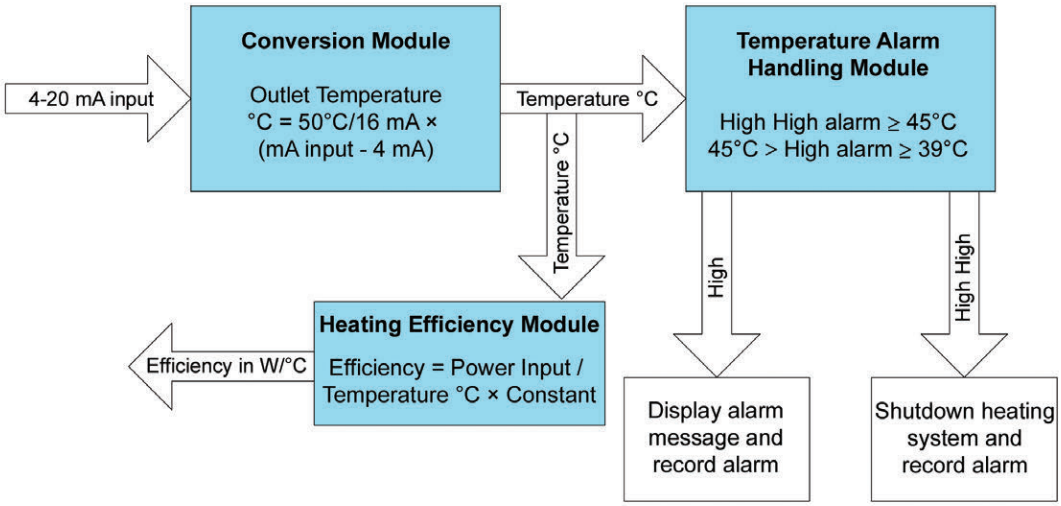
- **Input Combination Testing** – testing combinations of inputs to ensure correct outputs. The input combinations can be selected at random from the possible range of inputs or selected specifically because they are considered likely to reveal faults.

An example of this is shown in Table 4.1.

Table 4.1: Example Extract from Risk Assessment

Risk No.	FS ref.	Potential Failure Mode	Potential Effect of Failure	Severity	Probability	Detectability	Risk Priority	Required Action
41	6.12.5	Temperature incorrectly calculated	M	M	M	L	H	Test temperature conversion limits
42	6.12.5	Temperature exceeds upper limit	M	M	M	M	M	Test High alarm and normal operating condition
43	6.12.5	Aggregates form – drug becomes toxic	H	M	M	M	H	Test High alarm boundaries

Figure 4.3: Process Control Modules under Test



By injecting the input values listed below using a loop calibrator or other simulator, with power input artificially fixed at 100W for simplicity, output testing can be achieved by a combination of negative and positive case testing, to verify the functionality of the temperature, temperature alarm, and efficiency modules.

Table 4.2

Input	Temperature Displayed	Efficiency Displayed	Expected System Response
0	Error	Error	Wire broken alert message displayed and recorded
4.00 mA	0.0°C	Error	Unable to calculate parameter message displayed and recorded
15.84 mA	37.0°C	2.70	None
16.48 mA	39.0°C	2.56	High temperature alarm displayed and recorded
18.37 mA	44.9°C	2.22	High temperature alarm displayed and recorded
18.40 mA	45.0°C	2.22	Heating system shutdown and alarm recorded
19.97 mA	49.9°C	2.00	Heating system shutdown and alarm recorded
20.00 mA	50.0°C	2.00	Heating system shutdown and alarm recorded
20.01 mA	Error	Error	Out of range error displayed and recorded

4.3.3.3 Test Positioning within the Life Cycle

Functional testing may be performed during all phases of software testing, from unit or module testing to system level testing.

4.3.4 Performance Testing

4.3.4.1 Test Objective

The objective of performance testing is to evaluate the compliance of a system or component with specified performance requirements. These may include non-functional user requirements (e.g., speed of response to operator input).

4.3.4.2 Test Scope

Performance testing should normally cover all stated performance requirements. Use of automated performance (load) test tools which can simulate the load usually generated by a significant number of users and which also can measure and monitor system performance in real time can be beneficial. For a specific requirement, the number and type of performance tests executed may reflect the risk priority associated with the system or function. Types of performance tests include:

- **Environmental Tests** – testing to show that a system is capable of operating reliably in the specified environment (e.g., under specified temperature conditions). Testing performed by the supplier is normally leveraged. Where the operating environment falls outside the supplier's specification for the product, additional testing may be necessary.

- **Accuracy Tests** – testing to show that the system is capable of meeting the required accuracy of measurement or control (e.g., controlling temperature to within a specified range).

- **Repeatability Tests** – testing to show that a system is capable of repeatedly meeting the required performance (e.g., by running repeated trials using the same recipe to check that the product is always within specification).

- **Timing or Response Tests (also known as Latency Tests)** – testing to show that a system is capable of meeting the required timing, throughput or response (e.g., responding to operator requests within a specified period).

- **Load Tests or Capacity Tests** – testing to show that a system is capable of meeting the required performance while operating under realistic high load conditions (e.g., with a high number of concurrent users of a database). Load testing can be a complex area. For further information, see Section 4.3.4.4 of this appendix.

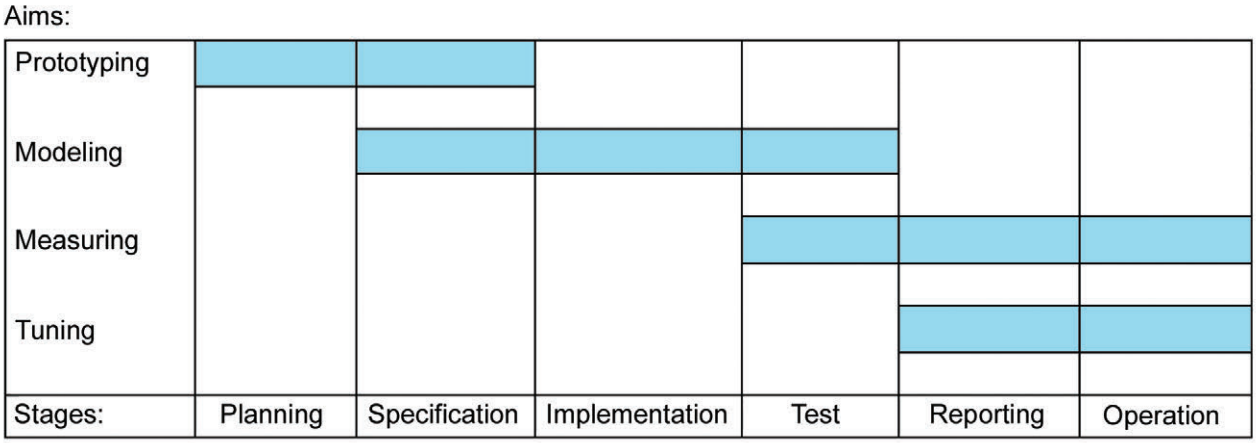
- **Usability Tests** – testing to evaluate the combined performance of a system and user (e.g., checking that the user is able to access and respond to information in a timely fashion via a menu system). Usability testing may be either structured (e.g., to check every menu option in sequence) or unstructured (which allows the operators to "play" with the system in a realistic manner, in order to detect any issues which may not arise through the use of formally scripted tests).

4.3.4.3 Test Positioning within the Life Cycle

Performance testing is usually performed during the factory and site acceptance test phases or before operational release. Performance testing also may be used to verify that hardware meets user requirements. Where possible, performance tests should be built into to earlier stages of testing to help to detect performance issues such as memory leaks. It may be possible to assess performance at an earlier stage using prototypes or theoretical models or by scaling up results from unit or module test phases. Where differences exist between the test environment and the operational environment, it also may be necessary to perform some performance monitoring and tuning within the operational environment.

Figure 4.4 provides an example of different stages where performance testing may be conducted using one or more of the types above. This shows that performance testing can be used to assure the robust performance of a system at various stages within a project.

Figure 4.4: Performance Testing in System Life Cycle



4.3.4.4 Load Testing Considerations

The goal of load testing is to show that a system works as expected at specified operating limits (within the standard range of operation). During load testing, the system is operated under high load conditions. In addition, load testing can be used to prove the design of the software architecture and to support operations throughout the life of a system through scalability or load balancing.

Where it is necessary to conduct load testing on systems with a large number of users, the use of automated performance test tools (load generation tools) can be advantageous in terms of reducing the number of staff required to generate the necessary load profile. This can be particularly beneficial in reducing the time required for full scope regression testing of web-based applications.

Potential issues with load testing include:

- Potential system limits being difficult to foresee during system development
- Costs for load testing being relatively high due to resource-consuming processing
- The generation of data for high volume tests

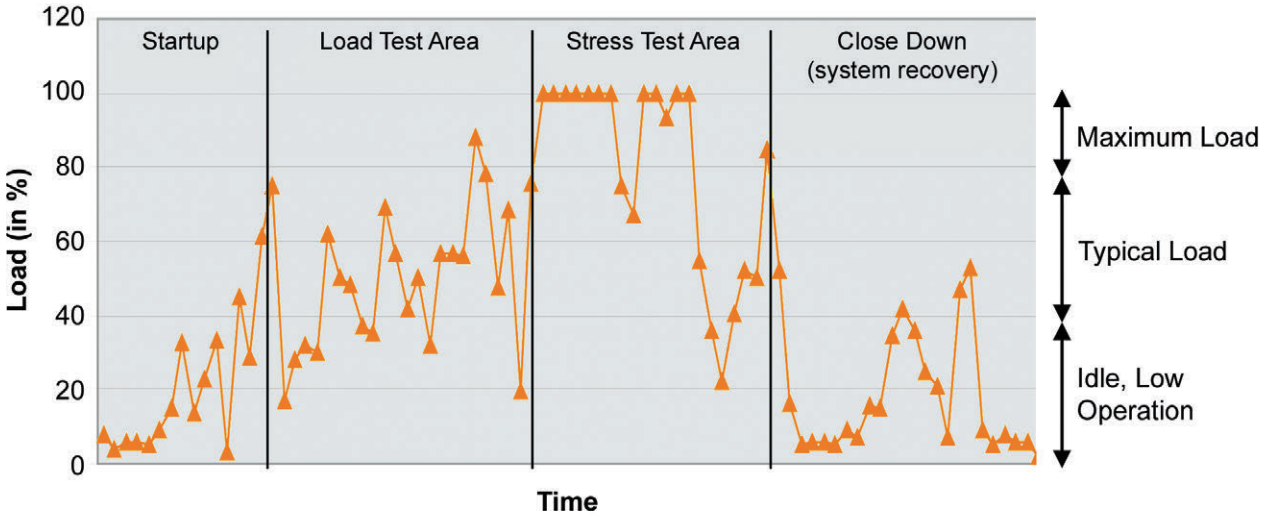
In addition, a proper definition of the expected load may not be available or may be unrealistic.

Typical problems that are detected during load testing include:

- Data transfer fails due to long processing times
- Memory leaks
- Long response times due to large number of concurrent users
- Poor response times due to heavy load on the system caused by audit trail generation on databases or software implemented encryption algorithms
- Scalability of an architecture is limited by network bandwidth
- Maintenance periods exceeded due to long batch processing runtimes

The phases and the load profile of a typical load and stress test (see Section 4.3.5 on Challenge Testing for details of stress testing) are shown in Figure 4.5.

Figure 4.5: Load and Stress Test Profile



The load test in this case addressed high volume tests specified to use between 40% and 80% of the available system capacity. A stress test then followed where an attempt was made to operate at maximum load.

In general, the more complex a system and the more data volume processed, the more difficult it is to predict the behavior of a system under load. Where system behavior is difficult to predict, formal load testing is recommended.

Load testing requires clear definition of expected operating conditions that define the normal operating condition. User requirements related to the expected behavior of the system should be available. These non-functional requirements should be measurable. An example is shown in Table 4.3.

Table 4.3: Examples of Loading Requirements

Metric	Performance Counter/Units	Definition/Example
Data throughput	Bits/second	Data transfer, bandwidth
Data volume	Mbyte, GByte	Database size, transferred data
Transaction rate	Transactions/second	Database transactions
Requests rate	Requests/second	Application server load, web server (hits per seconds)
Response time	Seconds	Expected average system response time to a user request
Number of concurrent users	Number	Number of users concurrently using a system, e.g., accessing a server in parallel
Processing time	Seconds	Allowed runtime for batch processes, e.g., report generation time

Load test scenarios should be based on critical business processes such as:

- Data migration, e.g., of production databases. For further information on testing data migration, see Appendix T9.
- Concurrent user system access, e.g., production start-up or month end accounting runs
- Batch processes, e.g., generation of reports
- Maintenance operations, e.g., database backup

It may be beneficial to measure system parameters during a load test, as well as measuring the performance against stated requirements. An example is shown in Table 4.4.

Table 4.4: Examples of System Parameters to Monitor during Load Testing

System Property	Performance Counter/ Units	Definition
Memory	Available Bytes/Bytes	The amount of physical/virtual memory in bytes available to processes running on the computer.
Disc I/O	Disk Read (write)/sec	The rate of read operations on the disk, e.g., database cache writes
	% Disk Time	The percentages of elapsed time that the selected disk drive is busy servicing read or write requests.
	Avg. Disk Queue Length	The average numbers of both read and write requests that were queued for the selected disk during the sample interval.
CPU	% Processor Time	The percentage of time that the processor is executing a non-idle thread. This counter is a primary indicator of processor activity doing useful work.
	Processor Queue Length	The number of threads in the processor queue. There is a single queue for processor time even on computers with multiple processors. A sustained processor queue of greater than two threads generally indicates processor congestion.
Network	% Network Utilization	Percentage of network bandwidth in use on a network segment.
	Bytes Sent (received)/sec	The rate at which bytes are sent (received) on the interface, including framing characters.
Database	Number of Commits and Rollbacks/sec	Rate of database transactions commits and rollbacks.
	Avg. Time per Transaction	Average time per database transaction.
	Number of Cache Misses	Measures the efficiency of data access and storage.
	Size in GByte	The size and the growth of the database, e.g., physical available and allocated table size.
	Database Locks	Number of tables or data locked due to modification of data.
	Number of Active Database Sessions	Measures concurrent access onto database.

Some parameters may be available from within a system's normal diagnostic information. For other parameters, monitoring tools may be available from the hardware or operating system supplier or from a third party. The measurement of response times can be achieved by using a clock or more precisely by using specific load testing tools. Load test tool manufacturers may provide system monitoring tools and result evaluation modules. Load generators may be needed when test data is required to fully test the system.

For complex systems, a graphical representation that correlates actual load to the response of the system may be used to evaluate measured results. See BS ISO/IEC 14756:1999 [9].

An example of a load test script is given in Table 4.5.

Table 4.5: Excerpt from Typical Load Test Script (Manual Testing)

No.	Test step/ Activity	Expected result	Actual Result	Pass/Fail	Date
1	Access to system with 10 concurrent users executing mix of business operations: • 30% idle (holding connections) • 30% executing goods receipt • 40% modifying master data with audit trail Start-up 2 users in parallel until 10 users are operating, repeat 10 times, wait 15 seconds between each user action.	90% of response times should be below 20s. 100% of goods receipt workflow should be completed within 10 minutes. 10 users have active sessions in the system.	The actual response times are _____. See measured values and evaluation in attachment _____. The total time for the business processes are _____. See measured values and evaluation in attachment _____. _____ users are active in the system.		
2	Perform migration of production database of size 900 Mbyte following migration procedure.	The database migration should be completed within 24 hours.	The database migration was not completed OR The database migration was completed after _____ hours		

When dealing with web-based applications, load testing is further complicated, as the capacity of the application may be difficult to simulate, e.g., cloud computing, where the server is entirely outside of the scope of the regulated organization. For further information on cloud computing implications, see Appendix E2.

Table 4.6 example covers the same load test example as Table 4.5, but with 100 concurrent users.

Table 4.6: Example of Loading Testing Using an Automated Test Tool

Phase	Steps
Create Test Scripts	1. Clearly identify the scenario for which the test script is to be created.
	2. For the manual test script scenario given in 4.5, three automated test scripts can be created as follows: a. Create a script to login, and wait for a defined amount of time, for example 10 to 20 minutes and then logout. b. Create a script to login, continue creating goods receipts and then logout. c. Create a script to login, search for some records, edit and repeat the process and then finally log out.
	3. For any of the scripts above, follow the steps: a. Do a manual walkthrough of the scenario to be automated. b. Open the automated tool and record the scenario. For example, at the start of recording once the application is launched, start a transaction called “Login,” then enter the username and password and hit Login. Once the home page of the application appears, end the transaction. Then start another transaction, say “Create Goods Receipt” and start entering info to create a goods receipt. Once complete, end the transaction. c. Similarly, create as many transactions as needed. d. Once the recording is stopped, the tool will generate the lines of code for the test script. e. Save the script, and re-run the script. This would make sure that the script is working and is correctly creating transactions or activities that have been recorded. f. Now auto-correlation needs to be done, so that the script can be executed any number of times. Sometime auto-correlation will not fix all the problems and it will be necessary to manually correlate the script. g. Based on the type of script and application, it may be necessary to parameterize the script so that when the script is executed for real, the script would dynamically change the data it uses to run the script.
Simulate the load and perform the performance test.	1. Identify the type of test and set the required performance levels. For the scenario below, you may want to add a constraint that the transaction to “Create Goods Receipt” should not exceed 10 minutes.
	2. Identify the load generators. Based on the geographical locations in which the load is to be simulated, it may be necessary to have Load Generator agents installed in remote machines to identify latency.
	3. Add monitors, like the App Servers, Databases, Network Sniffers and any other servers that may need to be monitored that are related to the application architecture or usage.
	4. Add all the automated test tool scripts to the scenario.
	5. Based on the requirements, assign the total number of users to each of the test scripts. For the example, if the tool is to simulate the load for 100 users, it could have 30 users for the first script, 30 users for the second script and 40 users for the third script.
	6. For each of the scripts and users, assign the Load Generator machines, so that the load can be simulated from that geographical location.

Table 4.6: Example of Loading Testing Using an Automated Test Tool (continued)

Phase	Steps
Simulate the load and perform the performance test. (continued)	7. Identify and set the Ramp Up and Ramp Down strategy to the scenario. Ramp Up would mean prioritizing the scripts, and how many users need to start initially and at what intervals. Once defined, the next set of users should start. For example, the tool could start with 2 users from the first script and every 5 seconds another 2 users would start until all users are on the system. Similarly, Ramp Down will tell when to stop the scripts, and how each script and user need to log out. Say, 5 users should log out at a time and after every 10 seconds, another set of users would log out and so on.
	8. After all the settings and durations are set, run the scenario.
	9. Ensure the results are stored in a secure location.
Generate reports based on the saved execution results.	1. Open the saved raw results.
	2. Run the analysis which will analyze the results and gives the options for generating the report. Note, sometime this process can take hours depending on the amount of data being analyzed.
	3. Identify the type of graphs and/or charts that are need as part of the report for the various servers monitored, and also specify the metrics that are important and applicable for the application to be reported.
	4. Generate the report, which will provide the average transaction time and many other metrics, and compare to the desired required performance levels. Based on the time taken for any transaction, the developers or administrators can go back to their code or servers to optimize the application performance.
	5. Re-run the scenario, capture results, generate reports and identify improvements.

4.3.5 Challenge Testing

4.3.5.1 Test Objective

The objective of challenge testing is to evaluate the robustness of a system or component under abnormal operating conditions.

Challenge tests may be automated, but specific consideration should be given to ensuring that the automated test continues to run once the condition being challenged has been tested, i.e., the automated script should continue to run even when expected error messages or faults occur. This requires detailed design of the automated script including steps to allow the system under test to recover and automated testing to continue.

4.3.5.2 Test Scope

The scope of challenge testing should reflect the risk priority associated with the system or function. Types of challenge test coverage include:

- **Data Validity Tests** – testing to show that a system is capable of avoiding or detecting invalid inputs (e.g., operator entries in the wrong format, values outside the allowed range).

- **Security Tests** – testing to show that a system offers protection against control actions or data access by unauthorized users, and to verify the correct configuration of roles and permissions. These may include:

  - User Access Level

- User Management Testing
- Role Based (Authentication) Privacy Testing
- Security of Backup and Recovery
- Security of Media Handling and Exchange
- System Access Monitoring and Reporting

- **Fault Tolerance Tests** – testing to evaluate a system's ability to continue operation in the presence of faults and to recover when the fault condition clears (e.g., the effect of network failure, power down of one hardware item, printer failure, simulation of hardware errors, such as hard disk full, recovery of deleted data or configurations from the archive or backup location).

- **Stress Tests** – testing to evaluate the system's ability to continue operation under abnormally high load conditions and to recover when the condition clears (e.g., with many concurrent users of a database or abnormally high network traffic). Stress testing can be a complex area and further discussion is given in section 4.3.5.4 of this appendix.

- **Environmental Tests** – testing to evaluate the system's ability to continue operation under extreme environmental conditions.

4.3.5.3 Test Positioning within the Life Cycle

Challenge testing is carried out during all phases of software testing, from unit or module testing to system level testing.

4.3.5.4 Stress Testing Considerations

The goals of stress testing include:

- Evaluating a system's ability to continue operation under abnormally high load conditions and to recover when the condition clears.
- Providing an indication of the scalability of a system and to provide an indication of the point at which a growth in anticipated load can no longer be supported by the current system.
- Providing benchmarks that can be used to set alert limits during operational phase performance monitoring.

In addition, stress testing should prove that a system recovers from extreme situations and works as specified afterward.

Challenges associated with stress testing include:

- Potential limits and stress situations being difficult to foresee during system development.
- Systems may not work as specified beyond expected operating limits or do not recover following stress testing.
- Damage to a system may occur during stress testing.
- Producing extreme situations for test execution can be difficult and resource consuming.

Typical problems detected during stress testing include:

- Instability and system failures due to system overload.
- System crashes due to lost connections or incorrectly terminated processes.
- Denial of system access due to an excessive number of access requests.
- Infrastructure capacity limitations (run out of memory, high processor load, bandwidth exhausted, running out of possible network connections).

In general, the more complex a system, the more difficult it is to predict the behavior of the system under extreme situations. Stress testing is recommended especially for systems which are designed for managing extreme situations (including safety critical systems).

Many aspects of performing stress tests are analogous to load tests (see Section 4.3.4.4 of this appendix); considerations include:

- The system and its environment can become unstable or even unusable after stress testing. A recovery plan should be prepared for the test environment including backup and restore of test databases. During test execution, it may be necessary to protect the environment and personnel involved in the test, e.g., if a production machine is included in the stress test.
- Stress testing should be executed in a protected and isolated test environment which is fully representative of the operational environment. The stress test environment should be separated from the functional test environment, either physically or in time, to avoid disruption. In particular, critical data required to document test execution and problem analysis should reside in a secure environment, but not on the system under test, because data could be lost or may not be accessible during or after stress testing.

4.3.6 Other Testing

Other testing is performed for specific purposes as part of the development process or within later phases of the system life cycle. This testing may include any of the test types defined in this appendix.

4.3.6.1 Conference Room Pilots

Conference room pilots comprise a combination of usability testing and a walkthroughwalkthrough of the software with the system owner and/or end user. They do not typically form part of formal testing even though they often involve an amount of informal (undocumented) testing. Conference room pilots should be regarded as a means of verifying design requirements and of building confidence before formal (documented) testing.

4.3.6.2 Ad hoc or Exploratory Testing

Ad hoc or exploratory testing involves simultaneous test design, execution, and learning. Typically ad hoc testing is conducted on a system against the design specifications, but the tests are not scripted in advance and the results are not documented. Testers usually use their intuition and testing experience to challenge the system functionality.

This form of testing can be beneficial for discovering system defects that more formal scripts will not detect during early system evaluation. It is a supplement to, (but not a replacement for) formal testing.

4.3.6.3 Interoperability Testing

Interoperability testing is used to verify that a system is functional across different hardware and infrastructure configurations, e.g., a web-based application will function across different combinations of browsers and operating systems.

4.3.6.4 Testing Web Applications

Test Objective

The objective of testing web-based applications is to demonstrate that:

- Users' activities in the web browser have the desired effect within the core application and database.
- The security constraints that exist for a standard network are maintained when accessed via the web.

The testing can be grouped into three distinct areas:

- Testing of the system functionality
- Testing of the system security
- Load testing

Automated test tools are used extensively to test web-based applications, but care needs to be taken to ensure that the specific automated test tool is able to simulate the user input using the specific web technology deployed, e.g., Java, Flash, or Active X.

System Functionality

As the system can be client centric, server centric or a mix of both, the functional testing should be scripted to suit the focus area. Testing would typically be split into unit testing at the development level and then user testing to verify the end to end functionality. Unit testing can be grouped into three distinct areas:

- The web client
- The web server (middle platform)
- The core application including the database

Automated testing tools are frequently used for user testing. For further guidance, see Appendix T11.

System Security

Security should be tested to verify that the integrity of a system can be maintained from all points, i.e., the web, web server, and core application. Consideration should be given to the intentional hacking of the system, and appropriate controls established and tested.

Load Testing

Load testing of web applications should be performed to ensure that the system performs as intended. The following tests should be considered:

- Transaction
- Throughput
- Utilization
- Service Time

- Queue Time
- Response Time
- Scalability
- Performance

The number of users can vary significantly during load testing for web-based applications due to the nature of the system.

4.3.7 Regression Testing

4.3.7.1 Test Objective

The objective of regression testing is to demonstrate, following a change, that aspects of the software not involved in the change were not adversely impacted. This is in addition to testing that evaluates the correct functionality of the changes to the system. Regression testing is typically associated with implementing changes in the operational phase. For further information, see Appendix T8.

As stated for a number of test types above, the use of automated test tools can significantly reduce the time required to execute regression testing and/or can significantly extend the scope of regression testing which can be achieved for each new release of software.

4.3.7.2 Test Scope

Regression testing is normally achieved by the re-execution of original test cases that have already been proven to give the expected outcome. The scope of all regression testing should be based upon regression analysis to determine the scope of functionality potentially affected by the change and should reflect both the risk priority associated with the system or function and the likely impact of the change being made. The outcome of the regression analysis may indicate that additional new test cases are required.

Where an automated test tool was developed as part of the original software development, it may be modified as necessary under configuration management, or managed within another test management tool, to align with the revised software, and then re-used for regression testing. For further information on the use of automated test tools, see Appendix T11.

4.3.7.3 Test Positioning within the Life Cycle

Regression testing is necessary following changes to previously baselined systems both during the project phase (e.g., following a change made to correct a test incident) and during the operational phase of the system life cycle.

4.3.8 Disaster Recovery Testing

This section is designed to complement Appendix O10 of GAMP® 5 [1] and be read in conjunction with the GAMP® Good Practice Guide on IT Infrastructure Control and Compliance [10] and Section 14 of the GAMP® Good Practice Guide on A Risk-Based Approach to Operation of GxP Computerized Systems [11].

Disaster Recovery Planning is a sub-set of Business Continuity Management that focuses on regaining access to an IT system, including software, hardware, and data following a disaster.

4.3.8.1 Test Objective

Disaster recovery testing has two objectives:

- To check, as part of disaster recovery planning, that elements of a system can be recovered in the event of foreseeable disasters such as loss of the normal operating hardware, e.g., restoration testing.
- To verify, following a disaster, that recovery of the system and associated data has been successful.

Disaster recovery planning seeks to ensure that, in the event of a disaster, the disruption to critical or strategic business activities can be minimized. This includes:

- Prior identification and a documented assessment of possible risk (disaster) scenarios.
- Formulation of appropriate contingency plans, which may, e.g., include transferring operations to another location.
- Specification of the Recovery Point Objective – the point in time to which data must be restored following a failure or disaster loss.
- Formulation of appropriate test plans to demonstrate that the recovered system is working to specification.
- Formal recording and communication of these plans.

4.3.8.2 Test Scope

Testing as Part of Disaster Recovery Planning

Although it may be impractical to test an entire disaster recovery strategy, there may be parts of the plan which can be tested at a system level (e.g., checking that a server backup can be successfully restored to new hardware). If there are elements of the system that are operated and maintained by external suppliers on behalf of a regulated organization, their testing in a disaster situation should be considered as a requirement in their Service Level Agreement (SLA). This testing should align with any in-house testing necessary and should be subject to the same periodic testing.

The frequency of disaster recovery testing for a system should be risk-based and take account of business and technological changes.

Disaster recovery test types also should be chosen based on risk, taking into account the criticality and complexity of the system. A variety of disaster recovery test methods are available, such as:

- Paper Test: this is a paper walkthrough of the plan that involves key personnel in the plan's execution. During the exercise, personnel involved should reason out what might happen in a particular type of service disruption. This may involve a walkthrough of the entire plan or just a portion of the plan. A paper test usually precedes a parallel test.
- Parallel Test: this type of test usually involves a local version of a full test where actual resources are utilized in the simulation of a disaster. Parallel tests are performed regularly on different aspects of the Business Continuity Plan (BCP) and can be a cost-effective way to gradually obtain evidence about the effectiveness of the plan. It also provides a means to improve the BCP in increments.
- Full Operational Test: this is one step away from an actual service disruption. The organization should have thoroughly tested the plan on paper and locally as a parallel test (i.e., within an equivalent, but non-operational environment) before endeavoring to completely shut down operations.

The scope of testing also should also consider verifying reporting/escalation lines within the organization.

Testing to Verify Correct Recovery from a Disaster Situation

Depending on what a regulated organization deems to be the most appropriate way of recording this information, it may be decided to split the overall plan into two separate, but complementary, documents:

- A BCP – to outline how the core business processes would operate until such time as the disaster is declared to be over. Business continuity plans should be developed in line with the relevant SLAs agreed between the business and the supporting IT organization.

- A Disaster Recovery Plan (DRP) – to include how and in what order the supporting IT infrastructure and applications would be recovered and tested to ensure that they are ready to support the core business processes again.

DRPs are usually developed with the assumption that a disaster would involve major incidents, such as natural disasters, accidents, man-made disasters, or financial crises. The DRP also should be sufficiently scalable to ensure that individual parts of the plan are applicable for lesser (and more common) emergency situations, such as hardware failure, power outages, or virus attack, as well as confined fire or flood damage. The amount of testing required to be undertaken also should be scalable in accordance with the overall DRP.

The scope of testing to be undertaken needs to strike a balance between providing confirmation that all required elements of any system are recovered and fully operational and the likely business pressure to restore normal operations as soon as possible. Configuration items, such as databases, software libraries, and test data sets that might not normally form part of the testing life cycle may all need to be tested in a disaster situation to verify that they have been correctly restored. Specific test cases may need to be developed to cover these areas. These may be in the form of a checklist that ensures certain activities take place at defined periods of time and coordinated across sites to confirm the testing activity.

Where application test scripts are required, existing documentation, such as installation plans/checklists, configuration guides, user manuals, and test cases used for regression testing can all provide important input to the final test documentation set, as their successful use has generally been proven beforehand.

Should a disaster occur which has serious physical and resource implications, it may be the case that personnel who are being asked to execute this form of testing may well have less experience of the system than would ideally be required. Therefore, it may be prudent to ensure that the steps documented in any new scripts developed to cover this area contain slightly more detail than would normally be expected.

4.3.8.3 Test Positioning within the Life Cycle

Testing as Part of Disaster Recovery Planning

Disaster recovery tests, such as restoration from back-up, are normally performed as part of the verification activities for a system. This testing may be performed by the supplier and leveraged by the regulated organization or the testing may be done directly by a regulated organization.

To derive full benefit from disaster recovery testing, it is important that test protocols or specifications and individual test scripts are updated when systems are subsequently modified. Periodic readiness testing should ensure that any hardware and infrastructure retained as an alternative means of supporting business operations remains in a fit state to be used at short notice. Similarly, checks also should take place at regular intervals to ensure that any data stored on backup tapes or other durable media is being created correctly and can subsequently be restored in the event that this becomes necessary. For further information on testing as part of data management, see Appendix T9.

Testing to Verify Correct Recovery from a Disaster Situation

No testing can take place until the required hardware, infrastructure, and network elements of the system have been recovered, and there may be some lead time involved in this activity. While the test team may not have any responsibility for the recovery of these items, according to the nature of the event that has arisen, they may nevertheless be required to provide support in these areas.

Once the required hardware, infrastructure, and network elements are in place, the main DRP would normally then specify the order in which the software applications are to be recovered, with live production instances and any other systems that have an immediate impact on health and safety being placed ahead of less critical systems.

Consideration also should be given to ensure that any documentation required to support the disaster recovery testing is readily available in the event of a disaster occurring. Administrative file servers and document management systems may have been assigned a lower priority in a systems recovery so alternative storage arrangements should be considered, such as off-site hard copies or on-site electronic copies stored in a fire safe.

4.3.9 Decommissioning Testing

4.3.9.1 Test Objective

The objective of decommissioning testing is to demonstrate, following the decommissioning of a system, that associated systems are not adversely affected and that archived data can still be accessed. Data migration testing may be an important part of this, and automated test tools can assist greatly with this process. For further information, see Appendix T11.

4.3.9.2 Test Scope

Decommissioning testing is normally achieved by re-running original test cases for other systems associated with the decommissioned system and the execution of test cases to verify the integrity and availability of any data archived from the decommissioned system. For further information on testing as part of data management, see Appendix T9.

4.3.9.3 Test Positioning within the Life Cycle

The testing of associated systems will be conducted upon the decommissioning of the system in question. Data integrity and availability testing also should be conducted upon the decommissioning of the system in question and also on a periodic basis.

4.4 Determination of the Scope of Testing

4.4.1 Testing to Reduce Risk

Testing aims to confirm that effective risk controls are in place and to identify defects in the system as early as possible in the life cycle. There are physical and environmental constraints which will determine when in the life cycle certain types of testing can be performed, e.g., it may not be possible to perform load testing in a simulated test environment.

The scope of the testing is determined as output from the risk management process, in particular from the functional risk assessment which determines the risk priority of critical functions. The identified risk mitigations should be used to generate test cases to ensure that risk scenarios have been adequately resolved. The level of test coverage should be commensurate with the risk priority associated with the system, application, or individual function. Except for the simplest of programs, software cannot be exhaustively tested.

The scope of testing should be determined by a justified and documented risk assessment (see Chapter 2 of this Guide), taking into account the potential effect on patient safety, product quality, and data integrity, and the intrinsic risk associated with the method of implementation, e.g., implemented using mature standard product or custom code.

The risk assessment may categorize system or business requirements according to their associated risk priority. For further information on a risk-based approach, see Appendix M3 of GAMP® 5 [1]. This may then be used to justify the amount and type of testing; focusing the resource and effort to areas of the system that represent the highest risk priority.

For example, a risk assessment could define three categories; low, medium, and high risk priority. The resulting combined test coverage requirements for the supplier and regulated organization across all phases of testing may then be defined. The following example is intended to be illustrative only, and not definitive.

Note that test type selection also should be based on the actual requirement and how it is best verified or challenged, and not just risk. Note also that the primary link between testing and Quality Risk Management (QRM) is that testing should verify the effectiveness of identified risk management controls, rather than the risk assessment being used only to identify types and levels of testing.

Table 4.7: Example of Test Type Selection Based Upon Risk Assessment

	Low Risk Priority Requirement	**Medium Risk Priority Requirement**	**High Risk Priority Requirement**
Structural Testing	(Not required – coverage during functional test assumed to be adequate).	100% branch coverage required – typically by supplier.	100% condition coverage required – typically by supplier.
Functional Testing	Test normal, invalid and special cases as defined in user requirements.	Test normal, invalid and special cases as defined in user and functional requirements. Ensure that test coverage includes exercising of all software outputs.	Test normal, invalid and special cases as defined in user and functional requirements. Ensure that test coverage includes exercising of all software outputs. Also test a selection of multiple input conditions.
Performance Testing	Test performance against defined user requirements.	Test performance against defined user requirements. Test performance under expected load conditions.	Test performance against defined user requirements. Test performance under expected load conditions.
Challenge Testing	(Not required)	Test data validity and security. Test for tolerance of hardware faults.	Test data validity and security. Test for tolerance of hardware faults. Perform stress test.

Note: Table 4.7 refers to development testing and not all test types are included. However, the concept of risk-based testing can be extended to all test types. Examples of coverage for particular types of systems are given in the E Appendices.

The system development life cycle stage in which each type of test occurs will depend upon the type of system (or application) being developed. When each test type is used should be clearly defined in the development life cycle. An example of documenting appropriate test types based upon the outcome of a risk assessment is shown in Table 4.8.

Table 4.8 shows the assessed risk priority, the test types required to demonstrate effective risk mitigation, and the test phase in which the test types will be executed. For a low risk priority, not all test types may be required. The example is intended to be illustrative only, and not definitive.

Table 4.8: Example Test Scope Defined in Risk Mitigation Strategy

User Req. ID	Description	Risk Priority	Risk Mitigation	
			Testing	
			Type	Description
5.1.1	Pressure Test Sequence	☐ High ☒ Med ☐ Low	☒ Structural Testing	Test Specifications or Protocols (module test) should ensure that all possible branches are covered.
			☒ Functional Testing	Test Specifications or Protocols (factory acceptance) should ensure that normal path (pass) through pressure test operates correctly. Test Specifications or Protocols (factory acceptance) should ensure that failure path on pressure test fail operates correctly. Test Specifications or Protocols (factory acceptance) should ensure that all defined failure conditions (e.g., valve failures) initiate the correct failure action.
			☒ Performance Testing	Test Specifications or Protocols (site acceptance) should repeat factory acceptance tests with all inputs live.
			☒ Challenge Testing	Test Specifications or Protocols (factory acceptance) should establish that the sequence cannot be run by an unauthorized user. Test Specifications or Protocols (factory acceptance) should establish that hardware faults on inputs or outputs initiate the correct failure action.

5 Appendix T2 – Test Planning and Test Management

5.1 Introduction

Test planning and management activities are fundamental to the overall effectiveness of testing.

5.2 Test Plan or Strategy

For each project, a test plan or test strategy document should consider:

- Required test coverage based on risk assessment and justification for leveraging of any previous testing
- Test phasing, including location and timing of the tests, and split of coverage between the phases
- Required test environments
- Responsibilities within a test team
- How test incidents will be managed and the interface to change management and configuration management systems
- Any requirements for test metrics to assess the effectiveness of testing
- The use of computer based test tools to support manual testing and any requirements for automated testing
- Reference to standard test procedures, templates and conventions; including those covering pre-approval of tests and post-test review, reporting and system release.

The test plan or test strategy document should be clearly understood and agreed by the project stakeholders.

5.2.1 Planning the Necessary Test Coverage

Test planning and management activities should include the method of demonstrating that the types and scope of testing required by the test policy have been included. The test plan or strategy should require that there is clear, documented traceability between the requirements outlined in the controlling specifications, identified risk mitigations, and the testing undertaken.

Requirements traceability should be initiated at the earliest appropriate point in a project (e.g., after approval of the User Requirement Specification (URS) for a linear method or during/at the completion of the first iteration for a non-linear method).

Traceability to appropriate requirements and risk mitigations should be maintained at the test case or test script level. Where a test covers several requirements, further breakdown within the test steps may be considered in order to demonstrate precisely where a specific requirement is tested.

A requirements traceability matrix (or equivalent) can provide benefits for testing, including:

- Documented test coverage of all critical (GxP) requirements and all identified risk mitigations
- Traceability of test cases to requirements and risk mitigations
- Determination of which testing documents may require update as a result of a change

- Reduction of redundancy in test specifications or protocols (and, therefore, of test effort)
- Ease of presentation to regulators, in order to demonstrate the completeness of testing within the development process for a given application

It may be used to demonstrate that the scope and rigor of the executed test cases is appropriate to the risk priority associated with the requirements.

Computerized test tools may be able to manage the traceability between requirements and test cases. In some cases, this may be limited, e.g., it may not be possible to:

- Manage many-to-many relationships between multiple requirements and multiple test cases
- Provide traceability to other objects, such as risk assessments

The need for requirements traceability should be considered on a system-by-system basis.

Some requirements may be verified by methods other than testing and may be traceable to documentation other than test cases or scripts.

For further information, see Appendix M5 of GAMP® 5 [1].

For further information on the types of testing and appropriate scope, see Appendix T1.

5.2.2 Planning the Appropriate Test Phasing

Verification of correct functioning should commence at the smallest possible module (or unit) within a system. On confirmation that each module operates successfully in a standalone mode, these verified modules can be integrated into larger subsystems and eventually into the completed system. Testing of the larger subsystems and of the system as a whole can then assume correct functioning of the smaller elements from which they have been built and focus on the interfaces between these elements.

Depending on the type and size of the system, there may be several test phases that support the validation process. The test phases can include:

- Module testing
- Integration testing
- Formal factory acceptance testing at supplier premises
- Verification of correct installation
- Formal acceptance testing to verify that a system meets a regulated organization's requirements

The split of the test scope between the test phases, their location, and whether it is the supplier or regulated organization which takes responsibility for each test phase will depend on factors such as:

- The competence and expertise of the supplier as judged by the supplier assessment process
- The degree of packaging of the system with the process equipment (if any)
- The ease with which a representative test environment can be created.

- The degree to which interfaces could be changed by the move from one test environment to another and require re-testing in the new environment

- Any planned phasing of delivery into vertical slices (e.g., one plant or business area) or horizontal slices (e.g., input/output layer then control modules then sequential and batch control)

- Any potential efficiencies that can be gained by leveraging prior testing and/or combining testing of the system or application with that of the wider manufacturing, business, or other regulated process

Planning should aim to make clear the relationship of test activities to other system development activities. The timing of these activities should complement and integrate with existing life cycle activities. For example, installation verification may be required as a prerequisite for acceptance of a given environment (e.g., development, testing, or operational).

Planning also should recognize that, in some test phases, computerized system testing may be most efficiently conducted as part of testing a wider manufacturing or business process.

Testing of a system or application should attempt to leverage prior testing and supplier documentation. This will minimize duplication of effort for testing and has the potential to increase the efficiency of the testing phase. Testing documentation should be reviewed during the supplier assessment in order to determine the risk to the regulated organization.

5.2.3 Planning and Managing the Test Environment

Planning and managing test environments are important to the overall success of each test phase.

When planning and managing test environments, the following should be considered:

- The test environments required for each phase of testing and how these will be separated from the operational environment (e.g., physically separated into a test environment or performed on an operational system, but segregated from actual GxP processes)

- Verification that a test environment is appropriate and adequate

- Requirements for specific equipment and timely ordering and integration of that equipment, e.g.:

  - Test items to represent those in an operational environment

  - Test equipment (possibly needing calibration)

  - Peripheral equipment, e.g., printers

- Simulations (hardware or software) of business process responses, e.g.:

  - Where performance testing is to be undertaken, the sizing and response times of equipment can be significant in avoiding misleading test results.

- Supporting services required by each test environment, e.g.:

  - Test user accounts

  - Allocation of shared equipment

- Data required in each test environment
- Documentation required as a basis for tests in each environment
- How computer based test management tools will be used and controlled in relation to test environments (test management tools may require project specific configuration, with different test benches being created for each test cycle)
- How closely each test environment simulates the operational environment
- How risks introduced by the differences between the test environment and the operational environment will be managed
- How modules will be promoted to the next environment and/or changes backtracked into previous environments, e.g.:
    - Where a number of test environments are in use simultaneously as in a project with phased delivery requirements

For further information on the selection and control of test environments, see Appendix T4.

5.2.4 Planning and Managing a Test Team

Sufficient time should be allocated to recruit suitable staff to a test team. Test planning should ensure that:

- Test team resource requirements are identified early in a project life cycle
- The skill sets required by testers are identified

Personnel taking responsibility for each test team role should be identified. Conflicts of interest should be avoided when combining test team roles. Several people may be needed to view and record results, e.g., several testers may be needed to execute and observe a test which has multiple outputs or a rapid sequence of events that needs to be observed.

The use of witnesses during testing should be carefully considered and may incur significant costs. The selection of test witnesses should be based on training and experience criteria similar to those applied to the selection of testers. The role of test witness can usually be combined with that of test result reviewer.

The use of witnesses during testing is not normally a regulatory expectation and not necessary except for specific practical or commercial reasons. In limited instances (such as meeting contractual requirements for acceptance testing), a regulated organization may require a test to be witnessed.

For in-house projects, testers may be from other areas of the business. These personnel should be fully conversant with an organization's operation, but may require training in IT skills in order to support a role on the test team.

External resources may be recruited to work within a test team. These testers may be recruited with the necessary IT skills, but may need to acquire knowledge about business processes and regulations within the life sciences industry.

Where the resource profile allows, testers may come from within the project team. It can be financially beneficial to utilize project team members as much as possible, so project team members having the correct skills required to fulfill testing roles (e.g., business analysts, deployment personnel, and training personnel) may be considered.

Testers also may come from within a supplier organization.

Testers should not be expected to participate in a testing program in addition to another full time job role. Where personnel have been seconded to a test team for a period of time, test managers should ensure that testers do not come under undue pressure from their original role, unless there is a business emergency.

It is preferable to locate a test team in one place. This may not be feasible where a team is large or needs to be geographically dispersed in order to be able to perform the testing objective (e.g., verifying that global interfaces work).

Once the testing is in progress management of and communication within the test team should be unambiguous, particularly where the team includes remote testers or testers who are part of a subcontract organization. The type of tasks testers will be expected to perform and line reporting arrangements should be established, e.g., the terms of reference for test roles, and arrangements for catering facilities, security access, travel, etc., especially if testing will involve extended hours.

Testing roles may be specialized, e.g., developing automated test scripts, or planning and executing performance testing. Extensive training should be provided or professional testers should be hired for specialist roles and sufficient time should be allowed of these activities.

5.2.4.1 Dedicated Test Teams

A dedicated test team is a team whose skills and experience allow them to test a variety of systems and whose core role is to provide this service at a consistent level across an organization.

Advantages of Maintaining a Dedicated Test Team

- Team members have the opportunity to acquire specialist testing knowledge and skills that can add value to future testing activities. It also provides a good opportunity for knowledge sharing.
- The ability to define and implement good practices (through a central group rather than multiple teams). For example, dedicated test team members may already know how to develop effective test cases and test scripts and how to record incidents, etc.
- Provision of consistent delivery of testing services. This can be difficult to establish when forming new test teams for every new project.
- Minimization of potential disruption, e.g., personnel having to cover more than one role at once.
- The dedicated test team will normally be assigned to a new project once the current system has been tested. Maintenance and re-testing of that system through its operational life will not be able to rely on the experience and expertise gained by the test team.
- Prioritization of tasks can be easier.
- Testing tool ownership, administration, and support can be rationalized, leading to potential cost savings.
- A simpler management layer may be feasible, rather than multiple team managers.
- Better accountability/measurability may be feasible, allowing the capture of metrics for an entire stream of work (testing) to help management to determine the value delivered by the testing team.
- Outsourcing of testing activities could be easier.

Disadvantages of Maintaining a Dedicated Test Team

- The potential and ability to find specific knowledge for testing a specific system is reduced.

- Team members may not understand specific requirements and business processes under test.
- Professional IT testers may not be able to identify key features in new software, e.g., level of usability of input screens may not be appreciated by testers. Business users are likely to be more aware of the usability in the operating environment and may provide feedback that a dedicated test team may not.
- If the project pipeline is not particularly strong, test resources may become underutilized, which may not be cost effective.
- Test team expertise is lost from the operational unit at the end of testing. This may make system maintenance and on-going regression testing more difficult.
- Test cases and test scripts may be understood only within the testing team and are not re-useable in the operational environment (e.g., during regression tests).
- Where external (contract) test teams are used, there may be issues over accountability.

Outsourced Testing Teams

It may be beneficial to outsource components of testing. Considerations for outsourcing test activities or components of the test activities include:

- Supplier selection
- Assess supplier skill levels and process maturities with respect to specific requirements for the life sciences/ pharmaceutical industry
- Compatibility of the supplier/user QMSs
- Contracts and governance (e.g., SLAs, management of deviations)
- Defined minimum standards for documentation and other deliverables
- Communication content, frequency, and audience
- Acceptance criteria of delivered content

Expectations for these and any additional items required for a project should be established to help to ensure the success of the user/supplier relationship. Outsourcing can provide benefits for both users and suppliers, but the extent and scope of the relationship should be clearly defined at the onset of the contract.

Third party suppliers may provide testing and validation support for their applications or other applications. The supplier relationship should be considered.

For further information, see Appendix S5 of GAMP® 5 [1].

5.2.4.2 Roles and Responsibilities

The roles and responsibilities within a test team can vary significantly depending on the nature, scope, and phase of testing, as well as an organization's size and structure. Specific roles required for each project should be documented and approved in a test plan or test strategy.

Roles can be split into two distinct groups:

1. Roles within a test team

2. Supporting roles performed by personnel operating outside the test team, but whose responsibilities nevertheless have a direct link to the success of any testing process

Typically only large, complex projects will have different individuals fulfilling the roles defined in this section of the Guide. The roles described may not all be necessary for every test team. In addition, depending on the size of a project, some roles may require only part-time involvement or full time involvement only for a short time, e.g., those that interface with the test team.

Some roles may be combined, providing that this does not introduce conflicts of interest, or compromise of the independence of review. For example:

- Test script authors should not formally review and approve their own test scripts prior to execution.

- Testers should not formally review and approve their own test results/reports.

Personnel with a particular specialist skill, e.g., computer system validation personnel, may fulfill the same part-time role on multiple concurrent projects.

Table 5.1: Test Roles and Responsibilities

Role	Test Policy (T1)	Test Planning (T2)	Test Specification (T3)	Test Environment (T4)	Test Execution (T5) and Results Recording (T6)	Test Results Review (T6)	Test Incident Management (T2)	Test Reporting (T7)	Summary of Responsibilities
Project Manager	A	C	I	I	I	I	I	I	• Ensures that all testing activities are included in the project plan and may provide input into the development of the test plan or strategy • Resolves any escalated issues that have wider implications than just the test team
Test Manager		A	I	A	I	I	I	A	• Plans and coordinates all the test activities and reports the status of the testing activities to the project manager
Test Team Leader		R	A	R	A	A	A	R	• Oversees the coordination and execution of the testing for discrete areas of the application (in large test teams) • Reports progress back to the test manager, allowing the test manager to concentrate on the planning and strategic activities necessary to ensure the testing meets its agreed timelines
Key: R = Responsible (the "doer"), A = Accountable (liable for the outcome), C = Consulted (provides input), I = Informed (aware of the activity status)									

Table 5.1: Test Roles and Responsibilities (continued)

Role	Test Policy (T1)	Test Planning (T2)	Test Specification (T3)	Test Environment (T4)	Test Execution (T5) and Results Recording (T6)	Test Results Review (T6)	Test Incident Management (T2)	Test Reporting (T7)	Summary of Responsibilities
Test Analyst or Test Script Author			R						• Defines test cases or test scripts including the step-by-step instructions in order to test a specific functional area of the system • May analyze incidents arising during the test process to determine their root cause and document this to aid the resolution process
Test Case or Test Script Reviewer			R						• Ensures that all requirements/risk mitigations have been adequately covered by the test scripts • Ensures that test cases or test scripts include a requirement for producing documented evidence that requirements have been implemented and tested • Within the regulated organization, this role should be fulfilled by a suitable SME, which may include IT Quality staff
Testers					R				• Executes the tests according to the approved test cases and test scripts • Informs the incident manager of suspected issues • Produces and collates the required documented evidence as defined in the test cases and test scripts
Test Witnesses					I				• In limited instances (such as meeting contractual requirements for acceptance testing), the regulated organization may deem it necessary to have the test witnessed • Where test witnesses are used, the selection of the witnesses should be based on similar training and experience criteria as applied to testers. This role should be fulfilled by a suitable SME or SMEs, whose expertise includes both test quality requirements and technical requirements of the project. • Where the use of a test witness is required, this role can usually be combined with that of test result reviewer.

Key: R = Responsible (the "doer"), A = Accountable (liable for the outcome), C = Consulted (provides input), I = Informed (aware of the activity status)

Table 5.1: Test Roles and Responsibilities (continued)

Role	Test Policy (T1)	Test Planning (T2)	Test Specification (T3)	Test Environment (T4)	Test Execution (T5) and Results Recording (T6)	Test Results Review (T6)	Test Incident Management (T2)	Test Reporting (T7)	Summary of Responsibilities
Test Result Reviewer						R			• Ensures that all test steps of the test scripts have been executed • Ensures that documented results or other evidence has been correctly produced, signed, dated, and referenced for all applicable test steps • Ensures that exceptions/incidents have been documented • Confirms the output (pass/fail status) of the test, based on documented evidence
Exceptions/ Incident Manager							R		• Takes responsibility in larger projects for ensuring that all exceptions/incidents are reported and that solutions are implemented and retested. In smaller projects, this role may be performed by the test manager.
SMEs	C	C	C	C	C	C	C	C	• May include test team administrators, technical personnel, software developers, and test infrastructure support personnel
Corporate Quality Unit	C	C	C	C	C	C	C	C	• Where permitted by corporate policy, this role may be delegated to suitably qualified individuals outside of the QA/QC Department, e.g., computer systems validation specialist or IT Quality staff. • Independently reviews and rejects or approves the test plans prior to implementation in order to confirm that they cover all of the required quality objectives of the project • Provides one of the review/approval signatures on the test reports to confirm that stated requirements have been verified from a quality perspective as part of the testing activities • The IT Quality staff, where available, should provide guidance and support throughout the testing cycles, and also may review deliverables such as test cases.

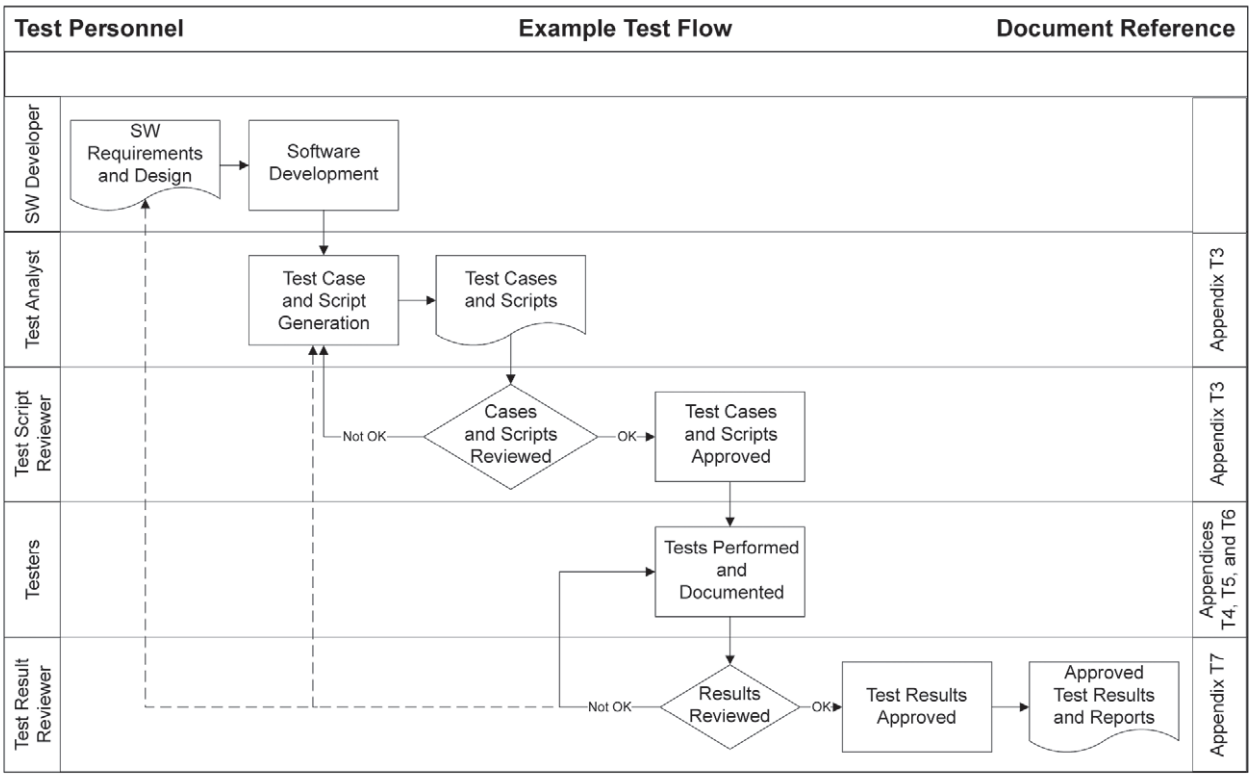
Key: R = Responsible (the "doer"), A = Accountable (liable for the outcome), C = Consulted (provides input), I = Informed (aware of the activity status)

Table 5.1: Test Roles and Responsibilities (continued)

Role	Test Policy (T1)	Test Planning (T2)	Test Specification (T3)	Test Environment (T4)	Test Execution (T5) and Results Recording (T6)	Test Results Review (T6)	Test Incident Management (T2)	Test Reporting (T7)	Summary of Responsibilities
Process/ Data Owner	I	I	I	I	I	I	I	I	• The person to whom the test report is generally presented for review and sign off, in conjunction with QA, as evidence that the system is in compliance • When a change to critical data deviates from the described validated usage of a system, the change should be pre-approved by the data owner, and the implementation of the change should be approved by the data owner's Quality Unit. • The data owner and the process owner may be separate people for ERP systems, etc. • Note: while Process Owners may not be involved with detailed activities, they are responsible, from both a business and a regulatory perspective, for the fitness of systems for their intended use.
Key: R = Responsible (the "doer"), A = Accountable (liable for the outcome), C = Consulted (provides input), I = Informed (aware of the activity status)									

Figure 5.1 illustrates the test flow and some of the roles involved in the test process. It also indicates which Appendices in this Guide provide further detailed information on some of these processes.

Figure 5.1: Example of Test Process Flow and Associated Roles



5.2.4.3 Training for Test Roles

Purpose of Training

Training a test team is an integral part of ensuring its effectiveness. It is important that testing is accepted as a value added activity rather than just another task that needs to be ticked off on the project plan. Training should be properly planned to address a business need with clear, measurable objectives.

The benefits in ensuring that all test team members receive the correct training to perform their roles can apply across an entire project, e.g.:

- Trained test personnel are more likely to be able to spot potential issues or defects earlier on in the testing life cycle
- Trained test personnel are less likely to encounter problems in producing meaningful test cases and test scripts; therefore, minimizing the likelihood of tests having to be re-executed due to test case and test script problems
- Trained test personnel are less likely to encounter problems in determining how activities should be completed during testing cycles
- The number of avoidable human errors generated across the entire range of testing activities should be minimized
- Evidence of an effective testing process can give a business increased confidence that the product received will be of a high standard and will be fit for its intended use

Test team members should participate, as a minimum, in an induction program which explains how testing is expected to be performed on a project, as test strategies and supporting processes can vary greatly from one project to the next. Test team members should participate in the induction program even if they have undertaken a testing role prior to the new project.

Regardless of where testing personnel have been sourced from, it is important that all testers should undertake some form of GxP awareness training so that they can understand and appreciate the significance and impact of the work.

Test Staff Attributes

Testers should be selected on the basis of their ability to effectively participate in the testing process. While formal testing qualifications may not be necessary in order to be an effective tester, skills such as attention to detail and the ability to apply common sense are considered very valuable.

The skills required to participate in the different types of testing within the software development method can vary greatly, e.g., performance, stress, and regression testing are all test types that need specific experience.

Personnel who have successfully run unit or module testing within the development phase of a project may not have the necessary business process awareness to be able to produce test cases and test scripts for User acceptance testing.

Areas Covered by Training

Areas that should be covered in test team training include:

- Project organization and structure
- Where testing fits within a project/program
- Interfaces with other non-test team roles key to the success of the testing process
- Adherence to special instructions or SOPs (including overarching relevant project procedures such as the validation plan or project quality plan)
- The project test plan or strategy to be followed
- Introduction to the testing deliverables (including the use of templates)
- Introduction to good testing practice:
    - How to produce effective test cases, test scripts, and other testing deliverables
    - How to specify the correct type of test data
    - How to record the test output
    - How to collect and handle documented evidence
    - How to review the test output
    - How to report test events and other incidents
    - Project reporting cycle

- How to use any incident management reporting tool/database
- Use of computerized/automated test tools and/or test management tools
- Any required interface to the project change control mechanism
- Familiarization with project filing structure (both electronic and paper)
- Project contacts for information and guidance

5.2.5 Planning for and Managing Test Incidents

Unexpected incidents may occur during the development life cycle of a system or product, predominantly in the testing phase. Incidents should be promptly and correctly dealt with by the test team in order to minimize the likelihood of avoidable defects being passed into the operational environment.

Incident notifications may be received via different media (e.g., paper, e-mail, a computerized test tool, or a dedicated incident reporting system) and may require variations in the way they are handled. An incident management plan (or procedure) should be prepared to define the approach for dealing with individual input types. The incident management plan should be communicated across a test team before testing starts.

Depending on the size of the project, the incident management plan may be specifically developed for the test team or may form part of an overall project incident recording process. On small projects, the plan may form a sub-section of the validation plan, quality plan, or test plan.

Computerized test management tools may include a module for test incident management. This may allow the automatic generation of a test incident record from within a test run with links to the captured results or evidence (e.g., screen shots).

The process of managing a test incident may be automated using workflow and extensive reports are also usually available, summarizing the categorization, prioritization, and status of test incidents. For large or complex systems with a large number of tests, this can significantly enhance the efficient management and resolution of test incidents.

The aim of a test incident management process should be to:

- Establish and operate an effective service to manage the recording and subsequent handling (disposition) of test incidents and any other observations noted by testers
- Act as the single point of contact between end-users and problem solvers, service suppliers, system developers, and testers
- Handle all incidents in a consistent manner
- Demonstrate that all incidents have been properly resolved
- Achieve the agreed level of service criteria
- Feed forward into test metrics and to report performance against the service levels set out in the pertinent SLAs

5.2.5.1 Incident Analysis and Logging

Details of new test incidents should be recorded fully and an index of test incidents should be maintained.

Details of all incidents should be logged centrally for each project and each incident should be allocated a unique reference number, which should be communicated to the relevant parties for future reference and tracking. It may be beneficial to correlate incidents with subsequent change requests in order to identify the trigger for the change request and to ensure that the incident can be closed when the change has been implemented.

It can be beneficial to log incidents in a manner that allows visibility by all relevant personnel, allowing effective tracking and monitoring to take place. The method selected to do this should be appropriate to the scale and type of testing. Smaller projects may find a central spreadsheet approach to be sufficient. Larger or more complex projects may use a database or an application specifically developed to log incidents.

Test incidents may feed either into an existing change control system or into a separate process for resolving test incidents. An example of an incident report that feeds into an existing change request/change note system is provided on the GAMP® 5 resource CD [1]. An example of a simple test incident report suitable for a small project and which covers incident reporting and the resulting change and required retest and signoff is included below.

Where a test incident occurs during a particular test step, the overall test should not be continued if the failure produces an output, which prevents entry into a subsequent step. Where a test is continued following a failure, the failed step should be clearly identified on the test results sheet.

If an incident is a duplicate of an existing incident record, linked incidents should be cross-referenced so that they can all be closed when the incident is resolved.

Figure 5.2: Example of an Incident Report

<table>
<tr><td colspan="4">INCIDENT REPORT</td></tr>
<tr><td>Project</td><td>Incident Report Number

(page ___ of ____)</td><td colspan="2">Source of Change:
Test Reference
Run Number
Other</td></tr>
<tr><td colspan="4">Incident Description</td></tr>
<tr><td colspan="3">Raised By:</td><td>Date:</td></tr>
<tr><td colspan="4">INCIDENT REVIEW</td></tr>
<tr><td colspan="2">Incident Classification:</td><td>Priority:</td><td rowspan="2">Modification and retest requirements reviewed and authorized:

Date:</td></tr>
<tr><td colspan="3">Modification Agreed</td></tr>
<tr><td colspan="3">Details of Items Requiring Modification</td><td>IMPLEMENTATION</td></tr>
<tr><td>Files Affected</td><td>Original Version</td><td>Modified Version</td><td>Implemented By:

Date:</td></tr>
<tr><td>Documents Affected</td><td></td><td></td><td>Implemented By:

Date:</td></tr>
<tr><td>Hardware Affected</td><td></td><td></td><td>Implemented By:

Date:</td></tr>
<tr><td>Retest Requirements</td><td colspan="2"></td><td>RETEST
Test Passed By:

Date:</td></tr>
<tr><td colspan="4">CLOSURE following changes and successful re-test or agreement to abort test</td></tr>
<tr><td colspan="3">Incident Closed By:</td><td>Date:</td></tr>
</table>

5.2.5.2 Test Incident Classification

In addition to correcting an identified fault, test incidents should be evaluated to determine their most likely cause. Previously documented risk assessments may help to identify potential sources of error. Addressing the cause is part of a continual process improvement, and may include corrective and preventive actions. Metrics on the causes of avoidable test incidents can provide a useful indicator of areas within the overall software development method that may benefit from improvement activities. This can help to ensure that they reduce the likelihood that they could occur again.

It is important to formally define test incident categories before starting testing, based on the types of test incidents provided (the number of categories may vary from project to project based on size and complexity), i.e., more granular categorization will better serve larger, more complex projects running over many years. Table 5.2 shows typical incident types that can occur in software testing programs.

Table 5.2: Example Test Incident Classifications

Incident Type	Definition	Typical Resolution
Incorrect Software Installation	Errors such as program dumps, abnormal terminations, or the inability to access the application may be a result of the failure of the installation process, e.g., if the application has not been installed or configured correctly. Installation of a wrong software version also may cause a test incident.	It is usually necessary to postpone any further testing until the test environment has been correctly set up. A test incident should be raised to control correction of the test environment.
Incorrect Configuration/ Programming/ Coding	These incidents may result in the actual system output not agreeing with the expected system output (i.e., not as per the design specifications).	A test incident should be raised and unless the defect would invalidate the rest of the test steps, the execution of the test case can continue. Once the cause has been identified, the defect should be corrected and the corrected software included in a subsequent software build for retesting.
Incorrect Test Data	Testing failures may occur as a result of a failure to create the correct data in the test database in advance of the test case being run. For example, the test case requires the tester to select a particular value from a list of options available, but the required value is not in the list because it has not been entered in the appropriate data table.	Test data should be corrected and the test procedure re-executed. A test incident should be raised to control correction of the test data.

Table 5.2: Example Test Incident Classifications (continued)

Incident Type	Definition	Typical Resolution
Inadequate Specification (Incorrect Understanding of Program Functionality)	Testing failures may occur because the controlling Design Specification is not sufficiently clear about what is required from a specific piece of functionality. For example, where a custom system is being developed to satisfy a new business process that may not be fully established.	A change may be required to the relevant controlling Design Specification. In this case, the Functional Design and Package Configuration Specifications should be checked to ensure that they were not the source of the confusion. If the design documentation is found to be acceptable, the cause may lie with the User Requirements Specification (i.e., the functional decomposition of the user requirements was inaccurate). A test incident should be raised to control correction of the documentation and/or the configuration and coding.
Poorly Specified Test Case/Script	Tests can fail if the test case or test script (or other relevant document) produced is incorrect and indicates an outcome that is not as documented in the corresponding requirements. For example, if the test case indicates that something should happen in a particular order, and the software under test executes transactions in a different order, the software may not be at fault.	Where a test case or test script has been modified during execution, a test incident should be raised to manage changes to the test case or test script and to confirm the pass/fail status of the test. The incidence of this type of error should be minimized by ensuring that the independent review of the test cases before they are approved, including a cross check of the expected output as specified in the controlling requirement.
Incorrect Design Solution	Test errors can arise where the software may work correctly against design, but the design implemented did not satisfy the original stated requirements or failed to reflect subsequently agreed change requests.	A test incident should be raised to control correction of the design documentation and/or the configuration and coding. Where requirements are to be revised after the design phase has commenced, these should be brought to the attention of the design team to allow them to assess the impact of these changes on their current work.
Inconsistent Controlling Design Specification	Test incidents can occur where the content of the relevant controlling Design Specification contains inconsistencies. For example, if one requirement states that a particular default value for a data field should be zero, and another requirement states that the default value required for the same data field should be 10. The actual software code may be either of the values, according to whichever one was incorporated into the Technical Design.	A test incident should be raised to control correction of the Design Specification to prevent further confusion. This incident type should not be confused with Incorrect Programming/Coding, (where the software coded does not match a particular requirement of the controlling Design Specification, and the code needs to be corrected). Dependent on the nature of the inconsistency, the test reviewer may still categorize the test as having passed, providing evidence has been obtained to confirm that the functionality did work as intended.

Table 5.2: Example Test Incident Classifications (continued)

Incident Type	Definition	Typical Resolution
Unexpected Test Events	During the course of executing a test case or test script, the tester may notice an anomaly in the software that, although not affecting the success of the overall test objective, does require further investigation. An example would be if, on completing a particular test step, an unexpected pop-up window is generated.	A test incident should be raised to control correction of the Design Specification (and corresponding test case or test script) to include the unexpected event.
Test Execution Errors	Tests can be classified as failures if the tester fails to follow the steps defined in the test script or the overall test specifications or protocols governing the activity. For example, where the tester has failed to fill in a result for a particular step to indicate whether it has passed or failed or where the tester has ignored instructions to take screen prints at appropriate points to demonstrate correct operation of desired functionality or pressed the wrong button. Missing signatures and timestamps for dates and other important cross-reference information is another area that could cause a test to be considered a failure.	These errors typically result in re-execution of the test procedure. These errors also may be identified during the review of the testing documentation. In those cases, the initial test outcome documented by the tester may require modification. This also may result in re-execution of the test procedure.
External Events	Test incidents of this nature reflect an unexpected event over which the test or project team may have no control and which brings testing to a premature halt. For example, unexpected power outages, fire alarms, or peripheral or network failures unrelated to the test program.	These events can be raised as issues by the project team, but are generally resolved outside of the project.

5.2.5.3 Test Incident Prioritization

As software projects do not have unlimited resources, when incidents occur a process of prioritization should be followed to ensure that these resources are used in the most efficient manner. This should ensure that a balanced view is taken to determine the most effective way of allowing the overall project to make progress while any incidents undergo further investigation. For example, assigning a higher priority to the fixing of bugs that prevent new areas of software from being tested for the first time can often add more overall benefit than the higher prioritization of bugs that may affect only isolated, occasionally accessed areas of the code.

During the formal recording of a test incident, it is common for the test team to undertake some form of categorization of the incident, against a predefined list that has been agreed as part of a project's test plan or test strategy.

This categorization can provide the test/project management team with an indication of the severity of the problems and allow them to:

- Discuss with the user whether this initial severity is valid or whether they require it to be revised according to its impact on their business processes

- Allocate appropriate resources to undertake further investigations and rectify the cause of the incident, scaled according to the final severity assigned

- Decide to carry the non-conformance forward for resolution in a subsequent stage of the project, putting in place any necessary workarounds

Where a non-conformance needs to be carried forward:

- Discussion with interested parties, e.g., process owners and Quality Unit, relating to the possibility of to releasing a system with known minor GxP deficiencies for production use may be required. Users may be willing to operate with a workaround in the short term to obtain a business benefit from the introduction of the overall system functionality.

- A documented risk assessment should be undertaken and appropriate documented procedures which detail the workaround should be established. The risk assessment should include timescales for correction and how that correction will be implemented within the operational environment.

- Where the risk assessment shows unacceptable risk to patient safety, product quality or data integrity, either while the proposed workaround is in place or during the implementation of the correction in the operating environment, the system should be put into production only after the deficiencies are corrected.

5.2.5.4 Change Control and Configuration Management as Part of Test Incident Resolution

Incidents found during testing should be recorded or referenced. Where a change is required, the affected items (hardware or software or documentation) should be identified, including the version number in which the incident occurred. The change control process should ensure that all details of the incident are recorded and that the necessary corrective actions are implemented and tested.

A separate process for resolving test incidents may be defined, but its scope should be clearly described with regard to the change control process. The boundaries of each process and the inter-relationship between test incident management and change control should be clearly defined, including inputs to and outputs from each process.

For example, the impact and regression analysis performed during the assessment of a test incident may render unnecessary any subsequent formal impact analysis required under the change control process. However, implementing any change to the system will still require formal approval.

Configuration management records may be used to help determine the underlying cause of test incidents. For example, reviewing the inter-relationships between configuration items may help to determine the reason for a test failure. This also may help to rationalize the scope of any regression testing.

Where a change is made, configuration management records should be updated as part of the test incident resolution.

Different levels of formality (see Figure 5.3) in the change management process may be appropriate to different:

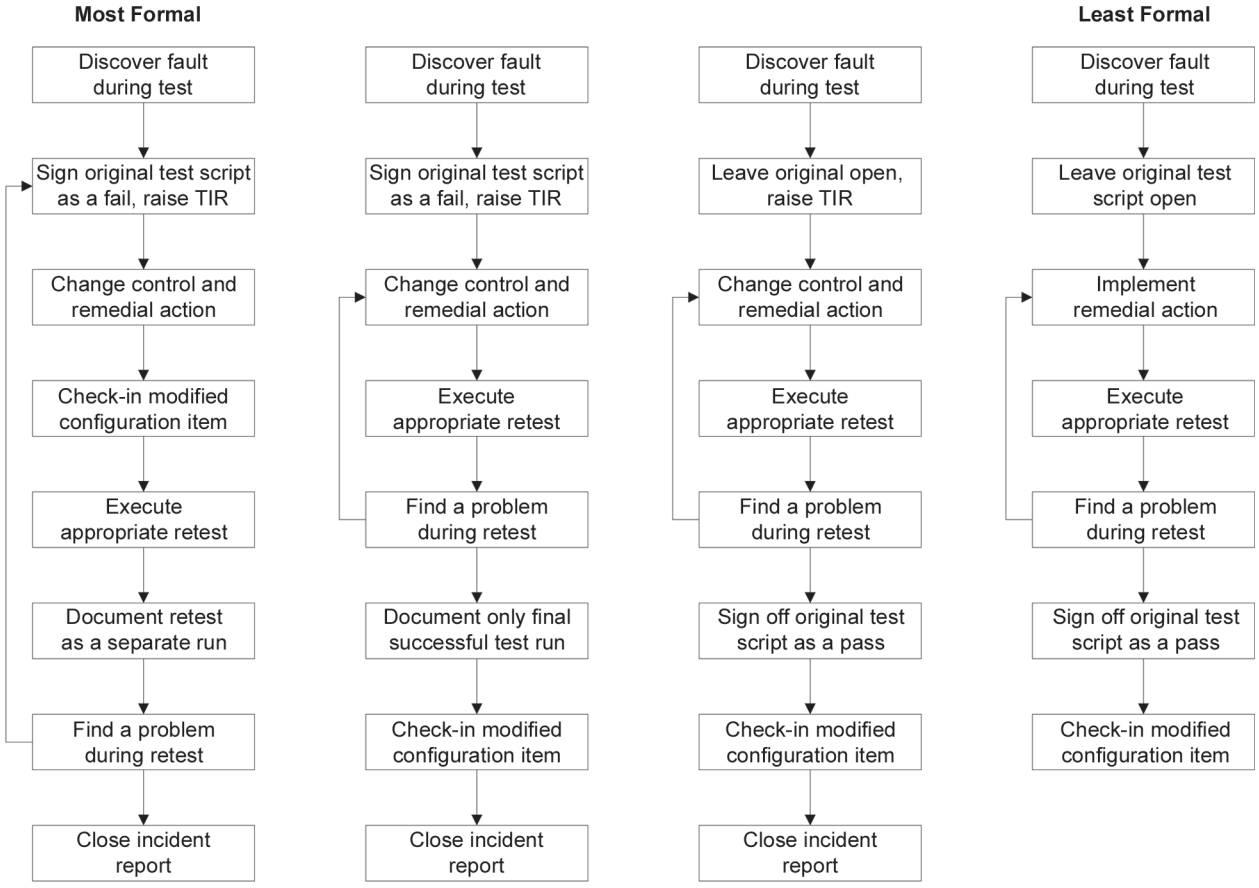
- Test types

- Risk priorities

- Phases of the life cycle

The least formal may be suitable only for early stages of development testing while the most formal might be a requirement if making a change to a high risk operational system. The expectations for each phase of testing should be defined in the test plan or test strategy.

Figure 5.3: Possible Levels of Formality in Incident Management



For further information, see Appendices M8, M9, and O6 of GAMP® 5 [1].

5.2.5.5 Testing as Part of Test Incident Resolution

Re-execution of failed tests, i.e., those for which test incidents have been raised, should verify that test incidents have been resolved successfully. Regression analysis and testing should provide confidence that changes made as a result of incident resolution have not introduced any further defects or undocumented features into the software. This is usually achieved by re-running original test cases that have been proven to give the expected outcome. Additional test cases also may be developed to verify that this is the case.

Automated regression testing can improve the efficiency of regression testing for change controls or configuration changes that may be implemented.

Regression test suites should be built with many small tests, rather than fewer large tests, for both manual and automated systems. This can help to isolate defects and allow more than one tester to become involved in running regression testing, minimizing the time spent on re-testing activities.

Depending on the extent of the impact of the incident, it may be necessary to re-run only a subset of previous test cases that would logically be affected by the original incident. Where the impact of the incident has been considerable, increasing the scope of regression testing may be considered.

5.3 Test Metrics

Test metrics provide a process for the formal assessment and reporting of quality and performance measures during the verification of a system.

5.3.1 Selection of Metrics

There is a trade-off between the effort taken to gather and present the metrics and the efficiency gains from using those metrics; therefore, metrics should be selected carefully according to their intended use and the ease with which the relevant base data can be gathered.

Table 5.3

Intended Goal	Possible Metrics	Example Use of those Metrics
Improving the quality of the lifecycle activities prior to testing or Improving the quality of released products or systems (end point metrics)	• Defect density (e.g., number of defects found per 1000 lines of code or per function) • Percentage of test cases passed • Severity distribution of the identified defects (e.g., as measured by risk to patients or by cost to correct) • Root cause of the identified defects (e.g., classified as having been introduced during design, implementation, previous changes)	• Feed back into continuous improvement of activities such as coding and code reviewing
Improving the quality of the test process itself (in process metrics)	• Test coverage (e.g., percentage of requirements for which test cases exist) • Escape rate and phase containment measures (e.g., proportion of defects discovered and resolved within a particular test phase) • Residual error rates (e.g., proportion of defects identified in the operational environment not previously discovered by testing)	• Help define when a system is of an acceptable quality to be released (by comparison to testing benchmarks such as software defects detected per test cycle) • Provide information that is fed into risk assessments • Allow managers to manage the testing process
Improving testing efficiency	• Test generation or execution productivity (test cases per unit time) • Test generation or execution efficiency (e.g., test effort for a set number of lines of code or set number of functions) • Relative test effort (e.g., as a percentage of development effort) • Defect discovery rate or defect removal rate (defects per unit time or per unit cost of testing) • Test case effectiveness (proportion of test cases which did detect faults) • Rework savings compared to test effort (based on benchmarks within the organization about the cost to correct defects at each stage of the lifecycle)	• Justify, refine and improve the extent and type of testing used in the organization • Improve the efficiency of the testing process • Estimate test effort required for future projects

Table 5.3 (continued)

Intended Goal	Possible Metrics	Example Use of those Metrics
Improving the predictability of system readiness for release	• Percentage of test cases attempted • Percentage of test cases passed	• Assess the completeness of testing and identify risks to project schedule
Improving the quality of rework activities (during project) or maintenance activities (post release)	• Defect backlog over time • Average age of open defects or average time to fix defects • Numbers of defects introduced during rework	• Assess the completeness of testing and identify risks to project schedule • Assess the stability of a product post release

5.3.2 Collection of Data

Once preferred metrics have been selected, the necessary base data can be identified, (e.g., defect numbers, hours of test effort, numbers of test cases). The base data should be measurable, necessary, relevant to the desired metrics, and valid.

Procedures for collecting base data should be included within the test plan or strategy and the importance of collecting the base data should be communicated to the test team members. The purpose of the procedure is to ensure consistency of approach when recording data, e.g.:

- Where a metric requires an assessment of defect severity, is there a consistent method for assessing this?
- Where a metric is “per unit time” is this elapsed time or time expended?
- Where a desired metric is “per 1000 lines of code,” do these include executable lines, data definition lines, comment lines, blank lines? Is it total lines on screen or total lines measured by separators?
- Where a metric includes data for defects found post-release, is this for a set time period after release or a set time period of actual operation or over the entire operational lifespan of the system?

Data can be collected manually or automatically, using a suitable test management tool. Automatic collection may have advantages not only in efficiency, but also because it avoids bias resulting from (deliberate or unconscious) recording errors, delays, or omissions.

5.3.3 Calculation of Metrics

Data can be analyzed and the desired metrics calculated either manually or automatically.

In many cases, metrics are directly measured (e.g., number of test cases passed) or easily calculated (e.g., rate of detection of defects). This section provides further detail for some of the more complex test metrics.

5.3.3.1 Test Coverage Metric

Test coverage is a percentage measuring how much an item of software has been exercised by tests.

Measuring Test Coverage

At the simplest level, test coverage could be the percentage of specified requirements for which evidence of verification exists. In this case, the activities involved in measuring test coverage for a specific test phase are:

- Determination of the overall required test coverage (based on both coverage of requirements and coverage of mitigations identified during risk assessment)
- Determination of the required test coverage for the test phase in question (which may include consideration of which tests are most appropriately conducted at this phase and what has already been covered in previous phases, including by suppliers)
- Cross referencing the proposed tests to ensure they meet the required coverage for the phase
- Measuring completion of the tests
- Measuring closure of any test incidents raised

At a more complex level, test coverage can consider execution of individual business process paths or program paths and statements:

- Has each statement been executed?
- Has each decision point been challenged such that each possible outcome has been tested at least once?
- Has each decision point been challenged such that each combination of conditions has been tested at least once?
- Has each possible path through the software been executed?

Making Use of the Test Coverage Matrix

Test coverage results can be used to:

- Justify leveraging of previous (e.g., supplier) testing and identify gaps where use cases may not have been covered and additional testing may be required
- Identify test redundancy and allow the testing process to be made more efficient
- Justify when a system is of an acceptable quality to be released (i.e., 100% critical functions have been tested)
- Provide evidence for coverage of GxP requirements, commensurate with the risk priority, i.e., 100% of critical GxP functions within a system's intended use; this can help to provide ease of presentation to regulators
- Understand and justify the use of testing benchmarks, such as software defects detected per test cycle

Test coverage also can be considered when seeking to explain why a defect was not detected and feed forward into improvements in the testing process:

- Has that particular statement ever been executed?
- How many times has that statement been executed?

- If the defect was tested, was the test coverage comprehensive enough? i.e., if the defect was found with function X, was this functionality originally covered? If yes, but only at a high level, could additional testing such as boundary testing have uncovered the defect?

- If the function was not tested originally, and the defect is critical, then either the risk priority or the level of test coverage assigned to that risk priority was inadequate. What needs to change for the future?

5.3.3.2 Defect Escape Rate Metric

An escape is a defect that was not discovered by the test team.

Escape rate metrics can be “end point” (i.e., defects escaping into the released product) or “in process” (i.e., defects escaping from one test phase to the next).

Escape rate results also can be subdivided into defects that were within the tested functionality (i.e., the rigor of testing was insufficient) and defects that escaped because the functionality was not tested (i.e., the test coverage was insufficient).

Measuring Escape Rate

The simplest escape rate is that of defects into the released environment.

$$\text{Escape rate} = \frac{\text{Defects reported post release}}{\text{Total defects found}}$$

Activities involved in this are:

- Logging of the number of defects found during all test phases

- Logging of the number of defects reported post release

The concept of escape rate also can be used for individual test phases, but here the emphasis is changed to look backwards from the defects identified in the current phase and measure how many of them ought to have been caught by each previous phase.

Example showing calculation of in-process escape rates as test data builds up:

- The in-process escape rate for each phase has been calculated as

$$\text{Escape rate} = \frac{\text{Defects reported post test phase which should have been caught by that phase}}{\text{Total defects which were caught or should have been caught by that phase}}$$

Table 5.4

Post Module Test		Previous phase where defects should have been caught				
	Total Defects	Module	Integration	FAT	SAT	None
Module Testing	30	N/A	N/A	N/A	N/A	30
Escape Rate for each Test Phase		Not yet known	Not yet known	Not yet known	Not yet known	N/A
Post INT Test		Previous phase where defects should have been caught				
	Total Defects	Module	Integration	FAT	SAT	None
Module Testing	30	N/A	N/A	N/A	N/A	30
Integration Testing	20	10	N/A	N/A	N/A	10
Escape Rate for each Test Phase		10 from 40 25%	Not yet known	Not yet known	Not yet known	N/A
Post FAT		Previous phase where defects should have been caught				
	Total Defects	Module	Integration	FAT	SAT	None
Module Testing	30	N/A	N/A	N/A	N/A	30
Integration Testing	20	10	N/A	N/A	N/A	10
FAT	10	1	8	N/A	N/A	1
Escape Rate for each Test Phase		11 from 41 27%	8 from 28 29%	Not yet known	Not yet known	N/A
Post SAT		Previous phase where defects should have been caught				
	Total Defects	Module	Integration	FAT	SAT	None
Module Testing	30	N/A	N/A	N/A	N/A	30
Integration Testing	20	10	N/A	N/A	N/A	10
FAT	10	1	8	N/A	N/A	1
SAT	5	0	1	1	N/A	3
Escape Rate for each Test Phase		11 from 41 27%	9 from 29 31%	1 from 11 9%	Not yet known	N/A

Table 5.4 (continued)

Release + 1 Year		Previous phase where defects should have been caught				
	Total Defects	Module	Integration	FAT	SAT	None
Module Testing	30	N/A	N/A	N/A	N/A	30
Integration Testing	20	10	N/A	N/A	N/A	10
FAT	10	1	8	N/A	N/A	1
SAT	5	0	1	1	N/A	3
Post-release	3	0	0	0	3	0
Escape Rate for each Test Phase		11 from 41 27%	9 from 29 31%	1 from 11 9%	3 from 8 38%	N/A
Escape Rate (end point)	3 from 68 4%					

Similar escape rates also might be derived for design review and code review activities.

Making Use of the Escape Rate Metric

Escape rate results can be used to:

- Justify the adequacy of previous test coverage
- Identify the relative effectiveness of different test phases and target improvement activity
- Measure the effectiveness of supplier pre-release testing and make risk-based judgments about how much prior testing to leverage

5.3.3.3 Phase Containment Metric

Measuring Phase Containment

Phase containment measures the opposite of escape rate – the proportion of defects which are successfully caught by the test team.

Metrics can be “in-progress” or “end point:”

$$\text{Phase Containment Effectiveness (PCE)} = \frac{\text{Defects caught within the phase}}{\text{Defects caught within the phase or found later}}$$

$$\text{Total Defect Containment Effectiveness (TDCE)} = \frac{\text{Defects caught within testing}}{\text{Defects caught within testing or found later}}$$

Table 5.5: Example Showing Calculation of Phase Containment as Test Data Builds Up

Post Module Test		Previous phase where defects should have been caught				
	Total Defects	Module	Integration	FAT	SAT	None
Module Testing	30	N/A	N/A	N/A	N/A	30
PCE for each Test Phase		Not yet known	Not yet known	Not yet known	Not yet known	N/A
Post INT Test		Previous phase where defects should have been caught				
	Total Defects	Module	Integration	FAT	SAT	None
Module Testing	30	N/A	N/A	N/A	N/A	30
Integration Testing	20	10	N/A	N/A	N/A	10
PCE for each Test Phase		30 from 40 75%	Not yet known	Not yet known	Not yet known	N/A
Post FAT		Previous phase where defects should have been caught				
	Total Defects	Module	Integration	FAT	SAT	None
Module Testing	30	N/A	N/A	N/A	N/A	30
Integration Testing	20	10	N/A	N/A	N/A	10
FAT	10	1	8	N/A	N/A	1
PCE for each Test Phase		30 from 41 73%	20 from 28 71%	Not yet known	Not yet known	N/A
Post SAT		Previous phase where defects should have been caught				
	Total Defects	Module	Integration	FAT	SAT	None
Module Testing	30	N/A	N/A	N/A	N/A	30
Integration Testing	20	10	N/A	N/A	N/A	10
FAT	10	1	8	N/A	N/A	1
SAT	5	0	1	1	N/A	3
PCE for each Test Phase		30 from 41 73%	20 from 29 69%	10 from 11 91%	Not yet known	N/A

Table 5.5: Example Showing Calculation of Phase Containment as Test Data Builds Up (continued)

Release + 1 Year		Previous phase where defects should have been caught				
	Total Defects	Module	Integration	FAT	SAT	None
Module Testing	30	N/A	N/A	N/A	N/A	30
Integration Testing	20	10	N/A	N/A	N/A	10
FAT	10	1	8	N/A	N/A	1
SAT	5	0	1	1	N/A	3
Post-release	3	0	0	0	3	0
PCE for each Test Phase		30 from 41 73%	20 from 29 69%	10 from 11 91%	5 from 8 62%	N/A
TDCE (end point)	65 from 68 96%					

Making Use of the Phase Containment Metric

Phase containment results are used in a similar way to escape rate results to:

- Justify the adequacy of previous test coverage
- Identify the relative effectiveness of different test phases and target improvement activity
- Measure the effectiveness of supplier pre-release testing and make risk-based judgments about how much prior testing to leverage

5.3.4 Analysis and Presentation of Results

The format of results for presentation should be consistent and easily understood.

Insight can be obtained through the use of graphs and charts to provide summary and trending of any test metrics collected. Visual methods are often more quickly understood than large tables of numbers and can provide valuable predictors of the quality of an application through its test phase.

5.4 Automated Tools for Planning and Management

5.4.1 Test Management Tools

Computerized test management tools can be useful for testing large or complex systems or in organizations where there is a need for on-going testing of software and/or computerized systems. Such tools generally improve the efficiency of test processes, by managing activities such as:

- Requirements traceability
- Test script development, review, and approval
- Test planning
- Test execution (including automatically running test cases)

- Test result and test evidence capture and management
- Test case review
- Test incident management
- Verification activities in non-linear development models (due to the reliance on regression testing)
- Overall test management, through the use of test monitoring tools

Test management tools can provide the ability to configure test process workflows and user roles and may significantly reduce the need for paper based test records. These tools also can provide test metrics, reports, and graphs. For use in the pharmaceutical and related industries, test management tools should be appropriately assessed and the integrity of test records should be assured (see Appendix T11 and Appendix D5 of GAMP® 5 [1]).

The use of test management tools is recommended for those organizations that are:

- Implementing large or complex systems where there are likely to be multiple rounds of regression testing during the life of the system (such as ERP, LIMS, or desktop client images)
- Developing expertise in the testing of computers and software across all areas of the business and where there is an opportunity to develop expertise in the use and support of a test management tool and achieve ROI across multiple projects
- Using non-linear development methodologies for system development

5.4.2 Automated Testing Tools

Automated test execution tools can expedite and improve the efficiency of the test execution activities. These tools vary in their implementation and use. They may be used at any level within the test strategy from module testing to user acceptance testing. The implementation and use of these tools should be discussed in test strategy documentation. Assessment and management of these tools should ensure their outputs are consistent and compliant with a regulated organization's procedures.

For further information, see Appendix T11 and Appendix D5 of GAMP® 5 [1].

6 Appendix T3 – Test Specifications, Cases, and Scripts

6.1 Test Specifications or Test Protocols

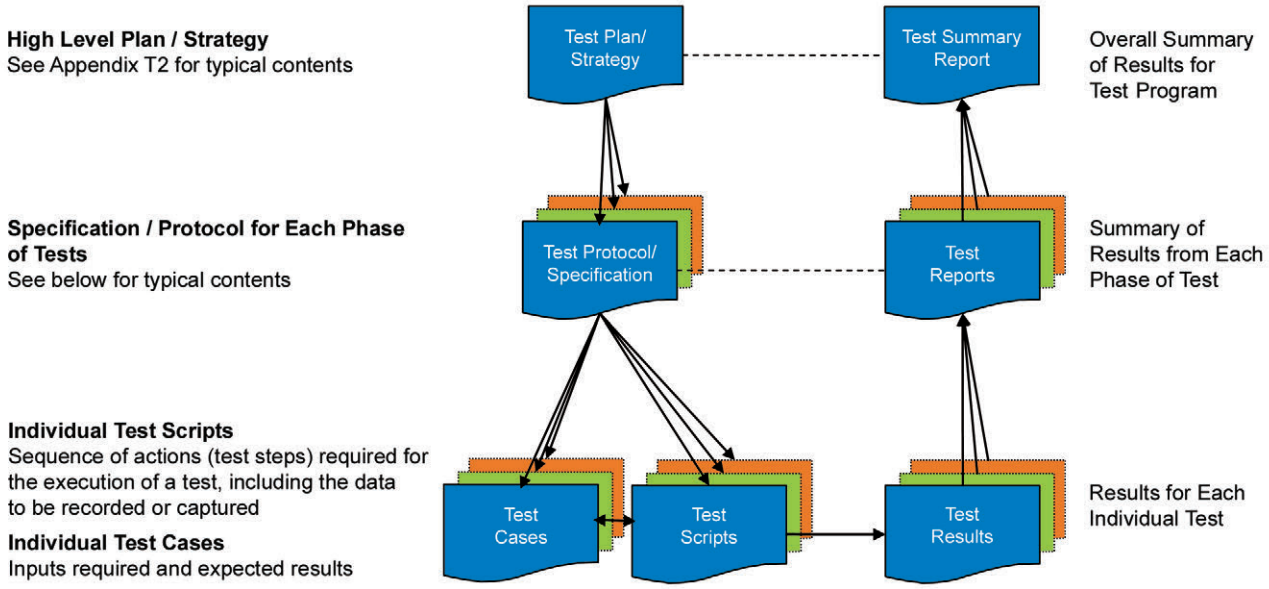
An outline of the approach to testing of a system should be defined within the project validation master plan/validation plan.

For a larger project, separate documents may be developed for:

- Test plan or test strategy
- Test specification or test protocol
- Test cases and test scripts

For a small project, these documents may be combined into a single document.

Figure 6.1: Typical Structure for Testing Documentation



A separate test specification or protocol may be written for each phase of testing and should cover the detail relevant to the test phase or test cycle. Testing should be considered during development of user requirements and design requirements and the test specifications should be developed early in the development life cycle.

The test specification should include details of:

- The scope of the test phase and how this relates to the overall test plan or strategy
- Test coverage within the phase, including:
  - Type of testing to be completed (e.g., black box, prototyping, inclusion of informal testing)
  - Linkage to requirements and to risk assessment
  - Sequencing and logical grouping of tests within the phase

- Specific areas not tested

- Any pre-requisites or dependencies for the phase (e.g., other test phases which must be complete)

- The test environment including recording details where necessary (refer to Appendix T4 for further detail on controlling test environments)

- Any phase-specific naming conventions or templates or formats (e.g., for test scripts and incident references)

- Any phase-specific procedures (e.g., for how pre-execution review, test results, test review, test reporting and release will be documented)

Test specifications or test protocols, test cases, and test scripts should be reviewed and where necessary, updated before the initial use or first re-use of automated testing.

For further information on testing in non-linear life cycles, see Appendix T10.

6.2 Test Cases and Test Scripts

Test Case

A test case is defined as a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to mitigate an identified risk or to verify compliance with a specific requirement.

Test Script

A test script specifies the sequence of actions required for the execution of one or more test cases. A test script should contain sufficient detail to enable consistent execution by different testers and between repetitions of tests.

Where automated testing is used, a test script may be a tabular set of instructions which can be executed routinely by the automated testing tool (similar to a manually executed test script) or it may be editable software code.

Description of the data to be recorded or captured may be included in either the test script or the test case.

Where a test case and test script are combined into a single test document, the combined document should provide details of the test inputs and expected results, in addition to the actions to be performed and the data to be recorded.

In order to establish the suitability of test cases and test scripts, an independent person (i.e., not the test script/case author) should informally execute a test (but not in a controlled environment) before finalizing the testing documentation and formal test execution. This can help to ensure that the test case/script is executable, is written clearly, and meets the test objective.

For automated test cases, the complexity of a test script is typically developed and built iteratively, e.g.:

- First recording and editing/commenting a script

- Replaying a script to ensure that it executes

- Using the script to run different test cases by setting up different inputs and expected results

When designing test scripts and test cases, the likelihood of re-use for regression testing (where a modular format for ease of re-use of test cases may be useful) should be considered.

6.2.1 Determination of Test Cases

Test planning involves selecting appropriate test cases from a large set of possibilities. The output from the system risk assessment may be used to select test appropriate cases by, e.g.:

- Choosing the rigor of testing according to the risk priority associated with a particular requirement or function (e.g., choosing to test only normal operation and defined fault paths for a medium risk priority module while testing each possible path through the code for a high risk priority function)
- Identifying each agreed mitigation for a risk scenario and defining test cases to ensure that these are effective

Before writing a test case (and associated test script), the test case author should:

- Understand the source document(s) and functionality to be tested and the associated test plan or strategy
- Understand the overall scope of testing and define the scope of the test case objective
- Understand the testing environment in terms of:
  - Correct hardware (peripherals and interfaces)
  - Software (validated tools, software configuration)
  - Data units (inputs, outputs, quality and quantity of data)
  - Procedures (especially for user acceptance testing)
  - People (training and experience)
- Understand any testing prerequisites that need to be met before testing and any interdependencies between tests
- Ensure that all input references from source documents are accounted for and justified as either tested, partially tested, or not tested
- Consider aspects of positive and negative testing
- Consider breaking the test objective down into smaller objectives that are easy to repeat, if required
- Ensure each test objective supports an area of scope and has acceptance criteria that clearly demonstrate correct operation
- Consider which tests will be good candidates for test automation or which may be used as the basis for performance/load test scripts

User testing also can be designed as scenario based testing.

6.2.2 Traceability of Test Cases

Test cases should be traceable to requirements or design statements in the controlling specification and to the risk assessment which determined the need to test the given scenario. The requirements, design statements, and functional risk assessment should be documented either in the test case itself or via a separate traceability matrix (or equivalent).

The use of a tool to manage traceability should be considered, particularly for complex test cases.

The acceptance criteria for a test case should be clearly defined such that the overall pass/fail determination is objective, i.e., the output from the test case can be compared clearly against acceptance criteria to determine the pass/fail status. Requirements and design statements should be unambiguous and testable.

A test objective may be stated in terms relative to requirements or design statements.

Although the expected results of an individual test step may not be fulfilled because of test script or test execution errors, it may be still possible to determine that the overall acceptance criteria have been met.

6.2.2.1 Design of Test Cases

The design of test cases should leverage existing testing, such as supplier testing or previous testing by the regulated organization.

Where the intended use remains the same, test paths, branches, functions, etc., that have previously been tested may not need further testing. Where there is a change in use or where the maturity of the product and/or supplier testing is considered insufficient, testing should be increased accordingly. Risk assessment should be used to determine and document the required level of testing.

Equivalence class analysis and analysis of time dependencies techniques can be helpful in designing effective test cases.

Equivalence Class Analysis for Structural Testing

Using the concept of equivalence classes, it is assumed that any input value from an equivalence class will test a program as effectively as any other value from that range. Using a single value from an equivalence class rather than hundreds of values from a range can be sufficient for a test.

Design documentation should help to identify equivalences.

Tables 6.1 and 6.2 list typical equivalence class assumptions.

Table 6.1: Equivalence Class Assumptions for Structural Testing

Statement Coverage	Input combinations which result in execution of a particular line of code are assumed to be equivalent.
Decision (Branch) Coverage	Input combinations which result in the same program branch or decision are assumed to be equivalent. For an example of branch coverage testing. See Appendix T1.
Condition Coverage	Input combinations which result in the same outcome for a particular condition within a decision are assumed to be equivalent.
Multi-Condition Coverage	Input combinations which result in the same outcome for all conditions within a decision are assumed to be equivalent.
Loop Coverage	Input combinations which result in execution of the same program loop are assumed to be equivalent.
Path Coverage	Input combinations which result in exactly the same path through a program segment from start to finish are assumed to be equivalent.
Data Flow Coverage	Only input combinations which result in the same data flow as well as the same path through a program segment are assumed to be equivalent.

Table 6.2: Equivalence Class Assumptions for Functional Testing

Normal (Positive) Case Testing	Selection of test cases from within the expected input values
Invalid (Negative) Case Testing	Selection of test cases from within equivalence classes such as: • Out of range inputs • Wrongly formatted inputs • Repeated inputs
Special Case Testing	Selection of test cases from within equivalence classes such as: • Boundary or limit values (e.g., maxima and minima) • Singularities (e.g., zero, null strings)
Output Testing	Selection of test cases from the domain of input combinations that lead to a specific output
Input Combination Testing	Selection of test cases made up of combinations of inputs either at random or such that faults are likely to be revealed, e.g., “worst case” testing combining individually valid conditions and inputs so that it forces a system to operate at its limits – or beyond

Analysis of Time Dependencies

Where input combinations are required as part of a test case, the order in which inputs occur may affect the program action (sometimes referred to as “race conditions”). Considerations for designing test cases are shown in Table 6.3.

Table 6.3: Test Case Design Considerations

Asynchronous inputs to a decision	Where inputs to a decision can occur asynchronously, test cases may need to include instructions for the order of inputs.
Handshaking between modules	Where data is passed from one module to another, test cases may need to include late or early availability of data.
Transitions out of a state	Where more than one exit transition is possible from a state, test cases may need to include input combinations which result in more than one transition being possible at once in order to check that program behavior is deterministic.
Modules updating the same data structures	Where more than one module updates a particular data structure, test cases may need to include conditions in which more than one module attempts to update data simultaneously in order to check that data corruption does not occur. (Although this is not good design practice, it may be necessary to consider this for testing legacy systems.)
Modules locking data structures	Where modules lock data structures during updates, test cases may need to include conditions in which more than one module attempts to update data simultaneously in order to check that deadlock does not occur.

6.2.2.2 Design of Test Scripts

Test scripts can be written so that:

- Results are recorded directly onto a copy of an approved test script
- Data is recorded onto a separate test results sheet

Where results are to be recorded directly onto an approved test script, sufficient space should be provided for:

- Recording the test results
- The tester to sign and date the test execution
- The reviewer to sign during post execution review (this may be on the test script itself or on a separate summary sheet if preferred) and for recording the test run number (or iteration number)

If further runs of a test are required following a test incident, a copy of the master test script should be made and clearly marked with the run number.

Where separate test results sheets are used to record results, space should be provided for:

- Including a clear cross reference to the test case and/or test script
- Space to record the test run number (or iteration number)
- The tester to sign and date the test execution
- The reviewer to sign during post execution review

Where the test case and/or test script contain all necessary test information, a pre-execution review of separate test results sheets should not be necessary.

Traceability of Test Scripts

Test scripts should be traceable back to the requirement or design statement in the controlling specification and to the functional risk assessment which determined the need to test the scenario. This traceability can be documented either in the test script (via a referenced test case) or via a separate requirements traceability matrix (or equivalent).

Content of Test Scripts

Test scripts should contain sufficient detail of tests to enable consistent re-execution of a test.

Each test script should be:

- Specific
- Measurable
- Achievable
- Realistic
- Time-framed

Each test script should, where possible, include the following as shown in Table 6.4.

Table 6.4

Item	Description/Comment
Unique test reference	
Cross reference to controlling specification	May be covered in a separate traceability matrix
Title of test	
Description of test, including the test objective	Individual test objectives should be kept to a manageable size. For example, if a 100-step test script can be divided into five logical sections with separate objectives defined for each logical section and only one logical section needs re-executing, the number of steps to be re-executed can be reduced significantly.
Pre-test steps, including any test pre-requisites or set-up	This may include cautionary measures, test constraints, or entry criteria following the end of a previous test. Pre-requisites for the complete test phase are not usually repeated in each test script.
Test steps	A step-by-step description of the actions to be performed by testers. Where data is required, instructions for entry of variables should be provided rather than specific values for variables, e.g., “Enter today’s date” or “Enter today’s date plus 7 days” rather than “Enter 29-Feb-04.” Where the creation of new data is not required, wording such as “Ensure that the following data exists” or “find/create” provides flexibility. For example, material created according to the characteristics detailed in the test case for the first iteration of a test could be re-used in additional executions, rather than creating more material. Test steps can lead to an outcome or result or may just be navigational or preparatory. An illustrative example of a classification scheme based on this concept is given in Table 6.5. Example test script formats applying the scheme are also provided.
Expected results/acceptance criteria	The defined set of expected results and criteria that should be met for the test to be deemed to have passed.
Data to be recorded	A description of the test specific data to be collected and recorded. This can be input, output, or descriptive data and should include serial numbers of any test equipment used and supporting calibration certificates where necessary. Screen shots or printouts may be used to reduce the level of effort involved in manually recording data. The data recorded should be sufficient to allow an independent reviewer to determine whether a test met the defined objective. Any required screen shots and other electronic documentation should be specified.
Post-test actions	This is an optional section which details those actions required to return the system to a known state, e.g., resetting process parameters, putting the system in a safe state, or rebooting the system

Where the test script is used to record results, sufficient space is required to allow entry of results in accordance with Appendix T6 Test Results Recording and Reviewing, including, where appropriate:

- Result of each step

- Overall result of test
- Signatures and dates of tester(s) and reviewer(s)
- Data to be recorded
- Cross-reference to documentation recording test issues and resolution

It should be possible to ascertain the number of times a test has been executed, for example, following a test incident. One method for achieving this would be to allow entry of the run number on the test script.

Automated test scripts should contain or reference information similar to that in manual test scripts. The information may be included in the test script meta-data as comments in a coded script, or by reference to a test case containing the necessary information. The design of an automated test script should clearly identify which steps or instructions need test results to be recorded.

6.2.2.3 Pre-Execution Review of Test Documentation

Test specifications or test protocols, test cases, and test scripts should be reviewed and approved before starting testing. Reviewers of the test documentation should:

- Not be authors of the test specifications or test protocols, test cases, and test scripts
- Have the appropriate subject matter expertise
- Represent appropriate functions as defined by an organization's test policy

The review process should be consistent and the outcome of the review process should be recorded against each test case or test script. It may be helpful to create a "review checklist" template based on the test script contents, to assist in reviewing test specifications or test protocols, test cases, and test scripts.

The review process should be scaled to the risk priority or complexity of the test scenario. For example, a high risk priority or complex scenarios may require a formal walkthrough of a test case or test script. A low risk priority or simple scenarios may require a less formal peer review. For development/informal testing, less detailed documentation of the review process may be appropriate. For final acceptance tests, more formal documentation of the review process may be appropriate. The review process should be defined to ensure that:

- Reviews effectively evaluate deliverables against standards and requirements, identify issues, and propose required corrective action
- Reviews are planned systematically and are performed at appropriate points throughout the life cycle of the system
- Reviews are performed by appropriate Subject Matter Experts (SMEs)
- Individuals performing reviews are identified
- Relevant documentation of the organization is used
- Relevant document control policies of the organization are followed

6.3 Example Test Step Classification Scheme

6.3.1 Introduction

Some test steps lead to an outcome or result, while some are purely navigational or preparatory. This section provides an example of a test step classification scheme based on this concept. This example is intended to be illustrative only, and is neither definitive nor mandatory.

6.3.2 Example Types of Steps

Test script steps may be classified as follows:

- Basic Function Step – provides instructions for entry or exit conditions
- Non-Proving Navigation or Set Up Step – required to navigate to Proving Step
- Supporting Step – Evidence Required – provides evidence of the correct execution of the test script
- Proving Step – demonstrates that the test objective has been met

Table 6.5: Example Test Script Content for Different Test Step Types

Test Step Type	Requirement Reference	Action	Expected Result	Actual Result	Pass/Fail	Comments
Basic Function Step	N/A	Contains tester instructions	NPS	NPS	Fail if abnormal operation of system observed	No impact on Acceptance Criteria Typically Log On or Off system, or start or end a transaction
Non-Proving Navigation or Set Up Step	N/A	Contains tester instructions	Describe what the tester should observe	NPS (and possibly supporting data)	Fail if observation does not match Expected Result	No impact on Acceptance Criteria Typically data set up or progress system to a required state
Supporting Step – Evidence Required	N/A	Contains tester instructions	Describe what the tester should observe	Response required from tester and/or Exhibit Numbers	Fail if observation does not match Expected Result	Potential impact on Acceptance Criteria Key progress of process being shown or evidence being collected which will impact a Proving Step
Proving Step	Requirement Reference	Contains tester instructions	Describe what the tester should observe	Response required from tester and/or Exhibit Numbers	Fail if observation does not match Expected Result	Important in demonstrating that Acceptance Criteria have been met
N/A = Not Applicable, NPS = Non-Proving Step						

6.4 Example Test Scripts and Cases

Tables 6.6. to 6.9 show examples of manual test cases and test scripts. These provide the option of recording results directly on to a copy of the test script or using a separate test results sheet. In many cases, the execution of such manual test cases and test scripts will be the basis of recording editable automated test scripts.

Table 6.6: Example Test Script and Test Case Combined, Integral Test Results

Test Script Reference		Data-Batch-001	**Title**	Test Script for Batch Number Input Transaction		
Objective		To show that the Batch Number Input Transaction works successfully				
Acceptance Criteria		Valid batch number (positive integer) can be entered. Invalid batch numbers are rejected.				
Run Number			**Start Date and Time**		**End Date and Time**	
Pre-Requisites: Log on available with batch number input transaction capability. There are no specific data requirements.						
Test Step	**Requirement Reference**	**Action**	**Data to be Recorded**	**Expected Result**	**Actual Result/ Incident Record**	**Pass/Fail**
1.1	N/A	Log on and Enter Batch Number Input Transaction	Log on used	Batch Number Input Transaction successfully entered	(NPS)	
1.2	SR_001	Enter 0 (zero)	Screen Print Exhibit 1.2	Rejected with error message		
1.3	SR_001	Enter negative integer (use -123)	Screen Print Exhibit 1.3	Rejected with error message		
1.4	SR_001	Enter non-integer (use 12.3)	Screen Print Exhibit 1.4	Rejected with error message		
1.5	SR_001	Enter non-numeric value (use Abc)	Screen Print Exhibit 1.5	Rejected with error message		
1.6	SR_001	Enter valid batch number (use 123)	Screen Print Exhibit 1.6	Value Accepted		
Post Test Actions: Abort the batch ready for the start of the next test						
Comments:						
Test Outcome			**Incident Report Reference**			
Tester Signature		**Date**	**Review Signature**		**Date**	

Table 6.7: Example Test Script and Test Case Combined, Separate Results Sheet

Test Script Reference	Data-Batch-001	**Title**	Test Script for Batch Number Input Transaction
Objective	To show that the Batch Number Input Transaction works successfully		
Acceptance Criteria	Valid batch number (positive integer) can be entered. Invalid batch numbers are rejected.		
Pre-Requisites: Log on available with batch number input transaction capability. There are no specific data requirements.			

Test Step	**Requirement Reference**	**Action**	**Data to be Recorded**	**Expected Result**
1.1	N/A	Log on and Enter Batch Number Input Transaction	Log on used	Batch Number Input Transaction successfully entered
1.2	SR_001	Enter 0 (zero)	Screen Print Exhibit 1.2	Rejected with error message
1.3	SR_001	Enter negative integer (use -123)	Screen Print Exhibit 1.3	Rejected with error message
1.4	SR_001	Enter non-integer (use 12.3)	Screen Print Exhibit 1.4	Rejected with error message
1.5	SR_001	Enter non-numeric value (use Abc)	Screen Print Exhibit 1.5	Rejected with error message
1.6	SR_001	Enter valid batch number (use 123)	Screen Print Exhibit 1.6	Value Accepted

Post Test Actions:
Abort the batch ready for the start of the next test

Test Script Reference	Data-Batch-001	**Run Number**	
Start Date and Time		**End Date and Time**	

Test Step	**Data to be Recorded**	**Actual Result/ Incident Record**	**Pass/ Fail**
1.1	Log on used	(NPS)	
1.2	Screen Print Exhibit 1.2		
1.3	Screen Print Exhibit 1.3		
1.4	Screen Print Exhibit 1.4		
1.5	Screen Print Exhibit 1.5		
1.6	Screen Print Exhibit 1.6		
Comments			
Test Outcome		**Incident Report Reference**	
Tester Signature	**Date**	**Review Signature**	**Date**

Table 6.8: Example Separate Test Script and Test Case, Integral Test Results

Test Script Reference	Data-Batch-001	**Title**	Test Script for Batch Number Input Transaction			
Objective	To show that the Batch Number Input Transaction works successfully					
Acceptance Criteria	Valid batch number (positive integer) can be entered. Invalid batch numbers are rejected.					
Run Number		**Start Date and Time**		**End Date and Time**		
Pre-Requisites: Log on available with batch number input transaction capability. There are no specific data requirements.						
Test Step	**Requirement Reference**	**Action**	**Data to be Recorded**	**Expected Result**	**Actual Result/ Incident Record**	**Pass/Fail**
1.1	N/A	Log on and Enter Batch Number Input Transaction	Log on used	Batch Number Input Transaction successfully entered	(NPS)	
1.2	SR_001	Enter batch number using text cases 1 to 5	Screen print Exhibit 1.2n where n = test case number	As detailed in test case definition	Record on test case definition	
Post Test Actions: Abort the batch ready for the start of the next test						
Comments:						
Test Outcome		**Incident Report Reference**				
Tester Signature	**Date**	**Review Signature**		**Date**		
Test Case Definition						
Test Script Reference	Data-Batch-001	**Title**	Test Script for Batch Number Input Transaction	**Run Number**		
Test Case	**Description**	**Characteristics**	**Expected Result**	**Actual Result/ Incident Record**	**Pass/Fail**	
1	Batch number of zero	0	Rejected with error message			
2	Negative integer batch number	-123	Rejected with error message			
3	Non-integer batch number	12.3	Rejected with error message			
4	Non-numeric batch number	Abc	Rejected with error message			
5	Valid batch number	123	Value accepted			

Table 6.9: Separate Test Script and Test Case, Separate Results Sheet

Test Script Reference	Data-Batch-001	**Title**	Test Script for Batch Number Input Transaction
Objective	To show that the Batch Number Input Transaction works successfully		
Acceptance Criteria	Valid batch number (positive integer) can be entered. Invalid batch numbers are rejected.		
Pre-Requisites: Log on available with batch number input transaction capability. There are no specific data requirements.			

Test Step	**Requirement Reference**	**Action**	**Data to be Recorded**	**Expected Result**
1.1	N/A	Log on and Enter Batch Number Input Transaction	Log on used	Batch Number Input Transaction successfully entered
1.2	SR_001	Enter batch number using test cases 1 to 5	Screen print Exhibit 1.2.n where n=test case	As detailed in test case definition
Post Test Actions: Abort the batch ready for the start of the next test				

Test Case Definition			
Test Script Ref.	Data-Batch-001	**Title**	Test Script for Batch Number Input Transaction

Test Case	**Description**	**Characteristics**	**Expected Result**
1	Batch number of zero	0	Rejected with error message
2	Negative integer batch number	-123	Rejected with error message
3	Non-integer batch number	12.3	Rejected with error message
4	Non-numeric batch number	Abc	Rejected with error message
5	Valid batch number	123	Value accepted

Test Script Reference	Data-Batch-001	**Run Number**	
Start Date and Time		**End Date and Time**	

Test Step	**Test Case**	**Data to be Recorded**	**Actual Result/ Incident Record**	**Pass/ Fail**
1.1	N/A	Log on used	(NPS)	
1.2	1	Screen Print Exhibit 1.2.1		
1.2	2	Screen Print Exhibit 1.2.2		
1.2	3	Screen Print Exhibit 1.2.3		
1.2	4	Screen Print Exhibit 1.2.4		
1.2	5	Screen Print Exhibit 1.2.5		

Comments

Test Outcome		**Incident Report Reference**	
Tester Signature	**Date**	**Review Signature**	**Date**

7 Appendix T4 – Test Environments

7.1 Introduction

The test plan or test strategy should consider and define the environments required for testing. For a typical Category 3 (see GAMP® 5 [1]) system, there will only be one environment. For more complex systems, testing may take place in different environments during a project, which may include:

- Development environment where prototyping or programming takes place
- Testing environment where formal testing is performed
- Operational environment where the system is in its target environment

The complete set of test environments used during the development and verification of a system may be referred to as the test landscape.

Formal tests may be performed either in the operational environment (where test records should be clearly distinguishable from production records or where test records can be archived prior to operational use) or in a separate test environment. Test documents should specify which environment to use. When using test environments, the test strategy chosen should justify the equivalency of test results, i.e., justify that similar results would have been achieved in the operational environment.

Formal tests executed at the supplier premises should take place in a dedicated supplier controlled test environment. Formal tests should be executed only in environments under configuration management.

The appropriate test environment should be determined based on the stage of the system life cycle and the outputs of risk assessments. Where computer based test tools are used, they should be considered as part of the test environment. Test tools and test software environment components are normally considered to be GAMP® Software Category 1 (see GAMP® 5 [1]).

7.1.1 General Considerations

The test planning process should take into account differences between the proposed test environment and the operational environment. Where these differences impact on the ability of tests to cover the identified requirements or risk scenarios, this should be detailed in the test specification or test protocol. If necessary, additional tests should be planned for the operational environment in order to ensure full coverage.

The test environment for some systems may not be adequately representative of the dynamic response of the process or sensors. Additional tests should be planned for the operational environment for functionality which could be impacted by process dynamics.

The test environment should be controlled and recorded to a level of detail that would allow it to be re-constructed if necessary. Test environment control includes:

- System hardware and software components
- Test hardware (versions, serial numbers as appropriate)
- Test software (version control of any simulators/emulators)
- Test data (backup and version control of any test data sets, test recipes, etc.)
- Test user accounts

Creation of a test environment may be achieved efficiently by, e.g., using a virtual machine running a clone of the operational environment, but this will still require control of the software under test, test data, user accounts, etc.

Where test hardware, software, data, and user accounts are applied in such a way that they may appear in the operational environment, controls should exist to ensure that they can either be removed cleanly or be isolated from use (either logically or in time). The application and removal activities can be managed under a single change control.

Where a temporary change to the test environment is required to facilitate the execution of specific tests, both the change and the removal of the change should be formally documented procedurally, or within the test specification or test protocol, or under change control.

Environments used for prototyping should be isolated from formal development, testing, and operational environments. Controls should be established to prevent software that has been informally developed from being released into formal environments.

It may be undesirable to conduct all testing on the final "as installed" operational environment due to the potential impact on patient safety, product quality, and data integrity if testing activities or outputs interfere with operations or cause confusion. In this situation, it will be necessary to create a separate test environment. This is particularly the case for:

- Situations where data produced as part of the test could not be adequately separated from production data
- Situations where testing might prove destructive to the operational environment (e.g., stress tests on the infrastructure/platform, such as denial of service, message overflows, or virus attacks)

However, there are circumstances where some tests have to be deferred to or are better performed in the operational environment, e.g.:

- Requirements and performance testing
- Tests affected by process dynamics, system loading, or business cycles
- Tests which could not be performed at earlier test phases due to lack of availability of infrastructure (e.g., communications networks) in the test environment
- Tests which could not be performed at earlier test phases due to lack of availability of specific interfaces in the test environment (e.g., because items are being produced by different suppliers)
- Tests which could not be performed at earlier test phases due to confidentiality issues which prevent testers from accessing suitable data outside of the operational environment

Test planning should address which areas of testing will need to take place in the operational environment and the controls required to allow this to be done.

The progression of a software build from a development environment through to an operational environment depends on the risk associated with the system being installed or the change being made, and on factors such as the ease with which the modification could be removed from the system if necessary. For example,

- It may be acceptable to apply a change of parameter (e.g., Proportional Integral Derivative (PID) controller tuning or alarm threshold) directly to the operational environment (a process controller) because the change has a low risk priority and all traces can be easily removed by re-setting the parameter to its original value.

- A change to a custom data processing module in a large business system may, however, require progression from a development environment, to a test environment, to a regression environment and then to the operational environment. This may be because the change is high risk, and even if the original module could be restored easily, the data from the test may remain in the operational environment.

There should be a controlled process for promoting code and configurations from one test environment to another and for ensuring that changes made in a later environment are returned to any previous environment that is still in use for subsequent development or verification activities. Controls should be established to prevent the release of the code and configuration under development into the operational environment before completion of testing.

7.2 Hardware Environment

Hardware test environment components may be standard or custom, and may be inherent to the system or be a separate test system. They may include components that represent part of an operational environment, where it is not feasible to include a specific element of the operational system in the test environment.

Table 7.1: Examples of Hardware Test Environment Components

Type of Hardware Test Environment Component	Operational Environment Represented	Separate Test System
Standard	Network hub and cabling used to connect together equipment for test	Signal injection/measuring equipment
Custom	Test environment copy of a custom component that is already in use in the operational environment	Custom built hardware simulator

7.2.1 Representative Hardware Test Environment

The hardware environment should be as representative as possible of the operational environment. It may not be possible, necessary, or desirable for the test environment to be identical to the operational environment due to technical, logistical, or financial constraints. The significance and impact of the differences in the test environment and the operational environment should be assessed, e.g.:

- If the test environment uses a standard network component of the same type as the operational environment, the substitution is unlikely to invalidate the test results in relation to the operational environment.
- If the test environment uses short patch cables for the network cabling even though the operational environment has cable runs close to the maximum recommended length, different network behavior may occur and additional tests on site may be needed to prove the network performance.
- If the test environment is representing a large enterprise system distributed on multiple processors with massive memory and storage, it is unlikely that the test system will have similar capacity due to logistical and economic constraints. The test environment may comprise a small number of clients on a single server and again, additional tests may be required on site to prove operation under load.

7.2.2 Control of Hardware Test Environment

For standard hardware test environments components, the manufacturer's reference number and the serial number should be recorded.

For custom hardware test environment components, the version of the item and its controlling specification also should be recorded.

For all test hardware, any applicable calibration status should be recorded in the context of the specific test.

7.2.3 Removal from Operational Environment

If test hardware is added in such as way that it may appear in the operational environment, this should be documented as a temporary modification to the operational system. Removal of the temporary modification should be documented as well. These activities should be performed under change control.

7.3 Test Software

An appropriate software test environment is required to test part or all of a system under test. The software test environment may consist of:

- Software representing that which will be used in the operational environment
- Separate test software or systems used to support, automate, or manage the testing activities

Examples of software test environment components are provided.

Table 7.2: Examples of Software Test Environment Components

Type of Component	Representing Operational Environment	Separate Test System
Standard	Operating system running on test environment PC	Operating system of PC used to represent data collection PC to which files will be archived in operational environment
	Standard software package running on test environment PC	Standard data collection package used to collect test data or standard tool used to force input and output values
Configured/Custom	Test environment copy of a configured module that is already in use in the operational environment	Simulation package configured to imitate feedback on valve position
	Test environment copy of a custom module that is already in use in the operational environment	Custom coded simulation used to represent temperature and pressure conditions in a vessel

Components within the separate test system should be appropriately assessed and controlled according to their complexity, novelty, and the risk priority of the functionality being tested.

7.3.1 Representative of Software Test Environment

The software environment should be as representative as possible of the operational environment for formal functional, performance, and regression testing. For earlier test phases, it may not be possible, necessary, or desirable for the test environment to be identical to the operational environment due to technical, logistical, or financial constraints. It may be necessary to use test harnesses or breakpoints to inject data at particular points in the processing logic. The test setup to achieve this is unlikely to be representative of operational conditions, especially for dynamic aspects. The significance and impact of the differences in the test environment and the operational environment should be assessed, e.g.:

- If the test environment uses a process controller running the same firmware version as the operational environment, the substitution is unlikely to invalidate the test results in relation to the operational environment.

- If the test environment simulates a particular interface, there is a possibility that different timing factors or process dynamics could affect operation so that the interface behaves differently during testing and in the operational environment and additional testing of the operational interface may be appropriate.

- If the test environment has a different implementation of operating system software (a different version, or a virtualized environment), any impact should be assessed and any appropriate additional tests introduced for the operating environment.

7.3.2 Control of Software Test Environment

For standard software components, the manufacturer's reference and the version number (including installed service packs and patches) should be recorded. Any configuration or set-up parameters should be controlled.

For configured or custom software components, the component should be placed under configuration management and the version in use recorded.

Any separate test software or systems should be assessed and controlled for the intended testing activities.

7.3.3 Removal from Operational Environment

If test software can appear in the operational environment, this should be documented as a temporary modification to the operational system. Removal of the temporary modification also should be documented. These activities should be performed under change control.

7.4 Test Data Sets

The required quantity and coverage of data sets to be created or generated should be based on the risk scenarios identified in a risk management process.

Test data sets can be created by copying operational data. If confidential operational data is involved it should be made anonymous. Data protection controls should not be compromised.

Generated data sets may be used where the test environment does not permit the use of real data for reasons of availability or confidentiality, or where the real data are not sufficiently extensive to cover certain test types (i.e., challenge testing at boundary conditions or stress testing). Another reason for generating data sets is to ensure known characteristics, e.g., number of occurrences of data items and values.

Data sets for unit testing are more likely to be generated, as these are likely to be smaller and more focused. Operational data may not include all necessary ranges of values for this level of testing, e.g., boundary testing, or limit testing.

It is desirable to be able to save and restore datasets to:

- Speed up test preparation

- Ensure consistency between test runs

- Efficiently exploit automated testing

- Facilitate the use of large data sets

- Encourage more frequent testing, especially regression testing

The mechanisms used for saving and restoring data sets should be appropriately assessed and controlled.

Most computerized test management tools provide the capability to manage test data and recorded results. Test data sets can be used by automated test scripts to parameterize a test execution and automated test tools also can automatically record actual results.

7.4.1 Verification of Data Sets

Data sets should be formally verified for their suitability to confirm the correct functioning, or otherwise, of the software to be tested. The verification method should be independent of the software. Verification could be by manual inspection, review, verified queries, or other appropriate mechanism. The software to be tested should not be used to verify the data that will be used to verify the software itself. The data set should be designed to challenge the software, not to enable it to pass. Data sets should be derived from design specifications in accordance with stated test objectives.

7.4.2 Representative Test Environment

For formal functional, performance, and regression testing, test data should be as representative as possible of the actual data to be operated on in terms of both volume of data and range of possible values (including invalid entries to check that these are correctly handled).

The test specification or test protocol should detail differences between the proposed test data and the expected actual data. Where these differences impact on the ability of tests to cover the identified requirements or risk scenarios, this also should be detailed in the test specification or test protocol. If necessary, additional tests should be planned for the operational environment in order to ensure full coverage.

For example, if injection of test data is not available and the test environment has gathered only a few weeks of data, but the operational environment may eventually accumulate 10 years of data; there is a possibility that different timing factors or system dynamics could affect operation in the operational environment. Regular performance monitoring of the operational environment may be appropriate.

7.4.3 Control of Test Environment

Test data sets should be placed under configuration management and the version in use recorded.

For automatically generated data, it may be appropriate to also assess and control the utility used for the generation of data (as well as the test data set).

7.4.4 Removal from Operational Environment

If test data can appear in the operational environment, this should be documented as a temporary modification to the operational system. Removal of the temporary modification also should be documented. These activities should be performed under change control.

If the operational environment possesses automatic audit trailing, it should be recognized that audit trail entries from the testing process will remain. This should be recorded in the relevant test documentation.

7.5 Test User Accounts

Test user accounts may permit testers to access system at different levels and may help to ensure that activities carried out during testing are easily identified within a resulting audit trail.

The management of user accounts in computerized test management tools should be considered.

7.5.1 Representative Test Environment

Where test user accounts are used, these should be set up to represent each group of users within the system, and include corresponding authorizations. For a multi-lingual system, test user accounts using foreign character sets should be included.

Where existing individual accounts are used for testing, representatives from each group of users should be included.

7.5.2 Control of Test Environment

Where test user accounts are used, the set-up of the accounts should be retained as part of the test documentation.

Where there are issues of data confidentiality, controls should be exercised to ensure that the use of test accounts does not cause breaches of confidentiality.

7.5.3 Removal from Operational Environment

If test user accounts can appear in the operational environment, this should be documented as a temporary modification to the operational system. Removal of the temporary modification also should be documented. These activities should be performed under change control.

7.6 Test Documentation

The test environment should include the documentation used during testing. This should include the test documentation (test plans and strategies, protocols and test specifications, test cases, and test scripts) and the controlling design specifications. It also may include operating procedures such as SOPs.

The test documentation should be controlled and recorded to a level of detail that allows it to be retrieved as part of later review of the test results. This control would, as a minimum, include the recording of current document version levels. It may be beneficial to keep full working copies of all relevant documentation during testing, as this allows any changes which are found to be necessary during testing to be marked up or formally recorded.

It should be possible to use the test documentation to reproduce the test conditions so that testing can be repeated if necessary, especially at a later date, e.g., during the operational phase.

7.7 Managing Test Environments

Test landscapes can vary considerably in size and complexity depending on the scale of the software development method. For example, a small packaged equipment supplier developing a PLC based “one off” system may need only a single physical test environment for development and formal testing. The necessary separation of activities is achieved by setting up the environment sequentially or through the use of appropriate configuration management.

When developing systems with multiple suppliers, a number of local and interim test environments may be established to facilitate unit and module testing. Management of these local environments may be performed manually, as the potential benefits of tools may not justify the necessary investment.

As the system components are brought together, more complex test environments may be required to support the module integration and formal testing. Other situations requiring complex test environments may include:

- Non-linear life cycles
- Prototyping
- Parallel implementation

The appropriate tools should be used to ensure the correct alignment of large test environments, code, test data, and documentation.

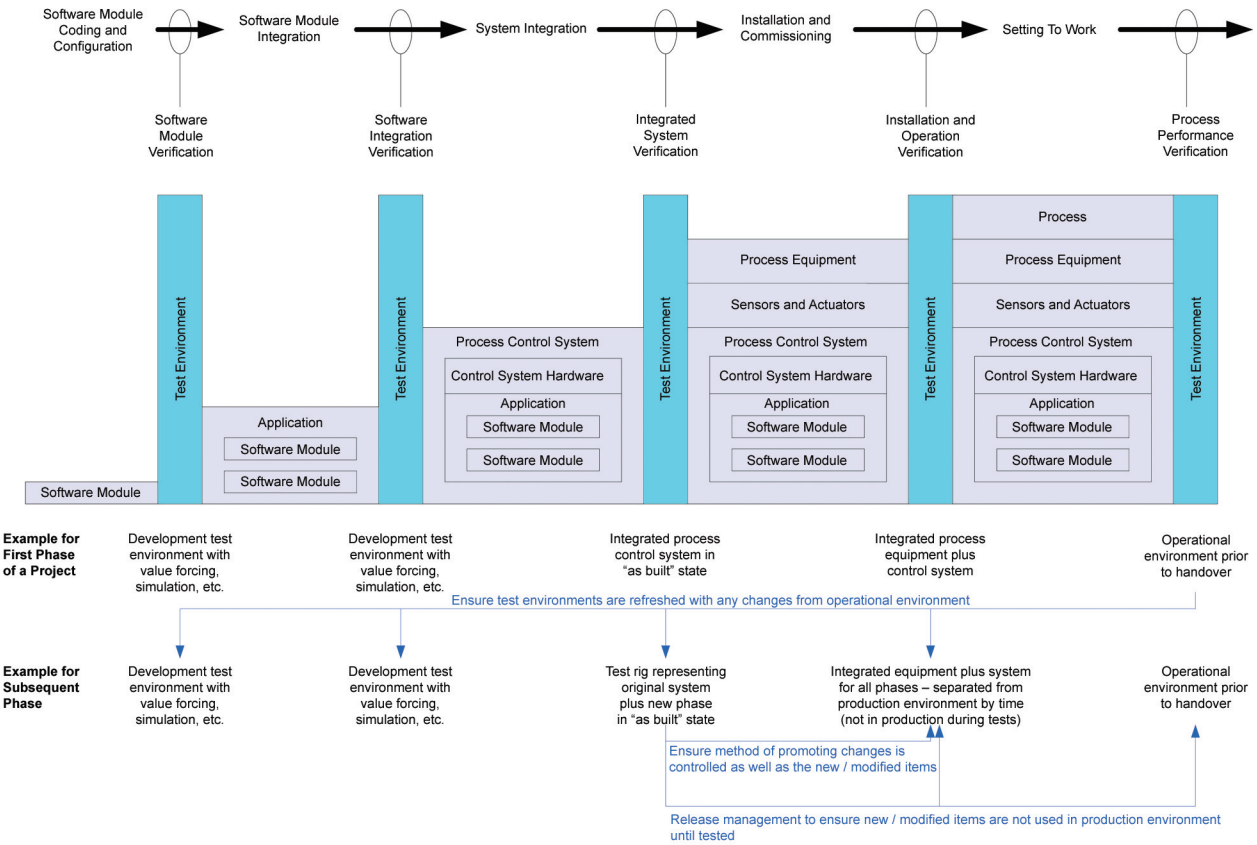
An example of a test landscape for a process control system is illustrated in Figure 7.1.

During development and early operational life of a system, the volumes of data and users may be sufficiently small for the testing environment to be adequate for performing load and performance testing. As the system matures, additional environments may be required for load and performance testing.

A separate environment for regression testing may be required.

The expansion of a test environment to incorporate additional environments should be managed in a controlled and verifiable manner.

Figure 7.1: Test Landscape for a Process Control System



The process for moving developments and data sets through the test environment should be clearly defined and controlled with acceptance criteria for promoting or demoting developments, see Figure 7.2.

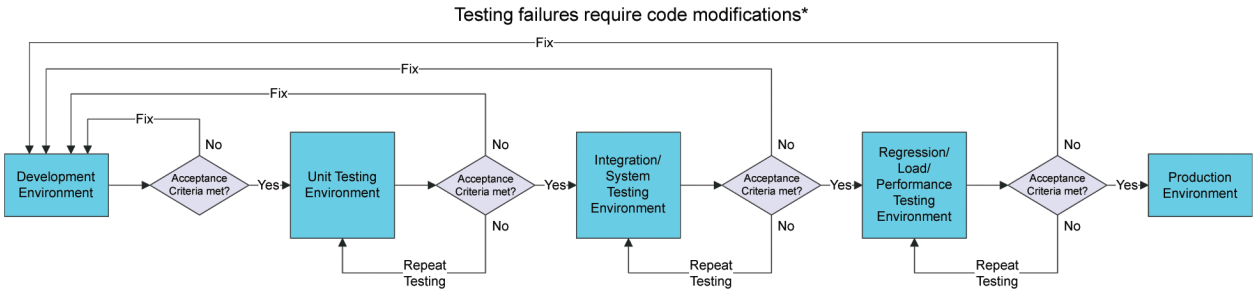
Testing failures may require retesting or the module to be modified. In some systems (e.g., enterprise resource planning systems), it may not be possible to physically remove a defective module from the environment where it has failed and move it back to the development environment. There should be a controlled process for promoting modules from one test environment to another and for ensuring that changes made in a later environment are returned to any previous environment that is still in use for subsequent development or verification activities. Controls should be established to prevent the release of the modules under development into the operational environment before completion of testing.

Controls should be designed, appropriately verified, and coordinated to:

- Avoid conflicts arising from different testers attempting to use the same data sets or accessing functionality at the same time. (The gains in productivity from parallel working can be lost because of such conflicts causing test failures and subsequent retesting.)
- Avoid conflicts between operational changes and new developments
- Ensure that the refreshing of test environments is optimized
- Preserve data protection in the test and operational environments
- Assist scheduling and efficient resource management
- Prevent inappropriate access to test environments
- Prevent the accidental release of informal developments, e.g., prototyping, to the operational environment
- Ensure the proper promotion of successful developments and demotion of failed ones

Testing to verify additions or modifications to an operational system should be performed as part of a release management process, which may co-coordinate multiple changes and associated testing activities.

Figure 7.2: Examples of Promoting and Demoting Developments through the Test Landscape



*In some systems, such as SAP, it is not possible to physically remove a defective development from the environment where it has failed and move it back to the Development environment. In these situations, the fixed development and the defective development have to be promoted in sequence through that the defective development is overwritten in each environment.

7.8 Automated Testing

Management of a test environment should be considered when implementing automated testing and test management tools. The anticipated efficiency improvements and levels of control from automation may not be achieved without an integrated approach to managing test environments. It is recommended that the requirements for managing test environments described in this Appendix are included when selecting, implementing and operating test automation and test management tools.

7.9 Non-Linear Software Development Methods

The adoption of non-linear software development approaches is likely to require dynamic usage of test environments. Instead of testing being performed at a specific period during a single testing phase of the life cycle, testing will be performed during each iteration of development. Data sets also will need be added or modified for each iteration.

Except for small scale developments, tools should be used to manage test environments in order to establish the necessary control and efficiency.

8 Appendix T5 – Testing Execution

8.1 Introduction

This Appendix defines good test execution practices for tests that are executed using:

- Hard copy test cases
- Using computerized test management tools
- Automated test execution tools

Additional specific guidance on the assessment, set-up, and use of computerized/automated test tools is provided in Appendix T11.

A test protocol, test specification/case, or test script is considered ready for execution after it has been approved.

8.2 Test Pre-Requisites

Pre-requisites for the test phase should be verified and recorded before starting testing using individual test cases or test scripts. For example:

- Test environments should be established, controlled, and documented. This may include hardware (e.g., serial numbers and calibration certificates), software (e.g., configuration settings and any custom code), data sets, and user accounts (see Appendix T4).
- Relevant personnel should be trained in the use of the software being tested and in the applicable test execution procedures. Details of training should be recorded (e.g., documentation of names, positions, sample signatures and initials, and training record or résumé (*curriculum vitae*) and references).
- Completion of any other test phases listed as pre-requisites
- Software to be tested should be available and under change control.
- All required documentation (including approved test documentation and approved requirements/design documentation being tested against) should be available.
- Critical instrument inputs should be calibrated, where applicable.

For non-linear software development methods, tests used for formal testing of a test case should be formally approved, subject to on-going change control/configuration management and relevant pre-requisites should have been verified and recorded.

8.3 Test Execution

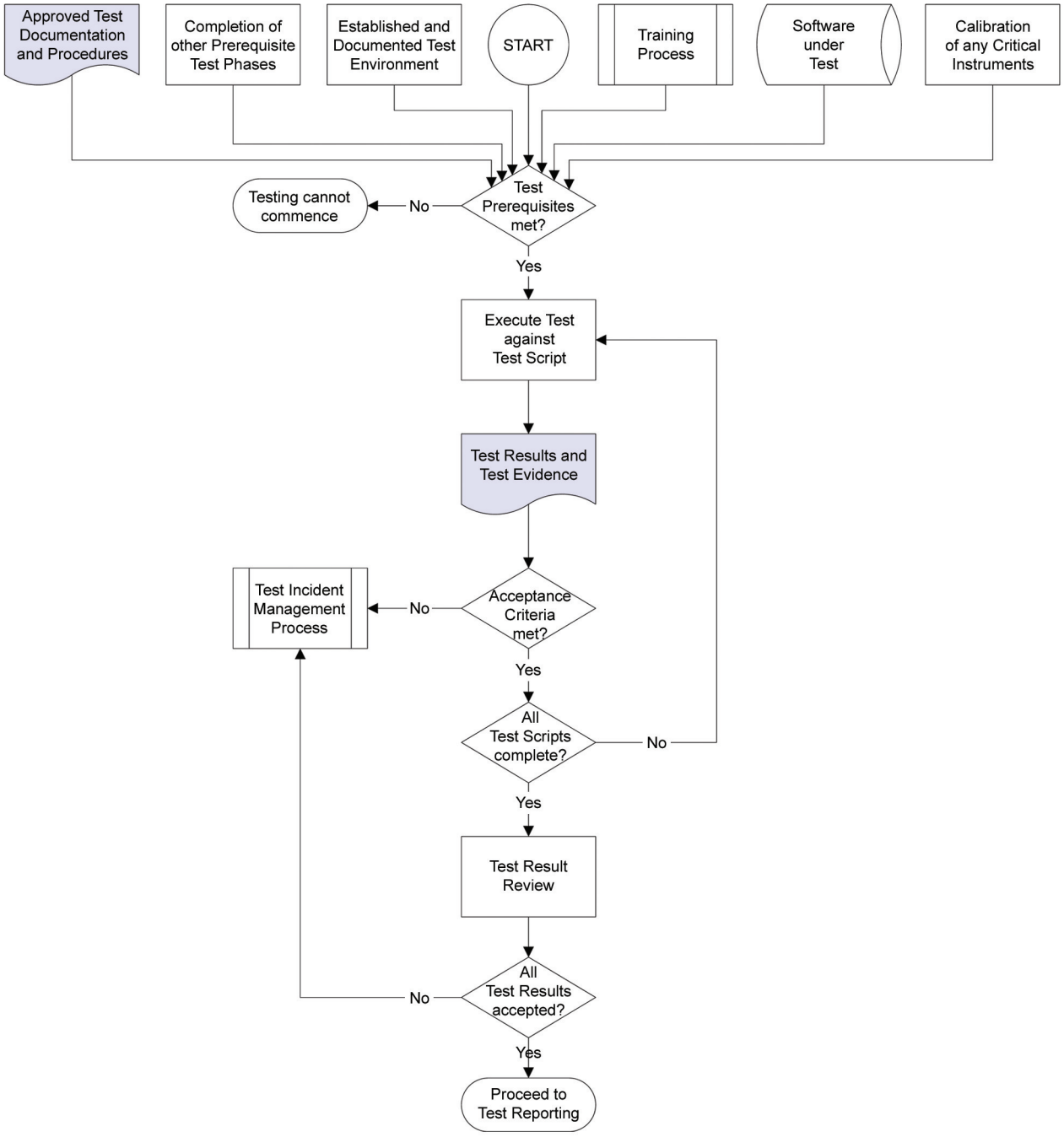
Tests may be executed manually or may be executed automatically using an appropriately assessed automated test tool.

Where testing is stopped, the reason for the stop should be documented. The documentation should be managed in accordance with defined test procedures.

If the stop is because of time constraints (e.g., a test is stopped overnight), it should be recorded in the test script, i.e., the time/date of the stop and the reason for stopping and the time/date of restarting. Such notations will usually be initialed by the tester. Where possible, tests should be started at such a time as to allow the test to be completed in a realistic and representative time scale.

If the test is stopped for a significant time due to a test deviation (e.g., where time is taken to determine whether the test can be continued or must be abandoned), it should be recorded in the test script as well as a reference to the test deviation record.

Figure 8.1: Overview of Test Execution



8.3.1 Manual Execution

Tests may be executed manually, using paper based test documentation or a computerized test management system, and will be based on inputs made manually by the tester.

A computerized test management tool does not necessarily interface directly with the system or software under test (i.e., the computerized test management tool is used only to manage the development and execution of the test case or test script). Inputs to the system or software under test are made manually (or using a separate tool). The computerized test management tool can be used to facilitate the testing of the software or system.

For further information on the assessment of computerized test management tools (and the difference between such tools and automated test execution tools), see Appendix T11.

For paper based tests, test results should be recorded on a controlled copy of the original test script or a specific test record sheet. Test evidence should be attached to the original test script or a specific test record sheet.

For manually executed tests, include:

- Pre-requisites for the test should be confirmed
- The test should be executed following the test instructions given within the test script
- Each test should be run, results recorded, and any test evidence attached. For further information on test results recording of the original test script or a specific test record sheet, see Appendix T6.
- The tester should decide whether the acceptance criteria for the test have been met and should record whether the test has passed or failed. The tester should sign and date the test results. Where the tester feels that the test has passed, but a test incident requires an independent review or second opinion, a separate category can be helpful, e.g.:
  - Refer for review
  - Conditional pass
  - Pass with observation
- Supporting documentary evidence required by the test case or test script should be collated. For further information, see Appendix T6.
- Test incidents should be recorded on the test script (if possible), or on a test incident sheet, or within the test incident system. A record of the test incidents should be retained as a permanent part of the test record. For further information on incident logging and management, see Appendix T2.
- A test progress summary should be used to record overall test results and number of test runs. Depending on an organization's test policy, the summary may be regarded as:
  - A status and scheduling tool
  - Part of the post-execution review and be included in or referenced from the test summary report or validation report as a formal document

For a computerized test management tool, test results should be entered directly into the system with screen shots or other records being attached electronically to the test record, see Appendix T6.

Computerized test management tools may provide statistics, reports, or dashboards automatically to allow the progress of the test cycle to be monitored.

8.3.2 Automated Execution

Where test execution is automated, the automated test tool usually interacts directly with the system under test via simulation of user input via a user interface.

The principles which apply to manual execution of tests also apply to tests executed using an automated test execution tool, but with additional considerations, e.g.:

- Test execution and the comparison of actual and expected results is performed automatically by the test tool.
- In the case of a test failure, careful review of the failed test may be required in order to determine which test step has failed and the reasons why.
- Post execution review of the test results, test evidence, and the overall test result can be performed only at the end of the test execution. This may require specialist knowledge of the automated tool, to be able to access and review the data captured during the execution.

For further information on the use of computerized test management and automated test execution tools, see Appendix T11.

8.3.3 Test Progress

After completion of all tests or a group of tests (e.g., at the end of the day), there should be a review of progress. A review group should assess all tests and incidents.

Possible actions for failed tests and incidents include:

- Repeat the test (e.g., in the case of tester error or where test prerequisites were not met)
- Apply a change via change control and if necessary repeat the test(s), including any tests as determined by assessment of the test incident and/or regression analysis (e.g., revise the test script, test data, software being tested)
- Abandon one, several, or all tests. This may limit the scope of requirements successfully tested in the release and this may require a workaround to be put in place (see Appendix T7).
- Review result and upgrade to a “pass” status (with a record of the rationale for the change in status)

The review group should decide which course of action to take and re-testing required. Justification for the action(s) taken should be documented. For further information on potential problems encountered during testing and incident management, see Appendix T2.

9 Appendix T6 – Test Results Recording and Reviewing

9.1 Test Evidence

A secure process for the creation, identification, handling, and storage of test evidence should be defined. The aim is to document (or otherwise record) test evidence with sufficient detail to allow the test to be independently reviewed and subsequently repeated if necessary. How test evidence is dealt with should be defined in one of:

- General test strategy or test plan
- Phase specific test specification or test protocol
- Test specific test case or test script (or a separate procedure)

Test evidence can be manually or electronically recorded, and can include:

- Quantitative data
- Observations
- Printouts (reports)
- Screen shots
- Automatically captured data (e.g., in Portable Document Format (PDF) format from automated test tools)

Requirements for additional test evidence in the form of printouts, screenshots, etc., should be clearly defined in the test method and focus on:

- Test steps which produce complex results, which may be difficult or time consuming to record manually
- When it is faster than manual recording of sufficient evidence to allow independent review
- When the result is something essentially visual and easier to review from a screenshot
- When there is a need for “before and after” comparisons

The test plan or test strategy should document how a test which provides only an observation should be dealt with before testing commences. A tester may record their observations which should be reviewed.

In limited circumstances, a test manager may deem it necessary to have the test witnessed (e.g., to meet contractual requirements for acceptance testing). If a witness philosophy is used, the selection of the witness should be based on similar training and experience criteria applied to the tester. The use of witnesses during testing may involve a significant overhead and is not typically required.

Test execution and test results recording may need one or more testers and the person executing testing may be different from the person recording results. The person recording the results should observe the test execution and the outcome of testing as it occurs. The strategy should be clearly defined in the approved test plan.

A combination of paper and electronic evidence may be appropriate (i.e., an electronic screen shot and a hard copy report) for some test cases.

Test evidence may be retained electronically if adequate security and retention mechanisms are established. Regulated organizations should justify and document the decision on how much hardcopy test evidence to retain, based on impact, novelty, and complexity; there can be a significant associated cost.

9.1.1 Paper Evidence

Suitable methods of managing and controlling paper evidence should be established, e.g.:

- In the case of printouts or reports, the date, test reference, test step, and the test run should be recorded on each page. Alternatively, where each page contains continuous page numbering (i.e., page "n" of "m"), a single reference and date for the whole document may be sufficient if the context is clearly defined (i.e., that it applies to all pages of the document).

- In the case of manually recorded data, this may be written directly onto a copy of the approved test script or onto a separate test result sheets. The test run number of the test should be clearly recorded.

For an example test result sheet, see Appendix T3.

9.1.2 Electronic Evidence

Where test evidence is retained electronically, the test evidence should be clearly linked to the test run. For example, this could be in the form of a CD-ROM appended to a paper test script, in which case the linking will be by hand annotation. Alternatively, the linking of test evidence to a test run could be within a computerized test management tool or an electronic document management system. For an electronic document management system, consideration should be given to:

- Security and integrity, including the audit trails

- Results recording with time/date stamp

- Verification of process for archival and retrieval of electronic evidence

- Approvals and signatures

- Availability for inspection

9.2 Test Record Integrity

Test records (test plans or strategies, test cases and scripts, test results, test evidence, and test reports) provide essential test evidence to demonstrate that system requirements have been met and that the system or application is fit for its intended use.

The management of test records should ensure their integrity. The principles of good documentation practices include:

- Handwritten entries should be made in clear, legible, indelible way.

- Records should be made or completed at the time each action is taken and in such a way that all significant activities are traceable.

- Any alteration made to the entry on a document should be signed and dated. The alteration should permit the reading of the original information. Where appropriate, the reason for the alteration should be recorded.

Guidelines should be established for the management of test records and the use of annotations on test records. For further information, see Appendix M9 of GAMP® 5 [1].

9.2.1 Paper Records

Where testing is based upon paper records, established GxP good documentation practice should be followed.

9.2.2 Electronic Records

Similar record integrity principles also apply to the use of test records in electronic format, e.g., records generated or maintained by computerized test management or automated test tools.

Tools and test environments used to maintain electronic test evidence should have documented assessments.

The type and level of assessment should be commensurate with potential risk. Testing tools and test environments do not typically need to be validated.

Where testing tools are used to maintain electronic records, specific controls and verification may be appropriate, based on risk.

9.3 Post Execution Review

The purpose of the post execution review is to assure that:

- Test procedures have been adhered to correctly
- Adequate test evidence was collected and recorded to approve the overall test result

The outcome of a post execution review is usually a completed test progress sheet and/or a test review report approved by the reviewer. Depending upon an organization's policies, individual test cases or test scripts also may be signed by the reviewer.

Post execution review activities should be conducted using the basic quality assurance principle of independence of review. Typically executed tests are subjected to peer review.

Executed test cases are typically summarized, documented, and reviewed at a point specified in the test plan or test strategy, i.e., at the end of each cycle of testing (as defined in the test specification or protocol for the phase), for each unit being tested, or at the end of the test phase.

Test reviewers may identify situations that should have been or need to be classified as a test incident. For example, manually altered test scripts, inadequate or insufficient evidence having been collected by the tester or automated test tools, actual results deviating from expected results. An incident report should be raised to control any necessary changes and/or manage any necessary additional testing, including appending additional test evidence with linkage to the original test evidence.

It may be helpful to create a general review checklist template to assist in reviewing test results. Example items for a review checklist include:

- For the results of individual tests:
  - Test result sheets (or test case or test script) and captured test evidence should be marked with the unique test reference and the run number

- Testers should be identified by appropriate means

- Appended evidence showing captured results should have been signed and dated by the tester (see Sections 9.1.1 and 9.1.2 of this appendix)

- That test results should be recorded in such a way that an independent reviewer can compare the documented acceptance criteria against the (written or captured) test evidence and determine whether the test met the relevant criteria. Typically, this requires the tester to record an actual result, not just “as per expected result” or similar, e.g., if the acceptance criterion is a range of values, the tester should record the range of values observed.

- Test failures should have been logged and raised as test incidents

For an automated test, linking of the test results to the correct test step should be confirmed. The testing data should be associated with the correct version of the data file.

- For the test phase:

  - Test results and test evidence are complete and in a format which meets the data integrity requirements set out in the overall test plan or test strategy

  - Test incidents should have been logged, classified, reviewed, and resolved (where required by the test plan or test strategy) before handover of the system

  - Software defect data that is collected and analyzed during a development life cycle should indicate the suitability of the software product for release for commercial distribution (where required by the test plan or test strategy)

  - Suitable regression analysis should have been performed. Agreed regression testing should have been completed following resolution of test incidents (where required by the test plan or test strategy)

  - Test completion criteria should have been achieved - in which case the test phase/cycle may be deemed complete

  - Test reports should comply with the requirements of the overall test plan or test strategy

For an automated test, tests in the test set should have been approved. Appropriate versions of the tests should be in the test set.

For further information on test phase reports, see Appendix T7.

9.4 Test Evidence for Packaged Systems

In a packaged system, the control system should be delivered with the process equipment it controls. Packaged systems can be positioned within a wide spectrum of software categories, including:

- Standard product software

- Custom written software

- Combination of standard product software and custom written software

For further information on testing considerations for packaged systems, see Appendix E7.

Test evidence can vary with the type of software delivered. For testing of a standard product (such as type testing or product release testing), test evidence should be recorded, reviewed, and retained by the supplier. Test evidence may be requested for inspection by a regulated organization, either as part of a supplier audit or project activity (e.g., design review). For testing of custom software or customized standard software, test evidence may be witnessed or reviewed by the regulated organization and supplied to them as part of the project documentation.

10 Appendix T7 – Test Reporting and System Handover

10.1 Test Reports

Typically, a test report is prepared at the end of each test phase, as defined in the test plan or test strategy.

For small or well understood systems, it may be practical to consolidate reporting of multiple test phases into a single, combined test summary report.

For large or complex systems or projects involving several test phases, each with a separate test report, it may be helpful to produce a final test summary report to present to auditors. The test summary report can provide a consolidated view of test progress, open issue resolution, and suitability for system release. Overall conclusions may be documented in a test summary report for large or complex systems or projects which have multiple test reports.

The level of detail in the test report should be scaled to suit the intended audience, taking into consideration that personnel outside of the original project team will not have the same familiarity with the system, and that those personnel may be required to present the report to inspectors at some time in the future.

Auditors may need to be given background information about a project, its status, and any issues encountered in order to be able to support the conclusions of the test report author.

Test reports should reference and track progress with the activities identified in the test plan or test strategy. Where a test report may be read without knowledge of the test plan or test strategy, the test report should include sufficient information to be understandable.

Test reports usually contain:

- Introduction
- Scope of testing
- Organization of testing:
    - Configuration management
    - Any changes to the roles and responsibilities compared to the test plan
- Details of any ad-hoc or non-routine testing performed as a result of incidents or observations
- Details of who performed and independently reviewed the testing
- Statement on whether requirements of the test plan or test strategy were met. Commentary to cover any deviations from the test plan, including justification that such deviations did not invalidate any testing should also be included.
- Summary of test results in tabular form
- Summary of test incidents including their remediation
- Reference to supplier or other test activities leveraged to support the conclusions (see Section 10.2 of this appendix)
- Conclusions (including known issues, outstanding incidents, and any restrictions on use)

- Approval of the test report and where applicable, authority to move to the next phase in the life cycle

- Where a system has been released to the next test phase with known open exceptions, the test report should provide any cross references that will be needed to allow the subsequent resolution of these exceptions to be traced.

Where summary information exists in a suitable format within the test results, it should be cross-referenced rather than repeated in the test report. For example, existing summary information may include:

- The documented test environment

- Test results tabulated on test progress sheets

- Indexes of test incidents

- Summaries of personnel involved with sample signatures

Test reports should be reviewed by appropriate functions as defined in the test plan or test strategy. Test reports should be approved, as a minimum, by the process and system owner(s) and circulated to the personnel responsible for any outstanding actions.

The approver(s) of a test report should not be the software developer, tester, or report author, see Appendix T2. Test reports may reference test results and evidence securely held in a supplier's or regulated organization's computerized test management tool. The assessment of a computerized test management tool (see Appendix T11.) should include an assessment of the security of executed test results or other attached evidence to ensure that test results and evidence cannot be edited or deleted once the executed tests have been approved.

The security of test results and test evidence may require additional configuration setting or customization in some computerized test management tools. The assessment of the security of the test results/evidence should consider the security of the underlying database in which the test data are usually held.

Where computerized test management tools are used to test embedded medical device software, the test results may be considered to be inspection records under 21 CFR Part 820 [12] and appropriate controls will be required to meet specific regulatory requirements.

Where it is intended to delete the test bench at the completion of testing (e.g., to free up capacity in the tool to support future test phases or other projects), the test results and evidence should be archived to a secure document repository in a controlled and verifiable manner. The assessment of the computerized test management tool should ensure that test results and evidence can be exported in a secure, non-editable format. The test report should reference the archive location.

10.2 Leveraging Supplier Testing

As a matter of good practice, suppliers are encouraged to make copies of test records available to regulated organizations. Regulated organizations should confirm access to supplier test records during assessment and selection of suppliers.

Supplier test records may be used in support of testing in accordance with the approved test plan or test strategy, provided that the following items are adequately addressed:

- Supplier capabilities and practices have been confirmed to be adequate, based on the supplier assessment process.

- The location of the referenced test records is stated and valid.
- The referenced records have sufficient detail to support testing conclusions.
- The referenced records have been completed to good documentation practices.
- The regulated organization should ensure that the test summary report demonstrates the linkage between user requirements and the specific supplier test evidence that demonstrates compliance.
- Access to test records is provided through the required retention period.
- The regulated organization may consider securing a certified copy of the test records or even written agreements covering access in order to ensure they are always available should an auditor wish to inspect them.

Where supplier testing is leveraged, the test report should confirm that the items listed have been demonstrated and that the requirements of the test plan or strategy have been fulfilled.

The outcome of the supplier assessment process may provide adequate assurance that software is “fit for intended purpose” without further testing. This outcome should be documented and may occur where:

- A supplier’s standard software modules are used without modification or customization.
- The supplier assessment process has demonstrated that the supplier has developed and uses robust software life cycle management processes.

Examples where this situation may occur could be mature software developed for a three-term controller or standard dispensing software algorithms developed by a weighing equipment supplier with a proven track record.

10.3 Determination of Residual Risk

On completion of the formal testing, the risk assessment should be revisited to:

- Assess the effectiveness of the testing in verifying risk mitigation measures
- Consider any previously unrecognized hazards highlighted during the testing process

This will lead to an assessment of the residual risk, i.e., the level of risk to patient safety, product quality, and data integrity still present in the system. A review should be performed to determine if the level of residual risk is acceptable.

Where residual risk is deemed unacceptable, possible courses of actions include:

- Identify new test cases and scenarios to further verify the controls
- Document any procedural workarounds required to provide additional mitigation
- Instigate change control to amend the system functionality.

10.4 Formal Handover and Release

For simple systems, handover processes may be defined in planning documents with handover/release handled as an integral part of the test summary report.

For complex systems, there may be several milestones during the testing process at which the system is formally handed over from one group to another, including:

- Handover from application development environment to test environment
- Handover from one test environment to another (e.g., factory acceptance to site acceptance)
- Release from a test environment to an operating environment

Handover from a supplier to the regulated organization usually coincides with one of these milestones and forms a special case because of the contractual implications of the handover (e.g., stage payments).

10.4.1 Handover Process

The handover process should be carefully controlled, documented, and communicated for each testing milestone.

The method for ensuring that the baseline recorded at the end of one test phase matches the baseline recorded at the start of the next phase should be defined, e.g.:

- To ensure that the software at the start of site testing is the same as that released at the end of factory testing
- Any changes made between test phases are traceable to agreed change requests or test incident reports

Where projects involve phased release of software on a system by system basis (e.g., in an S88 Batch Structure) and retain some element of common, baseline functionality for continued testing of other systems, then robust methodologies for tracking changes to either the baseline or released system software should be developed. These methodologies should ensure that changes to software in either environment are correctly assessed for impact on the other and that adequate regression testing should be performed to minimize risks.

A handover plan should be created that documents the tasks, responsibilities, timings, and acceptance criteria to establish when handover is complete. Depending on the criticality and complexity of a system, the handover plan may be a detailed document or a simple checklist.

The handover plan should be approved by the transferring and receiving parties and may be a separate document or incorporated within relevant project documentation.

Once the activities listed in the handover plan have been performed, a handover report should be created that provides a summary of completion of these activities and that acceptance criteria have been met.

10.4.2 Handover of System

The handover process should ensure that:

- Configuration items relating to software and hardware are transferred to the regulated organization and support organizations.
- All documentation is complete.
- Operational and support organizations are established and roles defined.
- System management, use, administration, and operational processes are established.
- Support services are defined.

- System users, administrators, and support organizations are trained.
- Data migration is complete.
- Security setup is complete.
- Residual risks and issues are transferred to user and support organizations.

10.4.3 Handover of Responsibilities

Transfer of responsibilities should be made alongside transfer of the system itself. Consideration should be given to:

- Responsibility for change control
- Responsibility for configuration management (and build management)
- Responsibility for documentation management
- Responsibility for system administration
- Acceptance of residual risks and responsibility for management of mitigation strategies or remedial actions

10.4.4 Conditional Release

The system should be accepted for use in the operating environment and released into that environment in accordance with a controlled and documented process. Acceptance and release of the system for use in GxP regulated activities should require the approval of the process owner, system owner, and Quality Unit representatives.

The handover process may need to consider circumstances in which a conditional release can be made. These circumstances may include:

- When test incidents are still open or changes are still pending
- With tests still to be completed because they are not possible outside of the operational environment (e.g., performance tests)
- When workarounds have been established

Conditional release should be treated as an acceptable exception to the standard testing procedure rather than adopted as common practice. It should include the use of a documented risk assessment and putting into effect any risk mitigation that is required, and may include review of the following:

- The number of open test incidents
- The number of tests still to be completed
- The maturity and effectiveness of the workarounds

Where a conditional release is agreed, the method and timescales for closing the outstanding actions should be agreed as part of the handover process.

For further information on the handover process, see Appendix O1 of GAMP® 5 [1] and Section 5 of the GAMP® Good Practice Guide on A Risk-Based Approach to Operation of GxP Computerized Systems [12].

11 Appendix T8 – Testing in the Operational Phase

11.1 Types of Change

This appendix considers systems that have previously been put into service (released for operational use), but which now require a software upgrade, patch, or change of use.

Testing of changes as defined in this section should be viewed as part of maintaining the compliant state of the system.

After the decision has been made to implement the change, but before implementation begins, change control and configuration management should be applied to the change. See Appendix O6 of GAMP® 5 [1] and the GAMP® Good Practice Guide on A Risk-Based Approach to Operation of GxP Computerized Systems [11].

Other support and maintenance processes may trigger testing, but are out of scope of this Guide.

11.1.1 Supplier Driven Change

Typically, a supplier driven change would be made as a result of the identification and correction of a defect in the supplier product or the introduction of new product features. The supplier should provide:

- Detailed documentation on the scope of the release
- Corrective actions taken
- Regression testing performed prior to release

Suppliers should determine the likely impact on the regulated organizations and may include within their standard testing of the new release regression testing of specific requirements, such as data integrity and critical functionality. Regulated organizations should document a risk-based decision about whether and how a change should be implemented.

Considerations affecting whether a change should be implemented include:

- Whether a defect has a negative effect on the regulated organization's business process (e.g., a potential impact on product quality or data integrity)
- Whether a new product feature provides functionality which is needed by for business process
- Whether new or modified product features are likely to have a detrimental effect on the business process
- Whether there are any risks associated with not implementing the change (e.g., on warranty or service level agreement conditions or on the ability to accept future upgrades)

Considerations affecting how the change is implemented and what level of verification is required after the change include:

- The extent to which supplier testing and documentation for the change can be leveraged
- The complexity and novelty of the change
- The number of other customers who have already implemented the change

- The impact on process controls and data collection during the change and any temporary measures that may need to be established
- The system documentation and operating procedures which need updating as a result of the change
- Whether any new features need configuring or disabling
- The appropriate level of regression testing, reflecting the identified risks

11.1.2 Regulated Organization Driven Change

Changes driven by the regulated organization are usually in response to business issues. The level and scope of testing should be based on risk to patient safety, product quality, data integrity, and business processes. Changes driven by the regulated organization may include:

- Change in requirements, e.g., changes in regulations, continuous improvement, change in business requirement, data changes
- Corrective actions arising from incident or problem resolution, performance issues
- Changes such as decommissioning an associated system
- Infrastructure changes and virtualization

Depending on the type of change, the change may be handled internally by the regulated organization, by a dedicated (internal or external) support group, or by supplier involvement.

The impact assessment for a regulated organization driven change should consider:

- Potential benefits (business advantage)
- Documentation and training burden
- Fit with organizational strategy
- How the change can be implemented (how and where it will be resourced)
- The technical and business impact of implementing or not implementing the change
- The scope of supplementary and/or regression testing required to release the system back into operational use
- If the change is for a software upgrade, but is not part of the supplier's standard release process, the regulated organization should consider the implications of this type of request and a decision should be based on a detailed risk assessment that should identify:
  - Does the supplier have a QMS process that covers this type of non-standard release?
  - Will the supplier still support this additional modification?
  - Will the requested modification be affected by the next standard version release?
  - If the supplier cannot or will not update the standard code to meet this single request, but will incorporate it at the next version release, is this acceptable?

- Are there internal processes that can be applied by the regulated organization until the next version is released with the requested modification, e.g., SOP driven controls or workarounds?

The regulated organization should ensure requirements are updated, risk assessments documented, correct code reviews performed, and appropriate testing is applied to this modification. Any supplier contracts/management plans should be appropriately updated to reflect the change.

11.2 Testing Planning and Test Management

Appendix T2 outlines test planning and test management processes conducted during the development and implementation of a new system or application.

During the operational phase of a system, system testing is usually the responsibility of the regulated organization (depending on the contractual support agreed between a regulated organization and a supplier). Where change is driven by the supplier of the software or system, a decision for reduced testing by the regulated organization may be supported if the regulated organization has an understanding of the:

- Scope of testing
- Rigor of testing
- Type of testing
- Quality practices of the supplier

The decision for reduced testing should be documented as part of a risk management process. The scope of testing should be determined by a justified and documented risk assessment, and consideration should be given to revalidating parts of a system potentially affected by the change. The following should be considered:

- A clear, concise description of the change should be documented in order to understand which other systems are impacted or have the potential to be impacted by this change.
- Reviews should involve SMEs, system owners, business owners and possibly end users (operators), project managers (if applicable – dependent on the scale of the change) and test managers. The impact of a change should be understood; its implementation and the testing/verification required to ensure the change meets its intended use should be agreed.
- Testing should still be conducted against documented requirements (updated if necessary).
- Updates to, or development of, a traceability matrix to aid the testing process should be considered.
- Testing should follow a pre-approved test plan or test strategy. This should take into account the time and resources required due to the complexity and scale of change(s) to the software.
- The nature of the software and hardware change(s) should be considered as part of the documented risk assessment (e.g., bug fix versus major upgrade). Changes in requirements may be implemented using mature software and may require less testing; new software required to address software problems may require more testing.
- There may be requirements to review any interfaces that could be affected by the change.
- The change may affect the risk of the system, i.e., if the system has gone from lower risk priority to medium or high risk priority, more testing may be required.

- Where possible, the regulated organization should seek to leverage supplier quality assurance processes and associated testing.

If a patch management process is being used, appropriate levels of testing should be applied to ensure that credibility of the patch is confirmed and no detrimental affects to the business process occur.

11.3 Testing Changes

11.3.1 Testing of the Changed Element

Testing should evaluate the implementation of the changed or new elements. This testing should be performed with the same rigor as the original testing, unless there has been a change in the associated risk priority. The execution and management of tests may differ from the original testing because:

- A separate QA/test environment may no longer be available. This environment may need to be recreated for major changes.

- It may be possible to take the operational use environment offline for testing during inactive periods (isolated in time). The system should be restored if the change is not successful.

- Where testing is conducted in an on-line operational use environment, additional consideration should be given to contingency planning and/or backing out from changes.

- Test data should be removed from, or isolated within, the operational use environment

- The project implementation team may no longer exist and the responsibility for testing should be shared between the process owner and the various IT maintenance and support groups. Appropriate handover of responsibilities should have been completed with the project team (see Appendix T7).

- The test environment may be in a state of change, see Appendix T4.

- When testing in a separate test environment, the test methods, process, and environment should be as close as possible (replication) to the operational phase to ensure no or minimum impact to the business occurs when the change is released for operational use.

11.3.2 Regression Testing

In ISO/IEC/IEEE 24765, regression testing is defined as:

1. “Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.

2. Testing required to determine that a change to a system component has not adversely affected functionality, reliability or performance and has not introduced additional defects.”

11.3.2.1 Supplier Driven Change

The regulated organization should review the documentation provided by the supplier and assess the sensitivity of the business area to which the change relates. From this, regression analysis should be used to determine if regression testing needs to be performed by the regulated organization.

Before releasing a new version, the supplier is responsible for proving that the change has been adequately tested. Regression analysis should be used to determine the scope and nature of regression testing, including any performance, stress, negative, or challenge testing.

11.3.2.2 Regulated Organization Driven Change

For a regulated organization driven change, the regulated organization should consider the potential of the change to de-stabilize the system. Focus will usually be on testing to ensure that the system meets the regulated organization's revised requirements. Regression analysis should be used to determine the scope and nature of regression testing, including any performance, stress, negative, or challenge testing.

The scope of testing for a regulated organization driven change should take into account:

- Business area (impact)
- Maturity of the area of the system which is affected
- Cost and time to effect the change
- Technical complexity
- Regulatory impact

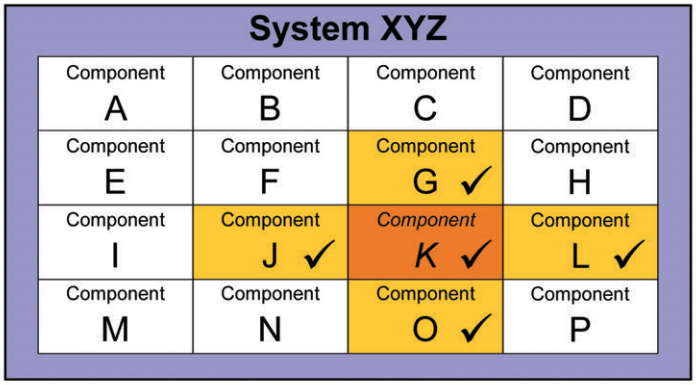
11.3.2.3 Example of Regression Analysis

Consider a component **K** of System **XYZ** that has been modified and tested as part of a change control process:

- Based on an impact analysis and risk assessment of the change, it was determined that there was some risk that the change to Component **K** also could impact Components **G, L, O,** and **J**.
- Components **G, J, L,** and **O** were re-tested based on risk (their associated test cases/scripts were re-run) to verify that changes to Component **K** didn't adversely impact them.
- Based on the risk it was also determined that the change was not likely to impact components A, B, C, D, E, F H, I, M, N, or P (see Figure 11.1).

System design documentation should support the decisions made by documenting the relationships between modules.

Figure 11.1: Regression Analysis



If component K is used in more than one system, interface agreements should be established between the different systems to make sure that the governance structures related to configuration management, incidents, and usage are in place.

11.3.2.4 Regression Testing Considerations

Aspects of regression testing which should be considered include:

- As part of the risk assessment for change control, a review should be completed to determine the level of regression testing required based on risk the change presents to patient safety, product quality, and data integrity (and business processes). Regression testing may be needed for major infrastructure changes (e.g., operating system upgrade)

- A traceability matrix may be useful as an input to determine the level of regression testing required, based on the impact the change may have on the existing requirements, design, and risk assessment documents. Focus should be given to ensuring the change has not affected the ability to review and retrieve data from systems that are required to have audit trail capability.

- The use of test automation tools can significantly extend the scope of the regression testing that can realistically be achieved in limited timescales and can provide a good return on investment where there are frequent changes or complex inter-dependencies within the software. For further information, see Appendix T11. Documentation of effective testing should be based on good test plans/test strategies and evidence of the completed tests being captured in good test cases/scripts. For further information, see Appendix T3.

- Appropriate SMEs and business quality units should be involved in approval of high impact changes.

- Reviews of associated test documentation, i.e., test cases/test scripts should be completed by an SME other than the tester to ensure adequate test coverage of the changed component and any associated components impacted by the change.

Table 11.1: Types of Change

Type of Change	Definition	Appropriate Verification to Ensure Continued Fitness for Purpose	Appropriate Management Method
Repair	HARDWARE: repair or like-for-like replacement STANDARD SOFTWARE: Unchanged or re-loaded APPLICATION: Unchanged or re-loaded DATA: Unchanged or re-loaded	• Confirm product code if like-for-like swap • Basic regression tests (correct power up, connection on all interfaces, live data, etc.) • Calibration if required • Record changed serial numbers • If standard software or application re-loaded, confirm version and set-up • If data re-loaded, confirm availability	Standard procedure which already takes into account the necessary verification elements
Patch	HARDWARE: Unchanged STANDARD SOFTWARE: Minor change APPLICATION: Unchanged DATA: Unchanged	• Record new standard software version and confirm set-up • Basic regression tests (correct power up, connection on all interfaces, availability of live data, etc.) • Risk-based regression tests based on: - Potential of changes to affect functions used within the system - Scope and rigor of testing already performed by the supplier as part of product release	Formal method for considering impact of changes Site change control for a one-off Standard procedure (possibly automated) if regular
Fine Tuning	HARDWARE: Unchanged STANDARD SOFTWARE: Unchanged APPLICATION: Adjustment to parameters within previously verified design space DATA: Unchanged	• Confirm adjustment has been correctly made • Capture final parameter values	Standard procedure which already takes into account the necessary verification elements
Major Operational Change	HARDWARE: Unchanged STANDARD SOFTWARE: Unchanged APPLICATION: Major change to required functionality, e.g., addition of new plant area DATA: Unchanged	• Record new application version(s) • Basic regression tests (correct power up, connection on all interfaces, availability of live data, etc.) • Risk-based regression tests based on potential of changes to affect other functions. • Tests to verify new or changed functionality	As a mini-project in its own right with planning, specification, implementation, verification and reporting phases

Table 11.1: Types of Change (continued)

Type of Change	Definition	Appropriate Verification to Ensure Continued Fitness for Purpose	Appropriate Management Method
Upgrade	HARDWARE: May be replaced by newer versions STANDARD SOFTWARE: Major change APPLICATION: May be changed to make use of new functionality DATA: Unchanged or possibly migrated	• Record new hardware product code(s) and serial number(s) • Record new standard software and application version(s) • Basic regression tests (correct power up, connection on all interfaces, availability of live data, etc.) • Risk-based regression tests based on: - Potential of changes to affect other functions - Potential for accidental alteration or deletion of existing data - Scope and rigor of product testing already carried out by the supplier as part of product release • Tests to verify new or changed functionality	As a mini-project in its own right with planning, specification, implementation, verification and reporting phases

11.4 Releasing a Change

Both the supplier and the regulated organization should have a release strategy for any change.

Key areas for consideration include:

- Where possible, batching of changes should be used to optimize efficiency in the testing process.
- A formal and documented mechanism for release is required and may include a release checklist as well as acceptance criteria and acceptance with known issues going forward. (**Note:** issues should not be major in nature.)
- Once a change is released into the operational phase, there should be an agreed time period for monitoring the effectiveness of the changes to ensure that:
  - The change meets its intended use
  - Any interfaces are successfully working
  - No detrimental impact to the business has occurred
- The level of documentation updates
- Education and training requirements
- Communication of change(s) – notification to management and end users

- Change control closure/approval of change to be placed into operational use (appropriate SMEs from the business areas)

For further information see GAMP® Good Practice Guide: A Risk-Based Approach to Operation of GxP Computerized Systems [11].

12 Appendix T9 – Testing as a Part of Data Management

12.1 Introduction

In several stages of the system life cycle, the integrity of GxP relevant data should be assured and suitable data verification activities should be planned and implemented, either as a standalone activity or aligned with functional testing.

This appendix discusses suitable approaches to data verification during a range of data management activities, during the project phase (usually data migration), the operational phase, and retirement phase of the system life cycle. Management of the data may be a:

- Key objective, e.g., clinical data management during a clinical trial
- Supporting process, e.g., data migration during the project phase or data restore during the operational phase

12.2 Data Migration

Data migration is the process by which data is transferred into a new system or environment and the data in the original environment is subsequently removed or deleted. Data migration involves:

- The extraction from the original system
- A potential need to transform the data into a format compatible with the new system.
- The loading of the data into the new system

A design (migration routing) may be required to map the data from the old system to the new.

Data migration activities should be considered during the development and testing of systems.

For further information on the broader aspects of data migration, see the GAMP® Good Practice Guide: A Risk-based Approach to Operation of GxP Computerized Systems [11].

12.2.1 Data Migration Planning

Data migration planning should include a documented assessment of risks associated with the migration process based on:

- The GxP impact of the data set involved
- The initial quality of the data sets involved
- The novelty and complexity of the migration process

Specific issues that may need to be considered by the risk assessment include:

- Appropriate issues of data segregation and confidentiality, dependent on the sensitivity of the data being managed
- Access to data by third parties, such as those responsible for migrating and converting data or administering data using processes such as backup and restore

- Any requirement for, or risk of, changes to legacy system data, especially master data, before final system cutover. For further information, see Section 12.5.1 of this appendix.

- The existence of complex data structures such as LIMS and PAT applications

- The existence of multiple data sets, multiple data sources, or large data volumes

- Whether any part of the data is currently stored as paper records, in spreadsheets, or in an application identified as decommissioned

Depending on the risk associated with the data migration, the data migration plan may be a separate document or may be entirely within another document such as a project validation plan. In either case, the plan should be a controlled document, as the data migration methodology is likely to be subjected to both internal and external audits.

Within the document set which cover the data migration, the following should be included:

- Documentation of the outcome of the risk assessment and any controls that were identified

- The specification of the data migration path including the processes used for:

  - Data collection (extraction)

  - Data cleansing (correction of erroneous data)

  - Data transformation (changing the format or context of data or metadata)

  - Data enrichment (adding new or missing data required by the new system)

  - Data loading (uploading or importing the data into the new system)

(**Note**: these may be complex technical processes, which are outside the scope of this Guide, but they should be formally defined and documented to retain an audit trail of any changes made to GxP relevant data or metadata).

- Definition of the data verification methods that will be used (see section 12.3.1 of this appendix).

- Details of any automated tools used in migration and /or verification and the assessment of these

Data migration processes may be conducted on an iterative basis during a project, e.g., to provide data for prototyping, integration testing, or user acceptance testing, as well as for final operational cutover. Typically, subsequent iterations will have higher data volume requirements and lower acceptable error rates; these should be defined in the data migration plan.

12.3 Data Verification

Where a data set and the data migration process are considered low risk, data verification may be included as part of the software verification activities.

Where a system is considered high risk or when data volumes are high, separate data verification steps should be considered as part of data migration activities.

Manual data verification may be more suitable for small data volumes and one-off data migrations.

Automated data verification tools may be more suitable for data migration processes, which deal with large data volumes or data migrations that are to be repeated many times. Automated data verification tools can take a significant time to assess, acquire, and set-up.

The balance between using an automated data verification tool and manual data verification should be based on risk. For high-risk priority data migrations, a combination of automated and manual data verification may be appropriate, e.g., some manual data verification may be required until the repeatability and reliability of the automated tool verification process is established.

12.3.1 Data Verification Planning

Data verification planning should define:

- Traceability to any controls identified as part of the risk assessment for the data migration process
- Methods used to verify the data migration, including:
  - Manual data verification
  - Automated data verification
- Acceptable quality limits at each stage in the migration process (i.e., what is the acceptable error rate?)
- Sampling plans (i.e., what percentage of data will need to be sampled to achieve the stated acceptable quality limits, the use of stratified data sampling)
- The need for generation of new data within the migrated system to verify the uniqueness of automatically generated values
- Any requirements for the assessment of automated tools (see Section 12.5.2 of this appendix)

12.3.2 Data Sampling

For some types of data, it has been standard practice to sample data for verification based on a minimum percentage. While this approach does not always have a sound statistical basis, this may be appropriate where the data volume is low and where a relatively high percentage (e.g., 5%, 10%, 20%) is required to ensure that a representative number of data values are sampled. In the case of very low data volumes, it may even be necessary to perform 100% verification.

For larger data volumes and/or higher risk priority datasets, it is more acceptable to sample data based on a sound statistical sampling plan (based on recognized sampling standards e.g., MIL STD 1916 [13], ISO 24153:2009 [14], ANSI/ASQ Z1.4 [15]), where the acceptable quality limits and data volume determine the number of data samples that should be verified. For large datasets, this can significantly reduce the number of data samples that need to be verified.

Acceptable quality limits should be risk-based, considering:

- The GxP impact of the data
- The data quality (based upon data profiling of data at source)
- The risk likelihood associated with the data migration process

An increased sample size effectively increases the risk detectability and thereby lowers risk priority.

Where a data migration process is used in a repeatable manner, e.g., in multiple roll outs of the same system, increasing or decreasing sample sizes for successive data migrations may be justifiable, based upon the results or previous data migrations and using reduced, normal, or tightened sample sizes.

For examples of reduced, normal, or tightened sample sizes, see the ANSI/ASQ Z1.4 standard [15] (or equivalent).

Data may need to be sampled from within defined data strata (stratified data sampling). This may be necessary where:

- The data set does not represent a single contiguous set of data (e.g., the data has been merged from different source environments or database tables).

- The data migration routines process different records in a different manner, depending on specific key values (e.g., patient records with a “male” gender are migrated using different migration routines to patient records with a “female” gender).

- Specific records have a higher impact than others, based upon the data values contained in the record (e.g., it is more useful to verify batch records for batches still on sale rather than those retained for other reasons).

Where each strata is migrated using a separate set of migration routines, each strata should be treated as a separate dataset and sampled appropriately.

Where the data strata within a data set are migrated using the same data migration routines, the overall sample size can be based upon the overall data population. Sampling should ensure that values are selected from within each stratum to ensure that the migration routines work correctly for all strata.

Within a data set or data strata, samples are usually selected at random or with a fixed record interval. It also may be useful to verify first and last records being migrated to ensure that there have been no errors introduced as a result of programmatic issues in looping data migration routines.

12.3.3 Master Data Verification

Master data may be migrated prior to the final operational cutover. Master data usually has a higher risk, e.g.:

- Product master data – recipes, bills of material, product routings, product test specifications

- Customer master data – contact details, ship to addresses used to facilitate product recall

Master data verification activities should be executed in a formal manner. Data migration specifications and test plans used for the verification activities should be approved and maintained in a controlled manner. Verification activities should follow good documentation practices to ensure that evidence is clear and concise.

Where the volume of master data is relatively low, a high sample rate may be required to assure acceptable quality limits and it may be feasible to conduct 100% verification.

Errors encountered during master data verification activities should be documented and given to the data migration team for investigation and resolution. Once a resolution is identified and implemented re-verification should be performed to ensure that the data has been migrated correctly and is meeting its intended functional requirements (usually confirmed during functional testing).

12.3.4 Dynamic (Transactional) Data Verification

Dynamic data also may be referred to as transactional data. Dynamic data is usually migrated just prior to a new system being released for operational use. These data are usually changing in an operational setting until the moment of final data migration, e.g., production orders, quality records, batch records, and customer orders. The volume of data and the short timescales usually available at this stage of a project mean that it is normally not feasible to verify all of the data.

Dynamic data verification activities are similar to master data verification activities and should be conducted in the same controlled manner. These activities may be performed by end-users or implementation team members who are familiar with the data.

Duplication of automatically generated transactional data, e.g., batch numbers or purchase order numbers, between source and target environments should be avoided.

Dynamic data sets can range from having no GxP significance (but nevertheless be business critical) to having high GxP significance and this should be reflected in the data verification approach (verification methods used, acceptable quality limits and sample plans). It may be beneficial to use appropriately assessed automated data migration and verification tools in order to minimize the extent of manual data verification activities.

12.3.5 Data Migration to the Operational Environment

Once verification processes are considered accurate, reliable, and repeatable, the master and dynamic data can be promoted to the operational environment. Typically, the master data will be promoted to the operational environment prior to the dynamic data.

Once all data is promoted to the operational environment, final checks should be performed to verify that the data has been promoted successfully (e.g., review of data migration logs, error messages, and database status). Where the data migration process is considered to be accurate, reliable, and repeatable (i.e., low risk likelihood of there being a data migration error, based upon previous data migration verification activities conducted in earlier phases) this final verification may be limited to a confirmation that the final data migration activities were executed in accordance with the defined process, using appropriately assessed data migration/verification tools.

Depending on risk (e.g., where data has a high impact or where the data migration process is innovative), this final data verification may include sampling of data in the operational system.

12.3.6 Data Migration and Verification Reporting

Once all data migration activities have concluded, a data migration report should be developed to summarize the data migration process and the final outcome. Process deviations and data discrepancies should be reported. The data migration report should include any changes made to data in the final operational system. For further information on reporting and system release considerations, see Appendix T7.

For small or simpler data migration activities, the data migration reporting may be included in other reports, i.e., test summary reports or the validation report.

12.4 Manual Data Migration and Data Verification

Manual data migration is a repetitive activity prone to human error. It requires a high level of data verification to mitigate the risks involved in the manual mitigation. For high-risk priority data or large volumes of data, consideration should be given to automating the migration process.

Manual verification consists of comparing data from the source environment with data in the target environment (which may be a development, test, or operational environment). Comparisons of data should allow for any expected changes in format, value, or record count.

Migrated data may be manually verified by appropriate personnel.

Where it is difficult to compare directly data from two systems, manual verification may be assisted by the use of a data sheet containing a copy of the data extracted from the source environment. Data sheets should be controlled to ensure that the contents are accurate and up-to-date. Expected values may be included in a data sheet and should be in the same format as the expected values in the target environment.

Where the data migration process transforms or enriches data, calculations may be used to derive the expected data values from the source data values. Calculations (e.g., in spreadsheets) used to derive expected data values should use different tools and developers (i.e., verified independent methods) to those used in the actual data migration process.

Where manual data verification relies on data sheets, test scripts may be developed with instructions on how the data should be verified. For further information on the use of spreadsheets, see GAMP® 5 [1].

The scale of data verification documentation, review, and approvals should be commensurate with the risk associated with the data migration process.

It may be beneficial to have a second person to assist with manually verifying large data volumes, both to help maintain focus and to review verification activities.

12.5 Automated Data Migration and Data Verification

Where data migration activities are automated, it may be appropriate to:

- Document an assessment of the automated data migration/data verification tool for fitness for purpose
- Manage and control the data migration tool, e.g., configuration management, test data integrity, and version control
- Manually verify the successful migration of the data

While data migration programs and routines should be inherently reproducible and while it should not be necessary to verify the output data migrated by appropriately assessed data migration tools, data migration is a process and may be prone to human error, e.g., the wrong input data set is selected, the wrong version of a migration routing may be used, or a data file moved to an incorrect directory prior to import to the target environment.

The balance between tool assessment and manual verification should be based on risk and for high risk priority data migrations a combination of the two may be appropriate, e.g., some manual data verification may be required until such a time as the repeatability and reliability of the automated tool based process is established.

12.5.1 Using Automated Tools for Data Migration and Data Verification

Automated data migration or verification tools should be introduced and used in accordance with good practices, including consideration of:

- Assurance that the tool satisfies organizational (or project) standards and policies
- Verification that the tool delivers the required functionality, does not inadvertently modify critical records, or affect system/application performance significantly

- Documentation of the assessment confirming its ability to deliver accurate and reliable results
- Logging of users, major changes, problems, and observations pertaining to the continuing correct use of the tool

Automated tools may be used at several points in the migration process including:

- Discovery, extraction, and profiling of source data
- Mapping and converting data (applying rules)
- Interim verification of data before and after using the transformation logic
- Final data verification

Typical data migration and verification activities facilitated by the use of automated tools include:

- Identification of the source data and data profiling by the development team
- Identification of the target data
- Identification of transaction identifiers
- Definition of mapping/transformation rules
- Definition of folder mapping (within the data migration tool)
- Content comparison
- Execution of the automated routines
- Inspection of logs

Discovery and profiling of source data is part of the planning and development process, but forms the foundation for the final verification activities. Source data could be taken from single or multiple systems with data that could be filtered based on time periods or the status of transactions.

The quality of the source data will form one input to the risk assessment of the data migration process and thus have an impact on the level of final data verification required.

Automated tools may be used to identify the data within the source environment, extract the data, and also to profile the data (e.g., identifying missing, corrupt, or inconsistent data).

Automated tools also may be used to cleanse (correct) data, enrich data (add missing or additional data values to a dataset), and convert/transform data. Automated tools usually involve the use of programmed migration routines developed in accordance with the data migration specifications.

Automated tools also can be used to perform interim or final data verification tasks based upon similar rules, i.e.:

- Initial verification involves a frozen copy of the converted data along with the record count on the source data. The automation tool may be used to verify that the data is correctly extracted from the source environment (usually based on a record count) and may check defined data values (i.e., that extracted data has been filtered correctly).
- Interim verification includes checking the cleansed or enriched data. The verification checks that there are no corrupted, illegal, or inconsistent data and that all mandatory data values are present.

- A final level of verification compares the original data with the converted data to ensure that the cleansing, enrichment, and migration process has not introduced any errors. Depending upon the capability of the automated tools this may be executed within the data migration environment (comparing extracted and converted data) or across the source and target environments.

Automated data verification can eliminate data sampling and can provide reporting and verification of all migration results. Automated tools may automatically generate a report on the results of the migration activity. Data elements extracted and loaded are usually verified, which includes:

- Verification of all data migrated
- A report on the total record count
- Confirmation that expected transformation rules were met
- Generation of a log (audit trail) for all data

For high GxP impact data or a high risk migration process, a subsequent manual verification in the target environment may be performed to confirm results. This should exclude human error and ensure that the automated data migration process was correctly executed, e.g., that the correct routines were used in the right order (an incorrectly selected migration routine may not generate any errors, but may still incorrectly migrate operational data).

Where automated tools are used both to convert data and to verify the migrated data, different programmed routines should be used, following defined procedures or instructions. This should prevent the potential that the same error is introduced into both the migration and verification process. This can be achieved by using different tools, programming techniques, or developers.

Critical factors for the successful use of automated data migration and verification tools include:

- Domain knowledge of the source data
- Domain knowledge of the target data architecture
- Configuration of the migration and verification tools
- Availability of key data stakeholders
- Adaptability (functional) testing by examining the behavior of the new system with the migrated data
- User acceptance testing following migration

12.5.2 Assessment of Data Migration and Verification Tools

Automated tools used for data migration or data verification should be assessed before use. This assessment should provide assurance that the tools can perform migration or verification activities in a reliable, robust, and repeatable manner.

An appropriate, documented assessment should be performed for automated tools that are used as part of the data migration or data verification process.

Depending upon the impact of the data being migrated and the complexity and novelty of the automated tools being used, the assessment may include:

- An initial market assessment of available tools that may be suitable

- An assessment that the tools can meet the general technical data migration/verification requirements (including the ability to:

- Handle expected data input formats and types,

- Process expected data volumes, execute expected data migrations,

- Provide expected data output in the necessary formats and to automatically verify data across the expected source and target environments)

- An assessment that the tools meet the project/system and data specific migration/verification requirements. This may be achieved by manually verifying data converted/verified during the early phases of a project, in order to build confidence in the accurate, reliable, and repeatable operation of the software developed for the specific project. It also may include verification of the migration process using a test dataset containing records specifically selected to challenge aspects of the migration and/or transformation process.

The extent to which the assessment is documented will depend upon the risks associated with the use of the tool, and the criticality of the data involved. Where data migration /verification tools are used across more than one system or project the assessment and documentation should be applicable to the most critical data expected to be migrated.

12.6 Data Management in the Operational Phase

12.6.1 Data Change Control

Manual changes to master or dynamic data that are made during the operational life of a system are usually verified at the point at which the change is made. Where changes to data are made en-masse, manual or automated data verification methods, similar to those described for data migration, may be used.

When a change to GxP critical data deviates from the described validated usage of a system (e.g., data corrected by a user or database administrator), the change should be pre-approved by the data owner, and the implementation of the change should be approved by the data owner's Quality Unit.

In the operational phase, manual intervention to GxP data as described, should be made in a controlled manner.

12.6.2 Managing Master Data

Approved change control procedures or an approved operational SOP should be followed when implementing changes to master data.

The degree of verification required is dependent upon the impact of the data and the method of change, e.g., standard updates to a bill of material would be verified at the point of entry, in accordance with an operational SOP, but changes to data in the database executed by a database administrator (e.g., following data corruption) would require a change control and additional verification by the data owner.

Verification should ensure that:

- The required data changes have been correctly identified

- The specified data changes have been correctly applied

- For high risk data, only the specified changes have been made (through a scope of regression test that is appropriate to the GxP risk)

12.6.3 Managing Dynamic Data

Dynamic data is usually created or updated as part of on-going operations, in accordance with operational SOPs. Changes to dynamic data should be managed with verified tools to ensure that the relational integrity of the database is maintained. Change control should be used for changes to data that are outside the scope of operational SOPs, in order to maintain the integrity of the data audit trail. Where standard tools cannot be used, a higher level of verification should be applied for each change.

The degree of verification required is dependent upon the impact of the data and the method of change.

12.6.4 Data Backup and Recovery

Depending upon the system, data may be backed up using processes which backs up:

- Data and associated metadata together
- Data and associated metadata separately
- A complete system (which includes a full image of the system including the data, metadata, and configuration of the system)

Individual data backups may be automatically verified using the verification features of a standard backup solution, but this does not test the ability to recover the data.

Data backup and recovery processes should be appropriately tested before releasing a system for operational use. Depending upon the architecture of the system this may be part of a wider system image backup and recovery process or a system specific data backup and recovery process.

The overall data backup and recovery process may be verified by comparing data from an image of the original (pre-backup) system with the recovered data, using similar techniques to those described for data migration. See Tables 12.1 and 12.2 which describe risk scenarios and typical test types and phasing for data management.

For further information on backup and recovery, see GAMP® GPG: A Risk-based Approach to Operation of GxP Computerized Systems [11].

12.6.5 Data Archiving and Retrieval

It may be possible to archive and retrieve data, depending upon the architecture of a system. It may not be possible to merge retrieved data with operational data in a live system. Archived data may be retrieved only to an alternative image of the operational system, e.g., an image created in a test environment specifically for this purpose. Individual data archive operations may be automatically verified using the verification features of a standard archive system, but this does not test the ability to retrieve the data at a later date. Data archiving and retrieval processes should be appropriately tested prior to releasing a system for operational use and the nature of the data archive and retrieval process should be considered.

The overall data archiving and retrieval process may be verified by comparing data from an image of the original (pre-archive) system with retrieved data, using similar techniques to those described for data migration. See Tables 12.1 and 12.2 which describe risk scenarios and typical test types and phasing for data management.

For further information on data archiving, see GAMP® Good Practice Guide on Electronic Data Archiving [16].

Table 12.1: What is special about Data Management? – Risk Scenarios

Difference from "Generic" Functional Testing	Risk Scenario	Impact on Data Verification Strategy
Data verification relies on manual or automated comparison of two data sets rather than functional script based testing (e.g., data from source and target environment, data from pre-data archive and post data retrieval). Test scripts will be simplified with greater emphasis on test case definition and associated data sets.	Missing or corrupt data	Data sets should be selected to detect any missing or corrupt data. This should include appropriate data sampling and the inclusion of any specific data or records that may be of particular risk likelihood, e.g., first and last records, records with missing data, records of known poor data quality. Data sampling should be risk and statistically based, to ensure that sufficient data is sampled to identify (and correct) any missing or corrupt data. If corrupt or missing data exceeds the acceptable quality limit, the source of the error should be identified and the data management process (migration, backup, archiving) should be re-executed and re-verified. An alternative approach is to iteratively correct the missing or corrupt data and re-sample and verify until the acceptable quality limit is achieved.
	Falsified data	Based on a risk assessment, data integrity and system security should be tested to ensure that data cannot be falsified by normal means. During normal operations data should only be created, updated, or deleted as defined in operational SOPs or under appropriate change control.
	Quality of Source Data (data migration)	Source data should be profiled (sampled and analyzed to detect missing, corrupt, or inconsistent data) to identify data quality issues, which should be considered as part of the data migration planning.
	Process failure (data migration)	Complete failure of the data migration process can be identified in the project phase. Complete failure of the data migration process may be more difficult to detect where a previous migration process fails and leaves old data in place. The source and time stamp of data sets should be verified before use. The partial failure of a data management process (e.g., the wrong program is executed or an incorrect dataset is selected) should be detected by sample based data verification.

Table 12.1: What is special about Data Management? – Risk Scenarios (continued)

Difference from "Generic" Functional Testing	Risk Scenario	Impact on Data Verification Strategy
Data verification relies on manual or automated comparison of two data sets rather than functional script based testing (e.g., data from source and target environment, data from pre-data archive and post data retrieval). Test scripts will be simplified with greater emphasis on test case definition and associated data sets. (continued)	Technical failure of the data management process (migration, backup, archiving, etc.) including unavailability of source or target environments or media.	Complete failure of the data migration routines are easy to identify in the project phase, i.e., the migration routine or program does not run. This may be more difficult to detect where a previous migration process fails and leaves old data in place. The source and time stamp of data sets should be verified before use. Complete failure of data backup/restore or data archiving/retrieval should be captured by alarms or alerts, which should be tested as part of the process verification. The partial failure of a data management routine (i.e., the routine executes, but corrupts or loses data) should be detected by sample based data verification.
	Media failure (data backup/ restore or data archiving/ retrieval)	The use of supplier standard data verification routines should be used based upon risk. Media manufacturers may not recommend the periodic testing of data restore/retrieval processes as this reduces the expected life of the media (all data should be copied to new media and verified within the expected life of the original media). Data restore/retrieval processes should initially be tested prior to operational use of the system and successive backups/archives should be retained to reduce data loss in the event of a media failure (e.g., the full system backup for the previous two months, weekly data backup, and daily incremental backups for the previous week). Where incremental backups are used a full system restore should be tested using the latest full backup and all necessary incremental backups. Where partial data archives are used it may be useful to test a full data retrieval to verify that different data sets can be integrated into the target retrieval system and the data used for operational purposes.

Table 12.2: Typical Test Types and Phasing for Data Management

Test Phase	Typical Coverage	Opportunities for Risk-Based Leveraging or Efficiencies
Project – Design Verification	Data migrated to support initial project design activities (e.g., prototyping, design qualification) usually only require limited data volumes with representative data (mostly master data and limited transactional data). This helps to develop system requirements and design and the accuracy of the data migration is not usually essential at this stage.	Pre-existing or supplier standard data migration tools and routines should be used.
Project – Functional testing (Integration Testing, SAT or FAT, User Acceptance Testing)	Data migrated to support functional testing should have defined acceptable quality limits and the data migration should approach full data volumes in order the test the performance, reliability, and robustness of the data migration process.	Pre-existing or supplier standard data migration tools and routines should be used. Risk and statistically based data sampling may be used. Automated data verification tools may be used.
Project – Operational Cutover	Data migrated (or promoted) to the operational system should have defined acceptable quality limits and requires full data volumes.	Pre-existing or supplier standard data migration tools and routines should be used. Risk and statistically based data sampling may be used. Automated data verification tools may be used.
Operational – Master or Dynamic Data Management	Manual verification of changes to data in accordance with a defined operational SOP (or change control).	Verification process should be appropriate to the risk associated with the data. Software based input checks should be used to limit data management risks at the point of data edit/entry and these checks should be validated. Automated data verification tools and/or statistically based data sampling may be used for mass updates to data.
Operational – Data Backup and Recovery	May be part of full system backup or a separate data backup with defined data sets.	Supplier standard backup and recovery solutions should be used wherever possible. Dependent upon the technology platform, existing data backup and recovery solutions/technology may be applied to new systems.

Table 12.2: Typical Test Types and Phasing for Data Management (continued)

Test Phase	Typical Coverage	Opportunities for Risk-Based Leveraging or Efficiencies
Operational – Data Archival and Retrieval	May be an archive of all data sets and records or just selected data sets and/or selected records.	Supplier standard archive and retrieval solutions should be used where possible. Dependent upon the technology platform, existing archive and retrieval solutions/ technology may be applied to new systems (e.g., at the database level).
Retirement	Migration of data to a new environment to ensured continued availability after a system has been retired.	Pre-existing or supplier standard data migration tools and routines should be used.

13 Appendix T10 – Testing in Non-Linear Software Development Methods

13.1 Introduction

One of the drivers for GAMP® 5 [1] was to:

"Acknowledge that traditional linear or waterfall development models are not the most appropriate in all cases."

Waterfall and V diagram based approaches to software development imply linear-sequential life cycles, where each life cycle phase is completed before the next one can be started. Linear-sequential approaches may have disadvantages, including:

- Projects do not always follow the sequential flow demonstrated by these models and as a result change is not easily accommodated.
- Users in a regulated organization may not be able to state all of their requirements explicitly at the beginning of a project.
- Users in a regulated organization usually do not see a working version of the application until relatively late in the project timescale. Major omissions may not be identified; therefore, until it is costly and time consuming to correct them.

These disadvantages have led to the development of a number of non-linear software development methods to address these issues. These life cycles are used both by software suppliers and some regulated organizations developing their own applications.

13.2 Non-Linear Software Development Life Cycles

Non-linear software development life cycles range from variants of Rapid Application Development (RAD) through to Agile Methods, which include Extreme Programming (XP) and Scrum:

- Dynamic Systems Development Method (DSDM) provides detailed descriptions of phases and team roles with supporting documentation.
- Rational Unified Process (RUP) provides tools to support life cycle activities.
- Scrum describes a process, but details such as testing and coding are left to the project team to decide.

Prototyping is a phase within a non-linear software development method. For further information on prototyping, see Section 3.3 of this appendix.

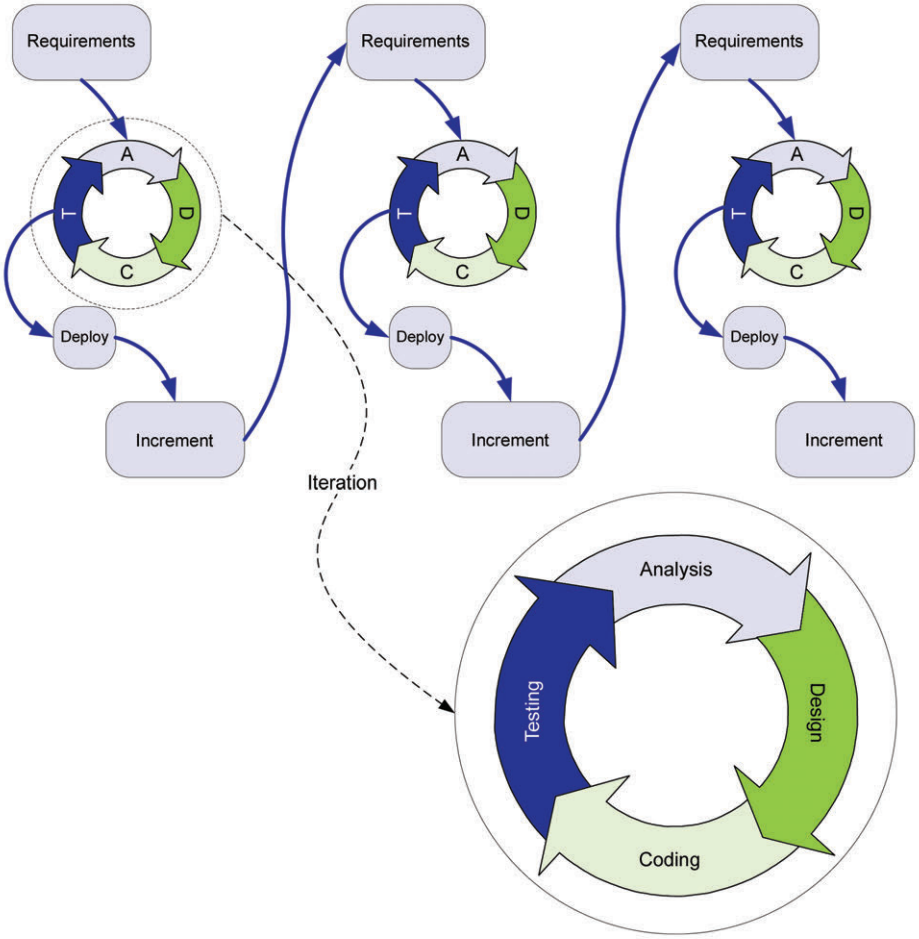
Non-linear software development methods can be iterative and incremental:

Iterative Development – each increment is developed by a repeated, cyclic process of analysis, design, coding, testing, and review which produces a working version of the application that delivers a subset of the user requirements.

Incremental Development – an application is delivered in a series of planned releases or increments, each delivering functionality that builds on previous releases. Incremental development normally results in more frequent delivery of functionality which provides a faster return on investment for a system. The method of delivery of each individual increment depends on the approach used and may be an iterative process.

A generalized form of an iterative and incremental process showing three increments is shown in Figure 13.1.

Figure 13.1: Iterative and Incremental Software Development



Features of non-linear methods may include:

- Prioritization of user requirements using, e.g., the Must, Should, Could, Won't (MoSCoW) method.
- Time-boxing – each increment is delivered within a fixed time period. This may result in the re-assessment of the user requirements to be delivered in that increment during the iterative process. Prioritized requirements help to decide which can be left for a future increment, as required.
- User participation – the level of involvement of the user representative is much higher than in traditional linear-sequential approaches.
- Co-located teams – during each iteration, where practicable, the user representative, software developers, and testers are located together geographically.
- Team autonomy – because the requirements to be implemented and the resulting application design can change during an iteration, the project team should include representatives who are empowered to make these decisions
- Documentation – high level requirements for each increment can be defined prior to the iterative cycle design and test documentation can be drafted during the iterative development of the application, but not finalized and approved until the cycle is complete.

13.3 Prototyping

Prototyping can be incorporated into an existing life cycle. Prototyping addresses the issues with linear-sequential approaches that users often cannot state all of their requirements explicitly at the beginning of a project. User interaction with a prototype of the application can allow:

- Requirements that would otherwise have been missed to be identified
- Users to see a working version of the application earlier in the project timescale

Depending on the type of prototyping, the output of a prototyping phase can include user requirements specifications and an initial version of the application. Two types of prototyping are:

1. Throwaway (or rapid or mock-up or wireframe)
2. Evolutionary

Throwaway (or Rapid or Mock-up or Wireframe) Prototyping

This is a prototype built to discover user requirements. Once the requirements have been confirmed, the prototype is discarded and the application is built from scratch following a linear-sequential or non-linear software development method.

There are no testing implications because the throwaway prototype is discarded. The use of this prototype can result in improved URSs.

Evolutionary Prototyping

In evolutionary prototyping, a prototype is developed which, after a series of iterations and based on user feedback, becomes the final system. Evolutionary prototyping is considered equivalent to iterative development.

Guidance provided on testing within iterative development also applies to evolutionary prototypes.

13.4 Non-Linear Software Development Risk Scenarios

Table 13.1 shows additional risk scenarios introduced by the use of non-linear approaches and the impact these have on testing.

Table 13.1: Risk Scenarios Introduced by the Use of Non-Linear Approaches

Difference from Linear Development Life Cycle	Risk Scenario	Impact on Test Strategy
User requirements may be completed/ delivered in more than one increment.	Requirements necessary for a particular increment are not included.	The test strategy should aim to ensure that all requirements delivered in each increment have been adequately tested. The user representative should confirm that the requirements for each increment will deliver a system with sufficient functionality.
	Requirements in later increments cannot be implemented due to the absence of functionality on which they are built.	Prioritization and sequencing of requirements should be planned before selection of requirements to be delivered in the initial increment. The test strategy should ensure that this can be verified.
	Functionality delivered in earlier iterations may change with later iterations.	Test cases from earlier iterations should be reviewed (and updated if necessary) with each iteration. Requirements should be reviewed to ensure that new requirements are correct (and that existing requirements remain correct), complete, consistent, unambiguous, compliant, measurable, and testable.
The number of requirements to be delivered in a specific increment may change due to time limitations.	Requirements necessary for a specific increment are not included.	The test strategy should aim to ensure that all requirements delivered in each increment have been adequately tested. The user representative should confirm that the requirements for each increment will deliver a system with sufficient functionality.
	The scope of testing may be incorrect if the functional scope of the increment changes relatively late in the iterative phase.	The scope of the test plan should be carefully reviewed before formal testing commences; to ensure that the iterative phase tests included all delivered requirements.

Table 13.1: Risk Scenarios Introduced by the Use of Non-Linear Approaches (continued)

Difference from Linear Development Life Cycle	Risk Scenario	Impact on Test Strategy
The design during the iterative phase of an increment is not fixed.	Regular updates to application software may result in: • Inadvertent changes to functions that do not require updating. • Poor quality code, e.g., presence of dead code, inconsistencies in structure and annotation.	Regular regression testing involving the re-testing of some or all of the application functionality should be performed. This testing should cover functionality delivered in previous increments, if applicable. The use of automated test tools is recommended because of the high level of repetition during multiple iterations. Regular, formal source code reviews should be performed to verify that the code is maintained to acceptable standards.
	Testing is performed at the end of an iteration instead of as a parallel activity, i.e., a linear approach is taken. This can result in rushed and, therefore, insufficient testing at the end of an iteration.	The development team should be competently trained in the chosen non-linear approach.
	There is a mix of linear and non-linear development within the same project, e.g., an interface with another system may be developed separately. This could result in difficulties in planning and collaboration between the two teams and potential errors in design or coding between the linear and non-linear development flows.	This should be recognized and incorporated into the test strategy. Scheduling should ensure availability of both applications for testing. Periodic re-testing should be required as the non-linear software component evolves.
Design and test documentation is subject to change during the iterative phase and cannot be formally approved until this phase is complete.	Coverage and depth of testing is insufficient to demonstrate that the application meets user requirements and design specifications.	Regular review of test scripts and results during development. Coverage and depth of testing based on a documented risk assessment. Maintenance of traceability of requirements, through design specifications to test scripts. User requirements, design specifications, and test protocols are reviewed and approved prior to formal testing in the deployment phase and afterwards are subject to change control.
	Test scripts evolve to ensure that the software passes the test, not that the software meets the requirements.	Test scripts should be regularly reviewed to ensure that they challenge the requirements and specifications sufficiently throughout the iterative phase.

Table 13.1: Risk Scenarios Introduced by the Use of Non-Linear Approaches (continued)

Difference from Linear Development Life Cycle	Risk Scenario	Impact on Test Strategy
Lack of formal testing during iterative development.	Test coverage and documentation is insufficient to demonstrate regulatory compliance.	The set of test scripts evolves with each iteration. Test scripts and results should be reviewed with each iteration to ensure that they challenge the requirements and specifications sufficiently throughout the iterative phase. The test environment should be set-up and managed during the iterative phase to ensure that it is as representative as possible of the final operational environment. Formal testing of the software or system should be against approved test scripts prior to deployment.
	The test environments are not representative of the operational environment, resulting in errors being discovered following deployment.	Deployment methods between the environments should be efficient, and installation testing may need to be performed in a different manner compared to traditional installation testing for linear developments. Configuration management should be employed and use of a configuration management tool can be beneficial.

13.5 Testing by Phase

The section describes how testing is performed within an increment. A risk-based approach, as described in this Guide, is relevant in non-linear methods.

13.5.1 Iterative Development Phase

Testing usually occurs more frequently and earlier in the development life cycle in non-linear methods compared with linear-sequential life cycles.

In some non-linear methods, e.g., (XP and developments of XP, such as test driven development), test cases are written prior to coding.

The development and execution of test scripts should occur early in an iterative phase. Test protocols and test scripts can be updated as a design evolves and should be sufficiently robust to be approved and used for acceptance testing by the end of an iterative phase (see Section 13.5.2). The Quality Unit should provide early and regular review to ensure the adequacy of test scripts and test results.

Testing should occur repeatedly during the iterative phase. Testing of individual modules should be refined and re-executed during development. Integration tests may be developed to test two or more modules and a complete suite of tests may be executed, e.g., after each build of an application.

Where the overhead in designing and maintaining automated tests can be justified, the use of automated test tools in the iterative phase is recommended. This allows tests to be easily repeated. In iterative development, regression testing is used to verify that design changes in successive iterations have not introduced bugs in other parts of a system, so automated tools can be beneficial. Automated tests also can be beneficial in a time-boxed iterative phase. For further information on the use of automated test tools, see Appendix T11.

It may be impractical to repeat all tests in regression testing using manual testing. The choice of tests should be based on a documented assessment of risk of the functions being developed adversely affecting existing functionality.

The use of an error- or bug-tracking tool is recommended; to aid testing in non-linear methods. Each error found in an application, particularly in the early stages of development, should be tracked to ensure that it has been resolved. Minor errors may be left unresolved until a later increment and a tracking tool can allow these to be identified and incorporated into the release notes for that increment.

Other forms of verification, such as code inspections and reviews, should be used to maintain the quality of code throughout the development process. Successive increments should be regularly reviewed for adherence to coding guidelines and presence of dead-code to prevent poor quality software. For further information on performing and documenting source code reviews, see Appendix D4 of GAMP® 5 [1].

13.5.2 Deployment Phase

Once the application development has been completed, formal acceptance testing should be performed against approved user requirements and design specifications prior to release. Acceptance test scripts should have been developed during the iterative phase and formally approved prior to execution.

Testing throughout the iterative phase should result in only a few test failures.

A formal release management process should be followed. This should document which functionality is being released and should identify any unresolved issues.

Once deployed a formal change management system should be applied to the further development of the application in subsequent iterations with the change impact analysis feeding into the test strategy for the next increment.

13.6 Team Roles

Non-linear methods usually have a higher level of involvement by user representatives. This can help to ensure that the evolving design during the iterative phase will deliver a system that meets the user requirements to be delivered in that increment. User representatives may be full time members of the iterative development team.

Non-linear methods may not have defined roles associated with testing. Depending on the size of the development team a test manager or coordinator may need to be appointed to ensure that testing is correctly planned and executed.

Draft test documentation and results should be reviewed during the iterative phase to help to ensure that a GxP compliant system is delivered.

13.7 Test Documentation

The principles of test planning, management, execution, and reporting as described in this Guide are relevant in non-linear development methods. Testing during the iterative phase can be less formal due to the evolving system design. Test cases and/or scripts usually evolve alongside the design and can be formally approved once the iterative phase is complete. Some or all of the scripts can be executed to demonstrate that an application can be released for use. The selection of which test scripts to execute should be based on a documented assessment of risk and challenge all of the critical functions being delivered in the increment.

A test report (see Appendix T7) should be prepared to document testing performed during the iterative and deployment phases. The test report should identify any unresolved errors and justify why they have not been corrected.

14 Appendix T11 – Automated Test Execution and Computerized Test Management Tools

14.1 Introduction

Automated test execution tools can be used to improve test execution efficiency and effectiveness.
The use of automated test execution tools should be defined in the test strategy. Tools should be used in accordance with defined instructions and manuals as appropriate, and the tool should be held under configuration management.

Like other software, test automation software may be standard, configurable, or custom, and should be specified and verified appropriately based on an assessment of risk. Commercial or established tools are considered to be GAMP® Software Category 1 and usually have a low impact on product quality or patient safety; therefore, they are considered to have a low risk priority and do not typically need to be validated. Appropriate assessment of the tool should be carried out before use and specific controls and verification may be appropriate, based on risk and the need to ensure the security, integrity, and reliability of GxP test records.

This appendix describes the objectives and benefits of test automation and computerized test tools. It also defines criteria for successful computerized testing, provides guidance on how to select and assess an appropriate tool, and describes some frequent pitfalls and misconceptions.

There are several types of test automation software and a large number of software testing tools available. The functionality of these tools can be extensive and covers the entire software development method including:

- Business process modeling
- Requirements development
- Configuration management
- Defect tracking
- Syntax checking
- Test coverage analysis
- Unit and functional testing

The main purpose of test automation should be to develop automated support for testing activities.

Test design should be separated from test implementation. This should allow test designers to concentrate on test planning, developing test requirements, and test case design, while test implementers build and execute test scripts.

Three types of computerized test tools are:

- **Test Management Tools** – facilitate the task of test case and test script authoring, requirements management (i.e., mapping of requirements to tests), review and approval (pre- and post- execution), test execution, and test deviation management. Tests are executed manually and usually a dashboard captures the test metrics.
- **Test Automation Tools** – assist in authoring or recording test scripts, automatically executing test scripts, and recording test evidence. (This appendix focuses on test automation tools for test execution.)

- **Performance Test Tools** – specific type of automated test tool, allowing defined and realistic system loads to be generated automatically, providing realistic performance benchmarks.

The investment in computerized test tools can be significant and should be regarded as a software development activity. Computerized test tools usually have their own assessment and implementation life cycle. This will place additional requirements on the test team; particularly in terms of the skills required by test script developers.

Test automation tools can be used effectively only if supported by adequate processes and if testers have the necessary skills to use the tools successfully. Testers should receive appropriate training in the operation of automated test processes. Where the wrong test cases are selected for automated testing, this may take longer than manual test execution. A proof of concept should be performed on the test automation system.

Users of computerized test tools should be trained in the use of the tools and in good testing practices. Testers should understand the requirements being tested, to help ensure that the test script adequately tests a requirement and the results are appropriately captured.

When using computerized test tools, user roles and responsibilities should be defined with respect to each test process. Appropriate technical or procedural access controls should be used to ensure that only authorized users are allowed to fulfill a defined role and that the independence of the test process can be verified or enforced.

Computerized test tools should provide the same level of test data integrity as an equivalent paper based process.

Features which should be considered include:

- Role based authorities and user restrictions
- The use of workflow to enforce test processes
- The use of biometric, electronic, or digital signatures
- Secure computer generated audit trails

Not every aspect of a computerized system can be tested using automation. A key factor in the successful use of test automation tools is the early identification of the types of tests that should be automated and the type of tests that should be executed manually. For example, when testing requires the use of hardware (e.g., barcode scanning), manual testing will probably be preferred over automated testing.

In general, the use of computerized test tools should be conducted in accordance with a test plan or test strategy that requires all tools used to test GxP systems being subjected to assessment before use. This assessment is to establish a high degree of assurance that the tool functionality:

- Is capable of meeting the needs of the users (commercial computerized test tools may be highly configurable and some can be customized)
- Will perform as intended
- Is capable of returning accurate and secure results

This appendix provides guidance on how to assess computerized test tools.

14.1.1 Features and Functions of Computerized Test Tools

The architecture or main programming language of an application under test may determine which tool is suitable. Tools can be:

- Purchased as Commercial Off-the-Shelf (COTS)
- Downloaded as Open Source Software (OSS) packages
- Developed in-house as custom software with the test team then configuring the package locally to meet the particular use required

Less complex test automation tools, such as test harnesses (a framework or library with common functions to support automated testing in combination with a test execution engine) can be developed within a project team.

Organizations should define their requirements before acquiring computerized test tools.

14.1.1.1 Test Management Tools

Test management tools facilitate:

- The task of test case and test script authoring
- Requirements management (i.e., mapping of requirements to tests)
- Review and approval (pre- and post- execution)
- Control of test execution and storage of test results
- Test deviation management

This is accomplished through the use of:

- Workflow enabled processes
- Electronic test artifact management (i.e., test cases, test scripts, requirements coverage analysis, defect tracking reports, test deviation reports)
- Electronic signatures (possibly)

Where a test management tool is used, a tester performs the actual execution of the test, even when this task is supported by the presentation of online test scripts and electronic capture of test evidence. Testers manually follow the test steps, make inputs to the system under test, and record the test evidence.

General test management tools may:

- Support/manage testing across a project
- Help in mapping the requirements to test cases
- Facilitate the authoring, review, and approval of test cases and test scripts, often using workflow and electronic signatures
- Allow developers, testers, and project managers to track tests that are being run on various applications by a management overview screen (e.g., cockpit, dashboard, or report)
- Help trace tests back to their original requirements and assure complete test coverage
- Provide summaries of defects found during the testing process

- Support following up on defect handling

Test management tools may not provide the facility to automate test execution.

14.1.1.2 Test Automation Tools

Automated testing can be particularly important where non-linear, iterative, or incremental methods are used, as test automation makes the necessary testing and regression testing of various development iterations practical and cost effective (see Appendix T10).

In general, test automation includes the automation of software and hardware testing activities, including:

- Test data and test script generation
- Test execution
- Test result analysis
- Test documentation
- Test administration

Test automation tools can range from broadly applicable to very specialized tools, e.g., for a specific operating system, programming language, or an application server. In general, these tools support test execution activities in various operational areas like infrastructure, functional testing, PLCs, interfaces, performance testing, or monitoring.

Automated test execution is a main area of interest for most organizations. Test automation tools also can address automation of test data and test script generation, as well as configuration management. In addition, test automation tools may interact directly with other test management tools through the use of a test tool suite involving, e.g., the automatic:

- Starting of an automated test script
- Recording of test results
- Generation of defects in a defect management tool in case an automated test script fails

Typically, test automation tools that support functional testing include one or more of the following features:

- Testing that the application code being developed is acting as expected
- Capture and replay utilities, which run sample tests against working versions of a program, capturing the activity it generates. During the replay phase, developers can see whether or not they are generating the expected results.
- The use of keyword or table driven execution engines, which allow the development and execution of repeatable tests using parameters from a table, file, or spreadsheet.
- Scripting capabilities, including parameterization; allowing testers to modify what they have captured to test additional items. This is usually used to record a manual test script and script an automated test script that can run the same script several times using different input data (from a table or file).
- Integration with other packages providing requirements capture and tracking capabilities; therefore facilitating comprehensive requirements traceability

- The ability to simulate user input and capture the results, including the ability to interface to specific common COTS applications (e.g., popular ERP systems) and technologies (e.g., web browser controls using ActiveX, Java™, or Adobe® Flash®)

- The ability to test large applications where different developers are working on different parts of the application

- The ability to integrate with software configuration management and deployment tools

14.1.1.3 Performance Test Tools

Performance test tools can help to validate the interaction between the software application under test and its supporting infrastructure with respect to response times, load, and scalability. Different infrastructures offer different underlying resources and services, such as Central Processing Unit (CPU) or memory (e.g., random access memory (RAM), hard disk).

Performance tests check whether the infrastructure and the software application meet the regulated organization's performance requirements. A test environment should be prepared which is as representative as possible of that which will be used during operation (see Appendix T4). If another setup is used, the boundary conditions should be documented. It is unlikely that the results of a small-sized test scenario can be extrapolated to predict an application's behavior in a real-life scenario, as scalability, break points, and race conditions are typically not computable.

Typically, performance test tools:

- Are particularly useful to test what happens when several users (sometimes thousands, depending on the application) access the application simultaneously or transactions are required to be submitted or responses achieved in tight deadlines that manual testing could not reproduce accurately or consistently.

- Are particularly useful for critical web-based applications because it can often be difficult to predict the volatility of load change. Tools developed for this type of work may drill down to lower levels of the code to find out where bottlenecks exist and what is causing delays.

- Are useful in terms of process based acceptance testing because performance test tools enable a repeatable test execution of performance tests which is almost impossible by manual test execution (most performance test tools can incorporate variable "think time" to simulate real users).

- Can be valuable when testing the scalability of the application

- Capture the capacity of a system for a known configuration

- Baseline the response time from remote locations as part of latency testing

14.2 General Benefits of Test Automation Tools

Automated testing should help to improve quality and test scenarios and should be easy to run, write, and maintain with an emphasis on minimal maintenance. Testing objectives should include implementation of a strategy that will allow tests to be developed and executed both manually (initial test cycle) and via an automation framework (regression test cycles) or using a modular testing approach. In addition, a good test design can be developed and reviewed individually before deciding whether to automate.

Using automated testing of software can be especially beneficial when the software is custom developed, requires multiple versions (typical of non-linear methods), or third party software that gets regularly updated. The advantages of automated testing can include:

- **Increased consistency and quality:** computerized testing and automation can provide increased consistency in an organization's deployed systems through consistent application testing. There is a faster execution with more defects found earlier; therefore, improving quality. In addition, the actual test execution data (e.g., timestamps, tester, and results) is always precise, as it is electronically recorded and not subject to human failure.

- **Time and cost reduction:** automation technology can mean faster test execution that saves both time and money while allowing the involvement of non-technical personnel in the validation process. Automation technology also can run a larger number of tests in more situations (i.e., run a more complete suite of regression tests on patches or upgrades).

- **Repeatability and enhanced agility:** with automation technology, organizations can test more frequently (e.g., whenever changes occur); therefore, reducing risk, helping to maintain compliance, and supporting the overall verification process. The use of automated testing enables the feasible release of minor upgrades or installation of monthly operating system critical updates rather than saving up several months' worth of upgrades or patches before installation.

- **Broader scope and/or more rigorous testing:** with data-driven testing (i.e., using a broader set of test data in repetitive, automated testing), it becomes easier to execute multiple test iterations using different data sets, allowing coverage of more data permutations than would be possible using traditional manual methods. In addition, data from external sources can be incorporated into the testing process.

- **Stress testing:** the ability to test applications under load with multiple users is very difficult, ineffective, and costly (in many cases even impossible) if done manually. Automated performance testing tools allow stress testing of the application in a repeatable fashion against large numbers of virtual users.

- **Utilization optimization:** automated testing can run unattended and interpretation of the pass/fail result is simplified with evaluation being required only for exceptions.

- **Test coverage:** using integrated data tables to provide large volumes of test data (inputs and expected results) or using parameters to call separately maintained business rules (therefore exercising all normal, alternative, and exceptional use cases) can be used to effectively increase test coverage

- **Reduced paper work and improved efficiency:** electronic traceability of requirements to test cases, as well as the clear and concise reporting on testing activities can reduce paper work and improve efficiency.

Test automation can be used in combination with planned software changes, e.g., where an application is under development or is being further customized or extended. The results from automatically executed development tests may provide timely feedback and help detect unforeseen side effects. This can be especially true in combination with other automated life cycle activities, e.g., continuous integration and nightly build generation.

14.3 How to Use Test Automation Tools

A test lead or tester who has the skills to act as an administrator of the computerized test system should lead the test automation effort. The test lead should have knowledge of the tool and be able to:

- Assign work appropriately

- Recognize the need to support changing software development priorities

- Shift resources quickly

Requirements should have detailed workflows or use cases, complete and consistent requirements, and developer assistance for the system.

Test automation should be implemented as a full-time effort. Detailed guidelines for developing test scripts should be created to help to prevent potential issues with maintenance.

A work practice or standard operating procedures should be created to cover naming standards, which should apply to:

- Variables
- Environmental variables
- Functions
- Test data
- Test scripts
- Data sheets
- Script headers
- Comments
- Exception handling
- Reporting
- Coding
- Think times
- Folder structure
- Object identification
- Parameterization

General testing standards and steps to take after automation scripts are created should be addressed.

14.3.1 Common Pitfalls and Misconceptions

Pitfalls in test automation include:

- Object recognition (the ability to interact with software objects in the system under test, simulate user input, and recognize result output)
- Attempting to automate a legacy system or outdated versions of software where there are no standard interface mechanisms or common technologies
- Failing to manage unreasonable expectations about automated test tool capability - particularly the opinion that “automating the tester” will result in an ability to do everything from test planning to test execution without manual intervention

The Return On Investment (ROI) in testing automation is unlikely to be achieved within the first six months.

Automated tests lack variability, testing exactly the same scenario repeatedly, whereas manual testing usually includes some degree of variation; therefore, increasing the chance to disclose previously undetected anomalies. This may be partly overcome by introducing randomized, automatically generated test data, but it is unlikely to match an experienced human tester hunting for errors in an exploratory manner (unstructured usability testing).

Test automation should be able to run unattended, i.e., without human interaction, including the evaluation of test results. This can be achieved only where the testing objectives and acceptance criteria (expected qualitative or quantitative results) are defined and specified as part of the automated test itself (i.e., assertions). This can be challenging and should be considered when deciding which tests cases to automate.

14.4 Selection and Benchmarking of Computerized Test Tools

The following points should be considered when deciding whether to use computerized test tools (see Section 14.2 of this appendix):

- Will the tools be used on a single project or more broadly within the organization?
- How many test cycles are typically executed and how much regression testing is required?
- Will a dedicated test team be using the tool or will other project members use the test tool? This has an impact on on-going administration and training requirements.
- Which test strategy and which test types will be supported by the tool?
- What would be alternative solutions for test document management and versioning when not using the automated test tools?

Test processes that will need to be supported by computerized test tools should be identified. The test process implemented through the use of such test tools should follow testing good practices, along with the specific test policies and processes of the regulated organization.

The requirements for the use of the computerized test tool(s) should be derived from the regulated organizations' standard test policies and processes. The defined requirements are then used as a basis for generating the test tool procedures and work instructions. Overly complex test processes should be avoided. The configuration of the test tool(s) should not drive test processes, as this may lead to non-compliance with an organization's policies and procedures.

Note: ROI is likely to be easier to achieve if all automated testing follows the same process, rather than a project specific process.

The following tables provide a compilation of requirements for potential use when evaluating computerized test tools. These requirements cover several fundamental features and functions that computerized test tools should provide. These requirements may be used for benchmarking computerized test tools as part of an assessment process (see Tables 14.1, 14.2, and 14.3).

Table 14.1 lists the criteria that apply to test management, test automation, and performance test tools (test suites).

Table 14.1: Criteria and Basic Requirements for Computerized Test Tool Suites

Criteria/Requirement	Comments
The test tool should be assessed and established, and it should support validation activities.	Consider if a COTS test automation tool and/or open source software tools are appropriate and whether they can be assessed in a comparative manner.
The test tool should show the current status of script execution (metric or traffic-light system): How many scripts have been executed in total, how many test runs have been executed, how many have passed, and how many failed?	Cockpits, dashboards, or overview matrices provide easy to understand views of execution ratios, overall status, and remaining tasks, etc.
The tool should support secure test document management, e.g., test cases, test data, test results and evidence, including compliance for electronic records where applicable (see below).	Requires secure storage by test management tool or separate document management system. May include 21 CFR Part 11 [17] requirements (see Section 14.5 of this appendix).
The tool should allow electronic signing of test plan and report documents, including compliance for electronic signatures where applicable.	Requires documented review and approval of test cases, and test plan and report documents. May include 21 CFR Part 11 [17] requirements (see Section 14.5 of this appendix).
All test scripts should be handled as controlled records or documents. The adherence to defined and adequate good documentation practices should be ensured.	Test plans and reports should be subject to appropriate document management controls. Test cases may be managed as records or documents, according to regulated organization procedures.
The following activities are additional functional areas which should be considered for computerized test tools: • Requirements mapping • Test case creation (with test data and test script) • Test execution • Test evaluation (deviation management) • Test documentation management • Test administration	None

14.4.1 Test Management Tools

Table 14.2 defines some basic requirements and selection criteria for test management tools (computerized tools used to manage manually executed tests).

Table 14.2: Criteria and Basic Requirements for Test Management Tools

Criteria/Requirement	Comments
Test management tools should provide appropriate administration functionality for test plans, cases, scripts, and reports, and should provide an efficient access to all test documents.	Administration and management tools for all test documents should include version control, including appropriate review and approval functions (which may include workflow).
A standard interface to a common document management system should exist if no appropriate functionality for the administration of test documents is included as part of the test management tool.	Interface to Document Management System (DMS) also can be an organizational or manual interface, following a defined procedure.
Test management tools should support defect handling and resolution with reference to the original test result.	In cases where the tool does not provide an automated support, this may be incorporated into a training program.
Test management tools should manage the status of test cases (number of runs, open, passed, failed, number of defects, etc.)	Provides test managers with an up-to-date overview regarding the current test situation.
Test management tools should support the management of different concurrent projects and test cycles and different configurations of the same test specification.	Should allow the efficient re-use of standard default configurations.
Test management tools should capture the traceability of requirements to testing.	The requirements management module should interface to the test execution module within the test tool to provide information on requirements coverage (which requirements do not have associated test cases, which requirements have not been successfully tested).

14.4.2 Test Automation Tools

Table 14.3 defines some basic requirements and selection criteria for test automation tools.

Table 14.3: Criteria and Basic Requirements for Test Automation Tools

Criteria/Requirement	**Comments**
Automated testing tools and test environments should have documented assessments for their adequacy.	Tools should be assessed for their adequacy before usage and the assessment should be appropriately documented (see Section 14.5 of this appendix).
The test automation tool should be configurable regarding how to manage and execute test scripts.	May include the use of a script language and/or capture and replay tool. The test automation tool should be compatible with the test management tool, if used.
The test automation tool should have a user interface that allows testers to write/record test sequences with the following possibilities: • Analogous recording of mouse movement, clicks, and keystrokes and their replay • Context sensitive recording of graphical objects by their object "name" calling Both possibilities should be supported at the same time.	Both requirements do not need to be fulfilled within the same test automation tool.
Test scripts should be producible by personnel with general knowledge about testing and test tools (ease of learning of test script language, no sophisticated programming skills needed).	Minimal programming knowledge advisable for test script writing. The ease of use of scripting will determine the decision for capture and replay versus modularization/ parameterization of scripts.
With test automation, the test script is also the test case (contains the test data, expected results, test steps, etc.) Test scripts should be available in a format allowing easy review and approval by non-technical personnel.	It should become clear from reviewing a test script what the test steps are, which parts of the application are being tested, what the expected results are, and which test objectives are being fulfilled.
The test automation tool should enable command line scripting for input of parameters and their values or input by a scripting language for processing large file sequences.	Needed to support data driven testing. An alternative is reading test values by a test script.
Black box/white box testing should be supported together with access to the application source code.	Required for structural testing of custom software – may require the custom software to provide "hooks" or for the test automation software to be specific to the software development suite.
The test automation tool should support test script maintenance, e.g., if the application or the infrastructure changes and the test scripts need to be adapted accordingly.	In such cases, version control should be adopted to differentiate the test script maintenance activity.

Table 14.3: Criteria and Basic Requirements for Test Automation Tools (continued)

Criteria/Requirement	Comments
The test automation tool should have a test data generator to provide random test values (within defined boundaries).	Alternatively, test data can be generated by an external randomization tool and test data should be maintained using version control.
The test automation tool should automatically generate log files or other documented evidence of the executed scripts. The level of detail should be (at least partially) controlled by the test script itself.	Although global log level settings are common, it is beneficial to increase or reduce the level of detail that is being recorded per test (supports risk-based testing).
Log files must not be editable or changeable; they should be retained for future reviews.	Log files are part of the test documentation and should be appropriately secure.
Test scripts should be executable repeatedly for continuous test operations or for later re-testing (e.g., release upgrade or updates/patching) and should be usable for regression testing.	Useful for non-linear life cycles and continuous integration.
It should be possible to automatically start test scripts at any time and they should support a continuous test operation.	Useful for non-linear life cycles and continuous integration.
Test scripts should run without human interaction (unattended mode).	Useful for non-linear life cycles and continuous integration.
Interface testing should be possible. Further comprehensive system testing should be supported.	May require the test automation tool to be able to interact with all interfaced applications under test.
Keyword-driven testing should be supported.	None
Data-driven testing should be supported.	None

14.4.3 Performance Test Tools

Performance test tools should meet the requirements of automated test tools, except for the requirement for white box testing.

Table 14.4 defines some additional basic requirements and selection criteria for performance test tools.

Table 14.4: Criteria and Basic Requirements for Performance Test Tools

Criteria/Requirement	Comments
Performance test tools should have a very low impact on the infrastructure in order to provide accurate performance test results. During the test execution the use of services and resources by the performance test tool itself should be clearly shown.	None
Performance test tools should monitor the use of resources occupied by the tested software application.	Often achieved by the integration to commercially available infrastructure monitoring tools.
Performance test tools should support test execution on different infrastructures. This can include different versions of operating systems as well as different hardware platforms.	None

14.5 Assessment of Computerized Test Tools

Commercial or established tools are considered to be GAMP® Software Category 1 and usually have a low impact on product quality or patient safety; therefore, they are considered to have a low risk priority and do not typically need to be validated. Appropriate assessment of the tool should be carried out before use and specific controls and verification may be appropriate, based on risk and the need to ensure the security, integrity, and reliability of GxP test records.

The assessment and verification of a computerized test tool should focus on demonstrating that the tool is fit for purpose as far as critical requirements are concerned. This may include verification of the core product or of a specific configuration of the product, using techniques such as scripted demonstrations witnessed by the users, or the development of configuration settings through a prototyping process. Although such test tools will have a broad range of requirements, the critical requirements that should form the basis of the tool assessment and verification are:

- **Security of the test data:** facilitated by role based authorities and user restriction and the use of secure computer generated logs and audit trails. This may include the use of electronic or digital signatures. Where the test tools do not provide such features, the verification of the tools should focus on the establishment of security through the use of logical or physical security controls and procedural controls.

- **Ability of the test tool to enforce the defined test processes:** through the use of workflow or equivalent controls. Where this is not possible, the verification of the tools should focus on the ability of the tools to provide logs or audit trail evidence of the test processes.

The scope of the verification activities may be scaled based upon risk and should take into account the maturity of supplier practices and tools in the life sciences industry.

Test data managed within computerized test tools would not usually be determined to be an electronic record within the scope of predicate rules. The use of electronic signatures would not usually be determined to be within the scope of the predicate rules, e.g., the requirements of regulations such as 21 CFR Part 11 [17] would usually not apply. Specific considerations may apply in the case of testing of software that is part of a medical device.

Note: the development of software embedded within medical devices is outside the scope of GAMP® 5 [1].

Table 14.5 shows eight steps which should provide some guidance for the selection and verification of computerized test management tools.

Table 14.5: Selection and Verification Guidance

Assessment Step	Output
1. Understand general organizational test strategies that will leverage test automation and test automation objectives. At this point, it does not matter if the tests will be executed manually or automatically later.	Test objectives (general), test strategy (general).
2. Assess the risk of the business processes and software applications that will be automatically tested and document the result.	Risk assessment (general).
3. If test automation is considered to be useful, define the selection criteria for a test automation tool (see above). When doing so, the risk associated with such a tool should be assessed.	Selection criteria for a test automation tool, risk assessment (for test automation in general).
4. Select a test automation tool that fulfils the selection criteria of step 3. If an existing tool is being selected, understand if it has an established track record in regulated industries. For a tool that would be developed in-house, a written specification would be needed. The selection should be justified and well-documented. The general risk assessment of step 2 should be extended based on the selected tool.	Selected test automation tool, risk assessment (specific for the selected tool), specification (if needed).
5. Decide whether it will be sufficient to provide proof of an assessment that the tool is fit for its intended purpose or whether further verification is needed. The decision should be based on the risk assessment of the previous steps and the extent of any configuration and/or customization. Novelty and track record of the tool should be taken into account as well.	Decision on whether additional verification is required.
6. Define how to ensure the security of test data (or test artifacts, in general). Evaluate the usage of a version control system. Where a version control system will be used, plan and define its usage. Define the review and approval process of the test artifacts, if needed in combination with the version control system. Keep in mind that different artifacts may need different handling.	Test data security concept, SOPs, and guidelines.
7. Proof that the selected test automation tool fulfils the selection criteria of step 3. In addition, proof how the tool in combination with the processes defined in step 6 ensures the security of the test data and documentation. If needed, plan complementary measures (e.g., internal quality assessments).	Proof that the tool is fit for its intended purpose and test data are secure.
8. Only if a verification of the test automation tool is needed (see step 5), verify the tool against the specification written in step 4. Review and approve the verification documents.	Verified test automation tool.

Steps 1 and 2 will likely be done for any computerized test management tool.

Steps 7 and 8 are mutually exclusive and depend on the decision taken in step 5. In many cases, the test management tool will be used out of the box or will be slightly modified, so that a fit-for-intended-purpose assessment is sufficient. Typically, the complexity and risk are to be found in the test artifacts, not in the tool.

14.5.1 Test Management Tool Assessment

Test specifications should be defined and approved before testing is started, see Appendix D5 of GAMP® 5 [1]. In addition, the relation to the requirements should be given and defects should be identified.

For the main requirements for test management, the assessment and verification of test management tools should focus on:

- **Setup of the tools:** check and verify that the configuration (and/or customization) of the tool meets the main requirements. Check the defined roles and responsibilities.

- **Setup the requirements module:** check that all requirements can be baselined and signed off by the business users.

- **Test case creation:** check and verify that test case creation is uniquely defined and that a relationship to one or more requirements is provided.

- **Release and versioning of test specification:** check and verify that a proper release strategy is implemented and that tests can only be executed when the test case has been approved. After having started the test, the appropriate test case should not be modified, other than by creating a new version or variant

- **Test execution:** check and verify that results are properly documented and attachments (e.g., screen shots or reports) are securely linked and uniquely identified. If there are defects identified, they should be linked to the defect tracking module.

- **Test documentation:** check and verify that released documentation cannot be updated or modified without an entry to audit trail. It should be possible to generate printouts indicating if any of the data has been changed since the original entry.

- **Defects and follow-up actions:** check and verify that a controlled defect life cycle is implemented and that defect data cannot be deleted afterward.

- **Managing of test status:** check and verify that open test cases and unresolved defects can always be recognized.

- **Version control:** set up a version control:

    - Regarding the test management tool

    - Regarding the test specifications and test cases, managed by the system

Additional assessment and verification should be based on the specific requirements of the user's organization. Such verification may be required to be repeated after significant changes to the tool.

14.5.2 Test Automation Tool Assessment

Automated testing tools and test environments should have documented assessments for their adequacy.

Any use of test automation tools should be defined in the test strategy, and the tool should be held under configuration management.

Test automation tool used on a GxP regulated system should be subject to appropriate specification and verification based on a risk assessment. This should consider whether the test automation tool will be used to test multiple systems with varying risk, and the assessment and verification should consider the highest risk of system to be tested.

The test strategy should not initially be too closely aligned to the test automation tool, but should foresee its usage and be flexible enough to allow for both manual and automated test execution.

A detailed specification of the tool is typically not needed, but the operational area(s) and the objectives should be defined as part of the test strategy. In addition, the selection criteria (see Section 14.4 of this appendix) and any decisions based on an assessment process should be documented.

The level of required verification depends on nature of the tool and the degree of configuration or customization. Typically, if an existing tool (whether COTS or open source software) with an accepted track record is used, a proof of its fitness for the intended purpose (as defined in the test strategy) would be sufficient. This proof should focus on (but not be limited to) the critical requirements, including:

- Audit trail
- Security of test data
- Enforcement of defined test processes or work flow

Custom or highly configured test automation tools may, based on a risk assessment, require detailed requirements definition, and specific and sometimes extensive specification and verification.

Configuration management should be applied to any artifact associated with test automation, e.g., the test harness (if any), libraries with central functions, test scripts, or test data. If possible, these test artifacts should be managed by a version control system, which may be part of the test management tool.

In addition, guidelines or SOPs should be established to define review and approval processes for all the test artifacts.

15 Appendix E1 – Testing Configurable IT Systems

15.1 Introduction

Configurable IT systems can range from large multiuser, multifunction, complex systems through to small single user applications that have a very limited range of use. These do not meet the users' needs "out of the box," i.e., some form of configuration is required. This configuration can be in the form of user interface, user profiles, data, system functionality, or the system outputs in the form of reports or other information. When any configuration has been performed then the configuration should be tested to verify that it performs as intended.

15.1.1 Core Package

With a configurable IT system, there is a core package, which has been developed and tested by the supplier. This core package should be subject to an assessment to determine the quality of the development life cycle, which in turn will determine how much of the supplier or suppliers effort can be leveraged. This is already described in Appendix T2, and is outlined in Section 7 of GAMP® 5 [1].

15.1.1.1 Supplier Test Evidence

If the supplier has been assessed in accordance with Appendix M2 of GAMP® 5 [1], there will be an understanding regarding the supplier's software development method. One of the primary outputs from a successful assessment will be the determination as to how much of the core package development test evidence can stand alone and as such not require repeating when installed by the regulated organization. There may be a significant amount of system functionality that has already been tested and will not be subject to configuration, and as such would not give a different result if the test was repeated by the regulated organization. The scope and rigor of testing to be performed by the regulated organization will depend on the risk; the risk-based approach is described in Section 3 of this Guide.

15.1.1.2 Supplier Implementation Support

Suppliers of systems can provide an implementation package or methodology and may provide a complete implementation solution. This can save a considerable amount of time as the suppliers' knowledge of the system's functionality and capability will outweigh the regulated organization's knowledge. Although there are many advantages to this approach, every organization does not work the same, and as such different risks and issues are present. It is widespread for the system integrator to carry out unit, functional, and integration testing on behalf of the regulated organization, and then for the regulated organization to conduct user acceptance testing. Regardless of the division of testing responsibilities, it is essential that the responsibilities are clearly defined within the test strategy or plan and that the regulated organization maintains quality oversight of the system integrators testing. Reviews should be performed by the appropriate personnel to ensure that none of the user requirements are missed and not verified.

15.1.1.3 Terminology

The terminology should be clearly understood and detailed within test plans.

For larger configurable systems where different business functions may be being configured at different times, there can be multiple test cycles that are performed concurrently by different levels of expertise.

15.1.2 Testing Approach

The approach to testing configurable IT systems will depend on the complexity of the system and the way the system is to be implemented.

15.1.2.1 Small Configurable IT Systems

Smaller configurable IT systems such as a pharmacovigilance, trending, or data logging are generally implemented to meet a number of predetermined tasks or business processes with a limited scope. The different types of test cycles can normally be combined into a single test scenario which verify or challenge the configurations and be based on the business processes or predefined tasks.

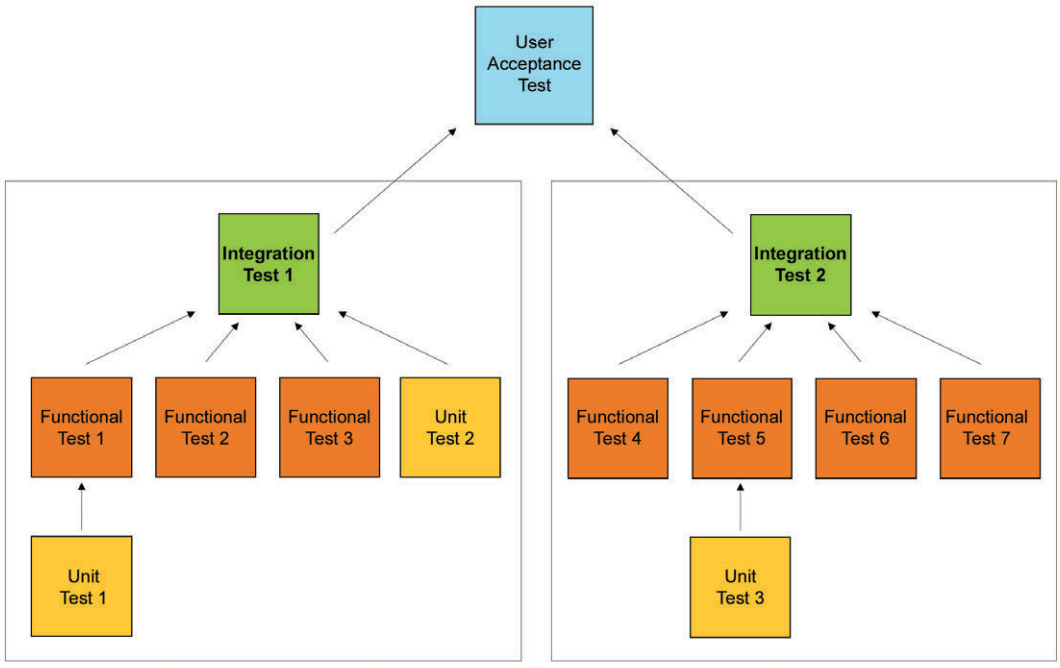
15.1.2.2 Large Configurable IT Systems

For large configurable IT systems, such as an ERP, LIMS, CRM, MES, etc., there are likely to be multiple test cycles, each containing different test types, that are performed in order to implement the complete systems. These activities may or may not be taking place concurrently.

15.2 What is special about these systems? – Risk Scenarios

The testing of configurable IT systems can be complex and involve a multi-cycle approach as shown in Figure 15.1.

Figure 15.1: Multi-cycle Approach to Testing



The multi-cycle approach will focus on different aspects of testing at the different levels.

15.2.1 Unit Testing

Although unit testing generally refers to testing where source code has been developed or modified and this appendix is regarding configurable IT systems, there is usually a degree of customization with large configurable IT systems. This level of testing is typically performed by a technical expert or SME and is specific and technical in format.

15.2.2 Functional Testing

At the functional testing level, verification is performed to ensure that the specific configuration has been performed to meet the predefined requirements. This level of testing will generally include challenging failure modes. The approach to testing will be prescriptive with the input and outputs being clearly defined.

15.2.3 Integration Testing

At the integration testing level, multiple elements of customization, configuration, and core package are brought together and tested. The approach to testing tends to be based on system (modules) or business processes. Testing can include both positive and negative cases and should not require technical expertise in the configuration of the system. Consideration should be given to the data that is required to adequately test the system at this level and the method for documenting this.

15.2.4 User Acceptance Testing

At the user acceptance testing level, the testing is of the system in its final, complete form and is best executed by the end users or process owners. The approach is generally end-to-end testing of business processes replicating real life operation. The testing will often take a number of days to ensure that time and data critical functionalities are verified. At this level of testing, the data used is critical and a documented approach to setting up this data is essential. Data may be imported or migrated from other systems and this testing should be performed with data that represents the data that will be used when the system goes live. User acceptance testing also can be performed as a scenario based testing, e.g., test for a scenario that would perform a 3 way match for an ERP system that would test for the creation of a purchase order and test it through the packing slip, and the creation of an invoice.

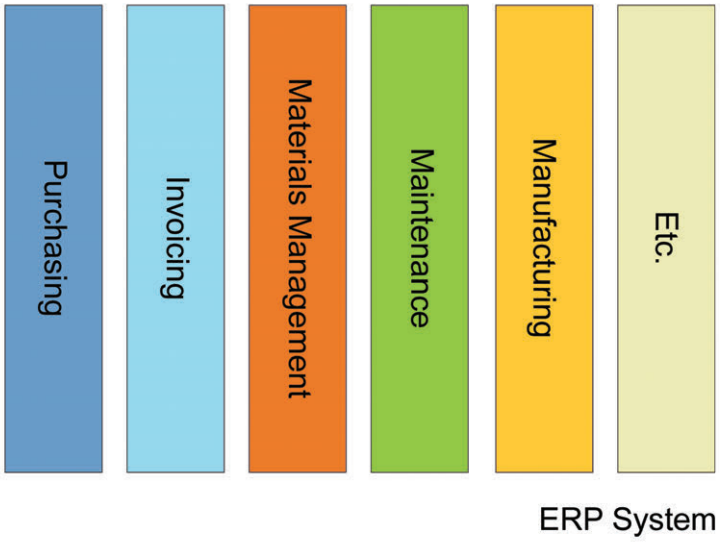
15.3 Business Process versus Functional Process Testing

At the integration and user acceptance level testing there are two basic approaches to testing.

15.3.1 Testing Aligned with System Function

With this test approach, each of the systems core functions are tested as separate units. For example, the purchasing module would be tested as a single unit from creation of a purchase requisition, authorization, creation of a purchase order, authorization of order, through to sending out to the supplier.

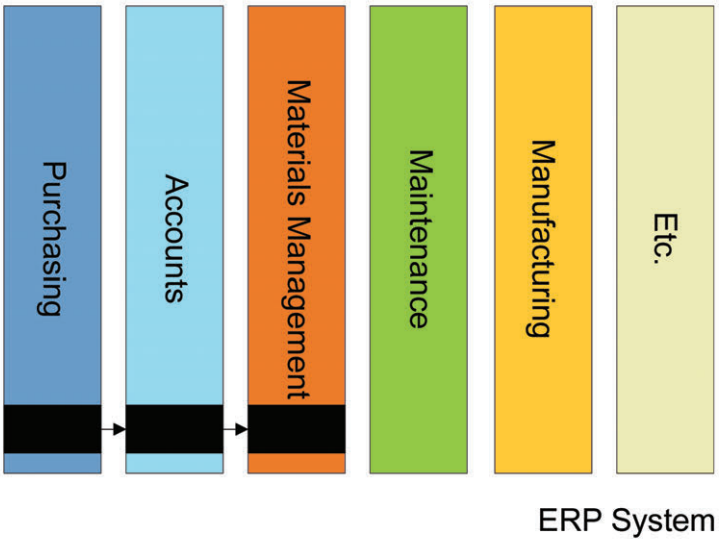
Figure 15.2: Testing Aligned with System Function



15.3.2 Testing Aligned with Business Process

With this test approach, each business process is tested and this may cross multiple functional units. An example may be the creation of a purchase requisition, through to goods receipt, invoice receipt, and then invoice payment.

Figure 15.3: Testing Aligned with Business Process (highlighted in black)



When planning business process based testing, it is important to efficiently plan test cases to ensure that:

- All necessary business process paths are tested appropriately (leveraging negative case testing where needed and supplier testing where possible)
- Test steps and the gathering of test evidence are not needlessly repeated

It may be necessary to repeat some test steps to be able to reach the point in the business processes where alternate or exceptional business process flows branch from normal processes. This is because many transaction based configurable IT systems do not allow business process execution to start midway through a process.

Business process testing is an application of branch coverage – a detailed worked example of this is provided in Appendix T1.

Table 15.1: Risk Scenarios for Configurable IT Systems

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Test script approval.	Test script creation and approval is rushed and does not provide adequate coverage. As complex systems are likely to have a layered approach, there should be a robust process for creation, approval, and issue of test scripts.	Careful thought should be given in determining appropriate approvers of the various levels to ensure that the correct resources can be made available.
Test defect management.	During a multilayered approach where different layers may be tested concurrently, it is essential that a robust process be put in place to manage test defects or the impact on other layers and process may not be know. Defects may or may not have an impact on different layers and processes and as such should be appropriately assessed.	Once the corrective action has been implemented, the test will likely require re-executing and may trigger regression testing within other functional areas of the system and across different test cycles.
Defect impact management at integration and user acceptance level testing.	Test defects and corrections can have an impact on multiple functions and require comprehensive retesting.	Ensure that the relationship between scripts and system functionality is understood. Carry out regular test reviews and ensure defect corrective actions are reviewed and approved by process leads.
Integration testing and user acceptance test script author's knowledge and experience.	Test script authors or executors may not have the appropriate system or business knowledge.	Test scripts may not be effective and adequate coverage may not be achieved.
User requirements.	User requirements are not defined in sufficient detail to enable adequate testing to be defined.	Define business process requirements and integrate into the risk assessment process and then incorporate into the testing strategy.
Supplier provides an implementation package.	The implementation package may not be specific for the life science industry and the assumed scope and rigor of testing and test documentation may not be adequate.	Map business process in accordance with intended system use. Risk assess critical steps and incorporate high and medium risk functionality into the test plans.
	Generic system implementation does not capture regulated organization's actual intended use of the system.	Map business process in accordance with intended system use. Risk assess critical steps and incorporate high and medium risk functionality into the test plans.
	Ensuring that there is adequate test coverage for the regulated organization's specific configuration.	Ensure that there are sufficient risk assessment and design review stages through the implementation.

Table 15.1: Risk Scenarios for Configurable IT Systems (continued)

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Supplier provides an implementation package. (continued)	Validation effort is incorrectly targeted. Classify requirements into primary, secondary, and other controls. Where primary controls are requirements that have direct impact on financial authorization, data integrity, or financial reporting or other regulatory requirements.	System requirements classified as primary controls will follow a more rigorous functional testing approach. Secondary controls are requirements that have an indirect impact on data integrity or financial reporting or other regulatory requirements. System requirements classified as secondary controls will be tested from a user perspective through scenario based testing. Other controls are requirements that have an impact on system configuration. System requirements classified as other controls are those requirements that will be detailed within the configuration management plan and verified during system configurations.
The core package which is to be configured may have been developed over a large period of time – it is essential that the core package functions as expected.	Core package has not been developed in a robust manner.	More testing required on core package.
	Supplier does not have documented system testing.	More testing required to provide objective evidence that system performs as expected.
Testing of the supplier core package.	Core package is not used as per design intent.	Use of supplier's testing potentially invalid. Bespoke testing will be required to ensure that all risk scenarios are covered.
	Supplier's design, development and test data not available.	Perform risk assessment to determine level of core application testing required taking into account the maturity of the application and supplier.
Functional test phase.	System configuration is contracted out to a third party or contract resource not familiar with the project standards.	Testing may not cover all scenarios include all scenarios identified by the risk assessment. Testing may be ineffective and would only be detected during later testing (if at all).
Application is available to many users though configuration of a widely available core application.	System available to non-authorized users.	Include security requirements in user requirements, design and verify during testing.

Table 15.1: Risk Scenarios for Configurable IT Systems (continued)

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
The need to manage multiple, complex test cycles.	Testing does not provide sufficient coverage.	Ensure that the output from risk assessments and user requirements feed directly into the test scripts and are governed by an adequate test plan.
	The impact of a change (either new functionality or defect correction) is not adequately tested across all test cycles and functional areas.	Ensure that the appropriate personnel are available to carry out a review on the impact that changes to the system may have on other functional areas or test cycles.

15.4 Typical Test Types and Phasing for Configurable IT Systems

Operation and disaster recovery for configurable IT systems require no special treatment and will be in accordance with the recommendations in the T appendices.

Table 15.2: Test Types and Phasing for Configurable IT Systems

Test Phase	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Test Planning	The creation of a test plan or strategy is an important task and will define the approach to testing at the individual layers and interaction between the various layers.	Group tests into interdependent tasks so that they can be managed more easily.
Integration Test Phase	Testing may be taking place for different business process at different times. If any changes are made, the test scenarios may no longer be relevant.	Risk ranking of functionality and detailed traceability matrix can aid impact determination when changes are made.
Business Process Based Testing	Can include both functionality that is intended to be used and functionality that is not intended to be used.	Determine critical functionality early through design review and risk assessment.
User Acceptance Test Phase	End to end business processes replicating real life time bound scenarios.	Schedule testing to coincide with time critical events.

15.5 Example ERP Test Script

Overall Test Execution Procedure
1. Sign the Protocol Signature Log after having reviewed the associated Validation Protocol – Documentation Requirements Section. 2. Log into ERP. 3. Perform required steps. Evaluate actual results against the “Expected/Observed Results” column in the Text Execution section of the test script. Mark the Pass/Fail column (with either a P or F). 4. Print the screen shots/screen prints as required in the “Expected/Observed Results” column of the test script. 5. Print any unexpected error messages that appear on the screen during the test execution and attach them to the test script. 6. Label attachments with initials, date, test number, and step number. 7. In the event of a potential deviation follow SOP-0000-0000-000-000.

Setup Data/Prerequisites

Test Script Reference	ERP_002.21	**Title**		Batch Search Strategy	
Objective	To verify the creation of a batch search strategy				
Requirement Reference	URS_002				
Run Number		**Start Date and Time**		**End Date and Time**	
Testing Scenario					
Create batch search strategy.					

Dependant Data			
Data Object	**Value/Code**	**Comments and Notes**	**Initials and Date**
Strategy type	CO01		
Order Type	FA		
Display UoM	B	Display in unit of entry for document	
Class	WE_DATUM		
Sort rule	DATUM_ LETZTER_WE		
Order number		Created in step 2 of this test case	

Master Data			
Data Object	**Value/Code**	**Comments and Notes**	**Initials and Date**
Plant	0001	Customer GmbH	
Material 1	1011718	Customer-tip 20 (1)	
Material 2	10023	Customer-tip 20 (25)	

Transactional Steps (Actions)

No.	Business Process Steps	Input Data/Special Information	Expected/Observed Result	P(ass)/ F(ail)	Initials/ Date
1.	Create batch search strategy.	Execute transaction COB1 On the initial screen, type in: • Strategy type Hit ENTER or click In the following screen, type in: • Order Type • Plant • Material 1 • No. batch splits (e.g., 1) • Tick box Dialog batch determination • Display UoM • Quantity proposal (e.g., 1) Hit ENTER or click Click on Selection criteria In the following screen, type in: • Class • Tick box Standard class • Status (e.g., 1) • Item (e.g., 1) Hit ENTER or click then click In the following screen, click Sort In the following screen, type in • Sort rule Save	Expected: The creation of a batch search strategy was successful. Observed: (add screenshot) Incident Report Reference (if applicable):		
2.	Test of new created batch search strategy	Execute transaction CO01 On the initial screen, type in: • Material 2 • Production Plant = Plant • Planning Plant = Plant • Order Type Hit ENTER or click In the flowing screen, type in: • Total Qty • Start = test execution date Hit ENTER or click and Save **(note the order number)** Execute transaction CO02 On the initial screen, type in: • Order Hit ENTER or click In the following screen, click on In the following screen, select the row with: • Material 1 And click on In the following screen, click on Strategy analysis	Expected: The batch search strategy functioned correctly. Observed: (add screenshot) Incident Report Reference (if applicable):		
Comments:					
Post Test Actions: None					
Acceptance Criteria: the batch search strategy creation works correctly. The created batch search strategy works correctly.		**Yes/No**	**Performed By:**	**Date:**	
Reviewed By QA/CSV:			**Date:**		

Approvals

Test Script Pre-Execution Approval			
Role	Name (Please print)	Signature	Date
Business Owner			
IT			
QA – CSV			

Overall Test Script Completion			
Overall result summary: e.g., all tests passed without incident/tests passed with minor deviations/test phase failed			
Role	**Name (Please print)**	**Signature**	**Date**
Tester			
Reviewed By			

15.6 Example Pharmacovigilance Test Script

Test Script Reference	R2.02.001		**Case Description**	On-line ADR, Serious, Unlabeled	
Requirements Reference	RS_124.23		**Test Description**	Initial Case Entry (Full Data Entry)	
Objective	Test case entry, completing required information in the various screens		**Acceptance Criteria**	Form correctly accepts data entry specified in each field. Role 2 is unable to encode information or close or lock a case. The completed form can be printed and routed.	
Run Number			**Prerequisites**	Workstation, User defined as role 2. Medwatch form of testcase is attached. Printer is installed.	
Start Date and Time			**End Date and Time**		
Tester Name	Tester 21	**User ID**		**Password**	tester
Report Type	LIT	**Version**	Initial	**Database**	

Test: DATA_ENTRY_R2.02						
No	**Step**	**Test Description**	**Test Instruction**	**Expected Result**	**Observed Result**	**P/F**
1	1	Logon to XX	Enter your assigned user Id, password.	XX displays a screen with menu toolbars		
2	2	Choosing a Case	Click on **File** select **Worklist**.	Worklist screen will be displayed		
		New Case				
3	1	Worklist	Highlight the case to be selected **DATA_ENTRY_R2.01**.	Case is displayed in lower window of the Case Selection screen.		
	2		Double click on the case number.	Folder opens with summary information, case form and revision folder.		
	3		Click on Case form and then press Open button.	Case form window of the selected case opens with the 7 folders.		
		General Information				
4	1		Select General tab of the case form.	General screen is displayed.		

Test: DATA_ENTRY_R2.02						
No	**Step**	**Test Description**	**Test Instruction**	**Expected Result**	**Observed Result**	**P/F**
4	2	General Information Section	Enter all missing, but available **general** information into corresponding fields from the attached MedWatch form section **G.3**.	All available data is displayed in corresponding fields of **general** information as indicated on the MedWatch form.		
	3	Reporter Information Section	Enter all missing, but available **reporter** information into corresponding fields from the attached MedWatch section **E.1** to **E.4**.	All available data is displayed in corresponding fields of **reporter** information as indicated on the MedWatch form.		
	4		If more than one, **Reporter** is indicated on the Medwatch form Select **New** at bottom of Case Form and repeat step 3.3.	Empty **Reporter Information** screen is displayed.		
		Patient Information				
4	1		Select the Patient tab of the case form.	Patient screen is displayed.		
	2	Patient Information Section	Enter all missing but available, **patient** information into corresponding fields from the attached MedWatch form section **A.1** to **A.4**.	All available data are displayed in corresponding fields of **patient** information as indicated on the MedWatch form.		
	3	Other Relevant History Section	Enter all missing, but available **patient other relevant history** information into corresponding fields section **B.6** and **B.7**.	All available data are displayed in corresponding fields of **other relevant history** as indicated on the MedWatch form.		
		Product Information				
5	1		Select the Product tab of the case form.	Product screen is displayed.		
	2	Product Information Section	**Suspect Drug** Select the radio button **Suspect Drug** and enter all missing, but available Product information into corresponding fields from the attached MedWatch form section C.1.	All available data are displayed in corresponding fields of product as indicated on the MedWatch form		

Test: DATA_ENTRY_R2.02						
No	**Step**	**Test Description**	**Test Instruction**	**Expected Result**	**Observed Result**	**P/F**
5	3		If more than one **Suspect Drug** is indicated on the Medwatch form, select **New** at bottom of Dosage Regimens section and repeat step 5.2.	Drug Coding (WHO-DRUG) windows will pop-up.		
	4		Highlight the correct drug and press the **Select** button.	Product information will be displayed.		
	5		Check the radio button **Suspect Drug**.	Suspect drug radio button is checked.		
	6		**Concomitant Drug** Select the **New** at the bottom of Dosage Regimens section. Enter all missing, but available Product information into corresponding fields from the attached MedWatch form section **C.10**; (This role cannot encode information).	Drug Coding (WHO-DRUG) windows will pop-up.		
	7		Highlight the correct drug and press the **Select** button.	Product information will be displayed.		
	8		Select the radio button **Concomitant Drug**.	Concomitant drug radio button is checked.		
	9		If more than one **Concomitant Drug** is indicated on the Medwatch form, select **New** at bottom of Dosage Regimens section and repeat step 5.6.	Empty **Product Information** screen is displayed.		
	10	Dosage Regimens Section	**Suspect Drug** Enter all missing but available Dosage information into corresponding fields from the attached MedWatch form section **C.2** to **C.8**.	All available data are displayed in corresponding fields of Dosage as indicated on the MedWatch form.		

Test: DATA_ENTRY_R2.02						
No	**Step**	**Test Description**	**Test Instruction**	**Expected Result**	**Observed Result**	**P/F**
5	11		If more than one **Suspect Drug** is indicated on the Medwatch form, select **New** on Dosage Regimens section and repeat step 5.	Empty Dosage Regimens screen is displayed.		
	12		**Concomitant Drug** Enter all missing, but available Dosage information into corresponding fields from the attached MedWatch form section **C.10**; (This role cannot encode information).	All available data are displayed in corresponding fields of Concomitant Drug as indicated on the MedWatch form 98/03424-USE.		
	13		If more than one **Concomitant Drug** is indicated on the Medwatch form, select **New** on Dosage Regimens section and repeat step 7.	Empty **Product Information** screen is displayed.		
		Events Information				
6	1		Select on the Events tab of the case form.	Events screen is displayed.		
	2	Event Information Section	Enter all missing but available Event information into corresponding fields from the attached MedWatch form section **B.1** to **B.4** and **G.8**.	All available data are displayed in corresponding fields of Event as indicated on the MedWatch form 98/03424-USE.		
	3		Select **Outcome of Event** from values list.	Selected outcome is displayed.		
			If Outcome of Event is **NOT** “Fatal” go to step 7. If Outcome of Event is “Fatal” go to step 4.	**Note:** If you proceed with step 7 please state in the Observed Results column “Not Applicable” for steps 4 to 6.		
6	4		Check “Death” in **Seriousness Criteria**.	“Death” box is checked and a yellow button **Details** appears.		
	5		Select the yellow button **Details**.	Death screen appears.		

Test: DATA_ENTRY_R2.02						
No	**Step**	**Test Description**	**Test Instruction**	**Expected Result**	**Observed Result**	**P/F**
6	6		Enter **Death Date** and check **Autopsy Done** radio button. Enter rest of available data from the attached MedWatch form.	Entered data is displayed in the corresponding fields.		
	7		If more than one **Event** is indicated on the Medwatch form. Select **New** at the bottom of Event Information section and repeat step 6.2.	Empty **Event Information** screen is displayed.		
	8	Event Assessment Section	Enter all missing, but available Event Assessment information into corresponding fields from the attached MedWatch form section Only Causality as reported can be entered or modified by this role.	All available data are displayed in corresponding fields of Event Assessment as indicated on the MedWatch form.		
		Analysis Information				
7	1		Select on the Analysis tab of the case form.	Activities screen is displayed.		
	2	Case Assessment Section	Select yellow button **Generate** beside the **Narrative** field.	Narrative is automatically generated and displayed.		
	3		Do first clean up of case auto-narrative (in **Narrative** field) and add relevant data manually for items that cannot be included automatically from the attached MedWatch form section **B.5**.	Manually added information is displayed.		
		Activities Information				
8	1		Select on the Activities tab of the case form.	Activities screen is displayed.		
	2	Contact Log Section	Leave empty.	Contact is empty.		
	3	Routing Comments Section	Routing information and comments entered so far are displayed.	Routing information is displayed.		

Test: DATA_ENTRY_R2.02						
No	**Step**	**Test Description**	**Test Instruction**	**Expected Result**	**Observed Result**	**P/F**
8	4	Action Items Section	Leave empty.	Action items are empty.		
	5	Case Lock/ Closure Section	Case Lock and Closure not accessible (This role cannot Close or Lock a Case).	No fields are displayed in this section.		
		Additional Information				
9	1		Select on the Add'l Info tab of the case form.	Add'l Info screen is displayed.		
	2	Notes and Attachments Section	Check information.	Notes and Attachments is displayed.		
	3	References Section	Enter MedWatch form # in the **ID#**.	# is displayed.		
	4		Select from value list [MW 3500A Mfr. Rpt #] into the **Type** field.	The selected type is displayed.		
		Print the Case				
10	1	Print entered case data.	Select **File** select **Print**.	Print Case window opens.		
	2		Select all.	All sections are marked as selected; not selected are Datasheet Only and Blind Study Product.		
	3		Press **Print** to print off case dump.	Print-out on defined printer destination.		
	4		Add print out to testscript as a reference result.	Printout added.		
		Routing				
11	1	Main Menu Bar	Select **File** select **Route** and **Forward Case**.	Message box "Save case before executing the Routing" appears		
	2		Press **Yes** button.	Case Routing window will appear.		
	3	Case Routing	Select "Coding" in the **Route to Next State** window.	"Coding" is displayed as the next state to which the case will be moved to in the workflow.		
	4		Leave **Route to User** as displayed.	The field displays default value (Any).		

Test: DATA_ENTRY_R2.02						
No	**Step**	**Test Description**	**Test Instruction**	**Expected Result**	**Observed Result**	**P/F**
11	5		Enter a "Routed to Coding" in **Comments**.	"Routed to Coding" is displayed in the comment.		
	6		Press **OK** button.	The routing window disappears.		
	7		Click on **File** select **Close**.	Message "Saved Case XXX-nnnn-nnnnnnn" appears.		
	8		Press OK button.	Case Form is closed – XX displays a screen with menu toolbars.		

Post Test Actions: None

By signing below, I indicate that I have received a complete set of test script, test case form, and test instructions and that I have executed each step of this document as instructed.

Tester			
Tested By:	Tester's Name (Print)	Tester's Signature	Date
Tester's Comments:			

By signing below, I indicate that I have reviewed this script and its attachments and I find it to be properly executed.

Reviewer			
Are all the expected results met?		☐ Yes ☐ No	
Incident Report Reference:			
Reviewed By:	Reviewer's Name (Print)	Reviewer's Signature	Date
Reviewer's Comments:			

16 Appendix E2 – Testing of Cloud Applications

16.1 Introduction

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly developed and implemented with minimal management effort or supplier (service provider) interaction.

Although different definitions of cloud computing exist, the cloud model promotes availability and is generally considered to be composed of five characteristics, three service models, and four deployment models (see also *The NIST Definition of Cloud Computing* – NIST SP 800-145 [18]).

The characteristics of cloud computing are:

- On-demand self-service – the regulated organization (often referred to as a consumer of cloud services) can easily provision computing capabilities, such as server time and network storage, as needed.

- Broad network access – capabilities are available over the network and accessed through standard mechanisms such as mobile phones, laptops, and Personal Digital Assistant (PDA).

- Resource pooling – the provider's computing resources are pooled to serve multiple users using a multi-tenant model with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. Examples of resources include storage, processing, memory, network bandwidth, and virtual machines, and the user may not know where these are located.

- Rapid elasticity – capabilities can be rapidly and elastically provisioned to quickly scale out or in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

- Measured Service – cloud systems automatically control and optimize resource use by leveraging a metering capability appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported providing transparency for both the provider and consumer of the utilized service.

The three cloud service models are:

- Cloud Software as a Service (SaaS) – the consumer is able to use applications running on a cloud infrastructure accessed from various client devices. The consumer does not mange or control the underlying cloud infrastructure or even individual application capabilities with the possible exception of limited user-specific application configuration settings.

- Cloud Platform as a Service (PaaS) – the consumer is able to leverage IT infrastructure hosted in the cloud to run applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure, including network, servers, operating systems, or storage, but still has control over the deployed applications and possibly application hosting environment configurations.

- Cloud Infrastructure as a Service (IaaS) – the consumer is able to leverage cloud based fundamental computing resources, such as processing, storage, networks, where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications and possibly limited control of select networking components (e.g., host firewalls).

Table 16.1: Summary of Cloud Service Models and Scope of Control

	Application		**Application Hosting Environment**	**Operating System**	**Infrastructure**
	Use	***Control***	***Configure***	***Control***	***Control***
SaaS	Regulated Organization	Service Provider	Service Provider	Service Provider	Service Provider
PaaS	Regulated Organization	Regulated Organization	Regulated Organization	Service Provider	Service Provider
IaaS	Regulated Organization	Regulated Organization	Regulated Organization	Regulated Organization	Service Provider

Note: both infrastructure as a service and platform as a service would be classified as IT infrastructure as defined in the GAMP® Good Practice Guide: IT Infrastructure Control and Compliance [10]. As such, IaaS and PaaS supporting GxP significant applications should be qualified (see Appendix E5) and applications running as software as a service should be validated, which includes appropriate testing.

The four cloud deployment models are as follows:

- Private Cloud – the cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premises or off premises.
- Community Cloud – the cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns.
- Public Cloud – the cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.
- Hybrid Cloud – the cloud infrastructure is a composition of two or more clouds (private, community or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

In the private cloud model, the regulated organization has direct control of the services being provisioned, whereas in the community, public, or hybrid cloud control should be established via a third party responsible for provisioning and managing the services.

This appendix focuses on the qualification and testing implications on the regulated organization (the consumer); however, the design quality and supplier (the provider) core software qualification and testing effort will impact upon the regulated organization's verification activities.

Computerized test management tools and automated test tools may be used to test cloud computing environments and may be leveraged extensively by the provider in the development of their SaaS application or the provisioning of their services. Regulated organizations should be able to use computerized test management tools in place of paper based testing.

However, there may be limited opportunities to leverage test automation unless the test automation tool is able to easily interface to the SaaS application user interface. It may be extremely beneficial to conduct performance/load testing of cloud computing environments, but in a public or community cloud this may be limited by the ability of the regulated organization's performance test tool to interface to the performance monitoring tools running on the provider's infrastructure. In this case, the service provider should be expected to provide infrastructure performance data, which should be analyzed alongside the regulated organization's performance data (as measured at the user interface).

16.2 What is special about these systems? – Risk Scenarios

Each cloud service can follow one of the four cloud deployment models. Each service/deployment model will have different risk scenarios depending on how they are deployed, e.g., on-premise or off-premise (see also *Guidelines on Security and Privacy in Public Cloud Computing* – NIST SP 800-144 [19]).

Risk management should be executed following the GAMP® 5 risk-management approach (Appendix M3 of GAMP® 5 [1]) and this should be used to determine the appropriate scope and rigor of qualification and testing.

The supplier assessment process should include all aspects of services provided in order to leverage testing of the cloud software, platform, and infrastructure.

Cloud based infrastructure (IaaS and PaaS) should be appropriately qualified and cloud based applications (SaaS) should be appropriately tested, either by the provider and/or the regulated organization (user). In order to ensure that this is done, it is important that regulated organizations are able to answer the following questions:

- What is the service being provided (IaaS, PaaS, or SaaS)?
- What is the GxP risk priority associated with the service?
- Who is provisioning the service?
- Who is managing the service?
- Where is the service located (on-premise or off-premise, and if off-premise, where)?
- What is the extent of the provider's infrastructure qualification (control and compliance) and is it appropriate to the risk priority?
- If SaaS, is the application multi-tenanted?
- What is the extent of the provider's software testing, does it cover the regulated organization's specific requirements and is it appropriate to the risk priority?

Answering these questions will help the regulated organization to understand whether they can exercise any effective control over the service and therefore fulfill their regulatory responsibilities. The answers also will help to identify where the regulated organization may need to perform additional testing of the application (it is unlikely that regulated organizations will be able to require additional infrastructure qualification activities to be performed, unless it is a private cloud model).

Cloud applications deployed in a private cloud should not be very different to other applications deployed in-house since the service is operated solely for one organization.

Cloud applications deployed in a community cloud are shared by other organizations within specific interest groups in the life sciences industry. As control of the application and the data will usually fall outside of the regulated organization's firewall, these applications are usually considered "open" systems. The need for more extensive supplier assessments may be considered, ensuring that the supplier developing, maintaining, and hosting the cloud application does so in a compliant and controlled manner.

It is expected that the regulated organization have established a requirement specification and use this together with the supplier documentation and risk assessment to verify the impact on the final test coverage, and procedures for operation and maintenance of the cloud solution from end users perspective.

Cloud applications deployed in a public cloud are made available to the general public or across large industry groups like the life science industry. A regulated user using such an application in a public cloud would not have effective ownership of the application and both the application and data would fall outside the regulated organization's firewall and their control, meaning that the application also may be considered an "open" system. Similar special considerations would need to be made for the supplier assessment on public cloud providers as for community could providers, but with special focus on how the supplier handles infrastructure qualification and software testing.

Cloud applications deployed in a hybrid cloud are deployed using two or more of the cloud deployment models; therefore, the risk management for this deployment model would be a combination of the risk scenarios stated above.

A typical approach to implementing the different levels of cloud computing by deployment model is summarized in Table 16.2.

Table 16.2: Typical Approach by Deployment Model

<table>
<tr><th></th><th>Private</th><th>Community</th><th>Public</th><th>Hybrid</th></tr>
<tr><td>SaaS</td><td rowspan="3">Treat as in-house</td><td>Validate as Open System</td><td rowspan="2">Focus additionally on supplier's testing and qualification</td><td rowspan="3">Combination of Risk Scenarios</td></tr>
<tr><td>PaaS</td><td>Focus on security and data integrity</td></tr>
<tr><td>IaaS</td><td colspan="2">Requires qualification</td></tr>
</table>

As some cloud providers create cloud solutions for organizations outside of the life science industries, careful consideration is needed regarding whether the provider under assessment has an understanding of the regulatory requirements specific to the regulated organization's, e.g., requirements for 21 CFR Part 11 [17] and EU GMP Annex 11 [8] compliance.

A risk assessment process should take into consideration whether:

- The provider's core application reflects the specific requirements of the regulated organization
- The provider's verification package (if any) reflects the regulated organization's configuration options in the application
- The provider's security accreditation
- Access control mechanisms
- Identification and authentication mechanisms
- Encryption mechanisms

Any shortcomings should be addressed and an appropriate risk mitigation strategy agreed upon. If gaps or deficiencies are discovered regarding the provider's service a determination should be made whether to:

- Train and support the provider under assessment
- Execute additional testing activities
- Select another provider
- Change to a different cloud model, e.g., IaaS or PaaS instead of SaaS

From a regulated organization's perspective, the configuration of cloud applications should be treated as GAMP® Software Category 4. Any custom development for GxP significant interfaces/data feeds that will communicate with the cloud application should be treated as GAMP® Software Category 5 and tested appropriately.

The risk assessment process also should determine which types of verification should be performed against requirements and/or software elements and the rigor of verification activities that should be conducted. Risk assessments should always be documented against the configurations documented in the package configuration specification documentation supplied by the provider.

Once a determination is made of the appropriate rigor and scope of the verification activities, the provider's standard verification activities should be assessed for any gaps. Such gaps should then be addressed by the regulated organization, e.g., additional testing of the application. A test plan or strategy should then be developed, identifying:

- Which of the provider's standard qualification and test activities will be leveraged?
- What additional qualification and test activities will be undertaken by the provider?
- What additional qualification and test activities will be undertaken by the regulated organization?
- What QMS should be followed in operation and maintenance?
- What operational manuals and SOPs/work instructions should be in place?

The key activities in managing these risks may include:

- Categorizing the systems that fall under cloud computing
- Selecting minimum (baseline) security controls
- Refining the security control set based on risk assessment
- Documenting security controls in system security plan
- Implementing the security controls
- Assessing the security controls
- Determining agency-level risk and risk acceptability
- Authorizing and monitor security controls on a continuous basis.

16.2.1 Infrastructure as a Service

Private IaaS does not differ greatly much from IT Infrastructures provisioned and managed in house and the same principles apply (for further information on infrastructure risk scenarios, see Appendix E5).

Table 16.3: Risk Scenarios for Private IaaS

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
The infrastructure is provisioned and managed by a provider	Provider does not follow regulatory or organization standards maintaining the infrastructure, potentially impacting data integrity and network availability/ performance.	All appropriate SOPs and SLAs are approved/effective and verified during the Installation testing process.
The infrastructure is managed by a provider off premises.	Hardware/Software components not readily available to internal IT/QA Staff, potentially impacting data integrity and network availability/performance if proper procedures are not followed.	All appropriate SOPs and SLAs are approved/effective and verified during the Installation testing process.

Public, community, and hybrid IaaS have additional qualification risks that should be addressed by the regulated organization.

Table 16.4: Risk Scenarios for Public, Community, and Hybrid IaaS

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Qualification of the IaaS is performed externally following the provider's Computerized Systems Validation (CSV) policy.	Provider does not have a CSV policy or applicable QMS.	Qualification is performed at the provider's site by the regulated organization's internal personnel and not the provider staff.
	Provider's CSV policy or applicable QMS is not adequate.	Additional qualification activities are performed at the provider's site by the regulated organization's internal personnel.
		Provide comments and edits to the provider's CSV policy or applicable QMS if possible.
		Ensure adequacy of provider's CSV policy or applicable QMS during the supplier assessment and perform periodic assessments.
		Use SLA to establish baseline CSV policy or applicable QMS.
Qualification Planning at the discretion of the provider.	Provider's qualification plan does not address all critical components and qualification is inadequate.	Have regulated organization's staff participate in the qualification planning for critical components.
Installation and build qualification typically performed by the provider.	Installation and build qualification is not properly performed, potentially resulting in hardware issues not discovered during the qualification.	IT/QA staff involved in the review of the installation and build qualification before and after execution.
		IT/QA staff performs an additional installation and build qualification to ensure all necessary hardware and software is installed.

Table 16.4: Risk Scenarios for Public, Community, and Hybrid IaaS (continued)

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Infrastructure controlled and hosted by provider during application testing.	Infrastructure not available during application testing, impacting testing resources and validation timeline.	Test prerequisites that require evidence that the infrastructure is available and qualified added to the test plans.
Configuration Management processes managed by the provider.	Provider's configuration management procedures are inadequate to control the test environment or the provider does not follow their procedures resulting in the configuration documentation not being properly updated.	Supplier assessment verifies the adequacy of the provider's configuration management procedures with respect to test environments (for further information, see Appendix T4).
		Periodic audits verify that the provider follows their procedures.
Performance Testing and Monitoring processes largely controlled by the provider with some shared client responsibility.	The provider's performance monitoring and testing processes are inadequate and/or not followed by the provider, potentially resulting in infrastructure performance issues or downtime if a failure is not identified or not acted upon in a timely manner.	Supplier assessment verifies the adequacy of the provider's performance monitoring procedures.
		Periodic audits verify that the provider follows their procedures.
		Stress Tests are included in the Operational Qualification to verify performance during peak usage times.

16.2.2 Platform as a Service (PaaS)

Most of the risks identified above as being applicable to infrastructure as a service are also applicable to private, public, community, and hybrid platform as a service (the GAMP® [10] definition of IT infrastructure also includes platforms). The following additional risks need to be considered.

Table 16.5: Risk Scenarios for Private, Public, Community, and Hybrid PaaS

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
The provider's platform does not properly support the application to be deployed (e.g., different version of operating system or virtual instead of a physical server).	Application performance issues or downtime occurs if not properly supported.	Supplier assessment determines whether or not the provider's platform can support the application selected or developed for implementation.
		Specific testing activities may be required to confirm compatibility.
		SLA states the organization's expected requirements governing the maintenance of the provider's infrastructure to support their application.
Storage of application and data outside of the regulated organization's control.	Application servers not partitioned correctly for shared applications, potentially impacting application security or reliability.	Supplier assessment determines whether the provider's approach to managing "Multi-Tenancy" (shared application servers) meets the regulated organization's requirements.
		SLA states the application management requirements expected by the regulated organization.
		Periodic supplier audits are conducted to verify that the application supported by the provider is being managed to the level of expectations stated in the SLA
		Security and reliability testing performed including the use of appropriate challenge and stress testing
	Database Servers not partitioned correctly, potentially impacting data integrity and security.	Supplier assessment determines whether the provider management of databases on shared database servers meets the regulated organization's requirements.
		SLA states the data integrity/security requirements expected by the regulated organization.
		Periodic supplier audits are conducted to verify that the application data supported by the provider is being managed to the level of expectations stated in the SLA.
		Data security testing performed including the use of appropriate challenge and stress testing.

Table 16.5: Risk Scenarios for Private, Public, Community, and Hybrid PaaS (continued)

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
PaaS applications deployed on a Public or Community deployment model are considered "Open" application since they are outside of the regulated organization's firewall.	Application and Data are compromised by outside attack impacting data integrity, patient safety and intellectual property.	Supplier assessment determines whether the provider's management of application and data security meets the regulated organization's application security requirements.
		SLA states the application security requirements expected by the regulated organization.
		Periodic supplier audits are conducted to verify that the application supported by the provider is being managed to the level of expectations stated in the SLA.
		Security testing of open system controls (e.g., encryption, digital signatures) performed including the use of appropriate challenge and stress testing. Testing such as external penetration test will be able to identify the vulnerability and system exploitation and compromise to security.

16.2.3 Software as a Service (SaaS)

Private, public, community, and hybrid software as service applications share the same risks as the IaaS and PaaS models described above and should be tested as any other GxP application. Public, community, and hybrid models have additional risks that should be considered in the development of the test plan or strategy.

Table 16.6: Additional Risk Scenarios for Public, Community, and Hybrid SaaS

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Configurable Components of SaaS application are typically implemented by the provider.	Configurable components of the application are not implemented properly or do not meet the regulated organization's requirements, impacting system functional use and data integrity.	Risk-based verification to be conducted on the Configuration Specification against the regulated organization's specific requirements.
		Functional testing with appropriate scope and rigor to ensure all requirements have been fulfilled.
SaaS provider not familiar with an outside application required to interface with the selected SaaS application.	SaaS application does not properly receive or does not properly process data sent from an outside application, interfaced to the SaaS application, impacting the functional use of the application and/or data integrity.	All interfaces requirements should be tested with appropriate scope and rigor to ensure all requirements have been fulfilled.
SaaS application does not provide a secure multi-tenanted solution (security model may be weak at the applications or database layer or may be incorrectly configured).	Loss of regulated organization's data security and confidentiality (potentially exposed to other users).	Test strategy should include negative case, challenge tests of security and backup/recovery.
SaaS application does not provide a robust multi-tenanted solution (impact of other users may cause performance to degrade).	Lack of acceptable performance during operational use or system failure leading to data loss.	Test strategy should include performance, and stress testing of contingency plans.

16.3 Typical Test Types and Phasing for Cloud Applications

Verification activities should include the following for each service model: configuration verification, functional testing (including security testing), and performance testing. In some cases, performance monitoring should be considered after the application or infrastructure is released for routine use in the operational environment.

As mentioned previously, the level of verification performed in the qualification/testing activities will be determined based on the risk assessments created against the configuration specification documentation and the audit performed on the supplier's core qualification package.

16.3.1 IaaS Test Types and Phasing – Private Cloud

As private cloud IaaS deployments do not differ much from a typical in-house managed infrastructure, test types and phasing will not detailed in this Appendix, for further information see Appendix E1.

16.3.2 IaaS Test Types and Phasing – Community and Public Cloud

Table 16.7: IaaS Test Types and Phasing – Community and Public Cloud

Test Phase	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Configuration Verification	Conducted at both the provider's and regulated organization's premises. May be conducted as part of design review or design qualification and infrastructure qualification.	Standard designs/builds may be leveraged to minimize the scope of functional testing required.
Functional Verification	Conducted at the regulated organization's premises. Performed after the infrastructure has been configured and installed. Extensive security verification will be conducted since the infrastructure is in a shared environment. Verification will include both positive and negative challenges.	Verification only on high risk Infrastructure components. Leveraging provider's core qualification to avoid redundant verification.
Performance Verification	Conducted at the regulated organization's premises. Performed after the infrastructure has successfully passed functional verification. Stress/Load testing should be conducted at usage times.	Verification can serve as user acceptance verification.
Performance Monitoring	Conducted at the provider's premises. Performed after the infrastructure has been released for routine use. Verification could include monitoring at peak transaction times.	If the provider has performed similar monitoring in the past, performance metrics may supplement the performance monitoring verification.
Disaster Recovery Testing	Testing is in two parts: the ability of the provider to recover the infrastructure, and the regulated organization's ability to recover the application on the provider's infrastructure.	Leverage provider's testing and ensure that SLA allocates responsibilities correctly.

16.3.3 PaaS Test Types and Phasing

Testing of PaaS deployments will not usually differ much from testing of IaaS deployments.

16.3.4 SaaS Test Types and Phasing – Private Cloud

Private cloud SaaS deployments will not differ much from a typical in-house managed configurable application test types and phasing will not be detailed in this Appendix. For further information, see Appendix E1.

16.3.5 SaaS Test Types and Phasing – Community, Public, and Hybrid Cloud

Table 16.8: SaaS Test Types and Phasing – Community, Public, and Hybrid Cloud

Test Phase	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Configuration Verification	Conducted at both the provider's and regulated organization's premises to ensure that the configured system is capable of meeting the regulated organization's specific requirements. May be conducted as part of design review or design qualification or as part of a thorough system demonstration.	Provider's standard configuration documentation may be leveraged to minimize the scope of the configuration verification required.
Functional Testing	Conducted at the regulated organization's premises. Performed after the application has been configured and installed, or after the SaaS is provisioned to the regulated user. Extensive security testing will be conducted since the application is in a shared environment and considered to be an "Open" application. Verification will include both positive and negative challenges.	Verification only on GxP significant and/or high risk configured application components. Leveraging provider's core test activities and documentation to avoid redundant test activities.
Performance Testing	Conducted at the regulated organization's premises. Performed after the application has successfully passed functional testing. Stress/load testing should be conducted at typical and peak loading times.	Performance testing can serve as user acceptance verification. It may be possible to leverage the provider's own performance test activities and documentation as far as the performance of the core application is concerned, leaving the regulated organization to focus on network connectivity performance.
Performance Monitoring	Conducted at the provider's premises. Performed after the application has been released for routine use. Verification could include monitoring at peak transaction times.	If the provider has performed similar monitoring in the past, performance metrics may supplement the performance monitoring.
Disaster Recovery Testing	Testing the ability of the provider to recover the infrastructure and the application.	Leverage provider's testing and ensure that SLA allocates responsibilities correctly.

17 Appendix E3 – Testing Analytical Instruments

17.1 Introduction

There is a GAMP® Good Practice Guide dedicated to the Validation of Laboratory Computerized Systems (2nd edition) [20], which deals with the complete life cycle of laboratory equipment and which will not be duplicated here.

This appendix is intended to build on existing guidance, taking a holistic approach to the risk scenarios specific to analytical instruments and looking at factors affecting both the instrument verification as well as the wider validation. Analytical instruments (depending on complexity) may possess a measurement capability and a user interface and/or interfaces to other systems, e.g., Laboratory Information Management System (LIMS).

Within the supplier product life cycle (which should be reviewed as part of the supplier assessment process, the extent of which should be determined based on risk) much of the operational testing can be completed, providing a record of instrument verification. This instrument verification may be leveraged by the regulated organization during their risk assessment, although some may need to be repeated (either by the supplier with regulated organization approval or by the regulated organization) depending on the extent to which the application is configured or customized.

Testing of interfaces and data storage and management, while not directly included within the instrument, also should be considered during the risk assessment.

The analytical methods associated with the instrument, e.g., sample preparation, assays, calibration buffers, and other reference standards, will most likely involve method validation within the regulated organization.

The testing of analytical instruments may be supported by the use of computerized test tools as an alternative to paper based testing. Opportunities to leverage automated test tools may be limited by the ability to interface to the instruments under test, but it is possible to interface to the computer based user interfaces of many such instruments. This can be beneficial where the same tests will need to be run through the operational life of the instrument (calibration, measurement checks, method validation through the use of test parameterization, etc.) In some cases, the instrument may provide a built-in facility to automatically run standard tests and checks.

17.2 What is special about these systems? – Risk Scenarios

Table 17.1: Risk Scenarios for Analytical Instruments

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Instruments require calibration (against reference standards).	An incorrect or inaccurate reference standard will result in unreliable measurements.	Calibration check at the median or upper limit of the operating range (as recommended by the supplier). Storage of the calibration data; monitor and compare for drift over time. Ensure reference standard is traceable to a national standard, and should be at least four times more accurate than the instrument being calibrated. For further information, see the GAMP® Good Practice Guide on Calibration Management (Second Edition) [21], and the ANSI/ASTM E617 [22].
Instrument configuration can radically affect measurement accuracy.	Incorrect configuration can give rise to inaccurate measurements or measurement output in incompatible units. An instrument used out of the box (i.e., not configured) may not operate correctly.	Check and record instrument configuration and document changes to it. Measurement check at the median of the operating range. Test to confirm output value is in correct format and units.
Instrument may require daily or weekly measurement checks to verify instrument remains within calibration.	Lack of measurement checks may result in inaccurate measurements or give rise to lack of confidence in critical data relied upon for batch release or product recall.	Testing is not required; the risk is mitigated by reviewing SOPs to ensure measurement checks are mandated at an interval agreed with the quality unit. Leverage drift testing done by the supplier where available to justify the proposed calibration regime. The use of control samples to monitor the performance should be considered.
• Accuracy • Deadbands or hysteresis • Measurement lag • Repeatability and reproducibility • Stability over time	Instrument specification does not meet requirements, e.g., variation between measurements could take the system outside the process limits.	Challenge the instrument across the full operating range (or as recommended by the supplier) as part of the calibration. Take readings both rising and falling, and assess the response time. Compare to process requirements.
Location of instrument can affect performance, e.g., vibration. For some instruments a special environment may need to be created.	Measurements are variable due to environmental impacts.	Installation verification required to ensure instrument is installed in accordance with manufacturer's instructions and guidelines.

Table 17.1: Risk Scenarios for Analytical Instruments (continued)

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Instrument may be moved (change of location or environment).	The performance of the instrument may be affected by the move and/or the new environment may not be suitable.	For instruments not intrinsically classed as “portable”, movement of the instrument from one location to another should be performed under change control and should include appropriate verification in the new location following risk assessment.
Instrument verification versus method validation.	An instrument may be composed of a measurement functionality, which may or may not involve complicated assays and standards, and an operational functionality, comprising the software, Human Machine Interface (HMI) and any network or data connections.	During test planning, have clear approaches for both the measurement qualification (method validation) and the instrument verification, acknowledging that verification activities need to be holistic, as software is required to use the hardware - they often cannot be separated. Method validation cannot be performed until the instrument is qualified.
Failure during use or verification.	The impact of an instrument failure may reduce confidence in analytical results generated.	To determine the extent of regression testing required following a breakdown, an understanding is required of the performance testing performed during verification, method validation and routine use (sometimes called point of use testing or system suitability tests). To simplify these decisions during maintenance when items are replaced, and to assure consistency, necessary tests for various common activities, such as replacement of detectors, pumps, filters, etc should be defined during system implementation.
Instruments may be used in series to provide a final analysis, e.g., sampler, High Performance Liquid Chromatograph (HPLC) then Mass Spectrometer.	An error in the first step may give rise to a cumulative error further down the process.	Verify each step (instrument) individually before proceeding to verify the combination of processes. Assess the sensitivity of the later steps to variation in input.
Consider standardizing on a specific configuration or firmware version.	The current version of the instrument may enter a non-supported phase (from the supplier). Subsequent purchases of instruments may be a later, non-compatible version.	The testing of second and subsequent instruments can be simplified if the user standardizes on a particular version. Decisions should take account of the of supplier’s release management strategy.

Table 17.1: Risk Scenarios for Analytical Instruments (continued)

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Detailed design specifications, e.g., Hardware Design Specification/Software Design Specification may not be available from instrument manufacturers.	Lack of confidence in the structured development may increase the risk priority of the system.	For a COTS system, the system maturity may be a more relevant input into the risk assessment/testing strategy. For a custom, highly configurable or high criticality system, consider supplier assessment and/or testing to a more detailed level, e.g., in addition to functional testing, include stress and boundary testing.
Instrument may not be "traditional" Windows based system.	The instrument may require special treatment for back-up and restore outside the existing IT infrastructure.	Procedures should be in place for back-up, archival, and restore, and these procedures need to be tested, to ensure data can be restored when needed. Test connection and interfaces.
Standalone instruments will store data locally rather than on a LIMS.	Increased risk of data loss.	Verify data retrieval functions. Test back-up and restore procedures.
Instrument may rely on another system for functionality, e.g., Chromatography Data Systems which manage methods and log data.	Interface to the other system is crucial to data integrity and correct functionality.	Subject to risk assessment, test the interface, including after power or communications failure; challenge the ability of the other system to manage multiple instruments simultaneously.
Achieving a "pass" result from the instrument may be crucial to releasing work in progress to the next stage.	A user may attempt to influence the result, by editing data.	Subject to risk assessment, challenge the data integrity features of the instrument, ensuring that any changes to the result are either prevented or are documented (irrevocably) in the audit trail.
Instrument requires users to have update access to test result storage area, as a result of design decisions made by supplier.	Test results can be modified after saving, possibly without an audit trail of such changes-potential compromises to data integrity.	May require use of a data movement tool to copy test results to a secured location where users cannot modify test results.
Use of remote diagnostics tool.	Potential compromises to data integrity of test results.	Careful security configuration management.

17.3 Typical Test Types and Phasing for Analytical Instruments

Table 17.2: Test Types and Phasing for Analytical Instruments

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Development (installation of measurement equipment)	• Confirm installation as per manufacturer's instructions. • Record serial numbers, versions, etc. • Perform regression testing to confirm that product quality has been maintained if equipment being back-fitted to an existing process.	Installation testing may well be done by supplier. Where regression testing is needed, re-use of original test scripts may reduce the work required.
Development (operation of measurement equipment)	Confirm control of instrument, and data integrity, by testing or verifying: • Configuration settings • Set-up routines • Start-up routines • Cleaning routines • User maintenance routines • Shutdown routines • Alignment of data from different sources • Calibration methods across full range of use • Verification of methods for detecting, and acting upon, analyzer failure • Security controls, including segregation of roles	Operational testing following installation may be done by supplier.
Operation (performance monitoring activities)	Continued monitoring of calibration. For systems that are composed of more than one module or component, some level of full system or "holistic test" should be considered. The use of control samples may be considered as mitigation.	The supplier may have generic metrology data on the measurement function against time across the full operating range. Use this data to determine the optimum intervals for calibration and calibration checks.
Operation (infrastructure use)	Testing of the interface to an existing IT network for data backup and/or archiving. Confirm data transfer is as expected, and that it recovers correctly after infrastructure outage.	Commissioning or laboratory team does not need to repeat tests done by the infrastructure team.

Table 17.2: Test Types and Phasing for Analytical Instruments (continued)

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Operation (following failure)	An amount of regression testing (also called re-qualification testing in a laboratory environment) may be required to confirm the instrument is fully operational after repair or replacement. Removal of the instrument from site for repair and/or extensive diagnostic evaluation may result in the need for repeating the installation verification prior to it being put back into service.	The supplier for the system should provide information about failures during routine maintenance, verification, the cause of the failure (if possible) and any remedial action taken to return a system to satisfactory performance, for inclusion in the impact assessment to determine the scope of the regression testing. Options for utilizing any "on line" or remote diagnostic capability, if available, for remote calibration and/or fault diagnostic should be considered as part of the system design. Work with the supplier to understand benefits and risks of implementing remote diagnostic capability, to eliminate the need for the instrument to leave site.
Operation (following disaster recovery)	Regression testing should be based on a risk assessment but is likely to include: • Confirm correct installation of any replaced items • Record serial numbers, versions, etc. • Confirm correct adjustment/ configuration/calibration of any replaced items • Perform appropriate testing to confirm that product quality has been maintained	Service level agreement should address which elements of post-disaster recovery testing are to be done by supplier. Where regression testing is needed, re-use of original test scripts may reduce the work required.
Retirement (data retention/migration)	• Confirm availability of all required data (e.g., from all sources) within final format • Confirm alignment/synchronization of all required data within final format	If the supplier provides a standard data retention or migration solution then it may be possible to leverage the testing done by the supplier during the development of this. In some instances, it may be simpler to retain the instrument to preserve the records rather than convert the data to a usable form.

18 Appendix E4 – End User Developed Applications

18.1 Introduction

End user developed applications, sometimes referred to as desktop applications, are usually developed for a very specific purpose often concerned with the analysis/manipulation of data in order that it can be “presented” in a more user friendly manner. The data is often a collation of information in order that decisions to be made from an examination of these several data sets can be done by analysis of the relevant facts from a single (or very few) sheets rather than having to search through many pages/screens of data.

The applications are generally run on desktop or laptop computers and are run by a single user at a time. There may be network drives associated for data backup and security, but the application can be considered stand alone.

The application is likely to have been developed around a COTS package, such as a spreadsheet or database package or business intelligence tool. The given package will have been configured or enhanced to create the application, but if the data within the application is to be used in any GxP related manner, particularly where the data analysis affects product release, then the application should be tested to verify that it performs as intended before the data can be used in any production scenario.

For further information on end user applications, see Appendix S3 of GAMP® 5 [1].

It is possible to use automated test tools, but the return on investment should be considered based upon how many users will use the application and how frequently the application will be updated (for further information on return on investment considerations, see Appendix T11). The ability of an automated test tool to interface to the application will depend on the user interface technology employed by (or developed for) the application. Where it is beneficial to use automated testing, the ability of the test tool to interface to the user interface may be considered as a functional requirement for the application.

18.2 Core Package

With any end user developed application, there is the core package which is likely to be a COTS package or possibly a generic package from a supplier which has been developed and tested by the supplier. The verification of the core package will need to be done by the regulated organization prior to development of the end user application. The verification activity will be package dependent and range from an acceptance of the package having been proven in use, likely for the well known spreadsheet or database packages, to requiring the full range of verification activities.

18.3 Testing Approach

As end user developed applications are usually created to allow specified users to undertake specific tasks, the desired output is already defined and the testing approach is to prove that the output is as expected at a functional level. Focus also should be given to applications ability to handle irregular inputs along with compatibility tests that ensure that the application functions correctly in the desktop environment.

18.4 Management of Testing

As end user developed applications can vary in complexity and application, the management of testing will need to be scaled to the regulated organization’s intended use.

18.4.1 Test Scripts

The creation of a test plan or strategy may be required for complex data analysis or manipulation, or applications with multiple functions that need to be tested. The more common approach, for less complex applications, is the creation of a test script or scripts. The scope and rigor of the testing should be based on the user requirements and the output of any risk assessments.

18.5 What is special about these systems? – Risk Scenarios

As end user developed applications are based on varying packages and have vastly differing levels of scope and complexity, there are a number of risk scenarios that can present themselves.

Table 18.1: Risk Scenarios for End User Developed Applications

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Core package or application functionality.	Failure to adequately test required functionality as it can be standard or configured through the core application in the form of macros or program coded functions.	Clear definition of functionality that is to be used should be documented and incorporated into the test scripts and any procedures for the use of the application.
Ability to handle irregular and unacceptable data inputs.	Failure to test irregular data input scenarios. It is important that not only is the software tested for the expected functions, but also tested to verify how it handles irregular inputs.	Failure modes need to be established and incorporated into any test scripts. Testing will cover both acceptable and unacceptable data inputs and formats and also will cover data at the boundary values of the working functions.
Compatibility between versions of the application and desktop builds.	Failure to allow for backward and forward compatibility of the software. For larger organizations, where multiple users may be using the same developed application across multiple sites, different versions of the desktop environment and different version of the desktop applications.	Consideration should be given to compatibility testing to ensure that the application functions correctly with different versions of the application, and that the application functions correctly with different versions of the core package and other desktop applications.
The core packages are available to most staff who have access to a computer and many people have the ability to create a desktop application	Applications are not developed or tested following life cycle approach. Applications are not tested to ensure continued fitness for purpose following upgrade of core package.	IT application strategy should be defined that prevents development of GxP application without going through some level of life cycle.
Core package development documentation is not available to user.	Development standard/testing of core package cannot be assessed as documentation is not available from the supplier.	Perform risk assessment to determine level of core package testing required taking into account the novelty of the package and supplier. Biggest risk is usually found to be user configuration and this is where the test effort should focus.

Table 18.1: Risk Scenarios for End User Developed Applications (continued)

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Application is available to many users through configuration of a widely available core package.	System available to non-authorized users.	Include security requirements in user requirements; design and verify during testing.

18.6 Typical Test Types and Phasing for End User Developed Applications

The following testing is typically applied to these applications.

Table 18.2: Test Types and Phasing for End User Developed Applications

Test Phase	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
User Acceptance Testing	Covers both the core and standard functionality and the developed functionality.	Use the output of the risk assessment and the user requirements specification and only test functionality that is going to be used. Restrict use by a standard operational procedure.
Disaster Recovery Testing	The ability to recover both the application and the data from what is likely to be a desktop image or laptop.	Leverage centralized backup tools.

Appendix E5

19 Appendix E5 – Testing of Infrastructure and Interfaces

19.1 Introduction

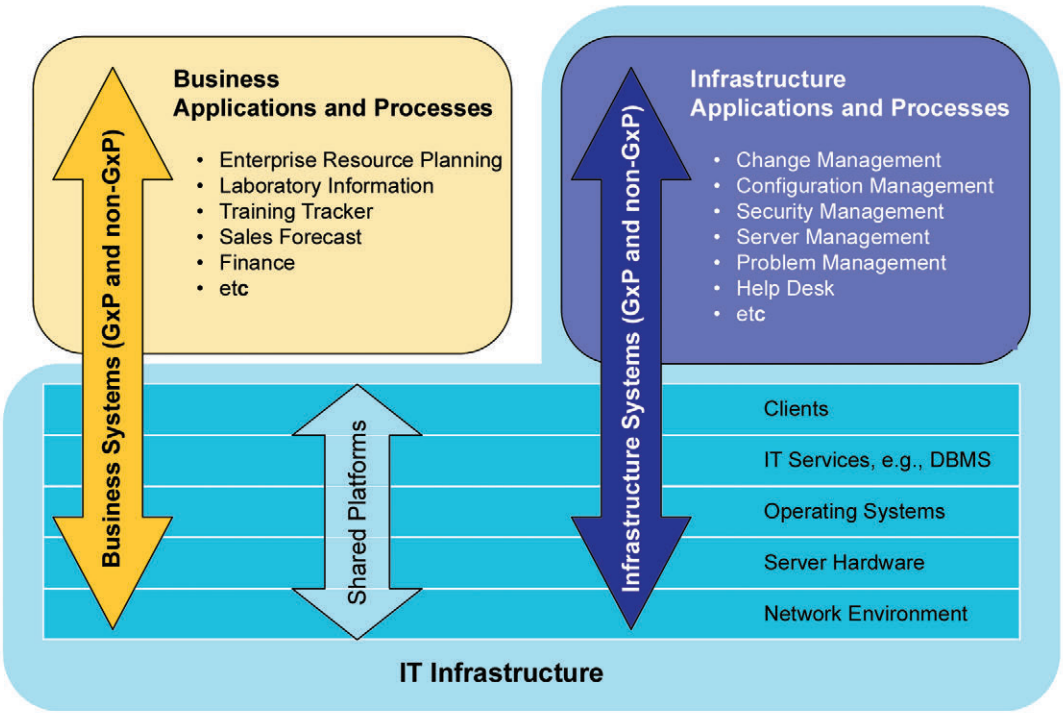
IT infrastructure should be brought into initial conformance through a planned qualification process, following the regulated organization's established standards and building upon acknowledged good IT practices. For further guidance on the general qualification of IT Infrastructure and maintaining a state of conformance, refer to the GAMP® Good Practice Guide on IT Infrastructure Compliance and Control [10].

19.1.1 IT Infrastructure

The scope of IT infrastructure is described more fully in the above referenced ISPE/GAMP® Good Practice Guide [10], but it basically can be defined as those platforms and components of the IT landscape that support specific software applications and/or are common to more than one software application. IT infrastructure may consist of hardware and/or software components. Different platforms and components (e.g., passive network components, servers, operating systems, and firewalls) generally have specific verification requirements, which may include an element of functional testing.

An overview of the scope of IT infrastructure is shown in Figure 19.1 (reproduced from the GAMP® GPG: IT Infrastructure Compliance and Control [10].)

Figure 19.1: Overview of the Scope of IT Infrastructure



19.1.1.1 Hardware Components

Most IT infrastructure hardware is commercially available, off-the-shelf equipment classified as GAMP® Hardware Category 1. In some instances, custom hardware may be developed to perform a specific function and this is GAMP® Hardware Category 2.

Where standard items of infrastructure hardware are to be used in a unique manner or for a purpose for which the piece of equipment was not originally intended (possibly operating outside the specified operating parameters),they cannot be justifiably claimed to be in common use. The risk likelihood is more suitably addressed by classifying the component(s) as GAMP® Hardware Category 2.

The build and installation of hardware components is usually verified using methods other than formal testing (e.g., verification against design specifications, visual inspection, etc., often referred to as installation qualification). Regulated organization procedures should state when it is appropriate to supplement basic installation qualification with additional functional testing.

19.1.1.2 Software Components

Most IT infrastructure software components are GAMP® Software Category 1 (Infrastructure Software) defined as:

- Established or commercially available layered software
- Infrastructure software tools

For further information on infrastructure software, see Appendix M4 of GAMP® 5 [1].

Infrastructure software components may be parameterized, configured, or custom developed and may provide GxP significant functionality independently of a traditional standalone software application. Such software should be classified as GAMP® Software Category 3, 4, or 5 and tested accordingly as part of a risk-based approach to validation.

This may include middleware, which can provide standard functionality across all areas of the business, both within the defined infrastructure boundary, but also outside the boundary. The use of standard middleware can now provide considerable basic functionality and this is often delivered via the organization's own Intranet or via a secure Extranet (operating across the wider Internet).

This is discussed below in the specific context of interfaces and this approach may be applied to other GxP significant functionality implemented using infrastructure software components.

While IT infrastructure software may have a significant impact on data integrity, it is significantly removed from any aspect of product quality and patient safety and this should be remembered when conducting any risk assessments and defining the scope of qualification and any testing activities.

19.1.1.3 Virtualized IT Infrastructure

Virtual IT infrastructure such as virtual machines (servers), virtual PCs, etc., are a combination of infrastructure hardware and software components in which software is used to create a representation of a hardware platform or environment. In virtualized environments, the hardware platform and software components are qualified as usual and the functioning of the virtual environment will be subject to risk-based functional testing. Novel virtualization technologies should be subject to greater initial testing.

Testing of the software applications that run in the virtual environment will provide the final verification. If an application is certified by the supplier to run in a specific virtual environment, no additional testing of the core functionality will be required, but where this is not the case additional testing of basic functions can provide useful evidence that the software application will run in a reliable, repeatable, and robust manner. This should be conducted prior to the regulated organization's testing of their specific user requirements.

For further details on the verification of virtualized IT infrastructure, see the *Pharmaceutical Engineering* article "Virtualization – Compliance and Control" by Ulrik Hjulmand-Lassen [23].

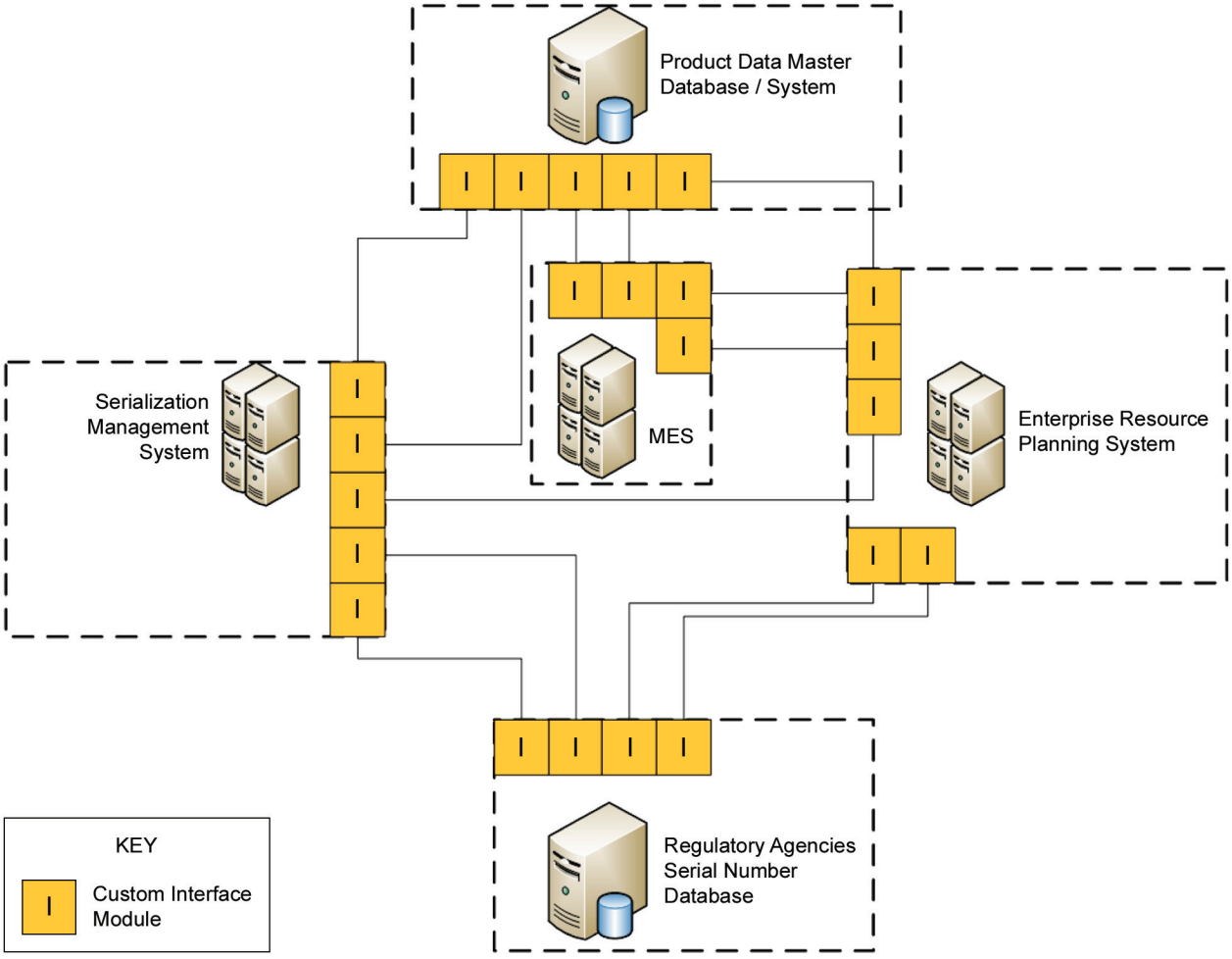
19.1.2 Interfaces

Interfaces allow separate applications or systems to exchange data, perform remote functionality, or support the interchange of data between different functions within the same application. Such interfaces may be tightly coupled with significant functional and timing interdependencies or may be loosely coupled using batch transfer of files or asynchronous messages and services. Depending upon the nature of the software involved, these also may be categorized as GAMP® Software Category 3, 4, or 5.

19.1.2.1 Point-to-Point Interfaces

Interfaces are often developed as point-to-point interfaces, where two computerized systems communicate directly with one another via interface routines which are part of the systems. Such interface software is configured or custom developed as part of each specific system and is generally tested as part of the test cycles associated with each system and as part of the validation of such systems. Depending upon the nature of the interface this may include unit, integration testing for custom developed interfaces, and functional testing for configured interfaces.

Figure 19.2: Point-to-Point Applications Specific Interfaces



19.1.2.2 Middleware/Service Oriented Architecture (SOA) Based Interfaces

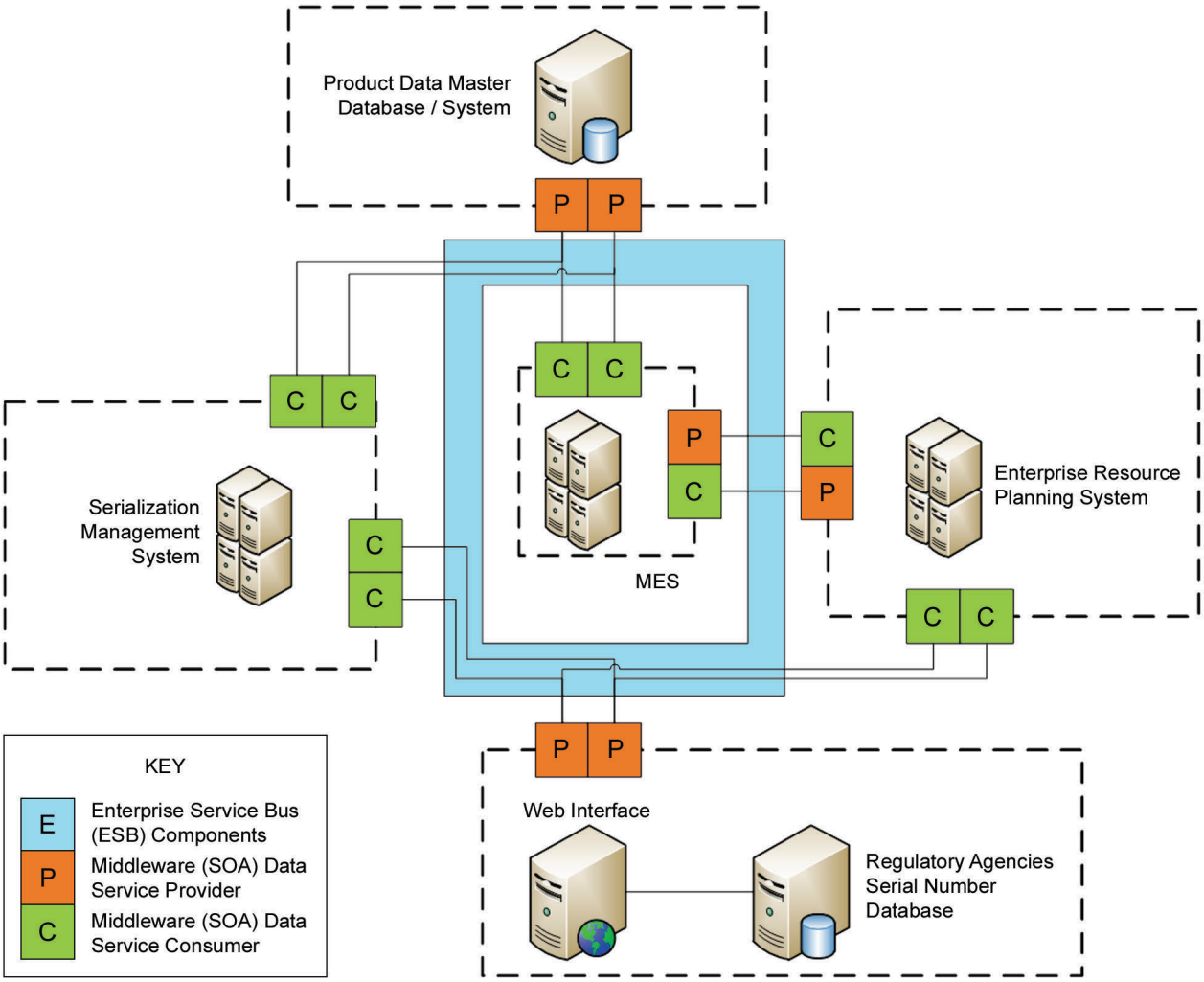
It is increasing common to interface systems using “middleware,” which may include the use of Service Oriented Architecture (SOA).

Middleware generally consists of software components that are independent of both systems, running on independent hardware components. SOA provides a framework for systems integration where systems group functionality around business processes and package this functionality as interoperable, reusable services which can be used by "consumers," which may be traditional users or other systems.

Using such technology, the individual software applications have generic interface routines which allow connections to be made to multiple other systems. The middleware provides data messaging using standard formats for data requests and responses, standardized data models (data abstraction), business process orchestration (including the resolution of "deadlock" where both processes/systems are waiting for the other to release resources), monitoring and event management/logging, error handling and security.

The "transport" of request and response messages is handled by the middleware layer, usually across an infrastructure based Enterprise Service Bus (ESB), which handles common functions, such as error handling, message routing, and message retries, etc.

Figure 19.3: Middleware/Service Oriented Architecture (SOA) Based Interfaces via an Enterprise Service Bus



The advantages of using middleware/SOA for implementing interfaces are that it:

- Allows the efficient reuse of software components
- Facilitates communication between disparate applications or languages

- Provides greater flexibility with respect to the management and conversion of data formats
- Permits separate computerized systems to be upgraded without a need to formally test other interfaced systems
- Gives improved robustness with respect to error handling and the management of asynchronous transactions

For further details on the broader qualification/validation of middleware and SOA, see the *Pharmaceutical Engineering* article "Controlling Service Oriented Architecture in Support of Operational Improvement" by David Stokes [24].

The use of middleware/SOA requires the testing of data provider and consumer service software used to connect each application to the ESB as well as the ESB components. This will require initial unit and integration testing when first developed/used, but when components are re-used there is little or no need for repeating unit testing. Integration testing is minimized through the use of standards and most subsequent testing is limited to functional testing.

This approach also can be used to test middleware used for other functions.

19.2 What is special about these systems? – Risk Scenarios

Testing of IT infrastructure and interfaces basically follows a "bottom-up" approach with lower levels of infrastructure being built, installed, and verified as required.

In most cases, the functional and performance testing of IT infrastructure components is not required. These devices are qualified following good engineering practice (developing specifications, verifying the correct build and installation against the specifications and recording key configuration parameters – see the GAMP® Good Practice Guide on IT Infrastructure and Control [10]) and any testing is limited to basic functions, e.g., "pinging" network devices to verify basic network connectivity. Functional testing is usually conducted as part of the validation of the overarching software applications.

Where middleware and SOA are used, some testing of the infrastructure components may be required. The effective unit and integration testing of middleware/SOA components reduces the need to execute such detailed tests for every system using these components and can reduce the overall burden of testing. However, this does need to be conducted independently of any of the interfaced applications and this needs separate test planning as part of the overall infrastructure qualification.

In some cases, a middleware/SOA based interface may:

- Change the format of data (e.g., integer to floating point)
- Manipulate data to ensure data compatibility between applications and data standards (e.g., round data up or down, substitute default values for "missing" data, calculate average values). In many cases, this may include the conversion of data to use a canonical data model with defined data standards including the use of standard metadata (including data formatting), data ranges, field sizes etc.
- Change the meaning of data (e.g., change metadata)
- Require a more granular approach to testing, i.e., testing that each software application interfaces correctly with the middleware layer. This may require a greater number of simpler test cases e.g., two tests to check that System 1 and System 2 interface to the middleware layer, rather than one larger test to check that System 1 interfaces to System 2.

Where such data has GxP significance, the basic qualification (installation verification) of the middleware/SOA components is insufficient and the functional testing of such middleware/SOA components should be conducted as part of a risk-based validation.

It is recommended that regulated organizations establish clear guidelines for the testing of middleware/SOA as part of an overall approach to the qualification or validation of such technologies.

Table 19.1: Risk Scenarios for Infrastructure

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
IT infrastructure hardware components – GAMP® Hardware Category 1. (Note that standard IT infrastructure hardware is part of most computerized systems.)	Known hardware from a mature supplier.	Basic qualification (build verification and installation qualification) and risk-based functional testing required to confirm the correct installation of hardware.
	Novel hardware from a mature supplier.	Inter-connectivity should be verified as part of basic qualification (power supplies, network cabling, etc). May not be required where the supplier has conducted and/or "certified" for interoperability for stated components or against stated standards.
IT infrastructure hardware components – GAMP® Hardware Category 2.	Custom hardware. Standard hardware operating outside supplier's specified parameters, e.g., • Above temperature limits • With standard operating system replaced by a non-standard operating system	May require extended performance testing to ensure correct operation across expected range of operating conditions. May require basic functional testing/ interoperability testing to ensure that standard operations function correctly when operating outside specified limits (also see above).
IT infrastructure software components – GAMP® Software Category 1. (Note that standard IT infrastructure software is part of most computerized systems.)	Known software from a mature supplier.	Basic qualification (build verification and installation qualification) and risk-based functional testing required to confirm correct installation of software.
	Novel software from a mature supplier. Software from a novel supplier. Open Source software.	Additional functional interoperability testing may be required to ensure compatibility with installed infrastructure platforms and components (software and hardware). Usually conducted in a test environment. May not be required where the supplier has "certified" for interoperability for stated components or against stated standards.

Table 19.1: Risk Scenarios for Infrastructure (continued)

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
IT infrastructure software components – GAMP® Software Categories 3, 4 or 5 (middleware/SOA).	Novel middleware/SOA platform from a mature supplier. Middleware/SOA platforms from a novel supplier.	Additional risk-based functional interoperability testing may be required to ensure compatibility with installed middleware/SOA platforms and components (software and hardware). Usually conducted in a test environment. May not be required where the supplier has conducted and/or "certified" for interoperability against stated standards.
	Novel middleware/SOA components (may be developed in-house or by third party). Open Source middleware/SOA software.	Risk-based testing of middleware SOA components against: • Industry standards • In-house standards (e.g., canonical data model standards) May require formal risk-based validation if components are expected to be used by GxP critical applications.
	Multiple versions of middleware/SOA components may be in operational use at any one time.	Verification that specific applications use specific versions of middleware/SOA components is required. Interoperability testing may be required to ensure compatibility with previously installed versions of the middleware/SOA platforms and components. Usually conducted in a test environment. Confirm rigorous version control and release management is in place.
	Interface (traditional point-to-point or Middleware/SOA based) error handling.	Error handling should be tested as part of risk-based negative case unit or integration testing.
	Business process orchestration (Middleware/SOA based). *(Note that traditional point-to-point interfaces are usually designed to operate synchronously or via batch transfer to avoid such issues).*	Orchestration of business process flow should be tested as part of integration testing. May need to be conducted as part of performance testing to ensure correct orchestration at expected loads.
	Business process deadlock *(where two or more systems/processes are waiting for the same resources to be released).*	Deadlock management (via timeout, retry or reset) may need to be tested as part of risk-based negative case testing.

Table 19.1: Risk Scenarios for Infrastructure (continued)

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
IT infrastructure software components – GAMP® Software Categories 3, 4 or 5 (middleware/SOA). (continued)	Inappropriate access to services allowing unauthorized consumers (users or other systems) to call services from providers (Middleware/SOA based). *(Note that traditional point-to-point interfaces are usually designed with implicit security models).*	Security configuration settings may need to be verified to ensure correct configuration. High risk security settings may need to be functionally tested as part of risk-based negative case testing.

19.3 Typical Test Types and Phasing for Infrastructure and Interfaces

Table 19.2: Test Types and Phasing for Infrastructure

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Project Phase (Basic Qualification)	• GAMP® Hardware Category 1 and 2 hardware components. • GAMP® Software Category 1 software components. May be conducted as part of design review or design qualification and infrastructure qualification.	Standard designs/builds may be leveraged to minimize the scope of functional testing required.
Project Phase (Functional Testing)	• GAMP® Hardware Category 1 and 2 hardware components. • GAMP® Software Category 1 software components. Basic core functions to confirm the correct build and installation, e.g., • "Pinging" for interface connectivity. • Basic navigation to verify system administration account set-up, permissions. • Specific commands executed to verify versions, mounting of volumes, etc. Simple functional testing may be conducted as part of Infrastructure Qualification. More extensive functional testing may also be executed.	Installation/commissioning using good engineering practice (often conducted by suppliers) can provide evidence that core functionality operates correctly. Automated testing can be used to conduct some functional testing.

Table 19.2: Test Types and Phasing for Infrastructure (continued)

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Project Phase (Unit Testing)	• Custom developed point-to-point interfaces. • Risk-based unit testing of novel. GAMP® Software Category 3, 4, or 5 IT infrastructure software components (middleware/SOA). May include structural testing or challenge (negative case) tests.	Not required where the unchanged interface or middleware/SOA component has previously been unit tested (either by a supplier or in-house).
Project Phase (Integration Testing)	• Custom developed point-to-point interfaces. • Risk-based unit testing of novel GAMP® Software Category 3, 4 or 5 IT infrastructure software components (middleware/SOA). May include interoperability testing to ensure that novel middleware/SOA components (or different versions of a middleware/SOA component): • Operate correctly in an existing platform. • Comply with industry or in-house standards. May include performance testing where data volumes or timing and synchronization may be a risk.	Not required where the unchanged interface or middleware/SOA component has previously been integration tested (usually in-house). May not be required where the supplier has “certified” for interoperability against stated standards. Simple point-to-point testing of a new instance of a standard interface or middleware/SOA component that has previously been integration tested (to confirm correct configuration). Comprehensive interoperability testing may be required in complex middleware/SOA architectures. The use of automated regression testing can significantly reduce the burden of interoperability testing.
Project Phase (Acceptance Testing)	All IT infrastructure and interfaces. GAMP® Software Category 4 or 5 IT infrastructure software components (middleware/SOA) may require separate acceptance testing if not developed to support an initial set of software applications. It is important that a representative group of stakeholders define the requirements and execute the acceptance testing of such middleware/SOA components. May include challenge (negative case) tests.	Acceptance testing of IT infrastructure, point-to-point interfaces and GAMP® Software Category 3 middleware/SOA components is achieved through the risk-based testing of the overarching software applications conducted by the process owners. Separate acceptance testing is not required

Table 19.2: Test Types and Phasing for Infrastructure (continued)

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Operational Phase (Regression Testing)	• Custom developed point-to-point interfaces. • Risk-based unit testing of novel GAMP® Software Category 3, 4, or 5 IT infrastructure software components (middleware/SOA). Will usually consist of positive case testing to ensure the correct functioning of unchanged infrastructure components.	Comprehensive regression testing may be required in complex middleware/SOA architectures. The use of automated regression testing can significantly reduce the burden of regression testing. Comprehensive performance monitoring of IT infrastructure and robust incident and problem management may be used to rationalize a reduced scope of regression testing.
Disaster Recovery Testing	Disaster recovery will be contingent on the business continuity plan. The infrastructure will be the first item to be recovered under the plan. Consideration about how and when this will be tested should be built into the business continuity plan.	Leverage original test specifications or protocols.

20 Appendix E6 – Process Control Systems

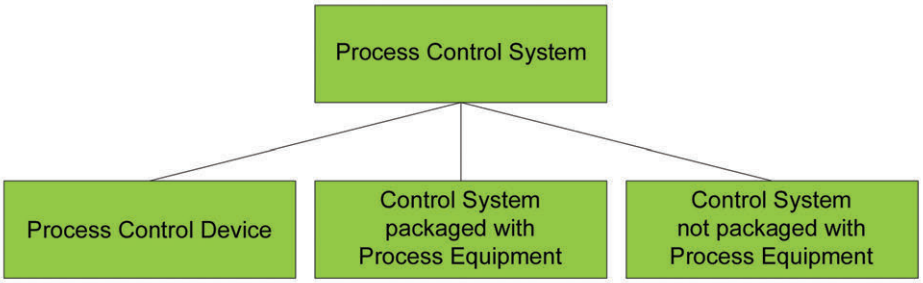
20.1 Introduction

Process control systems are defined as those systems that receive data relating to physical parameters from sensors and then process those inputs into related physical outputs in order to manage devices that control and monitor manufacturing processes and equipment.

20.1.1 Types of Process Control Systems

Within the ISPE GAMP® Good Practice Guide: A Risk-Based Approach to GxP Process Control Systems [25], reference is made to three generic types of systems.

Figure 20.1: Types of Process Control Systems



Note: this division into “device,” “packaged with process equipment” and “not packaged with process equipment” is independent of the GAMP® Software Category as defined in Appendix M4 of GAMP® 5 [1]. Table 20.1 provides some examples.

Table 20.1: Examples of Process Control Systems

System Type Detail of System Type	**Process Control Device** Single device from a single supplier comprises an entire standalone process control system.	**Control System Packaged with Process Equipment** Process control elements are embedded within process equipment into a "packaged" product from a supplier where the supplier quality management system provides assurance of the system in effect as though it were a commodity (may comprise multiple instruments, Programmable Logic Controllers (PLCs) and Supervisory Control and Data Acquisition/ Human Machine Interface (SCADA/HMI) components).	**Control System Not Packaged with Process Equipment** Process control system elements are developed independently in order to deliver a specific business process (may involve integrating multiple process control devices or packaged systems).
GAMP® Software Category 3 Non-Configured	Simple controller requiring parameter entry only (e.g., PID controller).	"Off the shelf" or packaged systems equipment where each instance requires parameter entry only.	Highly unlikely for a whole system though elements within the system – for example security setup – may fall into this Category).
GAMP® Software Category 4 Configured	Complex controller where library modules are selected and connected together to perform the required functions.	"Off the shelf" or packaged systems where each instance is configurable with appropriate library modules selected and connected together to provide the required control.	Much of a typical Distributed Control System (DCS) or SCADA system may fall within this Category with the process controls configured by selecting and connecting library modules.
GAMP® Software Category 5 Custom	Custom controller with functionality coded specifically for the application.	Custom machine or packaged systems with functionality coded specifically for the application.	Functionality coded specifically for the application such as custom sequencing and custom interfaces.

Note: some process control systems consist of elements of multiple categories. For example, a DCS application (typically classified as GAMP® Software Category 4) also may include a custom coded portion for interface to field device. When a system consists of multiple categories, the test planning for each portion of the system should be aligned with the relevant GAMP® Category based upon the risk level presented by that portion.

Computerized test management tools and automated testing can be used with process control systems and can prove beneficial where it is possible to:

- Automatically test the software related elements as with any other software based system. This is especially useful for control systems that are not packaged with process equipment or for testing the integration with enterprise level systems (see Section 20.1.3 of this appendix). Commercially available automated test management tools may not be designed to use the proprietary user interface used by many process control systems and this ability may be limited.

- Simulate inputs and record outputs (results) using a software based test harness. This facility is most likely to be provided by the supplier as part of the factory acceptance testing, but also may be available for use on site during commissioning.

20.1.2 Process Control Physical and Procedural Models

The ANSI/ISA-95 (S95) [26] standard overlaps with the ANSI/ISA-88 (S88) [27] standard for level 1 and 2 functionality. S88 defines physical and procedural models for batch processes which can be used to ensure modularity and flexibility within a control system design.

The physical model of S88 defines the hierarchy of equipment used in the batch process and is broken down into the following levels:

- ***Enterprise*** – defines the organization that owns the facility.

- ***Site*** – defines the location of the facility.

- ***Area*** – defines a specific section of the site, such as a particular section of the plant.

- ***Process Cell*** – contains all of the production and supporting equipment (units, equipment modules, and control modules) necessary to make a batch of product.

- ***Unit*** – is a major piece of equipment that performs a specific task within the batch process; a unit consists of equipment modules and control modules.

- ***Equipment Module*** – is a group of equipment that carries out a particular processing function.

- ***Control Module*** – is a single entity that performs state-oriented or regulatory control; a control module typically interfaces directly to plant input/output (I/O).

The procedural model of S88 defines the control that enables the equipment in the physical model to perform a process function and is broken down into the following levels:

- ***Procedure*** – is a sequence of unit procedures required in making a batch; it orchestrates the control of the equipment in the process cell.

- ***Unit Procedure*** – is a sequence of operations; it controls the function of a single unit. A unit may have more than one unit procedure, but only one unit procedure may control the unit at a time.

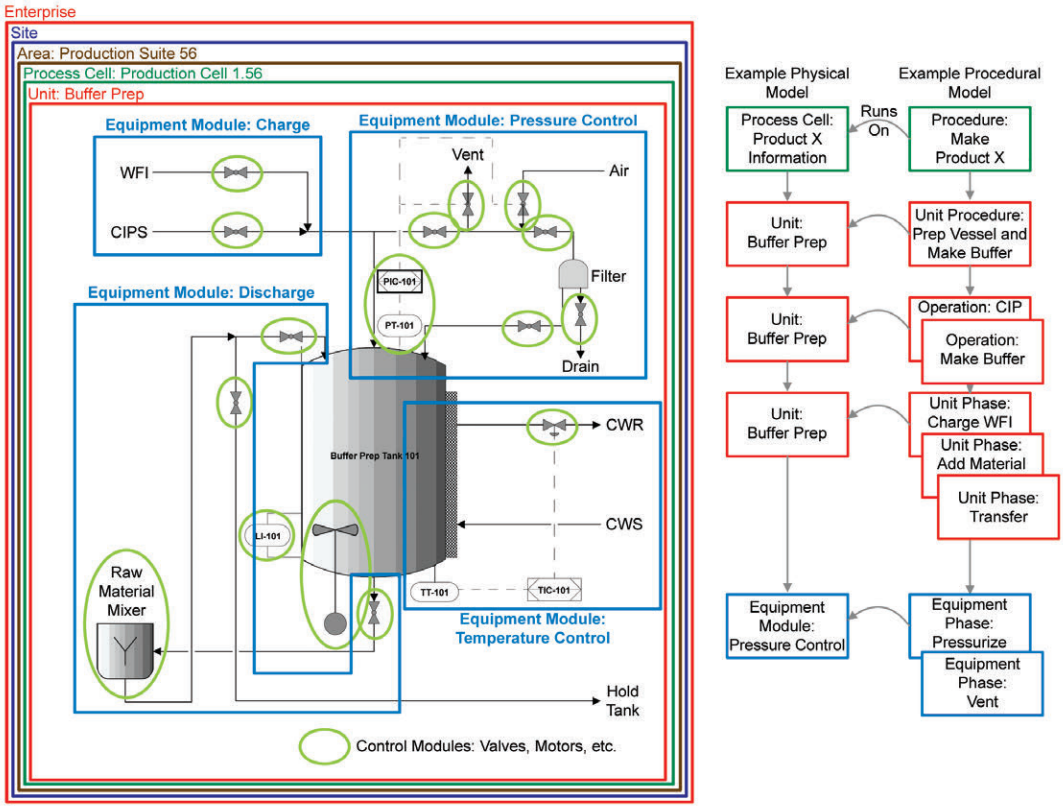
- ***Operation*** – is a sequence of phases; it controls a portion of the unit function.

- ***Unit Phase*** – performs a unique or independent process function of a unit; it coordinates the control of equipment modules and control modules.

- ***Equipment Phase*** – performs a simple process function on an equipment module; it coordinates the control of control modules.

Figure 20.2 illustrates a typical S88 model as it applies to a simple buffer preparation process.

Figure 20.2: Typical S88 Model of a Simple Buffer Preparation Process



20.1.3 Interfaces between Process Control and Enterprise Level Systems

S95 is the industry standard for the integration of enterprise and manufacturing control systems and their supporting IT systems. It defines a hierarchical model for different levels of system and can be used to help define the interfaces between process control systems and other layers of automation:

- ***Level 4*** – consists of business planning and logistics functions. These are activities not directly involved in manufacturing so they may need consideration from other parts of the organization. Example activities at this level are inventory tracking and management, collecting and maintaining quality control files, production scheduling, equipment maintenance scheduling, and capacity planning.

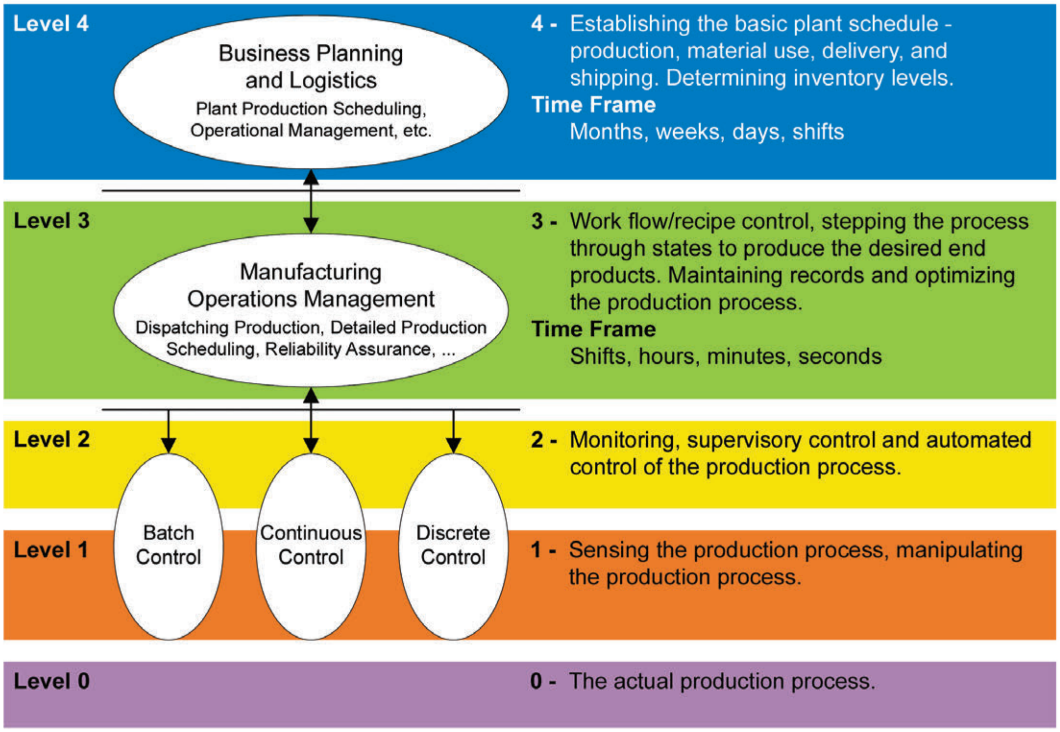
- ***Level 3*** – consists of manufacturing operations management functions. Example activities at this level include production reporting; collecting and maintaining production data; performing data analysis; managing the immediate production schedule for the area; managing personnel issues for the area; and optimizing costs for the area. Systems commonly included in level 3 are Enterprise Resource Planning (ERP) systems, Manufacturing Resource Planning (MRPII) systems, Manufacturing Execution Systems (MES), and process data historians.

- ***Level 2*** – consists of monitoring, supervisory, and automated control. Systems include data collection systems, SCADA systems, engineering workstations, and HMI systems.

- ***Level 1*** – consists of sensing and manipulating functions. Systems include controllers and PLCs.

- ***Level 0*** – consists of the production process elements themselves such as the I/O and sensors for discrete, continuous, and batch manufacturing including smart devices.

Figure 20.3: Model of Levels of Systems



Process control systems typically cover levels 1 and 2 and often also include some of the functionality of level 3, such as recipe management and data historian functions. The interfaces to the enterprise systems and the associated infrastructure should be tested (see Appendix E5). Testing of the enterprise applications should be conducted independently and the integration of the overall system should be tested as part of process validation.

20.2 What is special about these systems? – Risk Scenarios

Specific features of process control systems include:

- Multiple components and multiple suppliers – compared to other system types, process control systems may be made up from many more components of different types, often supplied by more than one supplier, requiring support and input from multidisciplinary teams. This can lead to a large number of additional interfaces, both in terms of interfaces between components and in terms of interfaces between different parties involved in producing the system.

- Interfaces to a physical process being controlled in real time – process control systems differ from other system types because there is a physical process being controlled in real time. As a result, process control systems often control parameters or attributes which can have a high impact on product quality. In addition, the process control system may have an impact on non-GxP requirements associated with the physical process.

- Interfaces to the process equipment – process control systems differ from other system types because they need to interface to the process equipment. The process control equipment as well as the process equipment work together to provide an overall system capability. The overall system capability is addressed by commissioning and qualification.

- Degree of embedding with the process equipment – the process control system can be more or less embedded with the equipment depending on the type of system being considered.

- Batch process modular architecture – process control systems for batch processes often are designed using a modular approach (such as ANSI/ISA-88 (S88) [27]) as described in Section 20.1.2.

- Interfaces introduced by phasing of delivery – large process control systems are often delivered in phases: either vertically (with I/O and control module layers followed by equipment modules, sequencing and batch control) or horizontally (with one plant area followed by another). This process introduces additional interfaces that should be considered between the "operational" and "in development" sections of the overall system at any time.

Table 20.2: Risk Scenarios for Process Control Systems

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Multiple components and multiple suppliers	Multiple components of different types result in effort wrongly focused leading to either insufficient effort or excessive effort on a specific item.	Categorization of software and hardware feeding into risk management and verification planning.
	Multidisciplinary input required resulting in misunderstandings and gaps in communication.	Careful communication, planning, and definition of scope, roles, interfaces and points of contact for each test phase.
	Multiple suppliers lead to lack of clarity about responsibilities – particularly for interfaces – and inadequate functionality in final system.	Care in ensuring that planning documentation sets out test scope and responsibilities correctly.
	Interface between components fails to function correctly.	Ensure test specifications cover interfaces in sufficient detail to mitigate identified risks. Ensure adequate integration testing of interfaces and the overall process.
Physical process being controlled in real time	System fails to measure critical parameters correctly resulting in poor control or possibly release of poor quality product.	Verify mitigation strategies such as independent measurements, calibration and maintenance routines, interlocks or alarms.
	System fails to control critical parameters resulting in poor quality product.	Verify control strategies, including mitigation strategies such as creation of independent checking functions, interlocks or alarms. Periodic testing in operational phase to detect deterioration in system performance.
	System malfunction or poor performance affects quality of product.	Verify mitigation strategies such as redundant components, interlocks or alarms, disaster recovery procedures, preventative maintenance requirements. Testing should include system response in reaction to failure of individual system elements.
	Change in environmental or process input conditions moves system outside its ability to produce good quality product.	Test full range of likely conditions. Verify mitigation strategies such as process or equipment design features, additional controls on process inputs, interlocks or alarms.

Table 20.2: Risk Scenarios for Process Control Systems (continued)

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Physical process being controlled in real time (continued)	May not know required process capability (e.g., if this is a multi-purpose facility).	Verify designed capability against which new processes can be assessed to determine suitability. Assess need for additional testing if a new product requires operation outside of the demonstrated capability.
	Physical process dynamic response may impact process control system performance.	Risk assessment of scenarios around dynamic response to identify aspects that may be verified using off-line or simulated conditions and which should be executed using actual process materials and conditions.
	Process control system has impact on environmental and safety risks of the physical process.	In addition to testing of the GxP impact described in the GAMP® Good Practice Guide on Process Control Systems [25], the test planning may need to consider testing the aspects of the application which impact environmental or safety aspects following appropriate guidelines.
Interface to process equipment	Impact of the process control system on the product quality	Test planning should consider the impact classification of the process system being controlled when deciding on test scope and rigor
	User requirements define the needs of the overall system.	Planning process should consider integration of process control system verification with the commissioning and qualification of the overall system.
Degree of embedding with the process equipment	Much more of the process expertise may reside with the equipment supplier who may be unfamiliar with working in a GxP environment.	Planning process should be clear about the split of verification responsibilities between regulated organization and supplier.
	Difficult to separate operation testing of the process control system software independently from the process equipment.	Verification includes review of process control system against design in addition to functional testing of the integrated process equipment system.
Batch process modular architecture	With a modular architecture, lower level functionality is relied upon for higher level functions.	Testing of higher-level functions (such as phases) should verify that the appropriate lower-level functions (such as control modules) are invoked correctly. Full testing of the lower level functionality does not need to be repeated.
Interfaces introduced by phasing of delivery	Second or subsequent phase identifies problem with an element that is already installed in a prior phase.	Change management strategy for how such changes will be back-tracked into existing elements including those in other test environments.

Table 20.2: Risk Scenarios for Process Control Systems (continued)

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Interfaces introduced by phasing of delivery (continued)	Installation of new elements has negative impact on those already present.	Consider use of standard (or pre-tested) export/import methods to minimize impact on existing configurations and, therefore, optimize regression testing requirements. Verification strategy may involve making the required changes within a test environment and verifying the change process as well as the end result. Verification strategy may need to include significant elements of impact analysis and regression testing.
	Incomplete system delivery does not meet the intended use of the delivery phase.	Verification should address the intended phases, capability required at each phase, and establish release criteria for each phase.

20.3 Typical Test Types and Phasing for Process Control Systems

The following table shows typical test coverage in each test phase and opportunities for (risk-based) leveraging of supplier's testing and other testing efficiencies. The test phases in this table are for a more complex process control system, a control system not packaged with process equipment with GAMP® Software Category 5 elements. Not all of these test phases will be applicable to all process control systems. Applicable test phases should be determined and documented in the project validation plan.

Computerized test management tools and automated testing can be used with process control systems and can prove beneficial where it is possible to:

- Automatically test the software related elements as with any other software based system. This is especially useful for control systems that are not packaged with process equipment or for testing the integration with enterprise level systems (see Section 20.1.3 of this appendix). Note: commercially available automated test management tools may not be designed to use the proprietary user interface used by many process control systems and this ability may be limited
- Simulate inputs and record outputs (results) using a software based test harness. This facility is most likely to be provided by the supplier as part of the factory acceptance testing, but also may be available for use on site during commissioning

Table 20.3: Test Types and Phasing for Process Control Systems

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Project Supplier's Application Software Module Testing	Module testing generally covers: • Module data handling • Interfaces to other modules • Operator interfaces • Module functionality Failure paths and response to fault conditions should be included within the tests.	Existing modules should be re-used where possible. Previously completed module tests can then be referenced rather than repeated. Risk-based test scope such that only modules of specific complexity/novelty/ risk priority undergo formal module test.
Project Supplier's Module Integration Testing	Module integration testing generally covers: • Correct operation of interfaces between modules Failure paths and response to fault conditions should be included within the tests.	Depending on complexity/novelty/ risk priority associated with interfaces between modules it may be possible to move directly from module testing to system integration testing.
Project Supplier's System Integration Testing	Integration testing generally covers: • Hardware/installation • I/O interface(s) • Operator interface • Interfaces to other equipment • System functionality • Data handling functions Failure paths and response to fault conditions should be included within the tests.	In order to leverage supplier's integration testing to minimize FAT testing, these tests should be documented to GxP standards. In addition, the software should be under configuration management during the supplier formal verification activities.
Project Factory Acceptance Testing (FAT)	Typical test coverage is as per supplier's integration testing.	The required coverage should reflect the relative risk priority associated with the system element under test. For example, where simple or low risk priority elements have already been covered by supplier testing, it may be appropriate for the regulated organization to select only a small sample for repeats. The required coverage can, of course, be increased if problems are found within the initial sample. In order to use suppliers' internal test results, they should be adequately documented to allow reference back to the results during an inspection. In determining the required coverage, the regulated organization should base decisions on the risk assessment output taking into account both the potential effect on product quality and safety resulting from the process and the intrinsic risk probability associated with the method of implementation.

Table 20.3: Test Types and Phasing for Process Control Systems (continued)

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Project Site Acceptance Testing (SAT)	SAT should focus on aspects that are impacted by the move from the FAT environment to installation at the site. The coverage should include: • Checking that the full system including hardware, software backups, and documentation has been delivered to site in a condition suitable for its intended purpose. • Checking the software configuration management documentation to ensure that the installed software matches what was tested during FAT. • Checking that the site environment is suitable for the specification of the installed equipment (temperature, humidity, dust, vibration, interference, etc.) • Checking that the equipment has been correctly installed. • Demonstration that the system is still operating as it was when accepted during factory acceptance testing (typically by re-recording system components and versions and by repeating a small sample of factory acceptance tests on site). • Testing any elements which could not be adequately tested in the factory acceptance test environment (typically interfaces to other equipment, networks, etc.) • Re-testing following remedial action on any elements, which were subject to conditional release at the end of factory acceptance testing.	Successful FAT tests should not be repeated if the transition from factory to site has no adverse impact on the function being tested. Typically those aspects are related to hardware and installation rather than software. This includes tests for: • System displays and navigation • Operator data entry • Data formatting and quality checks • Checks of calculated values SAT should concentrate on those system functions that cannot be fully tested or relied on simulation during FAT. SAT should include any tests required for process and equipment qualification activities (installation, operation and performance qualification) to avoid any duplication of on-site testing. **Note:** Be aware, that FAT/SAT from a supplier's point of view is a commercial test to finalize delivery. If the FAT/SAT is to be leveraged for qualification, the regulated organization should have a documented plan to use the FAT/SAT test results and provide additional on-site testing if the supplier testing does not meet all of the qualification needs.
Operation (Performance Monitoring Activities)	Performance Monitoring: • Process control systems typically include one or more networks and monitoring network performance is recommended. • Monitoring of calibration drift and sensor health may be required. • Monitoring of system health/capacities may be required. • Monitoring of manufacturing deviations.	Performance Monitoring Applications are available which can continuously monitor network performance, system health, sensor performance, etc. and alert administrators to unusual conditions.

Table 20.3: Test Types and Phasing for Process Control Systems (continued)

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Operation (Adjustments within the Previously Tested Design Space)	No additional testing required except where needed to confirm adjustment has been correctly made.	Ensure operational change management system allows for reduced burden of verification in this case, provided that the change is confirmed as not affecting the design space.
Operation (Following Repair/ Replacement of Items or Disaster Recovery)	Regression testing should be based on a risk assessment, but is likely to include the following: • Confirm correct installation of any replaced items. • Record serial numbers, versions, etc. • Confirm correct adjustment/ configuration/calibration of any replaced items. • Confirm continued availability of data from before disaster recovery. • Confirm availability of newly collected data. Perform appropriate testing to confirm that product quality has been maintained.	Service level agreement should address which elements of post-disaster recovery testing are to be done by supplier. Where regression testing is needed, re-use of original test scripts may reduce the work required. When like for like replacements are made the testing needed may be reduced.
Operation (Changes to Design Space)	System Modifications and Upgrades: • Recording of any new/updated hardware and/or software components. • Tests to demonstrate the new or modified functions. • Regression tests to demonstrate existing functionality is unaffected by the change.	System Modifications and Upgrades Assessment of the supplier's own testing of the modification. These will be implemented under change control, which should include a risk assessment of the impact of the change on the process control system. The results of the risk assessment should identify the required regression tests.
Retirement (Data Retention/ Migration)	Retirement and associated testing is addressed by Appendix T9.	Retirement and associated testing is addressed by Appendix T9.

21 Appendix E7 – Packaged Systems

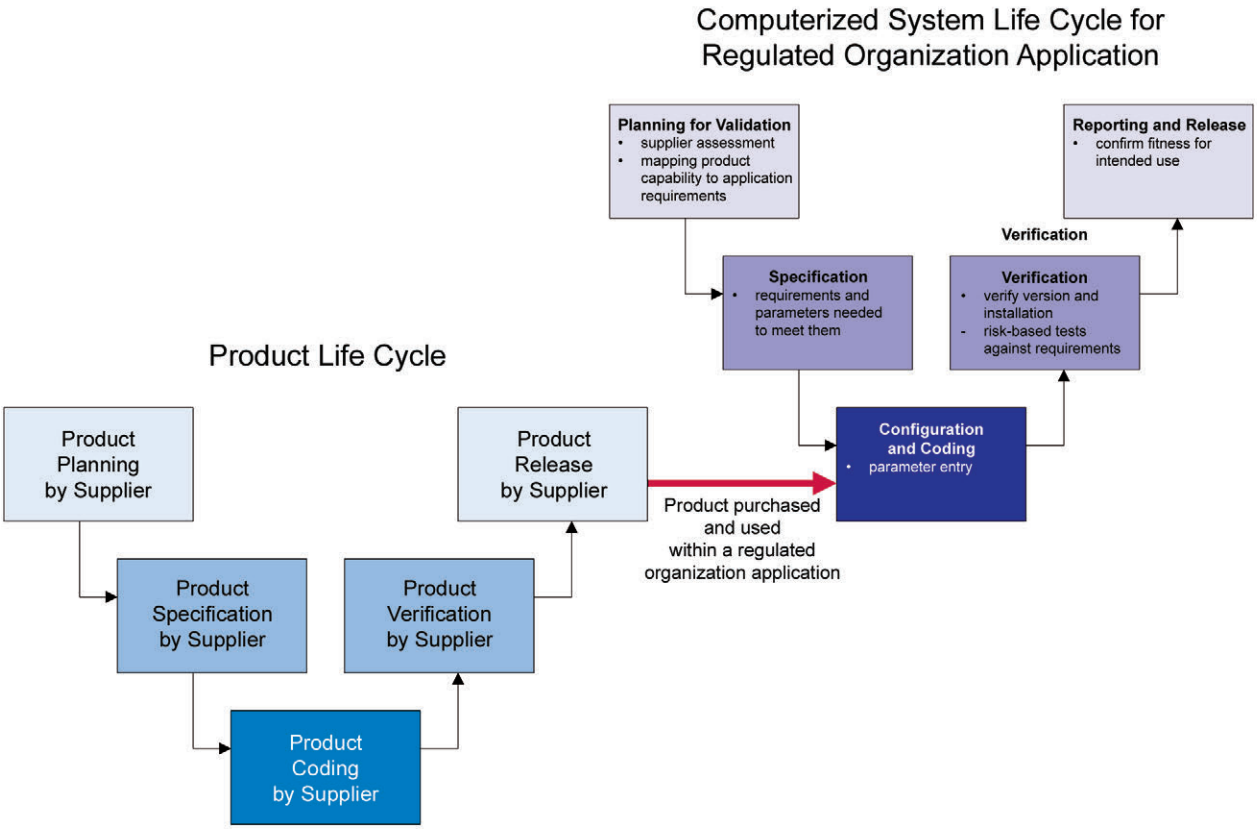
21.1 Introduction

This appendix addresses issues, which are particular to cases where a control system is delivered packaged with the process equipment it controls. The regulated organization is responsible for ensuring that the packaged solution is suitable for its intended use and to test this in accordance with the specific use of the packaged system.

For a particular regulated organization application, such systems may fall within a wide spectrum of software categories from standard software (GAMP® Software Category 3 – e.g., a sterilizer that has thousands of identical units already in use and which requires only parameterization to select the required cycles) through to custom written software (GAMP® Software Category 5 – e.g., a one-off machine to make a particular type of diagnostic test strip). For both regulated organization and supplier, the appropriate software categories should be identified.

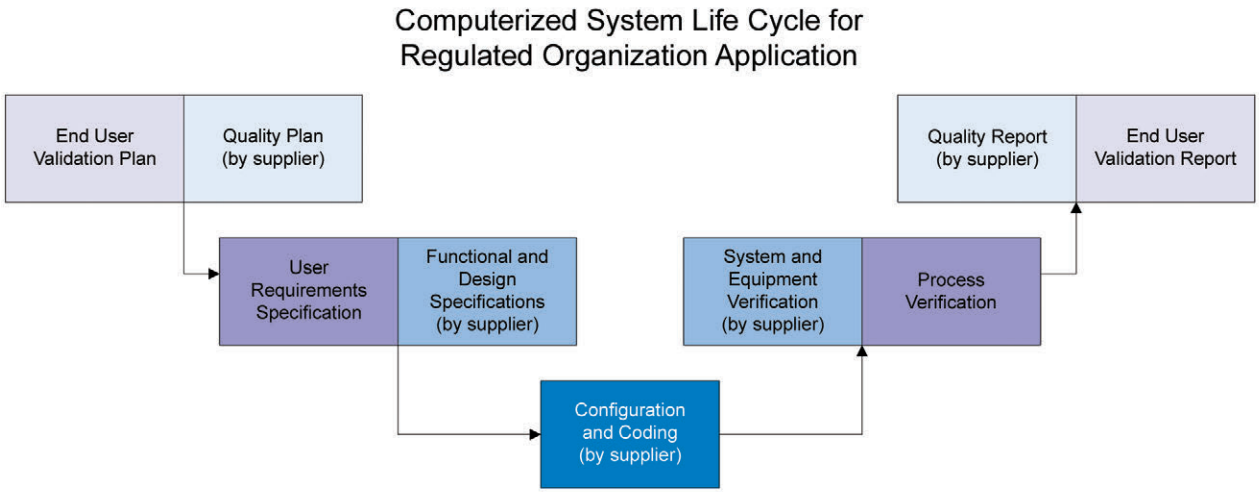
In the first case (standard product, parameterized for application), the expectation is that the supplier will have already gone through a full product development life cycle in order to release the standard product for use. Provided that the supplier assessment process demonstrates appropriate quality of these product development life cycle activities, the application life cycle can be limited to mapping of product capability onto user application requirements, specifying and implementing the required parameterization for those requirements and verifying that the requirements are met.

Figure 21.1: Product Life Cycle for a Standard Product



In the second case (custom written application), the supplier and regulated organization will need to work together within a single application development life cycle.

Figure 21.2: Product Life Cycle for a Custom Application



The impact that earlier software life cycle phases will have upon the overall testing regime should be considered. With packaged systems the supplier assessment process is key to understanding the maturity of the supplier and novelty of the base product, the quality and rigor of the development life cycle plus the history of use in similar applications.

The user requirements specification content also will be significantly different to other such requirements, concentrating upon functional process requirements/outcomes rather than more detailed task driven steps. These requirements will need to be specific and include sufficient detail to be testable with clearly definable acceptance criteria.

Regulated organizations may require specific security/administration functionality to be implemented on the packaged system platform and suppliers can achieve this through a mixture of configurable items and custom coding. Care should be taken in identifying this and other forms of post production phase modifications that have not been controlled as part of the supplier's development life cycle and ensure that such modifications are defined, risk assessed, and a robust test plan developed to ensure that they are thoroughly tested in the operating environment (the basic principles related to software categories 4 and 5 should be applied). Similarly, if physical security should be provided to supplement weak logical security, the implementation of this also should be verified.

This information from supplier assessments, user requirements, and analysis of customization will then drive the risk assessment process used to develop the test strategy

The need to test the software with the associated process equipment limits the use of test automation due to the difficulty of interfacing automated test tools to the process equipment. There is also a risk associated with allowing automated tests to execute where this also will exercise the process equipment and this risk is best mitigated by manual testing. However, it may be possible to use a computerized test tool to support manual testing, where the test tool is used in place of paper based testing.

21.2 What is special about these systems? – Risk Scenarios

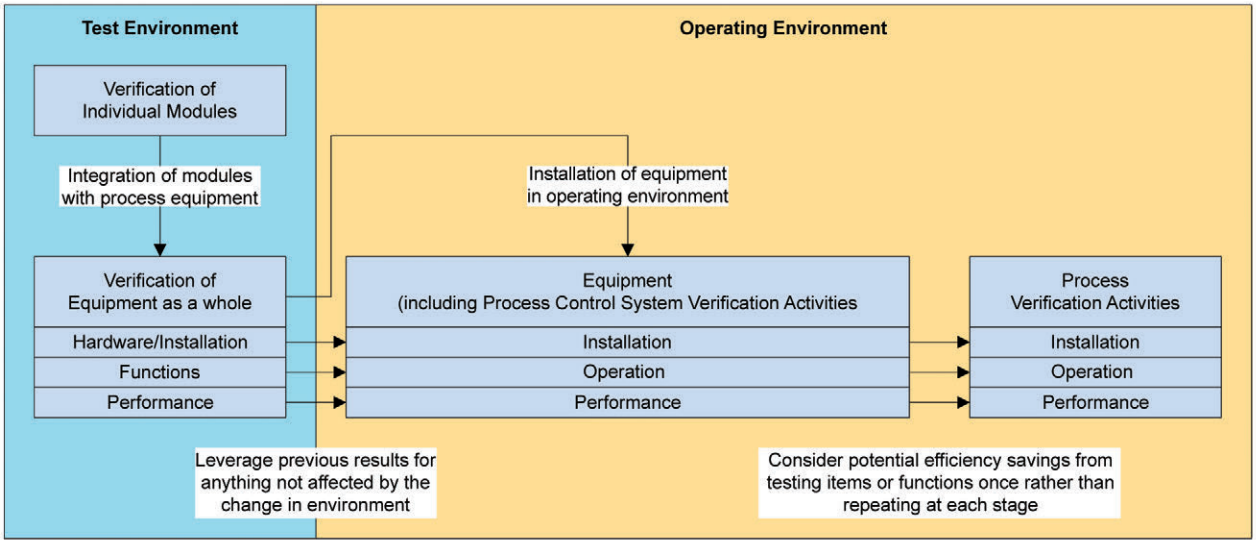
Table 21.1: Risk Scenarios for Packaged Systems

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Higher degree of expertise and knowledge of the process equipment (and in some cases, of the process) may reside with the supplier rather than the regulated organization.	Much more of the process expertise may reside with the equipment supplier who may be unfamiliar with working in a GxP environment.	Supplier typically should have more involvement in equipment and process verification activities. Appropriate split of verification activities between supplier and regulated organization should be considered at an early stage of test planning.
Where the control system and equipment are part of a standard product, the "type testing" of the product will be within the product development life cycle rather than the application life cycle.	Level of testing or control of configurations and test environment within product development life cycle may not be appropriate to the needs of the application.	Test strategy should reflect findings of the supplier assessment process; if necessary adding or repeating critical tests within the application life cycle.
Where the control system and equipment are part of a standard product, the supplier will have designed and tested them to a generic requirement.	Required process capability for the application does not map onto this generic capability; therefore, packaged system may need to be modified to meet specific requirements.	Test strategy should be scalable and verify existing product functionality, while thoroughly testing new/customized software, and its integration. Test strategy should ensure that the regulated organization's specific requirements have been tested, either by the supplier, or the regulated organization's own testing.
It can be difficult to separate operation of the process control system from the physical process equipment.	Inadequate testing of scenarios which are difficult to create on the physical equipment.	Test strategy should consider to what extent it is appropriate for software to be tested as part of equipment and to what extent module level testing with simulation or forcing of inputs is needed. Test strategy may need review by equipment, process and control system subject matter experts to ensure that all requirements have been adequately verified.
Packaged control system may be provided by a single supplier to the regulated organization which sub-contacts out responsibility for control system hardware and software.	Inadequate testing of software or communications failure scenarios because the main supplier is insufficiently familiar with the control system detail or the sub contractor is remote from and misinterprets user requirements.	Test strategy should reflect findings of the supplier assessment process (including, where appropriate, the supplier's assessment of their own sub-suppliers and/or regulated organization assessment of those sub-suppliers); if necessary, adding or repeating critical tests within the application life cycle.

21.3 Typical Test Types and Phasing for Packaged Systems

Figure 21.3 illustrates a typical flow of verification activities for a system where the controls are packaged with the process equipment.

Figure 21.3: Verification Activities for Packaged System



Depending on whether the packaged control system is part of a standard product or a “one off” application, more or less of the activities shown in the test environment may form part of a product development life cycle within the supplier’s quality management system.

Installation verification should be performed in both environments because a packaged control system is often qualified in an environment other than the operational environment. This is to verify that the operational environment is consistent with the environment in which qualification tests took place and that, e.g., no links remain to objects still residing in the test environment.

21.3.1 Pre-Delivery Test Types and Phasing for a Standard Product

Table 21.2: Pre-Deliver Test Types and Phasing for a Standard Product

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Product Development Life Cycle Module Tests	Module testing is often performed using some form of simulation or forcing of inputs and generally covers: • Module data handling • Interfaces to other modules • Operator interfaces • Module functionality Failure paths and response to fault conditions should be included within the tests.	Regulated organization should seek to leverage this testing during the application life cycle. Existing modules should be re-used “as is” where possible. Previously completed module tests can then be referenced rather than repeated. Risk-based test scope such that only modules of specific complexity/novelty/ risk priority undergo formal module test.

Table 21.2: Pre-Deliver Test Types and Phasing for a Standard Product (continued)

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Product Development Life Cycle Module Integration Tests	Module integration testing is often performed with control system already integrated with process equipment and generally covers: • Correct operation of interfaces between modules. • Correct interaction of software and process equipment. Failure paths and response to fault conditions should be included within the tests.	Regulated organization should seek to leverage this testing during the application life cycle. Depending on complexity/novelty/ risk priority associated with interfaces between modules, it may be possible to move directly from module testing to system integration testing.
Product Development Life Cycle "Type Testing"	Functional testing of combined process control system and process equipment against product requirements specification. Total scope as per integrated test for a generic process control system (see Appendix E6), but with control system and process equipment fully integrated and tests covering operation of equipment as well as control system hardware/software.	Regulated organization should assess whether their own specific requirements for the package are the same as the general product requirements. Where this is the case, they can leverage the supplier's standard type testing.
Application Life Cycle Supplier's System Integration Testing/ Factory Acceptance Testing	The coverage should include sufficient tests to confirm that this instance of the standard product is operating correctly: • Checking that equipment, control system hardware and software have been correctly installed on this instance of the standard product. • Demonstration that all sensors read correctly. • Demonstration that all actuators function correctly. • Demonstration that the standard product has been correctly configured for the application. • Demonstration that the overall machine functions as per the requirements for the specific application.	Tests from the product type testing should not be repeated if the transfer to a new instance of the product has no adverse impact on the function being tested. Where standard product is purely parameterized for the application, there is often only one integrated test phase performed rather than a separate supplier's test and factory acceptance test.

21.3.2 Pre-Delivery Test Types and Phasing for a "One Off" Application

Table 21.3: Pre-Delivery Test Types and Phasing for a "One Off" Application

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Application Life Cycle Module tests	Module testing is often performed using some form of simulation or forcing of inputs and generally covers: • Module data handling • Interfaces to other modules • Operator interfaces • Module functionality Failure paths and response to fault conditions should be included within the tests.	Assuming that existing modules have been managed according to a robust software development method then they should be re-used where possible. Previously completed module tests can then be referenced rather than repeated. Risk-based test scope such that only modules of specific complexity/novelty/ risk priority undergo formal module test.
Application Life Cycle Module Integration Tests	In a one-off, high risk priority, custom coded application, extensive module integration tests will be required to ensure a robust solution with predictable operation and performance. For a lower risk priority application where many of the interfaces are provided by the control system standard software package(s), less module integration testing will be needed. Module integration testing is often performed with control system already integrated with process equipment and generally covers: • Correct operation of interfaces between modules. • Correct interaction of software and process equipment. Failure paths and response to fault conditions should be included within the tests.	For simple, small scale, low risk priority software systems built up from proven software modules with tried and tested interfaces, it may be possible to move directly from module testing to system integration testing.
Application Life Cycle Supplier's System Integration Testing	Functional testing of combined process control system and process equipment against application requirements specification. Total scope as per integrated test for a generic process control system (see Appendix E6), but with control system and process equipment fully integrated and tests covering operation of equipment as well as control system hardware/software.	In order to leverage supplier's integration testing to minimize FAT testing, these tests should be documented to good documentation practices using documented processes for reporting and managing test exceptions, deviations, and handling attachments.

Table 21.3: Pre-Delivery Test Types and Phasing for a "One Off" Application (continued)

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Application Life Cycle Factory Acceptance Testing (FAT)	Typical test coverage depends on contractual agreement between supplier and regulated organization and on the results of the regulated organization's supplier assessment activities (including review of the supplier's internal test results). Coverage is often based on sampling from the same test scripts as used for the supplier's integration testing.	The required coverage should reflect the relative risk priority associated with the system element under test and the regulated organization's review of the supplier's internal test results.

21.3.3 Post Delivery Tests and Operational/Retirement Phases – Common to Both Types

Table 21.4: Post Delivery Tests and Operational/Retirement Phases - Common to Both Types

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Project Site Acceptance Testing (SAT)	Scope as per other process control systems (see Appendix E6) except that, because FAT was performed with control system and process equipment fully integrated, there are fewer interfaces affected by the move to the site environment, and therefore a smaller scope of retest.	Successful FAT tests should not be repeated if the transition from factory to site has no adverse impact on the function being tested and the software is verified to be the same version as that tested during FAT. If, however, regulated organization specific security/admin functionality or networked communications are added at this stage, successful configuration of those elements may need to be tested. Connection to different utilities on site may impact system performance and require further parameter optimization.
Operation (Performance Monitoring Activities)	Scope as per other process control systems (see Appendix E6) but also likely to include overall utilization/reliability of the equipment, e.g., to support Overall Equipment Effectiveness (OEE) measures to improve productivity.	Use performance monitoring tools provided as standard by the supplier.
Operation (Adjustments within the Previously Tested Design Space)	No additional testing required except where needed to confirm adjustment has been correctly made.	Ensure operational change management system allows for reduced burden of verification in this case.

Table 21.4: Post Delivery Tests and Operational/Retirement Phases – Common to Both Types (continued)

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Operation (Following Disaster Recovery)	Scope as per other process control systems (see Appendix E6), but regression testing is also likely to include testing of process equipment functionality as well as control system.	Service level agreement should address which elements of post-disaster recovery testing are to be done by supplier – for both process equipment and control system. Where such testing is needed, re-use of original test scripts may reduce the work required.
Operation (Changes to Design Space)	Scope as per other process control systems (see Appendix E6), but both testing of new functionality and regression testing are likely to include testing of process equipment functionality as well as control system.	System Modifications and Upgrades These will be implemented under change control which should include a risk assessment of the impact of the change on the process control system and the process equipment. The results of the risk assessment should identify the required regression tests.
Retirement (Data Retention/ Migration)	Test coverage should focus on the archiving and retrieval of all GxP data that should be retained following retirement of the system. • Confirm availability of all required data within final format. • Confirm correct migration (where appropriate) of data into final format.	There may be less option to retain the control system in order to retrieve archived data because the control system is embedded with the process equipment. It is more likely that the original requirement will have included an export route to make the data accessible outside of the system (it is possible that this is done via a standard viewing application provided by the supplier in order to read their proprietary format history files – in which case only the viewing application should be retained in order to read the data in its original form and there is no requirement for data migration and its associated test burden).

22 Appendix E8 – Testing Systems Applying Process Analytical Technology

22.1 Introduction

Process Analytical Technology (PAT) is defined (ICH Q8 [3]) as a system for designing, analyzing, and controlling manufacturing through timely measurements (i.e., during processing) of critical quality and performance attributes of raw and in-process materials and processes with the goal of ensuring final product quality.

PAT allows enhanced understanding and control of the manufacturing process. It is regarded as an enabler for Quality by Design (QbD), allowing quality to be built into pharmaceutical products. This allows benefits such as:

- Support for continuous quality assurance and real time release testing
- Reduction of rejects, scrap, and reprocessing
- Optimization of manufacturing output (yield) while remaining confident that quality is being maintained

Further information on the use of PAT is (or will be) provided in:

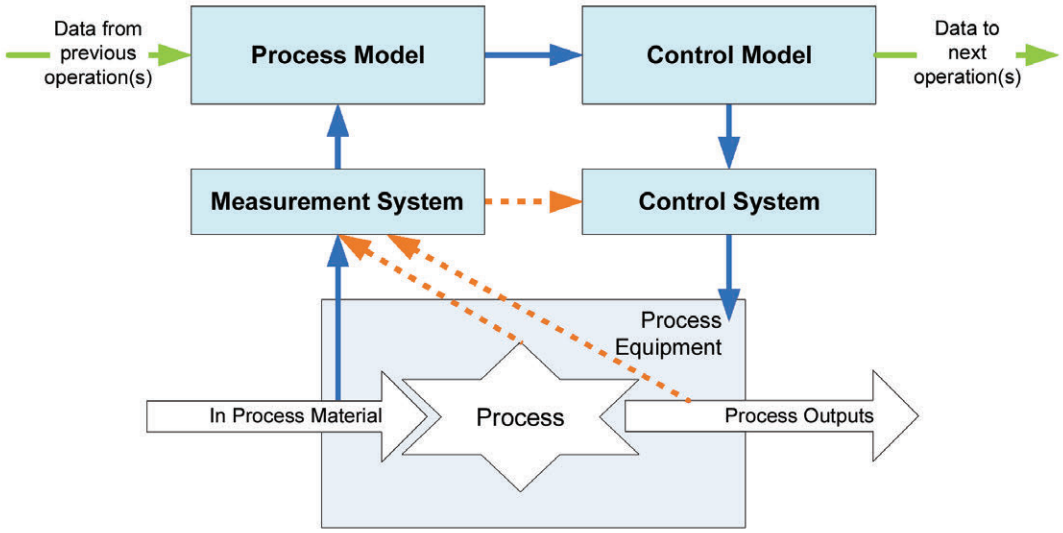
- ICH Q8, ICH Q9, ICH Q10 [3, 4, and 5]
- EMEA Reflection Paper: Chemical, Pharmaceutical, and Biological Information to be included in Dossiers when Process Analytical Technology (PAT) is Employed [28]
- FDA Guidance for Industry: PAT – A Framework for Innovative Pharmaceutical Development, Manufacturing, and Quality Assurance [29]
- ASTM E2629-11 Standard Guide for Verification of Process Analytical Technology (PAT) Enabled Control Systems [30]
- The ISPE PQLI® Guide Series [31]

This appendix focuses on giving an overview of what a PAT system is and how the additional risks associated with a PAT system can be mitigated by appropriate testing.

22.1.1 What Is a PAT System?

Figure 22.1 illustrates a general model for a control system.

Figure 22.1: Generalized Control System Diagram



In Figure 22.1, the light solid arrows represent data flow to and from the adjoining operations. The dark solid arrows represent the information flow (e.g., measured values, calculated responses) that comprises the control in a PAT system. The dashed arrows are supplementary information flows that may or may not be present, depending on the complexity of the control system.

This system could be end-point detection in a powder blending application:

- Measurement equipment = Near Infra Red (NIR) spectroscopy
- Process model (built into control system) = the spectra generated are analyzed against a knowledge base of historical spectra
- Control model (built into control system) = an algorithm is applied, based on the process model, to predict the optimum end-point of the blending cycle
- Control system = applies the endpoint of the blending cycle to ensure the required level of homogeneity for the next operation (tableting)

PAT takes into account multi-factorial relationships among materials, manufacturing process, environmental variables, and their effects on quality in order to enhance understanding and control the manufacturing process. It focuses on understanding and controlling those attributes and parameters that have an impact on critical quality attributes of the product. This improved understanding allows the process to be optimized while still maintaining quality product; therefore, the elements that make up a PAT system may be much more complex:

- Measurement equipment may include complex or innovative process analyzers in order to measure a material attribute directly or measure a process parameter that is known to have a strong correlation to the desired attribute.
- Process models may combine inputs from more than one measurement system to calculate the value of a material attribute based on a number of parameters; therefore, the model may involve multiple variables, which may or may not operate independently of each other. Multivariate tools are needed for the design of such models as well as for the acquisition and analysis of the real-time data. The model also may need to handle differing input data types such as time series and spectral data.

- Control models may be predictive based on a theoretical process model with adjustments then automatically made to bring them in line with actual measurements.

- Control systems may need to respond to commands from a separate control model rather than having all control algorithms built in.

The PAT toolkit needed to implement and optimize such a system includes:

- Multivariate tools for design, data acquisition and analysis

- Process analyzers

- Process control tools (which may also be multivariate)

- Continuous improvement and knowledge management tools

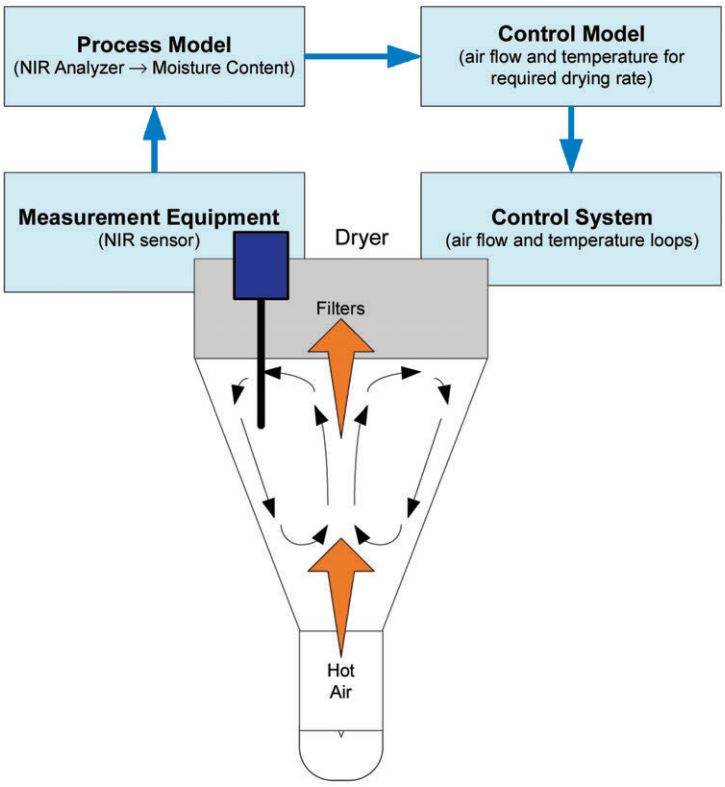
This increased process and product understanding allows the definition of “design space” within which the process should operate to guarantee quality product. This is defined in ICH Q8 [3] as: “The multidimensional combination and interaction of input variables (e.g., material attributes) and process parameters that have been demonstrated to provide assurance of quality. Working within the design space is not considered as a change. Movement out of the design space is considered to be a change and would normally initiate regulatory post approval change process. Design space is proposed by the applicant and is subject to regulatory assessment and approval”.

The testing of PAT applications may be supported by the use of computerized test tools as an alternative to paper based testing. Where multiple experiments are used to develop the PAT system, test automation tools may be used to efficiently conduct multiple experiments, i.e., repetitively enter input data to multiple runs of an experiment. There is a risk associated with allowing automated tests to execute where this will also exercise the physical process (even a pilot stage) and this risk is best mitigated by initial manual testing.

22.1.2 Examples of PAT Systems

A simple (univariate) PAT system for controlling the moisture content in a fluid bed dryer is shown in Figure 22.2. In this case, a NIR analyzer is used to measure the moisture content of the in-process material.

Figure 22.2: Simple PAT System for Controlling Moisture Content in a Fluid Bed Dryer (Batch Operation)

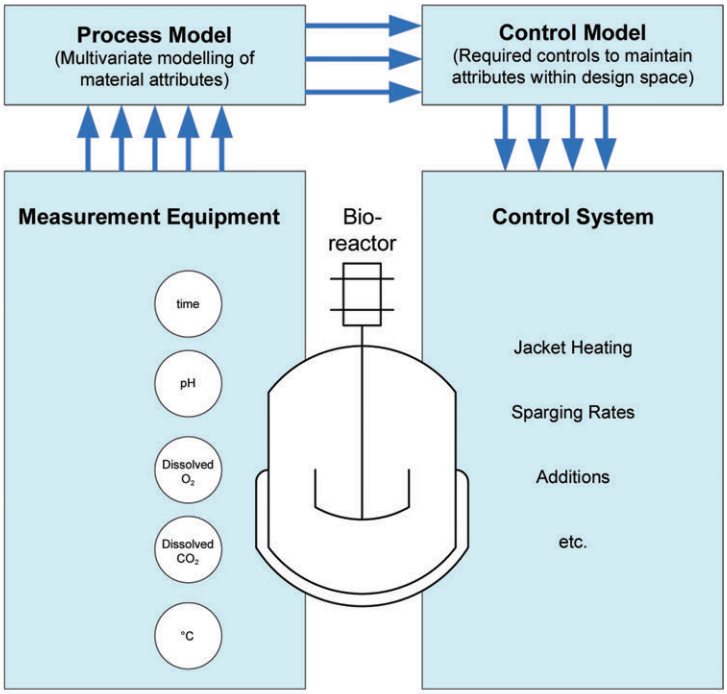


The implementation of this PAT system may be built up in stages, each of which improves the understanding and control of the process:

- Initial experiments with only the NIR sensor and data collection to allow a process model to be built up of how the NIR signal relates to moisture content measured by laboratory analysis
- Testing of the process model output against actual moisture content
- Creation of the control model based on how the controllable parameters (air flow and temperature) affect the measured drying rate
- Closed loop control to allow the desired drying curve to be followed

A multivariate PAT system for controlling a bioreactor is shown in Figure 22.3. In this case, there are conventional sensors, but a multivariate process model is used to derive the attributes of the in-process material.

Figure 22.3: Multivariate PAT System for Controlling a Bioreactor



The implementation of this PAT system may be built up in stages, each of which improves the understanding and control of the process:

- Initial experiments using formal statistical design of experiments to determine how the measurable parameters relate to the critical quality attributes (e.g., residual amounts of undesirable proteins in the final substance) and therefore develop the process model
- Creation of the control model (may be possible to define independent controls for some parameters while others require multivariate control)
- Closed loop control

22.2 What is special about these systems? – Risk Scenarios

A PAT system may differ from a conventional control system through the increased complexity and novelty of any or all of the measurement equipment, process model, control model, or control system. As PAT systems may support applications involving real time release testing, the risk assessment should take into account the complexity of these systems to establish controls to assure product quality at all times. Appropriate test strategies can help mitigate these risks.

Further guidance may be found in ASTM E2476-09 Standard Guide for Risk Assessment and Risk Control as it Impacts the Design, Development, and Operation of PAT Processes for Pharmaceutical Manufacture [32].

Table 22.1: Risk Scenarios for PAT Systems

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Requirement for modeling of process in order to derive values representative of a specific parameter.	The combination of measurement method plus process model does not produce results that are representative of the required attribute or parameter due to poor understanding of the underlying science.	Statistical verification of process model output against primary measurements across the full range of inputs. Continued verification of process model against independently measured/ sampled process values.
Presence of complex measurement equipment and/or process model means that PAT systems may be particularly prone to problems related to: • Accuracy • Deadbands or hysteresis • Measurement lag • Repeatability and reproducibility • Stability over time	The combination of measurement method plus process model does not produce results that are representative of the required attribute or parameter due to difficulty of making appropriate measurements.	Statistical verification of process model output against primary measurements including checks of accuracy, deadbands or hysteresis, measurement lag, repeatability and reproducibility, stability over time.
Presence of complex measurement equipment and/or process model means that PAT systems can be difficult to calibrate, particularly where multivariate models do not measure a parameter that can be traced to a formal independent standard.	Measurement equipment incorrectly calibrated.	Calibration frequencies and procedures, including appropriate conditions for calibration. Methods for detecting calibration drift including requirements for storage of the calibration data to allow detection or analysis of such drift and test methods for use as part of ongoing monitoring.
Presence of complex measurement equipment based on sampled values.	Sampled measurements are not representative of the target material.	Verification of any sampled measurements: • Representativeness of target material • Appropriate start points/end points/ timebases for analysis
Presence of complex measurement equipment.	Analytical instrument setup incorrect.	Confirm control of instrument is correct by testing or verifying: • Set-up routines • Start-up routines • cleaning routines • shutdown routines • alignment of data from different sources Verification of methods for detecting, and acting upon, analyzer failure.

Table 22.1: Risk Scenarios for PAT Systems (continued)

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
Process Analytical Technology (PAT) may be back-fitted to an existing system.	Addition of new analytical elements causes disturbance to existing process.	Regression testing should seek to confirm that product quality has been maintained.
Sensitivity of the process to system malfunction or poor performance can be difficult to predict in a multivariate system or one with feed forward elements.	System malfunction or poor performance goes undetected.	Assess sensitivity of process to variations in performance of individual PAT system elements: • Malfunction • Poor quality data Disaster recovery procedures need to include appropriate levels of adjustment/ re-verification following any intervention.
Data may come from multiple sources.	Misalignment of data from different sources or between different elements of the system.	Verify correct communication of data values between system elements. Verify synchronization of timing between different elements. Verify timely presentation of required data at each location. Verify data transformation processes. Load/challenge testing to verify robustness of communications.
Sensitivity of the process to changes in environmental conditions can be difficult to predict in a multivariate system or one with feed forward elements.	Change in environmental conditions moves system outside its ability to control.	Assess sensitivity of process to fluctuations in inputs. Assess model performance across full range of environmental conditions.
PAT systems can carry additional risks related to prediction failure or mode instability that are not normally present in conventional control systems.	Inability to control in an area within which the process is expected to operate.	Demonstrate control across full operating ranges and in response to both setpoint changes and process fluctuations: • Stability • Damping/settling times/overshoots • Steady state errors

Table 22.1: Risk Scenarios for PAT Systems (continued)

Aspects Specific to this Type of System	Risk Scenario	Impact on Test Strategy
System may contain adaptive or self tuning elements.	Adaptive or self tuning elements within models take control outside the design space.	Demonstrate that appropriate constraints are in force to ensure that adaptive algorithms explore only possibilities that are within the design space. (Note: where it is only the control model that is adaptive, it is relatively easy to conceive of how constraints and alarms may be applied and verified, based on the process model output, to ensure that the (multivariate) process parameters remain within the design space. Where the process model is itself adaptive, for example, in "neural network" style image processing systems, this could prove much more difficult to demonstrate. There is, at the time of writing, little literature within the industry offering advice in this area, but appropriate mitigation strategies may include things like independent measurement of selected parameters to create boundary conditions and the regular use of "calibration pieces" to check process model outputs).

22.3 Typical Test Types and Phasing for Systems Applying Process Analytical Technology

The development of PAT systems differ from that of a generic "computerized system" in that they are often built up in stages, each of which improves the understanding and control of the process:

- Initial experiments (using formal statistical design of experiments in the case of multivariate systems) to determine how the measurable parameters relate to the critical quality attributes and therefore develop the process model
- Testing of the process model output against actual process samples or analyses
- Creation of the control model (may be possible to define independent controls for some parameters while others require multivariate control)
- Closed loop control

Usually some or all of these stages take place "back-fitted" onto an existing process; therefore, involving changes to an operational system even during initial development of the PAT element.

During operation of a PAT system, changes may be part of this ongoing development process or may be deliberate adjustments of control within the design space in order to improve process yield or throughput. The risk to patient safety is likely to be very different between these different styles of activity and the required test strategy should be based on an appropriate risk assessment.

When retiring a PAT system, there is likely to be a large quantity of data to be considered. This data may come from a number of sources and the data retention or migration test strategy will need to cover both the availability and the alignment/synchronization of this data.

Table 22.2 shows typical coverage in each test phase and identifies opportunities for (risk-based) leveraging of supplier's testing/testing in earlier phases/system testing and process testing/operation and retirement.

Table 22.2: Test Types and Phasing for PAT Systems

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Development (Installation of Measurement Equipment)	• Confirm installation as per manufacturer's instructions. • Record serial numbers, versions, etc. • Perform regression testing to confirm that product quality has been maintained if equipment being back-fitted to an existing process.	Installation verification may be done by supplier. Where regression testing is needed, re-use of original test scripts may reduce the work required.
Development (Operation of Measurement equipment)	Confirm control of instrument is correct by testing or verifying: • Set-up routines • Start-up routines • Cleaning routines • Shutdown routines • Alignment of data from different sources Verification of methods for detecting and acting upon analyzer failure.	Operational testing following installation may well be done by supplier.
Development (Creation of Process Model)	• Verify correct communication of data values between system elements (and include challenge/ load testing to confirm robustness of communications). • Verify synchronization of timing between different elements. • Verify timely presentation of required data at each location. • Verify data transformation processes. • Statistical verification of process model output against primary measurements across the full range of input. • Checks of accuracy, deadbands or hysteresis, measurement lag, repeatability and reproducibility, stability over time. • Testing of calibration methods and of methods for detecting calibration drift. • Testing of any sampled measurements for representativeness and for appropriate start/end points and timebase of the analysis	If the supplier provides measurement equipment together with appropriate process model, it may be possible for the regulated organization to leverage the testing done by the supplier as part of their process model development.

Table 22.2: Test Types and Phasing for PAT Systems (continued)

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Development (Creation of Control Model)	• Assess sensitivity of process to variations in performance (malfunction, poor quality data) of individual PAT system elements. • Test disaster recovery procedures. • Assess sensitivity of process to fluctuations in inputs or changes in environmental conditions.	Where a PAT system is packaged with the process equipment, it may be possible to leverage the testing done by the supplier as part of their control model development.
Development (Closed Loop Control)	• Demonstrate control across full operating ranges and in response to both setpoint changes and process fluctuations (including stability, damping/settling times/overshoots, steady state errors). • Demonstrate that appropriate constraints are in force to ensure that adaptive algorithms explore only possibilities that are within the design space.	Where a PAT system is packaged with the process equipment, it may be possible for the regulated organization to leverage the testing done by the supplier as part of their closed loop control development.
Operation (Performance Monitoring Activities)	• Continued verification of process model against independently measured/sampled process values. • Continued monitoring of calibration. • Continued monitoring of communication robustness.	The supplier may have generic metrology data on the measurement function against time across the full operating range. Use this data to determine the optimum intervals for calibration and calibration checks.
Operation (Adjustments within the Previously Tested Design Space)	• No additional testing required except where needed to confirm adjustment has been correctly made.	Ensure operational change management system allows for reduced burden of verification in this case.
Operation (Following Disaster Recovery)	Regression testing should be based on a risk assessment, but is likely to include the following: • Confirm correct installation of any replaced items. • Record serial numbers, versions, etc. • Confirm correct adjustment/ configuration/calibration of any replaced items. • Confirm continued availability/ alignment of data from before disaster recovery. • Confirm availability/alignment of newly collected data. • Perform appropriate testing to confirm that product quality has been maintained.	Service level agreement should address which elements of post-disaster recovery testing are to be done by supplier. Where regression testing is needed, re-use of original test scripts may reduce the work required.

Table 22.2: Test Types and Phasing for PAT Systems (continued)

Life Cycle Phase (Test Phase)	Typical Coverage	Opportunities for Risk-based Leveraging or Efficiencies
Operation (Changes to Design Space)	Testing should be based on a risk assessment but is likely to include many of the same activities as described under "DEVELOPMENT" testing.	Leverage testing previously completed during development testing, using a risk-based approach to focus effort on critical areas impacted by the change.
Retirement (Data Retention/ Migration)	• Confirm availability of all required data (e.g., from all sources) within final format. • Confirm alignment/synchronization of all required data within final format.	If the supplier provides a standard data retention or migration solution, it may be possible for the regulated organization to leverage the testing done by the supplier during the development of this. Additional information on testing related to data management can be found in Appendix T9.

23 Appendix 20 – References

1. I*SPE GAMP® 5: A Risk-Based Approach to Compliant GxP Computerized Systems*, International Society for Pharmaceutical Engineering (ISPE), Fifth Edition, February 2008, www.ispe.org.

2. ISPE Website, www.ispe.org.

3. International Conference on Harmonisation of Technical Requirements for Registration of Pharmaceuticals for Human Use, ICH Harmonised Tripartite Guideline, *Pharmaceutical Development – Q8(R2)*, August 2009, www.ich.org.

4. International Conference on Harmonisation of Technical Requirements for Registration of Pharmaceuticals for Human Use, ICH Harmonized Tripartite Guideline, *Quality Risk Management – Q9*, Step 4, 9 November 2005, www.ich.org.

5. International Conference on Harmonization of Technical Requirements for Registration of Pharmaceuticals for Human Use, ICH Harmonized Tripartite Guideline, *Pharmaceutical Quality System – Q10*, Step 4, 4 June 2008, www.ich.org.

6. ASTM Standard E2500-07, "Standard Guide for Specification, Design, and Verification of Pharmaceutical and Biopharmaceutical Manufacturing Systems and Equipment," ASTM International, West Conshohocken, PA, www.astm.org.

7. EudraLex Volume 4 – Guidelines for Good Manufacturing Practices for Medicinal Products for Human and Veterinary Use, Chapter 4: Documentation, Revisions, ec.europa.eu.

8. EudraLex Volume 4 – Guidelines for Good Manufacturing Practices for Medicinal Products for Human and Veterinary Use, Annex 11 – Computerized Systems, ec.europa.eu.

9. BS ISO/IEC 14756:1999, "Information Technology -- Measurement and Rating of Performance of Computer-Based Software Systems," International Organization for Standardization (ISO), www.iso.org.

10. *ISPE GAMP® Good Practice Guide: IT Infrastructure Control and Compliance*, International Society for Pharmaceutical Engineering (ISPE), First Edition, August 2005, www.ispe.org.

11. *ISPE GAMP® Good Practice Guide: A Risk-Based Approach to Operation of GxP Computerized Systems*, International Society for Pharmaceutical Engineering (ISPE), First Edition, January 2010, www.ispe.org.

12. 21 CFR Part 820 – Quality System Regulation, US Code of Federal Regulations, U.S. Food and Drug Administration (FDA), www.fda.gov.

13. MIL STD 1916, "DoD Preferred Methods for Acceptance of Product" (available online at http://assist.daps.dla.mil), or ANSI/ASQ Z1.4, "Sampling Procedures and Tables for Inspection by Attributes" (available online at www.asq.org).

14. ISO 24153:2009, "Random Sampling and Randomization Procedures," International Organization for Standardization (ISO), www.iso.org.

15. ANSI/ASQ Z1.4-2008: Sampling Procedures and Tables for Inspection by Attributes, American National Standards Institute (ANSI), www.ansi.org / American Society for Quality (ASQ), http://asq.org.

16. *ISPE GAMP® Good Practice Guide: Electronic Data Archiving*, International Society for Pharmaceutical Engineering (ISPE), First Edition, July 2007, www.ispe.org.

17. 21 CFR Part 11 – Electronic Records; Electronic Signatures, US Code of Federal Regulations, U.S. Food and Drug Administration (FDA), www.fda.gov.

18. The NIST Definition of Cloud Computing – NIST Special Publication (SP) 800-145, National Institute of Standards and Technology (NIST), www.nist.gov.

19. Guidelines on Security and Privacy in Public Cloud Computing – NIST Special Publication (SP) 800-144, National Institute of Standards and Technology (NIST), www.nist.gov.

20. *ISPE GAMP® Good Practice Guide: A Risk-Based Approach to GxP Compliant Laboratory Computerized Systems*, International Society for Pharmaceutical Engineering (ISPE), Second Edition, October 2012, www.ispe.org.

21. *ISPE GAMP® Good Practice Guide: A Risk-Based Approach to Calibration Management*, International Society for Pharmaceutical Engineering (ISPE), Second Edition, November 2010, www.ispe.org.

22. ANSI/ASTM E617-97 (2008) "Standard Specification for Laboratory Weights and Precision Mass Standards," ASTM International, West Conshohocken, PA, www.astm.org.

23. Hjulmand-Lassen, Ulrik, "Virtualization – Compliance and Control," *Pharmaceutical Engineering*, July/August 2010, Vol. 30, No. 4, pp. 34-44, www.pharmaceuticalengineering.org.

24. Stokes, David, "Controlling Service Oriented Architecture in Support of Operational Improvement," *Pharmaceutical Engineering*, July/August 2011, Vol. 31, No. 4, pp. 56-64, www.pharmaceuticalengineering.org.

25. *ISPE GAMP® Good Practice Guide: A Risk-Based Approach to GxP Process Control Systems*, International Society for Pharmaceutical Engineering (ISPE), Second Edition, February 2011, www.ispe.org.

26. ANSI/ISA-95 (S95), Enterprise-Control System Integration, American National Standards Institute (ANSI), www.ansi.org / International Society of Automation (ISA), www.isa.org.

27. ANSI/ISA-88 (S88), Batch Control, American National Standards Institute (ANSI), www.ansi.org / International Society of Automation (ISA), www.isa.org.

28. EMEA Reflection Paper: Chemical, Pharmaceutical, and Biological Information to be included in Dossiers when Process Analytical Technology (PAT) is Employed, www.ema.europa.eu.

29. FDA Guidance for Industry: PAT – A Framework for Innovative Pharmaceutical Development, Manufacturing, and Quality Assurance (September 2004), www.fda.gov.

30. ASTM E2629-11 "Standard Guide for Verification of Process Analytical Technology (PAT) Enabled Control Systems," ASTM International, West Conshohocken, PA, www.astm.org.

31. *ISPE Guide Series: Product Quality Lifecycle Implementation (PQLI®), from Concept to Continual Improvement,* International Society for Pharmaceutical Engineering (ISPE), 2011, www.ispe.org.

32. ASTM E2476-09 "Standard Guide for Risk Assessment and Risk Control as it Impacts the Design, Development, and Operation of PAT Processes for Pharmaceutical Manufacture," ASTM International, West Conshohocken, PA, www.astm.org.

24 Appendix 21 – Glossary

24.1 Acronyms and Abbreviations

API Active Pharmaceutical Ingredient

BCP Business Continuity Plan

COTS Commercial Off-the-Shelf

CPP Critical Process Parameter

CPU Central Processing Unit

CQA Critical Quality Attribute

CRM Customer Relationship Management

CSV Computerized Systems Validation

DCS Distributed Control System

DMS Document Management System

DRP Disaster Recovery Planning

DSDM Dynamic Systems Development Method

ERP Enterprise Resource Planning

ESB Enterprise Service Bus

FAT Factory Acceptance Test

GxP comprises:

- **GCP** Good Clinical Practice
- **GDP** Good Distribution Practice
- **GEP** Good Engineering Practice
- **GLP** Good Laboratory Practice
- **GMP** Good Manufacturing Practice
- **GQP** Good Quality Practice

HMI Human Machine Interface

HPLC High Performance Liquid Chromatograph

I/O	Input/Output
IaaS	Infrastructure as a Service
LIMS	Laboratory Information Management System
MES	Manufacturing Execution System
MRPII	Manufacturing Resource Planning
NIR	Near Infrared Technology
NIST	National Institute of Standards and Technology
OEE	Overall Equipment Effectiveness
OSS	Open Source Software
PaaS	Platform as a Service
PAT	Process Analytical Technology
PCE	Phase Containment Effectiveness
PDA	Personal Digital Assistant
PDF	Portable Document Format
PID	Proportional Integral Derivative
QA	Quality Assurance
QbD	Quality by Design
QC	Quality Control
QMS	Quality Management System
QRM	Quality Risk Management
R&D	Research and Development
RAD	Rapid Application Development
RAM	Random Access Memory
ROI	Return On Investment
RUP	Rational Unified Process
SaaS	Software as a Service
SAT	Site Acceptance Test

SCADA	Supervisory Control and Data Acquisition
SLA	Service Level Agreement
SME	Subject Matter Expert
SOA	Service Oriented Architecture
SOP	Standard Operating Procedure
TDCE	Total Defect Containment Effectiveness
URS	User Requirement Specification
XP	Extreme Programming

24.2 Definitions

Acceptance Criteria (IEEE)

The criteria that a system or component must satisfy in order to be accepted by a user, customer or other authorized entity

Acceptance Test (IEEE)

Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.

See also Factory Acceptance Test (FAT), Site Acceptance Test (SAT).

Black Box Testing (IEEE)

See Functional Testing

Boundary Condition Testing (from IEEE definition of Boundary Condition)

Testing for correct operation when one or more variables are at a limiting value or a value at the edge of the domain of interest.

Calibration (ISO 10012)

The set of operations which establish, under specified conditions, the relationship between values indicated by a measuring instrument or measuring system, or values represented by a material measure or a reference material, and the corresponding values of a quantity realized by a reference standard.

Challenge Testing

Testing to check system behavior under abnormal conditions. Can include stress testing and deliberate challenges, for example to the security access system, the data formatting rules, the possible combinations of operator actions, etc.

Commissioning (ISPE)

A well planned, documented, and managed engineering approach to the start-up and turnover of facilities, systems, and equipment to the end user, that results in a safe and functional environment that meets established design requirements and stakeholder expectations.

Emulation (ISO 24765)

(1) A model that accepts the same inputs and produces the same outputs as a given system. (2) The process of developing or using a model. (3) The use of a data processing system to imitate another data processing system, so that the imitating system accepts the same data, executes the same programs and achieves the same results as the imitated system.

Environmental Testing (ISO 24765)

Testing that evaluates system or component performance up to the specified limits of environmental parameters (for example temperature or humidity).

Firmware (ISO 24765)

(1) The combination of hardware device and computer instructions and data that reside as read-only software on that device. (2) An ordered set of instructions and associated data stored in a way that is functionally independent of main storage, usually in a ROM.

Factory Acceptance Test (FAT) (IEEE)

An Acceptance Test in the Supplier's factory, usually involving the Customer. See also Acceptance Test. Contrast to Site Acceptance Test.

Functional Testing (IEEE)

(1) Testing that ignores the internal mechanism or structure of a system or component and focuses on the outputs generated in response to selected inputs and execution conditions. (2) Testing conducted to evaluate the compliance of a system or component with specified functional requirements and corresponding predicted results. Syn: black-box testing, input/output driven testing. Contrast with testing, structural.

Hardware (ISO24765)

(1) Physical equipment used to process, store, or transmit computer programs or data. (2) All or part of the physical components of an information system.

Hardware Testing

Testing carried out to verify correct operation of system hardware independent of any custom application software

Installation Qualification

Documented verification that a system is installed according to written and pre-approved specifications.

Instance

A single installation of a software application (plus associated databases, tools and utilities). Usually applied to configurable IT systems.

Integration (ISO24765)

The process of combining software components, hardware components or both into an overall system. Sometimes described as software integration and system integration respectively.

Integration Testing (ISO24765)

Testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them. Note: commonly used for both the integration of components and the integration of an entire system.

Load Testing

As stress testing but to evaluate a system or component up to or at the limits of its specified requirements.

Loop Testing

Testing in which control system inputs and outputs are exercised and their functionality verified.

Middleware

The hardware, computer instructions, and data which provide the infrastructure used by other system modules.

Module Testing (IEEE)

Testing of individual hardware or software components or groups of related components.

Negative Testing (BCS)

Testing aimed at showing that software does not work.

Operational Qualification (OQ)

The documented verification that the facilities, systems and equipment, as installed or modified, perform as intended throughout the anticipated operating ranges.

Performance Qualification (PQ)

The documented verification that the facilities, systems and equipment, as connected together, can perform effectively and reproducibly, based on the approved process method and product specification

Documented verification that a system is capable of performing or controlling the activities of the processes it is required to perform or control, according to written and pre-approved specifications, while operating in its specified operating environment.

Positive Testing

Testing aimed at showing that software does meet the defined requirements.

Quality (ICH Q9)

The degree to which a set of inherent properties of a product, system or process fulfills requirements (see ICH Q6a definition specifically for “quality” of drug substance and drug (medicinal) products).

Quality [Product] (ICH Q8)

The suitability of either a drug substance or drug product for its intended use. This term includes such attributes as the identity, strength, and purity (from ICH Q6A Specifications: Test Procedures and Acceptance Criteria for New Drug Substances and New Drug Products: Chemical Substances).

Quality by Design (PQLI®)

A systematic approach to development that begins with predefined objectives and emphasizes product and process understanding based on sound science and quality risk management.

Quality Management System (ISO)

Management system to direct and control an organization with regard to quality.

Quality Plan (ISO)

Document specifying which procedures and associated resources shall be applied by whom and when to a specific project, product, process or contract.

Quality Risk Management (ICH Q9)

A systematic process for the assessment, control, communication and review of risks to the quality of the drug (medicinal) product across the product lifecycle.

Quality System (ICH Q9)

The sum of all aspects of a system that implements quality policy and ensures that quality objectives are met.

Requirement (ISO)

Need or expectation that is stated, generally implied or obligatory.

Residual Risk (ISO 24765)

A risk that remains after risk responses have been implemented

Risk (ICH Q9)

The combination of the probability of occurrence of harm and the severity of that harm (ISO/IEC Guide 51). (May also be referred to as risk impact.)

Risk Assessment (ICH Q9)

A systematic process of organizing information to support a risk decision to be made within a risk management process. It consists of the identification of hazards and the analysis and evaluation of risks associated with exposure to those hazards.

Risk Control (ICH Q9)

Actions implementing risk management decisions (ISO Guide 73).

Risk Evaluation (ICH Q9)

The comparison of the estimated risk to given risk criteria using a quantitative or qualitative scale to determine the significance of the risk

Risk Identification (ICH Q9)

The systematic use of information to identify potential sources of harm (hazards) referring to the risk question or problem description.

Risk Management (ICH Q9)

The systematic application of quality management policies, procedures, and practices to the tasks of assessing, controlling, communicating and reviewing risk.

Risk Reduction (ICH Q9)

Actions taken to lessen the probability of occurrence of harm and the severity of that harm. [may also be referred to as risk mitigation]

Risk Review (ICH Q9)

Review or monitoring of output/results of the risk management process considering (if appropriate) new knowledge and experience about the risk.

Severity (ICH Q9)

A measure of the possible consequences of a hazard.

Simulation (IEEE)

A model that behaves or operates like a given system when provided a set of given inputs.

Site Acceptance Test (SAT) (IEEE)

An Acceptance Test at the Customer's site, usually involving the Customer. See also Acceptance Test, contrast to Factory Acceptance Test.

Software (ISO 24765)

(1) All of part of the programs, procedures, rules and associated documentation of an information processing system.
(2) Program or set of programs used to run a computer.

Software Life Cycle (NIST)

Period of time beginning when a software product is conceived and ending when the product is no longer available for use. The software life cycle is typically broken into phases denoting activities such as requirements, design, programming, testing, installation, and operation and maintenance.

Source Code (FDA) (1) (IEEE) (2)

(1) Computer instructions and data definitions expressed in a form suitable for input to an assembler, compiler or other translator. (2) The human readable version of the list of instructions (program) that cause a computer to perform a task.

Specification (IEEE)

A document that specifies, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a system or component, and often, the procedures for determining whether these provisions have been satisfied.

Subject Matter Expert (SME) (GAMP® 5)

Those individuals with specific expertise in a particular area or field. Subject Matter Experts should take the lead role in the verification of computerized systems. Subject Matter Expert responsibilities include planning and defining verification strategies, defining acceptance criteria, selection of appropriate test methods, execution of verification tests, and reviewing results.

Stress Testing (ISO 24765)

Testing conducted to evaluate a system or component at or beyond the limits of its specified requirements.

Structural Testing (IEEE)

1) Testing that takes into account the internal mechanism (structure) of a system or component. Types include branch testing, path testing, statement testing. (2) Testing to insure each program statement is made to execute during testing and that each program statement performs its intended function.

Contrast with functional testing. Syn: white-box testing, glass-box testing, logic driven testing.

Supplier (GAMP® 5)

An organization or individual internal or external to the user associated with the supply and/or support of products or services at any phase throughout a systems life cycle.

System Owner (GAMP® 5)

The person ultimately responsible for the availability, and support and maintenance, of a system and for the security of the data residing on that system.

System Testing (IEEE)

Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

Test (ISO 24765)

(1) An activity in which a system or component is executed under specific conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component. (2) A set of one or more test cases and procedures.

Test Case (ISO 24765)

(1) A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. (2) Documentation specifying inputs, predicted results and a set of execution conditions for a test item.

T**est Coverage** (ISO 24765)

(1) The degree to which a given test or set of tests addresses all specified requirements for a given system or component. (2) The extent to which the test cases test the requirements for the system or software product.

Test Environment (ISO 24765)

Hardware and software configuration necessary to conduct the test case

Test Incident Report (ISO 24765)

Document reporting on any event that occurs during the testing process which requires an investigation

Test Plan (IEEE)

A document describing the scope, approach, resources, and schedule of intended test activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning.

Test Procedure (IEEE)

Detailed instructions for the set-up, execution, and evaluation of results for a given test case.

Test Script (IEEE)

Documentation that specifies a sequence of actions for the execution of a test.

Test Specification

A document that describes the scope, management, use of procedures, sequencing, test environment, and prerequisites for a specific phase of testing.

Test Strategy

See Test Plan.

Unit Testing (ISO 24765)

(1) Testing of individual routines and modules by the developer or an independent tester. (2) Testing of individual programs or modules in order to ensure that there are no analysis or programming errors.

Usability Testing (ISO 24765)

A test to determine whether an implemented system fulfills its functional purpose as determined by its users.

User (GAMP® 5)

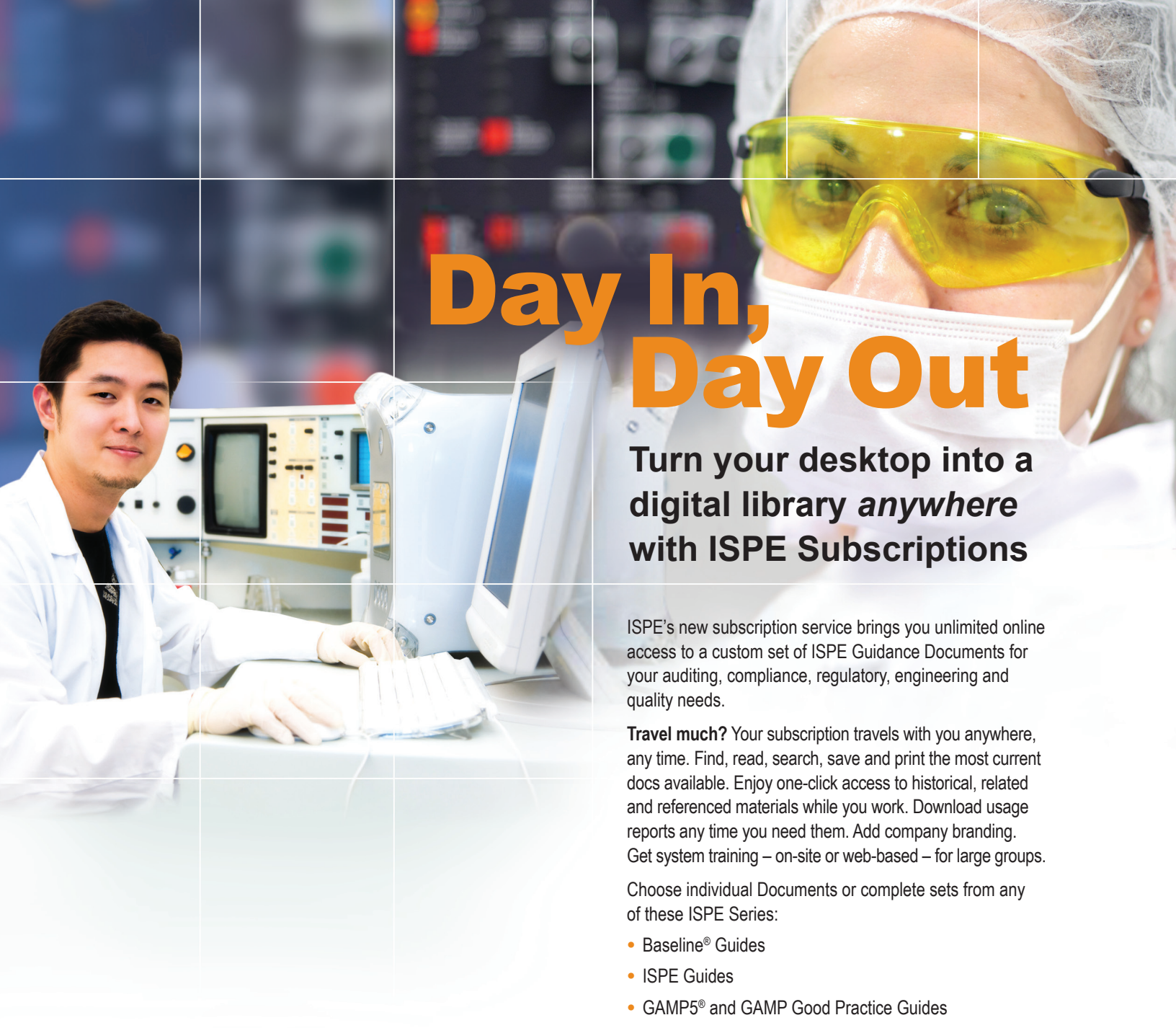
The pharmaceutical customer or user organization contracting a supplier to provide a product. In the context of this document it is, therefore, not intended to apply only to individuals who use the system, and is synonymous with customer.

Verification (ISO) (1) (ASTM) (2)

(1) Confirmation, through the provision of objective evidence, that specified requirements have been fulfilled. (2) A systematic approach to verify that manufacturing systems, acting singly or in combination, are fit for intended use, have been properly installed, and are operating correctly. This is an umbrella term that encompasses all types of approaches to assuring systems are fit for use such as qualification, commissioning and qualification, verification, system validation, or other.

White Box Testing (IEEE)

See Structural Testing.



Day In, Day Out

Turn your desktop into a digital library *anywhere* with ISPE Subscriptions

ISPE's new subscription service brings you unlimited online access to a custom set of ISPE Guidance Documents for your auditing, compliance, regulatory, engineering and quality needs.

Travel much? Your subscription travels with you anywhere, any time. Find, read, search, save and print the most current docs available. Enjoy one-click access to historical, related and referenced materials while you work. Download usage reports any time you need them. Add company branding. Get system training – on-site or web-based – for large groups.

Choose individual Documents or complete sets from any of these ISPE Series:

- Baseline® Guides
- ISPE Guides
- GAMP5® and GAMP Good Practice Guides
- ISPE Good Practice Guides
- Add related third party content from the world's leading standards authorities, including AAMI, ASTM, BSI, CLSI, DIN, IEC and others

In all, an efficient solution.

Go to www.ISPE.org/Subscriptions or call Techstreet today to start saving time and money with this essential tool.

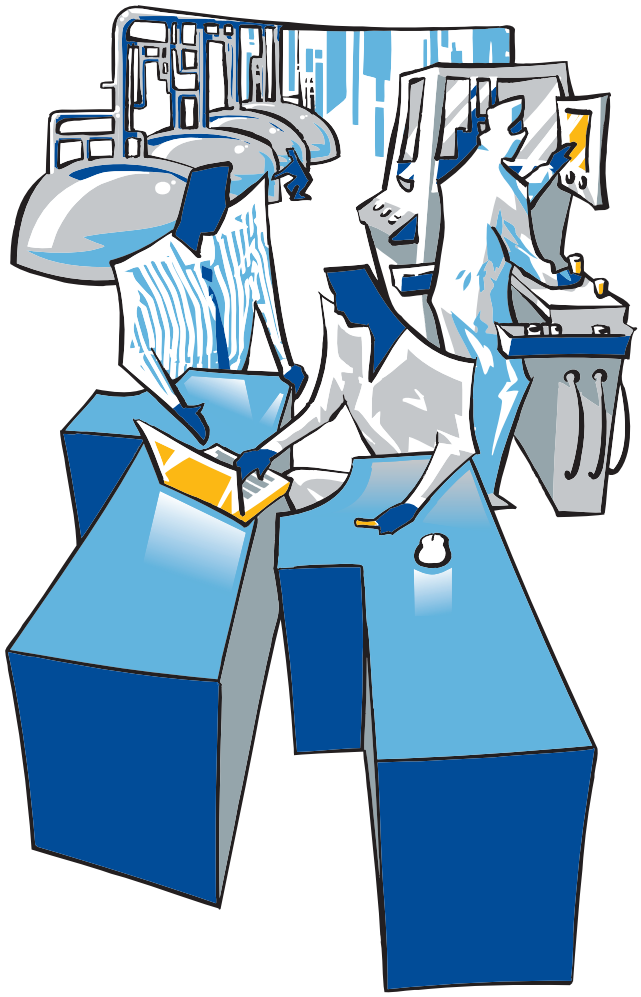
ISPE Subscriptions are offered in partnership with the Techstreet business of Thomson Reuters, the world's leading source of intelligent information for business and professionals.

TECHSTREET®
1.734.780.8000

THOMSON REUTERS

Connecting a World of Pharmaceutical Knowledge

ISPE®



HALFMANN
GOETSCH A
PARTNER G

HEALTHCARE INDUSTRY PARTNERS

Our services:

- Computer System Validation (CSV, GAMP)
- Quality Management
- 21 CFR 11
- Risk & Compliance
- Paperless Production (MES)
- GMP Consulting

Healthcare Industry Partners

Premier consultancy for the pharmaceutical industry

Switzerland • Germany • Singapore

www.hgp.ag Halfmann Goetsch Partner AG • info@hgp.ag • Tel. +41 61 544 0000