GAMP Good Practice Guide:
Testing of GxP Systems
Preface to the GAMP Good Practice Guide: Testing of GxP Systems

This document, the GAMP® Good Practice Guide: Testing of GxP Systems is intended as a supplement to Guide for Validation of Automated Systems (GAMP® 4). It is intended to provide pragmatic guidance on the testing of computerized and software based systems and to encourage Users and Suppliers to work together to ensure adequate test coverage and minimize any duplication of effort.

This document has been designed so that it may be used in conjunction with guidance provided in GAMP® 4 and other ISPE publications, such as the ISPE Baseline® Guides.

Disclaimer:
This Guide is meant to assist pharmaceutical companies in managing the Testing of GxP Systems. The GAMP Forum Testing Special Interest Group (SIG) cannot ensure and does not warrant that a system managed in accordance with this Good Practice Guide (GPG) 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 (including the GAMP Forum), or the officers, directors, employees, members, or agents of each of them, 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.

© Copyright ISPE 2005.

All rights reserved. No part of this document may be reproduced or copied in any form or by any means – graphic, electronic, or mechanical, including photocopying, taping, or information storage and retrieval systems - without written permission of ISPE.

All trademarks used are acknowledged.

ISBN 1-931879-44-3
ACKNOWLEDGEMENTS

The production of the GAMP® Good Practice Guide: Testing of GxP Systems was initiated by the GAMP® Europe Steering Committee and governed by a Special Interest Group chaired by David Stokes of Mi Services Group. The Testing Special Interest Group was sponsored by Kate Samways of KAS Associates Ltd.

The following Special Interest Group members and members of the GAMP® Forum provided the bulk of material, comments, and reviews:

Mats Andersson  
Kevin Ashley  
Karen Ashworth  
Nandakishore Banerjee  
Yoel Bergman  
Heinrich Berlejung  
David Brash  
Anne Broadbent  
David Brown  
Roger Buchanan  
Pierrette Cober  
Andy Dickson  
Jason Dow  
Dot Edwards  
Stephen Elms  
Bob Goodenough  
Martin Gregory  
Paul Harvey  
Louise Killa  
Richard Labib  
Mark Lawson  
Tony McDonagh  
Carsten Mulvad  
Carina Nilsson  
Darren Oakes  
Suzie Poulton  
ian Ransome  
Clive Rowland  
Matt Safi  
Alessandro Sciacca  
Kay Shafl-Roden  
Flemming Simonsen  
George Smerdon  
Gillian Speed  
Chandramohan Thiruvamkulam  
Anders Vidstrup  
Charlie Wakeham  
Nick Wheeler  
Robin Whitworth

AstraZeneca  
EuroDPC  
Karen Ashworth Consulting Ltd  
Isardata Software Quality Engineering GmbH  
Tescom International  
Propack Data  
Warrington Consultancy Ltd  
AstraZeneca  
GlaxoSmithKline  
Eli Lily & Company  
ps_testware  
G4 Consultancy Ltd  
ImClone Systems  
PL Consultancy  
Novartis Pharmaceuticals  
Evotec OAI Ltd  
consignit  
AstraZeneca  
LogicaCMG  
West Pharmaceutical Services  
Boots Healthcare International  
Elan Corporation  
Novo Nordisk Engineering A/S  
AstraZeneca  
Pfizer, PGM IT  
AstraZeneca  
AstraZeneca  
AstraZeneca  
Eurotherm  
Janssen-Cilag S.p.A.  
Tescom International  
Ferring Pharmaceuticals A/S  
Industrial Technology Systems Ltd  
Aptuit  
Amgen Inc  
Novo Nordisk  
Pall Europe Limited  
GlaxoSmithKline  
Specdoc

For ISPE Members only. Copyright ISPE 2005.
The SIG Chair wishes to thank everyone for their commitment and contributions; it has been instructive and pleasant to work with such a devoted group of professionals in a truly international environment.

Special thanks are given to those individuals and organizations who hosted face-to-face meetings and workshops.

Members of the GAMP® Forum Council and Steering Committees, along with the ISPE Technical Documents Committees are thanked for their participation in the review of this Guide.

The GAMP® Editorial Review Board on behalf of ISPE was Gail Evans, Colin Jones, Tony Margetts, Arthur ‘Randy’ Perez, and Sion Wyn.

The GAMP® Council would like to give special thanks to Karen Ashworth and Charlie Wakeham for their extensive work in preparing the final version of this Guide following industry review.
# Table of Contents

**SECTION I - KEY CONCEPTS**

1 INTRODUCTION .......................................................................................................................................................... 11
   1.1 OVERVIEW ............................................................................................................................................................ 11
   1.2 PURPOSE .............................................................................................................................................................. 11
   1.3 SCOPE ................................................................................................................................................................. 11
   1.4 BENEFITS ........................................................................................................................................................... 12
   1.5 OBJECTIVES ......................................................................................................................................................... 12
   1.6 STRUCTURE OF THIS GUIDE ............................................................................................................................... 13

2 KEY CONCEPTS........................................................................................................................................................... 14
   2.1 USE OF RISK ASSESSMENT ............................................................................................................................... 14
   2.2 TESTING AND THE GAMP® LIFE CYCLE ......................................................................................................................... 15
   2.3 TESTING PLANS OR STRATEGIES ............................................................................................................................... 16
   2.4 TESTING AND THE HARDWARE/SOFTWARE TYPE AND MATURITY ................................................................................. 20
   2.5 TESTING RESPONSIBILITIES - SUPPLIER AND USER ......................................................................................................... 20

3 KEY DEFINITIONS ........................................................................................................................................................... 27

4 GUIDANCE....................................................................................................................................................................... 28

**SECTION II – USER AND SUPPLIER CONSIDERATIONS** ............................................................................................... 29

APPENDIX C1 - USER CONSIDERATIONS.............................................................................................. 30
   1 MINIMIZING USER TESTING ........................................................................................................................................... 30
   2 AGREING ROLES AND RESPONSIBILITIES ........................................................................................................ 34
   3 DETERMINING APPROPRIATE TEST EVIDENCE ...................................................................................................... 34

APPENDIX C2 - SUPPLIER (INTEGRATOR) CONSIDERATIONS ............................................................................. 36
   1 SUPPLIER ASSESSMENT ................................................................................................................................................ 36
   2 USE OF THIRD PARTY PRODUCTS ............................................................................................................................. 36
   3 CONTRACTUAL ISSUES ................................................................................................................................................. 36

**SECTION III - TEST PRACTICES** ........................................................................................................................................ 38

APPENDIX T1 - TEST POLICY ............................................................................................................................................... 39
   1 TESTING POLICIES ................................................................................................................................................... 39
   2 DETERMINATION OF THE SCOPE OF TESTING ......................................................................................................... 39
   3 TEST TYPES WITHIN SDLC TEST PHASES .................................................................................................................. 42
      3.1 STRUCTURAL TESTING ........................................................................................................................................ 42
      3.2 FUNCTIONAL TESTING ........................................................................................................................................... 43
# GAMP® Good Practice Guide: Testing of GxP Systems

## 2 AUTOMATED TEST EXECUTION (AND COMPUTERIZED TEST MANAGEMENT TOOLS)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 The Benefits of Computerized Test Management Tools and Automated Test Execution</td>
<td>86</td>
</tr>
<tr>
<td>2.2 Computerized Test Management Tools and Automated Test Tool Features</td>
<td>87</td>
</tr>
<tr>
<td>2.3 Verification of Computerized Test Management and Automated Test Tools</td>
<td>90</td>
</tr>
</tbody>
</table>

## APPENDIX T6 - TEST RESULTS RECORDING AND REVIEWING

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 TEST EVIDENCE</td>
<td>91</td>
</tr>
<tr>
<td>1.1 Paper Evidence</td>
<td>91</td>
</tr>
<tr>
<td>1.2 Electronic Evidence</td>
<td>91</td>
</tr>
<tr>
<td>2 TEST RECORD INTEGRITY</td>
<td>91</td>
</tr>
<tr>
<td>2.1 Paper Records</td>
<td>92</td>
</tr>
<tr>
<td>2.2 Electronic Records</td>
<td>92</td>
</tr>
</tbody>
</table>

## APPENDIX T7 - TEST REPORTING AND SYSTEM HANOVER

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 TEST REPORTS</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>2 FORMAL HANOVER AND RELEASE</td>
<td>94</td>
</tr>
<tr>
<td>2.1 Handover of System</td>
<td>94</td>
</tr>
<tr>
<td>2.2 Handover of Responsibilities</td>
<td>95</td>
</tr>
<tr>
<td>2.3 Conditional Release</td>
<td>95</td>
</tr>
</tbody>
</table>

## APPENDIX T8 - TESTING IN THE OPERATIONAL PHASE

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 TYPES OF CHANGE</td>
<td>96</td>
</tr>
<tr>
<td>1.1 Supplier Driven Change</td>
<td>96</td>
</tr>
<tr>
<td>1.2 User Driven Change</td>
<td>96</td>
</tr>
<tr>
<td>2 CHANGE CONTROL AND CONFIGURATION MANAGEMENT</td>
<td>97</td>
</tr>
<tr>
<td>3 TEST PLANNING AND TEST MANAGEMENT</td>
<td>97</td>
</tr>
<tr>
<td>4 TESTING CHANGES</td>
<td>97</td>
</tr>
<tr>
<td>4.1 Testing of the Changed Element</td>
<td>97</td>
</tr>
<tr>
<td>4.2 Regression Testing of a Supplier Driven Change</td>
<td>98</td>
</tr>
<tr>
<td>4.3 Regression Testing of a User Driven Change</td>
<td>98</td>
</tr>
</tbody>
</table>

## Section IV - Examples

### APPENDIX E1 - TESTING PROCESS AUTOMATION SYSTEMS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 DEFINITIONS</td>
<td>100</td>
</tr>
<tr>
<td>1.1 Configurable Equipment</td>
<td>100</td>
</tr>
<tr>
<td>1.2 Embedded Systems</td>
<td>100</td>
</tr>
<tr>
<td>1.3 Standalone Systems</td>
<td>101</td>
</tr>
<tr>
<td>2 TESTING AND THE GAMP® LIFE CYCLE</td>
<td>101</td>
</tr>
</tbody>
</table>
Section I - KEY CONCEPTS
1 Introduction

1.1 Overview

GAMP Good Practice Guide: Testing of GxP Systems has been written to provide Users and Suppliers with pragmatic guidance on the testing of computerized and software based systems that impact upon (pharmaceutical or equivalent) product quality, patient safety, patient confidentiality, or data integrity. The key objective of this Guide is to encourage Users and Suppliers to work together to ensure adequate test coverage and minimize any duplication of effort.

Specifically, the Guide sets out to identify to Suppliers and Users the testing that should be conducted on products and applications and the level of documentation required. Where Suppliers do not meet these expectations, the Guide identifies suitable User risk mitigation strategies, including the execution of additional User testing or the selection and use of alternative Suppliers and/or products.

1.2 Purpose

The purpose of this Guide is to provide pragmatic advice to questions commonly asked by those responsible for the testing of GxP systems such as:

- 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 the concept of risk based validation (as established in GAMP® 4 (see Appendix G2, reference 1)) and provide practical advice on the application of these concepts in the planning and execution of risk based testing.

The Guide has been written to be of use to:

- Users
- Suppliers (usually external third parties, but may also include ‘in-house’ providers of IT services)
- Systems integrators (responsible for the configuration of the product into a specific application which may contain custom code).

1.3 Scope

This Guide has been written by Users and Suppliers primarily associated with the pharmaceutical industry. This is reflected in some of the terminology defined and used throughout this Guide. However, the concepts and guidance given may be of equal relevance in other sectors of the regulated life sciences industry such as medical devices, biotechnology, biomedical and healthcare.

In answering the questions in section 1.2 above, focus has been given to those areas that have traditionally caused most confusion and given rise to a variety of interpretation within the industry, specifically:

- General User and Supplier/integrator roles and responsibilities
- Testing configurable software packages (GAMP® software category 4)
- Testing custom software (GAMP® software category 5), often developed to supplement an ‘off-the-shelf’ application

In order to be applied to large complex systems, the Guide addresses many issues in detail and provides a breakdown of activities and deliverables. The concepts can, however, be applied to small, simple systems and it is important to consider the key issue of scalability when testing simpler systems, e.g., combining the content of...
documents and combining project roles. Examples given in Appendices E1 and E3 of this Guide apply the concept of scalability to different types of systems.

Finally, in the preparation of this Guide the members of the SIG have deliberately chosen not to redefine testing best 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 (see bibliography).

1.4 Benefits

The principal business benefit from testing systems is that it is more cost effective to move into the production environment with systems that are fit for their intended use.

Anyone who has been involved in a project with insufficient or inappropriate testing learns that those problems only exposed later in the system life cycle are usually the most time consuming and troublesome to resolve (see Section 10.3 of the GAMP® 4 ((see Appendix G2, reference 1)).

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’.
• Increasing the 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 production environment.

The net effect of insufficient or ineffectual testing is to increase the overall life cycle costs of implementing and owning the system and to delay or prevent the effective and efficient use of the system.

1.5 Objectives

For Users the basic underlying reason for testing a computerized system or application is to provide a documented a high level of assurance that the system is fit for its intended use, prior to the system being used in the live environment.

For Suppliers the basic underlying reasons for testing are to prevent the presence of avoidable defects in the supplied system and to ensure that the system is fit for the intended market.

There are a number of different reasons why this is desirable, which can be summarized as:

• Assuring data integrity, product quality the safety of users, consumers (patients) and members of the general public,
• Improving reliability and reducing system downtime, thereby increasing user confidence in the system,
• Reducing the cost of commissioning, start-up, and on-going support.

Testing is a fundamental requirement of current best practice with regard to achieving and maintaining regulatory compliance. Although the need to test computerized systems is defined by certain regulatory guidance (e.g., Section 5 of the FDA’s “General Principles of Software Validation: Final Guidance for Industry and FDA Staff” and Part 2, section 13 of PIC/S “Good Practices for Computerised Systems in ‘GxP’ Environments”), the way in which computer systems should be tested is not defined in detail.

As described throughout this guidance, the nature and extent of computer systems testing should be defined and justified on a function-by-function or system-by-system basis, and this may be based upon a documented risk assessment. It is, however, a basic regulatory expectation that GxP computer systems require some degree of testing.
Failure to test may undermine any validation case and the compliance status of the system. Where discovered during regulatory inspection, this may lead to citations and warning letters being issued and possibly a failure to grant new drug/device licenses, license suspension, or products being placed on import restrictions.

These regulatory expectations are based on the principle that computer systems are tested in order to confirm that user and functional requirements have been met and in order to assure data integrity. These, in turn, are driven by a need to assure patient safety and health.

Before a computerized system is brought into use, it should be appropriately tested and confirmed as being capable of achieving the desired results.

If a system is being replaced, consideration should be given to running the two in parallel for a time where practical, as a part of this testing and validation.

Any modification should trigger a risk assessment in order to determine the extent of any revalidation including any regression testing required. Alterations to a system should be made only in accordance with a defined procedure, which should include provision for approving, implementing, validating, and, if necessary, backing out the change.

There are other regulations which may impact the testing of GxP systems, for example, Health and Safety legislation and Environmental control.

### 1.6 Structure of this Guide

This introduction to the GAMP® GPG: Testing of GxP Systems provides introductory text describing the relation to other key ISPE documents and explains the scope, purpose, and benefits of this guidance.

The initial sections are intended to help readers to rationalize the scope and nature of the testing that they may already be conducting. This will help focus testing activities on areas of the greatest risk priority and optimize the use of valuable test resources.

Starting with a review of why testing is important, the Guide leverages some fundamental ideas of GAMP® 4 (see Appendix G2, reference 1) (software and hardware categories and risk assessment) and establishes some key concepts for testing GxP systems. These concepts are expanded upon in order to demonstrate their application to common types of system and software.

For those less experienced with the testing of GxP systems, the later sections of this Guide expand upon the key concepts and provide more practical advice and guidance in the planning and execution of such testing. For those entirely new to the testing of computer and software based systems, references are given to further reading on the subject.

Some templates and examples are also included in the appendices that should allow the less experienced reader to quickly develop a series of appropriate documents, suitable for planning, executing, and reporting on the testing of a GxP system.
2 Key Concepts

The following section outlines the key concepts used in the planning and execution of GxP system testing and builds upon the philosophy of risk-based validation established in GAMP® 4 (see Appendix G2, reference 1). Further details on the application of these key concepts are included in the appendices to this Guide.

2.1 Use of Risk Assessment

General concept: the scope of testing should be determined by a justified and documented risk assessment, taking into account both the potential effect on product quality and its effect on public safety and the intrinsic risk associated with the method of implementation.

As defined in GAMP® 4, Appendix M3, (see Appendix G2, reference 1) risk assessment should be a fundamental part of the validation process. This can be used to determine the appropriate nature and scope of testing and the extent of the testing documentation.

The GxP impact of the requirements is one input to the risk assessment process. This requires that the User understands and interprets the applicable GxP regulations. Combining GxP impact with risk likelihood and probability of detection provides an indication of risk priority for each requirement.

Individuals who understand the architecture of the software may be involved in the risk assessment and can often provide valuable insight. In many cases it may make sense to involve the Supplier in the identification of potential risk scenarios.

Consideration of the Supplier assessment and the maturity of the supplier and product (in terms of history of compliance with an appropriate quality system, number of existing users, length of time in market, product complexity, history of use of the Supplier in the regulated industry, etc.) will indicate an appropriate scope and rigor of testing. For example, in the case of a mature supplier and mature product only positive case acceptance testing may be required. Where the Supplier or product is less mature it may be appropriate to conduct additional testing (negative case testing, software module testing, etc.).

A key objective is to ensure that all necessary testing is completed without repetition or duplication of effort.

Figure 2.1: The Use of Risk Assessment in Determining the Scope and Rigor of Testing
The same risk assessment process should be included as part of the change impact assessment and regression analysis in order to determine the nature and rigor of change testing and regression testing.

2.2 Testing and the GAMP® Life Cycle

General Concept: testing should be carried out as part of a formal development life cycle and test cases should be written and executed against documented requirements and/or design specifications.

It is an assumption within this Guide that the system requirements and/or design specifications have been adequately defined and documented and that an appropriate development life cycle is in place. The guidance concentrates on the testing required within that life cycle.

The GAMP® Guide (GAMP® 4) (see Appendix G2, reference 1) describes a framework for specification and Qualification from the Users point of view.

Figure 2.2: GAMP® ‘V’ Model

The terminology is based on the three main qualification activities and their link to the basic levels of required specifications.

From a regulatory perspective the responsibility for qualification activities resides with the User. In a practical sense, testing is usually a shared responsibility of both the Supplier and User, which should be agreed at the start of the project.

Testing conducted in earlier project phases (whether conducted by the Supplier or User) should not be repeated unless required as the result of subsequent changes (i.e., changes to requirements, test environment, software version etc.).

Figure 3 (based upon GAMP® 4, Figure 8.2 (see Appendix G2, reference 1)) shows the different types of testing associated with various requirements and specifications. Software module and some levels of integration testing are usually the responsibility of the Supplier (software vendor or system integrator) whereas the User is usually responsible for the functional, acceptance and performance testing (qualification).
The relationship between the testing of a computerized system and the commissioning and qualification of associated equipment or processes should clearly be defined for applicable projects.

Appendices E1 to E5 of this Guide further develop this key concept for different system types as listed in Table 2.1:

### Table 2.1: System Types and Associated Appendices

<table>
<thead>
<tr>
<th>SYSTEM TYPE</th>
<th>APPENDIX REFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Automation Systems (DCS, PLC, SCADA)</td>
<td>E1</td>
</tr>
<tr>
<td>Analytical Instruments</td>
<td>E3</td>
</tr>
<tr>
<td>Desktop Applications (Spreadsheets, Databases)</td>
<td>E4</td>
</tr>
<tr>
<td>Infrastructure and Interfaces</td>
<td>E5</td>
</tr>
</tbody>
</table>

### 2.3 Testing Plans or Strategies

**General Concept:** testing should follow an agreed test plan or strategy.
The following summary points also should be considered when agreeing on a Test Plan or Strategy. More detailed guidance can be found in the appendices of this Guide.

### 2.3.1 Test Planning and Test Management

The following points should be considered when planning tests in accordance with an established Test Plan or Strategy:

- Alignment with the requirements of the Validation and/or Quality Plan
- Location and timing of test phases
- Resources required for each test phase
- Responsibilities for each test phase
- Format of test references and incident references
- Planned coverage for each test phase (and traceability against established requirements)
- The use of test metrics

The planned test coverage should reflect the relative risk priority associated with the system element under test. See Appendix T2 of this Guide for further details.

### 2.3.2 Test Documentation

This section discusses the basic test documentation requirements, presented diagrammatically in Figure 2.4. For Users this test documentation, including any separate Test Summary Report, will be summarized in the Validation Report. (See GAMP® 4 Appendix M7 for further information (see Appendix G2, reference 1).)

Test documentation should be subject to appropriate independent review and approval. This is often conducted by a representative assigned by the relevant organization’s Quality function in accordance with their procedures.
When determining an appropriate Test Plan or Test Strategy it should be borne in mind that the content of documents may be combined when testing small/simple systems or the testing of minor changes to larger/complex systems, for example:

- Although one-to-many relationships are shown in the diagram, in a simple system, a single document may encompass the Test Plan or Strategy, Test Protocols/Test Specifications and Test Cases/Test Scripts and a single Test Report may be all that is required.
- When Testing small or simple systems, test inputs, test environment set-up and expected results may all be covered within the Test Scripts, and a separate definition of Test Cases may not be required (this is the most common approach).

(The term ‘test approach’ is used through this Guide to describe a general approach to testing that may be documented in a Test Plan or Test Strategy.)

For larger or more complex systems the following also should be borne in mind:

- The Test Plan or Strategy should define the common approach to the different phases of testing, which are further defined in subsequent Test Specifications or Protocols.
- Multiple Test Specifications or Protocols may be required (typically one per phase of testing). In this case each Test Specification or Protocol should have an associated Test Report, and the Test Reports should be summarized into a Test Summary Report (which is associated with the Test Plan or Strategy)
- Test Cases and Scripts may be grouped into Test Groups or Test Sets (not shown in Figure 4) for ease of test planning, monitoring, and execution.
- Separate Test Cases may be prepared for some tests, which may describe complex test data sets, test methods, test input data, test environment set-up, and expected results. In this case Test Scripts may be used solely to document the sequence of actions (test steps) required to conduct a specific test. One Test Script may be used as the basis for conducting multiple similar Test Cases.

For all systems the test documentation should describe the use of appropriate Test Cases and/or Test Scripts, including test objectives(s), necessary pre-requisites, steps involved in performing each test, data to be recorded, evidence to be collected, and acceptance criteria. See Appendix T3 of this Guide for further details.
2.3.3 Test Environment

There will typically be a test environment which is separated from the production environment logically, physically or in time (the latter being typical of laboratory and process control systems site acceptance testing). The Test Plan or Strategy should consider the hardware, software, test data sets, user accounts, and reference documents that will be part of the Test Environment.

The test environment should be verified as being as representative as possible of the production environment. Differences should be documented and assessed for the level of impact introduced by the differences to allow additional tests to be planned for the production environment if required.

The test environment should be documented and controlled to a level of detail that would allow it to be reconstructed or emulated if necessary.

Where test hardware/software/data/user accounts are applied in such a way that they may appear in the production environment, controls should exist to ensure that they can either be removed cleanly or be isolated from use (either logically or in time). See Appendix T4 of this Guide for further details.

2.3.4 Test Execution

Test execution can be performed manually or automatically. The Test Plan or Strategy should describe which method can be applied where.

When computerized test management tools and/or automated test tools are used, special care should be taken to assure they are fit for their intended purpose. See Appendix T5 of this Guide for further details.

2.3.5 Test Results Recording and Reviewing

The Test Plan or Strategy also should consider the recording and review of test results, including the method for recording and filing passed and failed tests. The method for documenting, processing, and closing down test incidents and the justification for the use of any test witnesses also should be addressed in the Test Strategy as well as the review of test results and associated documentation.

The requirements for recording and reviewing test results should reflect the relative risk priority associated with the system element under test. Where practical a single suitably qualified or trained tester signing off a test may be appropriate provided that the final results are independently reviewed. For complex functions, test execution may require multiple test roles (e.g., Supplier, engineer, and user production staff). See Appendix T6 of this Guide for further details.

2.3.6 Test Reporting and System Handover

The Test Plan or Strategy also should define key considerations when producing test reports and planning to handover the system from one test phase to another, including the format of and responsibility for final test reports, the method of and authority for system handover and any contractual implications.

The method for system handover may need to consider circumstances in which a conditional handover can be made – for example with workarounds in place, test incidents still open or tests still to be completed because they are not possible outside of the production environment. Such conditional handover should be supported by a documented risk assessment and defined risk mitigation activities being in place.

The method also should be defined for ensuring that the baseline recorded at the end of one test phase matches the baseline recorded at the start of the next phase (for example to ensure that the software at the start of site acceptance testing is the same as that released at the end of factory acceptance testing). See Appendix T7 of this Guide for further details.

2.3.7 Testing in the Operational Phase

Once a system has been implemented, there may be a need for future changes. This may be as a result of changes in requirements, changes implemented to correct software defects, system upgrades, or patches. For details on testing during change management see Appendix T8 of this Guide.
2.4 Testing and the Hardware/Software Type and Maturity

**General Concept:** as with other validation effort, the testing approach should reflect risk to product quality, patient safety, and data integrity. The nature of the software and hardware and the supplier maturity are all factors affecting this risk.

2.4.1 GAMP® Hardware and Software Categories

Since the likelihood of system failure increases with the progression from standard software and hardware elements to custom (bespoke) software and hardware elements, a classification of system elements into categories can help support a risk-based test approach. Categorization of hardware and software as described in GAMP® 4 Appendix M4 (see Appendix G2, reference 1) is assumed throughout the remainder of this document.

2.4.2 Hardware and Software Maturity

In deciding the test effort required for a system element, the maturity of the hardware or software also may need to be taken into account with additional effort devoted to test elements that are not considered ‘industry proven’. This can apply both to the maturity of the standard elements within a Supplier’s product and to the maturity of custom elements re-used from one application to the next.

For example, where a User is unable to accept the standard functionality offered by a Supplier and requests a modified software module, additional testing of the differences between the standard offering and the modified software module is likely to be appropriate. Conversely, where a custom module is re-used in later applications, a reduced test effort is likely to be appropriate.

2.4.3 Supplier Maturity

There is a higher probability that a product from a new Supplier will have faults compared to a product from an established Supplier. A mature Supplier is more likely to recognize the importance of quality management and to have established quality management processes.

In these cases, Users may rely on the documented testing conducted by such mature Suppliers and should not repeat such testing. See section 3 for further details.

2.5 Testing Responsibilities - Supplier and User

**General Concept:** where possible, users should seek to benefit from supplier quality assurance processes and associated testing.

Where a Supplier has been assessed and their quality management practices are found to be appropriate, the User may benefit from the testing already carried out as part of the product development lifecycle. This may reduce the need for additional testing carried out by the User.

Regardless of the categorization of the software acquired by the User, all software would at some stage have been written for the first time by the Supplier and could be considered as customized code during development by the Supplier (synonymous with GAMP® software category 5). It is, therefore, appropriate to consider the Supplier’s development life cycle (and any integral testing) when considering the scope and nature of testing to be conducted by the User.

The testing of the software (or system) is, therefore, 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
- Testing conducted by the User

Where there is assessed evidence that Supplier testing is appropriate to the risk impact associated with the software or system, and where testing is executed in a comparable test environment, the User need not repeat such testing as long as an appropriate Supplier assessment has been conducted, including a review of general Supplier test
activities and a review of system, software or release specific testing. User testing should then focus on customized, configured, and critical (high risk priority) functions.

The examples that follow show Supplier and User development and test activities and are based on the basic testing ‘V-model’ life cycle shown in section 2.2. Some Suppliers may use alternative development life cycles other than those based upon the ‘V-model’ shown.

Alternative development life cycles may be perfectly acceptable - the important issue is to focus on the purpose, nature and scope of the Suppliers documented test activities and the veracity of the test results. Where these are appropriate to the risk impact associated with the Users application of the software these activities need not be repeated by the User.

The following sections describe the approach to testing various software categories. Note that most systems contain multiple categories of software, and the system specific approach to testing may be a combination of these models. System specific examples of these are provided in the ‘E’ Appendices of this Guide.

2.5.1 Operating Systems (GAMP® Software Category 1)

Established operating systems are not subject to specific validation although their features are functionally tested and challenged indirectly during testing of an application (see GAMP® 4 Appendix M4 (see Appendix G2, reference 1)).

2.5.2 Firmware (GAMP® Software Category 2)

The creation of firmware typically involves code written by the Supplier. The Supplier should, therefore, generally follow a full product development life cycle (either to the life cycle recommended for GAMP® software category 5 or to equivalent standards).

On purchasing an item with embedded firmware the User does not need to repeat testing already carried out by the Supplier, assuming that the Supplier has suitable quality practices in place and that the firmware is ‘standard’ (rather than being developed or modified specifically for the Users’ application).

The application life cycle verification activities can be limited to verifying the firmware version and that the parameters entered give correct operation as defined in the user requirements.

The diagram below shows how the Suppliers development and test life cycle is used to develop a product which is then configured for use by the User.
Figure 2.5: Test Framework for GAMP® Software Category 2

- End User Application Life Cycle (responsibility)
- Supplier Product Life Cycle (responsibility)


Performance Testing

System Acceptance Testing

Software Integration Testing

Verify Operation versus Requirements

Configure Firmware

User Requirement Specification

Product used in end User application

For ISPE Members only. Copyright ISPE 2005.
2.5.3 Standard Software Packages (GAMP® Software Category 3)

The creation of Standard Software Packages typically involves code written by the Supplier. The Supplier should, therefore, generally follow a full product development life cycle (either to the life cycle recommended for GAMP® software category 5 or to equivalent standards).

On purchasing a standard package the User does not need to repeat testing already carried out by the Supplier, assuming that the Supplier has a suitable quality management system in place and that the package is ‘standard’ (rather than being developed or modified specifically for the Users’ application).

The application life cycle test activities can be limited to verifying the installed software package version and that the parameters entered give correct operation as defined in the user requirements.

Figure 2.6: Test Framework for GAMP® Software Category 3

| = End User Application Life Cycle (responsibility) |
| = Supplier Product Life Cycle (responsibility) |

2.5.4 Configurable Software Packages (GAMP® Software Category 4)

The creation of Configurable Software Packages typically involves code written by the Supplier. The Supplier should, therefore, generally follow a full product development life cycle (either to the life cycle recommended for GAMP® category 5 or to equivalent standards).

In order to meet the requirements of the specific user, the software is then configured by the Supplier, a systems integrator or the User.

On purchasing a configurable package the User does not need to repeat testing already carried out by the Supplier, assuming that the Supplier has a suitable quality management system in place and that the package is ‘standard’ (rather than being developed or modified specifically for the Users’ application).

The application life cycle test activities can be limited to those which verify that the configuration has been correctly implemented such that the overall system performs as defined in the user requirements.
Risk assessment should consider the need to test functions within the system that the User does not intend to utilize to ensure that there is no inadvertent interaction with the required functionality. It is not usually necessary for Users to test unused functions where the following conditions can be met:

- Supplier testing adequately demonstrates that unused functions do not interact with functions configured and utilized by the User.
- Supplier release notes adequately describe the extent of any interactions between functions that are and are not utilized by the User.

**Figure 2.7: Test Framework for GAMP® Software Category 4**

- End User Application Life Cycle (responsibility)
- Supplier Product Life Cycle (responsibility)
2.5.5 Custom (Bespoke) Software (GAMP® Software Category 5)

Where Users (possibly working with their Suppliers) develop a system that solely contains custom software, Figure 2.8 shows the User’s life cycle that should be followed. Note that because the system is not based upon a standard Supplier product, there is no Suppliers life cycle to be considered.

**Figure 2.8: Test Framework for GAMP® Software Category 5**

In the more likely case where custom development is required to modify or customize a Supplier’s standard product for use in a specific application, the full development and testing life cycle needs to be followed for custom modifications, as above. In addition, the User’s specific configuration also will need to be tested, as for GAMP® category 4 software described above.

The following example looks at the development of custom modules added to a standard configurable product (for example to provide an interface to a separate system). The life cycles already given above can be followed for testing the standard package but the custom modules require a full development and test life cycle of their own.
Figure 2.9: Test Framework for GAMP® Software Categories 4 and 5 (combined)
3 Key Definitions

The use of the terms ‘testing’, ‘verification’, ‘validation’, ‘commissioning’ and ‘qualification’ has often been a source of confusion and inconsistency. Within this Guide:

‘Validation’ is used to describe an overall activity of establishing documented evidence which provides a high degree of assurance that a specific process will consistently produce a product meeting its pre-determined specifications and quality attributes.

‘Verification’ is used to describe any means of confirming that one or more specific requirements have been met. Other forms of verification may include document or code reviews and manual inspection.

‘Testing’ is one form of verification activity that usually forms part of the validation process. This usually includes exercising a system or component under specific conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component.

‘Commissioning’ is used to describe the setting to work of process and equipment and is not defined as testing (although some testing may take place following commissioning).

‘Qualification’ is used solely in the context of IQ, OQ, and PQ. Testing of computerized systems forms a part of a broader qualification activity (which is in turn part of a broader process validation).

A full glossary of testing abbreviations, terms, and associated definitions used in this guidance is included in Appendix G1 of this Guide.
4 Guidance

Detailed guidance on User and Supplier considerations, testing good practice and examples of applying such good practice to specific types of application or system are given in the Appendices of this Guide. These are:

Section II – User and Supplier Considerations discusses important issues concerning the organizational responsibilities of the User and Supplier for testing during the system life cycle. This is organized into:

- Appendix C1 – User Considerations
- Appendix C2 – Supplier (Integrator) Considerations

Section III – Test Practices provides detailed good practice guidance for all aspects of testing and is organized into:

- Appendix T1 – Test Policy
- Appendix T2 – Test Planning and Test Management
- Appendix T3 – Test Specifications or Protocols, Cases and Scripts
- Appendix T4 – Test Environments
- Appendix T5 – Test Execution
- Appendix T6 – Test Results Recording and Reviewing
- Appendix T7 – Test Reporting and System Handover
- Appendix T8 – Testing in the Operational Phase

Section IV – Examples applies the key concepts defined above, the User and Supplier Considerations from section II and the test good practices from section III and shows how these can be applied to different types of systems and applications. These are:

- Appendix E1 – Testing Process Automation Systems
- Appendix E2 – Testing Configurable IT Systems
- Appendix E3 – Testing Analytical Instruments
- Appendix E4 – Testing Desktop Applications
- Appendix E5 – Testing Infrastructure and Interfaces

For ISPE Members only. Copyright ISPE 2005.
Section II – USER and SUPPLIER CONSIDERATIONS
Appendix C1 - User Considerations

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

Where the User is not developing the application themselves it is likely that the User will be acquiring their system from one of two sources.

- **Supplier**: who is developing a product or a custom system (and in the context of this Guide, does not take part in User application life cycle).
- **Integrator**: who is configuring a system specifically for the User and is part of the User’s application life cycle. Their supply is likely to be based on commercial off-the-shelf products which they have configured for the particular User application.

In some cases the same organization may both be the Supplier (product developer) and system integrator (developing the User’s application using their own company’s products). In other cases the Supplier may develop a custom solution on behalf of or with the User.

Although the role of the integrator is highlighted in certain sections of this document, the general use of the term Supplier also includes integrators.

The User should be aware of how their application has been developed, the methodology(s) adopted by the Supplier and the associated testing.

Depending on whether a Supplier is developing a custom system or an integrator is developing a system based on configurable software, the User should confirm the approach being adopted by the Supplier. Custom development should follow a development life cycle appropriate for GAMP® Category 5 software. Specification and configuration of commercial-off-the-shelf software should follow a development life cycle appropriate for GAMP® Category 4 software.

1 Minimizing User Testing

The User can minimize the volume 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 User on identical systems or pieces of equipment

In an ideal situation User testing may be reduced to a level which confirms that the system meets the User Requirements and verifies the adequacy of previous testing.

The User should confirm that the Supplier can meet the requirements described in this Guide. This may be done by an assessment process (See GAMP® 4, Appendix M2 (see Appendix G2, reference 1)) but whatever process is used it should be documented. Where shortfalls are identified the User needs to assess and mitigate the risk scenarios on a case-by-case basis.

In the case where Suppliers do not have a defined methodology for the testing of their systems then Users should consider additional testing. Where additional testing does not appropriately mitigate the Users risk scenarios then it may be appropriate that alternate products and/or Supplier be sought.

Users 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 an accredited quality system such as ISO 9000, TickIT, Software CMM (or CMMI) or an appropriate Testing Maturity Model.

The discussion of Supplier maturity below refers to a good track record within the pharmaceutical (or other life sciences) industry and a history of compliance with an appropriate quality system. Product maturity refers to a history of good product quality with a high level of customer satisfaction in the relevant industry.

Alternatively, any 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 User and Supplier, and this may include additional User testing.
Users 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. Should Suppliers then fail to conduct or document appropriate testing agreed under such a contract, Users may be able to reclaim the cost of additional User testing.

Products that are widely used in the pharmaceutical and associated industries are generally considered to have a lower likelihood of including undiscovered software defects than new products or those developed for general markets and used infrequently in the pharmaceutical industry.

Suppliers who are experienced in the industry and who implement appropriate quality practices generally produce products with a lower likelihood of containing undiscovered software defects. This is due to their greater understanding of regulatory requirements and of the risk impact associated with certain User product profiles.

Users should seek to purchase products with a lower likelihood of containing undiscovered software defects. Market review and an initial Supplier assessment should establish the relative maturity of a product and/or Supplier and this should include a consideration of Supplier testing.

The following diagram shows the relative risk associated with acquiring a solution (a specific computerized system from a specific Supplier). This is based upon the maturity of the product and the Supplier.

**Figure C1.1: Supplier and Product Maturity Model**

<table>
<thead>
<tr>
<th>Supplier Maturity</th>
<th>Product Maturity</th>
<th>High Risk Solution (least preferred solution)</th>
<th>Medium Risk Solution</th>
<th>Low Risk Solution (preferred solution)</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Medium Risk Solution</td>
<td>Less rigorous Supplier Assessment (e.g., postal audit)</td>
<td>Routine surveillance assessments</td>
<td>Less rigorous Supplier Assessment (e.g., postal audit)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rigorous review of product Test Evidence</td>
<td>Intermediate scope and rigor of User testing</td>
<td>Less frequent surveillance assessments</td>
</tr>
<tr>
<td>Low</td>
<td>High Risk Solution (least preferred solution)</td>
<td>Rigorous Supplier Assessment (Audit)</td>
<td>Frequent surveillance assessments</td>
<td>Less rigorous review of product Test Evidence</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rigorous review of product Test Evidence</td>
<td></td>
<td>Lowest scope and rigor of User testing</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td>Highest scope and rigor of User testing</td>
<td>Intermediate scope and rigor of User testing</td>
<td></td>
</tr>
</tbody>
</table>

**N.B.** A situation may arise when an established product/Supplier is acquired a new Supplier. The new Supplier should have ensured that relevant test evidence from the previous Supplier is secure and available for audit.

Regardless of the maturity of the product or the Supplier, the same level of testing should be conducted by all Suppliers and this should be appropriate to the Users determination of risk impact. This level of testing should
already be in place with a mature Supplier and this can be assured by initial Supplier assessment and routine surveillance.

Users may choose to select products which fall into the High or Medium Risk Solution category shown in the above figure, but this will usually result in an increase in assessment rigor (audit) or User documentation and testing to ensure that the increased risk likelihood is appropriately addressed. See GAMP® 4, Appendix M2 (Supplier Audit) for further details (see Appendix G2, reference 1).

Where it is not possible to conduct an assessment of the Supplier (e.g., open source software or uncooperative Suppliers) appropriate risk mitigation steps will need to be taken. This may include additional detailed testing depending upon the maturity of the product and the supplier.

The following diagram shows how, in the situation where a Supplier as an inadequate quality management systems and inadequate test process:

- The User must conduct additional testing,
- The Supplier may nevertheless support the User in the planning and execution of such testing, under the control of the Users quality management system.
Figure C1.2: Relative Users Test Burden when using Preferred versus Least Preferred Solutions

Preferred GAMP Software Category 2 Product (mature product, mature supplier)

- Market Requirement Specification
- Functional Specification
- Software Design Specification
- Software Module Specification
- Code Firmware
- Performance Testing
- System Acceptance Testing
- Software Integration Testing
- Software Module Testing
- User Requirement Specification
- Verify Operation versus Requirements
- Configure Firmware

Least Preferred GAMP Software Category 2 Product (new product, new supplier)

- User Requirement Specification
- Verify Operation versus Requirements
- Create Functional Specification
- Create Software Design Specification
- Create Software Module Specification
- Configure Firmware
- System Acceptance Testing
- Software Integration Testing
- Software Module Testing

Inadequate or un-assessed quality practices (Supplier willing to support User’s Validation)
2 Agreeing Roles and Responsibilities

Where the Supplier assessment identifies shortcomings the User can define appropriate actions to mitigate the risks. These would often be detailed within the contract or Project Plan as agreed by all parties. The level of testing and responsibilities required from both the Supplier and User depend on the category of system being supplied.

In cases where Supplier and product maturity indicate a lower level of risk likelihood the User may determine that they only need to be involved in the execution of User Acceptance testing, with all other testing conducted by the Supplier. In cases where Supplier and product maturity indicate a higher level of risk likelihood the User may determine that they also may need to witness Supplier testing, conduct additional negative case testing or repeat poorly documented Supplier testing.

General guidance on appropriate User/Supplier responsibilities for testing different software categories are described in section 2.5 under Key Concepts above. Examples appropriate to different systems are explained in greater detail in Appendices E1 to E5 of this Guide.

In the following example a third party system integrator is used, working with the User to implement a configurable software category 4 system. Note that the responsibilities in the application life cycle are shared between the system integrator and the User. The User would review the Function Specification written by the system integrator and the system integrator would support the System Acceptance Testing conducted by the User.

Figure C1.3: Test Framework for GAMP® Software Category 4 with shared System Integrator and User Responsibilities

3 Determining Appropriate Test Evidence

Users should define what level of test evidence should be in place before the system can be considered to be validated and how long that evidence should 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 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 it is only useful to retain the test evidence.

The User should build on evidence of testing provided by the Supplier and aim not to duplicate test evidence. Where Supplier test evidence is relied upon to support the validation of the system, Users should either request copies of the Supplier test evidence, or more practically in most circumstances, should assure themselves that Suppliers test evidence will be retained and available for the necessary period. This may be assured as part of the Supplier assessment, or specific contractual terms.
Appendix C2 - Supplier (Integrator) Considerations

The nature of the pharmaceutical and other life sciences industries requires that systems are developed, documented, and tested following good engineering practices. Suppliers should seek to develop and supply systems following a defined methodology such as that described in GAMP® 4 (see Appendix G2, reference 1). Appropriate testing conducted by the Supplier is likely to allow reduced testing by the User.

1 Supplier Assessment

For systems containing category 4 and/or 5 software (or highly critical software of category 2 or 3) it is usual for a User to carry out an assessment of the Supplier. The Supplier should make themselves aware of the areas likely to be covered by that audit (see GAMP® 4 Appendix M2 for an example (see Appendix G2, reference 1)). Being aware of the requirements and preparing for the audit will assist both parties in determining any shortfalls and where specific remedial actions or testing may be required. The audit may be an important step in developing a long term relationship between the Supplier and the User.

2 Use of Third Party products

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 supplier’s products when determining an appropriate level of testing. This Guide provides assistance to Users in the pharmaceutical industry as to how they should approach the testing of supplied systems. The same approaches need to be adopted 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 using good engineering practices 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 these should be tested and documentary evidence provided of fitness for purpose.
- Where third party products are considered to be a “widely used industry standard” then suitable evidence to this should be available.

Just like Users, Suppliers should seek to leverage the testing already executed by their own supplier(s), or testing conducted by themselves on identical systems or pieces of equipment.

3 Contractual Issues

Testing is often a milestone linked to a stage payment. Key points for success are:

- 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 User can assist with this.
- Ensure the Test Scripts are traceable to the various design specifications.
- Involve the Supplier and User 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.
- 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 factor 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 User in order for the project to progress to the next stage.
Section III - TEST PRACTICES
Appendix T1 - Test Policy

1 Testing Policies

It is essential for organizations to have a coherent policy for testing computer systems and this applies to GxP systems in particular. Ideally such policies should be defined at corporate level, but policies also may be established at site or department level if required by regional needs. User organizations should define this within the context of their own computer systems Quality Management System and this may be included within a specific Testing Policy document or may be part of a Validation Policy document.

The details of any organization’s test policy would be specific to the organization and can be based upon the key concepts and examples provided in this guidance. Particular considerations which may need to be covered by a Test Policy are discussed in this Appendix:

- Determination of the scope of testing
- Types of testing (and an agreed terminology for these tests)
- General approach to testing

Specified procedures for testing should lead to consistency in execution of the testing process. Such procedures should be based upon a common testing policy, which should in turn offer the User a series of benefits including:

- Common methodologies being adopted
- Common terminology (and less confusion)
- Common test documentation models
- Use of common test documentation templates
- Easier ‘learning’ by personnel
- Consistent testing processes throughout the organization
- Faster and more consistent reviews

2 Determination of the Scope of Testing

Except for the simplest of programs, software cannot be exhaustively tested. The level of test coverage should be commensurate with the risk priority associated with the system, application, or individual function.

As described in the Key Concepts section, the scope of testing should be determined by a justified and documented risk assessment, taking into account both the potential effect on product quality and patient safety (e.g., direct impact or indirect impact) and the intrinsic risk associated with the method of implementation (e.g., implemented using mature standard product or implemented using custom code).

The risk assessment should categories system or business requirements according to their associated risk priority. This categorization 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 might define three categories; low, medium, and high risk priority. The resulting test coverage requirements might then be defined as follows:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Low Risk Priority</th>
<th>Medium Risk Priority</th>
<th>High Risk Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table T1.1: Recommended Test Types Based upon Risk Assessment
<table>
<thead>
<tr>
<th>Test Type</th>
<th>Low Risk Priority Requirement</th>
<th>Medium Risk Priority Requirement</th>
<th>High Risk Priority Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural Testing*</td>
<td>(Not required - coverage during functional test assumed to be adequate).</td>
<td>100% branch coverage required – typically by Supplier.</td>
<td>100% condition coverage required – typically by Supplier.</td>
</tr>
<tr>
<td>Functional Testing*</td>
<td>Test normal, invalid and special cases as defined in user requirements.</td>
<td>Test normal, invalid and special cases as defined in user and functional requirements. Ensure that test coverage includes exercising of all software outputs.</td>
<td>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.</td>
</tr>
<tr>
<td>Performance Testing*</td>
<td>Test performance against defined user requirements.</td>
<td>Test performance against defined user requirements.</td>
<td>Test performance against defined user requirements.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Test performance under expected load conditions.</td>
<td>Test performance under expected load conditions.</td>
</tr>
<tr>
<td>Challenge Testing*</td>
<td>(Not required)</td>
<td>Test data validity and security.</td>
<td>Test data validity and security.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Test for tolerance of hardware faults.</td>
<td>Test for tolerance of hardware faults.</td>
</tr>
</tbody>
</table>

* see section 3 in Appendix T1 of this Guide for details of different test types. Note that this table refers to development testing and not all test types are included. The concept of risk-based testing can, however, be extended to all test types.

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 the following figure.

The Figure T1.1 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.
## Figure T1.1: Example of Test Scope Defined in Risk Mitigation Strategy

<table>
<thead>
<tr>
<th>User Req. ID</th>
<th>Description</th>
<th>Risk Priority</th>
<th>Type</th>
<th>Risk Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.1</td>
<td>Pressure test sequence</td>
<td>Med</td>
<td>Structural Testing</td>
<td>Test Specifications or Protocols (module test) should ensure that all possible branches are covered.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Med</td>
<td>Functional Testing</td>
<td>Test Specifications or Protocols (factory acceptance) should ensure that normal path (pass) through pressure test operates correctly.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Med</td>
<td>Performance Testing</td>
<td>Test Specifications or Protocols (factory acceptance) should ensure that failure path on pressure test fail operates correctly.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Test Specifications or Protocols (factory acceptance) should ensure that all defined failure conditions (e.g., valve failures) initiate the correct failure action.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Test Specifications or Protocols (site acceptance) should repeat factory acceptance tests with all inputs live.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Challenge Testing</td>
<td>Test Specifications or Protocols (factory acceptance) should establish that the sequence cannot be run by an unauthorized user.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Test Specifications or Protocols (factory acceptance) should establish that hardware faults on inputs or outputs initiate the correct failure action.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Stress testing is not required due to the destructive nature of such tests.</td>
</tr>
</tbody>
</table>
3 Test Types within SDLC Test Phases

This section defines different types of tests that can be executed as part of the product development and project testing. Which types of test are required and the scope and rigor of each test type is determined in part by risk assessment. Some types of test may be included in more than one test phase. Which test types are included in each test phase should be clearly defined.

3.1 Structural Testing

3.1.1 Test Objective

The objective of structural testing or "white-box" testing is to ensure that each program statement performs its intended function. Structural testing therefore identifies test cases based on knowledge of the source code. These test cases 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 the program is run.

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

3.1.2 Test Scope

The scope of structural testing should reflect the risk priority associated with the system or function. Some common levels of structural test coverage 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 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 criteria 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. Because 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 impact or criticality 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.

3.1.3 Test Positioning Within the Life Cycle

Structural testing is carried out primarily within the unit or module test phase.
Source code review is a means for documenting the structural verification of a custom coded module. It should include both review against the required coding standards and review against the design requirements. Source code review is normally carried out prior to the start of formal module testing.

3.2 Functional Testing

3.2.1 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.

3.2.2 Test Scope

Functional testing should normally cover all stated user and 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. Some common types of functional test include:

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

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

- **Special Case Testing** – testing to show that the 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 singularity (for example 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 input domains or selected specifically because they are considered likely to reveal faults.

3.2.3 Test Positioning Within the Life Cycle

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

Design Prototyping (sometimes referred to as Conference Room Pilots or other similar terms), does not form part of formal testing even though it often involves an amount of informal (undocumented) testing. Design Prototyping should be regarded as a means of verifying design requirements and of building confidence prior to formal (documented) testing.

It is in the nature of a prototype to be built up in a rapid, relatively uncontrolled manner. The conversion of a prototype to a real module should, therefore, be approached with caution – as a minimum it is recommended that a baseline be taken and a source code review carried out prior to testing.
3.3 Performance Testing

3.3.1 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).

3.3.2 Test Scope
Performance testing should normally cover all stated performance requirements. For a particular requirement the number and type of performance tests executed may reflect the risk priority associated with the system or function. Some common types of performance test include:

- **Environmental Tests** – Testing to show that the system is capable of operating reliably in the specified environment (for example under the specified temperature conditions). Testing performed by the Supplier is normally leveraged but additional testing may be necessary if the operating environment falls outside the Supplier’s specification for the product.

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

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

- **Timing or Response Tests** – testing to show that the system is capable of meeting the required timing, throughput or response (for example responding to operator requests within the specified period).

- **Load Tests** – testing to show that the system is capable of meeting the required performance whilst operating under realistic high load conditions (for example, with many concurrent users of a database). Load testing can be a complex area and further discussion is given in section 3.3.4.

- **Usability Tests** – testing to evaluate the combined performance of the system and user (for example checking that the user is able to access and respond to information in a timely fashion via a menu system).

3.3.3 Test Positioning Within the Life Cycle
Performance testing is normally carried out during the factory and site acceptance test phases or prior to ‘Go Live’. In order to avoid discovering performance problems when it is ‘too late’ to remedy them it is important to build in performance tests to earlier stages wherever possible. It may be possible to assess performance at an earlier stage using prototypes or theoretical models or by scaling up of results from unit or module test phases. Where differences exist between the test environment and the production environment, it also may be necessary to carry out some performance monitoring and tuning within the production environment.

The following figure provides an example of where various performance testing may be conducted at different stages within the SDLC. This shows that performance testing can be used to assure the robust performance of the system at various phases within the project.
3.3.4 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 realistic high load conditions. In addition, load testing is used to prove architectural design properties and to support operations throughout life of a system through scalability or load balancing.

The problems of 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 and the generation of data for high volume tests. In addition, a proper definition of the expected load is often not available or is unrealistic.

Typical problems that are detected during load testing are:

- Migration of production data fails due to long processing times
- 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 Challenge Testing below for details of stress testing) are shown in the figure below. 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 the system under load. Where system behavior is difficult to predict, formal load testing is recommended.

Load testing requires clear requirements for 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. For example:

**Table T1.2: Examples of Loading Requirements**

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

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

- Data migration, e.g., of production databases
- 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
In addition to measuring the performance against stated requirements, it can be useful to measure system parameters during the load test. For example:

**Table T1.3: Examples of System Parameters to Monitor During Load Testing**

<table>
<thead>
<tr>
<th>System property</th>
<th>Performance Counter/Units</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>Available Bytes/Bytes</td>
<td>The amount of physical/virtual memory in bytes available to processes running on the computer.</td>
</tr>
<tr>
<td>Disc I/O</td>
<td>Disk read (write)/s</td>
<td>The rate of read operations on the disk, e.g., database cache writes</td>
</tr>
<tr>
<td></td>
<td>% disk time</td>
<td>The percentages of elapsed time that the selected disk drive is busy servicing read or write requests.</td>
</tr>
<tr>
<td></td>
<td>Avg. Disk Queue Length</td>
<td>The average numbers of both read and write requests that were queued for the selected disk during the sample interval.</td>
</tr>
<tr>
<td>CPU</td>
<td>%Processor Time</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Processor Queue Length</td>
<td>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.</td>
</tr>
<tr>
<td>Network</td>
<td>% Network Utilization</td>
<td>Percentage of network bandwidth in use on a network segment.</td>
</tr>
<tr>
<td></td>
<td>Bytes Sent(received)/sec</td>
<td>The rate at which bytes are sent (received) on the interface, including framing characters.</td>
</tr>
<tr>
<td>Database</td>
<td>Number of commits and rollbacks/ s</td>
<td>Rate of database transactions commits and rollbacks.</td>
</tr>
<tr>
<td></td>
<td>Avg. time per transaction</td>
<td>Average time per database transaction</td>
</tr>
<tr>
<td></td>
<td>Number cache misses</td>
<td>Measures the efficiency of data access and storage.</td>
</tr>
<tr>
<td></td>
<td>Size in GByte</td>
<td>The size and the growth of the database, e.g., physical available and allocated table size</td>
</tr>
<tr>
<td></td>
<td>Database locks</td>
<td>Number of tables or data locked due to modification of data</td>
</tr>
<tr>
<td></td>
<td>Number of active database sessions</td>
<td>Measures concurrent access onto database</td>
</tr>
</tbody>
</table>

Some parameters are often available from within a system's normal diagnostic information. For others, monitoring tools are available from the hardware or operating system Supplier or from a third party. The measurement of response times can be done by using a clock or more precisely by using specific load testing tools. Load test tool manufacturers often also provide system monitoring tools and result evaluation modules.

In order to evaluate measured results for complex systems, a graphical representation is recommended that correlates actual load to the response of the system. See DIN ISO EN ISO/IEC 14756:1999(E), “Information technology — Measurement and rating of performance of computerised systems for a scientific based model”.

For ISPE Members only. Copyright ISPE 2005.
An example of a load Test Script is given below:

Figure T1.4: Excerpt from Typical Load Test Script

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

3.4 Challenge Testing

3.4.1 Test Objective

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

3.4.2 Test Scope

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

- **Data Validity Tests** – testing to show that the system is capable of avoiding or detecting invalid inputs (for example operator entries in the wrong format, values outside of the allowed range)
- **Security Tests** – testing to show that the system offers protection against control actions or data access by unauthorized users.
- **Fault Tolerance Tests** – testing to evaluate the system’s ability to continue operation in the presence of faults and to recover when the fault condition clears (for example 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 (for example 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 3.4.4.
- **Environmental Tests** – testing to evaluate the system’s ability to continue operation under extreme environmental conditions.
3.4.3 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.

3.4.4 Stress Testing Considerations

The goals of stress testing are to evaluate the system’s ability to continue operation under abnormally high load conditions and to recover when the condition clears. During stress testing potential system limits or data combinations should be identified from which point the system does not work as specified or becomes unstable. In addition, stress testing should prove that a system recovers from extreme situations and works as specified again afterwards.

Challenges associated with stress testing include potential limits and stress situations being difficult to foresee during system development and that systems may not work as specified beyond expected operating limits or do not recover following stress testing. During stress testing it is even possible that damage to the system may occur. Moreover, producing extreme situations for test execution is often very difficult and resource consuming.

Typical problems that are detected during stress testing are:

- 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 as discussed in 3.3.4. The following should, however, be borne in mind:

- Because the system and its environment might become unstable or even unusable after stress testing, a recovery plan should be prepared for the test environment. During test execution appropriate means might 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 for system acceptance. 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 during or after stress testing data might be lost or may not be accessible.

4 Other Testing

In addition, as part of the development life cycle or within later phases of the full system life cycle, other testing is performed for specific purposes. This testing may include any of the test types defined above.

4.1 Regression Testing

4.1.1 Test Objective

The objective of regression testing is to demonstrate, following a change, that portions of the software not involved in the change were not adversely impacted. This is in addition to testing that evaluates the correct functioning of the changes that were made.
4.1.2 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 impacted 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 new test cases are required.

4.1.3 Test Positioning Within the Life Cycle

Regression testing is necessary following changes to previously baselined systems both during the development life cycle (for example following a change made to correct a test incident) and during ongoing operation of the system. See also Appendix T2, section 2.3.5 of this Guide.

4.2 Disaster Recovery Testing

This section is designed to complement GAMP® 4 Appendix O7 (see Appendix G2, reference 1) and be read in conjunction with the GAMP® GPG: IT Infrastructure Control and Compliance (see Appendix G2, reference 5).

4.2.1 Test Objective

Disaster recovery testing has two objectives:

1) 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.

2) To verify, following a disaster, that recovery of the system has been successful.

4.2.2 Test Scope

Testing as Part of Disaster Recovery Planning

Disaster recovery planning seeks to ensure that, in the event of a disaster the disruption to 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 for example include transferring operations to another location.
- Formulation of appropriate test plans to demonstrate that the recovered system is working to specification.
- Formal recording and communication of these plans.

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 (for example 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 the organization, then their testing in a disaster situation should be considered as a requirement in their Service Level Agreement. This testing should align with any in-house testing necessary and should be subject to the same periodic testing.

Testing to Verify Correct Recovery from a Disaster Situation

Depending on what the organization deems to be the most appropriate way of recording this information, it may be decided to split the overall Disaster Recovery Plan into two different (but complementary) documents

- A Business Continuity Plan, 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 Service Level Agreements (SLA) agreed between the business and the supporting IT organization.

- A System Continuity Plan, 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.

Although most Disaster Recovery Plans are developed with the assumption that a disaster would involve major incidents such as natural disasters, accidents, man made disasters or financial crises, the Disaster Recovery Plan also should be scalable enough 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 plan.

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.

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 be invoked which has serious physical and resource implications, it may well be the case that the staff 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.

### 4.2.3 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 acceptance testing of a system.

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.

#### 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. Whilst 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 Disaster Recovery Plan 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 & Safety being placed ahead of development systems.

Thought also should be given to ensuring that any documentation required to support the disaster recovery testing is readily available in the event of a disaster being invoked. It is often the case that administrative file servers and document management systems may have been assigned a lower priority in any systems recovery, so alternative storage arrangements should be considered.
4.3 Decommissioning Testing

4.3.1 Test Objective
The objective of decommissioning testing is to demonstrate, following the decommissioning of a system, that associated systems are not adversely impacted and that archived data can still be accessed. Data migration testing may be an important part of this.

4.3.2 Test Scope
Decommissioning testing is normally achieved by the re-running of original test cases for systems associated with decommissioned system and the execution of test cases to verify the integrity and availability of any data archived from the decommissioned system.

4.3.3 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.
Appendix T2 - Test Planning and Test Management

1 Test Phases and Sequencing of Tests

In developing any system, whether it be hardware, firmware, software or any combination of these it is common sense that wherever possible verification of correct functioning should commence at the smallest possible module (or unit) within that system. With the confidence that each module operates successfully in a ‘stand alone’ mode, investigation of subsequent failure can focus the interfacing (or integration) of these modules.

As the verified integrated modules are combined with other verified modules the number of interfaces grows but testing can usually be restricted to verifying these interfaces and their impact on the overall functionality of the system. This principle of modular testing follows the life cycle model described in GAMP® 4 (see Appendix G2, reference 1).

Dependent on the size and type of system there may be several test phases that support the validation process. These may cover all or some of the following:

- Testing carried out by the Supplier,
- Formal Factory Acceptance Testing (FAT) at Supplier premises,
- Installation and commissioning testing (normally done by the Supplier)
- Formal Site acceptance Testing to verify the system meets the user requirements
- Installation Qualification (IQ)
- Operational Qualification (OQ)
- Performance Qualification (PQ)

Installation Qualification is typically conducted during the build of a qualified environment.

The Operational and Performance Qualification of a system or application is typically conducted as part of qualifying the wider manufacturing or business process.

Wherever the terms IQ, OQ or PQ are used, the relationship of these activities to the testing of systems and applications should be clearly defined.

For each project, the Test Plan or Strategy document should define the test phases required, including:

- Location and timing of the tests,
- Responsibilities (for example, for test execution and review and any necessary training),
- Required test coverage, based upon risk assessment (including, where applicable, the method of selecting a subset of tests already completed by the Supplier for contractual witnessing by the User representative),
- An overview of the test environment to be used,
- The documentation required (base documentation such as Design Specifications, Test Specifications or Protocols, test results proforma and Test Reports required on completion of testing)

2 Test Management Considerations

2.1 Test Management Tools

The use of computerized test management tools is extremely 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,
• Overall test management, through the use of test monitoring tools.

Most modern test management tools provide the ability to configure test process workflows and user roles and significantly reduce the need for paper based test records.

Such systems do require a level of investment. For use in the pharmaceutical and related industries, such test management tools should be appropriately verified and the integrity of test records should be assured (see also Appendix T5 section of this Guide and GAMP® 4 Appendix M4 (see Appendix G2, reference 1)).

The use of such tools is, therefore, recommended for those organizations either:

• Implementing large or complex systems where there is 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.

2.2 Test Coverage and Traceability

The Test Plan or Strategy should require that there is clear, documented traceability between the requirements outlined in the controlling Specifications and the testing undertaken. This traceability, once completed, should demonstrate that every stated requirement has been tested and show how these items relate to each other.

It is important to initiate requirements traceability as early as possible in the project (typically after approval of the URS). Beginning the process of requirements traceability later in the project can provide a real challenge and is much more complicated.

Traceability to requirements should be maintained at the Test Case or Test Script level. If a test covers multiple requirements, further breakdown within the test steps may be considered to demonstrate exactly where a specific requirement is being tested. A Requirements Traceability Matrix (or equivalent) provides the following benefits for testing:

• Documented test coverage of all critical (GxP) requirements,
• Determination of which testing documents 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 the regulators, in order to demonstrate the completeness of testing within the development process for a given application.

It may also be useful to include (or reference) risk assessments within the requirements traceability. Where this is done it is also possible to demonstrate that the scope and rigor of the executed test cases is appropriate to the risk priority associated with the requirements.

Note that not all requirements may be traceable to a Test Case or Test Script. Some requirements may be verified instead of tested, but verification activities and documents can also be included in requirements traceability. See GAMP® 4 Appendix M5 for further details (see Appendix G2, reference 1).
2.3 Management of Test Incidents (Exceptions)

The nature of the software development process means that unexpected incidents often occur during the development life cycle of a system or product, predominantly in the testing phase. Whilst defects and failures are unlikely to ever be totally eliminated, the incidents that do occur should be promptly and correctly dealt with by the test team in order to minimize the likelihood of avoidable defects being passed into the production environment.

Incident notifications may be received via different media (e.g., paper, e-mail or the use of a dedicated incident reporting system) and may, therefore, require variations in the way they are handled. In order to define the approach for dealing with individual input types, an Incident Management Plan (or Procedure) should be prepared and communicated across the test team before any testing starts. Depending on the size of the project, this plan may be specifically developed for the test team or may form part of an overall project incident recording process. On small projects, this plan may form a sub-section of the overall Quality Plan.

The aim of any 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.
- To act as the single point of contact between end-Users and problem solvers, service Suppliers, system developers and testers.
- To handle all incidents in a consistent manner.
- To demonstrate that all incidents have been properly resolved.
- To achieve the agreed level of service criteria.
- To report performance against the service levels set out in the pertinent Service Level Agreements.

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 is useful 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 is beneficial to log incidents in a manner that allows visibility by all relevant staff, allowing effective tracking and monitoring to take place. However, the method selected to do this should be appropriate to the scale and type of testing being undertaken. Smaller projects may find a central spreadsheet approach to be sufficient, whilst larger or more complex projects may wish to use a database, or an application specifically developed for this purpose.

2.3.1 Incident Analysis and Logging

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 any subsequent step. Where a test is continued following a failure, the failed step should be clearly identified on the test results sheet.

It is important that details of all new test incidents are fully recorded and an index of test incidents is maintained. If an incident is a duplicate of an existing incident record it is advisable to cross reference linked incidents so that they can all be closed when the incident is resolved.

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 (summarizing details of the incident, the proposed solution and retest requirements, the review, the implementation and the closure) is given in GAMP® 4 Appendix D6 (see Appendix G2, reference 1).

2.3.2 Test Incident Classification

In addition to correcting an identified fault, it is important that test incidents are evaluated to determine their most likely cause. Addressing this cause is an important part of any Corrective & Preventative Action (CAPA) process. Metrics on the causes of avoidable test incidents provide a useful indicator of areas within the overall software development life cycle that may benefit from improvement activities, in order to ensure that the reduce the likelihood that they might occur again.
Typical incident types that do occur in software testing programs include, but are not limited, to:

**Incorrect Software Installation**

Errors such as program dumps, abnormal terminations or the inability to access the application are often a result of the failure of the installation process, for example if the application has not been installed or configured properly. Installation of a wrong software version may also cause a test incident.

Where this is determined to be the cause of the incident, it is usually necessary to postpone any further testing until the test environment has been correctly set up.

**Incorrect Programming/Coding**

These incidents may result in the actual system output not agreeing with the expected system output. These incidents should be noted and unless the defect is considered sufficiently important to 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.

Where a defect has occurred for this reason, the Functional Design and/or Technical Design Specifications should be checked to ensure that they were not the source of the defect. If they were, they should be updated to prevent such defects happening again.

**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. An example would be where the test case requires the tester to select a particular value from a list of options available, only to find that the required value is not present in the list because it has not been entered in the appropriate data table.

**Inadequate Specification (Incorrect Understanding of Program Functionality)**

Testing failures may occur as a direct result of the fact that the controlling Design Specification is not clear enough about precisely what is required from a particular piece of functionality. This may be particularly evident where a custom system is being developed to satisfy a new business process that may not be fully established yet.

These difficulties may identify that a change is required to the relevant controlling Design Specification. Where a defect has occurred for this reason, the Functional Design and Package Configuration Specifications should be checked to ensure that they were not the source of the confusion.

**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 actually 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 any subsequently agreed change requests. Particular care should be taken to ensure that if any requirements are to be revised after the
design phase has commenced, these are brought to the attention of the Design Team in order 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, therefore, be either of the values, according to whichever one was incorporated into the Technical Design. It is, therefore, essential that the controlling Design Specification is corrected to prevent further confusion.

This incident type should not be confused with Incorrect Programming/Coding, which is where the software coded does not match a particular requirement of the controlling Design Specification, and the code needs to be corrected. Dependant 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.

However, such inconsistencies should be logged in the incident management system in order that the controlling Design Specification can be corrected following the appropriate document management process.

**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, nevertheless requires further investigation. An example would be if, on completing a particular test step, an unexpected pop-up window is generated. This should be recorded in the incident management system in order that the controlling Design Specification and corresponding Test Case or Test Script can be updated to reflect the presence of this window.

**Test Execution Errors**

Tests can be classified as failures if the Tester fails to follow the steps outlined in the Test Script, or the overall Test Protocol or Test Specification governing the activity. An example of this would be where the tester has failed to fill in a result for a particular step to indicate whether it has passed or failed. Another example would be where the tester has ignored instructions to take screen prints at appropriate points to demonstrate correct operation of desired functionality.

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.

**Force Majeure**

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. Examples might include unexpected power outages, fire alarms, 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.

### 2.3.3 Test Incident Prioritization

Software projects do not have unlimited resources, so when incidents do occur, it is necessary to follow a process of prioritization 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 whilst 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 only affect 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 the project's Test Plan or Strategy.

This categorization provides the test/project management team with some indication of the severity of the problems and can 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,
- According to the final severity assigned, allocate appropriate resources to undertake further investigations and rectify the cause of the incident.

An important area that needs to be discussed with the User is the extent to which it is acceptable to release for production use a system with known minor GxP deficiencies and manage these separately, provided that the business is able to operate effectively in the interim period. Risk Assessment can be used to support this process and to prioritize fixes.

The User may be willing to operate with a manual workaround in the short term for a small area of functionality that is not heavily used in order to obtain a significant business or regulatory benefit from the introduction of the remainder of the software functionality. However, there are instances where this is not appropriate. These would have to be rectified before the system could be released.

Although incidents encountered with functionality that falls within the scope of regulatory scrutiny can expect to receive a higher priority than those which do not have a regulatory impact, it also should be understood that incidents that have a potential impact on functionality with other Statutory implications (for example Health & Safety related functionality) may in practice need to take an even higher priority than those falling within the scope of regulatory scrutiny.

External regulatory influences such as Data Protection, Disability Discrimination (or other similar legislation) may also need to be considered alongside the more traditional ones in determining the priority of incident resolution.

2.3.4 Change Control and Configuration Management as part of Test Incident Resolution

Incidents found during test are recorded as described above. Where a change is required, the affected items (hardware or software or documentation) needs to be identified including the version number in which the incident was found. The change control process then needs to ensure that all details of the incident are recorded and that the necessary corrective actions are implemented and tested.

It is acceptable to define a separate process for resolving test incidents but the scope of such a process should be clearly defined with regards to the change control process. The boundaries of each process should be clearly defined and the inter-relationship between test incident management and change control should be clearly defined, including inputs and outputs to/from each process.

As an example, during the test phase of a system under development, impact and regression analysis during test incident assessment may replace the formal impact analysis included as part of the formal change control process. However, implementing any resultant change should still require formal approval.

Configuration management records may be used to help determine the underlying cause of test incidents. As an example, reviewing the inter-relationships between configuration items may help to determine the reason for a test failure. This may also 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.

For further information see GAMP® 4 Appendices M8, M9, and O4 (see Appendix G2, reference 1).

2.3.5 Testing as part of Test Incident Resolution

In addition to the re-execution of failed tests (i.e., those for which test incidents have been raised) in order to verify that test incidents have been resolved successfully, regression analysis and testing is a normal part of the software development life cycle. This establishes confidence that the changes made as a result of incident resolution have not introduced any further defects or undocumented features into the software. This may be
achieved by the development of additional test cases to verify that this is the case, but is more often achieved by the re-running of original test cases that have already been proven to give the expected outcome.

It is good practice to make the regression test suite out of many small tests, instead of fewer large tests. This helps isolate defects and allows more than one tester to become involved in running regression testing, thus minimizing the amount of time spent on necessary re-testing activities.

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

3 Test Roles

3.1 Roles and Responsibilities

The Roles and Responsibilities within any test team can vary greatly according to the nature, scope, and phase of testing to be undertaken as well as the company size and organizational structure. The specific roles that are required for each project should be documented and approved in the Test Plan or Strategy.

The following roles and responsibilities are associated with the testing process. These roles can be split into two distinct groups, roles within the test team itself, and other supporting roles carried out by staff operating outside the test team, but whose responsibilities nevertheless have a direct link to the success of any testing process.

It is important to note that these roles may not always be necessary in every Test Team and only large, complex projects will typically have individuals fulfilling all of the roles defined below. Depending on the size of the project some of these roles, particularly those that interface with the test team, may only require part time involvement or full time involvement for a short time only.

In smaller projects some roles may be combined with others, providing that this does not introduce conflict of interest, or compromise of the independence of review. For example:

- Test Script authors should not review their own test scripts prior to execution
- Testers should not review their own test results/reports

Staff with a particular specialist skill, for example Computer System Validation staff, may fulfill the same part-time role on multiple concurrent projects.

3.1.1 Roles Within The Test Team

Note that individuals within the test team may perform more than one of the following roles, providing that independence of review is not compromised.

**Test Analyst (Test Case or Test Script Author)**

- Defines the Test Cases or Test Scripts including the step-by-step instructions in order to test a specific functional area of the system.
- May undertakes analysis of incidents arising during the test process to determine their root cause, and documenting this to aid the resolution process.

**Test Case or Test Script Reviewer**

- Ensures that all requirements 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 Users organization this role may be fulfilled by QA/QC or IT Quality staff members, but this can also be performed by other Test Analysts.
Testers

- Execute the tests according to the approved Test Cases and Test Scripts.
- Inform the Incident Manager of suspected issues.
- Produce and collate the required documented evidence as defined in the Test Cases and Test Scripts.

Test Witnesses

The use of test witnesses may be required for contractual reasons, e.g., where the User wishes to observe tests conducted by the Supplier, but is not normally required by testing good practice. In many cases the User will rely upon the documented test results recorded by the Supplier, following previously assessed test processes and procedures.

In most cases, well written tests, with clearly defined acceptance criteria and expected results, executed by trained testers who record clear, actual results for subsequent independent review is all that is required.

On occasions where it is necessary for more than one person to view and record the actual results (in the case where a test has multiple outputs or a rapid sequence of events that needs to be observed) it may be necessary for multiple testers to execute and observe the test.

Where the use of a Test Witness is required (e.g., for contractual reasons) this role can usually be combined with that of Test Result Reviewer.

Test Result Reviewer

- Ensures that all test steps of the Test Scripts have been executed.
- Ensures that documented 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.

Test Manager

- Overall responsibility for planning and coordinating all the test activities and reporting the status of the testing activities to the Project Manager.

Test Team Leader

- 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, thus allowing the Test Manager to concentrate on the planning and strategic activities necessary to ensure the testing meets its agreed timelines.

Test Infrastructure Team

- Provides the hardware, middleware and network connections necessary to operate a test environment with the correct characteristics required for the testing to be undertaken.

Technical Staff

- Where the testing being undertaken requires installation of software or other technical tasks, the Test Team may expand to include the temporary resources of additional technical staff (for example, qualified engineers) in order to complete the required activities.
Test Team Administrator

- Ensures that the administrative tasks arising during the testing process are completed in a timely manner.
- This may include tasks such as document administration, assistance with resource assignment and time recording activities, minute taking at relevant meetings, staff induction and any other tasks delegated by the Test Manager.

3.1.2 Roles Interfacing with the Test Team

Software Developers

- Provide support to the Test Team during Test Case or Test Script creation, by providing early indications on the implementation of the design, for example, indicating the expected keystrokes that would be required for a given function in order to ensure that the Test Script is constructed in a logical sequence.
- Assist in the analysis of software related incidents that have arisen during the test cycle.
- Provide feedback to the test team on any design changes necessary to resolve software related incidents.

Exceptions/Incident Manager

- 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 carried out by the Test Manager).

Quality Assurance/Quality Control

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.

This role:

- Independently reviews and rejects or approves the Test Cases and Test Scripts prior to their execution in order to confirm that they cover all of the required quality objectives of the project (see Test Script Reviewer above)
- Provides one of the review/approval signatures on Test Reports to confirm that stated requirements have been verified from a quality perspective as part of the testing activities.

Project Manager

- Retains overall responsibility for ensuring that all testing activities are included in the project plan by the Test Manager and may provide input into the development of the Test Plan or Strategy.
- Resolution of any escalated issues that have wider implications than just the test team.

Program Manager

- Overall responsibility for managing multiple workstreams on very large projects that may or may not have interdependencies.
- Activities are generally coordinated from a Program Office where Administrative staff will assist in communicating relevant information and obtaining information on the progress, which is then reported to the management level in the organization.
System Owner

- The Representative from the End User Community empowered to make decisions on whether the tested software is of an acceptable standard to be placed into the production Environment.
- The person to whom the Test Report is generally presented for review and sign off, in conjunction with QA.

Internal Auditors

- Internal Auditors are part of the periodic internal audit activity to verify that the testing activities have been performed in accordance with defined procedures.

System Administrator

- Provides an important function to the test team by building any databases necessary in the test environment and also creating the Test ID’s to be used during the testing process in accordance with the profiles provided by the Test Team. May also administer any computerized test management tools.
- Once the testing is underway, they may also be required to assist the test team should problems be encountered during the test (for example, resetting flags on locked records or assisting in the analysis of test errors that have been attributed to database or test tool issues).

Configuration Manager

- Responsible for the establishment of the baseline for the configuration items for the system under test.
- Depending on the overall project structure, this role may also be responsible for creating the software build due to be tested.
- Maintains the baseline for any test datasets received, to enable their reuse if necessary.
- Manages the baselines for test documentation, in accordance with the Document Management Plan for the project.

Data Management Staff

- Provide test datasets with particular attributes in order to generate test data with the correct characteristics to be used during the test.
- This data may require some preparation, so it is important that the requirements for this data are adequately documented and given to Data Management Staff in a timely manner to allow them to locate/prepare the data in advance.

Users (Acceptance Testing)

The users of the system are normally responsible for performing the final user/system acceptance testing and for defining the Test Cases or Test Scripts used for systems acceptance testing.

The users are, in most cases, supported by the test organization (Test Analysts and Testers).

Supplier

Depending on the contract with the Supplier, a number of testing activities have to be carried out by the Supplier both prior and after the installation of the system.

The results of a Supplier assessment can be used to help define the test scope.
Figure T2.1 illustrates the test flow and some of the involved roles. The diagram also indicates which Appendices in this Guide provide further detailed information on some of these processes.
**Figure T2.1: Example of Test Process Flow and Associated Roles**

<table>
<thead>
<tr>
<th>Test Personnel</th>
<th>Example Test Flow</th>
<th>Document Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW Developer</td>
<td>Requirements &amp; Design Specifications</td>
<td>Software Development</td>
</tr>
<tr>
<td>Test Analyst</td>
<td>Test Case &amp; Script Generation</td>
<td>Test Cases &amp; Scripts</td>
</tr>
<tr>
<td>Test Script Reviewer &amp; Approver</td>
<td>Cases &amp; Scripts Reviewed</td>
<td>Test Cases &amp; Scripts Approved</td>
</tr>
<tr>
<td>Testers / Witnesses</td>
<td></td>
<td>Tests Performed &amp; Documented</td>
</tr>
<tr>
<td>Test Result Reviewer</td>
<td></td>
<td>Results Reviewed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Test Results Approved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Approved Test Results &amp; Reports</td>
</tr>
</tbody>
</table>

---

1 Review comment 340

---

For ISPE Members only. Copyright ISPE 2005.
3.2 Training for Test Roles

3.2.1 Purpose of Training

Training the 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. As with any other type of training, it needs to be properly planned to address a real business need, with clear, measurable objectives.

The benefits in ensuring that all test team members receive the correct training in order to perform their roles actually apply across the whole project:

- Trained test staff are more likely to be able to spot potential issues or defects earlier on in the testing life cycle
- Trained test staff are less likely to encounter problems in producing meaningful Test Cases and Test Scripts, thus minimizing the likelihood of tests having to be re-executed due to Test Case and Test Script problems
- Trained test staff are less likely to encounter problems in determining how activities should be completed during the test life cycle
- The number of avoidable human errors generated across the whole range of testing activities should be minimized.
- Evidence of an effective testing process can give the business increased confidence that the final product they will eventually receive will be of a high standard and will be fit for its intended use.

Even if they have undertaken a testing role prior to any new project, it is generally prudent to ensure that all test team members participate, as a minimum, in an induction program which will explain how testing is expected to be carried out on the project, as test strategies and supporting processes can vary greatly from one project to the next.

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

3.2.2 Test Staff Attributes

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

Individuals who have demonstrated a clear aptitude for using computerized systems do not necessarily always have the skills required to become effective test team members. Also, the skills required to participate in the different types of testing within the software development life cycle can vary greatly. Someone who has 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, and vice versa.

3.2.3 Areas Covered by Training

Areas that would be expected to be covered in test team training would include (but are not limited to)

- Project Organization & Structure
- Where testing fits in within the Project/Program
- Interfaces with other non-test team roles key to the success of the testing process
- Any procedures or SOP’s required to be adhered to (including overarching relevant project procedure such as the Validation Plan or Project Quality plan)
- The Project Test Plan or Strategy to be followed
- Introduction to the testing deliverables
- 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
- Any required interface to the project change control mechanism
- Familiarization with project filing structure (both electronic and paper)
- Project contacts for information and guidance

# 4 Test Resources

Resource constraints can often be of key importance in the execution of effective testing. These constraints are wider than the human resource elements of the test team, and cover ensuring that the team has adequate space and facilities in which to work and access to suitable equipment to undertake the type of testing required. It is recommended that the following factors be considered:

## 4.1 Staff Constraints

- Test planning should ensure that the resource requirements for testing are identified early in the project life cycle, in addition to identifying the skills set that the testers should have for the particular type of project. This is important because it may take some time to recruit suitable staff to the test team.
- On “in-house” projects the solution is often to provide staff from other areas of the business, and whilst these staff should be fully conversant with the company’s operation, they may not necessarily have the IT experience and skills to support a testing role.
- Another staffing option may be to recruit external resources to work within the test team. Whilst these resources may be recruited with the necessary IT skills, the fact that they may move employment from one industry to the next, and may be unlikely to be familiar with the business processes and regulations within the life sciences industry mean that they may take some time to acquire this knowledge.
- Budgetary constraints are often a factor for consideration in selecting how a test team should be staffed. Another factor that needs to be considered is the resource profile across the whole project. It is desirable from a financial point of view to keep project resources as utilized as possible without over-allocating them, so consideration may be given to identifying other project resources who have the correct skills required to fulfill testing roles (for example Business Analysts, Deployment Staff and Training Staff)
- All project resources should be given terms of reference for their roles to ensure that there is no misunderstanding regarding the type of tasks they will be expected to perform and to outline immediate line reporting arrangements.
- Testers should not be expected to participate in any testing program along with any other full time job role. Test Managers need to ensure that where members of staff have been seconded to the test team for a period of time, they do not come under any undue pressure from their original role unless there is an urgent business emergency.

## 4.2 Location Constraints

- It is often preferable to locate the test team all in the same place, but it is acknowledged that this is not always possible, especially where the team is large or needs to be geographically dispersed in order to be able to perform the testing objective (for example, verifying that global interfaces work).
Where the operation of the test team involves resources from subcontractors, their work is often conducted at their own premises, which may be some distance from the location of the main test team. This can sometimes present further difficulties, so management and communication with these resources should be carefully considered during the planning phase and agreed with all parties to ensure that no confusion arises once the testing is underway.

If members of staff are expected to work extended hours, adequate arrangements may need to be made for catering facilities, security access, travel arrangements, etc.

4.3 Equipment

The equipment selected to support the testing process should be suitable for the type of testing being planned. Where performance testing is to be undertaken, the sizing and response times of equipment is of particular importance if misleading test results are to be avoided.

Where new equipment is being purchased to create a test environment, orders should be placed with approved Suppliers far enough in advance for timely delivery.

The equipment used should be agreed with the User as being an acceptable replication of the characteristics of the intended production environment. If there are any deviations, these should be noted and the likely impact of the deviation assessed prior to any testing taking place.

Budgetary restriction may mean that the hardware and operating system needs to be shared between different test teams (sometimes running entirely different applications) so there may need to be agreement between these teams to make certain that usage is coordinated to ensure:

- Teams should not be scheduled to use the same equipment at the same time and
- If new developments require a change to be made to the test equipment (for example, a need to install a later version of an operating system) it should be communicated in a timely manner to all affected parties and managed effectively.

Consideration also should be given to the location of peripheral equipment such as printers, etc. Valuable testing time can often be lost walking from one side of the facility to another several times a day to collect printouts necessary to provide objective test evidence.

4.4 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 to a consistent level across the organization.

There are advantages and disadvantages to an organization maintaining a dedicated test team.

4.4.1 Advantages of Maintaining a Dedicated Test Team

- The 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 team has the ability to define and implement good practices (through a central group rather than multiple teams). For example, staff will know how to develop effective Test Cases and Test Scripts; they will know how to record incidents, and so on.
- A dedicated test team can provide consistency delivery of testing services which is sometimes difficult to establish where it is necessary to form new test teams from different backgrounds for every new project.
- Having a dedicated test team can minimize any potential disruption with staff having to cover more than one role at once.
- If the project pipeline is robust, utilization of resources can be high - the ‘pool resource’ mentality allows organizations to maximize the use of the dedicated 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 is often achievable, rather than multiple team managers.
• Better accountability/measurability can be achieved – thus allowing the capture of metrics for a whole stream of work (testing) to help management to determine what value the team is delivering.
• It would make it easier to outsource testing activities should this be the organizations longer term strategy.

4.4.2 Disadvantages of Maintaining a Dedicated Test Team
• The potential and ability to find specific knowledge for the testing of a particular system is reduced.
• Staff brought together to execute testing from different backgrounds or with different skills might not always understand the specific requirements and business processes being tested.
• Professional IT testers may not necessarily be able to identify key features in new software, e.g., they might not appreciate the usability aspects of particular screens if they only use them once or twice, whereas a business user who might reasonably use the input screen perhaps a hundred times during their shift might comment on features which would be less practical for the user.
• If the project pipeline is not particularly strong, test resources may become under utilized, which may not be cost efficient in the long run.
• Test team expertise is lost to the operational unit at the end of testing, making for more difficult system maintenance and on-going regression testing.
• Use of a separate test team may lead to Test Cases and Test Scripts that can only be understood within the team and are, therefore, not re-useable in the operational environment (for example during regression tests).
• Where an external (contract) test team is used there may be issues over accountability.

5 Test Metrics
Test Metrics provide various measures of testing. These allow the test process to be assessed and refined in a managed manner, leading to more effective testing. Testing often takes a large part of the software development life-cycle and, therefore, making it as efficient as possible is important for good management. In the Quality Management cycle of “Plan, Do, Check, Act”, test metrics provide a check on the effectiveness of testing.

Test metrics provide information that can justify, refine and/or improve the extent and type of testing used in the organization and, in certain circumstances, to define that a system is of an acceptable quality to be released (by comparison to testing benchmarks such as software defects detected per test cycle). The metrics provide information that is fed into risk assessments and allow managers to manage the testing process.

Managers should decide what aspects of the testing life cycle they would like to control and then look for suitable metrics for those features. These may include:
• Test coverage (i.e., percentage of requirements for which test cases exist)
• Error detection rate (i.e., number of defects discovered in each phase of testing)
• Residual errors (i.e., number of defects identified in the production environment but not discovered by testing)

The use of appropriate test metrics may provide input to subsequent risk assessments to justify a reduction in the nature and rigor of testing. This may be because testing confirms a low risk likelihood and/or a high probability of detection.
Appendix T3 - Test Specifications or Protocols, Cases and Scripts

1 Test Specifications or Test Protocols

For a small project, a single document may combine the Test Plan or Strategy, Test Specification or Protocol, Test Cases and Test Scripts. On a larger project it is often appropriate to have separate documents. In this case, the Test Plan or Strategy document will normally cover the ‘high level’ approach which applies to all of the test phases:

- Test Management (test phases required, responsibilities, coverage, resources, training, change control, etc.)
- Common templates to be used, common procedures for test execution and recording of results
- Naming conventions (e.g., format of test references and incident references)
- Reviewing of results
- Test Metrics (and, where applicable, their use in determining acceptable quality for release)
- Test Reporting and System Handover

A separate Test Specification or Protocol may then be written for each phase of testing. This will cover the detail relevant to the test phase or test cycle. Typical contents for such a Test Specification or Protocol are contained in GAMP® 4 Appendix D6 (see Appendix G2, reference 1).

2 Test Cases & Test Scripts

A Test Case is defined as 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.

A Test Script specifies the sequence of actions required for the execution of one or more Test Cases. It may include the data to be recorded or captured or this may be included in the Test Case.

In many situations, the contents of a Test Case and Test Script are combined. In this case the Test Script needs to stand alone – giving details of the test inputs and expected results in addition to the actions to be carried out and the data to be recorded.

A useful technique which is recommended in order to establish the appropriateness of Test Cases and Test Scripts is for an independent person (i.e., not the author) to ‘dry run’ the test prior to finalizing the testing documentation and before the final test execution.

2.1 Determination of Test Cases

Realistic test planning is characterized by the need to select appropriate Test Cases from a large set of possibilities. It is appropriate to take into account any risk assessment that has been performed, since this should highlight areas of criticality in the system or areas of potential failure within the system. Different Test Cases should be selected depending upon the level and type of testing that is being planned (e.g., Unit or Integration, Challenge or Performance). Thus Test Cases should be selected which exercise the system in its critical areas and in areas where a potential failure has been identified.

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), and people (training and experience).
• Understand any testing prerequisites that need to be met prior to testing and any interdependencies between tests.

• Ensure that all input references from source document(s) 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 that it has acceptance criteria that clearly demonstrate correct operation.

Test Cases should be traceable back to the requirement in the controlling Specification. This can either be documented in the Test Case itself or via a separate requirements traceability matrix (or equivalent).

Although the test objective may be stated in terms relative to requirements, the acceptance criteria for the Test Case should be clearly defined such that the overall pass/fail determination is not subjective, i.e., the output from the Test Case can clearly be compared against the acceptance criteria in order to determine the pass/fail status. This is aided by defining Requirements that are unambiguous and testable.

It is possible that, because of Test Script or test execution errors, the expected results of an individual test step may not be fulfilled, but that it is still possible to determine that the overall acceptance criteria has been met.

2.2 Design of Test Cases

The following techniques can be of assistance in designing effective Test Cases.

2.2.1 Equivalence Class Analysis

Using the concept of equivalence classes, it is assumed that any input value from an equivalent range will just as effectively test the program as any other value from the range. Consequently, there is no need to input hundreds of values from a range in the same equivalence class when a single value can do the job.

Typical equivalence class assumptions are as follows:

Table T3.1: Equivalence Class Assumptions for Structural Testing

<table>
<thead>
<tr>
<th>Coverage Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement Coverage</td>
<td>Input combinations which result in execution of a particular line of code are assumed to be equivalent.</td>
</tr>
<tr>
<td>Decision (Branch) Coverage</td>
<td>Input combinations which result in the same program branch or decision are assumed to be equivalent.</td>
</tr>
<tr>
<td>Condition Coverage</td>
<td>Input combinations which result in the same outcome for a particular condition within a decision are assumed to be equivalent.</td>
</tr>
<tr>
<td>Multi-Condition Coverage</td>
<td>Input combinations which result in the same outcome for all conditions within a decision are assumed to be equivalent.</td>
</tr>
<tr>
<td>Loop Coverage</td>
<td>Input combinations which result in execution of the same program loop are assumed to be equivalent.</td>
</tr>
<tr>
<td>Path Coverage</td>
<td>Input combinations which result in exactly the same path through a program segment from start to finish are assumed to be equivalent.</td>
</tr>
<tr>
<td>Data Flow Coverage</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table T3.2: Equivalence Class Assumptions for Functional Testing

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

2.2.2 Analysis of Time Dependencies

Where input combinations are required as part of a Test Case, there can be circumstances in which the order in which inputs occur may affect the program action (sometimes referred to as ‘race conditions’).

The following may be worth considering when designing Test Cases:

Table T3.3: Test Case Design Considerations

<table>
<thead>
<tr>
<th>Consideration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous inputs to a decision</td>
<td>Where inputs to a decision can occur asynchronously, Test Cases may need to include for order of inputs as well as input combinations.</td>
</tr>
<tr>
<td>Handshaking between modules</td>
<td>Where data is passed from one module to another, Test Cases may need to include late or early availability of data.</td>
</tr>
<tr>
<td>Transitions out of a state</td>
<td>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.</td>
</tr>
<tr>
<td>Modules updating the same data structures</td>
<td>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.</td>
</tr>
<tr>
<td>Modules locking data structures</td>
<td>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.</td>
</tr>
</tbody>
</table>
2.3 Design of Test Scripts

Test Scripts can be written in such a way that results are recorded directly onto a copy of the approved Test Script or they can be written such that data is recorded onto a separate test results sheet.

Where results are to be recorded directly onto the Test Script, sufficient space should be left for recording the test results, for the tester to sign and date the test execution and for the reviewer to sign. Space also should be left for the run number (or iteration number) since if further run(s) of a test are required following a test incident then a copy of the master Test Script should be made and clearly marked with the run number.

Where separate test results sheets are used a clear cross reference to the Test Case and/or Test Script should be included and space should be left to record the test run number and for the tester to sign and date the test execution and for the reviewer to sign. There is no need to conduct a pre execution review of separate test results sheets where the Test Case and/or Test Script contain all of the important test information.

Test Scripts should be traceable back to the requirement in the controlling Specification. This can either be documented in the Test Script itself, via a referenced Test case or via a separate requirements traceability matrix (or equivalent).

A Test Script should contain the following:

- Unique test reference *
- Test title or description *
- Objective of the test *
- Acceptance criteria to demonstrate that the objective has been met *
- Controlling Specification reference *
- Pre-requisites specific to the test * (for example cautionary measures, test constraints or entry criteria following the end of a previous test - there is no need to repeat pre-requisites for the whole test phase on each test script)
- Pre-test steps * (for example, a test step may either be a non-proving step (i.e., test or data set-up step, a navigation only step) or a proving step (i.e., a step which explicitly demonstrates part of the test objective). Test steps that are not proving the test objective and do not require evidence to be taken (e.g., purely navigational steps) should be clearly identified as 'Non-Proving Step' or 'NPS' in the expected results and actual results columns.) See table below
- Test instructions (step-by-step description of the actions to be performed by the testers). Repeatable tests may be achieved by considering the following points:
  - Where data is required use variable instructions, if possible; e.g., rather than "Enter 29-Feb-04" write "Enter today's date" or "Enter today's date plus 7 days"
  - If the creation of new data is not required wording such as "Ensure the following data exists" or "find/create" gives greater flexibility. For example, a material created according to the characteristics detailed in the Test Case for the first iteration could be re-used in any further executions, as opposed to creating another material.
  - Keep individual objectives to a manageable size. A 100-step Test Script might have 5 logical sections. By making these sections objectives, if only one objective needs re-executing then the number of steps to be re-executed would be greatly reduced.
- Expected results * (not required for Non Proving Steps). These should detail the content of any required screen shots and/or printouts.
- Data to be recorded by the tester * (which can include inputs as well as outputs if these are not pre-defined by the test case or test script; should include serial numbers of test equipment used and copies of supporting calibration certificates if necessary). Where useful, screen shots and/or printouts may be used to reduce the level of effort involved in manually recording large volumes of data. The data to be recorded should be such that an independent reviewer can compare the documented test objective against the test result and objectively determine whether the test met this objective.
• Space for run number * † (i.e., the number of times the test has been run, if it has been repeated following a test incident).
• Space for actual results if they are to be recorded on the test script†. This should include a pass/fail statement for each step, even for a non-proving step (since abnormal behavior will still need to be documented as a failure). Where it is anticipated that the test run will involve more than one test person, additional space may be needed for tester initials against each step.
• Post-test actions * † (actions to return the system to a known state if this is necessary after the test).
• Space to record the overall test result (e.g., Pass, Fail, Conditional Pass, Not Run) together with signatures and dates of the tester and reviewer. * †
• Space to record incident numbers to forward track resolution of issues/deviations encountered during testing. * †
• Space for Comments * †

* indicates that these items may be included in a specific Test Case, where the Test Script may be used with one or more Test Cases.
† indicates that these items can be recorded on a separate test results or test progress sheet.

The following table describes the content that should be included in different Test Script steps:

• 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 T3.4: Test Script Content for Different Test Step Types

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

*N/A = Not Applicable, NPS = Non-Proving Step

Each Test Script should meet the SMART rule insomuch that a Test Script should be:

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

### 2.4 Pre Execution Review of Test Documentation

Test Specifications or Protocols, Test Cases and Test Scripts should be reviewed and approved prior to the start of testing. The reviewer(s) of the test documentation should not be the author, and should represent appropriate functions as defined by company test policy.

It may be helpful to create a common 'review checklist' template to assist in reviewing Test Specifications or Protocols, Test Cases and Test Scripts. Whether or not a template is used, the review process should be consistent and the outcome of the review process should be recorded against each Test Case or Test Script. Issues with test documentation leading to test incidents should have an incident report raised to control any necessary changes and manage any necessary additional testing.
The following four figures show simple examples of Test Cases and Test Scripts. These provide the option of recording results directly on a copy of the Test Script or using a separate Test Results Sheet.
**Figure T3.1: Test Script and Test Case Combined, Integral Test Results**

<table>
<thead>
<tr>
<th>Test Script Reference</th>
<th>Data-Batch-001</th>
<th>Title</th>
<th>Test Script for Batch Number Input Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
<td></td>
<td></td>
<td>To show that the Batch Number Input Transaction works successfully</td>
</tr>
<tr>
<td><strong>Acceptance Criteria</strong></td>
<td></td>
<td></td>
<td>Valid batch number (positive integer) can be entered. Invalid batch numbers are rejected.</td>
</tr>
<tr>
<td><strong>Run Number</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Start Date &amp; Time</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>End Date &amp; Time</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Pre-Requisites:**
Log on available with batch number input transaction capability. There are no specific data requirements

<table>
<thead>
<tr>
<th>Test Step</th>
<th>Requirement Reference</th>
<th>Action</th>
<th>Data to be Recorded</th>
<th>Expected Result</th>
<th>Actual Result / Incident Record</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>N/A</td>
<td>Log on and Enter Batch Number Input Transaction</td>
<td>Log on used</td>
<td>Batch Number Input Transaction successfully entered</td>
<td>(NPS)</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>SR_001</td>
<td>Enter 0 (zero)</td>
<td>Screen print Exhibit 1.2</td>
<td>Rejected with error message</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>SR_001</td>
<td>Enter negative integer (use -123)</td>
<td>Screen print Exhibit 1.3</td>
<td>Rejected with error message</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>SR_001</td>
<td>Enter non-integer (use 12.3)</td>
<td>Screen print Exhibit 1.4</td>
<td>Rejected with error message</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>SR_001</td>
<td>Enter non-numeric value (use Abc)</td>
<td>Screen print Exhibit 1.5</td>
<td>Rejected with error message</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td>SR_001</td>
<td>Enter valid batch number (use 123)</td>
<td>Screen print Exhibit 1.6</td>
<td>Value Accepted</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

**Comments:**

<table>
<thead>
<tr>
<th>Test Outcome</th>
<th>Incident Report Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tester Signature</td>
<td>Date</td>
</tr>
</tbody>
</table>

For ISPE Members only. Copyright ISPE 2005.
**Figure T3.2: Test Script and Test Case Combined, Separate Results Sheet**

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

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

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

**Test Script Reference** | Data-Batch-001 | Run Number |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Start Date &amp; Time</strong></td>
<td></td>
<td><strong>End Date &amp; Time</strong></td>
</tr>
<tr>
<td><strong>Test Step</strong></td>
<td>Data to be Recorded</td>
<td>Actual Result / Incident Record</td>
</tr>
<tr>
<td>1.1</td>
<td>Log on used</td>
<td>(NPS)</td>
</tr>
<tr>
<td>1.2</td>
<td>Screen print Exhibit 1.2</td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>Screen print Exhibit 1.3</td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>Screen print Exhibit 1.4</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>Screen print Exhibit 1.5</td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td>Screen print Exhibit 1.6</td>
<td></td>
</tr>
</tbody>
</table>

**Comments**

**Test Outcome** | Incident Report Reference |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tester Signature</strong></td>
<td>Date</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Figure T3.3: Separate Test Script and Test Case, Integral Test Results**

<table>
<thead>
<tr>
<th>Test Script Reference</th>
<th>Title</th>
<th>Objective</th>
<th>Acceptance Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-Batch-001</td>
<td></td>
<td>To show that the Batch Number Input Transaction works successfully</td>
<td>Valid batch number (positive integer) can be entered. Invalid batch numbers are rejected.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Run Number</th>
<th>Start Date &amp; Time</th>
<th>End Date &amp; Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Pre-Requisites:**
Log on available with batch number input transaction capability. There are no specific data requirements.

<table>
<thead>
<tr>
<th>Test Step</th>
<th>Requirement Reference</th>
<th>Action</th>
<th>Data to be Recorded</th>
<th>Expected Result</th>
<th>Actual Result / Incident Record</th>
<th>Pass/ Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>N/A</td>
<td>Log on and Enter Batch Number Input Transaction</td>
<td>Log on used</td>
<td>Batch Number Input Transaction successfully entered</td>
<td>(NPS)</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>SR_001</td>
<td>Enter batch number using test cases 1 to 5</td>
<td>Screen print Exhibit 1.2.n where n = test case number</td>
<td>As detailed in test case definition</td>
<td>Record on test case definition</td>
<td></td>
</tr>
</tbody>
</table>

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

**Comments:**

<table>
<thead>
<tr>
<th>Test Outcome</th>
<th>Incident Report Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tester Signature</th>
<th>Date</th>
<th>Review Signature</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TEST CASE DEFINITION**

<table>
<thead>
<tr>
<th>Test Case Reference</th>
<th>Title</th>
<th>Description</th>
<th>Characteristics</th>
<th>Expected Result</th>
<th>Actual Result / Incident Record</th>
<th>Pass/ Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-Batch-001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Description</th>
<th>Characteristics</th>
<th>Expected Result</th>
<th>Actual Result / Incident Record</th>
<th>Pass/ Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Batch number of zero</td>
<td>0</td>
<td>Rejected with error message</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Negative integer batch number</td>
<td>-123</td>
<td>Rejected with error message</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Non-integer batch number</td>
<td>12.3</td>
<td>Rejected with error message</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Non-numeric batch number</td>
<td>Abc</td>
<td>Rejected with error message</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Valid batch number</td>
<td>123</td>
<td>Value Accepted</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure T3.4: Separate Test Script and Test Case, Separate Results Sheet

### Test Script Reference

<table>
<thead>
<tr>
<th>Test Script Reference</th>
<th>Data-Batch-001</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Script for Batch Number Input Transaction</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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 Case Definition

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Description</th>
<th>Characteristics</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Batch number of zero</td>
<td>0</td>
<td>Rejected with error message</td>
</tr>
<tr>
<td>2</td>
<td>Negative integer batch number</td>
<td>-123</td>
<td>Rejected with error message</td>
</tr>
<tr>
<td>3</td>
<td>Non-integer batch number</td>
<td>12.3</td>
<td>Rejected with error message</td>
</tr>
<tr>
<td>4</td>
<td>Non-numeric batch number</td>
<td>Abc</td>
<td>Rejected with error message</td>
</tr>
<tr>
<td>5</td>
<td>Valid batch number</td>
<td>123</td>
<td>Value Accepted</td>
</tr>
</tbody>
</table>

### Test Steps

<table>
<thead>
<tr>
<th>Test Step</th>
<th>Requirement Reference</th>
<th>Action</th>
<th>Data to be Recorded</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>N/A</td>
<td>Log on and Enter Batch Number Input Transaction</td>
<td>Log on used</td>
<td>Batch Number Input Transaction successfully entered</td>
</tr>
<tr>
<td>1.2</td>
<td>SR-001</td>
<td>Enter batch number using test cases 1 to 5</td>
<td>Screen print Exhibit 1.2.n where n=test case</td>
<td>As detailed in test case definition</td>
</tr>
</tbody>
</table>

### Post Test Actions:

Abort the batch ready for the start of the next test.

---

For ISPE Members only. Copyright ISPE 2005.
Appendix T4 - Test Environments

1 Introduction

The environment in which testing is conducted should be determined based on the stage of the life cycle and the output of the risk assessment. The following general principles apply:

- The test environment should be made as representative as possible of the production environment. Differences between the proposed test environment and production environment should be detailed in the Test Specification or Protocol and should be subject to an impact assessment. If necessary, additional tests should be planned for the production environment in order to cover additional risk scenarios identified.

- The test environment should be controlled and recorded to a level of detail that would allow it to be reconstructed if necessary. Such control includes: system hardware and software components; test hardware (versions, serial numbers as appropriate); test software (version control of any simulations); test data (version control of any test data sets, test recipes, etc.); and test user accounts.

- Where test hardware/software/data/user accounts are applied in such a way that they may appear in the production environment, controls should exist to ensure that they can either be removed cleanly or be isolated from use (either logically or in time).

- 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 need to be formally documented.

There may be many reasons why it is undesirable to conduct all testing on the final 'as installed' production environment. Common examples include:

- Non-availability of infrastructure (for example communications networks) at the point in the project life cycle at which the testing is carried out.

- Non-availability of certain interfaces (for example because items are being produced by different Suppliers).

- Requirement to test changes outside of the production environment prior to installation (for example because the data produced as part of the test needs to be isolated from the production data).

- Requirement to carry out tests which might be destructive to the production environment (for example stress tests on the infrastructure/platform such as Denial of Service, Message overflows or virus attacks).

The progression of a software build from a development environment through to a production environment depends on the risk priority 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.

It might, for example, be acceptable to apply a change of parameter (e.g.: PID tuning or alarm threshold) directly to the production environment (a process controller) because the change is 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 might, however, require progression from a development environment, to a test environment, to a validation environment and then to the production environment. This may be because the change is a high risk priority and even if the original module could be restored easily, the data from the test may remain in the production environment.

There are some tests, e.g.: the Performance Qualification (or part of it), which may need to be conducted in the production environment.

1.1 Hardware Environment

Hardware can be categorized according to both its GAMP 4 hardware category (standard or custom) and its function within the test environment. It may be:

- Part of the system under test i.e., part of the production environment hardware,
• Test hardware representing part of the production environment, which may be needed because it is not feasible to include a certain element of the production system in the test environment,

• A separate test system which may be used to represent an external system.

Some examples are provided below:

Table T4.1: Examples of Hardware Test Environments

<table>
<thead>
<tr>
<th>GAMP® Hardware Category</th>
<th>Function within the Test Environment</th>
<th>Part Of System Under Test</th>
<th>Representing Production Environment</th>
<th>Separate Test System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - Standard</td>
<td></td>
<td>Standard component (e.g., PC) undergoing testing prior to transfer into the production environment.</td>
<td>Network hub and cabling used to connect together equipment for test.</td>
<td>Signal injection/measuring equipment.</td>
</tr>
<tr>
<td>2 - Custom</td>
<td></td>
<td>Custom component undergoing testing prior to transfer into the production environment.</td>
<td>Test environment copy of a custom component that is already in use in the production environment.</td>
<td>Custom built hardware simulator.</td>
</tr>
</tbody>
</table>

1.1.1 Representative Test Environment

As previously stated, the hardware environment should be as representative as possible of the production environment.

For example, if the test environment uses a standard network hub of the same type as the production environment then the substitution probably introduces very little likelihood of the test results being invalid in the production environment. If, however, the network cabling in the test environment uses short patch cables whilst the real environment has cable runs close to the maximum recommended length, there is clearly a possibility of different network behavior and additional tests on site may be needed to prove the network performance.

1.1.2 Control of Test Environment

For standard (GAMP 4 hardware category 1) hardware, the manufacturer’s reference number and the serial number should be recorded.

For custom (GAMP 4 hardware category 2) hardware, the version of the item and its controlling Specification also need to be recorded.

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

1.1.3 Removal from Production Environment

If test hardware is added in such as way that it may appear in the production environment, then this should be documented as a temporary modification to the production system. Removal of the temporary modification should be documented as well.

1.2 Test Software

Test software can also be categorized according to both its GAMP 4 software category and its function within the test environment (part of the system under test, test software representing part of the production environment or a separate system). Some examples are provided below:

Table T4.2: Examples of Software Test Environments
<table>
<thead>
<tr>
<th>GAMP® Software Category</th>
<th>Type</th>
<th>Part Of System Under Test</th>
<th>Representing Production Environment</th>
<th>Separate Test System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Operating System</td>
<td>Operating system running on a PC which is part of the production environment</td>
<td>Operating system running on test environment PC</td>
<td>Operating system of PC used to represent data collection PC to which files will be archived in production environment</td>
</tr>
<tr>
<td>2</td>
<td>Firmware</td>
<td>Firmware running on a controller which is part of the production environment</td>
<td>Firmware running on test environment controller</td>
<td>Firmware running on test ‘slave’ instrument used to prove correct operation of a Modbus link</td>
</tr>
<tr>
<td>3</td>
<td>Standard Software Package</td>
<td>Standard software package running on a PC which is part of the production environment</td>
<td>Standard software package running on test environment PC</td>
<td>Standard data collection package used to collect test data</td>
</tr>
<tr>
<td>4</td>
<td>Configurable Software Package</td>
<td>Configured module under test prior to transfer into the production environment</td>
<td>Test environment copy of a configured module that is already in use in the production environment.</td>
<td>Simulation package configured to provide feedback to imitate valve position</td>
</tr>
<tr>
<td>5</td>
<td>Custom (Bespoke) Software</td>
<td>Custom module under test prior to transfer into the production environment</td>
<td>Test environment copy of a custom module that is already in use in the production environment.</td>
<td>Custom coded simulation used to represent temperature and pressure conditions in a vessel</td>
</tr>
</tbody>
</table>

### 1.2.1 Representative Test Environment

The software environment should be as representative as possible of the production environment.

For example, if the test environment uses a process controller running the same firmware version as the production environment then the substitution probably introduces very little likelihood of the test results being invalid in the production environment. If, however, a particular interface is simulated in the test environment, there remains a possibility that different timing factors or process dynamics could affect operation in the production environment and additional testing of the production interface may be appropriate.

### 1.2.2 Control of Test Environment

For standard (GAMP 4 software category 1, 2, or 3) software, 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 (GAMP® 4 software category 4 or 5) software, the item should be placed under configuration management and the version in use recorded.
1.2.3 Removal from Production Environment
If test software is added in such a way that it may appear in the production environment, then this should be documented as a temporary modification to the production system. Removal of the temporary modification also should be documented.

1.3 Test Data Sets
Test data sets are often 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 generic enough to cover certain test types (i.e., challenge testing at boundary conditions or stress testing).

1.3.1 Representative Test Environment
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).
Differences between the proposed test data and the expected actual data should be detailed in the Test Specification or Protocol and should be subject to an impact assessment. If necessary, additional tests should be planned for the production environment in order to cover risk scenarios identified.
For example, if injection of test data is not possible and the test environment has gathered only a few weeks of data while the production environment may eventually accumulate 10 years of data, there remains a possibility that different timing factors or system dynamics could affect operation in the production environment and regular performance monitoring of the production environment may be appropriate.

1.3.2 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 control the utility used for the generation of data (as well as the test data set).

1.3.3 Removal from Production Environment
If test data is added in such a way that it may appear in the production environment, then this should be documented as a temporary modification to the production system. Removal of the temporary modification also should be documented.
If the production environment possesses automatic audit trailing, then it should be recognized that audit trail entries from the testing process will remain.

1.4 Test User Accounts
Test user accounts are often used to permit testers to access the system at many different levels and to ensure that activities carried out during testing are easily identified within any resulting audit trail.

1.4.1 Representative Test Environment
Where test user accounts are used, these should be set up to represent each group of users within the system, including the corresponding authorizations. For a multi-lingual system, test user accounts using foreign character sets should be included. Similarly, if existing individual accounts are used for testing, representatives from each group of users should be included.

1.4.2 Control of Test Environment
If test user accounts are used then 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.

### 1.4.3 Removal from Production Environment

If test user accounts are added in such a way that they may appear in the production environment, then this should be documented as a temporary modification to the production system. Removal of the temporary modification also should be documented.

### 1.5 Test Documentation

The test environment includes the documentation used during testing. This should always include the test documentation (Test Plans and Strategies, Protocols and Test Specifications, Test Cases and Test Scripts) and the controlling Design Specifications. It may also 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 is frequently found helpful 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.
Appendix T5 - Test Execution

After a test is written, reviewed, perhaps rewritten, and finally approved, it is then ready for execution.

Before commencing tests using individual Test Cases or Test Scripts, any pre-requisites for the test phase should first be verified and recorded. For example:

- Test environment hardware (for example serial numbers and calibration certificates if required), software (for example software baselines), data sets and user accounts (see Appendix T4 of this Guide)
- Personnel involved (for example documentation of names, positions and sample signatures and initials)
- Completion of any other test phases listed as pre-requisites
- Availability of baselined documentation (including, most critically, Test Documentation and procedures)
- Where applicable, calibration of any critical instrument inputs

1 Manual Test Execution

1.1 Test Execution

Tests should be carried out as follows:

- Any pre-requisites for the test should first be checked as above.
- The test is then executed following the test instructions given within the Test Script.
- Each test should be run and the data collected as test results. See Appendix T6 section 2.3.5 of this Guide for further information on test results recording.
- The tester decides whether the acceptance criteria have been met and records whether the test has passed or failed and then signs and dates the test results. Sometimes a third category ‘refer for review’, or ‘conditional pass’; or ‘pass with observation’ can be helpful for cases where the tester feels that an independent opinion is required.
- Supporting documentary evidence required by the Test Case or Test Script should be collated as detailed in Appendix T6 of this Guide.
- Should an incident occur, it should be recorded on a test incident sheet (or within the test incident system) and be retained as part of the test record. The key to dealing with incidents during test script execution is to accurately record the incident and retain sufficient supporting information to help with future problem resolution. Incident logging and management is detailed in Appendix T2 of this Guide.
- It is helpful to maintain a test progress summary onto which overall test results and number of test runs are recorded. Depending on individual company test policy, the summary may be regarded purely as a status and scheduling tool or it may form part of the post-execution review and be included in the Validation Report as a GxP document.
- After completion of all tests or a group of tests (for example, at the end of the day), there should be a review of progress. A review group should assess all tests and incidents.

The following are possible actions for failed tests and incidents:

- Repeat the test
- Apply a change via change control and if necessary repeat the test
- Abandon one, several, or all tests
- Review result and upgrade to a ‘pass’ status (with a record of the rationale for the change in status)

For ISPE Members only. Copyright ISPE 2005.
The review group should decide which course of action to take and what re-testing is required and document the justification for the action(s). Appendix T2 of this Guide outlines some of the common problems encountered during testing and provides further information on test incident management.

2 Automated Test Execution (and Computerized Test Management Tools)

The execution of any testing using automated (computerized) test tools should be conducted in accordance with the Test Plan or Strategy. The Test Plan or Strategy should require that all automated tools used to test GxP systems have been subjected to assessment and appropriate validation prior to their use. This is to establish a high degree of assurance that they will perform as intended, returning accurate and secure results.

Before discussing the benefits and use of automated test tools it is useful to describe the difference between a computerized test management tool and automated testing.

A computerized test management tool facilitates the task of Test Case and Test Script authoring, review and approval (pre and post execution), test execution and test deviation management. This is accomplished through the use of workflow enabled processes, electronic test artifact management (i.e., Test Cases, Test Scripts, Deviation reports) and possibly the use of electronic signatures.

However, even with a computerized test management the actual execution of the test is performed by a tester, even when this task is aided by the presentation of an on-screen test scripts and electronic capture of test evidence. The tester must still manually follow the test steps, make inputs to the system under test, and record the test evidence.

With automated test execution the computerized test tool also executes the actual test and records the test evidence. This can significantly reduce the time to execute tests. Because most automated test execution is based upon initial manual test execution (the tool often ‘records’ the actions of the manual tester) automated test execution of most often used when executing tests during a second test cycle and is well suited to regression testing.

2.1 The Benefits of Computerized Test Management Tools and Automated Test Execution

When planned properly, automating testing activities can often bring considerable benefits within the project life cycle. However, if poorly planned computer based test tools and test automation may be a difficult, complex, and time consuming task with little or no return on investment.

Computerized test management tools can significantly reduce the amount of paper used during testing and can provide helpful test management support. This includes the ability to report on the status of test activities and facilitate test activities by the use of workflow. In most large testing projects, the use of such a tool can reduce testing timescales.

Not every aspect of a computerized system can be automatically tested. A key factor in the successful use of such tools is the early identification of the types of tests that should be automated, and the type of tests that should be executed manually.

However, depending on the type of testing being automated, it is possible that automated testing can provide significant benefits in terms of time saving and the ability to execute tests which are difficult to execute manually (where timing constraints are difficult to meet or where multiple inputs need to be coordinated).

While computerized test management tools and automated testing can both provide benefits, these should be considered separately. Some projects may benefit from the use of computerized test management tools but there may be little or no benefit in the use of automated test tools.

Before a decision is made as to whether computerized test management tools and automated test tools are useful, the following points should be considered:

- Will the tools be used on a single project or more broadly within the organization? Typically a large project or organization wide use is necessary to provide a return on investment
• How many test cycles are typically executed and how much regression testing is required? This will help determine the need for automated testing.

• Will a dedicated test team be using the tool, or will all projects use the tool? This has an impact on on-going administration and training requirements.

Provided the use of these tools has been given adequate consideration early enough in the project life cycle, the amount of elapsed time necessary for retesting or other regression testing can also be significantly reduced. In many instances, an automated test tool can run without any supervision at all. Automated test execution means that there is no intervention after launching the tests. This includes activities like executing the test, capturing relevant results, comparing actual with expected results and reporting analysis of whether the pass or fail criteria have been met. Greater automation, on an ongoing basis, can also significantly reduce the likelihood of test execution errors.

The use of computerized test management and test automation tools needs to be regarded as a software development activity which has a life cycle of its own. Such tools will have their own implementation and validation life cycle. This will place additional requirements on the test team; particularly in terms of the skills required by test script developers.

The use of test management tools and test automation tools is only effective if supported by adequate processes and if staff have the necessary skills to use the tools successfully. Too often people do not have the proper training in how automated test processes operates or are not given the necessary time to come up to speed with the tool set. In some cases the anticipated benefits are not realized and where the wrong test types are selected for automated testing this might take even longer than would manual test execution.

The usability of the test tool is significantly enhanced if it is application-independent (i.e., may be used to test multiple applications). Easy maintenance of a test suite is crucial for a successful implementation of an automated test tool.

Automated test tools which include a version and configuration management tool can improve the effectiveness and efficiency of retesting very significantly. Not only can the subsequent versions of test results be stored by test run or version, but a good version management tool also should be able to locate and display the changes between the results of two versions of each test script. In case of automated unattended regression testing, this would mean that only the tests with different results need to be viewed and analyzed.

### 2.2 Computerized Test Management Tools and Automated Test Tool Features

Different test management and automated testing tools exist within the IT industry and the architecture of the application under test often determines which tool is most suited. Most tools are purchased as Configurable-off-the-Shelf Software Packages with the Test Team then configuring the package locally to meet the particular usage required. Less complex automated tools, such as Test Harnesses, can often be developed within the project team.

Before acquiring computerized test management or automated test tools it is important that an organization understands how it will use such tools (i.e., defines their requirements)

Test tools can be grouped into four general types

#### 2.2.1 Test Management Tools:

General computerized test management tools may:

- Manage testing across the project
- 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,
- Trace tests back to their original requirements
- Summaries the defects found during the testing process
Determining whether changes need to be made to the automated testing tools themselves, as a result of changes to the applications they are used to test.

Note that test management tools do not provide the facility to automate test execution. Other tools exist for automated test execution (often integrated into the test management tools) and the type of automated test execution tool will depend to a large extent on the type of software or application under test and the type of test being executed. These are:

Developer-Orientated Tools:
- Are used in unit, component, and module testing to address issues such as memory leaks or other early performance problems.
- Are used to test specific pieces of the developer’s application code independent of other units of code within the software application.
- May incorporate capture-replay utilities, which run sample tests against working versions of a program, capturing the activity it generates. During the playback phase, developers can see whether or not they are generating the results they expected.
- Are particularly useful in testing large applications where different developers are working on different parts of the application.

Functional Test Tools:
- Test that the code being developed is acting as expected.
- Often contain keyword or table driven execution engines, which allow the development and execution of repeatable tests.
- Often have scripting capabilities, allowing testers to modify what they’ve written to test additional items.
- Can often be linked back to other packages providing Requirements Capture and Tracking capabilities, therefore facilitating comprehensive requirements traceability.

Load & Performance Related Tools:
- Are particularly useful to test what happens to the code when multiple 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 can often drill down to lower levels of the code to find out where bottlenecks exists and what is causing any delays.
- These tools can also be valuable when testing the scalability of the application.

2.2.2 The Use of Computerized Test Management and Automated Test Tools
As stated above, computerized Test Management and Automated Test Tools can prove advantageous on large projects or within an organization. However, the use of such tools will require:
- The use of testing procedures (SOPs or Work Instructions) for the various activities supported by the tools, such as:
  o Test Case/Test Script authoring
  o Test Case/Test Script review and approval
Test execution
- Test result review and approval
- Test defect reporting
- Test defect categorization
- Test defect disposition
- Test defect closure

Note that other activities such as test planning and test status reporting are less critical from a regulatory perspective and will not require formal procedures.

- That users of the test tools are trained in the use of the procedures
- Administration of the test tool, including:
  - The configuration of the tool
  - The creation of test projects
  - Administration of users
  - Support and maintenance activities (backup and restore, capacity planning and performance monitoring, etc.)

When implementing test tools it is important to initially define the test processes that will need to be supported by the tool(s). The test process implemented through the use of such test tools should comply with the testing good practices defined in this Guide and the specific test policies and processes of the owning organization.

Essentially this means defining the requirements for the use of the tool(s). These should be derived from the organizations standard test policies and processes and these should lead to the definition of the test tool procedures and work instructions.

Unless this is done there is a risk that the configuration of the test tool(s) will drive the test processes, which may not comply with the organizations policies and procedures and which may lead to non-compliances. In some cases, failure to adequately define the requirements for test tools leads to a situation where the test processes become over-complex and in some cases the test tool may not work as expected, because of configuration problems.

All users of the test tools should be trained, not only in the use of the test tools but also in good testing practices. Just because a test tool allows a user to author or execute a Test Case or Test Script more easily does not mean that the Test Case or Test Script will capture appropriate test results or adequately test a requirement.

When using test tools clear user roles and responsibilities should be defined with respect to each test process. Appropriate technical or procedural access restrictions should 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.

Where used, computerized test tools and automated test execution tools should provide the same level of test data integrity as an equivalent paper based process.

All of these can be facilitated through the use of features such as:
- Role based authorities and user restrictions,
- The use of workflow to enforce test processes,
- The use of electronic or digital signatures,
- Secure, computer generated audit trails.
2.3 Verification of Computerized Test Management and Automated Test Tools

Because of the need to assure the security, integrity and availability of test data, automated test tools should be appropriately selected and verified (see section 4 in Appendix M4 of GAMP® 4 for further details (see Appendix G2, reference 1)).

Because such tools usually have only an indirect impact on product quality they are considered to be low risk priority and do not require a detailed or lengthy validation process.

The verification of a computerized test management or automated test tool should focus on demonstrating that the tools are fit for purpose as far as critical requirements are concerned. Although such test tools will have a broad range of requirements (see above), the critical requirements which should form the basis of the tool verification are:

- The security of the test data, facilitated by role based authorities and user restrictions, the use of electronic or digital signatures and the use of secure computer generated audit trails. Where the test tools do not provide such features the verification of the tool(s) should focus on the establishment of such security through the use of logical or physical security controls, procedural controls and paper based audit trails.

- The 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 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 track record of the supplier and tools in the life sciences industry (see Key Concepts of this Guide).

Test data managed within test tools would not usually be determined to be an electronic record within the scope of predicate rules and the use of electronic or digital signatures would not usually be determined to be within the scope of the predicate rules, i.e., the requirements of regulations such as 21CFR Part 11 would not usually apply.

However, there may be times where the software or application being tested using such tools is of greater regulatory significance. An example of this would be where the software or application being tested is defined as a medical device or is part of a medical device.

In these cases a more formal validation of the test tool may be required. In such cases the test record may also be determined to be an electronic record (i.e., an acceptance record under 21CFR Part 820 Subpart H – Acceptance activities) and any associated signatures may be determined to be an electronic or digital signature (i.e., 21CFR Part 820.80 [e] [4]). In such a case, the test management tools may need to comply with the requirements of 21CFR Part 11 and other regulator guidance in electronic records and electronic signatures. (See Appendix G2, reference 8.)
Appendix T6 - Test Results Recording and Reviewing

1 Test Evidence

Given the stated preference by regulatory authorities for “documented evidence” of testing and validation activities, the creation, identification, handling and storage of test evidence becomes an area of importance. 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. Either the general Test Strategy or Plan, the phase specific Test Protocol or Specification of the test specific Test Case or Test Script (or a separate procedure) should define how test evidence is dealt with. The paragraphs below offer recommendations for dealing with different types of test evidence.

Test evidence can be manually recorded quantitative data or observations, printouts (reports), screen shots or automatically captured data (e.g., in PDF format from automated test tools).

There are instances where no physical printout or quantitative data is produced by a test, only an observation. The risk-based Test Plan or Strategy should document (before testing commences) how these tests should be dealt with. It may be acceptable to have the Tester record their observations which are then subsequently signed off by a Reviewer; or the Test Manager may deem it necessary to have the test witnessed. If a witness philosophy is used, then the selection of the witness should be given careful consideration.

Whatever the type of evidence used to support the test, positive affirmation of the overall test result should be recorded, along with formal signatures by the Tester, Witness (where necessary) and Reviewer. Note that for some test cases a mixture of paper and electronic evidence may be appropriate (i.e., an electronic screen shot and a hard copy report).

1.1 Paper Evidence

In the case of paper evidence, e.g., printouts or reports, the date (and where appropriate the time) of the test, the test reference and the test run number should be clearly traceable to the tester’s signature. This may be recorded on each page as suggested in GAMP® 4 Appendix D6 (see Appendix G2, reference 1). Alternatively, where each page contains continuous page numbering (i.e., page ‘n’ of ‘m’), test reference and run number, a single signature and date for the whole document may be sufficient if the context of the signature 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 either directly onto a copy of the approved Test Script or onto separate test result sheets. In either case the test run number of the test should be clearly recorded.

For an example test result sheet see section 2 in Appendix T3 of this Guide.

1.2 Electronic Evidence

Where test evidence is retained electronically, the test evidence should be clearly linked to the test run and consideration should be given to electronic audit trailing, and approval and signature of these records. Depending upon the context of the test, such records and signatures may fall within the scope of relevant regulations (e.g., 21CFR Part 820.80 – for medical device acceptance records where the software is part of a medical device).

2 Test Record Integrity

Test records (Test Plans or Strategies, Test Cases and Scripts, Test Results, Test Evidence and Test Reports) provide invaluable evidence to support the validation of the system. Combined with other verification activities, the documented testing of a system or application provides demonstrable proof that system requirements have been met and that the system or application is fit for its intended use.

The integrity of test records should, therefore, be assured by good practices in the management of such records. Sensible guidelines should be established for the management of all test records and over the use of annotations on such test records.
2.1 Paper Records
Where testing is largely based upon paper records this means ensuring that:

- Entries made on test records should be ‘permanent’. Although some change in color or tone may occur over time, entries should still be legible for the lifetime of the document. A minor change in color is not a problem (unless multiple colors are used for different purposes, and it becomes impossible to distinguish between the two).
- The ‘permanence’ may need to be confirmed by inspection over time and if entries deteriorate unexpectedly it may be necessary to make verified copies.
- Changes should have an audit trail. Any corrections should be made in a way that does not obscure previous entries or original data. Where the reason for the change is not obvious this should be explained in appropriate notes.
- It should be possible to make copies of documents, including all entries and annotations. Ideally it should be possible to distinguish between originals and copies (but copies can always be marked as such, for example, by use of rubber stamp).

It is important to focus on these underlying principles rather than establishing ‘rules’ (e.g., “black ink MUST be used”) that may be rendered out-of-date by changes in ‘stationery technology’ (i.e., the introduction of color photocopiers made many such ‘rules’ obsolete).

Figure T6.1: Example of Corrected Test Result

Correct Data: 132.45, PB, Jan 31, 2001 (Numbers entered in the wrong order)
Wrong Data: 123.45

2.2 Electronic Records
The same principles also apply to the use of test records in electronic format, e.g.: records generated by computerized test management or automated test tools. The risk-based validation of such systems should provide a high degree of assurance regarding the integrity of such test records (and in accordance with any relevant regulations).

In some cases, where computerized test management or automated test tools can not provide such a high degree of assurance, a hybrid approach may be more appropriate (i.e., executing tests within the test management tool and printing and signing hard copies of the executed test).

3 Post Execution Review
The purpose of the post execution review is to assure that test procedures have been adhered to correctly and to approve the overall test result. The outcome of this 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 may also be signed by the reviewer.

Such review activities should be conducted using the basic quality assurance precept of “independence of review.” The normal minimum requirement is that testers should not review their own results.

The executed test cases are typically summarized, documented, and reviewed:

- At the end of each cycle of testing (as defined in the Test Specification or Protocol for the phase) or
- For each unit being tested or
- At the end of the test phase.
Test reviewers may identify situations that should have been, or now need to be classified as a test incident. For example, manually altered test scripts, inadequate or insufficient evidence having been collected, actual results deviating from expected results, etc. In this case an incident report should be raised to control any necessary change and manage any necessary additional testing.

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

For the Results of Individual Tests
- That all test result sheets (or Test Case or Script) and captured test evidence is marked with the unique test reference and the run number.
- That test results have been signed and dated by the tester.
- That any appended evidence showing captured results has been signed and dated by the tester (see ‘paper evidence’ above).
- That test results are 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 these criteria. This requires the Tester to record an actual result, not just ‘as per expected result’ or similar.
- That any failures have been logged and raised as test incidents.

For the Test Phase as a whole:
- That 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 Strategy.
- That test incidents have been logged, classified, reviewed, and, where required by the Test Strategy or Test Plan, resolved prior to handover of the system.
- That software defect data that is collected and analyzed during a development life cycle indicates the suitability of the software product for release for commercial distribution (where required by the Test Strategy or Test Plan).
- That suitable regression analysis has been performed and agreed regression testing has been completed following resolution of any test incidents (where required by the Test Strategy or Test Plan).
- Test completion criteria have been attained - in which case the test phase/cycle is deemed complete.
- Test reports comply with the requirements of the overall Test Plan or Strategy.

For further details on Test Phase Reports see Appendix T7 of this Guide.
Appendix T7 - Test Reporting and System Handover

1 Test Reports

A test report is typically prepared at the end of each test phase. Test reports normally contain:

- Introduction
- Scope of testing
- Organization of testing (including confirmation that the test environment was suitably controlled and documented)
- Details of who performed and independently reviewed the testing
- Summary of test results in tabular form
- Summary of test incidents (including known issues, outstanding incidents and all restrictions on use)
- Conclusions, approval of the test report and, where applicable, authority to move to the next phase in the life cycle

The reviewer(s) and approver(s) of the Test Report should not be the author or tester(s), and should represent appropriate functions as defined by company test policy.

Where the necessary information already exists in suitable format within the test results (for example: documented test environment; test results tabulated on test progress sheets; indexes of test incidents; summaries of personnel involved with sample signatures) these can be cross-referenced rather than repeated in the report.

2 Formal Handover and Release

There may be several milestones during the testing process at which the system is formally handed over or released from one group to another. These may be:

- Handover from application development environment to test environment
- Handover from one test environment to another (e.g., Factory Acceptance to Site Acceptance)
- Handover from test environment to validation environment,
- Release from validation environment to production environment

Handover from Supplier to User usually coincides with one of these milestones and forms a special case only because of the contractual implications of the release (for example stage payments). In other respects, all handovers and releases require similar considerations.

2.1 Handover of System

It is a general principle that the system should be handed over in a controlled state. Such control includes:

- System hardware (versions, serial numbers as appropriate)
- Application software (versions, licenses, any known defects, any work-arounds which are in place, any pending change requests)
- Test modifications (where test hardware/software/data/user accounts have been applied to the system, controls should exist to ensure that they have been either removed cleanly or isolated from use)
• Dependencies (where movement from one environment to another involves loading of software into the new environment, it is important that any dependencies – between software item versions or on particular hardware/software platforms are documented).

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 (for example to ensure that the software at the start of site testing is the same as that released at the end of factory testing), or that any changes made between test phases are traceable to agreed change requests or test incident reports.

2.2 Handover of Responsibilities
Transfer of responsibilities needs to be made alongside transfer of the system itself. In particular, consideration should be given to the following:

• Responsibility for change control
• Responsibility for configuration management (and build management)
• Responsibility for documentation management
• Responsibility for system administration

2.3 Conditional Release
The method for system release 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 production environment (performance tests, for example)
• When workarounds have been established

Conditional release should include the use of a documented risk assessment and putting into effect any risk mitigation that is required.

Where a conditional release is agreed, the method and timescales for closing the outstanding actions needs to be agreed as part of the release process.
Appendix T8 - Testing in the Operational Phase

1 Types of Change

This section deals with the testing and maintenance of a system that has previously been put into service (released for production use), but which now requires a software upgrade, patch, or change of use.

1.1 Supplier Driven Change

A Supplier driven change would typically be made as a result of the identification and rectification of a defect in the Supplier product or the introduction of new product features. In this instance, the onus is on the Supplier to provide detailed documentation on the scope of the release, any corrective actions taken, and the regression testing performed prior to release.

Mature Suppliers to the pharmaceutical and associated industries should determine the impact on these Users and may include regression testing of general industry requirements such as data integrity and known critical functionality.

The User should perform an impact assessment to determine:

- Whether the defect impacts on the User business process i.e., whether to adopt the change.
- Any associated risk scenarios due to not adopting the change.
- The technical or business impact of the change.
- The extent to which the Supplier’s own testing can be relied upon.
- The effect of any new functionality included in the new release but not initially required by the User. Where the User chooses to use the new function they should update their Requirements to include such new functionality. Where the User chooses not to use the new functions a plan will be required to either disable the functionality or to prevent its use by some other means e.g., SOPs and training.
- The scope of supplementary and/or regression testing required to release the system back into production.

1.2 User Driven Change

Changes of this type are driven by the user group in response to business issues. Examples include:

- A change in business requirements
- A change resulting from a support call, e.g., print configuration issue
- An internal continuous improvement policy resulting in a change of use
- A performance issue e.g., poor performance in high load situations giving impaired response times
- A peripheral change which impacts a particular system, e.g., decommissioning of an associated system
- A change required as a result of regulatory change.

Depending on the type of change, the change may be handled internally by the user group, by a dedicated support group or by Supplier involvement.

The impact assessment for a user driven change should consider:

- Potential benefits (business advantage)
- Documentation and training burden
• Fit with company strategy
• How the change can be implemented (how/where resourced)
• The technical and business impact of implementing the change
• The scope of supplementary and/or regression testing required to release the system back into production

2 Change Control and Configuration Management

After the decision has been made to implement the change, change control and configuration management should be applied to the change. See GAMP® 4 Appendices M8, M9, O4 (see Appendix G2, reference 1).

3 Test Planning and Test Management

Appendix T2 outlines test planning and test management processes conducted during the implementation of a new system or application.

During the operational phase of the system, the responsibility for testing will rest mostly with the User. Nevertheless, the key concepts outlined in Section 2 still apply:

• The scope of testing should still be determined by a justified and documented risk assessment. Testing should be part of an appropriately scaled revalidation activity.
• Testing should still be conducted against documented requirements (updated if necessary).
• Testing should still follow a pre-approved Test Plan or Strategy.
• 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 whereas new software required to address software problems is unlikely to be mature and may require more testing.
• Users should still seek to benefit from Supplier quality assurance processes and associated testing.

4 Testing Changes

4.1 Testing of the Changed Element

There is a need for testing to evaluate the implementation of the changed or new elements. This would be carried out with the same rigor as the original testing (unless there has been a change in the associated risk priority). There may, however, be differences in the execution and management of tests for the following reasons:

• 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 production environment offline for testing during inactive periods. The system may need to be restored if the change is not successful.
• Where testing is conducted in an on-line production environment, additional consideration may need to be given to contingency planning and/or backing out changes.
• Test data may also need to be removed from or isolated within the production environment.
• The project implementation team may no longer exist and the responsibility for testing will then be shared between the System Owner and the various IT maintenance and support groups. The testing roles defined in Appendix T2 may need to be shared amongst a smaller group of people.
4.2 Regression Testing of a Supplier Driven Change

The User 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 what regression testing (if any) needs to be performed by the User.

Prior to the Supplier releasing a new version, the Supplier has the responsibility to prove 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.

4.3 Regression Testing of a User Driven Change

For a User driven change, the User should consider the potential of the change to de-stabilize the system. Focus will usually be on testing that the system meets the revised User Requirements, but regression analysis should still 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 user 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

5 Releasing a Change

Both the Supplier and the User should, independently, have a release strategy for any change.

Key areas for consideration are:

- Batching of changes to optimize efficiency
- Mechanism for release (checklist, acceptance criteria, final sign-off)
- Documentation, education and training requirements
- Communication of change
Section IV - EXAMPLES
Appendix E1 - Testing Process Automation Systems

1 Definitions

The following, taken from the GAMP® GPG: Validation of Process Control Systems (see Appendix G2, reference 3) shows the different types of process automation systems:

Figure E1.1: Process Control System Types

1.1 Configurable Equipment

Configurable Equipment is the collective name given to simple configurable instruments/devices, e.g., 3-term controllers, check weighers, bar code readers, etc. These have functionality via their configuration set up to meet the process requirements. The software components of such systems are typically defined as GAMP® software category 2.

1.2 Embedded Systems

Embedded Systems is the collective name for systems with a greater degree of configuration and programmability. Devices such as Integrated Circuits (IC) with configuration set-ups and Programmable Logic Controllers (PLCs) that are supplied as an integral part to an item of Process Equipment, e.g., PLCs controlling a centrifuge or packaging machine or IC embedded in High Performance Liquid Chromatography (HPLC) systems, Gas Chromatography
System (GCS), etc. Embedded systems typically contain software components belonging to multiple GAMP®
categories.

1.3 Standalone Systems
Standalone Systems is the collective name for large programmable control systems that distribute functionality
across a network e.g., Distributed Control Systems (DCS), Supervisory Control and Data Acquisition (SCADA). They
are engineered as an entity to control a complete plant. Standalone systems typically contain software components
belonging to multiple GAMP® categories.

2 Testing and the GAMP® Life Cycle
The following 'V-model' framework for specification and testing is taken from the GAMP® GPG: Validation of Process
Control Systems (see Appendix G2, reference 3).
2.1 Testing Strategies

As shown in the above V-model, a process automation system being developed for a new application typically requires some or all of the following test phases.

- Suppliers Module Testing
- Suppliers Module Integration Testing
- Suppliers Integration Testing
- Factory Acceptance Test (FAT)
- Site Acceptance Test (SAT)
- Installation Qualification
- Operational Qualification
- Performance Qualification
The exact combination of testing required for a particular system should reflect its complexity, the maturity of its underlying software and hardware elements and the risk impact on product quality, patient safety, and data integrity. Collectively these will determine the risk priority. Where the phrase 'low risk' is used below, this should be taken to mean ‘having a low risk priority as determined by a formal risk assessment’.

Testing of modifications, patches, or upgrades also should be related to the risk priority of the change. For example, it may be appropriate for parameter changes to be applied directly to the production environment, assuming that the systems have been range checked for such a parameter. It may not appropriate to make changes to the logic of a control phase unless they have been module tested in a separate environment under a change control system prior to installation in the production environment.

### 2.2 Typical Test Phases

The table below covers typical test phases for a complex process automation system. The example assumes that the system is configured by a Supplier and delivered to site after a Factory Acceptance Test. Systems can, of course, also be configured by a system integrator or by the User. In this case, the same test coverage is required but the phasing and location of tests may be different.

The User and Supplier should work together to develop an overall approach to testing that reflects the risk assessment output and ensures adequate test coverage of the functionality whilst avoiding unnecessary repeat testing.

**Table E1.1: Typical Test Phases for Process Automation Systems**

<table>
<thead>
<tr>
<th>Test Phase</th>
<th>Timing and Location</th>
<th>Coverage</th>
</tr>
</thead>
</table>
| Supplier’s Application Software Module Testing | At the Supplier’s premises. After the module has been placed under configuration control and code reviewed. Before system integration. | 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. |
| Supplier’s Module Integration Testing | At the Supplier’s premises. After individual modules have been tested and integrated into a single unit. Before full system integration. | Module integration testing generally covers:  
- Correct operation of interfaces between modules.  
Failure paths and response to fault conditions should be included within the tests. |
| Supplier’s Integration Testing | At the Supplier’s premises. After module testing and before the user is invited to witness factory acceptance testing. | Integration testing generally covers:  
- Hardware  
- 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.  
Hardware tests typically include:  
- Checking system build against approved hardware |
<table>
<thead>
<tr>
<th>Test Phase</th>
<th>Timing and Location</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• specification and/or drawings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Recording systems components, version numbers (including software versions) and capacities</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checking electrical supplies and earthing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checking correct power up of system components</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checking any self test/diagnostic information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checking correct communication on any standard interfaces</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I/O interface tests typically include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Exercising inputs and outputs to check correct configuration of ranges, alarm, etc.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operator interface tests typically include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• System displays and navigation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Security and access controls</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tests for interfaces to other equipment typically include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checks of communications protocol set-up</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checks that the required data can be transferred</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checks of actions in the event of a communications failure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tests for system functionality typically include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Monitoring functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Alarm strategies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Control functions (control modules, equipment modules, procedural control)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Power failure and recovery</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Component failure and redundancy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Performance checks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tests for system data handling typically include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Operator data entry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Data formatting and quality checks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checks of calculated values</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checks of recipes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checks of access to current process data, alarms and events (displays, alarm summaries, etc.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checks of access to historical process data, alarms, and events (trends, reports, alarm histories, etc.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checks of audit trail functionality</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checks of data capacity and retention times</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checks of archive and restore</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checks of provisions for electronic signatures</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Checks of disaster recovery procedures</td>
</tr>
<tr>
<td>Test Phase</td>
<td>Timing and Location</td>
<td>Coverage</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Factory Acceptance Testing (FAT)</td>
<td>At the Suppliers’ premises. After Supplier’s integration testing and before the system is released for delivery to the Users’ site.</td>
<td>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 User to select only a small sample for repeats as witnessed tests. The required coverage can, of course, be increased if problems are found within the initial sample. In determining the required coverage, the User needs to 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 likelihood associated with the method of implementation. Before performing a risk assessment to decide on the required coverage, it is appropriate for the User to review the Suppliers’ internal test results to confirm that they are adequately documented. (This would allow reference back to these test results during an inspection).</td>
</tr>
</tbody>
</table>
| Site Acceptance Testing (SAT)/Installation Qualification | At the Users’ premises. After installation of the system on 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 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. |
| Operational Qualification and Performance Qualification | At the Users’ premises. After Site Acceptance Testing/installation qualification. | If a system has been fully tested in FAT, after successful completion of Installation Qualification (along with any additional field functional tests and calibrations) the system is treated as an integrated part of the process equipment and is then qualified as part as the process equipment qualification. This should ensure that the full system, associated equipment, procedures and trained people are all ready for production. |
3 Testing and the Hardware/Software Type and Maturity

Process automation systems generally comprise several elements. Categorizing these elements according to those described in GAMP 4 (see Appendix G2, reference 1) can assist in targeting testing and validation effort.

In general, a process automation system has some or all of the following functional elements:

**Figure E1.3: Process Control System functionality**

Some functional elements may be fully contained within the standard packages or firmware of the system whilst others may require different degrees of configuration or customization for the application. In general, the greater the degree of customization, the greater the scope of testing required.

For example, if an item of process equipment is to be controlled, the User may have a number of options for how the process control functional element is implemented:

**Table E1.2: Examples of Test Phases for Different Process Control Software Category**

<table>
<thead>
<tr>
<th>Control option</th>
<th>Example</th>
<th>Application Life Cycle Testing</th>
<th>Typical Test Phases</th>
</tr>
</thead>
</table>
| Complex control exists as a single, mature, ‘equipment level’ library module on which functions can be selected /deselected and set-up parameters entering. | ‘Sterilizer’ module with selectable cycle types.                                                                | Selection of functions. Set-up parameters.      | • Factory and/or Site Acceptance  
• IQ/OQ/PQ of process equipment                                             |

For ISPE Members only. Copyright ISPE 2005.
<table>
<thead>
<tr>
<th>Control option</th>
<th>Example</th>
<th>Application Life Cycle Testing</th>
<th>Typical Test Phases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex control built up from mature, interlinked ‘device level’ library modules which must then have the correct parameters entered.</td>
<td>Sterilizer control built up from ‘valve’, ‘motor’, ‘PID’, ‘setpoint program’ modules.</td>
<td>Selection and linking of modules to give correct functionality. Set-up parameters</td>
<td>• Factory Acceptance&lt;br&gt;• Site Acceptance&lt;br&gt;• IQ/OQ/PQ of process equipment</td>
</tr>
<tr>
<td>Complex control built up from application specific modules built up from low-level mature library modules (e.g., ‘latch’, ‘compare’, ‘switch’, etc.) or from simple coded steps within a predefined structure (e.g., ladder logic or sequence function chart)</td>
<td>Sterilizer control configured in ladder logic</td>
<td>Functionality of application specific modules Selection and linking of modules to give correct functionality. Set-up parameters</td>
<td>• Module Test&lt;br&gt;• Factory Acceptance&lt;br&gt;• Site Acceptance&lt;br&gt;• IQ/OQ/PQ of process equipment</td>
</tr>
<tr>
<td>Complex control custom coded</td>
<td>Sterilizer control coded in C++</td>
<td>Functionality of custom coded modules Interaction between custom coded modules Selection and linking of modules to give correct functionality. Set-up parameters</td>
<td>• Module Test&lt;br&gt;• Module Integration Test&lt;br&gt;• Factory Acceptance&lt;br&gt;• Site Acceptance&lt;br&gt;• IQ/OQ/PQ of process equipment</td>
</tr>
</tbody>
</table>

Note that the suggested test phases are typical only and that the test requirements may be increased or decreased to reflect the complexity of the application and the impact on product quality, patient safety, and data integrity.

Note also that the above assumes mature ‘industry proven’ library modules. If library modules are created specifically for the application then these need to be treated as custom code.

### 3.1 Example Configured Equipment

A simple example of configured equipment would be a stand-alone 3-term controller:
Table E1.3: Functionality by Software Type – Simple Process Control System

<table>
<thead>
<tr>
<th>System Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O interface</td>
<td>Functionality is all contained within the (mature) firmware and is configured by entering set-up parameters to define range, etc.</td>
</tr>
<tr>
<td>Process Control</td>
<td>Functionality is all contained within the (mature) firmware and is configured by entering set-up parameters to define tuning.</td>
</tr>
<tr>
<td>Alarm Handling</td>
<td>Functionality is all contained within the (mature) firmware and is configured by entering set-up parameters to define thresholds and priorities.</td>
</tr>
<tr>
<td>Graphics</td>
<td>Functionality is limited to standard ‘faceplate’ displays and is all contained within the (mature) firmware. Configuration is by entering set-up parameters to define tags/descriptions.</td>
</tr>
<tr>
<td>Security</td>
<td>Functionality is all contained within the (mature) firmware and is configured by entering set-up parameters to define users and access levels</td>
</tr>
</tbody>
</table>

The system software elements can, therefore, be summarized as follows:

Figure E1.5: Software Elements Contained Within a Simple Process Control System

(Note that for complex 3-term controllers where additional logic can be ‘wired’ in software, profiles set up, etc., it would be more appropriate to treat the controller configuration as a software category 4 module. Where complex controllers also allow sequencing to be defined, there may also be a category 5 ‘coded’ module super-imposed on the configuration)
A typical test approach might, therefore, include a site acceptance test to confirm correct hardware installation (IQ) and parameter entry followed by OQ/PQ as part of the process equipment.

3.2 Example Embedded Control System

An example of an embedded control system would be the control system delivered as part of a skid mounted mixer unit and consisting of a PLC and operator panel with an electronic chart recorder collecting an independent record of critical parameters. This can be broken down into different elements in order to determine an appropriate test approach.

3.2.1 PLC:

Figure E1.6: Example of System Elements Contained Within an PLC

![Diagram of System Elements Contained Within an PLC](image)

### Table E1.4: Functionality by Software Type – PLC

<table>
<thead>
<tr>
<th>System Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O interface</td>
<td>Functionality is all contained within the (mature) firmware and is configured by entering set-up parameters to define range, etc.</td>
</tr>
<tr>
<td>Process Control</td>
<td>Functionality is configured in ladder logic.</td>
</tr>
</tbody>
</table>
3.2.2 Operator Panel

Figure E1.7: Example of System Elements Contained Within an Operator Panel

Table E1.5: Functionality by Software Type – Operator Panel

<table>
<thead>
<tr>
<th>System Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm Handling</td>
<td>Functionality is all contained within the (mature) firmware and is configured by entering set-up parameters to define thresholds and priorities.</td>
</tr>
<tr>
<td>Graphics</td>
<td>Functionality is configured by selecting icons from a library and connecting them to the correct tag in the PLC</td>
</tr>
<tr>
<td>Security</td>
<td>Functionality is all contained within the (mature) firmware and is configured by entering set-up parameters to define users and access levels</td>
</tr>
</tbody>
</table>
3.2.3 Electronic Chart Recorder

**Figure E1.8: Example of System Elements Contained Within an Electronic Chart Recorder**

**Table E1.6: Functionality by Software Type – Electronic Chart Recorder**

<table>
<thead>
<tr>
<th>System Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O interface</td>
<td>Functionality is all contained within the (mature) firmware and is configured by entering set-up parameters to define range, etc.</td>
</tr>
<tr>
<td>Alarm Handling</td>
<td>Functionality is all contained within the (mature) firmware and is configured by entering set-up parameters to define thresholds and priorities.</td>
</tr>
<tr>
<td>Graphics</td>
<td>Functionality is limited to standard trend displays and is all contained within the (mature) firmware. Configuration is by entering set-up parameters to define tags/descriptions.</td>
</tr>
<tr>
<td>Security</td>
<td>Functionality is all contained within the (mature) firmware and is configured by entering set-up parameters to define users and access levels</td>
</tr>
<tr>
<td>Data Capture</td>
<td>Functionality is all contained within the (mature) firmware and is configured by entering set-up parameters to define the values to be collected</td>
</tr>
<tr>
<td>Alarm History</td>
<td>Functionality is all contained within a (mature) standard viewing package and is configured by entering set-up parameters to define the source instrument and required period.</td>
</tr>
<tr>
<td>Trending</td>
<td>Functionality is all contained within a (mature) standard viewing package and is configured by entering set-up parameters to define the source instrument and required period.</td>
</tr>
<tr>
<td>Data Backup and Recovery</td>
<td>Functionality is all contained within the (mature) firmware and is configured by entering set-up parameters to define the target server and required period.</td>
</tr>
</tbody>
</table>

3.2.4 Combination of Elements and Resultant Test Approach

The overall software system elements included within the embedded controller can, therefore, be summarized as follows:
A typical test approach might, therefore, include the following phases:

- Suppliers Module (Unit) Testing of the PLC ladder logic program,
- Suppliers Integration Testing and Factory Acceptance Test (FAT) to demonstrate correct interaction between PLC, panel and recorder and correct operation of the process equipment,
- Site Acceptance Test (SAT) and IQ/OQ/PQ of the control system as part of the process equipment.

Note that if the embedded control system is a mature product (i.e., standard ladder logic and displays, and the only configuration is via entry of parameters to define a specific application) and the supplier has been assessed and their quality management practices are found to be appropriate then the test approach can be reduced to that appropriate to a GAMP® category 2 system with the control system undergoing Site Acceptance Test (SAT) and IQ/OQ/PQ as part of the process equipment.

### 3.3 Example Stand-Alone Control System

An example of a stand-alone control system, e.g., DCS
Figure E1.10: Example of Stand-alone Control System (e.g., Distributed Control)

Table E1.7: Functionality by Software Type – Distributed Control System

<table>
<thead>
<tr>
<th>System Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O interface</td>
<td>Functionality is all contained within the (mature) strategy engine firmware and is automatically configured from parameters entered into the DCS database to define channel, range, etc.</td>
</tr>
<tr>
<td>Process Control</td>
<td>Continuous functionality is defined using library modules for valves, motors, PID controllers, etc. Batch phases are defined as sequence function charts.</td>
</tr>
<tr>
<td>Alarm Handling</td>
<td>Functionality is all contained within the (mature) DCS package and is configured by entering parameters into the DCS database to define thresholds and priorities.</td>
</tr>
<tr>
<td>Graphics</td>
<td>Functionality is configured by selecting icons from a library and connecting them to the correct tag in the DCS database.</td>
</tr>
<tr>
<td>Security</td>
<td>Functionality is all contained within the (mature) DCS package and is configured by entering parameters into the DCS database to define users and access levels.</td>
</tr>
<tr>
<td>Recipe handling</td>
<td>Functionality is all contained within the (mature) DCS package and is configured via a tool which allows master recipes and plant units to be set up and then linked in order to create control recipes.</td>
</tr>
<tr>
<td>Batch scheduling</td>
<td>Functionality is all contained within the (mature) DCS package and is configured via a tool which allows batches to be scheduled across a number of plant units.</td>
</tr>
<tr>
<td>Data capture</td>
<td>Functionality is all contained within the (mature) DCS package and is configured by entering parameters to define which values to capture.</td>
</tr>
<tr>
<td>Alarm History</td>
<td>Functionality is all contained within the (mature) DCS package and history is automatically generated from parameters entered into the DCS database.</td>
</tr>
<tr>
<td>Trending</td>
<td>Functionality is all contained within the (mature) DCS package and is configured by entering parameters into the DCS database to define the required pens and chart details.</td>
</tr>
</tbody>
</table>
### System Element | Description
--- | ---
Reporting | Functionality is all contained within the (mature) DCS package and is configured via a tool which allows report layout, data elements, and scheduling to be set up.
Data Backup and Recovery | Functionality is all contained within the (mature) DCS package and is configured by entering parameters to define the target server and required period.

The system software elements can, therefore, be summarized as follows:

**Figure E1.11: Example of System Software Elements Contained Within a Distributed Control System**

A typical test approach might, therefore, include the following phases:

- Suppliers module testing of the batch phases
- Suppliers integration testing and Factory Acceptance Test to demonstrate correct configuration and functionality of all DCS elements.
- Site Acceptance Test and IQ to show correct installation; to demonstrate that the system is still functioning as accepted during factory acceptance testing; and to cover any elements such as networks which could not be adequately tested in the factory environment.
- OQ/PQ of the control system as part of the process equipment.

### 4 Testing Responsibilities - Supplier and End User

Where a Supplier has been assessed and their quality management system found to be acceptable, the User may benefit from the testing already carried out as part of the product development life cycle in order to minimize the additional testing carried out.

Similarly, the User may benefit from the testing carried out by the Supplier as part of the application development life cycle, for example to allow a small sample of tests to be repeated at witnessed factory acceptance testing.
4.1 Example Life Cycle for a Custom Application

The life cycle below illustrates an example where a new application is developed for an embedded control system.

**Figure E1.12: Development and Test Life Cycle for a New Embedded Control System**

4.1.1 Supplier Product Life Cycle

In this example the Supplier’s product may be software or tools used to develop the User specific application. The control system Supplier has been assessed and their quality management system found to be acceptable. There is, therefore, no need for the User to repeat testing of product functionality. The testing should instead concentrate on the application of the control system.

4.1.2 End User Application Life Cycle

Assuming that the application is critical to product quality and includes a custom sequence coded as a sequence function chart, a test approach could, therefore, be agreed including the following elements:

- Supplier’s module testing of the sequence function chart program,
- Supplier’s integration testing (100% test) and Factory Acceptance Test (sampled) to demonstrate correct interaction configuration of the control system and correct operation of the process equipment.
- Site Acceptance Test and IQ/OQ/PQ of the control system as part of the process equipment.

4.2 Example Life Cycle for a Standard Application

The life cycle below illustrates an example where a standard application is purchased containing an embedded control system.
4.2.1 Supplier Product Life Cycle

The control system Supplier has been assessed and their quality management system found to be acceptable. There is, therefore, no need for the User to repeat testing of product functionality (including mature library functions such as standard control modules) or of the standard application. The testing needs only to cover the set-up and use of the application.

4.2.2 End User Application Life Cycle

Assuming that the application is still critical to product quality, there is now much lower risk associated with the application development as the configuration is limited to selection of the required functions and entry of set-up parameters. A test approach could, therefore, be agreed including the following elements:

- Factory Acceptance Test to demonstrate correct set-up of the control system and correct operation of the process equipment.
- Site Acceptance Test and IQ/OQ/PQ of the control system as part of the process equipment.
Appendix E2 – Testing Configurable IT Systems

1 Definitions

A Configurable IT System is one where the user interface, sequence of operations and data verification and processing are configured to meet the specific needs of an individual Users organization. The different options are controlled by a category of data known as a ‘configuration’ (or ‘configuration settings’, ‘package configuration’, ‘applications configuration’ or similar).

In the context of this Guide Configurable IT Systems are large, highly integrated and, generally, mature systems implemented to integrate the working of many organizational units within the user community.

The degree of configuration varies for each Supplier’s product offering. Typically the core Supplier product provides Users with a set of high level configurable functions that allow users to:

• Turn individual modules on or off (e.g., an organization might want to use its own LIMS application instead of the Quality module within an ERP system)
• Specify within an individual module the degree of control and specific functionality (e.g., within Inventory Control, whether lot control is to be applied).

Examples of typical configurable core products include: -

• Enterprise Resource Planning (ERP)
• Manufacturing Resource Planning (MRPII)
• Laboratory Information Management Systems (LIMS)
• Document Management Systems
• Workflow Management

Note that configurable process automation systems are discussed in Appendix E1 of this Guide.

Such systems are often implemented in a phased manner over a period of time, either with an increasing use of functionality or geographically releasing a standard set of functionality (global template) to diverse sites within the organization.

The GAMP® GPG: Global Information Systems Control and Compliance (see Appendix G2, reference 6) addresses many of the issues associated with such applications, but with specific regard to testing the Test Plan or Strategy needs to address these incremental and reusable aspects.

Although this section focuses on providing testing guidelines for the User, an inevitable outcome is that the design quality and Supplier testing for the core product significantly influences the quality, time and cost of any application build (implementation).

From a Supplier perspective, these core software products should be regarded as GAMP® Software Category 5. For these types of products, Suppliers should ensure that their product testing covers as many of the main configuration options as possible.

The User driven risk assessment process should take into consideration whether the assessed Supplier’s test evidence reflects the anticipated User configuration. Any shortcomings should be addressed and an appropriate risk mitigation strategy agreed. From a User perspective, the configuration of these products to build an application should usually be treated as a GAMP® software Category 4. Development of custom software as part of the implementation of a configurable system should be treated as GAMP® software category 5 and the overall test approach should reflect this (see Figure 2.9 in Key Concepts for an example of a combined software category 4 and software category 4 testing approach).
1.1 Testing and the GAMP® Life Cycle

Figure E2.1 is included to show the two discrete project phases that are associated with a Configurable IT System project implementation.

The left hand “V” reflects the traditional Supplier life cycle that was used to develop the core configurable product. The right hand “V” reflects the User life cycle associated with building an application using the Supplier software.

**Figure E2.1: Typical Configurable IT System Double “V” Model**

- **End user application lifecycle**
- **Supplier product lifecycle**

The Package Configuration Specification details the configuration to be applied to the system and the Configuration Verification confirms that it has been accurately applied (IQ).

The Functional Test is performed on the configured system. This testing should be based on the operational methods (business processes) and procedures developed by the team during the implementation process. It should be based on the end-to-end testing of the business processes, together with challenges to specific high risk impact functions or business process by executing appropriate Test Cases, to demonstrate that the process as described in the Functional Specification operates as expected (OQ).

The Performance Test, which may incorporates User Acceptance Testing (UAT) against User Requirements is a series of Test Cases that are necessary to support a single instance of the implementation, for example a site. Where a site uses a different instance of the system, or where the operation of the system is different from one site to another (for instance, by varying workflow or modifying transactions based upon some site specific parameter) each site should conduct its own Performance Tests (PQ). This may also require a degree of regression testing to confirm that related functions that are not site specific still operate in accordance with system requirements.

1.2 Testing Strategies

The following issues should be considered in the Test Plan or Strategy and are in addition to those described in section 2.1 of the ‘Key Concepts’ section of this Guide and are of particular relevance for a Configurable IT System. These are:

For ISPE Members only. Copyright ISPE 2005.
• The extent of the system configuration (including the use of configuration settings not tested by the Supplier)
• Confidence in the Supplier – how well have they tested the system, particularly in the area of configuration:
  • Demonstrating the link between specific configuration settings and associated functionality
  • Controls & audit trails related to the application of configuration
• How the system will be configured (e.g., a single point and method of applying configuration settings versus multiple points and methods)
• How this configuration is to be propagated (transported) through the landscape (between different instances) and over time
• Central versus Local instances – what parts of the system will be tested where and when
• Scope of Configuration settings (global versus local)
• Configurable Security and Access Control – this also should be tested
• The extent of any project specific custom extensions added to the Suppliers standard software (or custom modification to the Suppliers standard code, where allowed), including workflow, reports, interfaces, data conversion routines, forms or functional enhancements

A Test Plan or Strategy is usually approved by both the Supplier (often a systems integrator) and User organizations. The scope of the Test Plan or Strategy should include defining the type of testing to be applied to the User specific configuration.

It may also be necessary to determine the extent of verification (demonstrating accurate application of the configuration settings) that may be required. This is a separate activity to testing (exercising the functionality that the configuration has invoked where it is important to consider what level of boundary, negative, performance and stress testing may be warranted. This will usually be determined on the basis of a risk assessment.

1.2.1 Application Configuration Verification

Risk assessment should determine what types of testing need to be carried out against which requirements and/or software elements and with what degree of rigor. This should always be mapped against the configuration documented in the Package Configuration Specification. Systems identified as high risk priority would, therefore, require more rigorous testing.

Within the risk assessment the nature of the configuration, how configuration settings are managed and the potential impact on the business (associated impact on product quality, patient safety, and data integrity) should be used to define and justify the rigor with which configuration settings are managed and verified. This may well differ for different configuration setting type, examples of which are:

• Technical configuration settings (e.g., Database performance and set up, database tables, search and sort keys, printer and print queue set up, etc.) versus configuration that drives User application functionality. Is this sufficiently covered by the support organization’s SOP, installation procedures, and controls?
• Transportable configuration settings (applied in one place and copied by the system to other instances)
• Manually entered configuration settings, applied to each instance
• Global configuration settings (a single setting that controls functionality in all instances)
• Local configuration settings (where each instance needs its own setting)
• Site specific configuration settings (affects defined plants or sites within an instance. May be a global or local configuration setting within an instance)

Configuration setting (affects the logic of processing a function) versus Master Data (a set of codes or values that a functionality uses, e.g., Material Types, or Location identifiers within a Warehouse); if these are both being managed using the same mechanisms within a system then the testing regime would apply equally to both.
1.2.2 Types of Testing

The Suppliers testing should be appropriate for a full product development life cycle (GAMP® Category 5 software). User related testing should usually be limited to those test types listed in the following table (note that this assumes no custom software development):

Table E2.1: Test Types for Configurable IT Systems

<table>
<thead>
<tr>
<th>Test Phase</th>
<th>Typical Timing and Location</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
<td>At the User’s premises. After the software has been configured (or while it is configured). The method of controlling and propagating the configuration through the testing and production environments will indicate if this has to be done for each implementation in the production environment or can be performed once in a test environment.</td>
<td>Verification of configuration against a predefined Package Configuration Specification.</td>
</tr>
<tr>
<td>Verification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional Test</td>
<td>At the User’s premises. After the software has been configured. This should generally be performed once using the standard set of functionality (global template) on a central instance in the test environment.</td>
<td>Process Testing against Process Descriptions Boundary Testing Negative Testing User Acceptance Testing of the standard set of functionality against User Requirements</td>
</tr>
<tr>
<td>Performance Test</td>
<td>At the User’s premises. After the software has been configured for each specific implementation. This is a part of the rollout program (successive releases). This may be performed for each implementation of the standard functionality in two parts. The first part may be performed typically in the test environment with data, people, and operating documentation for a specific implementation. The second may be performed in the production environment after a specific implementation has gone live.</td>
<td>In the test environment: • Acceptance Testing of the (sub-set) of standard functionality that a implementation instance is to adopt • Stress Testing In the production environment: • Functional Performance Testing • Product and Process Performance Testing</td>
</tr>
</tbody>
</table>

Where customization of the standard configurable system is conducted additional unit and integration testing would also be required.

1.3 Testing & Hardware/Software Type & Maturity

1.3.1 Test Planning

Implementing Configurable IT System applications to manage an organization’s business processes is a highly complicated task. Consequently, the creation of a Test Plan or Strategy is seen as an important task within a project plan and is essential not only to successfully verify the business process requirements but also to maximize the use of critical project resources (time, money, and people).

Creating a Test Plan or Strategy is not an exact science and the output for each project and each organization will be different. The Test Specifications or Protocols should align with the different scope sub-sections of the Test Plan.
GAMP® Good Practice Guide: Testing of GxP Systems

or Strategy. (Note that in many Configurable IT projects it is common to develop a Test Strategy and multiple Test Plans for each phase or cycle of testing. However, this Appendix uses the terms Test Plan and Strategy and Test Specifications or Protocols as defined in the Key Concepts section of this Guide).

All baselined requirements need to be addressed with clear links to appropriate Test Cases. Maintaining traceability of requirements through the design phase should alleviate unnecessary duplication of testing effort.

The output from the Test Planning phase should be used to generate the appropriate Test Cases.

In many large configurable IT systems there will be a combination of GxP and non-GxP functions and the Test Plan or Strategy will need to address how this will influence the scope and rigor of the testing. Non-GxP functions will also need to be tested, but not necessarily to the same extent as GxP functions (i.e., end-to-end business process testing may be adequate for less critical non-GxP business processes and negative Test Cases may not be required). Risk assessment can be used to determine the scope of rigor of non-GxP testing, depending upon the business criticality of the function being tested. It is possible to extend the scope of the risk assessment process to include business impact (of which GxP impact is a subset).

The examples given below are based around an ERP system but the same principles apply to other configurable IT systems.

Organization of Test Specifications or Protocols for Configurable IT Systems

For such projects, the structure and content of the Test Specifications or Protocols is one of the most important project activities and there are a number of important factors that should be considered.

In simple terms, the Test Specifications or Protocols should provide a more detailed statement of how the Test Plan or Strategy will be implemented and should take account of the following elements:

- The criticality (risk priority) of the function being tested (GxP and Business),
- Logical structure/flow of the modules,
- The most appropriate method for testing a specific function or risk scenario,
- The most appropriate time to test the business process requirements,
- The structure and level of detail of design documentation,
- Logistics for Creating, Maintaining and Approving test scripts,
- The availability/readiness of any data that is required to execute a test,
- The availability of skilled testing resources,
- The best use of testing resources,
- The status of any interfaces required.

The Breakdown of Test Scope into Test Specifications or Protocols (Work Packages)

The following sub-sections identify some of the basic options on how to structure a Test Specification or Protocol based purely on the structure of the configurable software. An example of an ERP system is used, but the principles can again be applied to any configurable IT system.

Module Approach

Figure E2.2 depicts a high level overview of the modules that could exist within a typical ERP product.

For ISPE Members only. Copyright ISPE 2005.
The Test Plan or Strategy could be established by breaking down the application into test phases based upon the application modules or software hierarchy. Each test phase would have a corresponding Test Protocol or Specification.

The figure below illustrates a simple modular approach to test planning.
This model depicts the situation where the initial testing is organized by individual business processes areas within specific functional modules (e.g., the creation of purchase orders within the purchasing module).

If the structure of the Test Strategy or Plan is generated using this modular approach, subsequent tests may need to be generated at a later stage in the Test Plan in order to test the module-to-module integration (e.g., to verify for example, that a Purchase Order results in an Account Payable activity). In this case additional Test Protocols or Specifications would be required for the integration testing.

The main advantage of this approach is that individual modules can be tested and released into production separately (phased release rather than a ‘Big Bang’ approach, where the release to production has to wait until all testing is complete). Whether such a phased approach is possible depends upon the details of the specific configurable system.

The main disadvantage is that it will most likely involve elements of regression testing for each release of new functionality.

**Cross-Module Approach**

An alternative approach to constructing a Test Plan is shown in Figure E2.4 and should typically be used in situations where there is a high degree of confidence in the low-level core functions and the quality of business process configuration.
This approach to testing may be more appropriate where the testing is based more on end-to-end business process (e.g., “Procure to Pay” or “Inspect to Release”) or Integration testing. Using the example above, the plan would be to test all of the business processes from the creation of a purchase order and the receipt of a sales order though to the accounts general ledger. This should cover all the intermediate elements of receipt, invoicing, and accounts payable.

The main advantage of this approach is that it should reduce the level of testing effort required (because it relies on basic software module testing executed by the Supplier) and is, therefore, well suited to mature configurable applications.

The potential disadvantage is that any defects may not be identified until later on in the project.

Other Factors to Consider when Establishing a Test Strategy or Plan

Impact of Custom Modifications, System Interfaces and Data Migration

The test plan approaches defined in Figure E2.3 and Figure E2.4 reflect the situation where an application can be configured without any custom modifications or system interfaces. This is a simple model and does not reflect a typical project.

A more realistic project scope for a configurable IT system is shown in the following figure:
This depicts a more realistic project structure in that the system:

- Has custom interfaces to other applications,
- Requires data import (migration) from existing legacy systems which may involve custom software,
- Involves some custom enhancement to the Supplier delivered standard software.
- Recognizes the likelihood that any application will most likely be operating on a central company infrastructure with network traffic from other applications.

Within this architecture each of the shaded components should be tested as GAMP® software category 5. Consequently the application build Test Strategy or Plan also should include unit and integration testing in addition to the functional and performance testing of the configurable software. Unit, integration (and any performance) testing of the custom software will usually require a separate Test Specification or Protocol and will be typically be executed to the functional and performance testing of the configurable software.

The Test Strategy or Plan also should cover any local site performance elements, which cannot be adequately tested elsewhere. An example could be to test the local site infrastructure, paying specific attention as to how the system will co-exist with other business applications.

## 1.3.2 Test Cases

Figure E2.2 in the Test Planning section shows how a typical ERP application could be broken down into a number of discrete modules.

The figure below illustrates how a Purchasing Module could be comprised of an integrated set of lower-level modules (e.g., Complete a purchase requisition, raise a purchase order).

This purchase order then allows purchase orders to be raised for a number of procurement methods as shown in the diagram below. The range of procurement methods required by the application should have been specified as a unique statement in the business user requirements.
Understanding the modules, processes, and requirements in such a fashion facilitates the effective generation of appropriate Test Cases. This approach also can be applied to other configurable IT systems, e.g., different test cases can be developed for the different test methods and calculations supported by a LIMS, or for the different workflows used to approve different document types in an EDMS.

This may be one situation where a single Test Script may be written to execute several different Test Cases, where each Test Case specifies different test input data and different expected results.

### 1.4 Testing Responsibilities – Supplier and User

The Supplier provides a system that has many optional paths through the software available. The Supplier also should test that each specific configuration option operates successfully and that the application of configuration settings selects the different options that are available. They may also demonstrate consistency across different major modules of the system and that the modules integrate with one another.

The User may engage an implementation team analyze the needs of their business and decide how the Configurable IT System should operate to best meet their needs. This is documented in the User Requirement Specification. This team may be made up of people from the Suppliers, third party integrators or the User community with specialist knowledge of the system in question. They develop and apply the configuration needed to implement the Users required functionality and document and verify the configuration settings. These configuration settings will usually enable or disable major software modules at the highest level and select specific transactions, data types, verification, and processing options at the lowest level.

The security or access to the system should usually be controlled as one key aspect of the configuration. To compliment the system operation, user instructions or operating procedures should be prepared to describe how
users perform specific tasks. It is the responsibility of the User to document and verify the configuration applied and to demonstrate that their (unique) way of operating the systems works with their documents, people and data. This will typically be only a fraction of the possible options that the Supplier has built into their system. It should be clear (and documented) which options are used and which ones remain unused.

Although the Supplier is responsible for testing all elements of the configurable product, the User (often supported by a system integrator) is responsible for:

- Configuring the system
- Specifying the transactions sequences that are used to satisfy the business process requirements
- Specifying and building any interfaces to existing systems
- Specifying and developing any data migration activities
- Testing that the final application meets the business user requirements

It is recommended that because of the complexity of such systems a Test Plan or Strategy and multiple Test Specifications or Protocols are created.
Appendix E3 - Testing Analytical Instruments

1 Definitions

Analytical instruments are laboratory systems which consist of integrated hardware and software in order to perform measurements and produce data.

Analytical instruments have been further categorized in the GAMP® Good Practice Guide: Validation of Laboratory Computerized Systems (see Appendix G2, reference 4). This system of categorization has been reproduced here for clarity:
### Table E3.1: Analytical Instrument Categorization

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
<td>Software and configuration is not modifiable</td>
<td>Software and configuration is not modifiable</td>
<td>Configuration parameters stored and reused</td>
<td>Configuration parameters stored and reused</td>
<td>Configuration parameters stored and reused</td>
<td>Configuration parameters stored and reused</td>
<td>Custom Systems</td>
</tr>
<tr>
<td>Interfaces</td>
<td>No computer interface used</td>
<td>No computer interface used</td>
<td>No computer interface used</td>
<td>May have 1 to 1 ratio (Computer to instrument interface, server to client interface)</td>
<td>May have 1 to many ratio (Computer to instrument interface, server to client interface)</td>
<td>May have 1 to many ratio (Computer to instrument interface, server to client interface)</td>
<td>Custom System</td>
</tr>
<tr>
<td>Data Processing</td>
<td>Conversion of A/D signals</td>
<td>Conversion of A/D signals</td>
<td>Data manipulated by a separate program external to the system</td>
<td>Post-acquisition processing done as part of the system (can analyze data with proprietary data handling system)</td>
<td>Post-acquisition processing done as part of the system (can analyze data with proprietary data handling system)</td>
<td>Custom System</td>
<td></td>
</tr>
<tr>
<td>Results and Data Storage</td>
<td>Information generated based upon instrument function is stored, i.e., Calibration is stored</td>
<td>Process Parameters input and Stored (runtime parameters, methods parameters)</td>
<td>Process Parameters input and Stored (runtime parameters, methods parameters)</td>
<td>Process Parameters input and Stored (runtime parameters, methods parameters)</td>
<td>Process Parameters input and Stored (runtime parameters, methods parameters)</td>
<td>Custom System</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Does not produce raw data or test results</td>
<td>Produces raw data or test results but records not stored or processed</td>
<td>Produces raw data or test results but records not stored or processed</td>
<td>Produces raw data or test results, which are stored but not processed</td>
<td>Produces raw data or test results, which are stored and processed</td>
<td>Produces raw data or test results, which are stored and processed</td>
<td>Custom System</td>
</tr>
</tbody>
</table>

### Table E3.1: Analytical Instrument Categorization (continued)

<table>
<thead>
<tr>
<th>Examples of Laboratory Systems</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ovens</td>
<td>PCR Thermal Cyclers</td>
<td>Spectrophotometers</td>
<td>Microplate Readers</td>
<td>Integrated Robotics Systems with Data Acquisition and Data Processing</td>
<td>Trademark Proprietary Data Acquisition Systems with configured customizations</td>
<td>Spreadsheets/statistical analysis software (templates)</td>
<td>Bespoke system</td>
</tr>
<tr>
<td>Centrifuges</td>
<td>pH Meter</td>
<td>Particle Counters</td>
<td>Microplate Readers</td>
<td>Integrated Robotics Systems</td>
<td>MS</td>
<td>Amino Acid Analyzer</td>
<td></td>
</tr>
<tr>
<td>Incubators</td>
<td>Thermometer</td>
<td>Simple Robotics Systems</td>
<td>Integrate Robotics Systems</td>
<td>MS</td>
<td>Spreadsheets/statistical analysis software (templates)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temperature Controllers</td>
<td>Viscometers</td>
<td>NMR spectrophotometer</td>
<td>NMR spectrophotometer</td>
<td>Amino Acid Analyzer</td>
<td>Spreadsheets/statistical analysis software (templates)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Environmental Chambers</td>
<td>Conductivity Meters</td>
<td>Pulse field gel equipment</td>
<td>Urine Chemistry Analyzer</td>
<td>NMR</td>
<td>Spreadsheets/statistical analysis software (templates)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sonicators</td>
<td>Titrators</td>
<td>GC system with key pad control</td>
<td>GC</td>
<td>Automated Hematology Systems</td>
<td>Spreadsheets/statistical analysis software (templates)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Glassware washers</td>
<td>Chart Recorders</td>
<td>HPLC key pad control</td>
<td>HPLC</td>
<td>HPLC/GC/MS with Trademark Proprietary Data Acquisition Systems</td>
<td>Spreadsheets/statistical analysis software (templates)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Electrolyte Analyzer</td>
<td>Immunoassay System</td>
<td>ECG</td>
<td>Spreadsheets/statistical analysis software (templates)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

129

For ISPE Members only. Copyright ISPE 2005.
Analytical instruments discussed in this testing Guide are those in the Laboratory Categories B to F. Laboratory Category A instruments are simple measurement devices; category G is fully custom and would require the same testing and validation as any other GAMP® Software Category 5 system.

Two laboratory categories are examined in detail from a User perspective in this section, category E (GAMP® software category 3) and category F (GAMP® software category 4).

It should be emphasized that from a Supplier perspective, all analytical instruments should be developed as GAMP® Software Category 5 (or equivalent).

### 1.1 Testing and the GAMP® Life Cycle

Testing is a part of the overall implementation process. Although some of the specification deliverables are shown in the tables below, the following pre-requisites are required for successful implementation:

- User Requirement Specification
- Validation Plan
- Supplier assessment
- Availability of Supplier documentation (critical documentation would be the development specifications and evidence of detailed testing)
- Predetermined change control processes
- Good project management (including risk management)

The above are out of the scope of this document, however, guidance in these areas can be found in GAMP® 4 (see Appendix G2, reference 1).

The overall acceptance criteria for any new equipment would be that the installed, configured, and qualified instrument meets the business and regulatory needs. The acceptance criteria for individual test phases and Test Cases should be structured with this aim in mind.

#### 1.1.1 Category D/E Lab System

The following diagram highlights the different phases of validation required by the instrument Suppliers and Users for GAMP® Software Category 3 (Lab Systems Categories D and E):
1.1.2 Category F Lab System

The following diagram highlights the different phases of validation required by the instrument Suppliers and Users for GAMP® Software Category 4 (Lab Systems Category F):

---

For ISPE Members only. Copyright ISPE 2005.
1.2 Testing Strategies

1.2.1 Category D/E Lab System

The following provides a test approach which may be used for instruments in Lab Categories D and E. It assumes the key concepts of testing (section 2) have been applied and the Supplier has followed the full development lifecycle.

The first step is to define the requirements for the analytical instrument; the Supplier should then respond with a copy of their standard Functional Specification and an assessment of any non-conformances to the URS.

A Supplier assessment may be required – this should be determined according to the business impact of the system and its complexity. The GAMP GPG: Validation of Laboratory Computerized Systems (see Appendix G2, reference 4) section on Vendor Assessment offers a table to assist with this decision-making process.

The User should then determine the test priority of the requirements in accordance with the GAMP GPG: Validation of Laboratory Computerized Systems (see Appendix G2, reference 4). When a list of the requirements to be tested has been compiled, the User can proceed to prepare the Test Cases and Test Scripts.

The Performance Test referenced in the refined V model for Software Category 3 systems would typically comprise an Installation Qualification, and a combined Operational and Performance Qualification. For instruments used for critical assay, the Performance Qualification may involve the actual assay method, in which case it will be executed separately from the Operational Qualification.

Test Case and Test Script preparation, review and test reporting should follow the process flow shown in Figure T2.1, although for this category of analytical instrument, the Supplier is unlikely to be involved - the process will usually start with a User defining the requirements.
On completion of testing, a review should be performed to determine whether the instrument met the acceptance criteria.

The table below provides a suggested qualification approach with specific relevance to analytical instruments in categories D and E:

**Table E3.2: Applicable Test Types for Category D/E Laboratory Systems**

<table>
<thead>
<tr>
<th>Test Type</th>
<th>Timing/Location</th>
<th>Coverage</th>
</tr>
</thead>
</table>
| Installation Qualification       | At the User’s premises.  
                                | After installation of the system on site.                                      | • Checking that the full system including hardware, software backups, and documentation has been delivered to site in a condition suitable for its intended purpose.  
                                |                                                                                  | • Recording the installed software version.                                           |
|                                  |                                                                                  | • 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.                            |
|                                  |                                                                                  | • Review of the calibration status of the instrument (with respect to good calibration management practice).                          |
| Operational Qualification/       | At the User’s premises.  
                                | After successful completion of the Installation Qualification                  | • Testing of the critical functions (as identified during the test priority assessment) to ensure they meet the documented requirements.  
                                |                                                                                  | • Testing of the physical and logical security of data.                                 |
| Performance Qualification        |                                                                                  | • Review of the training records for operations personnel.                        |
|                                  |                                                                                  | • Review of the procedures covering the instrument, and the implementation of these procedures.                               |
|                                  |                                                                                  | • Verification of the accuracy and reproducibility of the instrument’s measurement.                                               |
|                                  |                                                                                  | • Verification that unused functionality is either not accessible or does not cause problems.                                 |
|                                  |                                                                                  | • Verification of long-term continuous operation capability if applicable (e.g., long dissolution runs; extended filterability testing). |

**1.2.2 Category F Lab System**

The following provides a test approach which may be used for instruments in Lab Category F. It assumes the key concepts of testing (section 2) have been applied and the Supplier has followed the full development life cycle.

The first step is to define the requirements for the analytical instrument; the vendor should then respond with a copy of the Supplier’s Functional Specification for the core package. Non-conformances to the URS should be identified.

A Supplier assessment will probably be required – this should be determined according to the business impact of the system and its complexity. The GAMP GPG: Validation of Laboratory Computerized Systems (see Appendix G2, reference 4) section on Vendor Assessment offers a table to assist with the decision-making process.

The User, with input from the Supplier, should then be able to prepare a configuration specification to identify how the equipment should be configured to meet their needs.

On receipt of the equipment, the Supplier should provide training to the User, who can then configure the equipment in accordance with the specification. Configuration Verification would usually be a paper based, peer review exercise. This is done to check the configuration meets the pre-defined configuration specification. For further information, see section 7.8 of GAMP® 4 (see Appendix G2, reference 1).
The User should then determine the test priority of the requirements in accordance with the GAMP GPG: Validation of Laboratory Computerized Systems (see Appendix G2, reference 4). When a list of the requirements to be tested has been compiled, the User can proceed to prepare the Test Cases and Test Scripts.

Functional testing would consist of Installation Qualification to confirm that the correct version of the configured instrument/system has been successfully installed and commissioned, and Operational Qualification to test that the configuration of the instrument/system works as expected (as detailed in the configuration specification). OQ should include testing any functionality of the instrument/system that has been determined as high risk priority through a formalized risk assessment process.

The Performance Test requirements should be met in the Performance Qualification which should check that the instrument/system meets the User’s business workflows, (mapping back to the User Requirements).

Test Case and Test Script preparation, review and test reporting should follow the process flow shown in Figure T2.1. On completion of testing, a review should be performed to determine whether the instrument met the acceptance criteria.

The table below provides a suggested qualification approach with specific relevance to analytical instruments in category F:

**Table E3.3: Applicable Test Types for Category F Laboratory Systems**

<table>
<thead>
<tr>
<th>Test Type</th>
<th>Timing/Location</th>
<th>Coverage</th>
</tr>
</thead>
</table>
| Installation Qualification| At the User’s premises. After installation of the system on site.             | Checking that the full system including hardware, software backups, and documentation has been delivered to site in a condition suitable for its intended purpose.  
Recording the installed software version.  
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.  
Review of the calibration status of the instrument (with respect to good calibration management practice). |
| Operational Qualification | At the User’s premises. After successful completion of the Installation Qualification | Testing of the configuration against the configuration specification.  
Testing of the critical functions (as identified during the Test Priority assessment) to ensure they meet the documented requirements  
Testing of the physical and logical security of data.                                                                                      |
### Test Type | Timing/Location | Coverage
--- | --- | ---
Performance Qualification | At the User’s premises. After successful completion of the Operational Qualification | Review of the training records for operations personnel. Review of the procedures covering the instrument, and the implementation of these procedures. Verification that the system meets the User’s business workflows. Verification of the accuracy and reproducibility of the instrument’s measurement. Verification that unused functionality is either not accessible or does not cause problems. Review and testing of essential interfaces (it may be necessary to include the interaction between several qualified systems during this stage). Testing of the instrument under full production conditions to ensure that the equipment, procedures, and personnel are all ready for production. Verification of long-term continuous operation capability if applicable (e.g., long dissolution runs; extended filterability testing).

### 1.3 Testing and the Hardware/Software type and Maturity
While the above sections provide guidance, a risk-based approach should be taken in each instance of implementation/development. Factors to consider in this approach could be:

- Maturity of the product
- The history of the use of the vendor within regulated industries
- Existing knowledge of the Supplier
- Knowledge of the Supplier’s quality management practices
- Criticality of User application (business and GXP)

### 1.4 Testing Responsibilities – Supplier and User
The tables below summarize the Supplier and User responsibilities for the development and testing of analytical instruments, i.e., the right-hand (User) V-model.

As previously stated, the Supplier is responsible for following a full development life cycle on the product, as shown on the left-hand V-model.

#### 1.4.1 Category D/E Lab System

**Table E3.4: User and Supplier Testing Responsibilities – Category D/E Laboratory Systems**

<table>
<thead>
<tr>
<th>Validation Deliverable</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Requirement Specification</td>
<td>User</td>
</tr>
<tr>
<td>Assessment of Non-conformances to the URS (this may take the form of a Design Review, possibly in partnership with the User).</td>
<td>Supplier (providing the User with a copy of the standard Functional Specification for the instrument can be beneficial). For further information refer to GAMP® 4, Appendix D2, sections 3.2.2 and 3.2.3 (see Appendix G2, reference 1).</td>
</tr>
<tr>
<td>Parameter Specification</td>
<td>User, with input from the Supplier. This can be an interactive exercise, providing a training opportunity.</td>
</tr>
<tr>
<td>Installation Qualification</td>
<td>User, or by User request, Supplier</td>
</tr>
</tbody>
</table>
1.4.2 Category F Lab System

Table E3.5: User and Supplier Testing Responsibilities – Category F Laboratory Systems

<table>
<thead>
<tr>
<th>Validation Deliverable</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Specification</td>
<td>Supplier (this is likely to be a response to the User Requirement Specification, based upon the Supplier’s Functional Specification for the core package). Non-conformances to the User Requirement Specification should be highlighted. For further information refer to GAMP® 4, Appendix D2, sections 3.2.2 and 3.2.3 (see Appendix G2, reference 1).</td>
</tr>
<tr>
<td>Configuration Specification</td>
<td>User, with input from the Supplier, or system integrator. This can be an interactive exercise, providing training opportunity.</td>
</tr>
<tr>
<td>Configuration Verification</td>
<td>User, with input from the Supplier, or system integrator.</td>
</tr>
<tr>
<td>Installation Qualification</td>
<td>Supplier or system integrator.</td>
</tr>
<tr>
<td>Operational Qualification</td>
<td>User (may be supporter by Supplier).</td>
</tr>
<tr>
<td>Performance Qualification</td>
<td>User.</td>
</tr>
</tbody>
</table>
Appendix E4 - Testing Desktop Applications

1 Definitions

Desktop applications are broadly defined as those applications developed from software packages intended for use in office or similar non-production environments and executed in a personal computer (desktop or laptop), e.g., Access, Paradox, Excel, and Lotus 1-2-3. In the context of this Appendix a ‘desktop application’ is the collective use of the standard commercial-off-the-shelf (COTS) application along with any necessary configuration or customization necessary to meet the specific requirements of the User.

These applications are normally standalone and have no real time interface to any other application. They may import or export data to other applications but this will normally be under the control of the user.

Desktop applications will normally be used for manipulating data and possibly displaying, storing or printing GxP data in a format that is more easily readable than its’ raw format. It may also be used for analysis of raw data, e.g., trending, averaging etc.

Desktop applications often provide standard facilities for importing and manipulating data and also may provide additional facilities for writing application specific macros and modules to allow more comprehensive data handling.

The baseline packages and applications generally run on personal computers (PCs) which, although they may be connected to a company’s network infrastructure, will usually be run under the control of the local PC. Output from the application may be stored on the company’s network for back-up and security reasons.

Desktop applications may be developed in-house or by a Supplier. The ease with which simple and complex applications can be developed using these packages can tend to mask the need for testing that should be no less rigorous than applications developed using major software development toolsets.

Users should assess the risk priority associated with any desktop application, determine which of the GAMP® software categories are applicable, and execute an appropriate validation process. The risk assessment process should be fully documented with clear rationale for the decisions arrived at, including the scope and rigor of testing.

Desktop Application fall into one of GAMP® software categories 3, 4, or 5.

2 Testing and the GAMP® Life Cycle

The development and testing of desktop applications can fit the GAMP® life cycle irrespective that they may be considered to be developed in a less structured manner than large software projects. The User should ensure that correct design, development, and testing phases are executed, appropriate to the software category and risk priority. The validation of software categories 4 and 5 are likely to require a greater number of document types. The scope of each document type is dependent on the complexity of the application and the risk priority, e.g., an application posing higher risk priority will typically require a greater number of test scripts.

The GAMP® life cycle has the notion of a Supplier and User within the software development life cycle. Where a third party system integrator is used to develop the application then a Supplier/User relationship still exists and the same rigor should be applied by the User in verifying an appropriate life cycle development process.

Often desktop applications are developed by the User and the Supplier interface is no longer applicable. Large software companies who develop the baseline packages are unlikely to allow the User access to their development life cycle documents and test results. In this instance the User should consider this as part of the risk assessment. The maturity of the Supplier and the product should be considered as described in the key concepts section of this Guide and appropriate steps should be taken to mitigate any additional risk scenarios that result.

The GAMP® Guide Appendix M4, section 3 (see Appendix G2, reference 1), states:

"In summary, spreadsheets can fall into category 3, 4, or 5. The use of a spreadsheet package purely to generate a paper document should be considered as category 3. A more complex application involving, for example, templates should be considered as category 4. A spreadsheet application using custom macros, sophisticated logic or lookup functions should be treated as category 5".
The approach can be extended to any desktop applications. Companies could consider producing a generic set(s) of templates for desktop applications of this type. The ease of producing relatively simple applications by people who are inexperienced in the processes normally associated with software development means that without suitable guidance final desktop applications may not be suitably configured and tested and relevant documentation not to the required standard.

3 Testing Strategies

Test strategies should follow the same approach defined in this Guide, dependent in part on the software category determined for the application. The same types of documents should be produced although it is likely that, because the applications will normally be simpler, overall requirements and design details may be condensed into fewer documents and in some simple and straightforward instances a single design document may suffice. Similarly, a significantly smaller number of test documents might be required with Test Plans or Strategies, Test Specification or Protocols and Test Cases and Scripts all combined into a single document in many cases. However, the User should ensure that that the document set produced does cover all test requirements. In many cases the most cost effective method of testing such applications is by peer testing, conducted by another intended user of the application (recognizing the principles of independent review and testing).

Test documentation will typically include the following:

- Base documentation used during test
- Documented test coverage traceability (test cases to requirements)
- Record of test environment, specifically the Operating System Version and Service Pack Level, Application Version, and Application Add-in Version
- Record of test data
- Test result sheets
- Fault/incident reports

The Test Plan or Strategy will include the identification of test types, types of functions that require emphasis in tests and types of generic test methodologies. The testing of desktop applications will usually follow the generic approach for Category 3, 4 and 5 applications with emphasis on the following areas:

- Suitability & Accuracy of Functions,
- Ability to Handle Irregular Inputs
- Compatibility Tests
- Security Tests
- Tests for robustness (in case of the PC ‘locking up’ or ‘slowing down’ due to interactions with other applications or processes)

Depending upon the software category, unit, integration, functional and user acceptance testing may be required. The following table describes typical test types conducted on desktop applications which may require negative Test Cases for higher risk priority functions.
### Table E4.1: Desktop Applications – Test Types

<table>
<thead>
<tr>
<th>Test Goal</th>
<th>Variances</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Suitability &amp; Accuracy of Functions</strong></td>
<td>Functions can take the form of embedded functions that are a standard part of the software or functions developed through the enhancement of the underlying software to develop macros or program coded functions. Specific Test Cases may be required to test the accuracy of critical functions.</td>
<td>All functions, macros, and code developed as part of the enhanced application that operates within the application. For example, a macro in Excel that obtains data from multiple worksheets to perform a calculation. All functions, macros, and code developed to obtain and provide information to other applications. For example, code developed to extract data from an Excel application into an Access database.</td>
</tr>
</tbody>
</table>
| **Ability to Handle Irregular Inputs**        | Input data encompasses direct data entry by the user as well as data input through extraction from other applications.                                                                                                                                                                                                                     | The following represents the areas covered by this test:  
• Main file – ability to accept input from the main application file  
• Value Lists Entered By users – ability to accept user input  
• Values Transferred from Another Application or File  
• Formulas and Calculations – ability to process formulas and calculations using the data input.  
• Real Data Tests – ability to test real data in a simulated live environment.  
• Unacceptable Forms of Data – combinations of data known to be unacceptable formats or inputs.                                                                                                                                                                                                                                           |
| **Compatibility Tests**                       | Applications where compatibility limitations are known.                                                                                                                                                                                                                                                                             | Coverage for this testing is focused on:  
• Software versions where compatibility is known to cause problems  
• Software versions where compatibility is unknown or unproven, where the version is likely to be part of the common user base.                                                                                                                                                                                                                                           |
Security Tests
(With often a high level of computer literacy in users and varying forms of protection functions within the baseline, security testing should be conducted to ensure that data cannot be compromised under normal operating circumstances)

<table>
<thead>
<tr>
<th>Test Goal</th>
<th>Variances</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Often several levels of security within a company's infrastructure plus levels within the package itself.</td>
<td>Coverage for this testing is focused on:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Access levels at desktop and infrastructure network levels particularly for data storage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Security of access to pre-defined data fields, macros and any specific code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If to be executed on &gt; 1 system ensuring that the application will function consistently across the range of potential hardware platforms</td>
</tr>
</tbody>
</table>

3.1 Testing and the Hardware/Software Type and Maturity

The baseline application packages should be mature products with wide usage in the life sciences industry. The User can, therefore, expect that the execution of standard functions within these packages is unlikely to produce incorrect results. The User should verify that the functions have been used in the correct manner and this may be done by code review and/or appropriate testing.

The products should have been used on a wide range of relevant hardware platforms and unless the User has an unusual hardware configuration then any risk likelihood of most hardware platforms giving rise to unexpected results is minimal.

The following illustrates the development life cycles for each of the relevant GAMP® categories. The User may need to consider (as part of the risk assessment) whether access to the Supplier development life cycle is possible. All decisions and any corresponding actions taken should be documented.

3.1.1 GAMP® Software Category 3 Desktop Applications

The following diagram highlights the different phases of validation required by the desktop applications Supplier and by the User, for GAMP® software category 3. The focus is on recording and verifying key configuration settings and performance or User Acceptance Testing against user requirements:
3.1.2 GAMP® Software Category 4 Desktop Applications

Note that the greater flexibility provided by many desktop applications may require detailed design specifications and tests even for software set up that could be defined as configurable. This is because configuration of the application is much more flexible than for most IT configurable systems.

The following diagram highlights the different phases of validation required by the desktop applications Suppliers and by the Users, for category 4:
**Figure E4.2: Desktop Package Development Life Cycle and User’s Testing – Software Category 4**

3.1.3 **GAMP® Software Category 5 Desktop Applications**

The following diagram highlights the different phases of validation required by the desktop applications Suppliers and by the Users, for category 5:

Typically, where custom (bespoke) software is required in order to customize a desktop application for use in a specific application, the full design and testing life cycle needs to be followed. Note that the Supplier is expected to fully test their core application following a full development life cycle and the User is expected to fully test their customization of the application (e.g., Visual Basic macros or modifications to the Suppliers core application software if allowed) following their own development life cycle.
3.2 Testing Responsibilities - Supplier and End User

The nature of desktop applications is such that they are likely to be developed for a specific use by either the User or a third party integrator, who may be part of the User’s own organization (e.g., an IT group). It is likely that neither the User nor the integrator will be able to access the package Supplier’s design, development and test data.

3.2.1 Validation Plan

The User should consider producing a Validation Plan to cover the specification, development and testing of the application.

The nature of these applications and the frequency in which they may be used are such that companies may consider producing a generic plan which helps define a consistent approach and guide Users in the correct processes necessary to ensure a final desktop application that meets Users requirements.

3.2.2 Design Requirements

Irrespective of the complexity of the application requirements, design specifications should be produced appropriate to the risk priority and complexity of the desktop application. These may be a very simple single document or a suite of documents for a more sophisticated and complex application. The document(s) should define the overall requirements and design approach in order that suitable Test Cases and Test Scripts can be defined. A risk assessment process may be used to ensure that all critical aspects of the Users requirements are suitably addressed.
3.2.3 Suppliers/Integrators Document Review and Assessment

Where a third party integrator is used the User should take suitable steps to ensure that the application is designed, developed, and tested using defined methodologies. This should fall in line with the requirements defined within GAMP® 4 (see Appendix G2, reference 1) for the category of software determined for the application.

Where the original desktop application Suppliers design, development, and test results are not accessible the User should consider this as part of a risk assessment and appropriate measures put in place. It may be that this risk assessment is done at a corporate level for particular packages with relevant steps defined for given packages and software categories.

Where a third party integrator is used the User should consider the maturity of the integrator and possibly an assessment of the integrators methodologies if a risk assessment determines this appropriate.

3.2.4 Design Review

The nature of the development toolset(s) for these packages may mean that design reviews are not feasible. The User should consider this when determining the level of testing to be applied to the application.

Where the application is complex (most likely to be software category 5) and/or high risk priority, the User may consider having a review of the custom application code to ensure that is produced following good engineering practices.

3.2.5 Test Specifications or Protocols, Execution and Reports

Test Specifications or Protocols should be produced, reviewed, and approved by relevant parties. The Test Cases and Test Scripts should be designed to test the application with particular attention to the necessary scope and nature and testing. Testing should be done in a controlled manner by suitably qualified personnel.

Test Results should be reviewed by suitably qualified personnel and a report produced confirming that all testing has been completed satisfactorily.

It may be useful to produce a final validation report which collates information as to the design, development and testing processes carried out and the final result of the validation exercise. For small, simple desktop applications it may possible to include test results in a single document containing all of the necessary Test Cases and Test Scripts.
Appendix E5 - Testing Infrastructure and Interfaces

For further guidance on the qualification of IT Infrastructure, refer to the GAMP® GPG: IT Infrastructure Control and Compliance (see Appendix G2, reference 5).

1 Definitions

The scope of IT Infrastructure is described more fully in the above referenced GAMP® Good Practice Guide, but it basically can be defined as those components of the IT landscape that support specific applications and/or are common to more than one application. IT Infrastructure may consist of hardware and/or software components.

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) then they cannot be justifiably claimed to be in common use. Hence the risk likelihood is more suitably addressed by classifying the component(s) as GAMP® hardware category 2.

1.2 Software Components

IT Infrastructure software components are usually commercially available off-the-shelf items and may be classified as GAMP® software category 1, 2, 3 or 4, depending upon the intended use. In some circumstances custom software may be developed as part of the IT infrastructure and this is classified as GAMP® software category 5.

However, most IT Infrastructure software has only an indirect impact upon 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.

This may include Middleware, which provides standard functionality across all areas of the business, both within the defined infrastructure boundary, but also outside the boundary. The use of standard Middleware now provides considerable basic functionality and this is often delivered via the organizations own Intranet or via a secure Extranet (operating across the wider Internet).

1.3 Interfaces

Interfaces allow separate applications or systems to exchange data, perform remote functionality or may support the interchange of data between different functions within the same application. Depending upon the nature of the software involved, these may also be categorized as GAMP® software category 2, 3, 4, or 5.

Although infrastructure and interfaces are not synonymous, interface software executes on hardware and the interchange of data may often be supported by components of the IT infrastructure. Interfaces may be implemented as part of the Infrastructure functionality, either as relatively simple devices or as items of configurable Middleware.

2 Testing and the GAMP® Life Cycle

Depending upon the categorization of the hardware and software involved, IT Infrastructure and Interfaces are subject to development and testing in accordance with an appropriate life cycle. For software components this should follow recognized stages of the GAMP® validation life cycle, including testing.

In most cases, where an application uses standard corporate intranet and/or internet infrastructure (in a client/server application for example), the network may in most cases they may be treated as a black box and does not have to
be tested in its own right. However, some applications may modify the use of the network (such as requiring specific TCP/IP stack settings) and may require an element of white box testing.

Testing may be conducted only up to the stage of Integration Testing, and any further acceptance or performance testing is usually conducted as part of the validation of the associated GxP applications.

2.1.1 Infrastructure Hardware Components

Infrastructure hardware components usually have either no impact or an indirect impact upon product quality and patient safety, although there may be potential for a direct impact upon data integrity. However, in some cases it is possible for the infrastructure to have higher risk impact (such as the use of network enabled remote diagnostics). Hardware components should, therefore, be qualified for use within the User organization.

These should nevertheless be subject to a full development and testing life cycle by the component developer.

Based upon the risk impact (which may be assumed to be high for infrastructure platforms that support all applications in the organization) the User’s internal IT Group may:

- Specify how such components should be implemented and qualified within the organization
- Define standard configuration set-ups for the hardware component
- Conduct additional integration and functional type testing in a test environment, where determined necessary by risk assessment

This is to ensure that the hardware components (when set-up as defined in the design specification) are compatible with other items of IT infrastructure used in the organization.

Such hardware components may then be installed directly in the production Environment without further testing, subject to installation qualification (verifying that the installed component is equivalent to components that have been type tested in a test environment, and that any component specific configuration parameters such as IP address have been set-up correctly).

For some hardware components (such as more complex devices such as routers or bridges), some functional testing may also be required to confirm the correct operation of the component. Further details of such tests are given in the GAMP® GPG: IT Infrastructure Control and Compliance (see Appendix G2, reference 5).

The following diagram shows a situation where an internal IT group is installing and qualifying standard components of IT infrastructure hardware (obtained from a Supplier) into the production environment.
2.1.2 Infrastructure Software Components

Items of Infrastructure Software may be used to provide standard functionality for use across multiple areas of the business and to support multiple GxP applications. These should be subject to a full development and testing life cycle by the component Supplier.

The User’s internal IT Group may:

- Specify how such components should be used within the organization
- Define standard configuration set-ups for the software component
- Conduct additional integration and functional type testing in a test environment

This is to ensure that the software component (when set-up as defined in the design specification) is compatible with other items of IT infrastructure used in the organization.

The User’s internal IT Group can then qualify the standard IT infrastructure software component for release to development teams for use with specific computerized systems (e.g., a standard operating system build). Such development teams will usually need to verify the configuration of the qualified software and conduct additional integration testing to ensure that the software component functions as specified with their specific application. An example of this is shown in the following diagram.
2.2 Testing Strategies

2.2.1 Test Planning (High Level Issues)

Infrastructure and Interface test planning should be used to define the nature and scope of specific tests. Depending upon their nature, interface, or infrastructure component type tests may be conducted at Supplier premises. Functional and performance testing may be conducted on the Users site where an interface is to an existing system or where the end-to-end operation needs to be qualified. The Test Plan or Strategy:

- Identifies the items of infrastructure, systems, applications and interfaces to be tested
- Identifies the testing stages that will be performed and the scope of each stage
- Identifies the test environment and resources
- Identifies the testing schedule and deliverables
- May lists Individual tests to be performed (or these may be defined in separate Test or Qualification Protocols or Specifications)
### 2.2.2 Determining the Nature and Scope of Infrastructure Testing

The following table summarizes the nature and scope of infrastructure testing and appropriate test coverage. Additional considerations with regards to the nature and scope of specific test types are given in the paragraphs following.

**Table E5.1: Infrastructure and Interface Test Phases and Types**

<table>
<thead>
<tr>
<th>Test Phase</th>
<th>Timing and Location</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infrastructure or Interface Hardware or</td>
<td>Usually conducted by the component Supplier as part of a defined development life</td>
<td>• Structural testing,</td>
</tr>
<tr>
<td>Software component Unit Testing</td>
<td>cycle. May be conducted by the User (or their integrator) where custom infrastructure</td>
<td>• Functional tests on custom software modules,</td>
</tr>
<tr>
<td></td>
<td>or interfaces are being developed.</td>
<td>• Interoperability and compatibility tests,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Challenge tests for likely defects such as data over-run/under-run, checksum errors, loss of physical connectivity, illegal data format, or value and loss of packet synchronization.</td>
</tr>
<tr>
<td>Infrastructure or Interface Hardware or</td>
<td>May be conducted by the component Supplier as part of interoperability testing of a</td>
<td>• Basic integration testing of custom hardware and software components.</td>
</tr>
<tr>
<td>Software component Integration Testing</td>
<td>product range. May be conducted by the Users (or their integrator) where custom</td>
<td>• Interoperability and compatibility tests.</td>
</tr>
<tr>
<td></td>
<td>infrastructure or interfaces are being developed.</td>
<td></td>
</tr>
<tr>
<td>Acceptance Testing</td>
<td>Usually conducted by the Users (or their integrator) in a test environment.</td>
<td>• Reliability of the interface or infrastructure component across a broad range of specified operating parameters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Data throughput testing up to and including maximum specified data throughput</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Error handling challenge testing to verify that the infrastructure or interface is robust, reacts to errors in a predictable manner, assures data integrity and (if required) degrades gracefully</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Basic connectivity testing, data routing and available bandwidth</td>
</tr>
<tr>
<td>Interoperability Testing (functional compatibility)</td>
<td>Conducted by the User. This will also typically include interoperability testing before new components (or versions) are approved for use in the production environment. May be required as part of standardizing the enterprise infrastructure architecture.</td>
<td>• Interoperability and compatibility tests,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Performance and challenge testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Regression testing of software or hardware versions where compatibility is unknown or unproven</td>
</tr>
</tbody>
</table>
A high level Infrastructure Test Plan or Strategy may also be used to document the default installation qualification and/or testing requirements for each IT infrastructure component. This should be based on stated assumptions with regards to the GxP Impact, Likelihood of Failure and Probability of Detection for each component (this usually assumes a ‘worst case’ of high GxP impact, allowing the component to support any GxP application).

A more detailed, instance specific risk assessment and Test Specification or Protocol would determine any deviation from the default installation qualification or testing requirements (for example, when infrastructure middleware provides GxP critical functions or where a hardware component is used outside it’s Supplier specified operating limits).

Consideration of the issues described above allows risk scenarios to be identified (these may not only be technical risk scenarios, but may also include procedural risk scenarios due to human error). Once identified, the nature and scope of any additional testing can be determined for each instance of a given IT infrastructure component type.

Where the risk priority associated with the infrastructure components is high, additional testing may also be justified. This may include stress testing and performance testing.

For custom software components it may be appropriate to conduct software module testing, as with any other component of GAMP® category 5 software. Additional integration testing (demonstrating that the custom software component operates as specified and designed within the overall infrastructure) may also be required. This testing will usually be conducted in a dedicated test environment.

Once successfully tested, custom infrastructure software components can then be released for use in the production environment.

### 3 Interface Testing

As with all other systems, any modular interface may contain a combination of software of different categories and should be appropriately tested. In some instances, the interface to one system or application may be configurable, but the interface to a different system may require the development of custom software.

A generic interface Test Plan or Strategy should include the identification of appropriate interface test types and generic interface test methodologies.

An interface specific Test Specification or Protocol should specify test types appropriate to the mitigation of specific risk scenarios.

The following are examples of what should be considered:

1. **GxP Impact of:**
   - Lost or late delivery of data
   - Data corruption at destination (both data content and status information)
   - Wrong configuration in any module or component,
   - Sequencing or synchronization issues
2. Likelihood of failure:
   - Extent of customization – review all modules
   - Operation within specified parameters
   - Complexity of process technology
   - Message sizing, scalability and volumes – potential for performance impacts
   - Mix of interfaces sharing a technology

3. Probability of detection – the diagnostic and monitoring capability of:
   - Source system
   - Intermediate software or Middleware
   - Destination system
   - Communication media employed infrastructure
   - The interface modules themselves, if not covered above

If any areas are indicated as high risk priority then this is where the testing effort should focus.

This should drive the level of verification and testing to apply to each module and to the end-to-end operation of the interface. Matching this with the coverage the Supplier has or will achieve helps define the areas the User testing should address.

Consider also the need for regression testing. Where configurable Middleware interfaces transfer data between multiple systems and applications it may be necessary to conduct regression testing on all interfaces managed by that Middleware when the Middleware is upgraded or new functionality is implemented. This is to ensure that the configuration of additional functionality has not adversely impacted the functionality or performance of existing interfaces.

The Users testing of an interface (or infrastructure) should follow the generic approach for Category 3, 4 and 5 applications with emphasis on the following areas:
   - Compatibility
   - Reliability (across the specified range of operating parameters)
   - Data throughput (performance)
   - Error handling (robustness)
   - Management of configuration driven interfaces

4  Testing and the Hardware/Software Type and Maturity

Infrastructure components and Interfaces implemented as part of the basic operating system (GAMP® software category 1) are not usually tested separately, but are tested as an inherent part of the infrastructure or application.

Other infrastructure components or standard software interfaces may be considered as GAMP® software category 3. Such interfaces are typically intended to allow a limited number of defined software applications to exchange data using standard data models and communications protocols. These may be described as ‘plug and play’ interfaces and require a minimum of configuration such as baud rate settings. These are typically technical and not functional parameters.

Two systems or applications with such ‘plug and play’ interface capability may be directly interfaced, or may be interfaced through an independent software (Middleware) component.

Where Infrastructure Middleware components or interface software is configured to provide interface functionality, this should be validated as GAMP® software category 4. Examples of these may be standard Applications Program
Interfaces (APIs) which require configuration parameters to be entered in order to define the required interface functionality, or configurable browser plug-ins providing Intranet functionality.

Such interface parameters may define more complex options such as polling frequency, data format, data item mapping, acceptable data values boundaries, default substitute values and log file names. Middleware parameters may include items such as security or privacy settings, network routing and firewall port settings.

Custom infrastructure components should be treated as GAMP® software category 5 and/or hardware category 2.

4.1 Testing Responsibilities - Supplier and User

The roles and responsibilities for IT infrastructure and interface testing depend upon the nature of the infrastructure and/or interface, and the nature of the contractual relationship between the User and the Supplier.

For many IT infrastructure software components and simple interfaces these are similar to the roles and responsibilities defined for the testing of applications software and systems and as defined in the various models in Appendices E1 to E4 of this Guide.

However, for most items of IT Infrastructure (including Middleware for interfaces) the roles and responsibilities across the Users organization also need to be considered. This complicates the ‘ownership’ of the Infrastructure – which is usually implemented, operated and maintained by the IT Group(s), but relied upon by the owners of GxP applications that are supported by such infrastructure.

“Ownership” and the responsibilities for the IT Infrastructure (including testing) should be clearly defined in Service Level Agreements established with the Users organization and any third party Suppliers (where appropriate). Further details of such agreements are discussed in the GAMP® GPG: IT Infrastructure Control and Compliance (see Appendix G2, reference 5).

Examples of the supply of such services are given in the examples for hardware and software infrastructure components above.

The User (their IT group) is usually responsible for periodically and proactively testing the effectiveness of IT security controls through vulnerability testing or monitoring and responding to security issues on a day-to-day basis. It is not unusual for such testing or monitoring to discover ‘security threats’ that are really vulnerabilities resulting from insufficient or inadequate project testing or problems with the configuration of the IT Infrastructure. Resolving these issues will certainly require configuration management records to be updated and may also require regression testing of any vulnerable applications.

Standard interfaces should be used wherever possible and these allow the User to leverage testing conducted by the Supplier as part of their development life cycle. Where possible, advantage should be taken of interfaces that have been independently certified for use as part of an industry recognized interfacing standard.

Failure to implement interfaces to an established technical standard is one of the most common reasons for interface problems. Where proof of independent certification is not available Users should ensure that the Suppliers interpretation and implementation of industry standards is complete and review the scope and nature of Suppliers testing to gain a high degree of assurance in the quality of such interfaces.

Users should ensure that their requirement specifications fully state the operational and functional requirements for any interfaces and that Suppliers assume responsibility to demonstrate achievement of those requirements for the elements of the interface they are providing. This is particularly important for custom interfaces and/or those with a high GxP impact.

Responsibility for the functional end-to-end testing of all interfaces also should be clearly defined and this usually rests with the User (the System or Application Owner). Where such responsibilities also extend to the Users IT Group(s) or external third party Suppliers this responsibility should be written into any contracts or Service Level Agreements.
Section IV – GENERAL APPENDICES
Appendix G1 - Definitions

1 Definition of Terms Used in this Document

Different terminology is in use within the pharmaceutical and other life science sectors, and within the general IT industry. This Guide provides a consistent set of testing terminology intended to facilitate understanding in testing environments.

General testing terms are consistent with those in use in GAMP® 4 (see Appendix G2, reference 1) and additional terms used throughout this Guide are defined below. Where alternative testing terminology is widely used in other industries such as the information technology (IT) and control and automation industries these are also referenced in this Guide.

It is recommended that a definition of consistent testing terms should be made on an organization basis or on a project basis where members of User and Supplier organizations are working together. It can be helpful if the definitions are agreed prior to contract signing, to ensure that contractual issues are based upon a common understanding of activities and milestones.

1.1 Terms and Definitions

<table>
<thead>
<tr>
<th>Terminology</th>
<th>Definition</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceptance Criteria</td>
<td>The criteria that a system or component must satisfy in order to be accepted by a user, customer or other authorized entity</td>
<td>GAMP® 4 (IEEE)</td>
</tr>
<tr>
<td>Acceptance Test</td>
<td>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).</td>
<td>GAMP® 4 (IEEE)</td>
</tr>
<tr>
<td>Black Box Testing</td>
<td>See Functional Testing</td>
<td>IEEE</td>
</tr>
<tr>
<td>Boundary Condition Testing</td>
<td>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.</td>
<td>From IEEE definition of Boundary Condition</td>
</tr>
<tr>
<td>Calibration</td>
<td>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.</td>
<td>GAMP® 4 (ISO 10012)</td>
</tr>
<tr>
<td>Challenge Testing</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>Commissioning</td>
<td>The process of providing to the appropriate components, the information necessary for the designed communication between components</td>
<td>IEEE</td>
</tr>
<tr>
<td>Emulation</td>
<td>A model that accepts the same inputs and produces the same outputs as a given system.</td>
<td>IEEE</td>
</tr>
<tr>
<td>Terminology</td>
<td>Definition</td>
<td>Source</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Environmental Testing</td>
<td>Testing that evaluates system or component performance up to the specified limits of environmental parameters (for example temperature or humidity).</td>
<td></td>
</tr>
<tr>
<td>Firmware</td>
<td>The combination of hardware device and computer instructions and data that reside as read-only software on that device.</td>
<td>IEEE</td>
</tr>
<tr>
<td>Factory Acceptance Test (FAT)</td>
<td>An Acceptance Test in the Supplier’s factory, usually involving the customer. See also Acceptance Test. Contrast to Site Acceptance Test.</td>
<td>GAMP® 4 (IEEE)</td>
</tr>
<tr>
<td>Functional Testing</td>
<td>Testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected input and execution conditions. Also known as black box testing.</td>
<td>GAMP® 4 (IEEE)</td>
</tr>
<tr>
<td>Hardware</td>
<td>(1) Physical equipment used to process, store, or transmit computer programs or data. (2) Physical equipment used in data processing, as opposed to programs, procedures, rules, and associated documentation.</td>
<td>IEEE</td>
</tr>
<tr>
<td>Hardware Testing</td>
<td>Testing carried out to verify correct operation of system hardware independent of any custom application software.</td>
<td></td>
</tr>
<tr>
<td>Installation Qualification [IQ]</td>
<td>Documented verification that a system is installed according to written and pre-approved specifications.</td>
<td>GAMP® 4 (PDA)</td>
</tr>
<tr>
<td>Integration</td>
<td>The process of combining software components, hardware components or both into an overall system. Sometimes described as software integration and system integration respectively.</td>
<td>IEEE</td>
</tr>
<tr>
<td>Integration Testing</td>
<td>(1) Testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them. (2) An orderly progression of testing of incremental pieces of the software program in which software elements, hardware elements or both are combined and tested until the entire system has been integrated to show compliance with the program designed, and capabilities and requirements of the system.</td>
<td>IEEE</td>
</tr>
<tr>
<td>Instance</td>
<td>A single installation of a software application (plus associated databases, tools and utilities). Usually applied to configurable IT systems.</td>
<td></td>
</tr>
<tr>
<td>Load Testing</td>
<td>As stress testing but to evaluate a system or component up to or at the limits of its specified requirements.</td>
<td></td>
</tr>
<tr>
<td>Loop Testing</td>
<td>Testing in which control system inputs and outputs are exercised and their functionality verified.</td>
<td></td>
</tr>
<tr>
<td>Market Requirements Specification</td>
<td>A statement of generic industry requirements used by the Supplier as an input to their product development life cycle.</td>
<td></td>
</tr>
<tr>
<td>Terminology</td>
<td>Definition</td>
<td>Source</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Middleware</td>
<td>The hardware, computer instructions, and data which provide the infrastructure used by other system modules.</td>
<td></td>
</tr>
<tr>
<td>Module Testing</td>
<td>Testing of an individual hardware or software components or groups of related components.</td>
<td>IEEE</td>
</tr>
<tr>
<td>Negative Testing</td>
<td>Testing aimed at showing that software does not work.</td>
<td>BCS</td>
</tr>
<tr>
<td>Operational and Support Testing</td>
<td>(1) Testing conducted to evaluate a system or component in its operational environment.</td>
<td>IEEE</td>
</tr>
<tr>
<td></td>
<td>(2) All testing required to verify system operation in accordance with design requirements after the major component is energized or operated.</td>
<td></td>
</tr>
<tr>
<td>Operational Qualification [OQ]</td>
<td>Documented verification that a system operates according to written and pre-approved specifications throughout all specified operating ranges.</td>
<td>GAMP® 4 (PDA)</td>
</tr>
<tr>
<td>Performance Qualification [PQ]</td>
<td>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, whilst operating in its specified operating environment.</td>
<td>GAMP® 4 (PDA)</td>
</tr>
<tr>
<td>Positive Testing</td>
<td>Testing aimed at showing that software does meet the defined requirements.</td>
<td></td>
</tr>
<tr>
<td>Qualification</td>
<td>The Process to demonstrate the ability to fulfill specified requirements</td>
<td>GAMP® 4 (ISO)</td>
</tr>
<tr>
<td>Simulation</td>
<td>A model that behaves or operates like a given system when provided a set of given inputs.</td>
<td>IEEE</td>
</tr>
<tr>
<td>Site Acceptance Test (SAT)</td>
<td>An Acceptance Test at the customer’s site, usually involving the customer. See also Acceptance Test. Contrast to Factory Acceptance Test</td>
<td>GAMP® 4 (IEEE)</td>
</tr>
<tr>
<td>Software</td>
<td>Computer programs, procedures, and associated documentation and data pertaining to the operation of a computer system.</td>
<td>IEEE</td>
</tr>
<tr>
<td>Stress Testing</td>
<td>Testing conducted to evaluate a system or component at or beyond the limits of its specified requirements.</td>
<td>IEEE</td>
</tr>
<tr>
<td>Structural Testing</td>
<td>Examining the internal structure of the source code. Includes low-level and high-level code review, path analysis, auditing of programming procedures, and standards actually used, inspection for extraneous “dead code”, boundary analysis and other techniques. Requires specific computer science and programming expertise. Also known as White Box Testing</td>
<td>GAMP® 4 (Bluhm, Meyers, Hetzel)</td>
</tr>
<tr>
<td>Supplier</td>
<td>Organization or a person that provides a product</td>
<td>GAMP® 4 (ISO)</td>
</tr>
<tr>
<td>Terminology</td>
<td>Definition</td>
<td>Source</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>System Testing</td>
<td>Testing conducted on a complete, integrated system to evaluate the systems compliance with its specified requirements.</td>
<td>IEEE</td>
</tr>
<tr>
<td>Test</td>
<td>(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) Determination of one or more characteristics according to a procedure</td>
<td>GAMP® 4 (IEEE) GAMP® 4 (ISO)</td>
</tr>
<tr>
<td>Test Case</td>
<td>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</td>
<td>GAMP® 4 (IEEE)</td>
</tr>
<tr>
<td>Test Plan</td>
<td>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</td>
<td>GAMP® 4 (IEEE)</td>
</tr>
<tr>
<td>Test procedure</td>
<td>Detailed instructions for the set-up, execution, and evaluation of results for a given test case</td>
<td>GAMP® 4 (IEEE)</td>
</tr>
<tr>
<td>Test Protocol</td>
<td>See Test Specification</td>
<td></td>
</tr>
<tr>
<td>Test script</td>
<td>Documentation that specifies a sequence of actions for the execution of a test.</td>
<td>IEEE</td>
</tr>
<tr>
<td>Test Specification</td>
<td>A document that describes the scope, management, use of procedures, sequencing, test environment, and prerequisites for a specific phase of testing.</td>
<td></td>
</tr>
<tr>
<td>Test Strategy</td>
<td>See Test Plan</td>
<td></td>
</tr>
<tr>
<td>Unit Testing</td>
<td>Testing of individual hardware or software units or groups of related units.</td>
<td>IEEE</td>
</tr>
<tr>
<td>Usability Testing</td>
<td>Testing the ease with which users can learn and use a product.</td>
<td>BCS</td>
</tr>
<tr>
<td>User</td>
<td>The person or persons who operate or interact directly with the system.</td>
<td>GAMP® 4</td>
</tr>
<tr>
<td>Validation</td>
<td>Establishing documented evidence which provides a high degree of assurance that a specific process will consistently produce a product meeting its pre-determined specifications and quality attributes.</td>
<td>GAMP® 4 (FDA)</td>
</tr>
<tr>
<td>Verification</td>
<td>Confirmation, through the provision of objective evidence that specified requirements have been fulfilled.</td>
<td>GAMP® 4 (ISO)</td>
</tr>
<tr>
<td>White Box Testing</td>
<td>See Structural Testing.</td>
<td>IEEE</td>
</tr>
</tbody>
</table>
GAMP® 4  = Definition from the GAMP® Guide for the Validation of Automated Systems.
BCS    = Definition from Working Draft: Glossary of terms used in software testing Version 6.2 produced by the British Computer Society Specialist Interest Group in Software Testing (BCS SIGIST).
Appendix G2 - Bibliography

14. Software Inspection, Tom Gilb & Dorothy Graham. Addison-Wesley
Appendix G3 – List of Figures

Figure 2.1: The Use of Risk Assessment in Determining the Scope and Rigor of Testing ........................................... 14
Figure 2.2: GAMP® ‘V’ Model ................................................................................................................................. 15
Figure 2.3: ‘V-model’ framework showing basic Specification and Test Relationships .............................................. 16
Figure 2.4: Basic Test Documentation Model ........................................................................................................... 18
Figure 2.5: Test Framework for GAMP® Software Category 2 ................................................................................ 22
Figure 2.6: Test Framework for GAMP® Software Category 3 ................................................................................ 23
Figure 2.7: Test Framework for GAMP® Software Category 4 ................................................................................ 24
Figure 2.8: Test Framework for GAMP® Software Category 5 ................................................................................ 25
Figure 2.9: Test Framework for GAMP® Software Categories 4 and 5 (combined) ................................................... 26
Figure C1.1: Supplier and Product Maturity Model .................................................................................................... 31
Figure C1.2: Relative Users Test Burden when using Preferred versus Least Preferred Solutions ........................ 33
Figure C1.3: Test Framework for GAMP® Software Category 4 with shared System Integrator and User Responsibilities ........................................................................................................ 34
Figure T1.1: Example of Test Scope Defined in Risk Mitigation Strategy ............................................................... 41
Figure T1.2: Performance Testing in System Life Cycle ............................................................................................ 45
Figure T1.3: Load and Stress Test Profile ................................................................................................................ 46
Figure T1.4: Excerpt from Typical Load Test Script ................................................................................................. 48
Figure T2.1: Example of Test Process Flow and Associated Roles .......................................................................... 64
Figure T3.1: Test Script and Test Case Combined, Integral Test Results ................................................................. 76
Figure T3.2: Test Script and Test Case Combined, Separate Results Sheet ............................................................ 77
Figure T3.3: Separate Test Script and Test Case, Integral Test Results ................................................................. 78
Figure T3.4: Separate Test Script and Test Case, Separate Results Sheet ............................................................. 79
Figure T6.1: Example of Corrected Test Result ........................................................................................................ 92
Figure E1.1: Process Control System Types ............................................................................................................ 100
Figure E1.2: V-model’ Framework for the Specification and Testing of Process Control Systems ....................... 102
Figure E1.3: Process Control System functionality ................................................................................................ 106
Figure E1.4: Functionality Contained in a Simple Process Control System .......................................................... 108
Figure E1.5: Software Elements Contained Within a Simple Process Control System ........................................ 108
Figure E1.6: Example of System Elements Contained Within an PLC ................................................................. 109
Figure E1.7: Example of System Elements Contained Within an Operator Panel .................................................. 110
Figure E1.8: Example of System Elements Contained Within an Electronic Chart Recorder .............................. 111
Figure E1.9: Example of System Software Elements Contained Within an Embedded Control System ............ 112
Figure E1.10: Example of Stand-alone Control System (e.g., Distributed Control) ........................................... 113
Figure E1.11: Example of System Software Elements Contained Within a Distributed Control System .......... 114
Figure E1.12: Development and Test Life Cycle For A New Embedded Control System .................................. 115
Figure E1.13: Development and Test Life Cycle For A New Embedded Control System .................................. 116
Figure E2.1: Typical Configurable IT System Double “V” Model ........................................................................... 118
Figure E2.2: Typical ERP Overview ......................................................................................................................... 122
Figure E2.3: Simple Test Approach Aligned with Functional Overview .............................................................. 123
Figure E2.4: Cross-Module Test Approach ............................................................................................................. 124
Figure E2.5: Typical Configurable IT System Project Components ...................................................................... 125

For ISPE Members only. Copyright ISPE 2005.
Appendix G4 – List of Tables

Table 2.1: System Types and Associated Appendices ......................................................................................................................... 16
Table T1.1: Recommended Test Types Based upon Risk Assessment ........................................................................................................ 39
Table T1.2: Examples of Loading Requirements .................................................................................................................................................. 46
Table T1.3: Examples of System Parameters to Monitor During Load Testing ........................................................................................................ 47
Table T3.1: Equivalence Class Assumptions for Structural Testing ............................................................................................................... 70
Table T3.2: Equivalence Class Assumptions for Functional Testing .................................................................................................................. 71
Table T3.3: Test Case Design Considerations .................................................................................................................................................. 71
Table T3.4: Test Script Content for Different Test Step Types ....................................................................................................................... 74
Table T4.1: Examples of Hardware Test Environments ................................................................................................................................. 81
Table T4.2: Examples of Software Test Environments ................................................................................................................................. 81
Table E1.1: Typical Test Phases for Process Automation Systems .................................................................................................................. 103
Table E1.2: Examples of Test Phases for Different Process Control Software Category .......................................................................................... 106
Table E1.3: Functionality by Software Type – Simple Process Control System ................................................................................................. 108
Table E1.4: Functionality by Software Type – PLC ........................................................................................................................................... 109
Table E1.5: Functionality by Software Type – Operator Panel .......................................................................................................................... 110
Table E1.6: Functionality by Software Type – Electronic Chart Recorder .......................................................................................................... 111
Table E1.7: Functionality by Software Type – Distributed Control System .................................................................................................... 113
Table E2.1: Test Types for Configurable IT Systems ........................................................................................................................................ 120
Table E3.1: Analytical Instrument Categorization ........................................................................................................................................ 129
Table E3.2: Applicable Test Types for Category D/E Laboratory Systems ...................................................................................................... 133
Table E3.3: Applicable Test Types for Category F Laboratory Systems ........................................................................................................ 134
Table E3.4: User and Supplier Testing Responsibilities – Category D/E Laboratory Systems ................................................................. 135
Table E3.5: User and Supplier Testing Responsibilities – Category F Laboratory Systems ........................................................................................................ 136
Table E4.1: Desktop Applications – Test Types ............................................................................................................................................ 139
Table E5.1: Infrastructure and Interface Test Phases and Types ..................................................................................................................... 149