



Testing IT

An Off-the-Shelf Software Testing Process

by John Watkins

Foreword by Maurice Rozenberg,
author of *Test logiciel*

- All you need to know about testing software
- Covers object-oriented applications
- Includes case studies
- Includes customizable forms and templates

This page intentionally left blank

Testing IT

Testing IT

An Off-the-Shelf
Software Testing
Process

John Watkins



CAMBRIDGE
UNIVERSITY PRESS

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS

The Edinburgh Building, Cambridge CB2 2RU, UK
40 West 20th Street, New York, NY 10011-4211, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain
Dock House, The Waterfront, Cape Town 8001, South Africa

<http://www.cambridge.org>

© Cambridge University Press 2004

First published in printed format 2001

ISBN 0-511-03206-4 eBook (Adobe Reader)

ISBN 0-521-79546-X paperback

To Francesca, Julie, and Valerie



Contents

<i>Foreword</i>	<i>page xiii</i>
<i>Acknowledgments</i>	<i>xv</i>
1 Introduction	1
1.1 <i>Purpose of the Book</i>	<i>1</i>
1.2 <i>Readership</i>	<i>2</i>
1.3 <i>How to Read This Book</i>	<i>2</i>
1.4 <i>Structure and Content of This Book</i>	<i>3</i>
 Part 1	
The Traditional Testing Process	5
 2 An Overview of Testing	7
2.1 <i>Introduction</i>	<i>7</i>
2.2 <i>The Challenge of Testing</i>	<i>7</i>
2.3 <i>What Is Testing?</i>	<i>8</i>
2.4 <i>Verification and Validation</i>	<i>10</i>
2.5 <i>What Is the Cost of Not Testing?</i>	<i>11</i>
2.6 <i>Testing – The Bottom Line</i>	<i>11</i>
2.7 <i>Additional Information</i>	<i>12</i>
 3 Testing Techniques	15
3.1 <i>Introduction</i>	<i>15</i>
3.2 <i>General Testing Techniques</i>	<i>16</i>
3.3 <i>Functional Testing Techniques</i>	<i>19</i>
3.4 <i>Nonfunctional Testing Techniques</i>	<i>22</i>
3.5 <i>Further Reading on Testing Techniques</i>	<i>26</i>
 4 The Management and Planning of Testing	27
4.1 <i>Introduction</i>	<i>27</i>
4.2 <i>The Organization of Testing</i>	<i>28</i>
4.3 <i>Roles and Responsibilities</i>	<i>29</i>
4.4 <i>The Testing Phases</i>	<i>34</i>
4.5 <i>Role of the V Model in Planning</i>	<i>39</i>

4.6	<i>The Management of Test Requirements</i>	41
4.7	<i>The Role and Use of Configuration Management</i>	42
4.8	<i>The Role and Use of Defect Tracking</i>	42
4.9	<i>The Role of Risk in Test Planning and Management</i>	43
5	Unit Testing	45
5.1	<i>Overview</i>	45
5.2	<i>Unit Test Approach</i>	47
5.3	<i>Unit Test Data Requirements</i>	47
5.4	<i>Roles and Responsibilities</i>	48
5.5	<i>Planning and Resources</i>	49
5.6	<i>Inputs</i>	50
5.7	<i>Testing Techniques for Unit Testing</i>	50
5.8	<i>Outputs</i>	50
6	Integration Testing	53
6.1	<i>Overview</i>	53
6.2	<i>Integration Test Approach</i>	54
6.3	<i>Integration Test Data Requirements</i>	55
6.4	<i>Roles and Responsibilities</i>	55
6.5	<i>Planning and Resources</i>	56
6.6	<i>Inputs</i>	57
6.7	<i>Testing Techniques for Integration Testing</i>	57
6.8	<i>Outputs</i>	57
7	System Testing	59
7.1	<i>Overview</i>	59
7.2	<i>System Test Approach</i>	60
7.3	<i>System Test Data Requirements</i>	60
7.4	<i>Roles and Responsibilities</i>	61
7.5	<i>Planning and Resources</i>	62
7.6	<i>Inputs</i>	63
7.7	<i>Testing Techniques for System Testing</i>	63
7.8	<i>Outputs</i>	63
8	Systems Integration Testing	65
8.1	<i>Overview</i>	65
8.2	<i>Systems Integration Test Approach</i>	66
8.3	<i>Systems Integration Test Data Requirements</i>	67
8.4	<i>Roles and Responsibilities</i>	67
8.5	<i>Planning and Resources</i>	68

8.6	<i>Inputs</i>	69
8.7	<i>Testing Techniques for Systems Integration Testing</i>	69
8.8	<i>Outputs</i>	70
9	User Acceptance Testing	73
9.1	<i>Overview</i>	73
9.2	<i>User Acceptance Test Approach</i>	74
9.3	<i>User Acceptance Test Data Requirements</i>	75
9.4	<i>Roles and Responsibilities</i>	75
9.5	<i>Planning and Resources</i>	76
9.6	<i>Inputs</i>	77
9.7	<i>Testing Techniques for User Acceptance Testing</i>	78
9.8	<i>Outputs</i>	78
10	Operations Acceptance Testing	81
10.1	<i>Overview</i>	81
10.2	<i>Operations Acceptance Test Approach</i>	82
10.3	<i>Operations Acceptance Test Data Requirements</i>	83
10.4	<i>Roles and Responsibilities</i>	84
10.5	<i>Planning and Resources</i>	85
10.6	<i>Inputs</i>	86
10.7	<i>Testing Techniques for Operations Acceptance Testing</i>	86
10.8	<i>Outputs</i>	86
11	Regression Testing	89
11.1	<i>Overview</i>	89
11.2	<i>Regression Test Approach</i>	90
11.3	<i>Regression Test Data Requirements</i>	91
11.4	<i>Roles and Responsibilities</i>	92
11.5	<i>Planning and Resources</i>	92
11.6	<i>Inputs</i>	94
11.7	<i>Testing Techniques for Regression Testing</i>	94
11.8	<i>Outputs</i>	94
12	Improving the Testing Process	97
12.1	<i>Introduction</i>	97
12.2	<i>Overview of the Role and Use of Metrics</i>	98
12.3	<i>Metrics Typically Used within the Testing Process</i>	99
12.4	<i>Setting Up and Administering a Metrics Program</i>	102
12.5	<i>A Proposal for a Simple and Effective Metrics Set</i>	104
12.6	<i>Further Reading</i>	106

13 Introduction, Adoption, and Maintenance of the Testing Process	107
13.1 Introduction	107
13.2 Introduction and Adoption of a Testing Process	107
13.3 Maintenance of the Testing Process	111

Part 2

The Testing Process in the Real World: Illustrative Case Studies

113

14 Case Study 1: The British Library	121
15 Case Study 2: Reuters Product Acceptance Group	129
16 Case Study 3: Crown Quality Assurance Group	143
17 Case Study 4: The Wine Society	155
18 Case Study 5: Automatic Data Processing Limited	167

Part 3

The Appendices

181

A Terms of Reference for Testing Staff	183
B Testing Guides	197
C Test Plan Document Template	211
D Test Specification Document Template	221
E Test Script Template	231
F Test Result Record Form Template	237
G Test Log Template	241
H Test Certificate Template	245
I Re-use Pack Checklist	247

J	Test Summary Report Template	249
K	Equivalence Partition Example	255
L	Boundary Analysis Example	257
M	State Transition Example	259
N	Automated Testing Tool Selection Criteria	261
O	Usability Testing Overview	275
P	Testing Process Health Check	279
Q	The Testing of Object-Oriented Software	287
	<i>References</i>	<i>295</i>
	<i>Glossary</i>	<i>297</i>
	<i>Index</i>	<i>311</i>



Foreword

Why is astronomy considered a science while astrology is considered only a pseudo-science? In other words, how can we prove that a theory faithfully describes reality, and that this theory can then be used to predict unknown facts? Karl Popper, the well-known philosopher, studied these problems and summarized his conclusions in one phrase: “The criterion of the scientific status of a theory is its falsability, or refutability, or testability.”* For Popper, “confirming evidence should not count except when it is the result of a genuine test of the theory.”

The testing process of a scientific theory is quite similar to the process of providing confirmation either to risky predictions or to attempts to falsify that theory. Testing is a complex activity. It has to simultaneously bear in mind the theory and the external reality; it has to provide objective answers to complex questions related to our own perceptions of a rational reality.

When developing software, we follow the same thought process, since one builds an abstract model between the external world and the user. In our software, we define strict processes that will guide our actions, and we build the data we want to manipulate in complex databases and templates.

Can we test our software with Popper’s principles in mind? The answer is definitively yes, because software testing should not only be a confirmation that the application is working correctly but also that it will react correctly when unexpected conditions occur. This constant and complex relationship between the software one tests and external reality should guide testers in their daily work.

Although testing is usually perceived as a necessity in software development, it is rarely applied as a rigorous activity. Within many projects, testing is simply omitted; in others, it is executed with the distinct intent to prove that the application performs correctly under test conditions.

After reading John Watkins’s *Testing IT* you will be convinced that testing is not as complex as it seems and that it can be managed like any other development activity.

The first thing you notice in reading *Testing IT* is that John puts the accent on testing processes and real-world case studies, which are, in my opinion, the most important aspects of software testing, implicitly applying Popper’s conclusions.

Testing IT is divided into three logical, distinct parts: Part 1 focuses on traditional testing processes. Although technology is evolving at lightening speed, processes remain. They become even more important because they are at the heart of any activity. You will find this part very useful since it presents the testing phases starting from unit testing up to regression testing in the order found on all projects. Each phase is presented in the same coherent plan, facilitating access to the information.

*Karl Popper, *Conjectures and Refutations*. London: Routledge and Kegan Paul, 1963.

Part 2 gives practical case studies. Five projects are reported, each enhancing a different reality; we have again the confirmation that success is always related to a correct adaptation of processes to reality.

Part 3 presents ready-to-use templates and reference documents that you can adapt to your needs and that you will find very useful in your daily testing work.

Testing IT is not just another book on testing. It is, in addition, a guide for all testers who want to understand what testing is really about, and it proves once more that applying easy-to-understand processes is the key to success. In one word: indispensable.

I'm certain you will enjoy reading this book, as I did, and that you will keep it on your desk to profit in your daily work from John's rich experience.

Maurice Rozenberg, Paris
Author of *Test Logiciel* (1998, Eyrolles)



Acknowledgments

I would very much like to thank the following people for their advice, assistance, and encouragement with the writing of this book: Martin Adcock, Martin Allen, Steve Allott, Jo Armitage, Mike Ashton, Chris Ball, Bob Bartlett, Clive Bates, Judith Bays, Colin Bendall, Lesley Beasant, Duncan Briggins, Kevin Buchta, Miriam Bromnick, Dorothy Buck, Dave Burgin, Yann Cluchey, Jon Collins, Trevor J. Collins, Richard Coop, Victor Copeland, Chris Cupit, Jim Davies, Steve Dawson, Barbara Eastman, Will Eisner, Isabel Evans, Mark Fewster, Virpi Flyktman, Ian Francis, Iain Gavin, Ian George, Sam Guckenheimer, Jon Hall, Marion Hampson, Steve Hancock, Richard Hands, Steve Harcombe, Kim Harding, David Hayman, Mark Hirst, Dave Hodgkinson, Alan and Alison Jones, John Kent, Shelagh Kidd, Sami Korppi, Shaun Legeyt, Carol Li, Neil Love, Hatty and Nick Luft, Phillip March, Andrew Martin, Elaine Mathews, Peter McConnell, Alec MuCutcheon, Aidus McVeigh, Kevin Minier, Ian Murphy, Eric Nelson, Mike Newton, Tommi Oksanen, David “Saint” Palfreeman, Richard Pollard, Andy Redwood, Susan and Andy Roach, Angelina Samaroo, Ranj Sekhon, Darran Shaw, Graham Shore, David Shohet, Peter Terelak, Pete Thomas, Andrew Thompson, John Thompson, Richard Tinker, Paul Warne, Julie Valentine, Tony Wells, Martin Whitaker, and Helen White.

I would like to give particular thanks to Dorothy Graham for the benefit of her extensive experience in the testing field; Graham Titterington for his comprehensive and informed review comments; Geoff Quentin for his testing insight, encyclopedic knowledge of testing standards, and extensive collection of testing references; and James Bach, Paul Gerrard, and Steve Hancock for their input and informed discussion regarding risk-based testing issues.

I would also like to thank the following people for their assistance in writing the case studies in Part Two of this book, as well as for their general help and encouragement with the book and its contents: Morag Atkins, Ken Eves, Paul Goddard, Martin Kemble, David Marsh, Annette Philips, and James Toon.

I am also very grateful to Maurice Rozenberg for finding the time to write the Foreword, giving me the benefit of his extensive testing expertise in reviewing the chapters, case studies, and appendices, and giving me a signed copy of his testing book – *Test Logiciel*¹ (I will have to brush up on my French now so that I can actually read it!).

And last but certainly not least, I would like to express my appreciation for the insight and experience of my technical reviewer Kamesh Pemmaraju and for the constant “encouragement” and guidance from my editor Lothlorien Homet.

¹Maurice tells me (winking) that this is the only French text on testing because the French do not make mistakes, so why would they need to test anything?

Testing IT

Introduction

Hmm, The Wheel you say! Well, I don't wish to belittle your achievement, but I've traveled far and wide and I've seen a great many of these things invented by a great many people in a great many different caves!

Big Ugg, Neander Valley, 35,000 B.C.

1.1 Purpose of the Book

This book provides comprehensive and rigorous guidance to workers in the field of software testing for researching or setting up a software testing process within organizations.

The book provides advice and guidance on all aspects of the testing process, including:

- ▲ The need to test software and the approach to testing
- ▲ Specific details of testing techniques with worked examples
- ▲ The planning and management of testing projects
- ▲ Testing roles and responsibilities
- ▲ Comprehensive details of the testing phases
- ▲ Extensive testing document templates, proformas, and checklists
- ▲ Recommendations for testing process improvement and the role and use of metrics
- ▲ The testing issues facing developers of Object-Oriented and Component-Based systems.

The book covers the testing of software from a number of sources, including software developed or modified in-house, software that represents the modification or extension of existing legacy software systems, and software developed on behalf of an organization by a third party.

The book also covers the acceptance testing of *commercial off-the-shelf (COTS)* software procured by an organization, or COTS software that has undergone development either internally or by a third party on behalf of an organization.

This book should be used in a pragmatic manner, in effect providing a testing framework that can be used by all members of staff involved in software development and testing within an organization to improve the quality of the software they deliver and to reduce timescales, effort, and cost of testing.

Alternatively, the testing process described in this book can be customized to match the specific testing requirements of any particular organization, and a series of real-world case studies are provided to illustrate how this can be achieved.

1.2 Readership

The target audience for this book includes the following people:

- ▲ **Technical Director/Managers** who need to improve the software testing process within their organization (in terms of quality, productivity, cost, and/or repeatability of the process)
- ▲ **Quality Assurance (QA) professionals** (such as company QA Directors or Managers) who need to put in place a formal organization-wide approach to software testing
- ▲ **Project Managers/Leaders** who need to save time, effort, and money and improve quality by adopting a complete, standard, off-the-shelf solution to their testing requirements
- ▲ **Independent Information Technology (IT), QA, or Management Consultants** who provide advice and guidance to clients on their software testing process, for whom the book will represent a key item in their “Consultants Tool Kit”
- ▲ **Testing/QA Professionals** (such as Test Analysts, Testers, or QA Representatives) who wish to save time and effort by adopting predefined testing artifacts (such as standard templates for Test Script, Test Plan, and Test Specification documents)
- ▲ **IT Professionals** who need to understand the software testing process (such as developers involved in Unit or Integration testing)
- ▲ **Any staff members** who are keen to improve their career prospects by advocating a complete testing solution to their organizations’ software testing needs, particularly where there is a need to improve quality or save time, effort, and cost
- ▲ **Training Managers/Trainers** who are in the process of writing or amending testing training materials and who need to obtain a pragmatic view of the testing process and its application
- ▲ **Students** who need to obtain a pragmatic/real-world view of the application of testing theory and principles to organizational software testing requirements, or who have an interest in testing-process improvement and the role and use of metrics.

1.3 How to Read This Book

This book is divided into three parts, all closely linked, but each of which can be read and applied separately.

Part 1 (Chapters 2–13) documents the “traditional view” of the components comprising a software testing process. Part 1 provides detailed information that can be used as the basis for setting up a testing-process framework tailored to the individual requirements of any organization involved in software testing.

Part 2 (Chapters 14–18) provides a series of case studies that show how a number of organizations have implemented their own testing process based on the “classic view” described in Part 1. These case studies can be read to provide real world

guidance on how an individual organization can implement a testing-process framework to meet its own testing requirements.

Part 3 (the appendices) contains a set of standard testing document templates, proformas, and checklists plus a number of appendices that expand on topics described in passing in the main body of the book. The standard testing document templates, proformas, and checklists are also available from the following URL: us.cambridge.org/titles/052179546X so that they can be used immediately without modification or customized to reflect the particular requirements of any organization (such as a corporate style, branding, or documentation standard).

Terms in *italics* are fully defined in the glossary.

1.4 Structure and Content of This Book

Specifically, the chapters and appendices comprising this book are:

- ▲ Chapter 2, which discusses just how challenging it is to thoroughly test even the most simple software system, reviews a number of definitions of testing, provides a brief overview of the approach to software testing, and lists definitive testing references for further reading.
- ▲ Chapter 3, which describes the principal techniques used in designing effective and efficient tests for testing software systems and provides, where appropriate, references to illustrative worked examples in the appendices.
- ▲ Chapter 4, which deals with the issues associated with the management and planning of the testing process, provides guidance on the organization of testing and testing projects and on the need for thorough planning, describing a number of techniques for supporting the planning process.
- ▲ Chapters 5–11, which provide details on each of the testing phases (from Unit Testing to Acceptance Testing and on to Regression Testing¹) and their interrelationships. Each chapter is presented in a standard format and covers:
 - *the overall testing approach for that phase*
 - *test data requirements for that phase*
 - *the roles and responsibilities associated with that phase*
 - *any particular planning and resourcing issues for that phase*
 - *the inputs to and the outputs from that phase*
 - *a review of the specific testing techniques that are appropriate to that phase.*
- ▲ Chapter 12 considers the need for process improvement within the testing process and reviews the role of metrics (proposing a pragmatic metrics set that can be used effectively within and across testing projects). It also provides references to further sources of information on test process improvement.

¹While not strictly speaking a separate testing phase, Regression Testing is included in this list for the sake of completeness.

- ▲ Chapter 13, which for organizations adopting the testing process described in this book or using it as the basis for setting up their own testing-process framework, discusses the approach to introducing testing into an organization and managing its successful adoption, reviews the need to maintain that testing process, and proposes an approach to satisfy this requirement.
- ▲ Chapters 14–18 provide a series of real-world case studies describing how a number of commercial organizations have implemented their own customized view of the testing process described in Chapters 2–13. Specifically, the organizations covered in the case studies are:
 - *The British Library*
 - *Reuters Product Acceptance Group*
 - *Crown Quality Assurance Group*
 - *The Wine Society*
 - *Automatic Data Processing (ADP) Limited.*
- ▲ Appendices A–J provide a set of testing document templates, proformas, and checklists:
 - *terms of reference for testing staff*
 - *summary testing guides for each testing phase*
 - *a Test Plan document template*
 - *a Test Specification document template*
 - *a Test Script template*
 - *a Test Result Record Form template*
 - *a Test Log template*
 - *a Test Certificate template*
 - *a Re-use Pack checklist*
 - *a Test Summary Report template.*
- ▲ Appendices K–M present a series of worked examples of testing techniques described in Chapter 3.
- ▲ Appendices N–Q expand on topics described in passing in the main body of the book and include:
 - *a scheme and set of criteria for evaluating the relative merits of commercially available automated software testing tools*
 - *an overview of the process of Usability Testing and its application*
 - *a scheme and set of criteria for performing an audit of a testing process*
 - *a discussion of the issues involved in the testing of object-oriented and component-based applications.*
- ▲ A list of the references cited in the book.
- ▲ A glossary of terms used in this book.

Part **1**

The Traditional Testing Process



An Overview of Testing

As we strive to implement the new features of our applications, there is one thing we can say with absolute certainty – that at the same time, we also introduce new defects.

2.1 Introduction

This chapter gives an overview of testing in order to provide an understanding of what testing is and why it is such a challenge, and to emphasize that whenever we test software, the process must be made to be as efficient and effective as possible.

Readers familiar with the need for efficient and effective testing may not find it necessary to read this chapter.

2.2 The Challenge of Testing

So, just how difficult is testing? To help answer this question, consider the following example:

Imagine we have a requirement to test a simple function, which adds two thirty-two-bit numbers and returns the result. If we assume we can execute 1000 test cases per second, just how long will it take to thoroughly test this function?

If you guessed seconds, you are way out. If you guessed minutes, you are still cold. If you guessed hours or days or even weeks, you are not even slightly warm. The actual figure is 585 million years.¹

But surely, this is a daft example? Nobody in his or her right mind would test such a function by trying out every single possible value! In practice, we would use some formal test design techniques such as *boundary analysis* and *equivalence partitioning* to help us select specimen data to use in our test cases (see Chapter 3 for details

¹The calculation is quite straightforward (with a calculator): $2^{(32 \times 32)} / 1000 / 60 / 60 / 24 / 365.25 = 584542046$ years.

of test design techniques). Using this test data, we would make the assumption that if the function performed satisfactorily for these specimen values, it will perform satisfactorily for all similar values, reducing the time needed to test the function to an acceptable timescale.

However, as testers, we should not start feeling too confident too soon – there are many other issues that can complicate the testing of our “simple” function. For example:

- ▲ What if the function needs to interoperate with other functions within the same application?
- ▲ What if the data for the calculation are obtained across a complex Client/Server system and/or the result is returned across the Client/Server system?
- ▲ What if the calculation is driven via a complex Graphical User Interface with the user being able to type the addition values into fields and push the buttons to perform the calculation in any arbitrary order?
- ▲ What if this function has to be delivered on a number of different operating systems, each with slightly different features, and what if individual users are able to customize important operating system features?
- ▲ What if this function has to be delivered on a number of different hardware platforms, each of which could have different configurations?
- ▲ What if the application this function belongs in has to interoperate with other applications, and what if the user could be running an arbitrary number of other applications simultaneously (such as e-mail or diary software)?

These are all typical requirements for software systems that a great many testers face every day during their testing careers, and which act to make software systems highly complex and make testing an immense challenge!

2.3 What Is Testing?

The process of testing is by no means new. The Oxford English Dictionary tells us that the term “test” is derived from the Latin expression *testum*, an earthenware pot used by the Romans and their contemporaries in the process of evaluating the quality of materials such as precious metal ores.

Computer programs have undergone testing for almost as long as software has been developed. In the early days of software development there was little formal testing, and debugging was seen as an essential step in the process of developing software.

As the software development process has matured, with the inception and use of formal methods (such as Reference 6), the approach to testing has also matured, with formal testing methods and techniques (such as Reference 8) being adopted by testing professionals.

Most workers in the field of modern software development have an intuitive view of testing and its purpose. The most common suggestions include:

- ▲ To ensure a program corresponds to its specification
- ▲ To uncover defects in the software
- ▲ To make sure the software doesn't do what it is not supposed to do
- ▲ To have confidence that the system performs adequately
- ▲ To understand just how far we can push the system before it fails
- ▲ To understand the risk involved in releasing a system to its users.

Here are some more formal definitions of testing:

Testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. (Reference 1)

This definition addresses the traditional testing approach – that is, does the system conform to its stated requirements? This appears to be an intuitive view of testing: we have some statements about how the system should behave, and we confirm that these requirements are met. This approach is also known as *Positive Testing*.

Here is another view of testing:

Testing is the process of executing a program or system with the intent of finding defects. (Reference 2)

This definition is less intuitive and does not, strictly speaking, consider the requirements of the system.² Instead, it introduces the notion of actively looking for defects outside the scope of the software requirements, which in practice could be any problem or defect in the system. This approach is also known as *Negative Testing*.

In practice, testing will combine elements of both Positive and Negative testing – checking that a system meets its requirements, but also trying to find errors that may compromise the successful operation or usefulness of the system.³

Most recently, the notion of defining testing in terms of risk has become increasingly popular. In this use, the term “risk” relates to the possibility that the *Application Under Test (AUT)* will fail to be reliable or robust and may cause commercially damaging problems for the users. Here is a definition of testing in terms of risk:

Testing is the process by which we explore and understand the status of the benefits and the risk associated with release of a software system. (Reference 28)

Within this definition of testing, the role of the tester is to manage or mitigate the risk of failure of the system and the undesirable effects this may have on the user.

Defining testing in terms of risk provides the tester with an additional strategy for approaching the testing of the system. Using a risk-based approach, the tester is involved in the analysis of the software to identify areas of high risk that need to be tested thoroughly to ensure the threat is not realized during operation

²Although, a “defect” could be considered to be a failure of the system to support a particular requirement.

³It is possible to argue that in a perfect world of complete requirements and accurate specifications, there would be no need for negative testing, since every aspect of the Application Under Test (AUT) would be specified. Unfortunately, the reality is somewhat short of perfection, and so testing is always likely to include a degree of Negative Testing.

of the system by the user. Furthermore, the notion of risk in a project management context is well known and understood, and a great many tools and techniques exist that can be applied to the testing process (such as References 28, 29, and 30).

It may be difficult for the staff involved in the planning and design of tests to identify specific risks for a particular AUT (especially when they may not be familiar with the domain of operation of the software). In assessing risk, it is essential that the following issues be considered:

- ▲ The business, safety, or security criticality of the AUT
- ▲ The commercial/public visibility of the AUT
- ▲ Experience of testing similar or related systems
- ▲ Experience of testing earlier versions of the same AUT
- ▲ The views of the users of the AUT
- ▲ The views of the analysts, designers, and implementers of the AUT.

The need to analyze risk in the testing process is addressed in Chapter 4 – The Management and Planning of Testing.

2.4 Verification and Validation

Another two testing terms, which are frequently used but often confused, are *verification* and *validation*. Reference 40 provides a formal definition of these terms:

Verification is the process by which it is confirmed by means of examination and provision of objective evidence that specific requirements have been fulfilled (during the development of the AUT).

Validation is the process by which it is confirmed that the particular requirements for a specific intended use (of the AUT) are fulfilled.

Reference 26 provides a succinct and more easily remembered definition of these terms:

Verification: Are we building the product right?

Validation: Are we building the right product?

In essence, verification deals with demonstrating that good practice has been employed in the development of the AUT by, for example, following a formal development process (such as Reference 8).

Validation deals with demonstrating that the AUT meets its formal requirements, and in that respect conforms closely to the Hetzel definition of testing discussed earlier in this chapter (Reference 1).

Both verification and validation (also termed *V&V*) are key to ensuring the quality of the AUT and must be practiced in conjunction with a rigorous approach to requirements management. Chapter 4 provides guidance on the role of requirements management and its role within *V&V*.

2.5 What Is the Cost of Not Testing?

Many examples exist, particularly where systems have had safety critical, business critical, or security critical applications, where the failure of the system has, either through litigation or loss of public confidence, resulted in the provider of the software going out of business.

Even where a system does not deal with a critical application, failure of high-profile systems, such as an organization's Web Site, free shareware, or demonstration software, can still have serious commercial implications for the organization in terms of loss of public confidence and prestige.

Some defects are very subtle and can be difficult to detect, but they may still have a significant effect on an organization's business. For example, if a system fails and is unavailable for a day before it can be recovered, then the organization may lose a day's effort per person affected. If an undetected defect simply causes the performance of a system to degrade, then users may not even notice that a problem exists. If, however, the defect causes a loss of productivity of just 30 minutes per day, then the organization could lose in the order of 20 days effort per person per year!

2.6 Testing – The Bottom Line

Phrases like “Zero Defect Software” or “Defect Free Systems” are hyperbole, and at best can be viewed only as desirable but unattainable goals.⁴

In practice, it is impossible to ensure that even relatively simple programs are free of defects because of the complexity of computer systems and the fallibility of the development process and of the humans involved in this process.

In simple terms, it is impossible to perform sufficient testing to be completely certain a given system is defect free. When this problem is combined with the fact that testing resources are finite and (more typically) in short supply, then adequate testing becomes problematical. Testers must focus on making the testing process as efficient and as effective as possible in order to find and correct as many defects as possible.

Ultimately, testing can only give a measure of confidence that a given software system is acceptable for its intended purpose. This level of confidence must be balanced against the role the system is intended for (such as safety critical, business critical, secure, confidential, or high-profile applications) and against the risk of the system failing in operation, before the decision to release or to accept software can be made.

⁴Even with mathematically rigorous methods (such as Z and VDM), it is still impossible to say that any but the simplest pieces of software will be defect free.

The key to effective testing is making the process as efficient as possible: rigorous planning and project management must be employed; testers must make use of their knowledge of testing techniques and experience to guide them in devising effective tests; re-use must be introduced and managed at all stages of the testing process; and organizations need to consider carefully the benefits and role of automated software testing tools.

Each of these issues is considered in detail in the chapters that follow.

2.7 Additional Information

Both the Hetzel and Myers books (References 1 and 2), are excellent general introductions to testing, as well as useful reference documents for more experienced testing practitioners. Bach's paper (Reference 28) provides a risk-based view of testing and offers guidance on performing testing from a risk perspective.

Also recommended is the Fewster and Graham book (Reference 17), which although specifically dealing with software test automation, also provides a good general treatment of the subject.

Other testing books that are well worth reading include the Gilb and Graham book on Software Inspection (Reference 41), the Kit book (Reference 42), and the Beizer text on software testing techniques (Reference 43).

The CCTA IT Infrastructure Library series (Reference 5), published by the government Centre for Information Systems, provides thorough guidelines on setting up and administering a testing program within an organization.

The British Computer Society Special Interest Group in Software Testing (BCS SIGiST) is very active in promoting the role of software testing and the use of best practice and process. A visit to its Web site is worthwhile: www.bcs.org.uk/sigist.

Other testing Web sites well worth a visit include:

- ▲ The Professional Tester: www.professional-tester.com
- ▲ Software Quality Engineering: www.sqe.com
- ▲ QBIT: www.qbit-testing.com
- ▲ The Open University Faculty of Maths and Computing:⁵
www3.open.ac.uk/courses/subjects/computing.htm
- ▲ The Software Testing Institute: www.ondaweb.com/sti
- ▲ The Software Testing Laboratories: www.stlabs.com.

Also visit the Cambridge University Press Web site us.cambridge.org/titles/052179546X, which lists further references to testing books as well as links to other Web sites specializing in testing.

⁵M301 "Software Systems and Their Development" in particular contains useful reference material on software development, testing, and the role and use of automated testing tools.

The following standards are worth obtaining for reference purposes:

- ▲ “IEEE Standard for Software Test Documentation,” IEEE Std 829–1998, IEEE, December 1998.
- ▲ “Software Testing – Part 1: Vocabulary,” BS 7925–1:1998, BSI, August 1998.
- ▲ “ISO 9001” & “ISO 9003,” International Standards Organization.

Testing Techniques

I am for those means which will give the greatest good to the greatest number.

Abraham Lincoln

3.1 Introduction

This chapter describes a number of testing techniques that the *Test Analyst* can employ in designing and developing effective *Test Cases*.

This chapter focuses on practical, pragmatic design techniques that test designers employ in their everyday testing tasks and does not try to provide an exhaustive list of every possible testing technique available (such as Symbolic Testing or other research-oriented testing methods).

The testing techniques are organized into three main groups:

- ▲ **General Testing Techniques** (that is, high-level approaches to testing that can be applied to the specific testing techniques described in the second and third groups), which include:
 - *Positive and Negative testing*
 - *White Box and Black Box testing*
 - *error guessing*
 - *automated software testing.*
- ▲ **Functional Testing Techniques** (that is, testing techniques used to confirm that the *Application Under Test* [AUT] meets its *functional requirements*), which include:
 - *Equivalence Partitioning*
 - *Boundary Analysis*
 - *Intrusive Testing*
 - *Random Testing*
 - *State Transition Testing*
 - *Static Testing*
 - *Thread Testing.*
- ▲ **Nonfunctional Testing Techniques** (that is, testing techniques used to verify that the AUT meets its *nonfunctional requirements*), which include:
 - *Configuration/Installation Testing*
 - *Compatibility and Interoperability Testing*

- *Documentation and Help Testing*
- *Fault Recovery Testing*
- *Performance Testing*
- *Reliability Testing*
- *Security Testing*
- *Load Testing*
- *Usability Testing*
- *Volume Testing.*

Where appropriate, examples illustrating the use of a particular test design technique are provided in the appendices.

3.2 General Testing Techniques

3.2.1 Positive and Negative Testing

Positive and Negative testing are complementary testing techniques, and in practice both processes are likely to be used in the testing of a particular AUT (Figure 3.1).

The Video Cassette Recorder (VCR) can be in one of three states: Standby, On, or Playing.

When in Standby mode, the VCR can be turned on by pressing the Standby button once (an indicator light turns from red to green to show the VCR is On).

When the VCR is On, it can return to Standby mode by pressing the Standby button once (an indicator light turns from green to red to show the VCR is in Standby mode).

When the VCR is On, pressing the Play button causes the currently loaded video cassette tape to play. Pressing the Stop button when the VCR is playing a video cassette tape causes the VCR to stop playing the video tape.

Examples of positive tests could include:

- Verifying that with the VCR in Standby mode, pressing the Standby button causes the VCR to turn on and the indicator light changes from red to green
- Verifying that with the VCR in the On state, pressing the Standby button causes the state of the VCR to change to Standby and the indicator light changes from green to red.

Examples of negative tests could include:

- Investigating what happens if the VCR is playing a video tape and the Standby button is pressed
- Investigating what happens if the VCR is On and the Play button is pressed without a video cassette tape in the VCR.

The process of Positive Testing is intended to verify that a system conforms to its stated requirements. Typically, Test Cases will be designed by analysis of the *Requirements Specification document* for the AUT. The process of Positive Testing must be performed in order to determine if the AUT is “fit for purpose,” and where the application will be delivered to a customer, the process is likely to be a compulsory aspect of contractual acceptance.

The process of Negative Testing is intended to demonstrate “that a system does not do what it is not supposed to do.” That is, Test Cases are designed to investigate the behavior of the AUT outside the strict scope of the Requirements Specification. Negative Testing is often used to tests aspects of the AUT that have not been documented or have been poorly or incompletely documented in the specification.

Since the requirements for software systems are frequently poorly specified (or possibly missing, such as is the case with some legacy systems), it is likely that a certain amount of Negative Testing of the AUT will be required to provide confidence in the operation of the application. This must be balanced against the need to be as effective and as efficient as possible in testing, since Negative Testing is an open-ended technique and is not bounded in terms of possible expended effort in the same manner as Positive Testing.

Figure 3.1. Examples of Positive and Negative Testing.

To illustrate the techniques of Positive and Negative Testing consider the extract in Figure 3.1 from the specification for the operation of a video cassette recorder (VCR).

3.2.2 White Box and Black Box Testing

White Box and Black Box testing are complementary approaches to the testing of the AUT that rely on the level of knowledge the Test Analyst has of the internal structure of the software.

White Box tests are designed with the knowledge of how the system under test is constructed. For example, a Test Analyst designing a Test Case for a particular function that validates the input of an integer within a specified range of values may be aware that the validation is performed using an “IF-THEN-ELSE” instruction, and so will structure the test to check that the appropriate logic has been implemented. It is anticipated that the Test Analyst will have access to the design documents for the system or other implementation documentation, or will be familiar with the internal aspects of the system. Other sources of knowledge may include familiarity with systems similar or related to the AUT.

Black Box tests are designed without the knowledge of how the system under test is constructed. This is not as unlikely an event as it may at first seem. For example, the Test Analyst may be called upon to design *Regression Tests* for extensions to legacy applications where the system documentation has been lost, or to design acceptance tests for *commercial off-the-shelf* (COTS) products or systems implemented by third-party developers. In Black Box Testing, the design of Test Cases must be based on the external behavior of the system. If requirements exist for the system, these must be tested against. Where there are no requirements, then user guides or documentation describing how the system should behave can be used as the basis of test design.

The terms *behavioral* and *structural testing* (Reference 45) are often used synonymously with the terms “Black Box” and “White Box” testing. In fact, behavioral test design is slightly different from Black Box Testing because the knowledge of how the system under test is constructed is not strictly prohibited during test design, but is discouraged.

Because knowledge of how the software was constructed is necessary for White Box Testing, this technique is typically used during the early phases of the testing process where the programmer is responsible for designing and executing the tests (such as *Unit Testing* – see Chapter 5). Black Box techniques are typically used during the later phases of the testing process (such as *System Testing* – see Chapter 7), where the Test Analyst may not have access to information about how the software was constructed or whether the software has been developed by a third party or bought in as COTS software. In practice, Test Analysts often use a mixture of Black and White Box techniques in the design of Test Cases, in effect following a “gray box” or “translucent box” approach.

3.2.3 Error Guessing

Error guessing is not in itself a testing technique but rather a skill that can be applied to all of the other testing techniques to produce more effective tests (i.e., tests that find defects).

Error guessing is the ability to find errors or defects in the AUT by what appears to be intuition. In fact, testers who are effective at error guessing actually use a range of techniques, including:

- ▲ Knowledge about the AUT, such as the design method or implementation technology
- ▲ Knowledge of the results of any earlier testing phases (particularly important in Regression Testing)
- ▲ Experience of testing similar or related systems (and knowing where defects have arisen previously in those systems)
- ▲ Knowledge of typical implementation errors (such as division by zero errors)
- ▲ General testing rules of thumb or heuristics.

Error guessing is a skill that is well worth cultivating since it can make testing much more effective and efficient – two extremely important goals in the testing process. Typically, the skill of Error guessing is obtained through experience, although some people have a natural flair for finding defects in software systems.

3.2.4 Automated Software Testing

Although strictly speaking not a testing technique, automated testing is included in this section because it can provide an effective testing solution and can be used in conjunction with the later techniques.

The majority of test automation tools¹ allow re-usable tests to be created by recording the behavior of the AUT under its typical operation, then subsequently replaying those tests (against a later build or release of the software) to compare the observed behavior against the expected behavior (i.e., the originally recorded behavior). Where the observed behavior differs from the expected behavior, the tool logs this event and allows the tester to analyze how and why the differences have occurred.

Automated tools exist that support both the testing of functional and nonfunctional aspects of the AUT, such as testing against the functional requirements for the AUT and *Performance, Load, Volume, and Stress* testing.

A significant benefit of such tools is their ability to operate unattended (e.g., to run overnight or over the weekend), providing significant gains in productivity for the testing task. Automated tools can also provide additional confidence in the AUT by allowing more tests to be executed in the same time as that taken for an equivalent manual test.

The principal use of such tools is in *Regression Testing* (see Chapter 11) or retesting of the AUT where it becomes possible to rerun all of the previously generated tests against the application following modification or extension of the software,

¹Other examples of test automation tools include those where the test is created using a simple scripting language, such as Reference 47.

rather than the typical manual approach of attempting to assess the impact of the modifications on the AUT and selecting a “representative” subset of the tests to run. This is of particular benefit where an iterative approach to software development (Reference 8) has been adopted, since there is typically a demanding requirement to regression test the deliverables from earlier iterations.

While these tools have many potential benefits, they will not be appropriate for all testing tasks, and a critical review and evaluation of the advantages and disadvantages must be undertaken before their introduction. Appendix N provides both a scheme and a set of criteria to support the process of evaluating automated software testing tools.

Examples of where automated testing tools may be particularly appropriate include:

- ▲ Testing of applications that have a rapid build and release cycle
- ▲ Testing of applications that have to be delivered across diverse platforms
- ▲ Testing of applications with complex GUI and/or Client/Server architectures
- ▲ Testing of applications that require thorough, rigorous, repeatable testing (e.g., safety critical, business critical, or security critical systems)
- ▲ Where there is a need to reduce testing timescales and effort
- ▲ Where there is a need to do more thorough testing in the same timescales.

Where automated testing tools are adopted, they must be integrated with the existing testing management process and not just “tacked on” to the current process with the expectation that they will provide instant benefits.

Finally, although it is possible to gain rapid benefits from the introduction of automated tools (see Reference 17, for example), typically the benefits that such tools bring will not be realized during their initial use, since the tools will require integration into the existing testing process and there will be learning curve issues to be overcome. However, it is important that organizations persevere with their use; otherwise such tools are likely to quickly become “Shelfware,” and the initial investment in the tool and its use will be lost.

References 22 and 23 provide typical examples of such automated testing tools, while the case studies presented in Chapters 16 and 18 provide examples of the successful use of such tools.

3.3 Functional Testing Techniques

3.3.1 Equivalence Partitioning

Equivalence Partitioning relies on the fact that inputs and outputs from the AUT (which can include environment variables and logical operations) can be grouped or partitioned into coherent groups or classes, and that all instances of these classes will be treated in the same manner by the system. The principal assumption of this technique is that testing one instance of the class is equivalent to testing all of them.

The skill in using this technique lies in identifying the partition of values and selecting representative values from within this partition, and in deciding how many values to test.

A number of graphical methods can be used to support this technique. They involve visualizing the partition and tabulating the Test Cases and their values (Appendix K provides a worked example that demonstrates the technique of Equivalence Partitioning).

3.3.2 Boundary Analysis

Boundary Analysis is a technique related to Equivalence Partitioning and relies on the same underlying principal: that inputs and outputs can be grouped into classes, and all instances of these classes will be treated similarly by the system.

Whereas Equivalence Partitioning deals with selecting representative values from within the class, Boundary Analysis focuses on the testing of values from the boundary of the class. Specifically, the technique will design a Test Case for the boundary value itself, plus a Test Case for one significant value on either side of the boundary.

In practice, Boundary Analysis and Equivalence Partitioning are used together to generate a complete set of Test Cases providing tests of the “edges” of the ranges of values as well as values from the “body” of the range of values.

As with Equivalence Partitioning, a number of graphical methods can be used to support Boundary Analysis that involve visualizing the boundary (or boundaries) and tabulating the Test Cases and their values (Appendix L contains a worked example that demonstrates the technique of Boundary Analysis).

In practice, when using Boundary and/or Equivalence techniques, the Test Analysts may face a combinatorial explosion in terms of the seemingly infinite number of possible combinations of values when considering the interaction between the difference variables within the AUT. One possible solution the Test Analyst can use to control this complexity is the use of *pair-wise combinations*.

3.3.3 Intrusive Testing

As the name suggests, Intrusive Testing involves the deliberate modification of the AUT or its behavior for the purposes of testing.

For example, where the result of a test would not be visible to the tester (such as the modification of the value of a variable that would not be displayed to the user during the execution of the AUT), the Test Analyst may introduce additional statements into the application to display the value of the variable. Other examples include directly setting the values of variables using symbolic debuggers or deliberately triggering error conditions within the AUT.

Clearly, any modifications made to the system for the purpose of testing must not be delivered to the customer/user of the system. Therefore, rigorous change control and configuration management is essential.

While this technique can be useful during testing, there is a fundamental objection to this form of testing: what you deliver after testing is not what you actually tested.

This technique should be used with extreme caution because there are many exam-

ples of systems that have been released containing changes introduced during Intrusive Testing that have manifested themselves during normal operation of the system.

3.3.4 Random Testing

Random testing is one of the very few techniques for automatically generating Test Cases. Test tools provide a harness for the AUT, and a generator produces random combinations of valid input values, which are input to the AUT. The results generated by each input are logged, and the inputs and their respective results can be inspected following testing for defects.

Although automating the process of providing inputs and stimulating the AUT is relatively straightforward, automating the process of checking the outputs is difficult, which can make this a particularly labor-intensive process (as well as introducing the likelihood of human error). Depending on the approach employed, it may also be difficult to reproduce the results of a particular series of random tests. As a result, Random Testing is not a commonly used technique.

Where it is used, this technique can be highly effective in identifying obscure defects; however, Random Testing should not be used as a substitute for other systematic testing techniques but rather should be considered as an adjunct to the more rigorous techniques.

Random Testing may also be used successfully with automated testing tools as a source of test data.

3.3.5 State Transition Analysis

State Transition Analysis models the AUT in terms of the states the system can be in, the transitions between those states, the actions that cause the transitions, and the actions that may result from the transitions.

In designing Test Cases, State Transition information can be derived from the requirements document or may be documented in the design of development methods that support State Transition notations (such as the Unified Modelling Language, or UML – Reference 6).

Test Cases are designed to exercise transitions between states and specify:

- ▲ The initial state
- ▲ The inputs to the system
- ▲ The expected outputs
- ▲ The expected final state.

As well as being useful in Positive Testing, this technique is particularly useful in Negative Testing because the process of considering the system states and transitions will often suggest tests covering issues either overlooked or poorly specified in the specification document.

A number of graphical methods can be used to support this technique that involve visualizing the states and their transition (Appendix M contains a worked example that demonstrates the technique of State Transition Analysis).

3.3.6 Static Testing

As the name implies, Static Testing does not involve executing or running the AUT but deals with inspecting the system in isolation.

A number of methods and tools are typically used in Static Testing:

- ▲ Code review and inspection (by members of the development team)
- ▲ Code walk-through (using paper and pen evaluation)
- ▲ Static analysis tools (such as syntax and style checkers)
- ▲ Complexity estimating tools (such as cyclic complexity analysis).

Static Testing usually takes place early in the testing process and is particularly appropriate as an adjunct to Unit Testing. See Reference 24 for further information on Static Testing.

3.3.7 Thread Testing

Thread Testing is used to test the business functionality or business logic of the AUT in an end-to-end manner, in much the same way a user or an operator might interact with the system during its normal use.

For example, if the AUT is a library system for managing the book-lending process, one particular thread to test might be a request for a particular book. The process might involve the following steps:

- ▲ The operator logging onto the system
- ▲ Validation of the operator privileges
- ▲ Entry of the details of the book being requested
- ▲ Validation of the book details
- ▲ Access to the database to verify the availability of the book
- ▲ Modification of the database to show that the book has now been borrowed
- ▲ Printing of the appropriate paperwork to conclude the transaction.

This technique contrasts with earlier testing phases (such as Unit and Integration testing) where the tester is likely to have been more interested in the correct functioning of the object under test rather than on its business logic.

3.4 Nonfunctional Testing Techniques

3.4.1 Configuration/Installation Testing

Configuration/Installation Testing is used to ensure that hardware and software have been correctly installed, all necessary files and connections have been created, all appropriate data files have been loaded (such as database and/or historic information), system defaults have been correctly set, and interfaces to other systems or peripheral devices are all working.

3.4.2 Compatibility and Interoperability Testing

Compatibility Testing verifies that when the AUT runs in the live environment its operation does not impact adversely on other systems, and vice versa. It is often required when a new system replaces an existing system that had previously interacted with one or more other systems.

Interoperability Testing verifies that when the AUT runs in the live environment it is able to communicate successfully with other specified systems (such as invoking, passing data to, or receiving data from another system).

3.4.3 Documentation and Help Testing

The testing of the documentation and help facilities of the AUT is a frequently overlooked aspect of testing the complete system. The testing of these items is often omitted from the testing process (because of lack of time or resources), since it is thought to be outside the scope of the testing process, or through simple carelessness.

In fact, these aspects of the AUT may be vital to its successful operation and use of the system, particularly for new or naive users.

User documentation and help system information should be checked for conformance to the requirements specification document. Specific testing techniques involve review of the documentation, cross-referencing checks (e.g., against the document Contents and Index sections), and Thread Testing of typical user help scenarios.

3.4.4 Fault Recovery Testing

Fault Recovery Testing verifies that following an error or exception (such as a system crash caused by loss of power) the AUT can be restored to a “normal” state, such as the initial state of the application when it is first executed, and that the AUT can continue to perform successfully. Fault Recovery Testing may also be used to verify the successful roll-back and recovery of the data used or manipulated by the AUT following an error or exception.

3.4.5 Performance Testing

Performance Testing is traditionally a problem area because system performance is frequently poorly specified and can be measured using a variety of different criteria, including:

- ▲ User response times
- ▲ System response times
- ▲ External interface response times
- ▲ Central processor unit (CPU) utilization
- ▲ Memory utilization.

In conducting Performance Testing it is essential to have a “performance model” that specifies what aspect of system performance is being tested, what the performance requirements are, how they can vary under typical system usage, and how they will be tested. Once the performance model is established, the Test Analyst can design clearly defined Test Cases based on this information.

Another challenge in Performance Testing is the difficulty in accurately representing the test environment. For example, where there are likely to be large numbers of concurrent users and/or large amounts of data, it may be impractical or infeasible to set up and conduct realistic Performance Testing. Furthermore, recreating such a complex test environment to reproduce the tests results or to rerun the tests against a new build or release of the AUT will be extremely difficult.

Because of these difficulties, Performance (and Volume and Stress) Testing is often conducted using automated testing tool support. Examples of such tools include “low-level” performance testing tools (such as Reference 44) that are able to identify sections of the AUT which are executed frequently and which may need to be optimized to improve application performance, or “high-level” tools that can simulate very large numbers of users and their typical tasks in order to realistically exercise the AUT (see Reference 23).

3.4.6 Reliability Testing

Reliability Testing involves tests to ensure the robustness and reliability of the AUT under typical usage. It typically includes integrity tests and structural tests.

Integrity tests focus on verifying the AUT’s robustness (resistance to failure) and compliance to language, syntax, and resource usage. For example, a particular unit could be executed repeatedly using a test harness to verify that there are no memory leak problems. Integrity tests can be designed, implemented, and executed during any testing phase.

Structure tests focus on verifying the AUT’s adherence to its design and formation. For example, a structure test could be designed for Web-enabled applications to ensure that all links are connected, appropriate content is displayed, and there is no orphaned content.²

3.4.7 Security Testing

Depending on the intended role of the AUT, requirements may exist that specify the need to ensure the confidentiality, availability, and integrity of the data and software. Security Testing is intended to test whether the features implemented within the AUT provide this required level of protection.

Security Testing is mainly concerned with establishing the degree of traceability from the requirements through to implementation, and in the validation of those

²Orphaned content describes files for which there is no “inbound” link in the current Web site, that is, there is no way to access or present the content.

requirements. Where particularly rigorous Security Testing is necessary, it is typically conducted by a dedicated team, whose role is to evaluate the conformance of the AUT to the particular security requirements. Such teams may also verify that a standard approach or formal process has been followed in the development of the AUT.

Security tests can be designed, implemented, and executed within any testing phase but are typically conducted at System Test and rerun during User Acceptance Testing.

3.4.8 Stress Testing

Stress Testing (also termed Load Testing – Reference 45) examines the ability of the system to perform correctly under instantaneous peak loads with the aim of identifying defects that appear only under such adverse conditions. For example, if the specification for the AUT states that the system should accept 30 concurrent users, what happens if a 31st user attempts to log on? Similarly, what happens if all 30 users attempt to log on simultaneously?

Simulation is a frequently used method in Stress Testing. In the above example, it would be difficult from a planning and resource perspective to have 30 users simultaneously logging on at 30 terminals; however, it may be possible to simulate 30 virtual users performing this activity.

It is also worth considering commercial tool support for Stress Testing. Several software testing tools automate the process, some of which also incorporate simulation techniques for representing large numbers of users (see Reference 23, for example).

3.4.9 Usability Testing

The topic of software usability is increasingly important, particularly with the increase in popularity of visual programming environments and the use of development techniques such as user-centered design (Reference 9) and rapid prototyping (Reference 12).

Users themselves are also becoming increasingly sophisticated in their expectations of what a user interface should do and how it should support their business activities in a consistent and intuitive manner. Conversely, this must always be balanced by the inevitable users who are unfamiliar with computer systems but who are still expected to be able to use a particular application with little or no guidance or training.

Specific techniques employed in Usability Testing include:

- ▲ Conformance checks – testing the application against agreed user interface (UI) standards (such as the Microsoft Windows Standards – Reference 10)
- ▲ User-based surveys – psychometric testing techniques to analyze user perceptions of the system (such as that documented in Reference 11)
- ▲ Usability Testing – where users are asked to perform a series of specified business tasks with the AUT under controlled conditions in order to test the usability goals or requirements of the AUT.

Appendix O describes Usability Testing in greater detail and provides an overview of one particular approach to Usability Testing. Also see Reference 9 for a definitive treatment of usability.

3.4.10 Volume Testing

Volume Testing (sometimes also termed “flood testing” – Reference 45) examines the system’s ability to perform correctly using large volumes of data with the aim of identifying defects that appear only under such conditions.

Typically, at the lower levels of testing (such as Unit and Integration testing) a representative sample or copy of the data the system will process is used. Only at the higher levels of testing (such as System and Acceptance testing) is live data likely to be used. This may be an important issue where a system is expected to process particularly large quantities of data, since the ability of the system to function correctly under these circumstances will need to be tested.

Volume Testing may involve rerunning appropriate tests from earlier testing phases (from the Unit or Integration Reuse Packs, for example – see Appendix I) in addition to designing new Test Cases. One particular Volume Testing technique involves repeatedly running the same test while increasing the volume of data involved at each iteration in order to identify the point at which the system fails to cope with the volume of data.

The terms “Volume Testing” and “Stress Testing” are frequently confused. Myers in Reference 2 provides a useful typing analogy to help distinguish between these techniques: Volume Testing will confirm that a typist can cope with a very large document; Stress Testing will confirm that the typist can manage to enter 40 words per minute.

3.5 Further Reading on Testing Techniques

The following references provide additional information on testing techniques and their role within the testing process.

The Beizer book (Reference 43) provides extensive information on testing techniques, as does the Kit text (Reference 42), from a process-improvement context.

The testing unit of M301 (the third-level Open University Software Engineering course – Reference 24) provides a thorough treatment of testing techniques (in particular static analysis and review techniques) within the context of a formal engineering approach to software development, testing, and deployment.

Reference 46 provides descriptions of the common testing techniques within the context of an end-to-end software development process.

Reference 45 offers an extensive and thorough coverage of software testing techniques, with particular emphasis on a quality-assurance approach to Validation and Verification (V&V) of the AUT.

The Management and Planning of Testing

Plans are worthless, the process of planning is invaluable

Eisenhower

4.1 Introduction

This chapter deals with the issues associated with the organization, management, and high-level planning of the testing process.

A generic approach to organizing the testing process is described, which can be applied to businesses involved in developing and/or testing software. The chapter discusses the implications for the testing process of the size of the business, its prevailing quality culture, and (where appropriate) the maturity of its software development process.

The case studies in Part 2 provide specific examples of how a number of different businesses have customized this generic approach in response to their own particular organizational, management, and testing requirements.

This chapter also considers a number of key requirements that must be considered in implementing a rigorous, effective, and efficient testing process: the need for configuration management and defect tracking and the need to ensure that each of the requirements of the *Application Under Test (AUT)* has been verified. The issues associated with each of these requirements are discussed and recommendations for supporting them are provided.

The chapter concludes with a discussion of the role of risk-based testing and its use in the management and planning of testing projects.

The chapter is structured as follows:

- ▲ Section 4.2 discusses the high-level organization of the testing process and the roles that are required and their interrelationships. It also provides a diagram summarizing this information.

- ▲ Section 4.3 provides specific details of the roles and responsibilities of the staff involved in a testing program.
- ▲ Section 4.4 reviews the individual testing phases that comprise the complete testing cycle from *Unit Testing* through *Acceptance Testing* and includes *Regression Testing*.
- ▲ Section 4.5 describes the *V Model*, its role in the management of testing, and its use in the process of planning the testing activities and in controlling risk
- ▲ Sections 4.6, 4.7, and 4.8 consider the need for requirements management, configuration management, and defect tracking in the testing process and makes recommendations for supporting these requirements
- ▲ Section 4.9 briefly reviews the role of risk-based testing in the management and planning of testing and provides a number of references that give further information on this subject.

4.2 The Organization of Testing

One of the first steps in implementing a testing process is to consider the high-level organization of the staff involved in testing, their relationship to each other, and how the testing process will integrate with the existing management structure of the business.

The guiding principle in establishing the high-level organization is that there are a series of activities common to all testing tasks which must be managed effectively and efficiently to ensure they are completed on time, on budget, and to an acceptable level of quality.

The challenge facing the majority of businesses involved in implementing a testing process is that testing cannot take place in isolation; there will be existing departments and groups within the company with which the testing process must integrate and existing staff who will need to participate in the testing process. For example, many businesses incorporate the management of the testing process into the existing *information technology (IT)* group. Alternatively, for businesses with a well-established quality culture, the testing process is often managed from within the *quality assurance (QA)* group.

Further issues that will need to be considered include the size and complexity of the business, the geographic distribution of offices, and the balance of software developed/extended in-house, developed under contract by a third party, and bought-in as *commercial off-the-shelf (COTS)* products. All of these factors will affect the size and role of the testing process within a particular business.

Figure 4.1 gives a generic view of the organization of the testing process. Chapters 14–18 provide specific examples of how a number of different businesses have implemented the organization of their testing process based on their particular organizational, management, and testing requirements. The testing roles shown in Figure 4.1 and their specific responsibilities are documented in the following section.

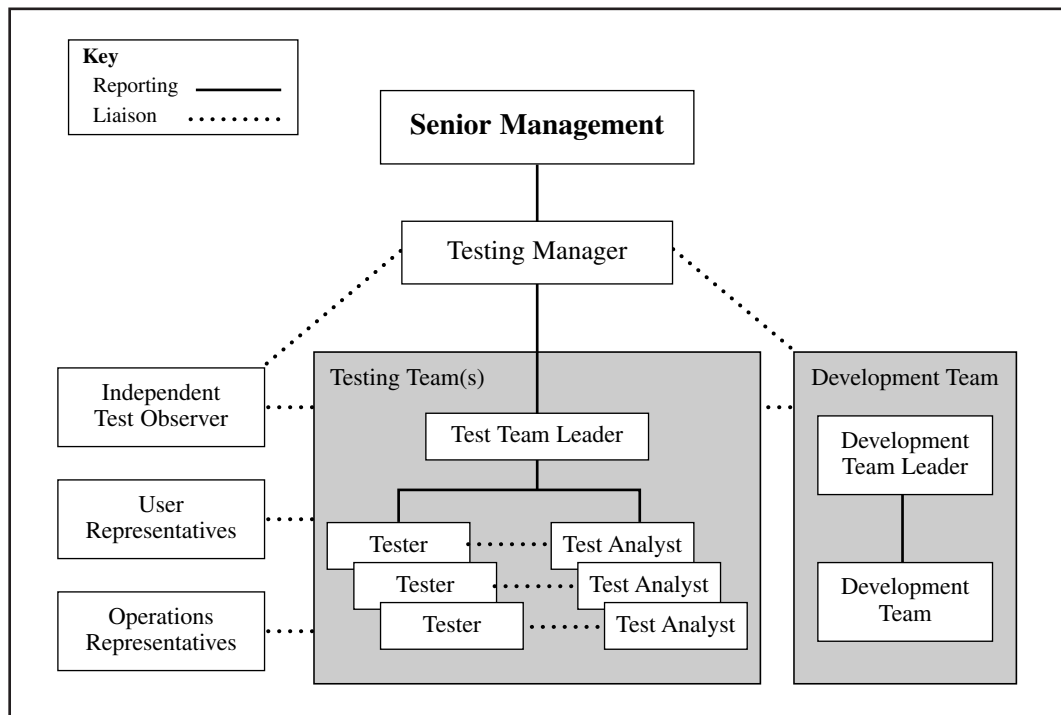


Figure 4.1. Organization of the Testing Program.

4.3 Roles and Responsibilities

4.3.1 Overview

This section describes the roles of the staff involved in the testing process, their individual responsibilities, and their requirements for reporting and liaison.

It is important to note when reading the following paragraphs that it is possible for a single person to assume more than one role within any testing process. For example, in a small organization in which there are few testing staff, one individual may assume the roles of both *Test Analyst* and *Tester* on a testing project. Similarly, the responsibilities of the *Testing Manager* and *Test Team Leader* may be assumed by one person if there is only one testing task or project to manage. The notable exception to this is the *Independent Test Observer*, who must maintain complete independence from the testing project and its progress.

A single person may also assume different roles on different projects. For example, a *Test Analyst* on one project may perform the role of *Test Team Leader* on another. Again, this will depend on the size of the business and the number of staff involved in testing.

In addition to the traditional roles described in the following paragraphs, Section 4.3.7 describes a number of additional roles that may be beneficial to include in testing projects with particular characteristics or requirements – such as a project in which automated testing tools are employed (see Reference 23, for example).

Concise terms of reference for the traditional testing roles described in this chapter are given in Appendix A.

4.3.2 Testing Manager

The Testing Manager has the authority to administer the organizational aspects of the testing process on a day-to-day basis and is responsible for ensuring that the individual testing projects produce the required products (that is, the outputs from the testing phases, including the fully tested AUT) to the required standard of quality and within the specified constraints of time, resources, and costs.

Where appropriate the Testing Manager is also responsible for liaison with the development teams to ensure that they follow the *Unit* and *Integration Testing* approach documented within the process. The Testing Manager will also liaise with the Independent Test Observer(s) to receive reports on testing projects that have failed to follow the testing process correctly.

The Testing Manager reports to a senior manager or director within the organization, such as the *Quality Assurance (QA) Manager* or *Information Technology Director*. In large organizations, and particularly those following a formal project management process (such as that described in Reference 3), the Testing Manager may report to a *Testing Program Board*, which is responsible for the overall direction of the project management of the *testing program*¹ (Appendix A also contains details of a typical Testing Program Board).

The Testing Manager may also be called upon to formally represent his or her parent organization during Acceptance Testing of software developed by a third party company, or COTS products acquired from a vendor. In this capacity, the Testing Manager will be expected to represent the interests of the organization as well as co-signing any Acceptance Test Certificate documentation (see Appendix H).

The position of Testing Manager may be implemented as a part time role, which is given as an additional responsibility to an existing senior manager or director (such as the QA Manager or IT Director). This is particularly likely in small organizations where the total number of staff involved in testing is relatively low.

4.3.3 Test Team Leader

The Test Team Leader is given the authority to run a testing project. His or her responsibilities include tasking one or more Test Analysts and Testers, monitoring their progress against agreed-upon plans, setting up and maintaining the testing project filing system, and ensuring the generation of the testing project artifacts, including:

- ▲ The *Test Plan document*, which will be used as the basis of project management control throughout the testing process (Appendix C contains a

¹The term “Testing Program” is used by organizations such as the British Library to denote the complete organizational specification of their testing process which is run under the PRINCE project management method (see Chapter 14 for the British Library Case Study).

specimen Test Plan document template). This document is produced by the Test Team Leader and is approved by the Testing Manager

- ▲ The *Test Specification document*, which outlines in detail the approach to testing the AUT, the resources needed, the testing environment, the evaluation criteria and acceptable defect frequencies, halting criteria, and so on. (Appendix D contains a specimen Test Specification template.) Although the Test Team Leader is responsible for ensuring that this document is produced, responsibility for its completion may be delegated to a Test Analyst.

The Test Team Leader, who reports to the Test Manager, is reported to by one or more Test Analysts and Testers. The Test Team Leader will liaise with the Independent Test Observer (to discuss availability to attend a particular test, for example), and (where appropriate) the *Development Team Leader* (to undertake early planning of the testing project and preliminary test design and to determine the availability of the AUT for testing). During Acceptance Testing the Test Team Leader is responsible for liaising with the *User Representative* and *Operations Representative* to obtain one or more users to perform User and/or Operations Acceptance Testing.

4.3.4 Test Analyst

The Test Analyst is responsible for the design and implementation of one or more *Test Scripts* (and their associated *Test Cases*), which will be used to accomplish the testing of the AUT. (Appendix E contains a specimen Test Script template, and Chapter 3 provides guidance on Test Case design.)

The Test Analyst may also be called upon to assist the Test Team Leader in the generation of the Test Specification document.

During the design of the Test Cases, the Test Analyst will need to analyze the Requirements Specification for the AUT to identify specific requirements that must be tested. During this process the Test Analyst will need to prioritize the Test Cases to reflect the importance of the feature being validated and the risk (see Section 4.9) of the feature failing during normal use of the AUT. The approach adopted can be as simple as assigning a high, medium, or low value to each Test Case. This is an important exercise to undertake, since such an approach will assist the Test Team Leader in focusing testing effort should time and resources become critical during testing.

At the completion of the testing project, the Test Analyst is responsible for the backup and archival of all testing documentation and materials, as well as the creation of a testing *Re-use*² Pack (Appendix I). These materials will be delivered to the Test Team Leader for filing. The Test Analyst is also responsible for completing a Test Summary Report briefly describing the key points of the testing project.

²Reuse is a key technique in ensuring that the testing process is as effective and as efficient as possible. This book promotes the philosophy of re-use within the testing process by providing a set of specimen templates, proformas, and checklists that staff involved in the testing process can use as the basis for developing their own project artifacts.

The Test Analyst reports to the Test Team Leader and liaises with one or more Testers to brief them on their tasks before the testing of the AUT commences.

4.3.5 Tester

The Tester is primarily responsible for the execution of the Test Scripts created by the Test Analyst and for the interpretation and documentation of the results of the Test Cases.

Prior to the execution of the Test Scripts, the Tester will set up and initialize the test environment, including the test data and test hardware, plus any additional software required to support the test (such as *simulators* and *test harnesses*).

During test execution, the Tester is responsible for filling in the *Test Result Record Forms* (TRRFs – see Appendix F) to document the observed result of executing each Test Script and for co-signing the bottom of each TRRF with the Independent Test Observer to confirm that the Test Script was followed correctly and the observed result recorded accurately. In interpreting the results of executing a Test Script, the Tester makes use of the description of the expected result recorded on the Test Script, the *Test Result Categories* specified in the Test Specification document, and his or her own personal experience. The Tester is also responsible for the recovery of the test environment in the event of failure of the system.

Following test execution, the Tester is responsible for the back-up and archival of the test data, any simulator or test harness software, and the specification of the hardware used during testing. These materials are delivered to the Test Team Leader for filing (in practice, much of this material is provided to the Test Team Leader in electronic format, such as floppy disks and CDs or copied to a shared area on a computer server). The Tester also delivers the completed TRRFs to the Test Team Leader.

The Tester reports to the Test Team Leader and liaises with the Test Analyst and the Independent Test Observer.

4.3.6 Independent Test Observer

In general terms, the Independent Test Observer is responsible for providing independent *verification* that correct procedures (that is, those specified in the testing process for the organization) are followed during the testing of the AUT.

Specifically during testing, the Independent Test Observer is responsible for ensuring that the Tester executes the tests according to the instructions provided in the Test Scripts and that he or she interprets the observed result correctly according to the description of the expected result. Where the observed and expected results differ, the Tester records the correct Test Result Categories score for the defect (as specified in the Test Specification document).

In organizations in which there is a formal *Quality Assurance Group*, Independent Test Observers may be drawn from the ranks of the *Quality Assurance Representatives* (QARs). In smaller organizations or where there is not such a strong quality culture, the Independent Test Observer may be a staff member drawn from another group or project within the organization. The key criterion in selecting an

Independent Test Observer is that she or he must be impartial and objective, that is, the individual must not have any other responsibilities within the project, be affected by the progress of the project, and be able to report accurately any failure to follow the testing process to the Testing Manager.

The Independent Test Observer liaises with the Testing Manager to report any deviations from the testing process, with the Test Team Leader to schedule attendance at the test execution, and with the Tester to confirm the result of executing an individual Test Script and to co-sign the Test Result Record form. The Independent Test Observer may also be invited to review some of the testing project artifacts produced during the testing project, such as the Test Specification document or Test Script(s) as part of the process for the testing project.

4.3.7 Supplementary Testing Roles

Depending on the specific testing requirements of a particular testing project, it may be appropriate to include a number of additional testing roles.

In practice, the responsibilities given to each of these supplementary testing roles may be assigned to existing testing roles within the project. This will depend on the size of the testing project, the number of testing staff available, the amount of testing that is required, the duration of the testing project, and so on. A review of the case studies in Part 2 will provide additional insights into how organizations and testing teams of varying sizes and with a wide variety of testing requirements have organized their roles and responsibilities to support effective and efficient testing.

The supplementary testing roles covered in this section include *Test Automation Architect*, *Automation Analyst*, and *Exploratory Tester*.

Test Automation Architect

The Test Automation Architect has the following responsibilities:

- ▲ Definition of the test automation architecture
- ▲ Specification of all hardware and software configurations needed
- ▲ Assisting the Test Team Leader to plan any programming and set-up tasks needed to install a new automation environment, including deploying the testing tools and environment-specific support code.

The Test Automation Architect reports to the Test Team Leader and liaises with the Test Analyst (Test Automation Analyst – see next section) and Testers.

Test Automation Analyst

The Test Automation Analyst has the following responsibilities:

- ▲ Implementing automated tests as specified in the Test Specification document and any supplementary design documents
- ▲ Implementing test support code required for efficiently executing automated tests

- ▲ Setting up and executing automated tests against the AUT
- ▲ Reviewing the automated test log following testing to identify and report on defects found within the AUT during testing.

The Test Automation Analyst reports to the Test Team Leader and liaises with the Test Automation Architect.

Exploratory Tester

The Exploratory Tester has the following responsibilities:

- ▲ Autonomous testing of a product without following a specified set of Test Cases
- ▲ Focusing on finding the highest severity defects in the most important features of the system using a risk-based testing approach.

This role requires a high degree of testing expertise as well as knowledge about the AUT (such as the design method or implementation technology), knowledge of the results of any earlier testing phases (particularly important in Regression Testing), experience in testing similar or related systems (and knowing where defects have arisen previously in those systems), knowledge of typical implementation errors (such as division by zero errors), and knowledge of general testing rules of thumb or heuristics. Although there are no detailed Test Cases to follow, an Exploratory Tester may follow a specified general strategy based on a combination of the above skills and experience.

The Exploratory Tester role is especially useful in testing projects with demanding timescales (such as e-business development projects) and projects where it is essential to find show-stopping defects before release to the client.

4.4 The Testing Phases

4.4.1 Overview

This section describes the management issues associated with each of the phases of testing that the AUT undergoes during its development life cycle. For each testing phase, the following items are covered:

- ▲ A brief overview of the testing phase
- ▲ A discussion of who is responsible for conducting the testing within that phase
- ▲ A discussion of who is responsible for managing the testing within that phase
- ▲ A discussion of the requirements for independent observation of the results of testing.

Chapters 5–11 describe in detail all aspects of each of the testing phases covered in the following sections (including their common synonyms), and Appendix B con-

tains a set of one-page testing guides, which can be used to provide summary advice and support to staff involved in the various phases of software testing.

This section concludes with a discussion of an important aspect of the management of testing and of each testing phase, that is, when to stop the testing process. The traditional halting criteria are reviewed, and a number of practical examples are also discussed.

4.4.2 Unit Testing

Unit Testing (also termed “software component testing”) represents the lowest level of testing that the AUT can undergo. It is conducted to ensure that reliable program units³ (or software components) are produced that meet their requirements.

Unit Testing is typically conducted by the *Development Team* and specifically by the programmer who coded the unit, the person responsible for designing and running a series of tests to ensure that the unit meets its requirements.

The Development Team Leader has responsibility for ensuring the Unit Testing is completed, for incorporating the testing task into the overall development plan, and for monitoring progress against that plan. Thus, it is unlikely that the Development Team Leader will need to produce detailed Test Plan and Test Specification documents for Unit Testing (particularly as the testing resources are likely to be provided from within the existing development team).

The results of Unit Testing should be formally documented using a Test Result Record Form, which is filed by the Development Team Leader at the end of the testing phase. The Development Team Leader may also produce a Re-use Pack (Appendix I) to allow selected Unit tests to be rerun during the later phases of testing or during Regression Testing of the AUT.

“Independent” observation of the testing process is likely to be performed by the Development Team Leader or by another member of the team, who will be expected to countersign the appropriate Test Result Record Form to confirm that the correct procedures (that is, those specified in the testing process) were followed during the testing of the AUT. If the organization has a particularly well-established quality culture, observation of the testing may be performed by a QAR assigned to assure the quality of the development project.

Chapter 5 provides comprehensive details of all aspects of Unit Testing.

4.4.3 Integration Testing

The objective of Integration Testing is to demonstrate that the modules which comprise the AUT interface and interact together in a correct, stable, and coherent manner.

³Chapter 5 provides a number of possible interpretations of the term “unit.”

The Development Team typically conducts Integration Testing, with responsibility for managing the testing falling to the Development Team Leader. Occasionally in large organizations with well-established quality and/or mature software development and testing processes, Integration Testing will be conducted by a dedicated *Testing Team*, which will be responsible for all aspects of the testing.

While the Development Team is responsible for Integration Testing, the Development Team Leader is responsible for ensuring that the Integration Testing is completed, and he or she is responsible for incorporating the testing task into the overall development plan and monitoring progress against that plan. For this reason, it is unlikely that the Development Team Leader will need to produce detailed Test Plan and Test Specification documents (particularly because the testing resources are likely to be provided from within the existing Development Team).

The results of Integration Testing should be formally documented using a Test Result Record Form, which is filed by the Development Team Leader at the end of the testing phase. The Development Team Leader should also produce a Re-use Pack (Appendix I) to allow selected Integration tests to be rerun during the later phases of testing or during Regression Testing of the AUT.

During Integration Testing and the later testing phases, it is increasingly important that truly independent observation of the testing process is performed. An Independent Test Observer may be drawn from another development project, or, if the organization has a particularly well-established quality culture, independent observation may be performed by a QA Representative assigned to assure the quality of the development project. Even where a dedicated testing team performs testing, independent observation is essential.

Another key issue to consider is the need to manage the user⁴ expectations regarding the look and feel and the operation and performance of the AUT. Many applications have successfully reached Acceptance Testing only to encounter difficulties because the user is dissatisfied with some aspect of the AUT (even though the software has already demonstrably satisfied its requirements during the earlier testing phases). Similarly, it is also possible for the requirements to have been misinterpreted and for the specification to be incorrect. The Development Team Leader must consider inviting the *User Representative* to the Integration Test (and even Unit Testing) in an informal capacity to observe the testing process. In this way, the User Representative will be exposed to the AUT prior to formal Acceptance Testing, and any show-stopping issues can be discussed and resolved in a timely manner.

Chapter 6 provides comprehensive details of all aspects of Integration Testing.

4.4.4 System Testing

The fundamental objective of *System Testing* is to establish confidence that the AUT will be accepted by the users (and/or operators) – that is, that it will pass its Acceptance Test.

⁴Where the user could be the customer representative for development or customization of an existing system with responsibility for formally accepting the AUT or could be an “internal” user with responsibility for accepting software developed by the IT department within the same organization.

System Testing is typically conducted as an independent testing project by a dedicated testing team, with responsibility for managing the testing falling to the Test Team Leader.

For small organizations in which there are few IT staff, it is possible that the development team will conduct System Testing (where this is the case, it is essential that the Development Team Leader take on all of the responsibilities of the Test Team Leader (as described in Section 4.3) and that the team nominate appropriately experienced staff from within the Development Team to fulfill the roles of Test Analyst and Tester).

During System Testing it is essential that truly independent observation of the testing process be performed. The Independent Test Observer may be drawn from another development project, or, if the organization has a particularly well-established quality culture, independent observation may be performed by a QAR assigned to assure the quality of the testing project.

As with Integration Testing, the Test Team Leader should consider inviting the User Representative to the System Test in an informal capacity to observe the testing process in order to manage the user's expectations of the AUT prior to formal Acceptance Testing.

Chapter 7 provides comprehensive details of all aspects of System Testing.

4.4.5 Systems Integration Testing

The objective of *Systems Integration Testing* is to provide confidence that the AUT is able to interoperate successfully with other specified systems and does not have an adverse affect on other systems that may also be present in the live operating environment, or vice versa.

Systems Integration Testing will be introduced into the testing process only if there is a significant requirement for the AUT to interoperate with a number of other software systems (such as the requirement for the British Library IT systems to communicate for the purposes of exchange data – see Chapter 14).

For many software systems with less rigorous requirements for interoperability, it is more likely that any aspects of Systems Integration Testing will be performed during the System Test.

Where Systems Integration Testing does appear as a distinct testing phase, it is typically conducted as an independent testing project, with responsibility for managing the testing falling to the Test Team Leader.

During Systems Integration Testing it is essential that independent observation of the testing process be performed. The Independent Test Observer may be drawn from another development project, or, if the organization has a particularly well-established quality culture, independent observation may be performed by a QAR assigned to assure the quality of the testing project.

Chapter 8 provides comprehensive details of all aspects of Systems Integration Testing.

4.4.6 Acceptance Testing

The purpose of Acceptance Testing is to confirm that the system meets its business requirements and to provide confidence that the system works correctly and is usable before it is formally “handed over” to the users. Acceptance Testing is frequently divided into *User Acceptance Testing* (involving the business or end users of the AUT) and *Operations Acceptance Testing* (involving the operations or administrative users of the AUT).

Acceptance Testing is performed by nominated user representatives under the guidance and supervision of the testing team. The Test Team Leader will obtain these staff by liaising with the appropriate User and/or Operations Representative.

During Acceptance Testing it is very important that independent observation of the testing process be performed, particularly where the user representatives are IT-naïve business staff. The Independent Test Observer may be drawn from another development project or, if the organization has a particularly well-established quality culture, independent observation may be performed by a QAR assigned to assure the quality of the testing project.

Chapters 9 and 10 provide comprehensive details of all aspects of User Acceptance Testing and Operations Acceptance Testing.

4.4.7 Regression Testing

The purpose of Regression Testing is to provide confidence that the AUT still functions correctly following modification or extension of the system (such as user enhancements or upgrades or following new builds or releases of the software).

Regression Testing is not strictly speaking a testing phase, but is a testing technique that can be applied to any of the other testing phases (for example, following enhancements to the AUT, it will be necessary to perform a System Regression Test to ensure that the existing functionality of the application is unaffected by the changes). Typically, Regression Testing is applied to the high levels of testing and to System and Acceptance Testing in particular.

Because of the need for effective and efficient testing, Regression Testing relies heavily on the re-use of existing test scripts and test cases created for previous testing phases such as System and Acceptance Testing. Similarly, it is unlikely that all of the test scripts from previous testing phases will be executed. Instead, the Test Analyst will consider the scope of changes to the AUT, the structure of the AUT, and the manner in which it was developed and his or her experience of testing the AUT (or related software) and select a subset of Test Scripts to re-use. Clearly, where higher confidence in the quality of the AUT is required (for safety critical or business critical systems, for example), more elaborate and complete testing will be necessary.

The responsibility for conducting and managing a particular Regression Test will be determined by the specific testing phase the testing is associated with. Similarly, the need for independent observation will also be dependent on the observation requirements for the associated testing phase.

Chapter 11 provides comprehensive details of all aspects of Regression Testing.

4.4.8 When to Stop Testing

One of the most difficult decisions the manager responsible for any testing phase has to make is when to stop testing.

Often the reason for halting the testing process is completely arbitrary and can include running out of the time allotted for testing, exhausting the allocated testing resources, or hitting a fixed milestone such as a contractual delivery date. From a testing perspective, such halting criteria are far from ideal and take no account of the quality of the AUT at the instant that testing has to stop.

Where the Testing Manager is able to influence the planning aspects of the software development and testing process, a number of additional criteria can be employed to provide greater confidence in the decision to halt the testing process, which provide a much clearer indication of the risk involved in stopping testing. These criteria include:

- ▲ Test Requirement Coverage – has it been demonstrated that all the requirements for the AUT have been verified? If it has not been possible to test all the requirements, have those with the highest risk associated with them been tested? (Section 4.9 discusses the need to review the requirements for the AUT and prioritize them according to their impact and likelihood of their failure.)
- ▲ Test Code Coverage – has it been demonstrated that all “parts” of the software have been exercised during testing (including exception handling and error handling routines)?
- ▲ Test Case Metric – how many Test Cases have been planned, designed, implemented, executed, and passed or failed? This is a useful metric to collect to measure the progress of the testing activity (see Chapter 12 for metrics).
- ▲ Defect Detection Metric – has the rate of defect detection been plotted, and has the rate of detection of defects leveled off? This provides a reasonable indication that the majority of the defects have been identified (however, caution must be exercised using this approach, since tester fatigue can produce similar results).

There are other, more esoteric methods of estimating the numbers of defects in the AUT, such as seeding the software with “known” defects and comparing the ratio of known to unknown defects observed during testing. Mathematically, this ratio can be used to provide an estimate of the total number of defects in the AUT, and hence provide an effective halting criterion. In practice, such methods are difficult to manage, provide imperfect results, and may be hazardous (for example, it is essential that the AUT be delivered without any of the seeded defects still in place).

4.5 Role of the V Model in Planning

Typically, 25–50% of the total costs of developing a software system are spent on test activities (Reference 1). This figure is likely to be significantly larger for high-reliability, business critical and safety critical systems. Project plans must reflect this issue in their allocation of time, effort, and resources to testing.

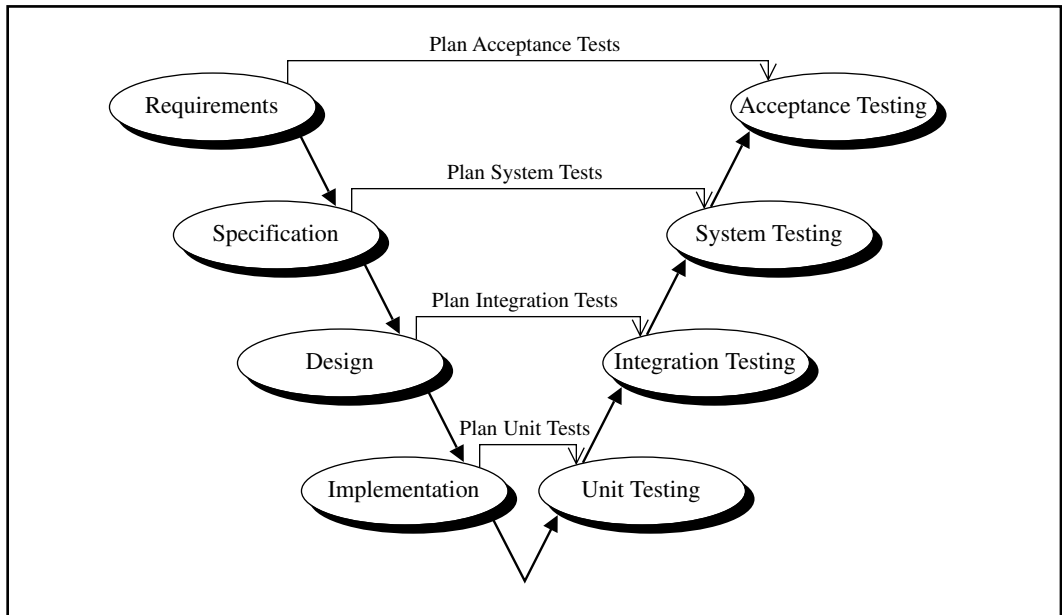


Figure 4.2. The V Model.

The *V Model* (see Figure 4.2) is a software development and testing model, which helps to highlight the need to plan and prepare for testing early in the development process. The left-hand descending arm of the “V” represents the traditional Water-fall development phases (Reference 18), whereas the ascending right-hand arm of the “V” shows the corresponding testing phase.

In the V Model, each development phase is linked to a corresponding testing phase. As work proceeds on a particular development phase, planning and some preliminary design work is performed for the corresponding testing phase. The benefits of the V Model include:

- ▲ The testing phases are given the same level of management attention and commitment as the corresponding development phases in terms of planning and resourcing, allowing any risks to be addressed early in the development cycle (for example, if a highly specialized piece of hardware necessary to perform Acceptance Testing has a long order and delivery time, identifying this requirement early allows the hardware to be ordered in a timely manner – mitigating the risk of late delivery of the tested system).
- ▲ The outputs from the development phases (such as specification and design documents) can be reviewed by the testing team to ensure their testability (for example, ensuring that a given requirement has been documented in such a manner that it is easily validated).
- ▲ The early planning and preliminary design of tests provides additional review comments on the outputs from the development phase (for example, if there are errors in the initial requirements, they have a greater chance of being detected if the Acceptance Test Specification is produced before development proceeds to detailed analysis and design. During this process, it is possible to identify duplicated and/or contradictory requirements and feed these back to the analyst for clarification).

Although the Testing Manager is responsible for the timely set-up and initiation of the testing for a particular software development project, the main responsibility for exploiting the power of the V Model will fall to the Test Team Leader, who will need to liaise with the Development Team Leader to ensure that the early planning and preliminary design tasks have been conducted and completed.

Although the V Model is described in terms of the traditional Waterfall view of software development, the principles of forward planning and design apply equally to the more modern iterative approaches to development (such as that described in References 8 and 12). In an iterative approach, each iteration can be viewed as a “mini-waterfall” with stages analogous to those seen in the Waterfall view of development.

In practice, the V Model provides a powerful tool for managing and controlling risk within the testing component of a software development project. The process of bringing testing planning and design into the development process as early as possible enables risks to be identified and strategies for removing or mitigating them to be put in place in a timely manner.

4.6 The Management of Test Requirements

In performing effective and efficient testing, it is essential that the Test Analyst know precisely what the requirements of the AUT are. This information is the basis of the formal *validation* that the AUT meets its requirements. Without access to this information, the Test Analyst has little or no basis for performing the design of the Test Scripts and Test Cases.

Incomplete or poorly maintained requirements may result in the Test Analyst not designing a test for an important requirement, or may cause the Test Analyst to duplicate testing effort by generating several tests that verify the same requirement.

The acquisition, maintenance, management, and communication of requirements is a traditional source of quality problems in software development and testing projects. Because of the complexity of the task combined with the number of requirements that are generated for any but the most trivial of development projects, the majority of software development projects use requirements management tools.

However, even where tools are used, the problem is often exacerbated by poor communication between the various roles involved in the development and testing process, with analysts, developers, and testers all using different tools to represent requirements, design information, and testing data.

For example, a particular software project the author was involved in auditing had problems with poor-quality deliverables, with a high rate of defects reported by the end users. At first glance, the project seemed to be running smoothly: the analysts were eliciting requirements using Use Cases (Reference 7) and recording them and managing them using a proprietary database system; the designers were taking printouts of the requirements and creating visual models in the Unified Modelling Language (Reference 6) using a proprietary drawing package; and the testers were taking printouts of the requirements and the UML diagrams, using good test design techniques to create the test procedures, and then representing them in a proprietary spreadsheet package. However, even though individually each of the staff on

the project was performing very professionally, using good techniques and good practice, in fact, the communication between the individual roles was very poor, with significant scope for loss, duplication, or omission of requirements and design information prior to it reaching the testers. Further, if any of the requirements or design information changed, these had to be provided manually to the testing team by means of supplementary printouts.

If tool support for requirements management is to be adopted, it is essential that analysts, developers, and testers all have access to the same tool and that it is easy to communicate the requirements information between the project roles. In this way, the Test Analyst can examine the most up-to-date copy of the requirements, ensuring complete testing coverage of the features of the AUT and avoiding duplication or omission of tests (Reference 19 provides a typical example of a requirements management tool that satisfies the above requirements).

4.7 The Role and Use of Configuration Management

A major requirement of testing-process management is that the results of testing be reproducible, allowing testers to recreate the conditions of a previous test and be able to repeat exactly the testing and accurately reproduce the results (for example, it may be necessary to repeat a particular test in response to an observation raised against the tested AUT by the users on the live system).

In order to support this requirement, testing must be conducted under strict and effective *configuration management* (CM). The minimum testing artifacts that must be placed under CM control are:

- ▲ The AUT
- ▲ The Test Plan and Test Specification documents
- ▲ The test data
- ▲ Any supporting software (such as test harnesses, simulators, or stimulators)
- ▲ The specification of the test hardware or test rig (if this is not already included in the Test Specification document)
- ▲ The Test Scripts and component Test Cases.

The CM of testing projects is a complex and difficult task, and tool support should be considered (a number of CM tools are commercially available, such as Reference 20).

4.8 The Role and Use of Defect Tracking

Effective testing is nothing without an accurate means of reporting defects, monitoring their progress, and managing their resolution. A formal approach to defect tracking must be employed to make sure the effort expended on testing is well spent by ensuring that defects detected during testing are corrected.

In setting up a defect tracking system, the following facilities must be provided:

- ▲ The ability to register a defect and its characteristics and to associate the defect with a unique identifier
- ▲ The ability to associate a severity value with that defect (see *Test Result Category* in the Glossary)
- ▲ The ability to associate a priority of resolution with that defect
- ▲ A defined workflow, in which the details of the defect will be provided to the appropriate staff within the organization, and its status (such as new, open, or closed) altered as it progresses through the system
- ▲ Defined rights and privileges of the staff involved in the defect tracking process (for example, in a particular organization, a Tester may be able to report a defect, but only the Testing Manager may close a defect following successful retesting)
- ▲ The ability to customize the above attributes of the defect tracking system to match more closely the particular requirements of the host organization.

As with CM, rigorous and effective defect tracking is a complex and difficult task, particularly where there are multiple development and testing projects operating simultaneously. The majority of organizations involved in testing employ one of the many defect tracking tools that are available (such as Reference 21) or develop one of their own in-house. The latter approach suffers from a number of problems, which include reliability, maintenance costs, lack of professional support, and possible loss of tool expertise due to career change.

4.9 The Role of Risk in Test Planning and Management

The term “risk” has appeared in several places in this chapter, including discussions of the planning and design of tests. “Management of risk” and “mitigation of risk” are popular terms in project management, and particularly in IT project management at present.

Chapter 2 showed how testing can be defined in the context of risk, and this leads us to consider the management and planning of testing in a risk-based manner as well. In this context, effective management and planning of the testing process provide opportunities for identifying risk within the AUT⁵ in a timely manner and allow such risks to be investigated (by means of testing), removed, or mitigated as quickly as possible.

The V Model provides the Testing Manager and Test Team Leader with an effective means of identifying risk as early as possible in the software development life-cycle and allowing the removal or mitigation of the risk. This process can involve

⁵This might be the risk of failure of some key feature during its normal operation following delivery of the software, or the failure of the AUT to meet a specific requirement specified by the user.

all the stakeholders in the development process, including the managers (in terms of the need to modify plans, for example), developers (in terms of the need to rework some aspect of the AUT, for example), and even the users (allowing respecification of missing, duplicate, or contradictory requirements, for example).

Within the testing process, the Test Analyst must take account of risk during the design of the Test Cases, ensuring that the Test Cases are prioritized in such a way as to test high-risk aspects of the AUT before less risky aspects of the software.

Reference 28 provides a high-level view of risk-based testing: generate a prioritized list of risks, perform testing to explore each risk, and as risks are removed (and new ones are identified), adjust the testing effort to focus on the remaining list of risks. A number of strategies for achieving this high-level approach to risk-based testing are described in Reference 28; References 29 and 30 provide further reading on methods and tools that can be used for risk management within testing projects.

Unit Testing

(The Programmer) personally defines the functional and performance specifications, designs the program, codes it, tests it, and writes its documentation. . . . They need great talent, ten years experience and considerable systems and applications knowledge, whether in applied mathematics, business data handling, or whatever!

Fred P. Brooks

5.1 Overview

The objective of *Unit Testing* is to ensure that reliable program units are produced that meet their requirements. The process is also intended to identify errors in program logic. Unit Testing is conducted by the development team under the supervision of the *Development Team Leader*. Typically, the software engineer or programmer who coded the unit will design and run a series of tests to verify that the unit meets its requirements. Unit Testing may also be termed component testing.

Each unit should be tested individually and in isolation (although it may be necessary to employ a *test harness*) by exercising its inputs and observing its outputs or behavior. It may also be possible for the unit to be tested using the facilities available in the development environment (such as stepping through the statements of code using a debugger). Once confidence has been established in the unit, it should then be tested in collaboration with other interoperating units.

Unit Testing is typically a *White Box* Testing activity, based primarily on the functional and data requirements expressed in the *requirements specification* for the *Application Under Test* (AUT), as well as any supplementary material (such as design documentation, user guides, or prototype code). Examples of areas typically covered in Unit Testing include:

- ▲ Correctness of calculations/manipulations performed by the unit
- ▲ Low-level performance issues (such as performance bottlenecks observed under repeated invocation of the unit and its functionality)
- ▲ Low-level reliability issues (such as memory leaks observed under repeated and extended invocation of the unit and its functionality)
- ▲ Screen/window contents and navigation (for graphical user interface [GUI] units) including consistency of the “look and feel” of the window, use of “hot keys,” and function keys and keyboard shortcuts

- ▲ Report contents, layouts, and calculations
- ▲ File/record creation, update, and deletion
- ▲ Communication between interoperating units.

The precise definition of a “unit” will depend on the approach to design of the AUT and the implementation technology employed in its development, for example:

- ▲ A unit in an application developed using a procedural programming language could be represented by a function or procedure, or a closely related group of functions or procedures.
- ▲ A unit in an application developed using an object-oriented programming language could be represented by a class or an instance of a class, or the functionality implemented by a method.
- ▲ A unit in a visual programming environment or a GUI context could be a window or a collection of related elements of a window, such as a group box.
- ▲ A unit in a component-based development environment could be a predefined re-usable component. In such circumstances, the developer testing the component will need to consider the component’s origin, maturity, and previous testing history and determine how much testing he or she needs to perform themselves.

In terms of the *V Model*, Unit Testing corresponds to the Implementation phase of the software development lifecycle (Figure 5.1).

Testing issues associated with Unit Testing (such as test planning and resourcing, and the review of testing requirements, and high-level test design) should be considered during the Implementation phase of the AUT. The V Model and its role in testing are described in Chapter 4.

A flow chart providing a high-level overview of the Unit Testing process can be found at the end of this chapter (Figure 5.2).

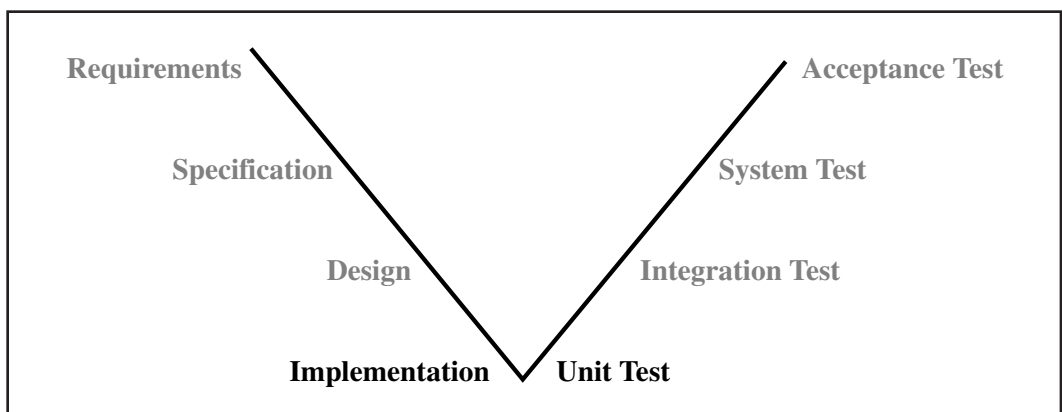


Figure 5.1. Unit Testing and the V Model.

5.2 Unit Test Approach

The following approach should be followed in conducting Unit Testing:

- ▲ Determine how a given unit may be invoked and what its response will be under each invocation (by reviewing the appropriate design documentation produced during the development of the AUT, for example).
- ▲ Determine what states the unit can be in and how the unit moves between these states (consider using the State Transition techniques described in Chapter 3).
- ▲ Determine the expected results for each state the unit may be in (or combination of states) and what will be necessary to confirm these results (it may be useful to generate a table of inputs and outputs to record this information).
- ▲ Review the requirements specification document for the AUT and the unit under test to confirm that the previous steps have identified all possible test conditions, and add further conditions as necessary (consider data initialization, boundary values, error conditions, and division by zero).
- ▲ Determine any prerequisites for testing the unit (such as output from other units, the need for a test harness, or specific handcrafted test data).
- ▲ Use the above information to generate a set of *Test Cases* to test the unit and incorporate them into a *Test Script*, describing in detail the steps to be followed during testing. Formal Test Case design techniques should be employed to ensure that boundary values and error conditions are correctly exercised.

Where they are available, consider re-using Test Cases from earlier Unit Tests. These can be obtained from the appropriate Unit Test *Re-use Packs* (see Appendix I, for example).

5.3 Unit Test Data Requirements

The data used during Unit Testing is unlikely to be *live data* because:

- ▲ There is a risk of corrupting commercially important or business critical information.
- ▲ The live data may contain sensitive information (such as commercial or personal details).
- ▲ The data may contain information of a secure nature (such as user passwords).

Where the function of the unit under test does not involve the manipulation or use of large volumes of data, it is likely that the *Test Analyst* will use a small representative set of handcrafted test data. In creating the test data, the Test Analyst must employ

formal design techniques, such as the *Boundary Analysis* or *Equivalence Partition* methods described in Chapter 3, to ensure the data adequately test the boundary conditions for the unit as well as to provide truly representative values.

Where the unit under test is responsible for manipulating large volumes of data, and where this will be a requirement for a number of units, it may be worth considering taking a copy of the live data (which may need to be “sanitized”¹ if it contains sensitive information), or a smaller representative sample of the live data, for the purposes of testing. The Test Analyst may still need to consider the introduction of some handcrafted data into the sampled data for the purpose of testing a specific feature of the unit, such as the response to error conditions.

Where a unit will receive data from a remote source when it is incorporated into the complete AUT (from a Client/Server system, for example), it may be necessary to *simulate* access to such data during the Unit Test using test harness software. In considering such an option, the Test Analyst must bear in mind the effort involved in developing the test harness software and the need to test the test harness itself before it can be used. Time and effort may be saved if such software is already available for re-use, perhaps having been developed for earlier testing of the AUT.

Similarly, in identifying the data requirements for Unit Testing, it is important for the Test Analyst to consider the role of re-use as a means of obtaining data for the current testing (from previous tests). If it has been necessary to hand craft some data for the purposes of conducting a Unit Test, the Test Analyst must make sure that a copy of this data is included in the Re-use Pack (see Appendix I) for possible use in later testing phases or for *Regression Testing*.

5.4 Roles and Responsibilities

Unit Testing is typically conducted by the development team under the supervision of the *Development Team Leader*, who is responsible for ensuring that adequate testing is performed using appropriate testing techniques and under suitable quality control and supervision (as defined in the Unit Test Plan and *Test Specification documents*).

Typically, the same member of the development team who coded the unit will design the required Test Case(s), test the unit, and correct any defects (effectively performing the roles of Test Analyst and *Tester*). Testing includes the design and execution of the *Test Scripts* and Test Cases and the completion of the *Test Result Record Form* and Unit Test Log.

At Unit Testing, it is desirable that the testing process be monitored by an *Independent Test Observer*, who will formally witness the results of individual Test Cases. The need for a genuinely Independent Test Observer is not as stringent at Unit Test

¹Where live data contains confidential data or information relating to security issues, it may be inappropriate to conduct unit testing using the original data. One solution is to use a copy of the live data that has been altered to remove or change those items of a confidential or secure nature.

as it is during the subsequent testing phases; hence the observer could be drawn from the development team (such as the Development Team Leader or another member of the team). A *Quality Assurance Representative (QAR)* could also fill the Independent Test Observer role, if such staff exist within the organization.

The Development Team Leader will liaise with the *Test Team Leader* (assuming a *Testing Team* for the AUT has been constituted at this stage of the development project) to report on the progress of the development of the AUT and likely dates for delivery of the AUT for subsequent testing purposes. The Test Team Leader should be invited to observe one or more Unit Tests (although in practice it is likely that the Test Team Leader will simply review the copies of the deliverables produced during the Unit Test process to see that the testing was conducted adequately).

The Development Team Leader may consider inviting a *User Representative* to informally observe Unit Testing, particularly where the testing involves aspects of the AUT that deal with the business logic of the system or the operation of the user interface. Unit Testing provides a valuable opportunity to expose the User Representative to aspects of the operation and appearance of the AUT, allowing (informal) feedback to be obtained and helping to manage the expectations of the User Representative prior to formal *Acceptance Testing* (see Chapter 4 for a fuller explanation of the need to involve users in the testing process).

The Development Team Leader will provide the Test Team Leader with copies of a number of outputs produced as a result of the Unit Testing process (and listed in Section 5.8), including the completed Test Result Record forms, Unit Test Log, and a brief Unit Test Summary report when Unit Testing is complete. In the event that the Testing Team is constituted after Unit Testing has been completed, the Development Team Leader should provide the Test Team Leader with copies of the appropriate outputs as soon as possible.

5.5 Planning and Resources

The planning of Unit Testing is the responsibility of the Development Team Leader and should be developed with reference to the overall development and testing plan for the AUT.

The human resources required for Unit Testing will be drawn from the development team (with the possible exception of the Independent Test Observer, who may be drawn from another project or from the QA Group, should one exist within the organization). At Unit Test, the Independent Test Observer role could be performed by the Development Team Leader or another nominated member of the development team.

It is assumed that Unit Testing will not take place in the *live environment* but in the *development environment* or in a separate *testing environment*. The time and duration of the Unit Test must be integrated into the development plan for the AUT in order to minimize the impact on the development process caused by Unit Testing and to ensure that no contention for resources will occur.

It is essential that the Unit Test Plan take account of the time and effort required for any correction of defects and retesting.

5.6 Inputs

The following items are required as inputs to Unit Testing and should be complete and available to the Development Team Leader before test execution can begin:

- ▲ The Requirements Specification document (for the AUT)
- ▲ The design documents for the AUT
- ▲ Any supplementary material, such as user guides or prototype software
- ▲ The Unit Testing Plan (see Appendix C)
- ▲ The Unit Test Specification document (see Appendix D)
- ▲ The Unit Testing Guide (see Appendix B)
- ▲ Any Unit Test Scripts and Unit Test Cases (see Appendix E)
- ▲ Blank Test Result Record forms (see Appendix F)
- ▲ Any appropriate Re-use Packs, such as any from previous Unit Tests (see Appendix I).

5.7 Testing Techniques for Unit Testing

The following testing techniques are appropriate for Unit Testing:

- ▲ *Functional Testing* against the requirements for the unit under test
- ▲ *Static Testing* (such as code review and code walk-through)
- ▲ White Box Testing (since the structure of the unit will be known to the developer)
- ▲ *State Transition Testing* (particularly where it is possible for the unit to be in a number of different states, and to assist in negative testing of the unit)
- ▲ *Nonfunctional Testing* (where appropriate, for *performance*, *stress*, or *reliability testing* of the unit, for example).

See Chapter 3 for details of these testing techniques.

5.8 Outputs

The following items are generated as outputs from Unit Testing:

- ▲ The fully tested units
- ▲ The completed Unit Test Certificate (see Appendix H) where formal proof of testing is required
- ▲ Any revised Test Script and Test Cases (where appropriate)
- ▲ The archived test data
- ▲ The completed Test Result Record forms (see Appendix F)
- ▲ The Unit Test Log (see Appendix G)
- ▲ The Unit Test Re-use Pack (see Appendix I)
- ▲ A brief Unit Test Summary Report (see Appendix J).

Unit Testing will be considered complete when all of the above deliverables (under strict configuration management, as described in Chapter 4) are complete and have been provided to the Development Team Leader.

Appropriate deliverables (that is, all of the above except the fully tested units) should be stored in the project file. The fully tested units (plus any associated test harness or simulation code and test data) should be backed up and archived.

Copies of the filed deliverables (under strict configuration management) should be provided to the Test Team Leader.

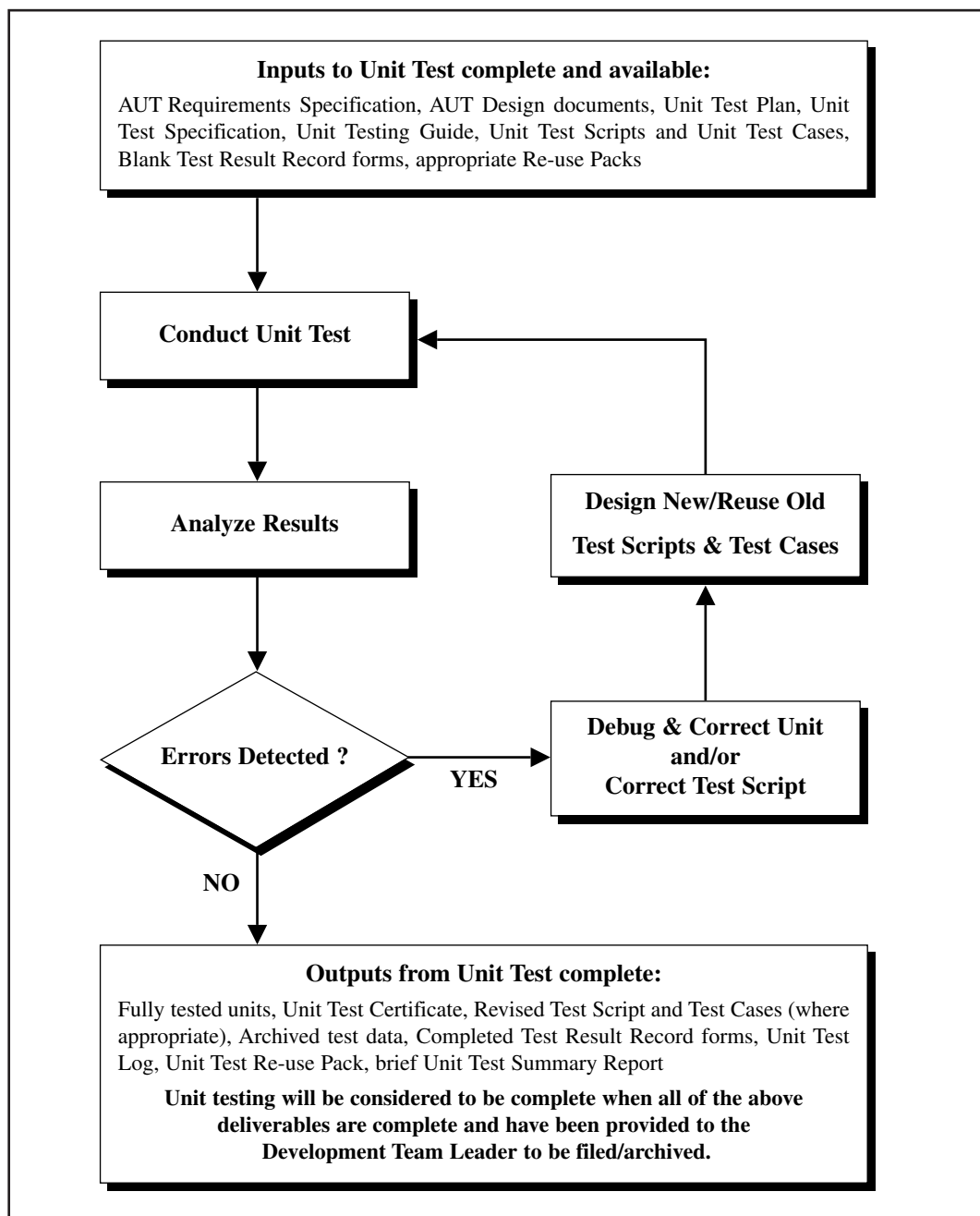


Figure 5.2. High-level overview of the Unit Testing Process.

Integration Testing

Computers are like Old Testament gods; lots of rules and no mercy.

Joseph Campbell

6.1 Overview

The objective of *Integration Testing* is to demonstrate that the modules which comprise the *Application Under Test (AUT)* interface and interact together in a correct, stable, and coherent manner prior to *System Testing*. Integration Testing is typically conducted by the development team and involves independent observation of the testing process. Integration Testing may also be termed *Link* or *Module Testing*, but should not be confused with *Systems Integration Testing* (see Chapter 8).

Testing should be performed against the *functional requirements* of the AUT using *Black Box* Testing techniques. *Test Case* design should demonstrate correct interfacing and interaction between modules, but should avoid any duplication of *Unit Testing* effort (however, where appropriate, re-use of unit tests should be considered in order to reduce testing timescales and effort).

The precise definition of a “module” will depend on the approach to the design of the AUT and the implementation technology employed; for example:

- ▲ A module in an application developed using a procedural programming language could be represented by a closely related group of functions or procedures that perform a well-defined service within the AUT and that communicate with other component modules via strictly defined interfaces.
- ▲ A module in an application developed using an object-oriented programming language could be represented by a collection of objects that perform a well-defined service within the AUT and that communicate with other component modules via strictly defined interfaces.
- ▲ A module in a visual programming environment could be a window or a collection of subwindows that perform a well-defined service within the AUT and which communicate with other component modules via strictly defined interfaces.
- ▲ A module in a component-based development environment could be a reusable component that performs a well-defined service within the AUT and that communicates with other modules via a strictly defined (and documented) interface. Under such circumstances, the developer testing the component will need to consider the component’s origin, maturity, and previous testing history in order to determine how much additional testing is required.

In terms of the *V Model*, Integration Testing corresponds to the design phase of the software development lifecycle (Figure 6.1).

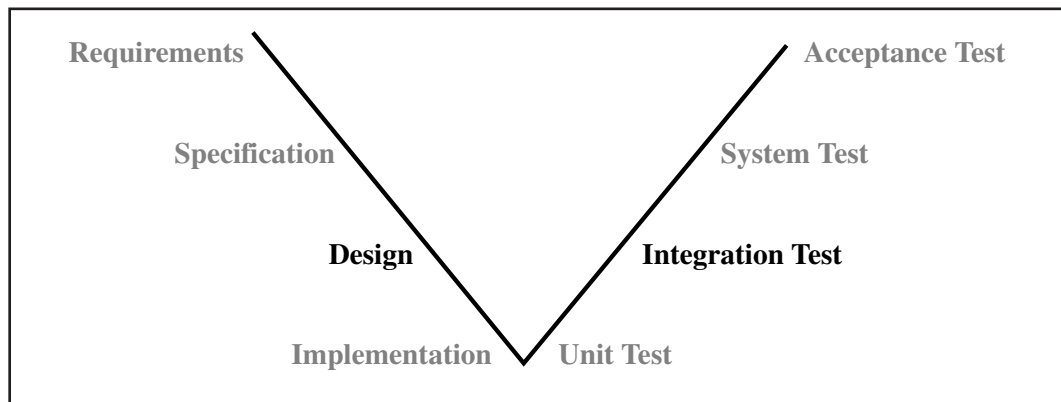


Figure 6.1. Integration Testing and the V Model.

Testing issues associated with Integration Testing (such as test planning and resourcing and the review of testing requirements and high-level test design) should be considered during the design phase of the AUT. The V Model and its role in testing are described in Chapter 4.

A flow chart providing a high-level overview of the Integration Testing process can be found at the end of this chapter (Figure 6.2).

6.2 Integration Test Approach

Integration Testing is typically a *Black Box Testing* activity, based primarily on the design of the AUT plus functional and data requirements expressed in the specification document(s). Examples of areas typically covered in Integration Testing include:

- ▲ Invocation of one module from another interoperating module
- ▲ Correct transmission of data between interoperating modules
- ▲ Compatibility (that is, checking that the introduction of one module does not have an undesirable impact on the functioning or performance of another module)
- ▲ Nonfunctional issues (such as the reliability of interfaces between modules).

The following approach should be followed in conducting Integration Testing:

- ▲ Identify the relationships between the modules that comprise the complete system (by reference to the design documentation for the AUT, for example).
- ▲ Review the requirement for interaction and communication between modules, and identify the interfaces between the modules (by reference to the *Requirements Specification* document for the AUT, for example).
- ▲ Use the above information to generate a set of *Test Cases* to test the module(s) and incorporate them into a *Test Script*, describing in detail the steps to be followed during testing.

- ▲ Incremental testing should be employed where successive modules are added to the (enlarging) system and the new combination tested (use of Re-use Packs should be considered). This process is repeated in a logical/functional sequence until all modules have been successfully integrated to form the complete system.

The process of Integration Testing involves a large component of Configuration Management: Are the proper versions of the components going into the integration build? Have the components themselves passed their earlier tests? Do the components compile together without any problems?

Where they are available, consider re-using Test Cases from earlier Unit Tests to verify the quality of the components. These can be obtained from the Unit Test Re-use Pack compiled during Unit Testing.

6.3 Integration Test Data Requirements

Because the principal purpose of Integration Testing is to demonstrate that the modules comprising the AUT interface and interact together correctly, the data requirement (both in terms of volume and content) is typically not very demanding.

Live data is unlikely to be used during Integration Testing because of the risk of corrupting commercially important or business critical information, because the live data may contain sensitive information (such as commercial or personal details) or because the data may contain information of a secure nature (such as user passwords).

Under these circumstances, it is likely that the *Test Analyst* will use a small representative set of handcrafted test data. In creating the test data, the Test Analyst must employ formal design techniques, such as the *Boundary Analysis* or *Equivalence Partition* methods described in Chapter 3, to ensure the data adequately test the boundary conditions for the module as well as to provide truly representative values.

Where a module will receive data from a remote source when it is incorporated into the complete AUT (from a Client/Server system, for example), it may be necessary to *simulate* access to such data during the Integration Test using *test harness* software. In considering such an option, the Test Analyst must bear in mind the effort involved in developing the test harness software and the need to test the test harness before it can be used. Time and effort may be saved if such software is already available for re-use, perhaps having been developed during the Unit Testing of the AUT.

Similarly, if it has been necessary to generate test data during Unit Testing, it may be appropriate to re-use this data during Integration Testing. This data should be available from the Unit Test Re-use Pack.

6.4 Roles and Responsibilities

Integration Testing is typically conducted by the development team under the supervision of the *Development Team Leader*, who is responsible for ensuring that adequate Integration Testing is performed using appropriate testing techniques and

under suitable quality control and supervision (as defined in the Integration Test Plan and Test Specification documents).

At Integration Testing, it is important that the testing process be monitored by an *Independent Test Observer*, who will formally witness the results of individual Test Cases. The Independent Test Observer could be drawn from the *Quality Assurance (QA) Group* in an organization where such a group exists or from another development team or project.

The Development Team Leader will liaise with the *Test Team Leader* to report on the progress of the development of the AUT and on likely dates for delivery of the AUT for System Testing purposes. The Test Team Leader will be invited to informally observe the Integration Test or to nominate a suitably qualified member of the *Testing Team* to attend in his or her place.

The Development Team Leader should consider inviting a *User Representative* to informally observe Integration Testing, particularly where the testing involves aspects of the AUT that deal with the business logic of the system or the operation of the user interface. Integration Testing provides a valuable opportunity to expose the User Representative to aspects of the operation and appearance of the AUT, allowing (informal) feedback to be obtained and helping to manage the expectations of the User Representative prior to formal *Acceptance Testing* (see Chapter 4 for a fuller explanation of the need to involve users in the testing process).

The Development Team Leader will provide the Test Team Leader with copies of a number of outputs produced as a result of the Integration Testing process (and listed in Section 6.8), including the completed Test Result Record forms, an Integration Test Log, and an Integration Test Summary report when Integration Testing is complete.

6.5 Planning and Resources

The planning of Integration Testing is the responsibility of the Development Team Leader and should be conducted with reference to the overall development plan for the AUT.

The human resources required for Integration Testing will be drawn from the development team (with the exception of the Independent Test Observer, who may be drawn from another project or from the QA Group, should one exist within the organization).

It is assumed that Integration Testing will not take place in the *live environment* but in the *development environment* or in a separate *test environment*. The time and duration for the Integration Testing must be integrated into the development plan for the AUT in order to minimize the impact on the development process caused by Integration Testing and to ensure that no contention for resources will occur.

In the event that Integration Testing does take place on the live environment, the Development Team Leader must liaise with the *IT Systems Administrator* to plan installation of the AUT prior to testing and to ensure that the manager is aware of the date, time, and duration of the Integration Test.

It is essential that the Integration Test Plan take account of the time and effort required for any correction of defects and for retesting.

6.6 Inputs

The following items are required as inputs to Integration Testing and should be complete and available to the Development Team Leader before testing can begin:

- ▲ The Requirements Specification document (for the AUT)
- ▲ The design documents for the AUT
- ▲ Any supplementary material, such as user guides or prototype software
- ▲ The Integration Testing Plan (see Appendix C)
- ▲ The Integration Test Specification document (see Appendix D)
- ▲ The Integration Testing Guide (see Appendix B)
- ▲ Any Integration Test Scripts and Test Cases (from the Test Analyst)
- ▲ Blank *Test Result Record forms* (see Appendix F)
- ▲ Any appropriate *Re-use Packs*, such as the Unit Test Re-use Pack (see Appendix I).

6.7 Testing Techniques for Integration Testing

The following testing techniques are appropriate for Integration Testing:

- ▲ *Functional Testing* using Black Box Testing techniques against the interfacing requirements for the module under test
- ▲ *Nonfunctional Testing* (where appropriate, for *performance* or *reliability testing* of the module interfaces, for example)
- ▲ Where appropriate, some functional testing against relevant inter-module functionality (again, using Black Box Testing techniques).

See Chapter 3 for details of these testing techniques.

6.8 Outputs

The following items are generated as outputs from Integration Testing:

- ▲ The fully tested and integrated modules
- ▲ The completed Integration Test Certificate (see Appendix H) (where formal proof of testing is required)
- ▲ Any revised *Test Scripts* and Test Cases (where appropriate)
- ▲ The archived test data
- ▲ The completed Test Result Record forms (see Appendix F)
- ▲ The Integration Test Log (see Appendix G)
- ▲ The Integration Test Re-use Pack (see Appendix I)
- ▲ A brief Integration Test Summary Report (see Appendix J).

Integration Testing will be considered complete when all of the above deliverables (under strict configuration management, as described in Chapter 4) are complete and have been provided to the Development Team Leader.

Appropriate deliverables (that is, all of the above except for the fully tested modules) should be stored in the project file. The fully tested modules (plus any associated *test harness* or simulation code and test data) should be backed up and archived.

Copies of the filed deliverables should be provided to the Test Team Leader prior to the commencement of System Testing.

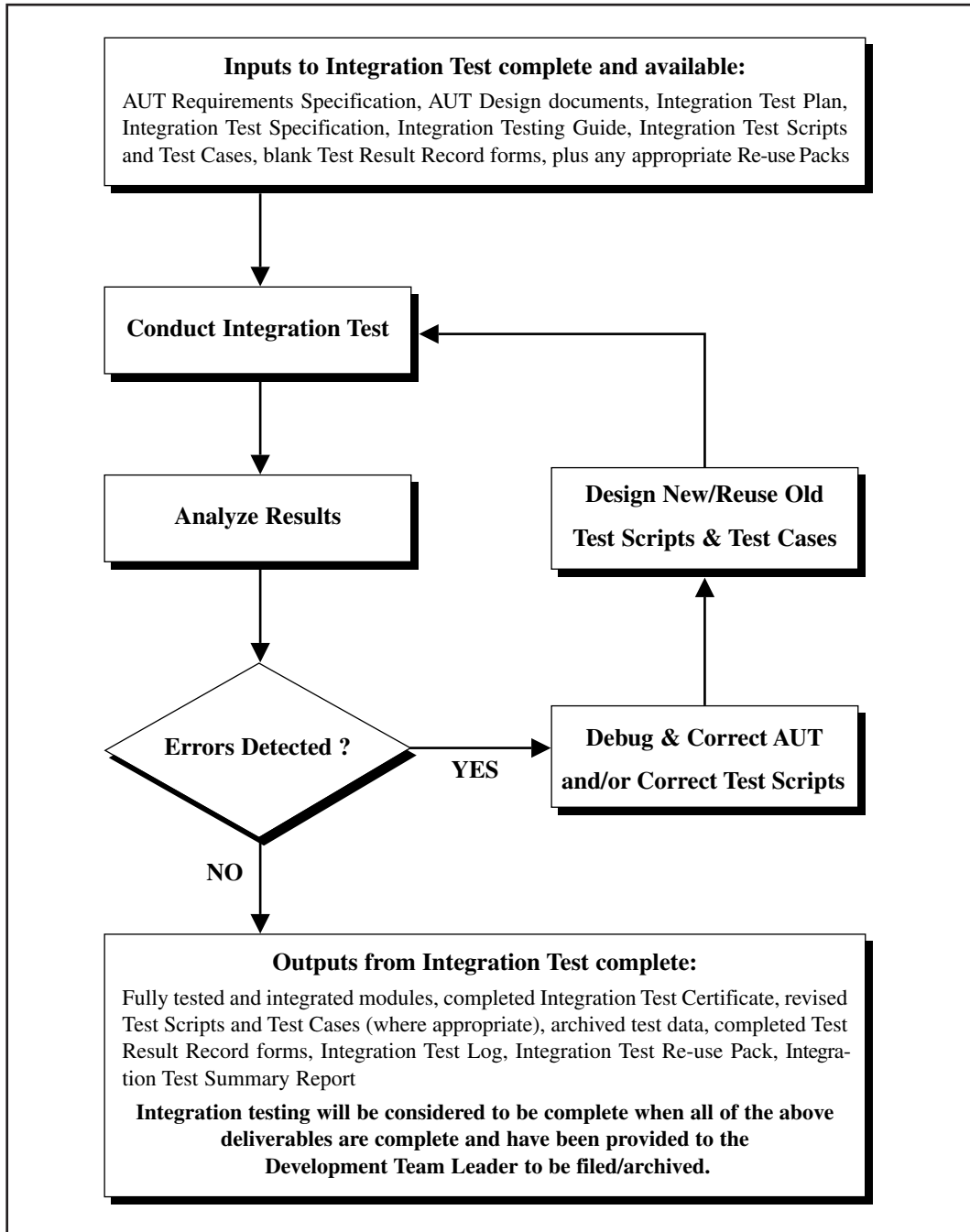


Figure 6.2. High-level overview of the Integration Testing process.

System Testing

Removing the Needles from the Haystack without disturbing the Straws

7.1. Overview

The objective of *System Testing* is to establish confidence that the *Application Under Test (AUT)* will be accepted by its users (and/or operators), that is, that it will pass its *Acceptance Tests*. During System Testing, the functional and structural stability of the system will be demonstrated, as well as *nonfunctional* requirements such as performance and reliability. System Testing is conducted by the *Test Team* under the supervision of the *Test Team Leader*.

Where the AUT has a moderate requirement to interoperate with one or more collaborating software systems, this may be tested during Systems Testing. Where the AUT has a significant requirement to interoperate with several collaborating software systems and/or the nature of the interoperation is complex, this will typically be tested during a separate testing phase termed *Systems Integration Testing* (see Chapter 8 for further details).

System Testing should employ *Black Box* Testing techniques and will test the high level requirements of the system without considering the implementation details of the component modules (for example, performing *Thread Testing* and/or testing complete transactions).

In terms of the *V Model*, System Testing corresponds to the Specification phase of the software development lifecycle (Figure 7.1).

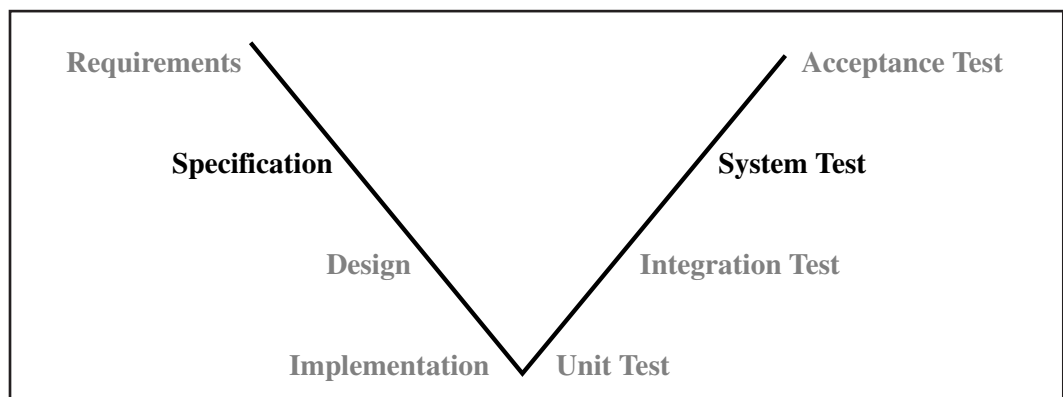


Figure 7.1. System Testing and the V Model.

Testing issues associated with System Testing (such as test planning and resourcing and the review of testing requirements and high-level test design) should be considered during the Specification phase of system development. The V Model and its role in testing are described in Chapter 4.

A flow chart providing a high-level overview of the System Testing can be found at the end of this chapter (Figure 7.2).

7.2 System Test Approach

The following approach should be followed in conducting System Testing:

- ▲ Review the requirements for the system (by reference to the *Requirements Specification* document for the AUT, for example) and identify:
 - the high level business requirements
 - the requirement for data handling and transaction rates for the live system
 - the requirements for system performance
 - backup and recovery requirements
 - any security requirements.
- ▲ Identify any requirements for the AUT to communicate with other systems and the means of communication (where this is not addressed by Systems Integration Testing – see Chapter 8).
- ▲ Review the computing environment in which the live system will run in order to identify any interoperability or compatibility issues with other systems (where this is not addressed by Systems Integration Testing – see Chapter 8).
- ▲ Examine the requirement for testing system procedures (such as the installation procedure), system documentation (such as User Manuals), and help facilities (both paper-based and interactive) plus recovery, back-up, and archive.
- ▲ Use the above information to generate a set of *Test Cases* to test the system and incorporate them into a *Test Script*, describing in detail the steps to be followed during testing.

Where they are available, consider re-using Test Cases from earlier *Unit* and *Integration* testing. These can be obtained from the appropriate Unit and Integration *Test Re-use Packs*.

7.3. System Test Data Requirements

Since one of the major goals of System Testing is to establish confidence that the AUT will pass its acceptance test, the data used for testing must be as accurate and as representative of the *live data* as possible. Similarly, since *Performance Testing* will take place at System Testing, the quantity of data available for testing must also be of equivalent size and complexity.

One approach to achieving the above requirement for test data is to use live data. A significant benefit of this approach is that the System Test will use the same data as the Acceptance Test, which will be one less issue to consider in maintaining uniformity between the System and Acceptance Tests, and will increase confidence in the test results.

In circumstances in which it is not possible to use live data (either because of risk to the live system [and other applications that rely on the live data] or the confidentiality of data involved or for security reasons), a copy of the live data should be used. The quality, accuracy, and volume of the copied data must be as representative of the live data as possible. Where confidentiality or security considerations result in “sanitization”¹ of the data, great care must be taken to ensure that the altered data adequately support the System Testing data requirements.

Where live data or a copy of the live data is used, it may still be necessary to introduce some handcrafted data (to exercise boundary or error conditions, for example). In creating the handcrafted test data the *Test Analyst* must employ formal design techniques, such as the *Boundary Analysis* or *Equivalence Partition* methods described in Chapter 3, to ensure the data adequately test the boundary conditions for the AUT as well as to provide truly representative values.

7.4. Roles and Responsibilities

System Testing is typically conducted by the Testing Team under the supervision of the Test Team Leader, who is responsible for ensuring that adequate System Testing is performed using appropriate testing techniques and under quality control and supervision (as defined in the System Test Plan and Test Specification documents).

Within the Testing Team, the *Test Analyst* is responsible for designing and implementing (and/or re-using) the Test Script and component Test Cases used in testing the AUT.

The *Tester* is responsible for executing the Test Cases documented within the Test Scripts. In a large Test Team, it is possible for several Test Analysts and Testers to report to the Test Team Leader, whereas in small Test Teams, the Test Analyst and Tester roles may be filled by single members of staff, or even by a single person with joint responsibility for design and execution of tests.

At System Testing, it is essential that the testing process be monitored by an *Independent Test Observer* who will formally witness the results of individual Test Cases. The Independent Test Observer could be drawn from the Quality Assurance Group in an organization where such a group exists or from another testing team or project.

The Test Team Leader will liaise with the *Development Team Leader* to determine the progress of the development of the AUT and likely dates for delivery of the

¹Where live data contain confidential data or information relating to security issues, it may be inappropriate to conduct system testing using the original data. One solution is to use a copy of the live data that has been altered to remove or change items of a confidential or secure nature.

AUT for System Testing purposes. The Test Team Leader will invite the Development Team Leader to informally observe the System Test or to nominate a suitably qualified member of the development team to attend in his or her place.

The Test Team Leader should consider inviting a *User Representative* to informally observe System Testing. System Testing provides a valuable opportunity to expose the User Representative to aspects of the operation and appearance of the AUT, allowing (informal) feedback to be obtained and helping to manage the expectations of the User Representative prior to formal *Acceptance Testing* (also see Chapter 4 for an explanation of the need to involve users in the testing process).

The Test Team Leader must liaise with the *IT Systems Administrator* (that is, the member of staff with responsibility for administering the corporate IT facilities) for the purpose of installing the AUT prior to testing (this will be particularly important if the System Test is scheduled to take place in the live environment). Similarly, the IT Systems Administrator should be kept fully apprised of the date, time, and duration of the System Test to ensure that there is no contention for system resources and that there are no scheduled IT tasks (such as preventative maintenance) and to allow the IT manager the opportunity to perform contingency operations (such as backing up the system) prior to testing.

The Test Team Leader will file copies of a number of outputs produced as a result of the System Testing process (and listed in Section 7.8), including the completed *Test Result Record forms*, System Test Log, and a comprehensive System Test Summary report on completion of System Testing.

7.5 Planning and Resources

The planning of System Testing is the responsibility of the Test Team Leader and should be developed with reference to the overall development and testing plan for the AUT in order to minimize the impact on the development process caused by System Testing and to ensure that no contention for resources will occur.

The human resources required for System Testing will be drawn from the testing team (with the exception of the Independent Test Observer, who may be drawn from another project or from the *QA Group*, should one exist within the organization).

It is assumed that System Testing will take place either in a dedicated *test environment* or in the *live environment*. The choice will depend on a number of issues, including:

- ▲ The availability of a suitably specified test environment (that is, one that is representative of the live environment)
- ▲ An assessment of the commercial or Safety Critical nature of the live environment and the likelihood of the System Test adversely affecting it
- ▲ The occurrence of commercially or confidentially sensitive data or security critical information within the live environment.

In the event that System Testing does take place in the live environment, the Test Team Leader must liaise with the *IT Systems Administrator* to plan installation of

the AUT prior to testing and to ensure that the manager is aware of the date, time, and duration of the System Test.

It is essential that the System Test Plan take account of the time and effort required for any correction of defects and retesting.

7.6 Inputs

The following items are required as inputs to System Testing:

- ▲ The Requirements Specification document (for the AUT)
- ▲ The design documents for the AUT
- ▲ Any supplementary material, such as user guides or prototype software
- ▲ The System Test Plan (see Appendix C)
- ▲ The System Test Specification document (see Appendix D)
- ▲ The System Testing Guide (see Appendix B)
- ▲ The System Test Scripts and Test Cases (from the Test Analyst)
- ▲ Blank *Test Result Record forms* (see Appendix F)
- ▲ Any appropriate *Re-use Packs*, such as the Unit and Integration Test Re-use Packs (see Appendix I).

7.7 Testing Techniques for System Testing

The following testing techniques are appropriate for System Testing:

- ▲ Black Box Testing against high-level system requirements
- ▲ *Thread Testing* against the high-level business requirements for the AUT
- ▲ *Nonfunctional Testing* (such as, Volume, Stress, and Performance testing)
- ▲ *Static Testing* (review of system documentation, such as User Manuals).
- ▲ Consider the role and use of automated testing tools.

See Chapter 3 for details of these testing techniques.

7.8 Outputs

The following items are generated as outputs from System Testing:

- ▲ The fully tested system
- ▲ The completed System Test Certificate (see Appendix H)
- ▲ Any Revised Test Scripts and Test Cases (where appropriate)
- ▲ The archived test data
- ▲ The completed Test Result Record forms (see Appendix F)
- ▲ The System Test Log (see Appendix G)
- ▲ The System Test Re-Use pack (see Appendix I)
- ▲ A System Test Summary Report (see Appendix J).

System Testing will be considered complete when all of the above deliverables are complete and copies (under strict configuration management, as described in Chapter 4) have been provided to the Test Team Leader.

Appropriate deliverables (that is, all of the above except the tested system) should be stored in the project file. The fully tested system (plus any associated test harness or simulation code and test data) should be backed up and archived.

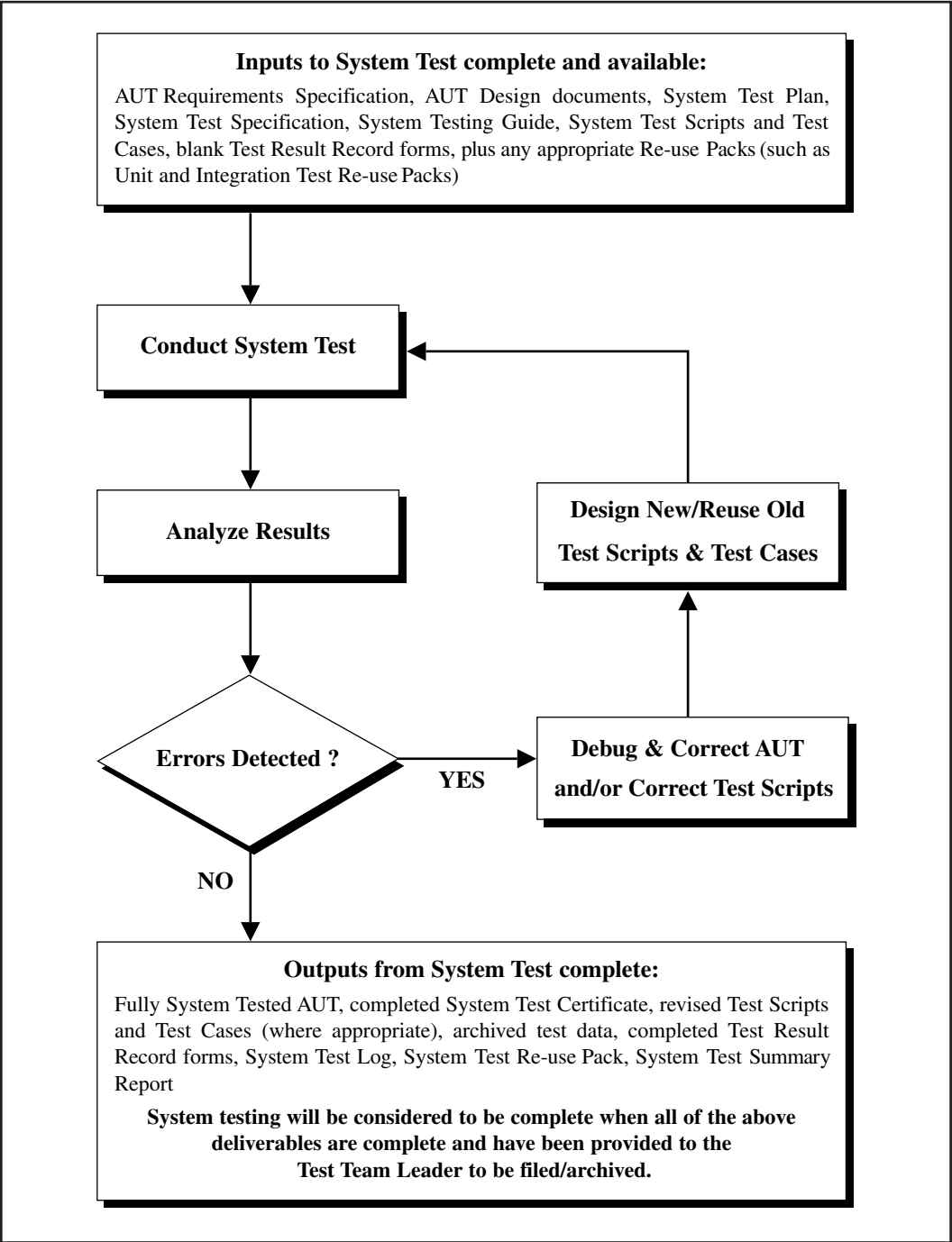


Figure 7.2. High-level overview of the System Testing Process.

Systems Integration Testing

There has never been an unexpectedly short debugging period in the history of computing

Steven Levy, "Hackers"

8.1 Overview

The objective of *Systems Integration Testing* is to provide confidence that the *Application Under Test (AUT)* is able to interoperate successfully with other specified software systems¹ and does not have an adverse affect on other systems that may also be present in the *live environment*, or vice versa. Systems Integration Testing is conducted by the *Testing Team* under the supervision of the *Test Team Leader*. Systems Integration Testing may also be termed *Compatibility Testing*, or simply Integration Testing² (as it is termed in the British Library Testing Process, for example – see Chapter 14).

It is possible that the testing tasks performed during Systems Integration Testing may be combined with *System Testing*, particularly if the AUT has little or no requirement to interoperate with other systems.

Systems Integration Testing should employ *Black Box* techniques and will test the high-level interoperability requirements of the AUT without considering the internal construction of the AUT (for example, testing business processes and complete transactions that require intersystem communication or interaction). The use of *Negative Testing* and *Error Guessing* techniques (see Chapter 3) are particularly appropriate during Compatibility Testing for the purposes of uncovering any unanticipated problems.

In terms of the *V Model*, Systems Integration Testing corresponds to the Specification phase of the software development lifecycle (Figure 8.1).

¹"Software systems" refers both to applications that may be co-resident in memory as well as external systems with which the AUT may need to communicate.

²Where Systems Integration Testing is termed "Integration Testing," the traditional Integration Testing phase will need to have a different title, such as "Link Testing."

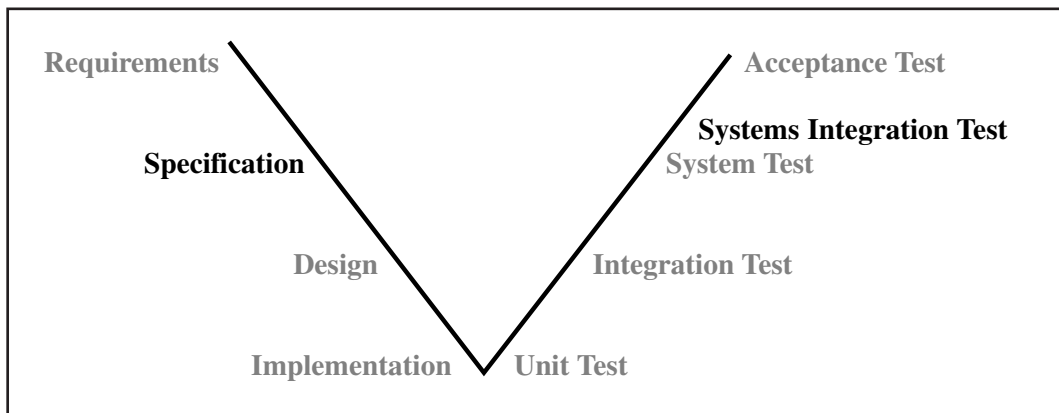


Figure 8.1. Systems Integration Testing and the V Model.

Testing issues associated with Systems Integration Testing (such as test planning and resourcing and the review of testing requirements and high-level test design) should be considered during the Specification phase of system development. The V Model and its role in testing are described in Chapter 4.

A flow chart providing a high-level overview of the Systems Integration Testing process can be found at the end of this chapter (Figure 8.2).

8.2 Systems Integration Test Approach

The following approach should be followed in conducting Systems Integration Testing:

- ▲ Review the interoperability requirements for the system (for example, by reference to the *requirements specification* document for the AUT) and identify:
 - any requirements for the AUT to communicate with other systems and the means of communication
 - the high-level business requirements that involve communication with other systems
 - the requirement for data handling and transaction rates for the live environment
 - the requirements for system performance
 - backup and recovery requirements (where these involve other systems)
 - any security requirements (where these involve interoperability issues)
- ▲ Review the computing environment in which the live system will run in order to identify any interoperability or compatibility issues with other systems
- ▲ Use the above information to generate a set of *Test Scripts* and *Test Cases* to test the system.

Where appropriate, re-using test cases from *Unit*, *Integration*, and *System* testing should be considered. These can be obtained from the *Unit*, *Integration*, and *System Test Re-use Packs* (see Appendix I).

Where the AUT will interoperate with external systems over which the *Test Team* has no control, the use of a *test harness* should be considered to provide reproducible testing results.

8.3 Systems Integration Test Data Requirements

Systems Integration Testing should take place in the live environment, and consequently will utilize *live data*. Without testing in the live environment, it will be extremely difficult to have confidence in the results of the Systems Integration Tests (in particular, the compatibility testing).

Typically, the requirements for data within Systems Integration Testing are not particularly demanding. This is because the focus of testing is on interoperability and compatibility testing and not on the ability of the system to access and process large volumes of data, a test requirement that should have been dealt with during System Testing.

The guiding principle in identifying the data requirements for Systems Integration Testing is to consider what data will be needed in supporting the *Thread Testing* that will be needed in verifying interoperability – such as passing, requesting or receiving data from other systems. In a large number of cases it will be possible to re-use or modify System Tests and their associated data during Systems Integration Testing.

8.4 Roles and Responsibilities

Systems Integration Testing is typically conducted by the Testing Team under the supervision of the Test Team Leader, who is responsible for ensuring that adequate testing is performed using appropriate testing techniques and under quality control and supervision (as defined in the Systems Integration Test Plan and Test Specification documents).

Within the Test Team, the *Test Analyst* is responsible for designing and implementing (and/or re-using) the Test Script and component Test Cases used in testing the AUT.

The *Tester* is responsible for executing the Test Cases documented within the Test Scripts. In a large Test Team, it is possible for several Test Analysts and Testers to report to the Test Team Leader, whereas on small Test Teams the Test Analyst and Tester roles may be filled by single members of staff, or even by a single person with joint responsibility for design and execution of tests.

At Systems Integration Testing it is essential that the testing process be monitored by an *Independent Test Observer*, who will formally witness the results of individual Test Cases. The Independent Test Observer could be drawn from the *Quality Assurance (QA) Group* in an organization where such a group exists or from another testing team or project.

The Test Team Leader will liaise with the *Development Team Leader* to determine the progress of the development of the AUT and likely dates for delivery of the AUT for Systems Integration Testing purposes. In the event that Systems Integration Testing takes place as part of System Testing or immediately after, then the AUT should already be available for testing. The Test Team Leader will invite the Development Team Leader to informally observe the Systems Integration Test or to nominate a suitably qualified member of the development team to attend in his or her place.

The Test Team Leader should consider inviting a *User Representative* to informally observe Systems Integration Testing. Systems Integration Testing provides another valuable opportunity to expose the User Representative to aspects of the operation and appearance of the AUT, allowing (informal) feedback to be obtained and helping to manage the expectations of the User Representative prior to formal *Acceptance Testing* (see Chapter 4 for a fuller explanation of the need to involve users in the testing process).

The Test Team Leader must liaise with the *IT Systems Administrator* (that is, the member of staff with responsibility for administering the corporate IT facilities) for the purpose of installing the AUT prior to testing (this will be particularly important if the Systems Integration Test is scheduled to take place in the live environment and if it is planned to make use of networked/intra/internet access to other systems or data). Similarly, the IT Systems Administrator should be kept fully apprised of the date, time, and duration of the Systems Integration Test to ensure that there is no contention for system resources and there are no scheduled IT tasks (such as preventative maintenance) and to allow the IT manager the opportunity to perform contingency operations (such as backing up the system) prior to testing.

The Test Team Leader will file copies of a number of outputs produced as a result of the Systems Integration Testing process (and listed in Section 8.8), including the completed Test Result Record forms, a Systems Integration Test Log, and a comprehensive Systems Integration Test Summary report on completion of Systems Integration Testing.

8.5 Planning and Resources

The planning of Systems Integration Testing is the responsibility of the Test Team Leader and should be developed with reference to the overall development and testing plan for the AUT in order to minimize the impact on the development process caused by Systems Integration Testing and to ensure that no contention for resources will occur.

The human resources required for Systems Integration Testing will be drawn from the Testing Team (with the exception of the Independent Test Observer, who may be drawn from another project or from the QA Group, should one exist within the organization).

It is assumed that Systems Integration Testing will take place in the live environment. The use of a dedicated *test environment* should be considered only after weighing the following issues:

- ▲ The availability of a suitably specified test environment (that is, one that is representative of the live environment)
- ▲ An assessment of the commercial or Safety Critical nature of the live environment and the likelihood of the System Test adversely affecting it
- ▲ The occurrence of commercially or confidentially sensitive data or security critical information within the live environment
- ▲ The ability to simulate representative interactions with the external system.

In the event that Systems Integration Testing does take place on the live environment, the Test Team Leader must liaise with the IT Systems Administrator to plan installation of the AUT prior to testing and to ensure that the manager is aware of the date, time, and duration of the Systems Integration Test.

It is essential that the System Test Plan take account of the time and effort required for any correction of defects and retesting.

8.6 Inputs

The following items are required as inputs to Systems Integration Testing:

- ▲ The Requirements Specification document (for the AUT)
- ▲ The design documents for the AUT
- ▲ Any supplementary material, such as user guides or prototype software
- ▲ The Systems Integration Test Plan (see Appendix C)
- ▲ The Systems Integration Test Specification document (see Appendix D)
- ▲ The Systems Integration Testing Guide (see Appendix B)
- ▲ Any Systems Integration Test Scripts and Test Cases (from the Test Analyst)
- ▲ Blank *Test Result Record Forms* (see Appendix F)
- ▲ Any appropriate *Re-use Packs* such as the Unit, Integration, and System test Re-use Packs (see Appendix I).

8.7 Testing Techniques for Systems Integration Testing

The following testing techniques are appropriate for Systems Integration Testing:

- ▲ Black Box Testing against high-level system requirements (specifically, requirements addressing interoperability issues)

- ▲ Thread Testing against the high-level business requirements for the AUT (again, focusing on interoperability requirements)
- ▲ *Negative Testing* and *Error Guessing* to uncover any unanticipated problems (such as contention for system resources).
- ▲ Consider the role and use of automated testing tools.

See Chapter 3 for full details of these testing techniques.

8.8 Outputs

The following items are generated as outputs from Systems Integration Testing:

- ▲ The Systems Integration Tested system
- ▲ The completed Systems Integration Test Certificate (see Appendix H)
- ▲ Any Revised Test Scripts and Test Cases (where appropriate)
- ▲ The archived test data
- ▲ The completed Test Result Record forms (see Appendix F)
- ▲ Systems Integration Test Log (see Appendix G)
- ▲ Systems Integration Test Re-use pack (see Appendix I)
- ▲ A Systems Integration Test Summary Report (see Appendix J).

Systems Integration Testing will be considered complete when all of the above deliverables are complete and copies (under strict configuration management, as described in Chapter 4) have been provided to the Test Team Leader.

Appropriate deliverables (that is, all of the above except the tested system) should be stored in the project file. The fully tested system (plus any associated test harness or simulation code and test data) should be backed up and archived.

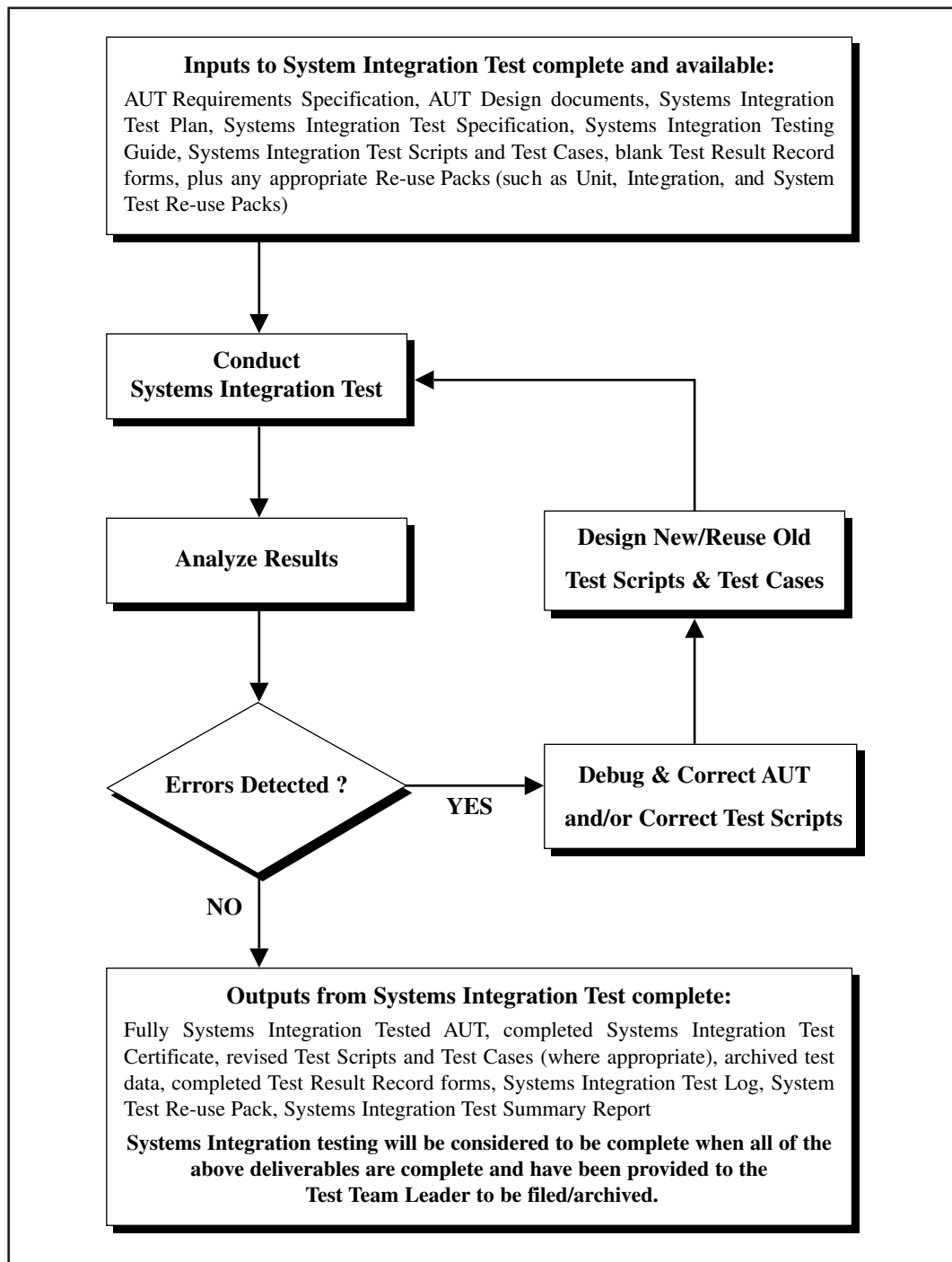


Figure 8.2. High-level overview of the User Acceptance Testing Process.

User Acceptance Testing

If love is like an extended software Q.A. suite, then true love is like a final Acceptance Test – one often has to be willing to endure compromise, bug fixes and work-arounds; otherwise, the software is never done

The Usenet Oracle

9.1 Overview

The objective of *User Acceptance Testing* is to confirm that the *Application Under Test (AUT)* meets its business requirements and to provide confidence that the system works correctly and is usable before it is formally “delivered” to the *end user*. User Acceptance Testing is conducted by one or more *User Representatives* with the assistance of the *Test Team*.

User Acceptance Testing is considered distinct from *Operations Acceptance Testing*, which is used to verify the operations and administrative aspects of the AUT (such as the installation of updates to the AUT, backup, archive and restoring the AUT and its data, and the registration of new users and the assigning of their privileges, for example).¹ In practice, where the AUT supports simple administrative facilities, User and Operations Acceptance Testing are often combined into a single testing exercise. Also see Chapter 10, Operations Acceptance Testing.

User Acceptance Testing should employ a *Black Box* approach to testing and should make use of *Thread Testing* techniques to verify the high-level business requirements of the system. In practice, the User Representative will test the AUT by performing typical tasks that they would perform during their normal usage of the system.

User Acceptance Testing should also address the testing of the system documentation (such as user guides) by the User Representative.

In terms of the *V Model*, User Acceptance Testing corresponds to the Requirements phase of the software development lifecycle (Figure 9.1).

¹Typically, the operations users will be experienced IT staff and will require less guidance and assistance in the testing process than the business users of the application under test.

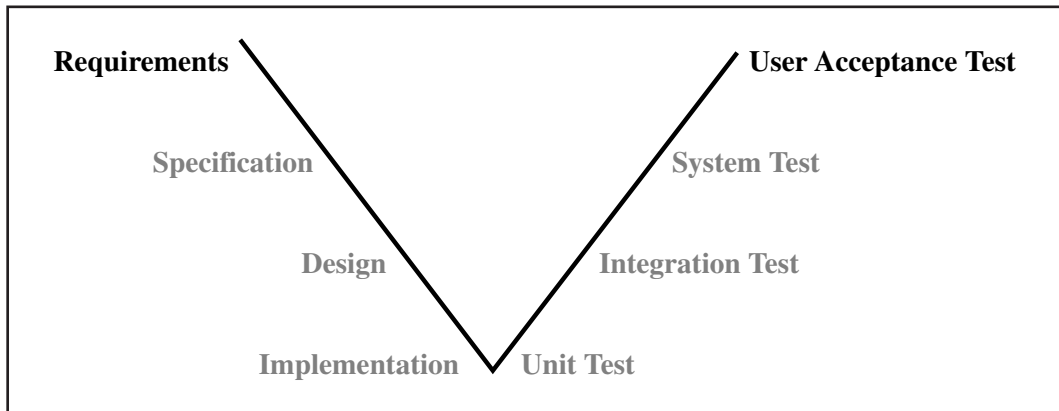


Figure 9.1. User Acceptance Testing and the V Model.

User Acceptance Testing issues (such as test planning, review of testing requirements, and identification of the User Representative) must be considered during the Requirements phase of the AUT. The V Model and its role in testing are described in Chapter 4.

A flow chart providing a high-level overview of the User Acceptance Testing Process can be found at the end of this chapter (Figure 9.2).

9.2 User Acceptance Test Approach

Typically, User Acceptance Testing involves the *Test Analyst* selecting a representative subset of the *System Test Scripts* and asking the User Representative to execute them. The selection of these *Test Cases* can be based upon:

- ▲ Discussion with the User Representative
- ▲ Review of the requirements for the AUT to identify any particularly important areas or functions that should be verified.

It is also likely that the Test Analyst will implement a number of additional Test Scripts to verify specific user-orientated aspects of the AUT, such as:

- ▲ Usability aspects of the AUT
- ▲ System documentation (such as the User Manual)
- ▲ User Help facilities (both document based and on-line).

These additional scripts should be produced using formal design techniques as described in Chapter 3.

If the System Test for the AUT did not take place in the *live environment* and/or without the *live data*, the following steps must be considered:

- ▲ Rerun those System Tests that address requirements for the AUT to communicate/exchange data with other systems (to validate user privileges, for example).

- ▲ Rerun those System Tests addressing the *interoperability/compatibility* requirements for the AUT.

It is very important for the success of the Acceptance Test that the User Representatives' expectations have been carefully managed throughout the development and testing process. For example, the *Test Team Leader* should have invited the User Representative to attend the earlier testing phases in an informal capacity so that he or she will be used to the look and feel of the AUT. In this way, the risk of the User Representative rejecting the AUT at Acceptance Testing because it does not conform to the appearance that was anticipated will be minimized. Chapter 4 discusses the need to involve the User Representative as early as possible in the development and testing process.

9.3 User Acceptance Test Data Requirements

Wherever possible, live data should be used during User Acceptance Testing.

Where the live data contains confidential or security information and this data will be observable during part or all of the Acceptance Test, steps must be taken to ensure that:

- ▲ The User Representative is cleared to work with such data.
- ▲ The Test Team Leader is cleared to work with such data, or the testing is organized to avoid the need for the Test Team Leader to see the data.
- ▲ The *Independent Test Observer* is cleared to work with such data, or is able to confirm and document the success or failure of the Test Cases without having to see the data.

In the event that it is not possible to use live data, Acceptance Testing should be conducted against a realistic and representative copy of the live data. Where security or confidential information needs to be altered or removed from the test data, care should be taken to ensure that “sanitizing” the data will not have undesirable effects on the accuracy of the User Acceptance Test.

Where it is necessary to introduce handcrafted data for the purposes of exercising boundary behavior or error conditions, these data should be produced using formal design techniques (see Chapter 3).

9.4 Roles and Responsibilities

User Acceptance Testing is typically conducted by the User Representative with the assistance of the Test Team. User Acceptance Testing is supervised by the Test Team Leader, who is responsible for ensuring that adequate testing is performed using appropriate testing techniques and under quality control and supervision (as defined in the User Acceptance Test Plan and Test Specifications documents).

The User Acceptance Test typically employs a subset of the tests run during the System Test for the AUT. The Test Analyst is responsible for liaising with the User Representative to determine which Test Scripts and Test Cases will be rerun during the Acceptance Test, as well as to determine the need to design and implement any additional Test Scripts to exercise specific aspects of the AUT that the User Representative may request. Any such additional effort must be discussed with and agreed upon by the Test Team Leader.

The *Tester* assists the User Representative in executing the Test Scripts comprising the User Acceptance Test and in interpreting the results of Test Cases in collaboration with the Independent Test Observer. The Tester is also responsible for the set-up and initialization of the *Test Bed* prior to testing. In a large Test Team, it is possible for several Test Analysts and Testers to report to the Test Team Leader, whereas on small Test Teams the Test Analyst and Tester roles may be filled by single members of staff, or even by a single person with joint responsibility for design and execution of tests.

It is particularly important in User Acceptance Testing that the testing process be monitored by an Independent Test Observer, who is involved in formally witnessing the result of executing the individual Test Cases. The Independent Test Observer can act as a safeguard against over-enthusiastic Testers attempting to coerce or persuade the User Representative (who, while knowledgeable about his or her domain of expertise, may be *information technology [IT]* naive) into accepting the results of a test with which they may have a genuine concern. The Independent Test Observer could be drawn from the *Quality Assurance (QA) Group* in an organization in which such a group exists or from another testing team or project.

The Test Team Leader will liaise with the Development Team Leader to determine the progress of the development of the AUT and likely dates for delivery of the AUT for User Acceptance Testing purposes. This is particularly important if the AUT required rework as a result of the System or Systems Integration Testing.

The Test Team Leader must liaise with the *IT Systems Administrator* (that is, the staff member with responsibility for administering the corporate IT facilities) for the purpose of installing the AUT in the live environment. Similarly, the IT Systems Administrator should be kept fully apprised of the date, time, and duration of the User Acceptance Test to ensure that there is no contention for system resources and there are no scheduled IT tasks (such as preventive maintenance) and to allow the IT manager the opportunity to perform contingency operations (such as backing up the system) prior to testing.

The Test Team Leader files copies of a number of outputs produced as a result of the User Acceptance Test (and listed in Section 9.8), including the completed Test Result Record forms, an Acceptance Test Log, and a comprehensive Acceptance Test Summary report on completion of User Acceptance Testing.

9.5 Planning and Resources

The planning of User Acceptance Testing is the responsibility of the Test Team Leader and should be developed with reference to the overall development and testing plan for the AUT in order to minimize the impact on the development

process caused by Acceptance Testing and to ensure that no contention for resources will occur.

The human resources required for User Acceptance Testing will be drawn from the Test Team (with the exception of the Independent Test Observer, who may be drawn from another project or from the *QA Group*, should one exist within the organization) and the User Representative.

The Test Manager will liaise with the Customer² to identify one or more User Representatives who will be selected from the user community for the AUT. It is very important that the User Representatives be genuine users of the system and not, as often happens, their line managers. For effective Acceptance Testing it is key that the User Representatives have a thorough understanding of the business aspects of the system and experience with the typical scenarios employed to achieve their business goals.

It is assumed that User Acceptance Testing will take place in the live environment. However, under certain circumstances Acceptance Testing may have to be performed using a dedicated *test environment*. The choice will depend on a number of issues, including:

- ▲ An assessment of the commercial or Safety Critical nature of the live environment and the likelihood of the User Acceptance Test adversely affecting it
- ▲ The occurrence of commercially or confidentially sensitive data or security critical information within the live environment
- ▲ The availability of a suitably specified test environment (that is, one that is fully representative of the live environment)
- ▲ The success of the System Test, the possible detection of a significant number of defects, and large-scale re-work of the AUT.

Assuming that the User Acceptance Test takes place in the live environment, the Test Team Leader must liaise with the IT Systems Administrator to plan installation of the AUT prior to testing and to ensure that the Administrator is aware of the date, time, and duration of the System Test.

It is essential that the User Acceptance Test Plan take account of the time and effort required for any correction of defects and retesting.

9.6 Inputs

The following items are required as inputs to User Acceptance:

- ▲ The Requirements Specification document for the AUT
- ▲ The design documents for the AUT
- ▲ Any supplementary material, such as user guides or prototype software

²The “Customer” could be an external client for whom the development of the AUT is being conducted under contract, or could be another member of staff within the same organization.

- ▲ The User Acceptance Test Plan (see Appendix C)
- ▲ The User Acceptance Test Specification document (see Appendix D)
- ▲ The User Acceptance Testing Guide (see Appendix B)
- ▲ The User Acceptance Test Scripts and Test Cases (from the Test Analyst)
- ▲ Blank *Test Result Record Forms* (see Appendix F)
- ▲ Any appropriate *Re-use Packs*, such as the System Test Re-use Pack (see Appendix I).

9.7 Testing Techniques for User Acceptance Testing

The following testing techniques are appropriate for User Acceptance Testing:

- ▲ Black Box Testing against high-level system requirements
- ▲ Thread testing against the high-level business (that is, user) requirements for the AUT
- ▲ *Usability Testing* to ensure the GUI for the AUT is intuitive, consistent, and easy to use and to ensure the on-line Help facilities are satisfactory
- ▲ *Static Testing* of system documentation (such as User Manuals) and of their use with the AUT.

See Chapter 3 for details of these testing techniques.

9.8 Outputs

The following items are generated as outputs from User Acceptance Testing:

- ▲ The user accepted system
- ▲ The completed User Acceptance Test Certificate (see Appendix H)
- ▲ Any revised Test Scripts and Test Cases (where appropriate)
- ▲ The archived test data
- ▲ The completed Test Result Record forms (see Appendix F)
- ▲ The User Acceptance Test Log (see Appendix G)
- ▲ The User Acceptance Test Re-use Pack (see Appendix I)
- ▲ A User Acceptance Test Summary Report (see Appendix J).

User Acceptance Testing will be considered complete when all of the above deliverables are complete and copies (under strict configuration management, as described in Chapter 4) have been provided to the Test Team Leader.

Appropriate deliverables (that is, all of the above except the tested system) should be stored in the project file. The fully tested system (plus any associated test harness or simulation code and test data) should be backed up and archived.

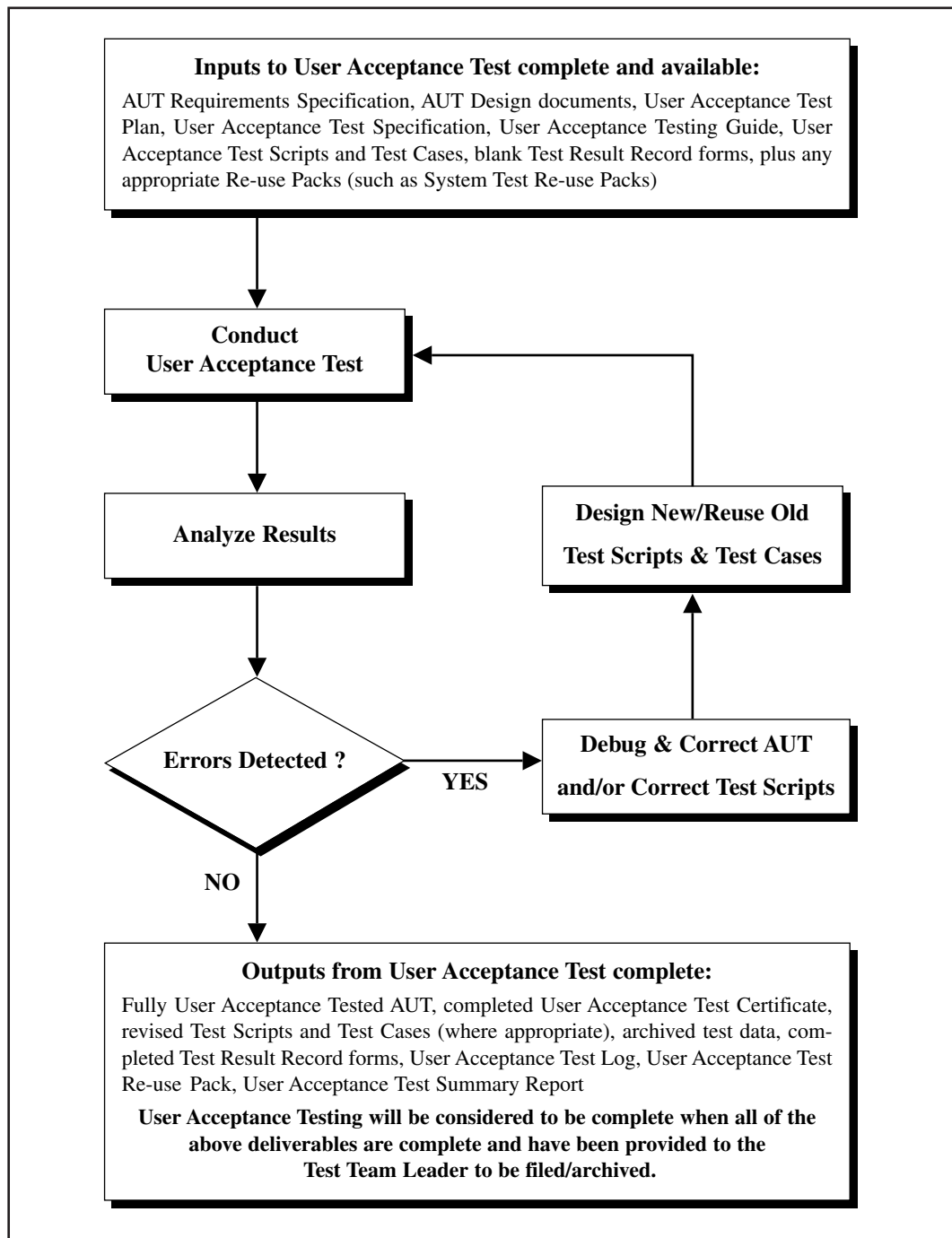


Figure 9.2. High-level overview of the User Acceptance Testing Process.

Operations Acceptance Testing

If a man will begin with certainties, he shall end in doubts; but if he will be content to begin with doubts, he shall end in certainties

Francis Bacon

10.1 Overview

The objective of *Operations Acceptance Testing* is to confirm that the *Application Under Test* (AUT) meets its operations requirements, and to provide confidence that the system works correctly and is usable before it is formally “handed over” to the *operations user*. Operations Acceptance Testing is conducted by one or more *Operations Representatives* with the assistance of the *Test Team*.

Operations Acceptance Testing is considered distinct from *User Acceptance Testing*, which is used to verify that the AUT meets its business requirements, and to provide confidence that the system works correctly and is usable before it is formally “delivered” to the *end user*. In practice, where the AUT supports simple administrative facilities, Operations and User Acceptance Testing are often combined into a single testing exercise. Also see Chapter 9, User Acceptance Testing.

Operations Acceptance Testing should employ a *Black Box* approach to testing, and should make use of *Thread Testing* techniques to verify the high-level operations requirements of the system. In practice, the Operations Representatives will test the AUT by performing typical tasks that they would perform during their normal usage of the system.

Operations Acceptance Testing should also address the testing of the system documentation (such as operations guides) by the Operations Representative.

In terms of the *V Model*, Operations Acceptance Testing corresponds to the Requirements phase of the software development lifecycle (Figure 10.1).

A flow chart providing a high-level overview of the Operations Acceptance Testing process can be found at the end of this chapter (Figure 10.2).

Operations Acceptance Testing issues (such as test planning, review of testing requirements, and identification of the Operations Representative) must be considered during the Requirements phase of the development of the AUT. The V Model and its role in testing are documented in Chapter 4.

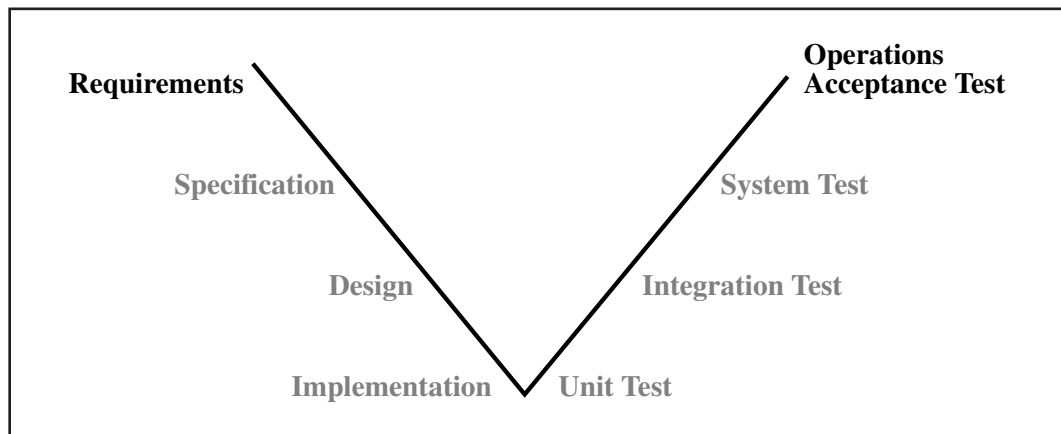


Figure 10.1. Operations Acceptance and the V Model.

10.2 Operations Acceptance Test Approach

For applications that support significant operations facilities, the operations aspects of the AUT are typically key to the successful running of the system. Under such circumstances, a high degree of confidence in the operations aspects of the system (and, hence, greater testing coverage) is required; the additional time, effort, and resources required for Operations Acceptance Testing must be reflected in the test plan.

Typically, Operations Acceptance Testing involves the *Test Analyst* selecting a representative subset of the *System Test Scripts* and asking the Operations Representative to execute them. The selection of these *Test Cases* can be based upon:

- ▲ Discussion with the Operations Representative to identify key aspects of the operations functionality
- ▲ Review of requirements for the system to identify any particularly important areas or functions.

It is also likely that the Test Analyst will implement a number of additional Test Scripts to verify specific operations-oriented aspects of the AUT, such as:

- ▲ The usability of operations aspects of the AUT
- ▲ System procedures (such as the installation procedure)
- ▲ System documentation (such as the Operations Manual)
- ▲ Operations Help facilities (both document based and on-line).

These additional scripts should be produced using formal design techniques as described in Chapter 3.

If the System Test for the AUT did not take place in the *live environment* and/or without the *live data*, the following steps must be considered:

- ▲ Rerun the System Tests addressing requirements for the AUT to communicate/exchange data with other systems (to validate administrator privileges, for example).

- ▲ Rerun the System Tests addressing the *interoperability/compatibility* requirements for the AUT.

The *Test Team Leader* should consider the benefits of involving the Operations Representatives in the early testing phases in an informal capacity in order to manage their expectations of the software they will ultimately be expected to accept and operate. This approach is likely to be less critical to the success of Operations Acceptance compared with User Acceptance because Operations Representatives are typically more *information technology (IT)* oriented than *User Representatives*; hence, they are likely to have more realistic expectations regarding the capabilities of the AUT and to accept compromises in terms of the facilities provided. Chapter 4 discusses the need to involve the User and Operations Representatives as early as possible in the development and testing process.

10.3 Operations Acceptance Test Data Requirements

Although it is preferable to use live data during Operations Acceptance Testing, it is not as critical as it is for User Acceptance Testing. In practice, it is possible to use a copy of the live data or even a handcrafted data set for the purposes of Operations Acceptance as long as:

- ▲ The available test data are realistic and representative of the live data.
- ▲ The test data set provides a sufficient quantity and complexity of data to support those Operations Acceptance Testing tasks involving manipulation of large volumes of data (such as the backup and recovery of system data or installation of the system database).
- ▲ The Operations Acceptance Test does not involve significant data access or manipulation.

Where the live data contains confidential or security information and this data will be observable during part or all of the acceptance test, steps must be taken to ensure that:

- ▲ The Operations Representative is cleared to work with such data (this will be very likely if, for example, the Operations Representative is the System Administrator, who is likely to have sufficient clearance).
- ▲ The Test Team Leader is cleared to work with such data, or the testing is organized to avoid the need for the Test Team Leader to see the data.
- ▲ The Independent Test Observer is cleared to work with such data, or is able to confirm and document the success or failure of the Test Cases without having to see the data.

Where it is necessary to introduce handcrafted data for the purposes of exercising boundary behavior or error conditions, this data should be produced using formal design techniques (see Chapter 3).

10.4 Roles and Responsibilities

Operations Acceptance Testing is typically conducted by the Operations Representative with the assistance of the Test Team. Operations Acceptance Testing is supervised by the Test Team Leader, who is responsible for ensuring that adequate testing is performed using appropriate testing techniques and under quality control and supervision (as defined in the Operations Acceptance Test Plan and Test Specification documents).

The Operations Acceptance Test typically employs a subset of the tests run during the System Test for the AUT (however, if the operations aspects of the AUT are considered vital to the successful functioning of the software, it may be decided to execute all of the System Tests that deal with operations facilities). The Test Analyst is responsible for liaising with the Operations Representative to determine which Test Scripts and Test Cases will be rerun during the Operations Acceptance Test.

The *Tester* assists the Operations Representative in executing the Test Scripts comprising the Operations Acceptance Test and also assists in the interpretation of the results of Test Cases in collaboration with the Independent Test Observer. The Tester is also responsible for the set-up and initialization of the *Test Bed* prior to testing.

Although it is important that the testing process be monitored by an Independent Test Observer, who is involved in formally witnessing the individual Test Cases, this requirement is not as strenuous as it is for User Acceptance Testing. This is because the Operations Representatives are typically more IT aware than the User Representatives and will require less support against over-enthusiastic Testers attempting to coerce or persuade the Operations Representative into accepting the results of a test with which they may have expressed a genuine concern. The Independent Test Observer could be drawn from the *Quality Assurance (QA) Group* in an organization where such a group exists or from another testing team or project.

The Test Team Leader will liaise with the Development Team Leader to determine the progress of the development of the AUT and likely dates for delivery of the AUT for Operations Acceptance Testing purposes. This is particularly important if the AUT required rework as a result of the System or Systems Integration Testing.

The Test Team Leader must liaise with the *IT Systems Administrator* (that is, the member of staff with responsibility for administering the corporate IT facilities) for the purpose of installing the AUT in the live environment. Similarly, the IT Systems Administrator should be kept fully apprised of the date, time, and duration of the Operations Acceptance Test to ensure that there is no contention for system resources and there are no scheduled IT tasks (such as preventive maintenance) and to allow the IT manager the opportunity to perform contingency operations (such as backing up the system) prior to testing.

The Test Team Leader files copies of a number of outputs produced as a result of the Operations Acceptance Test (and listed in Section 10.8), including the completed *Test Result Record Forms*, an Acceptance Test Log, and a comprehensive Acceptance Test Summary report on completion of Operations Acceptance Testing.

10.5 Planning and Resources

The planning of Operations Acceptance Testing is the responsibility of the Test Team Leader and should be developed with reference to the overall development and testing plan for the AUT in order to minimize the impact on the development process caused by Operations Acceptance Testing and to ensure that no contention for resources will occur.

The human resources required for Operations Acceptance Testing will be drawn from the Test Team (with the exception of the Independent Test Observer, who may be drawn from another project or from the QA Group, should one exist within the organization) and the Operations Representatives.

The Test Manager will liaise with the Customer¹ to identify one or more Operations Representatives who will be selected from the operations user community (typically, the IT Systems Group) for the AUT.

Since Operations Acceptance Testing does not deal with the execution of typical business scenarios dealing with the normal use of the AUT by the User Representatives, the requirement for using live data may be relaxed. However, typically, the volume and complexity of the test data should be representative of the live data for the purposes of accurately exercising the administrative scenarios typically followed by the operators (such as backup and recovery of the system and archival of the data). In making such a decision regarding the use of live data the following criteria should be considered:

- ▲ The need for realistic volumes of data to accurately exercise the administrative operations of the AUT
- ▲ An assessment of the commercial or Safety Critical risk involved in using the live environment and the likelihood of the Operations Acceptance Test adversely affecting it
- ▲ The occurrence of commercially or confidentially sensitive data or Security Critical information within the live environment (this is not normally an issue, since the operations staff members are likely to be cleared to work with such material)
- ▲ The availability of a suitably specified test environment (that is, one that is sufficiently representative of the live environment).

If the Operations Acceptance Test takes place in the live environment, the Test Team Leader must liaise with the IT Systems Administrator to plan installation of the AUT prior to testing and to ensure that the Administrator is aware of the date, time, and duration of the System Test.

It is essential that the Operations Acceptance Test Plan take account of the time and effort required for any correction of defects and retesting.

¹The "Customer" could be an external client for whom the development of the AUT is being conducted under contract, or could be another member of staff within the same organization.

10.6 Inputs

The following items are required as inputs to Operations Acceptance Testing:

- ▲ The Requirements Specification document
- ▲ The design documents for the AUT, specifically those dealing with the operations aspects of the software
- ▲ Any supplementary material, such as user guides or prototype software
- ▲ The Operations Acceptance Testing Plan (see Appendix C)
- ▲ The Operations Acceptance Test Specification document (see Appendix D)
- ▲ The Operations Acceptance Testing Guide (see Appendix B)
- ▲ The Operations Acceptance Test Scripts and Test Cases (from the Test Analyst)
- ▲ Blank Test Result Record forms (see Appendix F)
- ▲ Any appropriate *Re-use Packs*, such as the System Test Re-use Pack (see Appendix I).

10.7 Testing Techniques for Operations Acceptance Testing

The following testing techniques are appropriate for operations acceptance testing:

- ▲ Black Box Testing against high-level system requirements
- ▲ Thread Testing against the operations requirements for the AUT
- ▲ *Usability Testing* to ensure the GUI for the AUT is intuitive, consistent, and easy to use and to ensure the on-line Help facilities are satisfactory
- ▲ *Static Testing* of system documentation (such as Operations Manuals) and of their use with the AUT.

See Chapter 3 for details of these testing techniques.

10.8 Outputs

The following items are generated as outputs from Operations Acceptance Testing (see Figure 10.2):

- ▲ The fully Operations Acceptance tested system
- ▲ The completed Operations Acceptance Test Certificate (see Appendix H)
- ▲ Any revised Test Scripts and Test Cases (where appropriate)
- ▲ The archived test data
- ▲ The completed Test Result Record forms (see Appendix F)
- ▲ The Operations Acceptance Test Log (see Appendix G)
- ▲ The Operations Acceptance Test Re-use Pack (see Appendix I)
- ▲ An Operations Acceptance Test Summary Report (see Appendix J).

Operations Acceptance Testing will be considered complete when all of the above deliverables are complete and copies (under strict configuration management, as described in Chapter 4) have been provided to the Test Team Leader.

Appropriate deliverables (that is, all of the above except the tested system) should be stored in the project file. The fully tested system (plus any associated test harness or simulation code and test data) should be backed up and archived.

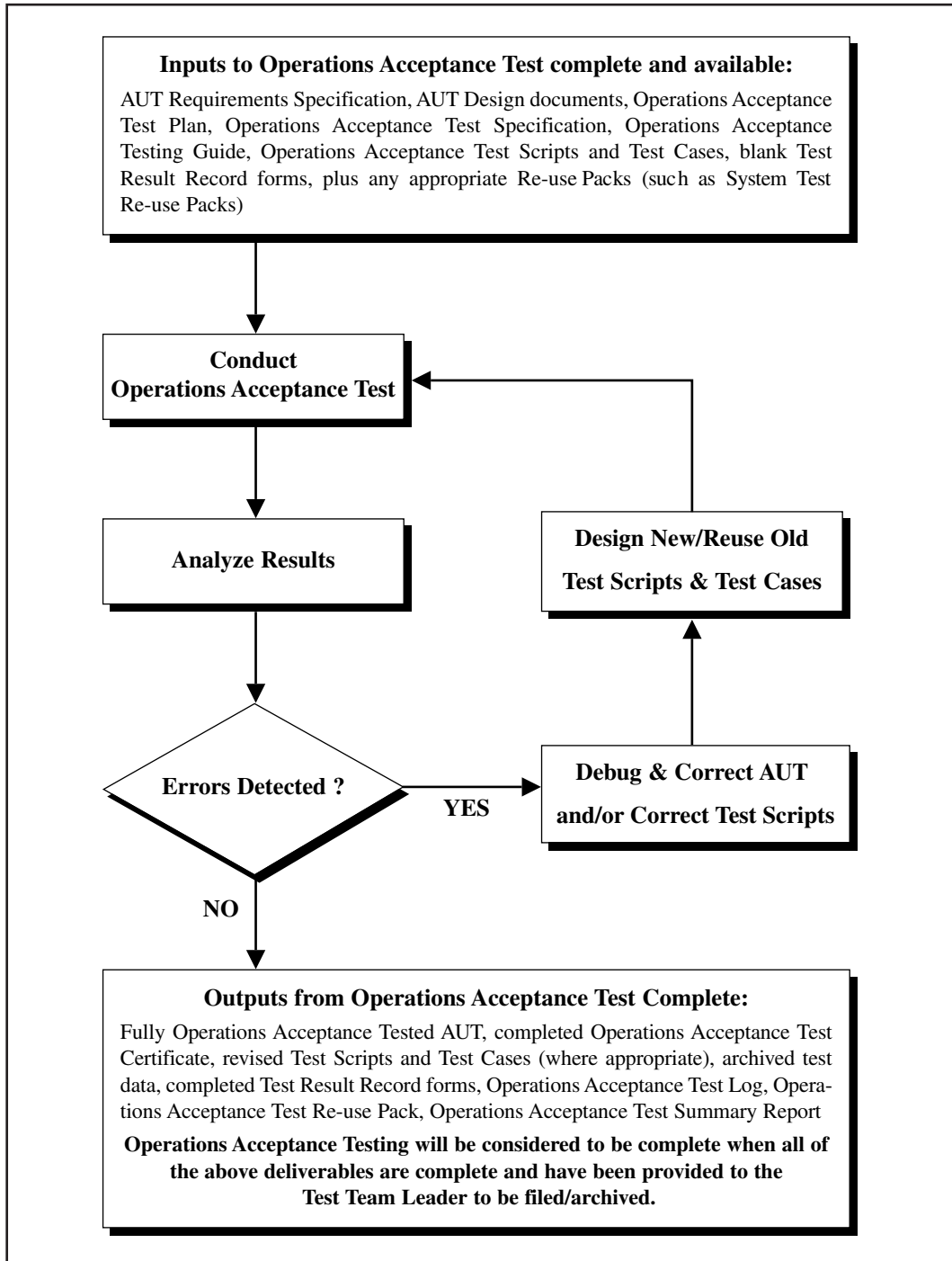


Figure 10.2. High-level overview of the Operations Acceptance Testing Process.

Regression Testing

Quality comes for free as long as you pay for it.

11.1 Overview

The purpose of *Regression Testing* is to ensure that the *Application Under Test (AUT)* still functions correctly following modification or extension of the system (such as user enhancements or upgrades).¹ Typically, the modifications or extensions will be tested to ensure that they meet their requirements, after which a series of tests will be run to confirm that the existing functionality still meets its requirements.

Regression Testing is not strictly speaking a testing phase, but is a testing technique that can be used with any of the testing phases described in the previous chapters. For example, if a *Test Team* were conducting a *System Test* of a new release of the AUT, the *Test Analyst* would need to design a number of tests to verify the correctness of the new or modified functionality. In addition to the new tests, the Test Analyst would select (perhaps from the appropriate *Re-use Pack* – see Appendix I) a number of the tests from the previous System Test of the AUT in order to determine if any defects had been introduced into the AUT during the implementation of the new functionality.

Regression Testing can also be employed following changes to the environment in which an AUT runs, such as the installation of a new version of the operating system, changes to the hardware platform (such as addition of memory), or a particular event external to the AUT. For example, *Millennium Testing* can be considered to have been a special case of Regression Testing. During a typical Millennium Test, the AUT would not have been altered, but it would still be necessary to ensure the AUT performed correctly following a particular event (in this case the year 2000 date change).

Typically, Regression Testing is employed in the context of System and *Acceptance Testing* (and where the AUT has a significant requirement to interoperate with other applications, at *Systems Integration Testing*), ensuring that the complete AUT still functions correctly following a new build or release or ensuring that the existing business functionality of the AUT is intact.

Regression Testing should employ *Black Box* techniques to test the high-level requirements of the AUT without considering the implementation details of the sys-

¹The literal meaning of Regression Testing is testing to ensure that the AUT has not regressed or lost functionality or reliability following the addition of new features or changes to existing functionality.

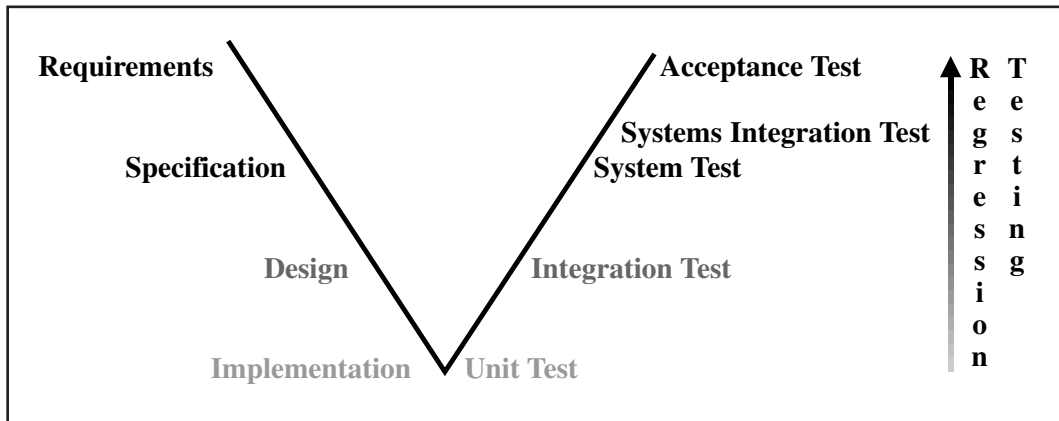


Figure 11.1. Regression Testing and the V Model.

tem (for example, testing business processes and complete transactions). Some *Nonfunctional Testing* may also be required to check whether the enhancements or extensions have affected the performance characteristics of the system. The role of *Error Guessing* is an important one in Regression Testing, with the Test Analyst using his or her testing experience to determine the scope and extent of the testing required to provide good confidence in the modified system (see Chapter 3).

Regression Testing is a particularly appropriate candidate for support by means of an *automated testing tool* (see Reference 22, for example), especially if frequent new builds and releases of the AUT are anticipated. The advantages of using a tool include:

- ▲ The ability to run the Regression Test “unattended,” that is, overnight or over the weekend, without the need for the testing staff to be present
- ▲ The ability to rerun all of the *automated test scripts* or a selected subset of scripts against the new build or release as required to provide maximum confidence in the software
- ▲ Establishing a high degree of confidence that the modifications or extensions to the system have not impacted unduly on its functionality.

A number of the case studies in Part 2 describe organizations that have successfully incorporated the use of automated testing tools in their testing process for use in Regression Testing (see Chapters 16 and 18).

As mentioned earlier in this chapter, although Regression Testing can be employed at all testing phases, it is particularly appropriate during the higher testing phases. Figure 11.1 shows the relationship between Regression Testing and the *V Model*.

11.2 Regression Test Approach

The following approach should be followed in conducting Regression Testing:

- ▲ Complete the functional testing of the new enhancements or extensions to the AUT to verify that they meet their requirements as appropriate to the

particular testing phase (such as System, Systems Integration, or Acceptance Testing).

- ▲ Obtain the testing Re-use Pack (see Appendix I) for the relevant testing phase.
- ▲ Update the Re-use Pack as necessary by reviewing the requirements for the enhancements or extensions to the AUT and the *Test Scripts* contained within the Re-use Packs to identify any existing *Test Cases* that may be redundant or that need to be updated (for example, a change in the requirements for the AUT may have led to the modification of an existing feature of the AUT, rendering the corresponding test obsolete or incorrect).
- ▲ Review the computing environment in which the *live system* will run in order to identify any *interoperability* or *compatibility* issues with other systems that may occur with respect to the enhancements or extensions to the system and the need to design and implement appropriate tests
- ▲ Compile a set of Regression Test Cases by re-using appropriate tests from the Re-use Packs, and execute these against the new version of the AUT. These tests should be selected on the basis of verifying that key aspects of the AUT still perform correctly, and by using experience and judgment, deciding which other aspects of the AUT may have been affected by the new extensions or enhancements.

For organizations employing an automated testing tool approach to Regression Testing, the need to determine which tests to re-use is not an issue, since all of the tests can be rerun. This is possible because of the ability of such tools to operate unattended, allowing more thorough testing of the AUT by allowing tests to be run overnight or over the weekend.

Organizations employing automated tools most efficiently use them to record the execution of the manual Test Script for a particular testing phase (such as System Testing). The resulting automated Test Script can then be replayed against a later build or release of the AUT to ensure the behavior of the AUT has not regressed. The new manual tests used to verify the extensions or enhancements to the AUT are also recorded, allowing these tests to be added into the existing library of automated test scripts (the case studies presented in Chapters 16 and 18 provide details of how this approach has been applied successfully within the context of a commercial software testing process).

11.3 Regression Test Data Requirements

Since the purpose of conducting Regression Testing is to determine whether or not the existing reliability and functionality of the AUT are unaffected because of the modification or extension of the software, it is important to keep other factors that may influence the outcome of the testing as similar as possible to those under which the tests were originally run on the unmodified software. *Test data* is one such factor.

For example, if live data is to be used within the Regression Test, it may be difficult to ensure that any defects observed are not due to changes in the data rather than changes to the AUT. Where live data is to be used, the ideal approach to

Regression Testing the AUT will involve first running the Regression Test against the test data set that was archived following the previous testing of the AUT, in order to isolate any observed defects from concerns about the data. If these tests are satisfactory, the Regression Test (or selected elements of it) can be rerun against the live data to provide further confidence in the software.

Where it has been necessary to introduce handcrafted data into the Regression Test (to exercise boundary or error conditions for the extended or enhanced functionality, for example), the Test Analyst must employ formal design techniques, such as the *Boundary Analysis* or *Equivalence Partition* methods described in Chapter 3.

11.4 Roles and Responsibilities

For Regression Testing associated with System, Systems Integration, or Acceptance Testing, the *Test Team Leader* supervises the testing process and is responsible for ensuring that adequate Regression Testing is performed using appropriate testing techniques and quality-control procedures (as defined in the *Regression Test Plan* and *Test Specification* documents).

Within the testing team, the Test Analyst is responsible for obtaining and reviewing the Re-use Packs from previous testing phases for the purposes of selecting an appropriate set of Test Scripts and Test Cases to rerun against the new version of the software, and for designing and implementing any new Test Scripts and Test Cases required to test the new extensions or enhancements to the AUT. The Test Analyst is also responsible for updating the Re-use Pack where existing tests are found to be incorrect or obsolete. Additionally, the Test Analyst may be required to create or modify existing test data for the purposes of exercising the new functionality or for verifying boundary or error conditions using the formal design techniques described in Chapter 3.

The *Tester* is responsible for executing the Test Scripts created by the Test Analyst and for observing and recording the results of testing with reference to the *Test Result Categories* recorded in the Regression Test Specification document (see Appendix D).

At Regression Testing, it is important that the testing process be monitored by an *Independent Test Observer*, who will formally witness the results of individual Test Cases. The Independent Test Observer could be drawn from the *Quality Assurance (QA) Group* in an organization where such a group exists or from another testing team or project.

The Test Team Leader must liaise with the *Information Technology [IT] Systems Administrator* (that is, the staff member with responsibility for administering the corporate IT facilities) for the purpose of installing the AUT prior to testing (this is particularly important if the Regression Test is scheduled to take place on the live environment). Similarly, the IT Systems Administrator should be kept fully apprised of the date, time, and duration of the Regression Test to ensure that there is no contention for system resources and there are no scheduled IT tasks (such as preventative maintenance) and to allow the IT manager the opportunity to perform contingency operations (such as backing up the system) prior to testing.

The Test Team Leader is responsible for filing and archiving copies of a number of the outputs generated as a result of Regression Testing (and listed in Section 11.8), including the completed *Test Result Record Forms* and Regression Test Log and a brief Regression Test Summary Report on completion of Regression Testing.

There is unlikely to be any significant benefit to inviting any *User* or *Operations Representatives* to informally attend Regression Testing associated with System or Systems Integration Testing, since the need to manage their respective expectations of the AUT should already have been considered in the previous testing phases. Where the extensions or modifications have had a major impact on the GUI, then the appropriate users should be invited to assist in any Regression Testing associated with Acceptance Testing of the AUT.

Where automated testing tools are being used to support the Regression Testing process, the Test Analyst will have responsibility for designing the automated Test Scripts and the *Verification Points* (analogous to Test Cases). The Tester will be responsible for setting up and executing the automated Test Scripts and for reviewing and interpreting the automated test log at the completion of testing.

11.5 Planning and Resources

The planning of Regression Testing is the responsibility of whoever is managing the associated testing phase. Thus, for Regression Testing conducted in conjunction with System, Systems Integration, or Acceptance Testing, the Test Team Leader plans the testing process with reference to the overall development and testing plan to ensure no contention for resources will occur.

The human resources required for Regression Testing will be drawn from the testing team (with the exception of the Independent Test Observer, who may be drawn from another project or from the QA Group, should one exist within the organization).

It is assumed that Regression Testing will take place either in a dedicated test environment or in the live environment. The choice will depend on a number of issues, including:

- ▲ The availability of a suitably specified test environment (that is, one that is representative of the live environment)
- ▲ An assessment of the commercial or Safety Critical nature of the live environment and the likelihood of the Regression Test adversely affecting it
- ▲ The occurrence of commercially or confidentially sensitive data or Security Critical information within the live environment.

In the event that Regression Testing does take place in the live environment, the Test Team Leader must liaise with the IT Systems Administrator to plan installation of the AUT prior to testing and to ensure that the manager is aware of the date, time, and duration of the Regression Test.

It is essential that the Regression Test plan take account of the time and effort required for correction of defects and retesting.

11.6 Inputs

The following items are required as inputs to Regression Testing:

- ▲ Any appropriate Re-use Packs from previous testing phases (see Appendix I)
- ▲ The *Requirements Specification* document and any supplementary design documents for the AUT (to assist the Test Analyst in determining the scope and extent of Regression Testing)
- ▲ Any supplementary material, such as user guides or prototype software
- ▲ The Regression Test Specification document (see Appendix D)
- ▲ The Regression Testing Plan (see Appendix C)
- ▲ The Regression Testing Guide (see Appendix B)
- ▲ Any additional Regression Test Scripts and Test Cases (where these need to be created by the Test Analyst)
- ▲ Blank *Test Result Record Forms* (see Appendix F).

11.7 Testing Techniques for Regression Testing

The following testing techniques are appropriate for Regression Testing:

- ▲ *Black Box Testing* against high-level system requirements to verify that key aspects of the AUT still perform correctly
- ▲ Black Box Testing against other aspects of the AUT that may have been compromised by the new extensions or enhancements
- ▲ *Thread Testing* against the high-level business requirements for the AUT
- ▲ *Nonfunctional Testing* (such as Volume, Stress, and Performance testing), where appropriate.

See Chapter 3 for details of these testing techniques.

11.8 Outputs

The following items are generated as outputs from Regression Testing:

- ▲ The fully Regression-Tested system
- ▲ The completed Regression Test Certificate (see Appendix H)
- ▲ Any revised Test Scripts and Test Cases (where appropriate)
- ▲ The archived test data (where additional data has been employed in the Regression Test)
- ▲ The completed Test Result Record Forms (see Appendix F)
- ▲ The Regression Test Log (see Appendix G)
- ▲ The Regression Test Re-use Pack (see Appendix I).

Regression Testing will be considered to be complete when all of the above deliver-

ables are complete (Figure 11.2) and copies (under strict configuration management, as described in Chapter 4) have been provided to the Test Team Leader.

Appropriate deliverables (that is, all of the above except the tested system) should be stored in the project file. The fully tested system (plus any associated *test harness* or *simulation* code and test data) should be backed up and archived.

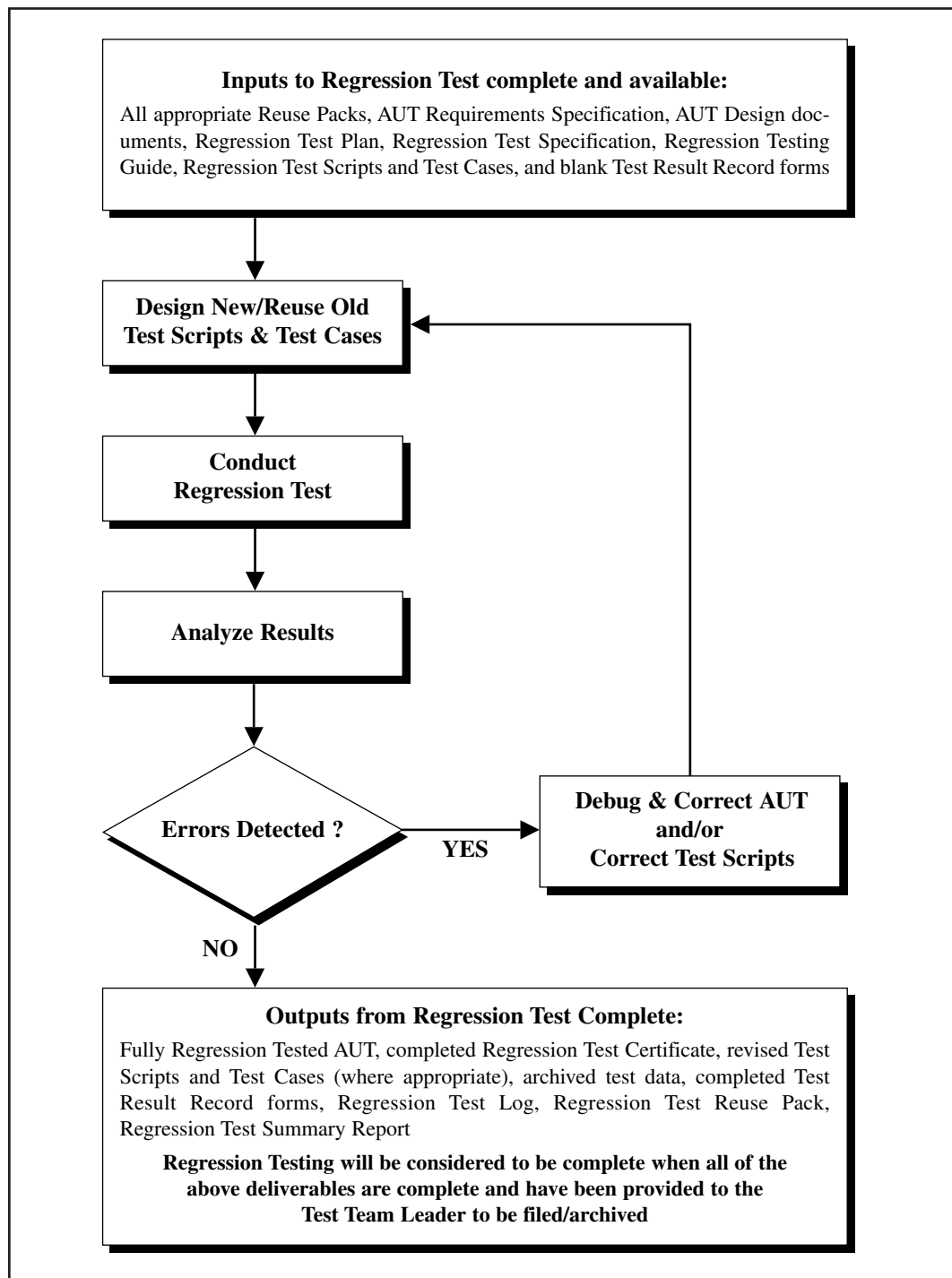


Figure 11.2. High-level overview of the Regression Testing Process.

Improving the Testing Process

Those who forget the lessons of history are doomed to repeat its mistakes.

George Santayana

12.1 Introduction

The idea that we should try to learn by our experiences is both natural and intuitive, but one we frequently seem to ignore in a software development and test context.

One particular approach to “learning by experience” in a project environment is to collect and use metrics observed during the course of the project, such as how much effort was expended in completing the project, how many defects were found during testing, or how many defects were found by the client¹ following delivery of the tested system. This information can subsequently be used to determine whether the testing process is improving over time, by, for example, comparing the current effectiveness in finding defects against previously observed results.

Metrics can also be used to investigate the benefits of adopting some specific approach to testing, such as the introduction of *automated testing*, by comparing the efficiency of the testing process before introduction with that observed following rollout of the tool. The Crown Management Systems case study (Chapter 16) provides just such an example of the role and use of metrics in a testing project environment.

If no formal, quantitative measurements are made, it is possible only to make qualitative statements about the effectiveness of the testing process, which may in the short term assure senior management, but which in the long term will not help to improve the testing process. Typical examples of where metrics can be used in the testing process include:

- ▲ For objective assessment of testing quality against agreed standards
- ▲ For estimating the testing effort required to complete a given testing project/task

¹The term “client” could refer to an external customer for whom the AUT was developed under contract or could represent an internal “customer” within the same organization.

- ▲ For highlighting complex elements of the system under test that may be error-prone and require additional testing effort
- ▲ To measure and track the cost of testing
- ▲ To measure the return on investment (ROI) of a particular approach or tool (such as an automated software testing tool [see Reference 25 for example])
- ▲ To assess and track the progress of the testing task
- ▲ To predict when it is appropriate to stop testing a particular *Application Under Test (AUT)*.

Formally, metrics are objective numerical measures obtained by inspection and analysis of the products and processes of a software project. Although metrics can be collected at all stages of the software development lifecycle, this chapter focuses on those of relevance to the testing process.

This chapter discusses the role and use of metrics in process improvement, reviews the metrics employed in such programs, discusses the issues involved in setting up and adopting a metrics program, and makes a number of proposals for a simple and effective metrics set that can be used to improve the testing process.

12.2 Overview of the Role and Use of Metrics

Although the Statistical Techniques clause of ISO 9001 provides guidance on the adoption of a metrics-based approach, there is at present no single universal standard for software metrics in the *information technology (IT)* or software development arena. However, a great deal of work has been performed in the IT industry on software metrics, and many metrics and metrics sets have been proposed and adopted by different authors and organizations.

The following statement provides a useful working definition of a metric:

A Software Metric is a standard means of measuring some attribute of the software development (and testing) process. [Reference 11]

A further useful distinction can be made between Primitive and Computed metrics:

- ▲ A *Primitive metric* is one that can be measured directly. Examples include:
 - the number of defects (the total number of reported defects, for example)
 - the complexity of the AUT (based on function points within the AUT, for example)
 - the cost of testing (the combined total cost of testing the AUT, for example).
- ▲ A *Computed metric* is one that must be calculated from other data or metrics. Examples include:
 - the number of noncomment lines of code written per day
 - the defect density (number of defects per line of noncomment code, for example)

- the number of defects detected/reported per unit time or per development phase.

Primitive metrics typically form the raw data for a metrics program and will represent the observed data collected during the project. Often, plotting the progress of Primitive metrics over time is a powerful means of observing trends within the testing project (or projects). For example, plotting the numbers of defects detected during a testing project against time can provide the *Test Team Leader* with one means of determining when to stop testing by observing when the rate of detection of defects declines.²

Computed metrics typically form the basis of forming conclusions regarding the progress of a process-improvement program. For example, observing the defect detection effectiveness percentage achieved by a *Testing Team* across a number of testing projects (which is based on the relative effectiveness of the Testing Team at detecting defects as compared with those reported by the user following delivery of the AUT – see Section 12.3) provides a valuable indication of the changing efficiency of the testing process over time.

12.3 Metrics Typically Used within the Testing Process

This section reviews the commonly collected and calculated metrics that can be used in improving the testing process. Where the metric discussed is Computed, the means of calculating the metric is described with reference to the appropriate Primitive metrics.

Size (how “big” is the AUT?)

This metric is typically measured as NCSS (noncomment source statements or lines of code). In an object-oriented or GUI-based development, information on the number of objects (and their methods) or windows could be collected. This metric must be collected pragmatically – it does not make good sense manually to count every line of code of a huge application (however, an automated approach using a line-counting tool might be appropriate).

In using the information on the size of different applications, it is important to compare like with like (for example, comparing lines of code in a traditional application with numbers of objects is unlikely to provide meaningful conclusions). However, if enough information is collected within organizations using a number of different implementation technologies, it may be possible over time and with sufficient observation to determine some form of “conversion” factor between such technologies, which will allow useful comparisons to be made.

²Great care must be taken when using such information within the planning process for the testing project, since decline in numbers of observed defects could be based on other factors, such as tester fatigue.

Size metrics on their own may prove insufficient to enable any significant conclusions to be made. The complexity of the software should also be considered within any metrics program.

Complexity (how “complicated” is the AUT?)

This metric attempts to take account of the complexity of the AUT in terms of iteration, recursion, conditional branching points, function points, and so on. Caution must be exercised in the use of this metric, since it will be difficult to produce generally applicable conclusions for organizations involved in a number of software development and testing projects, each of which utilizes different implementation technologies.

This is a difficult metric to collect; however, complexity-estimating tools are commercially available that will simplify the collection and use of complexity information (see Reference 33, for example).

As with size of the AUT, this metric works best where like is compared with like.

Cost/Effort (how much does it cost/how much effort is expended to test a given piece of software and associated documentation?)

This metric is typically measured as a payroll month and includes time taken by staff doing testing, as well as time spent by managers engaged on testing tasks.

It may also be of benefit to measure the Cost/Effort involved in administering and collecting the metrics, since comparison of this value with the total effort expended in the testing process will provide information about the efficiency of the metrics program.

Total defects found in testing (how many defects were detected during testing?)

In this context, a defect can be defined as any aspect of the behavior of the software that would not exist if the software were fit for purpose. It may be of benefit to consider recording the severity of observed defects to help in comparing their relative impact. A simple three-point scale of Critical, Serious, and Minor is often sufficient for this purpose.

Total defects found by client (how many defects were reported by the client after release of the software?)

This value should be recorded consistently after a standard interval following release of the AUT (such as after three months and/or six months – clearly the time period adopted will depend on the usage and expected lifetime of the system, as well as its planned release schedule).

If a severity scale has been adopted to record the total defects found in the testing metric, the same scale should be employed with this metric as well.

Test development progress (how many tests have been planned, designed, run, and passed/failed?)

This category also includes test coverage (how much of the AUT has been tested?). These metrics are particularly appropriate for generation by testing tools (such as the reporting facilities provided by a test-management tool – see Reference 25, for example).

This metric could also consider the numbers of defects reported but not rectified and the progress of correcting defects over time and/or by developer.

Difficulty (how difficult is the software to develop and test?)

Although the difficulty of a software project may seem to be a subjective issue, it is possible to introduce a qualitative approach by breaking the project down into sub-items and considering each in turn. Examples of categories could include:

- ▲ The stability of the requirements for the AUT (are the requirements for the AUT fixed, or do they change frequently in response to client requests?)
- ▲ The level of experience of the testing staff (for example, are the testers experienced practitioners, or are they new IT graduates who have been assigned to a testing project as their first task on joining an organization?)
- ▲ The level of familiarity of the testing staff with the technology being tested (have the testers worked with applications developed using similar technologies, other applications in the same product family, or previous releases of the AUT?)
- ▲ The ease of access to facilities used in the testing process (do the testers have full and free access to the AUT for testing, are there issues with confidentiality or security that mean real data cannot be used in testing, are productivity tools available to support the testers [such as automated testing tools]?).

For the majority of organizations that are using traditional tried and tested implementation technologies and methods, the use of this metric is unlikely to provide much benefit. It may be of benefit to projects using “leading-edge” methods or technologies.

Communications (how much effort is spent in communicating with other entities?)

This metric is typically measured as the number of interfaces that a given project team has, with the purpose of characterizing the constraints on the project team due to dependencies with entities organizationally and physically distant, such as users, managers, and/or the supplier.

This metric is most useful for large and/or complex organizations where development and testing take place across geographically distinct sites. In particular, the metric can be used to identify possible improvements to the organization and administration of complex testing projects and provides a means for assessing the effectiveness of such improvements. For small testing projects involving few staff located in the same office or site, there is unlikely to be a great deal of benefit from the use of this metric.

Defect detection effectiveness percentage (DDE)

This Computed metric will provide an indication of how effective the testing process is over time (the DDE should increase with an increase in the effectiveness of testing). DDE is calculated as follows:

$$\text{DDE} = (\text{TDFT} / (\text{TDFC} + \text{TDFT})) \times 100, \text{ where}$$

TDFT = Total Defect Found by Testing (that is, by the testing team)

TDFC = Total Defects Found by Client (measured up to some standard point after release – after 6 months, for example)

Defect removal effectiveness percentage (DRE)

This Computed metric will provide an indication of how effective the testing task is at removal of defects. DRE is calculated as follows:

$$\text{DRE} = (\text{TDCT} / \text{TDFT}) \times 100, \text{ where}$$

TDCT = Total Defects Closed during Testing

TDFT = Total Defects Found during Testing

Test case design efficiency percentage (TDE)

This Computed metric will provide information about the effectiveness of the Test Case design process. TDE is calculated as follows:

$$\text{TDE} = (\text{TDFC} / \text{NTC}) \times 100, \text{ where}$$

TDFT = Total Defects Found by Testing

NTC = Number of Test Cases run

12.4 Setting Up and Administering a Metrics Program

Prior to setting up an effective metrics program, a number of high-level issues must be considered.

It is important that measurable or objective targets be set. Similarly, the temptation to set targets that are too low in order to obtain “good” results, or the editing/optimistic interpretation of information to produce the desired result must be avoided.

Care must be taken to ensure that, in setting up the administrative aspect of the approach to metrics collection, excessive control through measurement is avoided. An overly intrusive and stringent regime may result in the stagnation of initiative in the staff involved by limiting experimentation and imagination.

Similarly, care must be taken to ensure that the process of collecting a metric does not affect the item being measured (for example, the manner in which you phrase a question can directly influence the way someone answers the question [known as the Hawthorne Effect – References 9 and 32]). It is important that the collection of metrics be as unobtrusive as possible.

In adopting such an approach, it is important to remember that the metrics program is itself an additional overhead to the software development and testing

process, and will involve additional effort as part of the initial learning curve. It is important to employ the KIS principle (Keep It Simple) at least to begin with, expanding and extending the scope of the program with increasing familiarity and success. Project control can be achieved using metrics quite quickly; project improvement will take longer – so be prepared to persevere.

Do not underestimate the importance of the involvement and motivation of the staff involved in the metrics program. Without staff buy-in, the process will be made much more difficult. Often, staff perceive metrics to be career threatening and under such circumstances may (deliberately or accidentally) provide incorrect or overly optimistic information. It must be made clear that the metrics program is not being set up to apportion blame but rather to improve the software development and testing process.

The following requirements need to be considered in setting up a typical metrics program, and should be reviewed to determine those appropriate within the context of the scheme planned as part of any specific testing process:

- ▲ The need to define organizational objectives for the process-improvement program to establish:
 - *the methods to be used*
 - *the costs that will be deemed acceptable in running the program*
 - *the urgency of the program*
 - *the required support from management for the program (and which should be formally recorded, if possible).*
- ▲ The need to establish the following roles and responsibilities:
 - *what part of the organization will have responsibility for the program?*
 - *who will manage, implement, and administer the program?*
 - *who will submit/collect the metrics?*
- ▲ The need to research which metrics to record and what analysis is required. During this process the following issues must be considered:
 - *Whatever metrics are selected initially will almost certainly change. They will be modified by experience during the metrics program itself.*
 - *Ensure that results of analysis are immediately useful to project management. If you continue to collect metrics with no visible benefit, management will become discouraged with the process.*
 - *Be aware of what is realistic (for example, although of potential interest, the number of keystrokes made by an engineer is not a realistic metric to record).*
 - *Err on the side of generosity when selecting metrics to collect. It may be difficult to go back and obtain metrics for a parameter that was initially not thought to be of interest, but which subsequently turns out to be useful. You can always drop a metric later if it turns out to be of no use.*
 - *Be aware that the more metrics you decide to record, the greater the effort to collect, the greater the project resistance to the process, the larger the requirements for storage and processing of the metrics.*

- ▲ The need to set up the infrastructure for recording the metrics. The following steps are likely to be implemented:
 - *Set up a system for holding the metrics (such as a database or spreadsheet).*
 - *Generate simple procedures for defining what metrics are to be collected and how this is to be achieved, and formally document them in a form that can be distributed and easily understood.*
 - *Set up the means for staff to retrieve metrics information (such as database access).*
- ▲ The need to “sell” the initial set of metrics. The success of the metrics program will depend upon the accuracy and coverage of the data collected, and this will depend upon the commitment of the staff in taking the time to collect and submit it. They will need to be convinced of the benefits. In particular, try to identify staff who may hamper the process and seek to involve them to gain their acceptance. Where such staff can be involved, they often become powerful advocates for the process.
- ▲ The need to obtain tools for data collection and analysis. Tools will help simplify collection of metrics, reduce time expenditure of staff, promote accuracy and consistency, and help reduce staff resistance to performing the collection. A tool could be as simple as a customized spreadsheet, which staff can complete each week and e-mail to the administrator, or as elaborate as a commercial metrics package.
- ▲ The need to review the process. It is important to ensure that the software metrics program is not static. It will be necessary to review the metrics being recorded and question their benefit at regular intervals so as to drop metrics that do not prove to be useful and to consider introducing new metrics. However, it is important to be aware of the impact of constant changes on the staff involved in recording and using the metrics. As a major goal of the program, you should aim to standardize a metrics set as quickly as possible.

12.5 A Proposal for a Simple and Effective Metrics Set

This section makes a number of recommendations for a basic set of metrics that can be used over time to improve the effectiveness of the testing process. The following metrics are straightforward to collect (and calculate in the case of the Computed metrics) and should not impose a significant overhead on the overall testing effort:

- ▲ Testing Effort – the total effort expended in testing (and retesting) calculated as person/hours
- ▲ Metrics Effort – an estimate of the effort expended in the reporting, collection, storage, and analysis of the metrics information. This information will be invaluable in determining how much influence the metrics program itself is having on the overall testing project results

- ▲ **Metrics Efficiency Percentage** – a Computed metric that shows the relative influence of the metrics program on the overall effort expended during the testing project. Clearly, the goal should be for this value to be as small as possible. If the value becomes a significant fraction of the overall testing project effort and if the benefits of running the metrics program are not clear (for example, it has not been possible to demonstrate any significant improvements in the testing process over time), then the continued role of the metrics program should be reviewed

$$\text{Metrics Efficiency Percentage} = (\text{Metric Effort} / \text{Testing Effort}) \times 100$$

- ▲ **Test Planning, Design, Execution, and Coverage** – information should be collected to show how many tests have been planned, designed, run, and passed/failed. If a test-management tool is being used, such products typically provide built-in reports to analyze and display this information. This information is a key input to the management aspects of the testing process in terms of the ability to measure and report progress
- ▲ **Number of *Test Cases* (NTC)** – the total number of Test Cases run against the AUT for a particular testing phase. This will also be useful for estimating how many Test Cases can be created for a given amount of effort and also shows how many Test Cases were run to find a given number of defects
- ▲ **Size** – an estimate of the size of the AUT based on the implementation technology (used in combination with NTC, this provides a useful planning metric for estimating the effort required to test other AUTs of a similar implementation technology)
- ▲ **Total Defects Found by Testing (TDFT)** – the severity value associated with each reported defect should be recorded when using this metric (perhaps based on the *Test Result Category* assigned to the defect during testing). For example, a simple three-value system of Critical, Serious, and Minor could be adopted to categorize observed defects and to allow more meaningful comparisons of defects to be made
- ▲ **Total Defects Found by Client (TDFC)** – the same severity values adopted for recording the TDFT metric should be used to allow meaningful comparison between TDFT and TDFC to be made. It is recommended that this metric be measured consistently at two standard points after delivery to the user, such as 3 and 6 months (but this will clearly depend on the release schedule for the AUT; for example, a 6-month sampling period will not be appropriate for software with a 4- month release schedule)
- ▲ **Defect Detection Effectiveness Percentage** – to determine how the effectiveness of the testing process improves over time, based on the numbers of defects detected during testing and those observed during normal operation of the AUT by the user
- ▲ **Defect Removal Effectiveness Percentage** – principally used to determine the progress of the testing process during testing
- ▲ **Test Case Design Efficiency percentage** – to determine how the efficiency of Test Case design changes over time (such as between testing different releases of the same AUT)
- ▲ **Rate of Defect Detection** – the rate of defect detection by the testing team during testing (and optionally, in live use by the client). It is recommended that defects be categorized as either serious or minor bugs to simplify analysis. Analysis of this information is useful in determining when to stop

testing,³ as well as for showing defect trends across the duration of the testing project

- ▲ Rate of Defect Correction – the rate at which the developers correct defects, deliver the software for retesting, and verify that defects have been rectified. It is also recommended that where several developers are responsible for correcting defects, their individual rates of defect correction are collected. This is very useful information for both the planning and management of testing, since it can be used to estimate timescales for correction of defects and also to predict which developers are most effective at clearing defects.

12.6 Further Reading

Reference 1 provides good advice regarding the role and use of metrics within the context of testing projects. Reference 17 contains very useful information regarding the use of metrics in association with automated software testing tools. Reference 31 is an excellent general text on the role and use of metrics in the context of the software development process.

³If the rate of defect detection is plotted against time, inspection of the resulting graph can be used to predict the most effective point to stop testing – that is, when the graph trace has leveled off.

Introduction, Adoption, and Maintenance of the Testing Process

Learn to Test, Test to Learn.

Motto of the Empire Test Pilot School

13.1 Introduction

Even the very best software testing process is nothing without the acceptance and consistent use by a community of testers within an organization. Technical issues aside, the introduction and adoption of a testing process is likely to be the most difficult step in providing a formal, rigorous, and re-usable process within an organization.

This chapter examines the issues involved in introducing a testing process into an organization and the management of its successful adoption and use. It also addresses the issues associated with the subsequent maintenance and update of the testing process.

13.2 Introduction and Adoption of a Testing Process

13.2.1 Overview

This section discusses the need to first establish that there is a requirement for a formal testing process within an organization, and where this is the case, examines the need to develop a strategy for the introduction and adoption of the process.

13.2.2 Establishing the Requirement

The first step in introducing a testing process within an organization is to establish that there is a requirement for such a process. In determining whether a particular organization has a requirement for a formal testing process, the following issues should be considered:

- ▲ How much software testing does the organization conduct, and how frequently does testing take place? Clearly, if an organization performs very little testing and/or this task takes place infrequently, there will be a weak requirement for introducing a large-scale testing process and the associated infrastructure. However, there may still be significant benefit to be gained from documenting a standard approach to testing with some associated testing templates, to ensure that when testing does occur it is as consistent and economical as possible
- ▲ How many departments/divisions of the organization are involved in software testing, with how many distinct testing projects; and for each project, how many staff are involved in testing tasks? If there are many staff members engaged in a number of different testing projects in separate departments across the organization, then there may be significant benefit from standardizing the testing process, and setting up a testing process infrastructure to offer guidance on testing and to provide access to standard reusable templates, checklists, and proformas
- ▲ How diverse is the testing requirement? Is the organization engaged in testing software developed in-house, under contract by third-party developers, or bought in as *commercial off the shelf (COTS)* software. Does the organization have a requirement for all phases of testing from *Unit* through *Acceptance Testing* and on to *Regression Testing*? Under such circumstances, an organization is highly likely to benefit from the introduction of a standard, re-usable testing process, with well-defined roles and responsibilities and with testing phases, each with well-defined relationships and clearly outlined inputs and outputs.

One main reason for adopting and using a process is to save the organization time, effort, and money. If the cost of introducing and using a testing process is greater than the cost of continuing with the current approach to software development and testing, then the benefits of introducing a process must be carefully considered.

Although an important issue for a great many organizations, saving time, effort, and money is not the only reason for adopting a formal development and testing process. Other reasons for introducing and following a process could include the need to address a serious flaw in the existing development and testing process – such as widespread quality problems, the need to formalize the software development and testing process as a prelude to gaining some form of quality certification (such as Reference 14, for example), or as part of a software process-improvement initiative (such as Reference 27).

13.2.3 Strategy for Introduction and Adoption

Once the requirement for a formal testing process has been established, it is necessary to determine a strategy for the introduction and adoption of the process. The process of formulating such a strategy includes the following key tasks:

- ▲ Gain management commitment and involvement.
- ▲ Identify a testing process champion.
- ▲ Identify baseline current software testing practices.
- ▲ Identify related organizational processes.
- ▲ Consider the scope and scale of the testing process.
- ▲ Ensure that training and mentoring are available.
- ▲ Monitor progress and improve the process.
- ▲ Advertise results.

Each of these tasks is described in the following sections.

The first key task is to ensure that the introduction of the testing process is fully supported by the appropriate senior management within the organization. It is important that they be made aware of the importance of a formal testing process as well as the potential cost of not following good testing practice, perhaps by means of a brief proposal. A good indication of management commitment is the promise of resources to the project, and a major goal of any strategy should be to obtain a guarantee of appropriate staff and facilities.

An important part of the process of gaining both management commitment and adoption by other staff in the organization is to identify a testing process champion to help promote the introduction and adoption of the testing process. The champion will need to be a relatively senior and/or well-respected staff member who can communicate at all levels within the organization from junior staff members through senior management. He or she will need to have the appropriate technical skills to understand the testing process as well as an appreciation of the need to introduce a formal approach to testing. Clearly, good communication skills combined with enthusiasm will be key attributes of such a person. The responsibilities of the champion can include lobbying individuals, making presentations to management, holding speaking events (such as lunchtime seminars), advertising events and successes, and identifying points of resistance and working to overcome them.

In order to understand how to introduce a testing process, it is important to identify the baseline for introduction of the process by taking stock of just where the organization is in terms of its current practices for software testing. Appendix P provides a Testing Process Health Check (in effect, a set of criteria and a scoring scheme that can be used to conduct a testing audit), which can be used to help identify exactly where an organization is in terms of its testing practices, identifying the positive aspects of the process as well as those that need improvement. This activity should be documented in a brief report summarizing the findings of the health check, which can be used as supporting information in the proposal to management for implementing the testing process.

Next, it is necessary to identify what other existing processes within the organization may impact on the testing process, such as development standards and management practices, since the testing process cannot exist in a vacuum and must interface and interact with these other processes. For example, a particular software development process may already mandate an approach to Unit Testing, which will have to be reflected in the overall testing process. Similarly, the existing management infrastructure may have implications for which part of the organization will administer the testing process (such as the *Quality Assurance (QA)* or *information technology [IT] groups*). The output from this activity will be a brief report, which will document the current organizational processes and how they will interface to the proposed testing process, as well as a diagram showing the present organizational infrastructure and how and where the testing process infrastructure will fit.

For a number of reasons, it is very unlikely that an organization (particularly a medium to large enterprise) will attempt to roll out an organization-wide testing process.¹ A more likely strategy will be to conduct a small-scale rollout of the process, which will have the benefit of being a much more attractive alternative to risk-conscious management. The use of a pilot study or project may represent a useful option to enable the testing process to be rolled out on a small scale to one or two testing projects within the organization, allowing the benefits of the testing process to be demonstrated prior to attempting a large-scale rollout. A pilot study will also allow the lessons learned during the use of the testing process to be addressed prior to large-scale adoption. If the launch of the pilot study and its subsequent achievements are effectively advertised to the rest of the organization (perhaps by the champion), this can provide an effective means of promoting the benefits of the testing process.

In ensuring the success of the introduction and adoption of the testing process, it is essential to provide suitable training for staff members expected to use the process and appropriate mentoring to ensure the continued successful use of the process. One goal of the pilot study should be to ensure that the staff involved will acquire the appropriate skills to be able to provide subsequent training and mentoring to other staff. The champion should also be considered as a resource to provide training and mentoring to other staff in need of guidance. The Testing Process Health Check provided in Appendix P can also be used to assess the progress of individual projects, with the health check used to identify aspects of their use of the process that may need improvement.

From the outset of the introduction of a formal testing process, it is vital that progress be monitored so that the benefits of introducing the process can be quantified and the process can be improved. Management, for example, will certainly demand progress reports that are likely to include figures on the return on investment of introducing a formal testing process, and tangible metrics will be needed to promote the success of the testing process to the rest of the organization. Chap-

¹Reasons include cost and organizational difficulties of large-scale rollout of any novel process, the risk of failure of the process in practice, and the cost of reverting to the original state, organizational inertia, and (particularly in very large, distributed organizations) traditional staff resistance to change.

ter 12 provides good advice on what metrics to measure and how they can be used both to show the progress of the introduction and adoption of the testing process and to help improve the overall effectiveness of testing.

The need to advertise the results of the introduction and use of the testing process has been discussed a number of times in the preceding paragraphs. However, the importance of letting the rest of the organization know about the benefits that a formal testing process can bring, and of the successes of such a process, cannot be underestimated. It is possible for a testing process to fail simply because the results of a successful pilot study were not effectively reported to anyone else in the organization! The availability of an effective champion will be a major channel for spreading the word, but the testing technical staff also have a roll to play in publishing articles on the organization's intranet, producing newsletters, holding lunchtime seminars, getting technical papers published at conferences, and so on. Setting up a testing special interest group within the organization is another powerful way of ensuring information is shared by everyone involved in the testing process. Finally, make sure management is briefed on a regular basis about progress and successes.

13.3 Maintenance of the Testing Process

Almost as soon as it has been implemented, any testing process will be out of date. At the very least, the process will need to change to take account of lessons learned during its use. Other reasons for change can include alterations to the organizational infrastructure (such as the introduction of a new QA Group), changes in the implementation technology or approach to software development, and introduction of automated testing tools or as the result of a formal process improvement initiative.

The testing process should be reviewed at key points, and where changes need to be made, proposals should be submitted to the *Testing Manager* (or equivalent, should this role not exist in a particular testing process) for appraisal and subsequent action. Mandatory review points should include:

- ▲ The end of individual testing projects (such as a project close-down meeting) in order to identify any lessons learned during the project
- ▲ Following changes in the software development process (such as the adoption of a rapid prototyping software development method [see Reference 12])
- ▲ As a prelude to the adoption/introduction of new testing tools or techniques (such as test automation tools [see Reference 22])
- ▲ Following changes in the management approach of the organization (such as the adoption of the PRINCE project management methodology [see Reference 3])
- ▲ As a result of trends observed or as a result of the analysis of metrics collected during a process improvement initiative.

Other desirable review points might include:

- ▲ At regular dates such as an annual or biannual review
- ▲ As and when required by the Testing Manager (or equivalent)
- ▲ As and when required by the QA Manager (should one exist for a particular organization).

Whatever review points are adopted as part of the review process, they should be documented as part of the testing process. Following such a review, where changes need to be made to the process, or where sections are found to be no longer relevant, the following measures should be taken:

- ▲ If a part of the process is no longer relevant, it should be removed from the testing process and the process document amended appropriately.
- ▲ If the existing material is only partially relevant (material that is out of date or superseded by more recent information, for example), it should be updated to reflect the new process view.

Where any part of the testing process is updated, removed, or otherwise changed,² the following parts of the testing process document should be checked and where necessary modified:

- ▲ Update the Contents of the testing process to reflect any changes.
- ▲ Update Chapter 1, Section 1.4, Structure and Content . . . , to reflect any changes.
- ▲ Review embedded chapter, appendix, section, figure, and table references and amend as necessary.
- ▲ Where references to external sources have been added or removed, update the references section to reflect these changes. Where these changes have altered the order (that is, numbering) of any existing references, those points in the body of the document where they are cited must also be amended
- ▲ Where terms, phrases, abbreviations, or acronyms have been removed, altered, or introduced, ensure that the glossary is amended to reflect these changes.

Finally, the configuration management details of the testing process document should be amended to reflect the changes.

²This assumes the testing process document has followed a content and structure similar to that given in Part I of this book. If this is not the case, the amendments should be made to the analogous sections of the process document.

Part **2**

The Testing Process in the Real World: Illustrative Case Studies





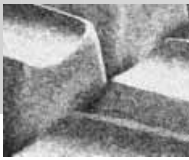
This section of the book provides a number of real-world examples of organizations that are involved in testing and that have implemented their own testing processes in collaboration with the author. The organizations are:

- The British Library
- Reuters Product Acceptance Group
- Crown Quality Assurance Group
- The Wine Society
- Automatic Data Processing (ADP) Limited.

These case studies are provided to allow readers to gain some insight into how they might adapt the classic view of the testing process presented in Part 1 of this book to more closely match their own specific testing requirements. This will be particularly valuable for readers who have been given the responsibility of setting up a testing process within their own organizations.

Each case study is presented in a standard manner, which follows the order of the testing process material presented in Part 1. For each case study, the following sections are provided:

- Overview of the Organization – providing general information about the organization and the nature of its business
- Characteristics of the Testing Requirement – describing the particular testing requirements and challenges faced by the organization
- The Management and Planning of Testing – discussing how the organization arranges its testing activities and how it plans and manages the testing process



-
- Roles and Responsibilities – describing the testing roles supported by the organization and reviewing their specific responsibilities
 - Testing Phases – describing the specific characteristics of the testing phases employed by the organization in the execution of its testing projects
 - Artifacts – describing the testing documentation used by the organization in its testing projects, as well as describing the means by which each organization delivers these artifacts to the testing projects
 - And, where appropriate, Process Improvement – describing the means by which the organization monitors its testing process, a review of the metrics it collects, and the approach adopted in improving the testing process.

The following sections provide brief descriptions of each of the organizations and the characteristics of their development and testing requirements in order for readers to identify quickly particular case studies that most closely reflect the characteristics of their own testing requirements.¹

British Library

The British Library (BL) can be characterized as a very large organization with a number of geographically separate sites. BL has a requirement to test a wide variety of software developed in-house, produced by third-party

¹Although the Wine Society and ADP case studies describe historical testing projects, the present tense is used throughout the descriptions for the sake of consistency. The correct tense is used within the individual case study chapters.



suppliers, and bought in as *commercial off-the-shelf (COTS)* software; as such, the BL testing process must cover all phases of testing. Typically, the software under test is destined for internal use within BL in support of its business.

Depending on the particular testing project, the BL testing process must address simple to complex data requirements and the need for test harness and simulation software use, as well as test automation tools where appropriate. In order to make their systems easier to use and to reduce the cost of training and Help desk support, BL has a particular interest in conducting *usability testing* wherever appropriate.

The BL testing process is administered from within the *Information Technology (IT)* department with strong input from the *Quality Assurance (QA)* Group. BL has a strong project management culture based on the PRINCE method, which has implications for the management and organization of testing as well as for the role of testing process improvement.

Reuters Product Acceptance Group

The Reuters Product Acceptance Group (PAG) can be characterized as a large testing group that has responsibility for *Acceptance Testing* a number of Reuters financial dealings products developed by a third-party supplier. Following successful testing, the AUT is provided to a number of Reuters customers for use in supporting their business practices.

PAG supports a large, configurable testing laboratory in order to provide confidence that the software under test



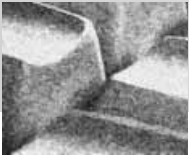
will perform successfully on the heterogeneous customer computing platforms as well as provide a realistic source of *test data*. *Usability testing* plays a significant role in the testing process as Reuters believe good usability is one of a number of key characteristics that give its products an advantage over its competitors. PAG does not use any automated software testing tools in its testing process.

PAG is administered from within the Reuters IT group (and specifically the Reuters Information Division). Although having reviewed and rejected ISO 9000, PAG does follow a number of aspects of the standard. PAG follows a process-improvement approach based on the Capability Maturity Model (CMM – Reference 35).

Crown Quality Assurance Group

The Crown Quality Assurance Group (Crown) can be characterized as a medium-sized testing group that has responsibility for the Acceptance and *Regression Testing* of new releases of commercial point of sales (POS) software systems. Following successful testing, the software is released to a number of Crown customers who trade in the hospitality industry.

Crown needs to support a flexible test rig that can be configured to match the specifications of its various customer installations. Rigorous configuration management is employed to ensure the correct set-up of hardware, software, and data for each test (and also to recreate the test rig where it is necessary to reproduce a particular test). Crown is committed to using automated software testing tools in order to reduce testing timescales, effort, and cost and to improve the quality of testing.



This approach is run under the auspices of a European Union Process Improvement project (EU ESSI – Reference 27).

The Crown testing group is administered from within the company's QA department. As part of the EU ESSI project, Crown collects and analyzes a comprehensive set of software development and testing metrics, which are fully documented in the case study.

The Wine Society

The Wine Society case study can be characterized as a small testing project with responsibility for Acceptance and Regression Testing of new releases of a commercial software product that supports the major aspects of the Wine Society business process. This software is derived from a standard commercial product that is customized under contract to the Wine Society by a third-party supplier. Additionally, there is a supplementary requirement for performance and usability testing within the testing process.

Since the Wine Society is Acceptance Testing the AUT for subsequent internal use, there is no significant requirement for complex test rig facilities. An interesting feature of the Wine Society testing group is the deliberate policy of resourcing members of the group from the user community. The testing process also makes use of external testing consultants to assist in the management and planning of testing, as well as to provide testing process advice.

Wine Society testing group is administered from within the Information Systems (IS) group and obtains user rep-



representatives from the various business areas of the Wine Society as and when required.

Automatic Data Processing Limited

The Automatic Data Processing Limited (ADP) Acceptance Testing Team (ATT) can be characterized as a small testing project with responsibility for Acceptance and Regression Testing of incremental releases of a commercial payroll package. Following successful testing, the package is delivered to a number of ADP customers to support their internal payroll processes. Some performance testing is also conducted to provide confidence that the AUT will perform adequately on the customer hardware.

For the majority of the ATT testing activities there is no particular requirement for test rig hardware or software, except for the performance-testing activities, where a small Client/Server rig was set up to perform *load* and *stress* testing. The testing project makes extensive use of automated software testing tools for both *functional* and regression testing, as well as *performance testing*.

The ATT is managed from within the ADP QA department.

Case Study 1: The British Library

Ken Eves, British Library IT Quality Manager

14.1 Overview of the Organization

The British Library (BL) is the national library of the United Kingdom. Its work is supported by the expertise of over 2000 staff members and by links with other organizations worldwide. The Library provides:

- ▲ Internationally important reading room and enquiry services
- ▲ The world's leading document supply services
- ▲ Specialist information services in key subject areas
- ▲ On-line and Web-based services
- ▲ Essential services for the library, archives, and information world
- ▲ Library facilities for the general public.

The Library's outstanding collection, developed over 250 years, of more than 150 million items represents every age of written civilization, every written language, and every aspect of human thought.

BL can be seen as the most complete of all the case studies, embodying aspects of each of the process elements described in Part 1 of this book.

14.2 Characteristics of the Testing Requirement

The British Library has a significant reliance on software systems to support its business for the purposes of:

- ▲ Maintaining very large quantities of information on document holdings
- ▲ Maintaining extensive lender personal details
- ▲ Recording details of document borrowings
- ▲ Maintaining on-line catalogue searching (involving millions of records)
- ▲ Supporting large-scale document supply.

The majority of these systems need to interoperate in order to inspect or exchange data and/or to invoke operations on other systems. The systems in use are of vari-

ous ages, based on a variety of implementation technologies, and frequently supported on different operating systems and hardware platforms.

In terms of the sources of these software systems, BL supports software development in-house, by third parties, and acquired from software vendors in the form of *commercial-off-the-shelf (COTS)* packages. Additionally, BL operates a number of legacy systems, which need to be extended and maintained. Typically, these systems must be integrated with existing BL software systems, leading to a significant systems integration and *testing* requirement. Additionally, there is a significant requirement for thorough and extensive *regression testing*.

Software is developed in BL at two separate sites – in Boston Spa Yorkshire and at the Saint Pancras offices in London, each with numerous software development and testing projects. In terms of the quality of its IT development and testing process, the BL Saint Pancras site has won medals in the annual British Computer Society Awards for IT.

In combination with BL's strong quality culture, the geographically separate development sites and the number of different software development and testing projects have been significant drivers in the development of the British Library Testing Framework.

14.3 The Management and Planning of Testing

BL follows a comprehensive testing process framework specifically tailored to its particular testing requirements and documenting all aspects of its testing process.

In terms of software project management, BL follows the PRINCE 2 methodology (Reference 3), with the testing process organized and run as a PRINCE 2 program.

It is important to note that the BL view of the organization and management of testing is likely to be more complex than that employed by most organizations engaged in testing. This organizational structure has been adopted because of the size and complexity of BL itself, the number of software development and testing projects and their geographical separation, the strong quality culture, and the adoption of the PRINCE 2 project management method.

Figure 14.1 provides a high-level overview of the organization of the BL software testing program.

The British Library Testing Framework is administered by a PRINCE 2 Testing Project Board, which is responsible for the overall direction of the project management of the testing program, and for representing the User and Supplier interests within individual testing projects.

The Testing Manager reports to the Testing Project Board and is responsible for the day-to-day management of the testing program. The precise responsibilities of the Testing Manager, and the other subordinate roles shown in Figure 14.1 are discussed in detail in Section 14.4.

In terms of the planning of testing, BL employs the *V Model* approach described in Chapter 4. However, because of its particular testing requirements, the BL interpretation of the V Model includes a number of extensions, which are shown in Figure 14.2.

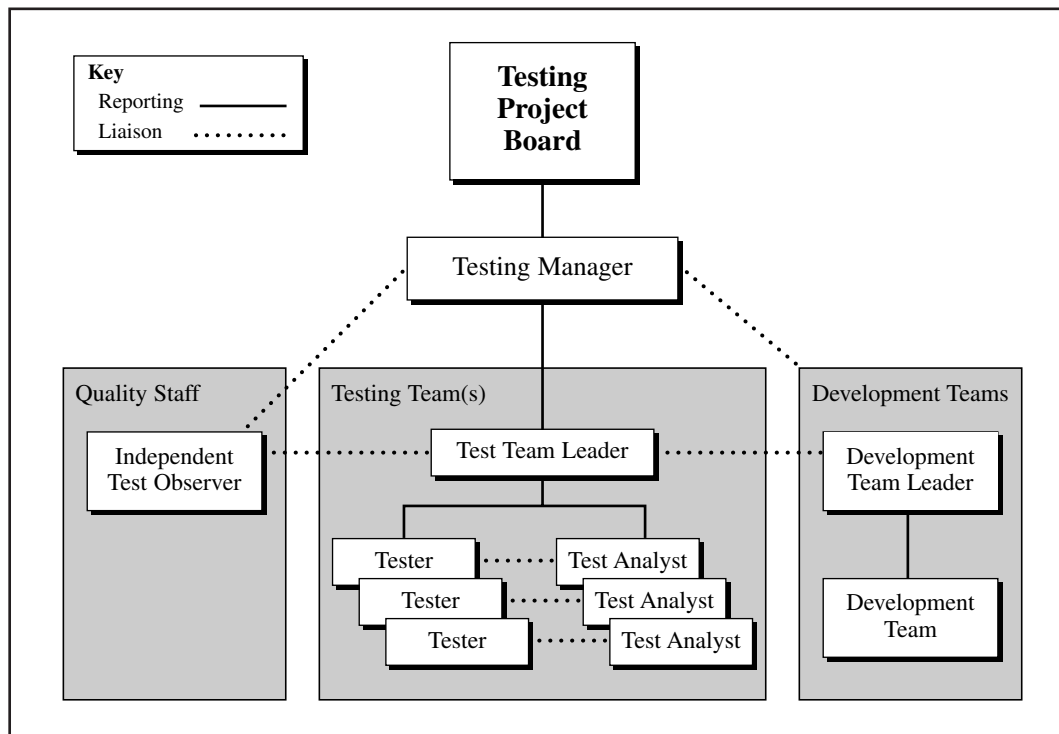


Figure 14.1. British Library software testing program organization.

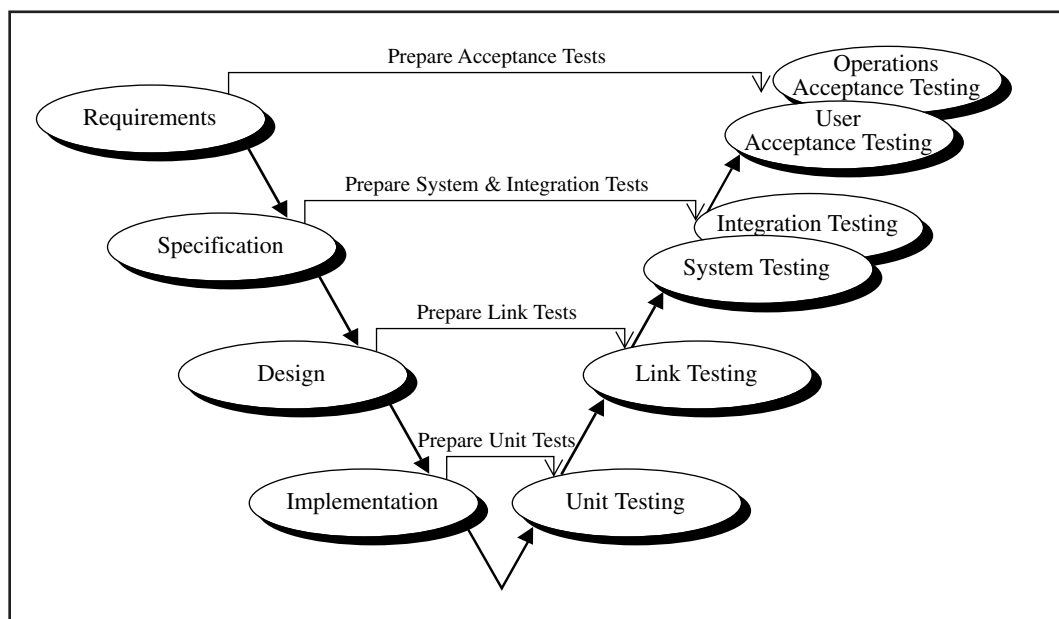


Figure 14.2. British Library extensions to the V Model.

Specifically, the BL term for the traditional Integration Testing phase is *Link Testing*. BL introduces a System Integration Testing Phase between System and Acceptance Testing – termed *Integration Testing*. BL also distinguishes between User and

Operations *Acceptance Testing*. The particular requirements leading to this interpretation of the V Model and the introduction of the additional testing phases are discussed in detail in Section 14.5.

The planning and preparation of tests for use in Integration Testing take place during the Specification phase of software development.

The planning and preparation of tests for use in User and Operations Acceptance Testing take place during the Requirements phase of software development.

In all other respects, the role and use of the V Model within the British Library Testing Framework remain the same as described in Chapter 4.

14.4 Roles and Responsibilities

The testing roles and responsibilities employed in the British Library Testing Framework closely follow those descriptions provided in Chapter 4 and Appendix A of this book. The following BL testing roles and responsibilities are described in this section:

- ▲ *Testing Manager*
- ▲ *Test Team Leader*
- ▲ *Test Analyst*
- ▲ *Tester*
- ▲ *Independent Test Observer*.

It is important to note that within the British Library Testing Framework it is possible for a single person to assume more than one role within a project. The one notable exception to this is the *Independent Test Observer*, who must maintain complete independence from the project and its progress.

It is also possible for a single person to assume different roles on different projects. For example, a Test Analyst on one project may perform the role of Test Team Leader on another.

Testing Manager

Within BL, the Testing Manager is responsible for ensuring that the individual testing projects produce the required products (that is, the outputs from the testing phases) to the required standard of quality and within the specified constraints of time and cost, ensuring that they produce a result capable of achieving the benefits defined in the PRINCE Project Initiation Document.

The Testing Manager reports to the Testing Project Board, is reported to by the Test Team Leader, and liaises with the Independent Test Observer and the *Development Team*.

The specific responsibilities of the Testing Manager include:

- ▲ Setting up and resourcing new testing projects (Project Initiation in PRINCE terms)

- ▲ Ensuring that (where appropriate) the development teams follow the *Unit* and *Link* (the BL term for *Integration*) testing approach documented within the British Library Testing Framework
- ▲ Assuring that where software development is conducted by a third-party organization on behalf of BL, good development and testing practices (for example, following the principles specified in the British Library Testing Framework) have been adhered to
- ▲ Formally representing BL during the co-signing of the Test Certificate (Appendix J) with the representative of a third-party organization responsible for development of software.

Test Team Leader

The Test Team Leader is primarily responsible for the day-to-day running of a testing project. The Test Team Leader reports to the Test Manager, is reported to by the Test Analyst and Tester within the project, and liaises with the Independent Test Observer and (where appropriate) the Development Team Leader.

The specific responsibilities of the Test Team Leader include:

- ▲ Generating the *Test Plan* and *Test Specification Documents* (Appendices C and D, respectively) for the testing project
- ▲ Tasking one or more Test Analysts and monitoring their progress against agreed-on plans
- ▲ Tasking one or more Testers and monitoring their progress against agreed-on plans
- ▲ Liaising with the Independent Test Observer (e.g., to determine his or her availability)
- ▲ Liaising with Development Teams (e.g., to determine availability date of the AUT)
- ▲ Setting up and maintaining of the testing project filing system
- ▲ Reporting testing progress and issues to the Testing Manager at agreed-on intervals.

Test Analyst

The Test Analyst is primarily responsible for the design and implementation of the *Test Cases* and *Test Scripts* within a testing project. The Test Analyst reports to the Test Team Leader and liaises with one or more Testers.

The specific responsibilities of the Test Analyst include:

- ▲ Assisting the Test Team Leader in generating the Test Specification and Test Plan Documents (Appendices C and D, respectively)
- ▲ Defining the Test Requirements (with respect to the Functional Specification for the AUT)
- ▲ Designing and implementing Test Cases and Test Scripts (Appendix E)
- ▲ Designing and implementing Test Data Sets
- ▲ Briefing the Tester for the AUT prior to testing

- ▲ The generation of Re-use Packs (Appendix I)
- ▲ The backup and archival of all testing documentation and materials.

Tester

The Tester is primarily responsible for the execution of the Test Scripts created by the Test Analyst. The Tester reports to the Test Team Leader and liaises with the Test Analyst. The Tester also liaises with the Independent Test Observer during testing.

The specific responsibilities of the Tester include:

- ▲ Executing Test Scripts and observation of test results
- ▲ Identifying and logging-in any observed faults (that is, departures from the expected result)
- ▲ Documentation of test results on *Test Result Record Forms* (Appendix F)
- ▲ Co-signing the Test Result Record Forms with the Independent Test Observer
- ▲ Maintenance and retention of Test Result Record Forms
- ▲ Creation of tests for retest of corrected defects
- ▲ Set-up and initialization of the test bed
- ▲ Backup and archival of the test bed
- ▲ Recovery of the test bed in the event of failure.

Independent Test Observer

The Independent Test Observer is principally responsible for ensuring that the testing process is followed correctly.

The Independent Test Observer reports to the BL Quality Assurance (QA) Manager and liaises with the Test Team Leader. Where the Independent Test Observer witnesses a serious departure from the formal testing process that cannot be resolved with the Test Team Leader, he or she may also directly report this matter to the attention of the QA Manager.

Specifically, the Independent Test Observer is responsible for:

- ▲ Attending the testing of the AUT
- ▲ Formally witnessing that the Tester correctly completes all Test Cases within the Test Script (Appendix E)
- ▲ Co-signing the appropriate section of the Test Result Record Form (Appendix F)
- ▲ Reporting any problems observed during the testing process to the Testing Manager.

BL has a well-established culture of QA in which the notion of peer review and approval is routinely followed. Independent Test Observers are likely to be selected on the basis of their ability to assure that testing projects adhere to the correct testing process.

The Independent Test Observer may also be invited to review the *Test Plan* and *Test Specification Documents* and *Test Scripts* for the testing project they are

involved in. Usually it is sufficient for Independent Test Observers to assure themselves that the correct process has been followed in the development of the documents rather than review the documents in detail.

14.5 Testing Phases

The testing phases supported by the British Library Testing Framework closely follow those described in Part 1 of this book with the following exceptions:

- ▲ BL calls the traditional Integration Testing phase *Link Testing*. The choice of this term was made to distinguish it from the BL testing phase, which tests the integration between separate and distinct BL software systems – termed *Integration Testing* within the British Library Testing Framework. The specific details of the Link Testing phase are in all other ways the same as described in Chapter 6 with the exception that no Test Summary Report (Appendix J) is produced
- ▲ The British Library Testing Framework includes an additional testing phase between System Testing and Acceptance Testing, termed *Integration Testing*. This testing phase is required because of the particular BL requirement that many of its software systems interoperate to a greater or lesser degree. Clearly, where there is no requirement for a particular application to interoperate with other systems, the Integration Testing phase is omitted. Similarly, if the requirement to interoperate with other systems is weak, then the tests normally conducted in Integration Testing may be integrated into the System Testing phase
- ▲ BL makes a clear distinction between User and Operations Acceptance Testing, and typically schedules separate tests for both end users and staff responsible for operating and administering the AUT
- ▲ As a general departure from the testing process described in Part 1, none of the testing projects produces Test Summary Reports as an output from each testing phase. This is because the underlying PRINCE 2 project management method employed by BL already includes comprehensive reporting of the sort that the Test Summary Report provides.

14.6 Artifacts

The artifacts produced within the British Library Testing Framework closely follow those described in Appendices C–I of this book. Specifically, each BL testing project is expected to produce the following standard artifacts:

- ▲ Test Plan document (Appendix C)
- ▲ Test Specification Document (Appendix D)
- ▲ Test Scripts (Appendix E)

- ▲ Test Result Record Forms (Appendix F)
- ▲ Test Log (Appendix G)
- ▲ *Re-use Pack* (Appendix I)
- ▲ And, for the higher levels of testing (that is, System, Integration, and Acceptance testing), and in particular of third-party applications, signed Test Certificates (Appendix H).

As discussed previously in this chapter, BL testing projects do not produce a Test Summary Report (Appendix J).

To ensure that all testing projects have access to the testing artifacts and make use of standard testing documentation, BL provides access to testing artifact templates and checklists via its Intranet system. In this way, any project requiring a particular testing document template can simply download it from a central repository.

14.7 Process Improvement

On completion, testing projects are expected to perform the equivalent of PRINCE 2 Post Implementation Review for the purposes of identifying aspects of the project that were successful and those that could be improved.

It is planned that this information will be provided to testing project staff via the BL Intranet system to ensure that future testing projects can benefit from the experiences of previous projects.

Case Study 2: Reuters Product Acceptance Group

Paul Goddard

15.1 Overview of the Organization

Over 521,000 users in 52,800 locations access Reuters information and news worldwide. Data is provided on over 940,000 shares, bonds, and other financial instruments as well as on 40,000 companies. Financial information is obtained from 260 exchanges and over-the-counter markets and contributed by 5000 clients.

Reuters services are delivered globally over one of the world's most extensive private satellite and cable communications networks. Typically, Reuters updates 6000 prices and items of other data per second and handles 65 million changes daily. The information available from Reuters' databases ranges from real-time to greater than 10 years old.

The Company's two main business divisions are Reuters Information (RI) and Reuters Trading Systems.¹

RI products include real-time information and historical information databases and focus on four main markets – foreign exchange and money, commodities (including energy), fixed income, and equities. Reuters corporate and media information business includes textual news services for print media, broadcast, and on-line clients. The company prides itself on the diverse content, freedom from bias, accuracy, and speed of information to its customers.

Reuters Trading Systems division groups together Reuters management systems, transaction products, risk management products, and other applications.

The Reuters Product Acceptance Group (PAG) operates within the RI division and is based in Singer Street, London. PAG is responsible for the *Acceptance Testing* of financial trading products on behalf of its customers, which is implemented under contract by third-party suppliers.

This case study is a snapshot of PAG's testing activities between October 1997 and May 1998 when Paul Goddard was the UK PAG Manager. Paul now works as Vice President, Database Technology Group, Data Services Division, Toronto, Canada.

¹For current information on Reuters, visit <http://www.reuters.com/about/reuters.background/>.

15.2 Testing Requirements

PAG is principally responsible for the Acceptance Testing of the Reuters 3000 product family.

The Reuters 3000 product family provides integrated access to extensive, high-quality historical reference data and real-time data to support pretrade decision-making and post-trade analysis. Data are presented to the user via a series of functionally related and customizable workspaces, allowing the user to navigate to the required information, to which sophisticated analytics can be applied. The Reuters 3000 product family includes:

- ▲ Reuters Treasury 3000 – which allows users to search, analyze, display, and export bond terms and conditions and price histories. Treasury 3000 is designed to be used by fund managers, portfolio managers, analysts, researchers, brokers, traders, salespeople, and mid- and back-office staff
- ▲ Reuters Securities 3000 – which performs a similar role for the equity market
- ▲ Reuters Money 3000 – which performs a similar role for the real-time money markets.

All of the Reuters 3000 products are expected to interoperate with each other, exchanging data and services, as well as with other Reuters systems such as the 2000-series real-time products and data (IDN), Reuters Graphics, and Reuters Terminal products. The Reuters 3000 products are also expected to interoperate with other *commercial off-the-shelf* (COTS) products, such as spreadsheets and word-processing packages, for purposes of analysis or report writing, for example.

The Reuters 3000 products are implemented under contract to Reuters by a third-party supplier, who is responsible for *Unit, Link, Systems, and Integration testing*² of the *Application Under Test (AUT)* before delivery to PAG. Following successful Acceptance Testing by PAG, the AUT is delivered to the customer site for User Testing prior to live use.

The customers deploy the Reuters 3000 products on many hardware and software platforms with a variety of specifications and configurations. This situation is further complicated by the availability of co-resident software products (such as spreadsheet, word-processing, and/or communications software).

As a consequence of the heterogeneity of customer computer platforms, PAG must ensure that thorough *Interoperability* and *Compatibility Testing* are conducted as part of the Acceptance Testing process. In order to support this requirement, PAG maintains a flexible, configurable test laboratory, which at the time of this case study, comprised some 40 PCs, servers, and networking equipment, plus a wide spectrum of test software. The PAG lab is a secure facility that can be quickly configured to model customer installations in order to perform a variety of realistic Acceptance Testing activities.

²As with many organizations with a strong requirement for testing the integration of a number of interoperating systems, PAG employs a *Systems Integration* testing phase (termed *Integration Testing* within PAG), with the traditional Integration Testing phase being termed *Link Testing*.

Each Reuters 3000 product delivered to PAG must undertake two separate testing streams:

- ▲ Reference Database (RDB) Acceptance Testing – verifying that the data service aspects of the AUT are correct
- ▲ Client-site (or Graphical User Interface [GUI]) Acceptance Testing – verifying that the customer aspects of the AUT are correct.

The testing requirements for each stream are sufficiently different for the RDB and GUI Acceptance Tests to be distinct and separate events conducted by separate testing teams (see Figure 15.1). The principal differences between the RDB and GUI testing teams in terms of skills and experience are:

▲ RDB Testing

- *low-level technical skills, including Client/Server knowledge*
- *thorough knowledge of Reuters information services*
- *thorough knowledge of the Reuters 3000 data requirement*
- *thorough knowledge of the Server operations, support environment, and methodology*

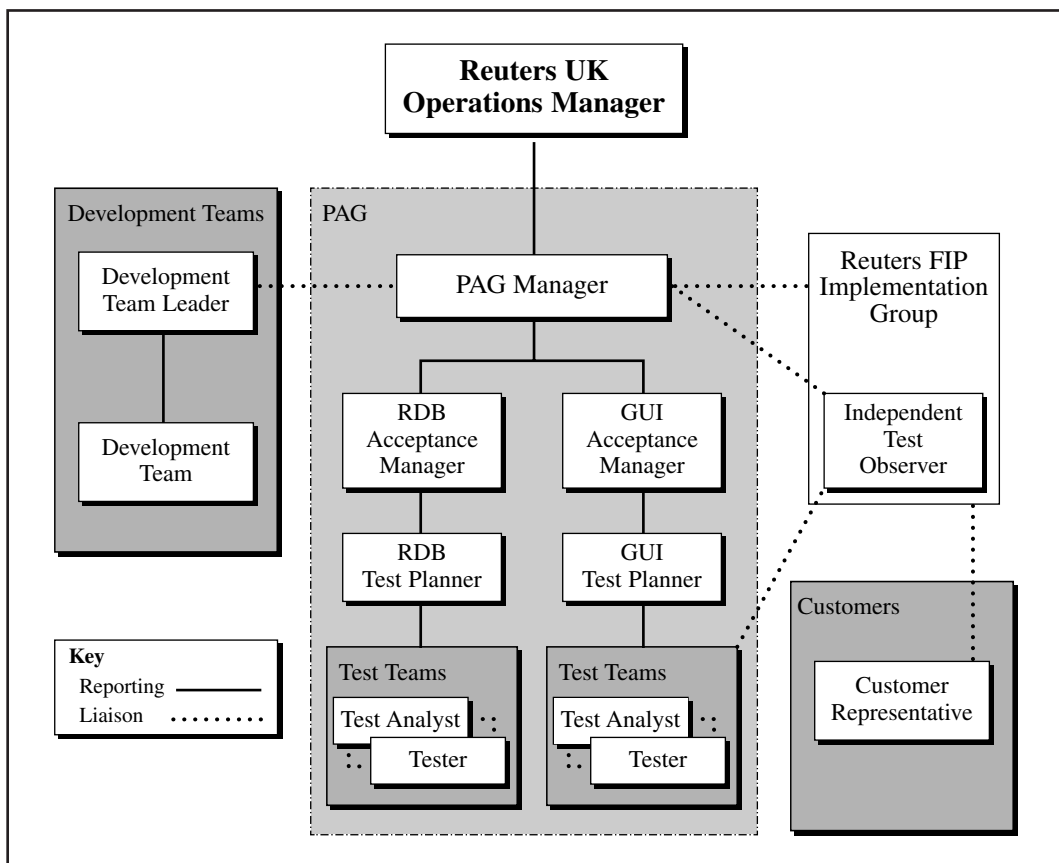


Figure 15.1. Reuters Product Acceptance Group organization.

▲ GUI Testing

- *knowledge of the customer business process*
- *knowledge of the Reuters standards for GUI design*
- *good Usability Testing skills*
- *knowledge of the User Interface design paradigms across the range of Reuters GUI products*

The organization of the RDB and GUI testing streams and their position within PAG are discussed in the following section.

15.3 The Management and Planning of Testing

PAG follows a formal testing process tailored to its particular testing requirements and documenting all aspects of the process. The details of the process are given in the “PAG Methods of Working” document, which is described in Section 15.6.

Figure 15.1 provides a high-level overview of the organization of PAG, its position within Reuters, and its relationships to the third-party developers and Reuters’ customers.

PAG is managed by the PAG Manager, who has responsibility for the day-to-day running of the group. The PAG Manager reports to the UK Operations Manager and is reported to by the managers of the RDB Acceptance Manager and GUI Acceptance Manager.

The RDB Acceptance Manager is responsible for ensuring thorough testing of the AUT to determine if all RDB application changes are ready for operational release. The GUI Acceptance Manager is responsible for testing the AUT from the customer perspective to determine if the AUT is ready for release to the users.

All of the PAG roles shown in Figure 15.1 are described in detail in Section 15.4.

In terms of the planning of testing, PAG employs the *V Model* approach, described in Chapter 4. However, because of its particular testing requirements, the PAG interpretation of the V Model includes a number of extensions, which are shown in Figure 15.2.

As described previously, the third-party supplier is responsible for conducting the early phases of testing, including Unit, Link, Systems, and Integration testing.

Employing V Model principles, the planning and preparation of the Acceptance Test begins within PAG during the Requirements phase of software development and is the responsibility of the respective RDB/GUI Test Planner. The involvement of these roles in planning and preparation for testing is described in detail in Section 15.4.

PAG routinely worked closely with the Product Manager/Business Unit Sponsor to ensure that what was required at the outset of the development was tested and delivered – or if it was not, what compromises along the way were acceptable to the Product Manager. The requirements are recorded in a Product Requirements Specification (PRS) document.

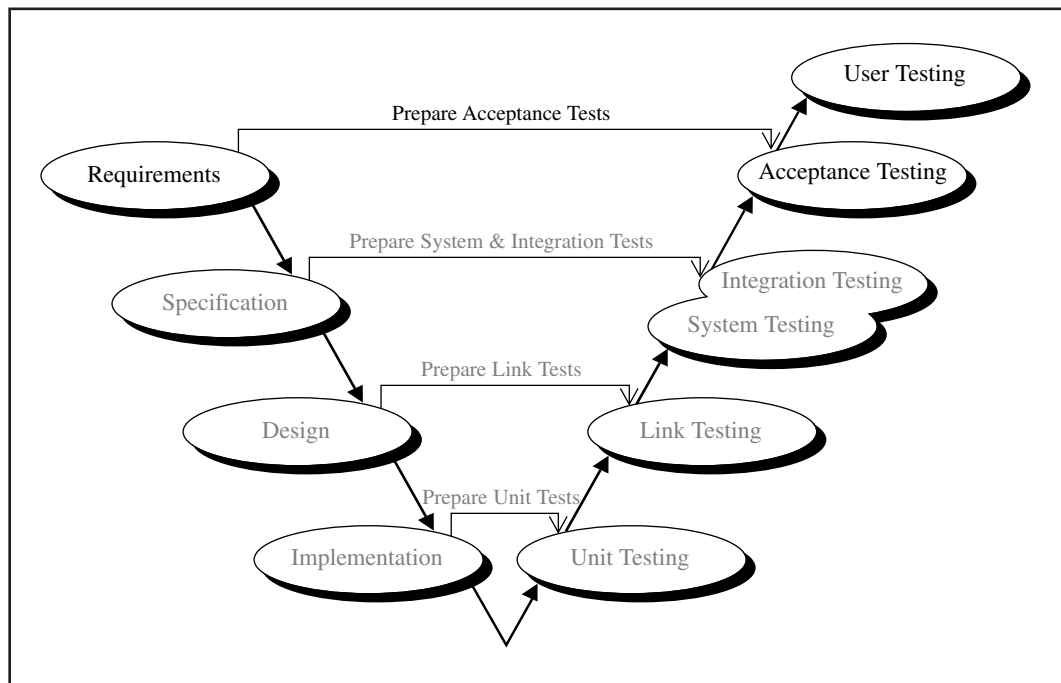


Figure 15.2. Reuters Product Acceptance Group View of the V Model.

Within the PAG Testing Process, re-use of existing testing materials, such as *Test Specification Documentation*, *Test Scripts*, and *Test Logs* is a key technique used to ensure that Acceptance Testing is as efficient and as effective as possible. In order to support this requirement, access to a Web-based document repository was provided for staff members requiring such material.

Although there was no specific librarian role, the RDB or GUI Acceptance Managers had responsibility for administering these facilities, with the project deliverables being reviewed by the PAG Manager.

Although PAG did not employ formal *Re-use Packs*, the Acceptance Managers were encouraged to make tools and documents generated during testing projects available to subsequent projects.

15.4 Roles and Responsibilities

The testing roles and responsibilities employed in PAG include:

- ▲ PAG Manager
- ▲ RDB/GUI Acceptance Manager
- ▲ RDB/GUI *Test Planner*
- ▲ *Test Analyst*
- ▲ *Tester*
- ▲ *Independent Test Observer.*

PAG Manager

The PAG Manager is responsible for ensuring that the individual testing projects produce the required products (that is, the outputs from the testing phases, which include the tested AUT) to the required standard of quality and within the specified constraints of time and cost.

The specific responsibilities of the PAG Manager include:

- ▲ Developing, documenting, and maintaining the PAG Testing Process
- ▲ Setting up and maintaining the PAG filing system
- ▲ Setting up and resourcing new testing projects
- ▲ Tasking the RDB and GUI Acceptance Managers and monitoring their progress against agreed-on plans
- ▲ Monitoring the quality of Acceptance Testing against the documented PAG Testing Process.

The PAG Manager reports to the UK Operations Manager, is reported to by the RDB and GUI Acceptance Managers, and liaises with the *Development Team Leader* to agree on delivery dates for the AUT and to verify that correct procedures have been employed in the development and testing of the AUT. The PAG Manager also liaises with the Financial Information Products (FIP) Implementation Group to deliver the Acceptance Tested AUT for subsequent installation and User Testing.

RDB/GUI Acceptance Managers

The RDB and GUI Acceptance Managers are responsible for determining if the AUT is ready for formal release to the customer.

The RDB Acceptance Manager has specific responsibility for verifying that the Reference Database aspects of the AUT (that is, all areas of the AUT's interaction with the data source for the Reuters 3000 products) are correct. This includes aspects of supportability and operability.

The GUI Acceptance Manager has specific responsibility for verifying that the customer aspects of the AUT (that is, testing the AUT from a user perspective) are correct. This includes verifying functional requirements, interoperability, and usability.

The specific responsibilities of the Acceptance Managers include:

- ▲ Working in conjunction with the Test Planner to develop the Acceptance Test Plan for a given testing project
- ▲ Working in conjunction with the Testing Team to develop the Test Specification Document for a given testing project
- ▲ Ensuring that the Test Planner and Test Analysts are aware of the need to consider re-use of existing testing materials
- ▲ Tasking the appropriate Test Planner and monitoring his or her progress against agreed-on plans
- ▲ Liaising with the Independent Test Observer (for example, to determine their availability)

- ▲ Setting up and maintaining the filing system for each testing project
- ▲ Reporting testing progress and issues to the PAG Manager at agreed-on intervals.

The Acceptance Managers report to the PAG Manager and are reported to by the respective Test Planners, Test Analysts, and Testers.

RDB/GUI Test Planner

The Test Planner role incorporates a number of the responsibilities of the Test Team Leader in a traditional testing project. The specific responsibilities of the Test Planner include:

- ▲ Developing the Acceptance Test Plan document in conjunction with the Acceptance Manager (the RDB and GUI Test Planners must liaise with their counterparts to ensure no contention for PAG Lab facilities occurs)
- ▲ Assisting the respective RDB/GUI Acceptance Manager in generating the Acceptance Test Specification Document
- ▲ Tasking the Test Analyst and Tester and monitoring their progress against agreed-on plans
- ▲ Reporting testing progress to the Acceptance Manager at agreed-on intervals
- ▲ Wherever appropriate, considering re-use of existing testing materials (such as Acceptance Test Plans and Test Specification Documents).

The Test Planner reports to the respective Acceptance Manager and is reported to by the Test Analyst and the Tester.

Test Analyst

The Test Analyst is primarily responsible for the design and implementation of the *Test Cases* and Test Scripts within a testing project.

The specific responsibilities of the Test Analyst include:

- ▲ Performing the Functional Analysis of the AUT and defining the Test Requirements (with respect to the Product Requirements Specification and the Project Functional Specification)
- ▲ Assisting the respective RDB/GUI Acceptance Manager and Test Planner in generating the Acceptance Test Specification Document
- ▲ Designing and implementing Test Scripts and Test Cases
- ▲ Briefing the Tester for the AUT prior to testing
- ▲ Wherever appropriate, considering re-use of existing testing materials (such as Acceptance Test Scripts and Test Cases)
- ▲ Backup and archival of all testing documentation and materials.

The Test Analyst reports to the respective Test Planner and liaises with the Tester.

Tester

The Tester is principally responsible for the execution of the Test Scripts created by the Test Analyst. The specific responsibilities of the Tester include:

- ▲ Executing Test Scripts and observation of test results
- ▲ Identifying and logging-in any observed faults (that is, departures from the expected result)
- ▲ Documentation of test results on *Test Result Record Forms*
- ▲ Maintenance and retention of Test Result Record forms
- ▲ Modification of tests for retest of fault correction
- ▲ Set-up and initialization of the test bed within the PAG Lab for each user configuration to be tested
- ▲ Recovery of the PAG Lab test bed in the event of failure.

The Tester reports to the respective Test Planner and liaises with the Test Analyst.

Independent Test Observer

The Independent Test Observer is principally responsible for ensuring that the testing process is followed correctly. Independent Test Observers are drawn from the Product Managers of RDB Operations Managers.

The Independent Test Observer reports to FIP or Operations and liaises with the Test Teams. Where the Independent Test Observer witnesses a serious departure from the formal testing process that cannot be resolved with the Test Team Leader, he or she may also directly report this matter to the attention of the PAG Manager.

Specifically, the Independent Test Observer is responsible for:

- ▲ Attending the testing of the AUT
- ▲ Formally witnessing that the Tester correctly completes all Test Cases
- ▲ Signing the appropriate section of the Test Result Record Form
- ▲ Reporting any problems observed during the testing process to the PAG Manager.

Independent Test Observers may also be invited to review the Acceptance Test Plan and Acceptance Test Specification Documents and Test Scripts for the testing project they are involved in. Usually it is sufficient for Independent Test Observers to assure themselves that the correct process has been followed in the development of the documents rather than review the documents in detail.

15.5 Testing Phases

Unit, Link, Systems, and Integration testing of the Reuters 3000 products is conducted by the developers on development computers (NCR 5100 series machines) using a representative copy of the *live data*. A copy of the Reuters 3000 data is

used to ensure that testing of the pre-Acceptance Test ready software does not corrupt the live data.

PAG is responsible for the Acceptance Testing of new members of the Reuters 3000 product range as well as Acceptance Testing of new versions/releases of the current Reuters 3000 products. In the latter case, PAG also needs to conduct rigorous *Regression Testing* of the AUT to ensure that the new functionality has not impacted on the correct performance of the existing functionality.

Acceptance Testing is conducted in the PAG lab using a range of configurations of hardware and software, which are set up to be representative of the various customer installations. Acceptance Testing is conducted against the live Reuters 3000 data source to provide further confidence in the accuracy of the results of testing.

Following successful Acceptance Testing, the AUT is delivered to the customer, where User Testing is conducted in the *live environment* using live data.

Figure 15.3 provides an overview of the Testing Phases, their location, and the data requirements.

Unit, Link, Systems, and Integration Testing

The developers are responsible for conducting adequate Unit, Link (the developer term for the traditional *Integration Testing* phase), System, and Integration (the developer term for *Systems Integration*) Testing. System and Integration testing take place in a single combined testing phase.

The data used for the developer-managed testing phases are a copy of the live data used by the Reuters 3000 products and supplied to the developer by PAG to ensure realistic and representative testing. The developer may augment the live

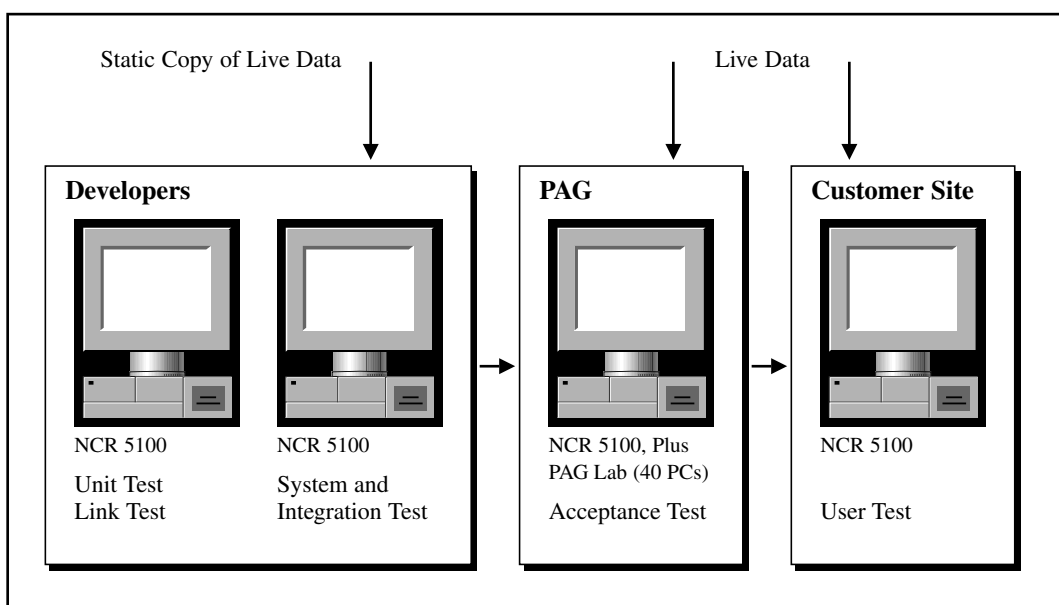


Figure 15.3. Summary of Reuters 3000 Testing Phases.

data using handcrafted data for the purposes of testing boundary and error conditions; any such data will be generated using formal design techniques (such as those described in Chapter 3).

Following successful System and Integration Testing, the AUT is delivered to PAG for Acceptance Testing. PAG is responsible for verifying that the developer has followed correct procedures during the development and testing of the AUT.

Acceptance Testing

The principal reason PAG performs Acceptance Testing of the Reuters 3000 products is to provide confidence that the products are of suitable quality for delivery to Reuters 3000 customers. Specific aspects of the products that are tested include:

- ▲ Installation testing
- ▲ Product impact – that is, Regression Testing to ensure the user continues to see the same level of functionality/performance/quality for all elements of the product that have not specifically been changed
- ▲ Interoperability testing
- ▲ Operability testing – that is, can the Operations group manage day-to-day monitoring and support? For example, do the error messages/information messages make sense – do they fit with existing Operations norms/requirements?
- ▲ Supportability – that is, can the development group support the product over the long term? For example, does it meet standards for naming/DBA conventions, and does it use standard interfaces/common functions?
- ▲ Usability testing
- ▲ Benchmarking – that is, performance testing (including performance results monitored over time to observe trends).

Each Reuters 3000 product that undergoes Acceptance Testing by PAG does so under two separate and distinct testing streams: RDB Acceptance Testing (verifying that the Reference Database aspects of the AUT are correct) and GUI Acceptance Testing (verifying that the customer aspects of the AUT are correct). These testing projects are conducted by two separate testing teams, each possessing skills and experience appropriate to the particular aspect of the AUT they are testing.

Since both RDB and GUI testing for a particular AUT take place in the PAG lab, adequate planning of the tests is required to avoid contention for the testing facilities.

The Acceptance Test Scripts and Test Cases are designed by the Test Analyst based on the information provided in the PRS and PFS documents (see Section 15.6) plus any supplementary design material (such as prototype software or user guide documentation).

The Test Scripts are executed by the Tester, who records the results of the Test Cases and documents any observed errors. Testing takes place in the PAG lab using a number of different arrangements of the lab facilities to accurately represent the possible configurations that different customers may use in their live environments. Similarly, live Reuters 3000 data is used in Acceptance Testing to pro-

vide confidence that the test results will match those expected on the live customer environment.

Following analysis of the Test Result Record Forms, any errors are logged in using a commercial defect-tracking system (Reference 21 provides an example of such a system) and are reported to the developers to correct within mutually agreed-on timescales. The Tester is also responsible for retesting the AUT to verify that the defects have been corrected and to perform adequate Regression Testing to ensure that the existing functionality of the AUT has not been compromised during correction of the defects.

At the successful completion of Acceptance Testing, the AUT is released to FIP for subsequent delivery to the customer and User Testing.

Regression Testing

Because of the business critical nature of the Reuters 3000 products, it was determined that new releases of the products (such as enhancement and defect-correction releases) would undergo thorough and rigorous Regression Testing to provide maximum confidence in the quality of the software.

In order to save time and effort expended in Regression Testing and to ensure that a consistent and thorough approach is maintained, the PAG Methods of Working document proposes that Regression Testing be made as effective and as efficient as possible by employing re-use of existing testing materials wherever possible (Appendix I provides further advice on exploiting re-use in testing projects).

At the conclusion of the Acceptance Test (and following a new release of the AUT), the Tester is responsible for making copies of the Test Scripts and Test Cases and any other relevant testing materials (such as the test data) and placing them into a Re-use Pack, which is then filed for potential re-use during the next Acceptance Test.

At the planning stage of the next Acceptance Test, the RDB/GUI Acceptance Manager, Test Planner and Test Analyst will review the existing Re-use Packs in order to identify appropriate testing materials that can be re-used in the current testing phase.

Apart from the obvious benefits in terms of saved time and effort, the testing process also benefits from the re-use of “tried and trusted” testing materials, which the testing staff has confidence in using.

User Testing

User Testing was conducted by FIP Product/Project managers, who are responsible for Alpha and Beta testing. Output from these tests was typically used as input to the requirements for the next release.

15.6 Artifacts

PAG employs a comprehensive set of testing documents within its testing process. Standard document templates are available for PAG staff to re-use in order to maintain the quality and consistency of the testing process and to save time, effort, and cost in the generation of testing documents.

The testing artifacts employed within the PAG testing process include:

- ▲ The Product Requirements Specification (PRS), which is drafted by FIP and subsequently amended by PAG to produce a generic PRS to represent the overall User/Operations/Product requirements for testing purposes
- ▲ PFS Project Plan
- ▲ The PAG Methods of Working document – a high-level policy document describing the philosophy and approach to testing the Reuters 3000 products, in effect providing the description of the PAG testing process. The Methods of Working document contains the following sections:
 - *a statement of the scope of testing to be performed by PAG*
 - *a statement of the approaches and processes to be used in testing*
 - *the Entry criteria to be fulfilled before AUTs will be received by PAG*
 - *the Exit criteria to be fulfilled before AUTs can leave PAG*
 - *a definition of the various test categories used in Acceptance Testing*
 - *standard templates to support the testing process, such as:*
 - ☐ Test Logs
 - ☐ Test Scripts
 - ☐ Test Result Record Forms.
- ▲ PAG Acceptance Test Plan documents, with sections providing:
 - *the project plan and schedule for testing the AUT*
 - *the scope of testing (together with any exclusions)*
 - *a description of the testing approach to be used in testing the AUT*
 - *details of any risks, constraints, and assumptions associated with testing the AUT*
 - *specific Entry and Exit criteria for the AUT*
 - *configuration details for the AUT, test data, and test hardware*
 - *any specific project controls to be used in testing the AUT.*
- ▲ PAG Acceptance Test Specification documents – with sections providing:
 - *a description of what is to be tested and what is not to be tested*
 - *for each test:*
 - ☐ a high-level description of the purpose of that test
 - ☐ a high-level description of how the test should be performed
 - ☐ the test pre-requisites that must be satisfied
 - *a specification of any specific tools required for testing the AUT*
 - *a specification of the platform and configurations to be used in testing the AUT*
 - *a specification of the test environment for the AUT.*

- ▲ Re-use Packs, to contain:
 - *a copy of the Test Specification document*
 - *a copy of the Test Log*
 - *a copy of the Test Script (and any revised Test Cases)*
 - *a floppy disk of the appropriate format containing the above*
 - *a floppy disk copy of any handcrafted data (if technically feasible), or a reference to an electronically archived copy of the data.*
- ▲ Test Logs – as specified within and available from the PAG Methods of Working document
- ▲ Test Scripts – as specified within and available from the PAG Methods of Working document
- ▲ Test Result Record Forms – as specified within and available from the PAG Methods of Working document
- ▲ A Testing Summary Report delivered to FIP or RDB Operations – this could also include the Deficient Management Plan (required for all releases) that documented what was planned in terms of the acceptable deficiencies and specified how they would (or would not) be addressed in future – this was signed off by the Product manager

15.7 Process Improvement

PAG reviewed and rejected the ISO 9000 scheme, but adopted elements of it in the PAG Methods of Working document.

In terms of process improvement, Reuters development federation has adopted the Capability Maturity Model (CMM, Reference 35). Most of what was the Corporate Technology Group (CTG) is now formally certified at level 2 (including the Data Services Division in Toronto). A number of groups have attained level 3, which is the current global goal by 2001.

Case Study 3: Crown Quality Assurance Group

James Toon, Crown QA Team Manager

16.1 Overview of the Organization

Crown Management Systems Ltd (Crown) was established in 1982 to provide software solutions to the hospitality industry.

The Company has developed several generations of software products, of which InnMaster, InnTrader, and InnTouch are the most current. These products have been designed to provide a business solution to multiple retailers in the hospitality sector.

Crown's services have been developed over the years so that a full turnkey solution can be offered to new clients, in conjunction with its business partners, including systems, hardware, services, support, and maintenance.

Crown's services also include the provision of outsourcing services. For example, it runs and manages Head Office Information Technology (IT) systems for a number of clients. Crown also provides site deployment and training outsourcing for larger clients with hundreds of sites. For more information visit Crown's Web site: www.crown.co.uk.

Since 1997 Crown has been involved in a European Union (EU) project intended to investigate the benefits of using automated testing tools in software development and testing. The project, sponsored by the EU as part of the European Software Systems Initiative (Reference 27), uses Rational Software's TeamTest tool (Reference 22) with the objective of quantifying the amount of time and effort that can be saved during the testing phase of the software development process. Rational TeamTest was selected over other competitive products because of its wide platform support (including Crown's platforms of choice, such as Delphi and SQL Server), its reputation as a world leader in software development and testing, and the broad range of functionality and integration it offers, above and beyond that of its competitors.

James Toon is the *Quality Assurance (QA)* Team Manager for Crown, and is responsible for all aspects of Crown's testing process.

16.2 Testing Requirements

Crown trades in a highly competitive market, with a number of other companies offering similar products and services. In order to maintain its market share, Crown constantly responds to requests for modifications to existing features and for the introduction of new functionality to the software systems it distributes. Thorough testing and the associated increase in the quality of the products Crown delivers to its customers are also perceived as key to Crown's continued business success.

This case study focuses on the testing activities of the Crown QA Team, which is responsible for performing the *System* and *Acceptance testing* of all software products that Crown delivers to its customers.

The need to test the latest functionality of new releases of the software and to ensure the existing functionality is unaltered, combined with the commercial cycle of changes to the underlying hardware platforms and operating systems, means that Crown's products must undergo frequent and thorough *Regression Testing*. As a consequence, it is key to Crown's commercial success that its software development and testing process is as effective and as efficient as possible.

Adoption of a formal testing process has provided Crown with the means of achieving this goal, allowing different testing projects to follow a common testing process with well-defined testing roles and responsibilities, standard templates, and documentation, with the opportunity to improve efficiency through re-use.

In addition to adopting a formal approach to testing, the Crown testing process also incorporates the use of automated software testing tools, with the aim of reducing the time, effort, and cost of testing and improving the quality of the delivered systems.

Crown selected its testing-tool solution through a rigorous review and evaluation of the testing-tool market and the testing products that were available (see Appendix N for an illustrative testing-tool evaluation scheme and criteria). The tool selected by Crown was the Rational TeamTest product, which is being used successfully on a number of Crown testing projects. In particular, Crown believes that test automation will:

- ▲ Reduce project testing time from 60% to 35% of the project duration
- ▲ Reduce the number of errors detected in the later phases of testing by 30%
- ▲ Lead to an increase in employee motivation through the automation of the repetitive elements of testing
- ▲ Reduce project delivery times by 20%
- ▲ Increase the reliability and robustness of the *Applications Under Test* (AUTs).

In selecting an automated software testing-tool solution, Crown has carefully managed the rollout and adoption of the tool. Specifically, Crown planned and managed the acquisition of the tool, ensuring that sufficient funds were available for on-site training and mentoring from Rational Software technical staff, as well as from expert process consultancy for the purposes of integrating the use of the tool into its existing testing method.

Testing automation plays a major role in supporting Crown's System and Regression testing requirements, while Acceptance Testing follows a more traditional manual approach.

Crown supplies a number of different customers with Point of Sale systems, each with slightly different hardware and software requirements. As a consequence, Crown has a challenging requirement for accurately simulating the *live environments* of each of its customers to ensure that the test environment accurately reflects the conditions under which the live system will operate. Because of this requirement Crown has developed a well-equipped and flexible *Test Rig* facility that allows it to simulate the live environments of its customers for the purposes of System Testing. Acceptance Testing typically takes place on the live customer system.

16.3 The Management and Planning of Testing

Crown follows a comprehensive testing process that is specifically tailored to its particular testing requirements and that documents all aspects of its testing process. The process incorporates the role and use of automated software testing tools that are employed at appropriate points in the process to make testing as effective and as efficient as possible.

Crown has a strong quality culture, and the individual testing teams are administered under the auspices of the QA Team. This contrasts with those organizations (such as British Library – Chapter 14) where the testing projects are administered within an IT group.

The Crown approach to managing its testing projects is heavily influenced by the PRINCE project management methodology (Reference 3), and many of the principles of this method are employed in the supervision of the QA Team activities. The Crown QA Team is managed on a day-to-day basis by the QA Team Manager, who reports to the Crown Development Director.

Figure 16.1 provides a high-level overview of the organization of the Crown QA Team and its relationships to the other entities involved in the testing process.

The precise responsibilities of the QA Team Manager and the other subordinate roles shown in Figure 16.1 are discussed in detail in Section 16.4.

In terms of the planning of testing, Crown follows the *V Model* approach described in Chapter 4. However, because of its particular testing requirements, the Crown view of the V Model is biased toward the higher levels of testing, as shown in Figure 16.2.

At the Requirements phase the QA group is involved in reviewing the Business Specification document (BSD) for the AUT. This document forms the basis of the planning, design, and resourcing of the Acceptance Testing phase. The document is reviewed to identify any omissions or duplication of requirements, as well as any requirements that are specified in such a manner that they will be difficult to verify during testing, allowing for subsequent clarification or restatement.

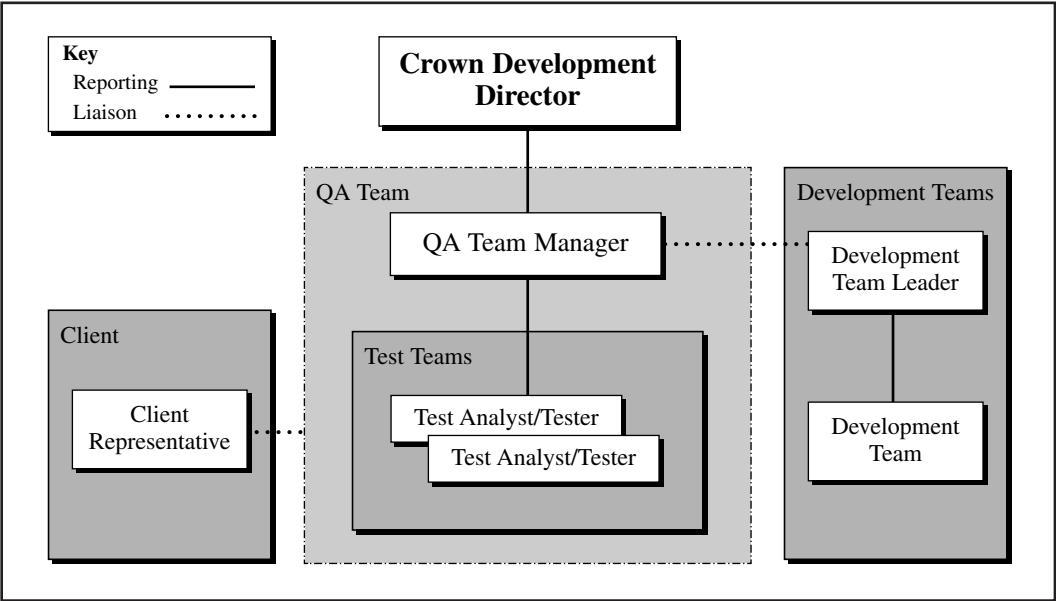


Figure 16.1. Crown QA Team organization.

A *Functional Specification* document (FSD) is produced by the developers, describing how the development work will be carried out. In the FSD, the QA Team includes a section on how the application will be tested and how the customer will know that the development has been a success. A Client Representative will formally review the FSD prior to development commencing.

When the FSD is agreed on, the *Test Scripts* are created in preparation for System Testing. Care is taken in the design of these scripts, since they will also be used for Regression Testing of later versions of the AUT.

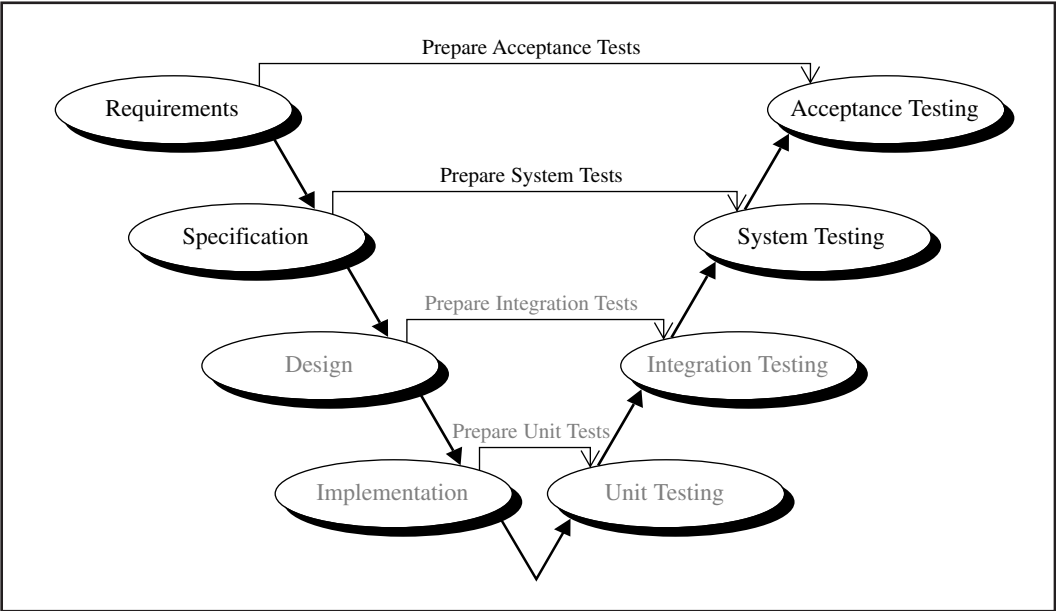


Figure 16.2. Crown Testing Group view of the V Model.

The Testing Team will accept only the application for System Testing when all defects raised during Unit and *Integration Testing* of the AUT by the developers have been cleared and it can be installed on a specially configured test computer from a delivery CD-ROM. In addition to testing the new functionality of the AUT, thorough and rigorous Regression Testing is performed using the automated testing tool.

For minor releases (such as defect fixes) a technique Crown terms “Crash Testing” is used to verify overall quality of the AUT before allowing the software to proceed to more complete testing on the Test Rig. This approach reduces the problems of scheduling a test, setting up the Test Rig, and installing the test data, only to find that the AUT is not of sufficiently good quality to test. In practice this approach reduces wasted time and effort, making the testing process more effective and efficient.

When System Testing of a major release is complete, Crown carries out the User Acceptance Test at the client site. With upgrade releases, Crown performs the Acceptance Test internally on behalf of the client.

16.4 Roles and Responsibilities

The roles and responsibilities defined for the Crown QA Team directly reflect the nature of the testing requirement for the organization. The relatively small number of staff involved in the testing process has a number of implications for overlap of responsibilities, with individual members of staff performing more than one role within the project. In effect, the QA Team Manager role also incorporates most of the traditional *Test Team Leader* responsibilities, and the *Test Analyst* and *Tester* roles are combined into a single function.

The QA Group testing teams also work closely with the developers of the AUT, hence the inclusion of the Developer role in this section (in this sense, Developers could refer to both internal development teams as well as any third-party organization supplying software to Crown). Similarly, the Client plays a close part in the Acceptance Testing of the AUT, with a Client Representative participating in the “independent” observation of testing.

The following testing roles and responsibilities are described in the following sections:

- ▲ QA Team Manager
- ▲ Test Analyst/Tester
- ▲ Developer Representative
- ▲ Client Representative.

QA Team Manager

Within Crown, the QA Team Manager is responsible for ensuring that the individual testing projects produce the required products (that is, the outputs from the testing phases, which include the tested AUT) to the required standard of quality

and within the specified constraints of time and cost. This is achieved by means of the PRINCE project management approach (Reference 3).

The Testing Manager reports to the Crown Development Director, who is reported to by the Test Analyst/Testers and liaises with the Developer and Client Representatives.

The specific responsibilities of the QA Team Manager include:

- ▲ Developing test strategy and test planning
- ▲ Set-up and maintenance of the QA Team project filing system
- ▲ Resource Planning and allocation of testing projects
- ▲ Writing the Test Strategy section of the Functional Specification document
- ▲ Ensuring that each testing project produces a result that is capable of achieving the benefits defined in the Functional Specification document (see Section 16.6)
- ▲ Tasking one or more Test Analysts/Testers and monitoring their progress against agreed-on plans
- ▲ Assuring that where development is conducted by a third party on behalf of Crown, that good development and testing practice (such as the principles specified in the Crown Testing Process) have been adhered to
- ▲ Reporting testing progress and issues to the Crown Development Director at agreed-on intervals
- ▲ Process Improvement and QA Metrics measurement and reporting.

Test Analyst/Tester

The Test Analyst/Tester is principally responsible for the design and implementation of the Test Cases within a testing project and reports to the QA Team Manager.

The specific responsibilities of the Test Analyst/Tester include:

- ▲ Assisting the QA Team Manager in generating the test strategy material for the FSD and the test plans
- ▲ Defining the Test Requirements (with respect to the FSD)
- ▲ Performing the functional analysis of the AUT for the purposes of the design of Test Scripts and *Test Cases*
- ▲ Designing and implementing the Test Scripts and Test Cases/Verification Points
- ▲ The design and specification of the Test Rig and Test Data Sets
- ▲ Implementing the Test Rig and Test Data Sets prior to testing
- ▲ The set-up and initialization of the Test Rig and *Test Data* and recovery of the Test Rig and data in the event of failure during testing
- ▲ Backup and archival of the Test Rig specification and Test Data sets
- ▲ Executing Test Scripts and observing of test results
- ▲ Documenting test results on *Test Result Record Forms* and their subsequent maintenance
- ▲ Identification and logging-in of any observed faults (that is, departures from the expected result)
- ▲ Design and implementation of tests for retest of fault fixes (or re-use of existing tests if appropriate)

- ▲ The generation of *Re-use Packs* for each testing phase
- ▲ Backup and archival of all testing documentation and materials.

Developer Representative

Because of the close working arrangement established by Crown with respect to the developers of the AUT, the role of Developer Representative is formally defined within the Crown testing process.

Specifically, the Developer Representative is responsible for:

- ▲ Providing the AUT specifications (that is, the Business Specification and Functional Specification documents, plus any supplementary design material)
- ▲ Providing build information on each delivered build/release of the AUT
- ▲ Providing confirmation of the correct execution of Unit and Integration Testing
- ▲ Providing supporting information regarding AUT data requirements
- ▲ Providing baseline information for setting up the Test Rig, such as Operating System details or the need for supporting or interoperating software
- ▲ Providing formal sign-off of the delivered AUT.

This formal definition of the developer responsibilities with regard to the testing process forms an effective contract ensuring the correct and timely delivery of the System Test-ready AUT and supporting information required for testing. Such an approach is highly recommended for organizations that receive test-ready software developed by a third-party organization for the purposes of System and/or Acceptance Testing.

Client Representative

The Crown QA Team works closely with clients to ensure that the delivered point of sale systems are of appropriate quality, and specifically, that they meet their functional requirements, are reliable and robust, and provide adequate performance.

As part of this close relationship, the role of Client Representative is formally defined within the Crown testing process. The role carries an unusual mix of traditional *User Representative* responsibilities (such as formal acceptance of the AUT), as well as some of the responsibilities of an “Independent” Test Observer (such as formally witnessing the testing process to ensure that correct procedures are followed).

Specifically, the Client Representative is responsible for:

- ▲ Formally attending the Acceptance Testing of the AUT (typically, at the client site)
- ▲ Formally witnessing that the Test Analyst/Tester correctly completes all Test Cases
- ▲ Signing the appropriate section of the Test Result Record Form

- ▲ Co-signing the *Acceptance Test Certificate* at the completion of a satisfactory Acceptance Test (or following an unsatisfactory Acceptance Test, after all outstanding defects in the AUT have been demonstrated to be corrected)
- ▲ Reporting any problems observed during the Acceptance Testing process to the QA Team Manager.

The Client Representative is also likely to be involved in the day-to-day monitoring of the progress of the software development and testing process with the QA Team Manager.

16.5 Testing Phases

As described earlier in this chapter, Crown obtains the test-ready software from developers who will have completed Unit and Integration testing on the AUT prior to delivery to Crown's QA Group on CD-ROM.

The AUT undergoes a cycle of enhancement and release in response to change requests from the applications customers as well as the need to periodically update the software (to take account of hardware or operating system changes for example).

The release schedule is demanding, with a major release twice a year and minor releases every quarter. Combined with the need to ensure the AUT will perform correctly on each of the customer sites, the QA Team faces a challenging testing load. In order to address the heavy testing load as well as to conduct rigorous Regression Testing, Crown employs an automated testing solution.

Each System Test involves manual testing of the new functionality for the AUT as well as Regression Testing to ensure the existing functionality has not been compromised. As the manual tests are run, the automated testing tool is used to create scripts by recording the steps and verifications employed in the test. These scripts can subsequently be replayed against later releases of the AUT for the purposes of Regression Testing and can even be run in an unattended manner overnight or over the weekend.

Crown is responsible for the following testing phases:

- ▲ System Testing
- ▲ Acceptance Testing
- ▲ *Crash Testing*
- ▲ Regression Testing.

The specific interpretation of each of these testing phases as a result of Crown's particular testing requirements is described in the following sections.

System Testing

Crown conducts System Testing of the delivered software to ensure it is fit for purpose and of suitable quality to pass its Acceptance Test. System Testing is conducted in-house by Crown's QA Team using its Test Rig facilities to simulate the customer's live environment.

The System Tests are designed by the Test Analyst/Tester with reference to the Business Specification and Functional Specification documents, as well as any other supplementary design information. The automated testing tool is used to record the System Tests as they are executed. In this way, a library of previously recorded tests is assembled, from which selected automated tests can be compiled into a larger Regression Testing suite and executed. The recorded tests can also be used at a later date to reproduce the results of running a particular System Test.

System Testing also incorporates any *Systems Integration Testing* that the AUT needs to undergo. This is a pragmatic testing solution to the problem of the complexity of the Test Rig that is used in System Testing, since it would be very time consuming to have to reconstitute an identical Test Rig for a subsequent testing phase.

The data used for System Testing is a copy of that used on the live system, extended by the introduction of handcrafted data for the purposes of boundary testing and simulating error conditions. The design of such data is based upon the testing techniques (such as boundary analysis and equivalence partition methods) described in Chapter 3.

On completion of System Testing (that is, when the Exit Criteria specified in the FSD have been satisfied), the AUT is ready to proceed to Acceptance Testing.

Acceptance Testing

Following successful System Testing, the AUT will undergo Acceptance Testing at the customer site as part of the formal process of delivering the application to the customer (minor releases of the AUT will be Acceptance Tested on behalf of the customer by Crown at its offices).

Although an automated testing tool has been used during System Testing and for Regression Testing, the Acceptance Test is conducted using manual testing techniques.¹ The Client Representative performs *Thread Testing* of the AUT using scenarios designed with reference to the BSD and in the presence of one or more members of the QA Team.

Crown does not run separate User and Operations Acceptance Testing but incorporates elements of both into a single Acceptance Test. This is a pragmatic solution to the problem of convening and setting up separate User and Operations Acceptance Tests at the customer site. In practice, running a single Acceptance Test has been shown to be both effective and efficient.

On the successful completion of the Acceptance Test, the Client Representative and a representative of the Crown QA Team co-sign an Acceptance Certificate (see Appendix H, for example) to show formal acceptance of the AUT.

¹Crown has considered the benefits of its customers using automation for Acceptance Testing – with the customer in effect rerunning the automated tests on the live environment. At present, this solution has not been adopted because of the difficulties in convincing each of Crown's customers to invest in the automated technology.

In the event that defects are observed during testing that prevent the AUT from being accepted, the Client Representative and QA Team representative discuss and agree on timescales for remedial work on the AUT and set a maximum period within which the Acceptance Test will be repeated.

Crash Testing

For the minor releases of the AUT (that is, for releases incorporating minor enhancements or bug fixes), the Crown QA Group performs a testing technique termed “Crash Testing.” In effect, the software is loaded from the delivery CD-ROM onto a standalone PC for the purposes of conducting a series of high-level tests to verify that the release is of sufficient quality for it to proceed with System Testing.

In practice, this approach has saved Crown significant time and effort in terms of setting up the Test Rig and Test Data only to discover that the AUT was not really ready for testing (Crown estimates that Crash Testing can save enough effort to allow up to 33% additional testing time for the AUT, providing further confidence in the quality of the application).

Regression Testing

Regression Testing of the AUT is conducted using the Rational TeamTest automated testing tool. As described previously, the Test Analyst/Tester designs a series of Test Scripts to verify the additional functionality incorporated into a new release of the AUT. As the Tester executes these Test Scripts, the testing tool is used to record the steps involved in the test as well as any Verification Points.

In subsequent System Tests, these recorded tests can be re-run against the latest release of the AUT in order to ensure that the existing functionality of the AUT has not been compromised by the new enhancements/modifications.

Typically, Crown runs these Regression Tests unattended (either overnight or over the weekend) to maximize the efficiency of the testing process. The results of Regression Testing are automatically written to a *Test Log* file, which can be inspected to determine the outcome of testing.

16.6 Artifacts

The following artifacts are employed in the Crown testing process:

- ▲ The Test Plan document (see Appendix C, for example)
- ▲ The Business Specification document (BSD; equivalent to the traditional Functional Specification document [FSD]). The purpose of the BSD is to specify the business requirements in such a way that they are unambiguous and can be readily interpreted by development and support staff. The principal sections covered in this document include:
 - *an overview of the function(s) covered by the specification*

- *a statement of the business or technical function that is to be satisfied by a module or object and the key features that are to be incorporated*
- ▲ The FSD (containing a section written by the QA Manager that is analogous to the traditional *Test Specification document* described in Part 1 of this book and in Appendix D). The purpose of the FSD is to specify the requirements in such a way that they are unambiguous and can be readily interpreted by development and support staff. The principal sections covered in this document include:
 - *an overview of the function(s) covered by the specification*
 - *a detailed breakdown of each function into its constituent parts (for example, file layouts, screen templates, report layouts, calculations, process characteristics, and logic)*
 - *an analysis of the impact the development will have on other areas of the system*
 - *any special conditions to be taken into consideration during testing.*
- ▲ Automated Test Scripts and Verification Points
- ▲ Manual Test Scripts and Test Cases (where required)
- ▲ Test Log – detailing the day-to-day events of each testing project
- ▲ Re-use Pack (see Appendix I)
- ▲ Acceptance Test Certificates.

To ensure that all testing projects have access to the same testing artifacts and to ensure the use of standard testing documentation, Crown controls all testing artifacts using the Visual SourceSafe system, which is used as a repository for all QA documents.

16.7 Process Improvement

Crown is committed to striving continually to improve its software testing process for two principal reasons: the firm is eager to benefit from the commercial advantage it gives them and because of its involvement in the EU process-improvement project (Reference 27).

Crown collects and utilizes the following metrics as part of its process-Improvement initiative:

- ▲ Total effort expended in testing. An important use of this information is in the analysis of the benefits of the use of automated testing
- ▲ Total number of Test Procedures or Scripts run against the AUT at System and Acceptance (and optionally, Regression) Testing. Useful for estimating how many automated scripts can be created for a given amount of effort
- ▲ Number of defects detected versus time in testing (and optionally, in use by the client). Analysis of this information is useful in determining when to stop testing (by plotting the rate of detection of defects and looking for this rate to level out), as well as for showing defect trends across the duration of the testing project. Defects are categorized as either significant or minor bugs to simplify analysis.

- ▲ Defect detection effectiveness (percentage), or DDE. This metric provides an indication of how effective the Crown testing process is over time and is a good metric to use in improving the process (that is, the DDE percentage should increase the more effective the testing process becomes). DDE is calculated as follows:

$$\text{DDE} = (\text{TDFT} / (\text{TDFC} + \text{TDFT}) \times 100, \text{ where}$$

TDFT = Total Defects Found by Testing (that is, by the Testing Team)

TDFC = Total Defects Found by Client (measured up to some standard point after release – 6 months, for example)

- ▲ Defect Removal Effectiveness (percentage), or DRE. This metric provides an indication of how effective the Testing Task is at removal of defects. DRE is calculated as follows:

$$\text{DRE} = \text{TDCT} / \text{TDFT} \times 100, \text{ where}$$

TDCT = Total Defects Closed during Testing

TDFT = Total Defects Found during Testing

- ▲ Test Design and Test Development Coverage (that is, the degree to which the test requirements have been addressed and tests planned, designed, implemented, and run). This information is easily obtained from within the Rational TeamTest tool as part of its standard test management and reporting facilities.

The process of collecting and analyzing testing project metrics is simplified by the use of the reporting facilities provided in the automated testing tool. For example, it is very straightforward to run predefined coverage reports that show how many tests have been planned, designed, run, and passed.

Case Study 4: The Wine Society

David Marsh, Morag Atkins, and Annette Phillips

17.1 Overview of the Organization

The Wine Society is a cooperative owned by its shareholders (its “members”). It was founded in 1874 to serve its members rather than to be run for profit. This ethos still holds true today. An elected body of members oversees the Wine Society and represents the interests of over 85,000 active members.

The objective of the Wine Society is to offer its members wines and spirits of good quality from a wide range of sources at the best prices possible, which it has consistently managed to achieve, with a list of more than 800 different wines available either by the bottle, in unmixed dozens, or in mixed cases of the members’ own selection. Pre-selected mixed cases are also available and come with comprehensive tasting notes. The Wine Society organizes delivery of member purchases to any address in the United Kingdom.

The Wine Society has extensive storage facilities with temperature-controlled warehouses in Stevenage. The longer-lived wines are allowed to mature gently and reach their optimum potential. It is also possible for members to take advantage of the Society’s storage facilities by storing wines bought from the Society in the designated Members’ Reserve section of the cellars. Updated drinking dates are provided to the members taking advantage of this facility annually to apprise them of the state of maturity of their stock.

The wines are selected by a team of experienced and highly knowledgeable buyers led by a Master of Wine.

The Wine Society has a significant reliance on *Information Technology (IT)* systems for much of its business requirements, including:

- ▲ Holding members’ details
- ▲ Holding current and historical stock details
- ▲ Holding information on stock held in the Members’ Reserve and its age
- ▲ Assisting in picking, packing, dispatching, and delivering goods
- ▲ Supporting the invoicing of members
- ▲ Sales Order Entry facilities.

Traditionally, these diverse business requirements had been supported using a number of different IT systems as well as manual facilities. In 1997, the Wine Society began the process of replacing these IT and manual systems with a single integrated

software solution – Maginus. This case study describes the details of the project (Project 2000), which was set up for the purposes of acquiring this software, and more specifically, for *Acceptance Testing* the application.

David Marsh is currently the Information Systems (IS) Manager for the Wine Society, and at the time of this case study was one of the members of the Program Board responsible for directing the Project 2000 Task Force (see Section 17.3). Morag Atkins is currently working as an analyst in the Wine Society IS department, and at the time of this case study was a *User Representative Tester* in the Project 2000 team. Annette Phillips is currently working as a QA Analyst in the Wine Society IS department, and at the time of the case study was also a User Representative Tester (Section 17.4 contains descriptions of these testing roles).

17.2 Testing Requirements

This case study deals specifically with a testing project set up by the Wine Society to Acceptance Test the Maginus Software system (the *Application Under Test [AUT]*) from MANCOS Computers Limited (the Supplier), which took place between March and August 1998.

The AUT is a *commercial off-the-shelf* (COTS) product, which is intended to support the majority of the Wine Society's business processes in a single comprehensive software system. In order for the software to more closely match its business processes, the Supplier undertook customization of the AUT under contract to the Wine Society.

The customization of the AUT employed a *Rapid Application Development* (RAD) approach (see Reference 12, for example), which involved a number of Wine Society User Representatives in regular reviews of prototype graphical user interfaces (GUIs). Rigorous configuration and change management was employed to ensure that there was no confusion between the version of the GUI delivered for testing and the earlier prototype versions.

The AUT was delivered to the Wine Society in a phased manner, with three major deliveries over a four-month period. It was planned that some interim testing would take place of the first two deliveries of the AUT, but given the rapid release cycle, it was decided to perform the full Acceptance Test following the final delivery.

For the purposes of planning and managing the testing process, the following characteristics of the AUT were identified:

- ▲ The AUT represents a business critical system since it will support the majority of the Wine Society's business processes
- ▲ The AUT comprises five major business modules (Stock, Sales, Purchase, Database, and Finance) plus an administrative module, each of which will require rigorous subsystem testing
- ▲ Testing to verify the communications between modules will also be required as and when business processes demand
- ▲ The AUT will need to interface to a number of "external" systems, and each of these interfaces must be tested

- ▲ The AUT stores and processes large volumes of data, including:
 - *Records on more than 200,000 members*
 - *Details on Wine Society stock, with granularity down to a single bottle*
 - *Member transaction data for more than 1000 orders a day.*
- ▲ Because of the large data storage and processing requirements, thorough *Performance testing* will be required, including *Volume*, *Load*, and *Stress testing*
- ▲ The AUT runs on a Sun Workstation under the Unix operating system
- ▲ The AUT will follow a regular planned release schedule following delivery of the customized software to the Wine Society to incorporate user change requests, to resolve defects, and to keep abreast of commercial changes in the underlying hardware and operating system. These releases will undergo thorough *Regression Testing* by the Wine Society prior to installation on the *live system* to ensure that existing functionality has not been compromised.

The Wine Society proposed the following high-level approach to testing the AUT:

- ▲ Testing would be resourced from and conducted within the Wine Society Project 2000 Taskforce group
- ▲ External Testing Consultants would be employed in the testing process as and when required to provide *Test Analyst* skills and to assure the correct adoption and use of the testing process (at Acceptance Testing, for example)
- ▲ The testing process would involve nominated User and Supplier representatives with specified formal roles and responsibilities within the process
- ▲ Since there would be regular releases/upgrades of the AUT, re-use of testing materials/tests for the purposes of Regression Testing would be a major goal.

The Wine Society is an interesting case study because although ostensibly dealing with the Acceptance Testing of a COTS software product, the testing process in fact encompasses many of the elements of a more complete software development and testing project, including:

- ▲ Traditional Acceptance Testing
- ▲ Elements of *Integration Testing* between the subsystems
- ▲ Extensive *Nonfunctional* Testing, including:
 - *Load*, *Stress*, and *Volume testing*
 - *Usability Testing*
- ▲ *Systems Integration Testing* involving external software systems
- ▲ Rigorous Regression Testing.

Section 17.5 describes the details of the individual testing phases.

17.3 The Management and Planning of Testing

Prior to the decision to acquire the customized Maginus system, the Wine Society IT Group had little direct experience in testing an application of the size and complexity of the AUT.

In Acceptance Testing the AUT, the Wine Society understood the need to follow a formal testing process. With the assistance of a number of testing consultants, the Wine Society developed a testing process specifically tailored to its particular testing requirements as well as to the Society’s internal organizational structure.

The Project 2000 Task Force (the Project) was created in October 1997 specifically to formally manage the acquisition of the AUT, with members being drawn from across the Wine Society organization. The day-to-day management of the Project was conducted using the PRINCE project management methodology (Reference 3), and the overall direction of the Project activities were managed by means of a Project Board, which contained technical, user, and director-level representatives (Appendix A provides information on the composition and responsibilities of a typical Project or Program Board).

In setting up the Project organization, a management decision was made to closely involve the user community. A number of User Representatives were co-opted onto the Project to perform the role of *Testers* (or *User Representative Testers*) and worked closely with the Testing Consultant to form an effective *Testing Team*.

Figure 17.1 provides a high-level overview of the organization of the roles of the staff involved in the Wine Society testing process and their interrelationships.

The Wine Society Testing Team was managed on a day-to-day basis by the Project Manager, who reports to the Project 2000 Task Force Project Board (Project Board). The detailed responsibilities of the staff shown in Figure 17.1 are discussed in detail in Section 17.4.

The planning of testing followed a pragmatic approach, based upon the delivery dates for the AUT and the final live date for the software to be installed, tested, and in use. Within these hard milestone dates, the testing had to be as effective and as efficient as possible, and the Testing Consultant and Project Manager employed

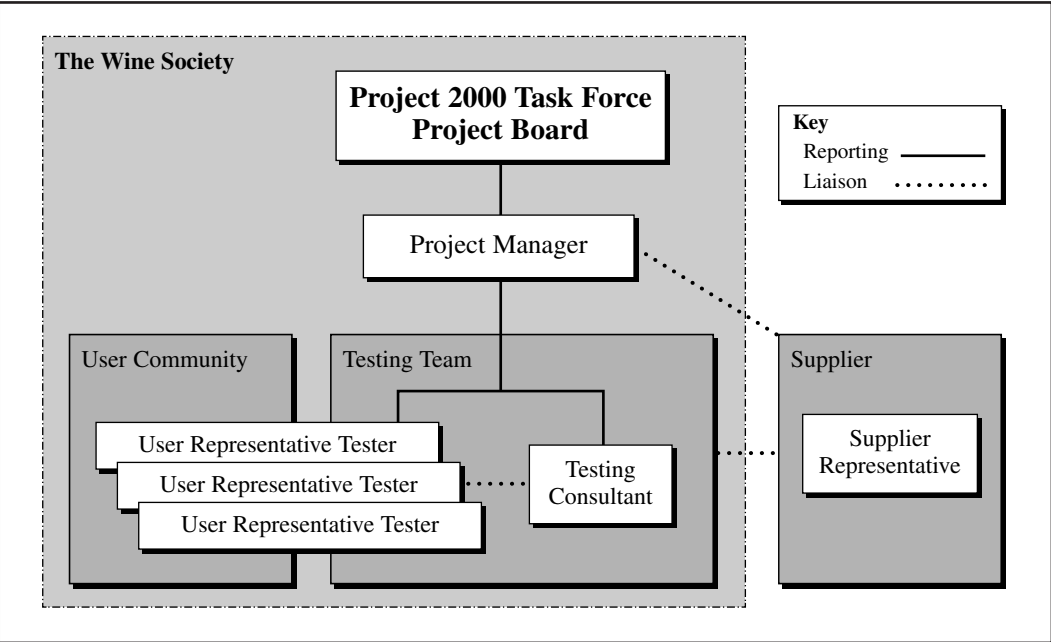


Figure 17.1. Wine Society roles, reporting, and liaison.

their previous experience of the software development and testing process, in combination with the PRINCE methodology, to create the plans for the individual testing tasks. Contingency time was allowed in the plan to accommodate any retesting that might be necessary.

Within the context of the PRINCE methodology, the Testing Consultant conducted a Risk Analysis of the Project, and specifically of the issues associated with the testing tasks. This activity was key to identifying particular areas of risk that needed to be explored during testing and to assist in the planning process by prioritizing the order of testing of the AUT. This was particularly important given the challenging fixed timescales involved in the delivery of the AUT and its live date.

The specific details of the testing plan were enshrined in a Test Plan document. Low level planning and design of the Test Scripts and Test Cases were based on the requirements for the AUT as listed in the *Requirements Specification* document (Section 17.6 contains details of the artifacts that were associated with the Project).

The human resources for testing were drawn from within the Project, which had been set up to accommodate such resource requirements. Details of resourcing were recorded in an *Acceptance Test Specification* document.

The Business Specification for the customizations were obtained by the Supplier in discussion with the members of the Project, and specifically with the User Representatives. This information was recorded in a formal *Requirements Specification* document.

17.4 Roles and Responsibilities

For the purposes of Acceptance Testing the AUT, it was determined that the following roles and responsibilities were needed:

- ▲ Project Manager
- ▲ User Representative Tester
- ▲ Testing Consultant
- ▲ Supplier Representative.

Each of these roles is described in the following sections.

Project Manager

The Project Manager was responsible for the day-to-day running of the Project and the testing associated with the AUT and had to ensure that the outputs from the Project (which includes the tested AUT) were fit for purpose. With respect to the testing aspects of the Project, the Project Manager's responsibilities included:

- ▲ Managing the set-up and resourcing of the Acceptance Test
- ▲ Managing the set-up of the Acceptance Test filing system
- ▲ Managing the planning of the Acceptance Test
- ▲ Ensuring the production of the Acceptance Test Plan document

- ▲ Ensuring the production of the Acceptance Test Specification
- ▲ Tasking the User Representative Testers and Testing Consultant and monitoring their progress against agreed-on plans
- ▲ Liaising with the Supplier Representative to agree on delivery dates for the test-ready AUT, and to agree on the time and date for attendance at the Acceptance Test
- ▲ Ensuring that the supplier followed good testing practice during development and testing of the AUT and reviewed the supplier Unit, Link (as the Supplier termed “Integration Testing”), and System Test documentation
- ▲ Formal attendance at the Acceptance Test for the purposes of co-signing the *Acceptance Test Certificate* at the conclusion of a successful Acceptance Test.

The Project Manager reports to the Project Board, is reported to by the User Representative Testers and Testing Consultant, and liaises with the Supplier Representative.

User Representative Tester

The User Representative Tester is responsible for the execution of the steps specified within the *Test Scripts*. The User Representative Tester should be a representative of the Wine Society who is familiar with using the business processes supported within the AUT. Specifically, the User Representative Tester is responsible for:

- ▲ Discussing the extent and scope of testing for the AUT with the Testing Consultant
- ▲ Review of the Test Scripts and *Test Cases* produced by the Testing Consultant for errors or omissions
- ▲ Execution of Test Scripts and observation of test results
- ▲ Maintenance and retention of *Test Result Record Forms*
- ▲ Set-up and initialization of the test bed
- ▲ Backup and archival of the test bed
- ▲ Recovery of the test bed in the event of failure
- ▲ Creation and maintenance of test *Re-use Packs* (see Appendix I)
- ▲ Reporting testing progress to the Project Manager at agreed-on intervals.

The User Representative Tester reports to the Project Manager and liaises with the Testing Consultant. In practice, the use of IT-naïve staff in a testing role proved to be very effective since the Testers were entirely familiar with the goals and operation of the AUT and had no preconceptions about what was possible or even expected of technically oriented testing staff. Such an approach was made possible because of the availability of the Testing Consultant, who was able to advise and assist the User Representative Testers.

Testing Consultant

The Testing Consultant, appointed by the Project Manager, is responsible for verifying that the Acceptance Test is conducted according to the documented test process and that the results of testing are interpreted and recorded correctly. Specifically, the Testing Consultant is responsible for:

- ▲ Assisting the Project Manager in the development of the Acceptance Test Plan document
- ▲ Assisting the Project Manager in the development of the Acceptance Test Specification document
- ▲ Discussing the extent and scope of testing for the AUT with the User Representative Tester
- ▲ Analyzing the *Functional Requirements* Specification document to design Test Scripts and Test Cases to Acceptance Test the customizations to the AUT
- ▲ Reviewing the Test Scripts and Test Cases with the User Representative Tester to check for errors or omissions
- ▲ Creating Test Scripts and Test Cases for retest of defects where necessary
- ▲ Determining the need for handcrafted test data for the Acceptance Test and its subsequent design and implementation using formal design techniques (see Chapter 3)
- ▲ Developing the standard documentation to be used in the Acceptance Test (such as Test Script templates, Test Result Record Forms, and Acceptance Test Certificates)
- ▲ Attending the Acceptance Test and observing the results of executing the Test Scripts and Test Cases, interpreting the result, and assigning the result to one of the Test Result Categories
- ▲ Formal attendance at the Acceptance Test for the purpose of providing “independent” observation of the execution of the Test Scripts and the results of the Test Cases
- ▲ Reporting testing progress to the Project Manager at agreed-on intervals.

At the end of the Acceptance Test, the Testing Consultant is responsible for agreeing on any remedial work that may be required on the AUT and the timescales involved in discussion with the Supplier Representative.

Supplier Representative

The Supplier Representative is responsible for attending the Acceptance Test on behalf of the Supplier and has the authority to sign the Acceptance Test Certificate on behalf of the Supplier.

The Supplier Representative is also responsible for agreeing on any remedial work that may be required on the AUT and the timescales involved in discussion with the Testing Consultant.

17.5 Testing Phases

As discussed in Section 17.2, although the Wine Society process for accepting the AUT is ostensibly a traditional Acceptance Test of a COTS product, in fact the process has to address a number of other testing issues, including *Integration*, *Performance*, *Usability*, and *Systems Integration Testing*.

In addition to Acceptance Testing, the AUT was expected to undergo further

releases (for the correction of defects and/or the inclusion of enhancements due to user change requests), and so rigorous Regression Testing would be necessary against new releases to ensure that the existing functionality and performance of the system would not be compromised by any changes introduced for a new release.

Acceptance Testing

Based on the Functional Specification document for the AUT, the Project Manager and the Testing Consultant scoped out the high-level testing requirements and produced the Test Plan and Test Specification documents. These documents provided information on how to approach the Acceptance Test, its duration, the resources necessary to perform the testing, and the scope of the Acceptance Test.

Specifically, the Acceptance Test needed to verify the following areas of the AUT:

- ▲ The functional aspects of the AUT implemented for the Wine Society as customizations by the Supplier
- ▲ The existing functionality provided by the AUT to ensure it had not become unstable due to the customizations
- ▲ The correct operation of each of the five main modules plus the Administrative module
- ▲ The correct operation of the business processes supported between the sub-systems
- ▲ The correct operation of the AUT with respect to the external systems it needed to interoperate with
- ▲ The adequate performance of the AUT with respect to its business processes under conditions of load and stress
- ▲ The suitability of the user interface in terms of the simplicity, consistency, and intuitiveness, and that appropriate levels of on-line Help were available to assist the user in successfully completing business tasks.

The specific Test Cases required to test the above areas were based on analysis of the Functional Specification and Business Process documents by the Testing Consultant.

Regression Testing

Because of the business critical nature of the AUT, it was determined that the scheduled releases of the software would require thorough and rigorous Regression Testing.

In order to save time and effort expended in Regression Testing and to ensure a consistent and thorough approach was maintained, it was decided to maximize the efficiency of the Regression Testing process by the use of Re-use Packs (see Section 17.6 for details on the components of the Re-use Pack; Appendix I provides further examples of the contents and use of such a pack).

At the conclusion of the Acceptance Test (and following a new release of the AUT), the User Representative Tester is responsible for making copies of the Test

Scripts and Test Cases and any other relevant testing materials (such as the test data) and placing them into the Re-use Pack, which is then filed for re-use during the next Acceptance Test.

At the planning stage of the next Acceptance Test, the Project Manager and User Representative Tester will obtain the Re-use Pack and review its contents to identify testing materials (such as Test Scripts) that can be re-used in the current testing phase.

Apart from the obvious benefits of saved time and effort, the testing process also benefits from the re-use of “tried and trusted” testing materials, which the testing staff have confidence in using.

17.6 Artifacts

The artifacts employed in the Wine Society testing process include:

- ▲ The Supplier Software Testing Approach document – a formal document produced by the supplier to record its approach to testing the AUT during development and customization. The role of this document within the Wine Society testing process was to provide confidence in the rigor of the Supplier approach to testing.
- ▲ The AUT Functional Specification document – this document contains the details of the specific customizations that were to be implemented in the AUT as specified by the user community via the User Representative Testers
- ▲ The Business Process document – in essence an annotated flow chart describing the business threads or scenarios that the AUT must support. This document was used heavily for risk analysis, prioritization, and planning of the testing process
- ▲ The Wine Society Acceptance Test Plan – this document is equivalent to the traditional Test Plan document described in Part 1 of this book and in Appendix C and contained the following sections:
 - *Introduction (giving the Background, Structure of the Document, and References)*
 - *Test Approach and Constraints (describing the approach to be adopted in testing the AUT and any constraints that the testers should consider)*
 - *Risks and Dependencies*
 - *Test Assumptions and Exclusions*
 - *Entry and Exit Criteria*
 - *Testing Project Controls*
 - *The Test Plan (recording tasks, durations, milestones and deliverables, and any dependencies).*
- ▲ Wine Society Acceptance Testing Specification document – this document is equivalent to the traditional Test Specification document described in

Part 1 of this book and in Appendix D and contained the following sections:

- *Introduction (Background, Scope, Structure of Document, and References)*
- *Acceptance Test Requirements (Introduction, Test Philosophy, Test Environment, and Test Requirements)*
- *Acceptance Test Procedure (Introduction, Pre-Test Activities, Conducting the Acceptance Test, Post-Test Activities).*
- ▲ Acceptance Test Scripts and component Test Cases
- ▲ Acceptance Test Certificates
- ▲ Test Result Record Forms
- ▲ Re-use Packs, containing both hard and soft copies of:
 - *Guidance on the use of the pack*
 - *Acceptance Test plan template*
 - *Test Scripts templates*
 - *Test Result Record Form templates*
 - *Acceptance Test Certificate template*
 - *Sets of previously conducted tests (Test Scripts and Test Cases).*

17.7 Process Improvement

No specific process improvement approach was adopted within the Project because of the fixed term of the Acceptance Testing task. However, at the completion of each testing task, a PRINCE Post Implementation Review was conducted in order to identify the positive aspects of the task as well as any aspects of the task that could be improved. Wherever possible, this information was used to improve the subsequent tasks in the testing process.

In August 1998, the customized Maginus system entered live operation and shortly thereafter the Project was closed. In practice, the testing process adopted by the Wine Society for the Acceptance Testing of the AUT proved to be successful, with the Project meeting its planned timescales and with the AUT being accepted into operation on time and at an acceptable level of quality.

Following the successful completion of the Project and the dissolution of the Task Force, the staff returned to their original jobs. One very positive result of bringing staff from different areas of the Wine Society (such as the User Representative Testers) into the Project is that following its dissolution, a very great deal of experience of testing and the testing process has become devolved into the different groups within the organization. For example, members of the user community now have a much greater empathy with staff involved in testing and a greater understanding of what is involved in accepting software systems. Similarly, members of the marketing department now have more realistic expectations about what testing can do for them (in terms of testing timescales and the quality of the software they use).

New versions of the Maginus software continue to be delivered by the Supplier following an ongoing release schedule. The testing of these new releases of the AUT is now the responsibility of staff in the Wine Society IT Department who were formally members of the Project 2000 Task Force, using the testing process, techniques, and experience gained on that Project.

Case Study 5: Automatic Data Processing Limited

Martin Kemble, Senior QA Analyst

18.1 Overview of the Organization

Automatic Data Processing (ADP) Limited is the leading provider of Employer Services in the world. Employer Services in the past was simply a synonym for payroll, but now it has a significantly wider definition, embracing human resources, expense management, time and attendance, recruitment services, and employer solutions based on Internet technologies. As each of these service extensions has matured, ADP has been in the vanguard of technical and business innovation, consistently redefining the market and expanding the value its services deliver to clients.

Founded in 1949, ADP has been an astonishing success story for more than half a century. Its unrivaled record of profitable growth over more than 150 consecutive quarters bears witness to the extraordinary stability of ADP, making it the partner of choice in critical aspects of a vast number of businesses. Through a single-minded commitment to providing high-quality, employment-related services, such as payroll, ADP has grown steadily and now serves a client community of in excess of 450,000 organizations worldwide.

ADP Employer Services began its United Kingdom (UK) operations in 1965, and these are now available throughout the UK via ADP's purpose-built client, sales, and support centers in Staines, Manchester, Birmingham, and Chessington.

ADP's business strategy is based on the need to grow through consistent improvement and innovation in the provision of Employer Services and where appropriate to exploit new and emerging technologies to assist in this goal.

ADP offers a broad spectrum of Employer Services, including:

- ▲ Surepay – a complete suite of facilities and tools to enable payroll offices to produce accurate and timely payrolls
- ▲ Managed Payroll Service – an outsourced payroll solution in which ADP takes responsibility for the preparation and delivery of the client payroll
- ▲ Teledata Payroll Service – a semimanaged payroll solution that is appropriate to clients with smaller numbers of employees

- ▲ International Payroll Service – a unique, fully managed payroll and tax filing service designed for the particular needs of overseas organizations with satellite operations in the UK.

This solutions-oriented approach has helped ADP become the world's leading provider of computerized transaction processing, data communications, and information services.

This case study deals with a relatively small-scale testing project set up to perform *Acceptance Testing* of the software element of a payroll application prior to its delivery to a number of ADP clients as part of a managed payroll solution. At the time of this case study, Martin Kemble was the Senior Quality Assurance Analyst within the ADP Software Development Department based in Staines, Middlesex, and was responsible for the day-to-day management of the Acceptance Testing Team.

18.2 Characteristics of the Testing Requirement

While ADP is a very large organization with diverse requirements for software development and testing, this case study focuses on the testing process adopted by the Software Development Department of the Employer Services division in the UK for the purposes of Acceptance Testing a payroll management application (the *Application Under Test*, or AUT). Following successful Acceptance Testing, the AUT was to be delivered by ADP to its clients as part of a managed payroll solution.

The software requirements for the application were provided by members of the Client Implementation Team at ADP, who employed their knowledge and experience of the prospective clients' business processes to specify the details of the system. These requirements were recorded in a *Software Requirements Specification* document (the ATT Requirements Specification document – see Section 18.6).

The AUT underwent an iterative development process with incremental releases of the software delivered to the ADP Acceptance Testing Team (ATT). As a result of the incremental release cycle, the ATT was responsible for both Acceptance Testing the new functionality for a particular release and rigorous *Regression Testing* of the previously accepted functionality.

A *Thread Testing* technique was employed to test the functional aspects of the AUT (see Chapter 3) based on a Business Thread document produced by the testing team *Quality Assurance (QA)* Analyst (details of the QA Analyst role and ATT Business Thread document are provided in Sections 18.4 and 18.6, respectively). In addition to the functional aspects of the software, the ATT was tasked with verifying some usability issues (such as the “ease of use”) of the AUT from a client perspective.

Because the AUT was planned to be delivered to each client as a distributed system with up to twenty users, the AUT also required *Performance Testing* to provide confidence that the software would execute with adequate speed under a variety of different conditions of *Load*, *Volume*, and *Stress*.

In conjunction with their formal testing process, the ATT also incorporated the use of an automated software testing tool for both *Functional* and Performance Testing of the AUT. The decision to employ an automated testing solution was based on the goal of providing ADP with a commercial advantage over its competitors by reducing the timescales, effort, and cost of testing and improving product quality. The need to provide rigorous and complete automated Regression Testing of each incremental release of the AUT was another important factor in the selection of a testing tool.

ADP's approach to the selection of a tool involved the release of a formal Invitation to Tender (ITT) to a short list of leading testing tool and consulting companies. The principal requirements stated in the ITT were:

- ▲ To provide an effective testing tool suite capable of matching the characteristics of the software development and testing process
- ▲ The availability of functional and performance testing facilities seamlessly integrated with the ability to represent test requirements and provide rigorous defect tracking
- ▲ The availability of effective training, mentoring, and consulting in the use of the testing tool suite from the selected vendor
- ▲ The financial stability of the tool vendor and a commitment to continue to support and update the testing tool suite in line with industry and technology trends
- ▲ The availability of and continued commitment to “thought leadership” from the tool vendor who should be involved in researching and developing industry leading technologies and techniques and delivering these to their customers
- ▲ The availability of testing process consulting to extend its existing manual testing process to fully integrate the automated testing approach.

Following a rigorous review and evaluation of the proposals, ADP selected the Rational Suite (or SQA Suite, as it was known at the time – Reference 25) testing product.

Chapter 3 of this book gives additional information on the role and use of automated testing tools, and Reference 17 provides a comprehensive treatment of automated software testing tools and their application. Appendix N contains a scheme and criteria for use in evaluating automated testing tools.

The ATT project can be characterized as a small testing project with relatively few staff involved in the testing process. ADP has a strong culture of quality, and the testing project was organized under the auspices of the Quality Assurance (QA) Group. The Acceptance Testing of the AUT took place between 1997 and 1998, and this case study describes the project structure and organization during this period.

18.3 The Management and Planning of Testing

The Acceptance Testing Team project followed a rigorous testing process, specifically tailored to its particular testing requirements and documenting the principal aspects of its testing process. The ATT testing process was formally recorded in the ATT Testing Procedures document (see Section 18.6).

Figure 18.1 provides a high level overview of the organization of the ATT and its reporting and liaison links to senior management and the third-party company developing the AUT.

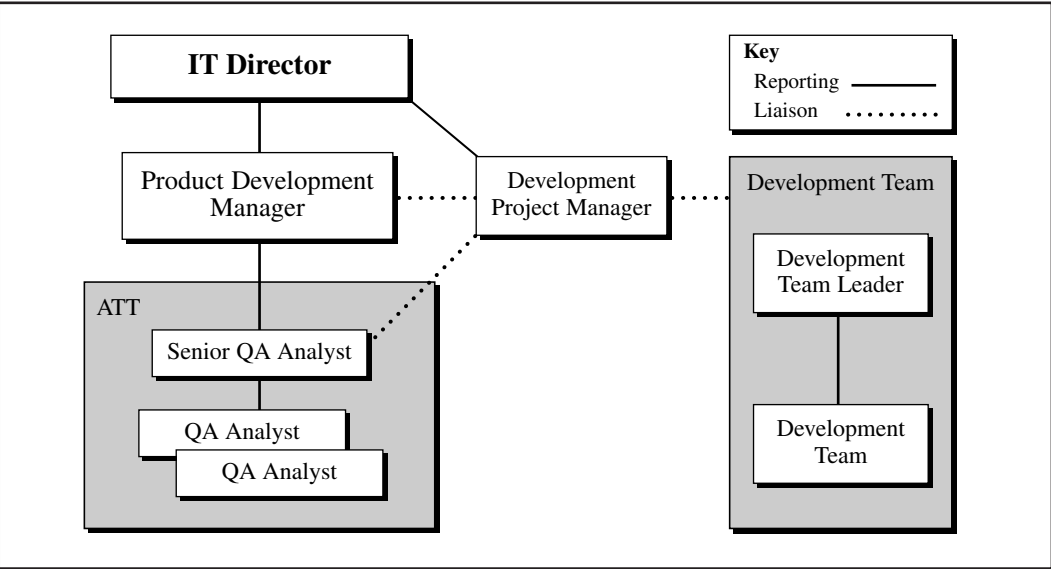


Figure 18.1. ADP Testing Process organization.

The ATT is managed by the Product Development Manager, who reports to the IT Director. The ADP Development Project Manager provides the interface to the Development team for the IT Director, Product Development Manager, and the Testing Team members. The precise responsibilities of the Product Development Manager and the other subordinate roles are discussed in detail in Section 18.4.

Because of its particular testing requirements, the ATT view of the *V Model* (see Figure 18.2 and Chapter 4) focuses on the Acceptance phase of Testing, with the earlier testing phases being conducted by third-party developers during implementation of the AUT.¹ In addition to Acceptance Testing, ATT also has a responsibility for Regression Testing the AUT.

Following the V Model approach, the Senior QA Analyst and QA Analysts are involved in planning and early test design of the Acceptance Test during the Requirements phase of development. These activities are described in detail in the following section.

Defect management within the ATT was performed using the built-in defect tracking facilities within the Rational Suite testing product. This approach allowed defect tracking to be integrated with the *automated testing* process, with defect

¹Even though the development of the AUT follows an iterative approach, the V Model (which traditionally applies to the waterfall style of development) is equally relevant to the planning of the ATT Acceptance Testing. This is possible because each iteration can be considered analogous to a small-scale waterfall.

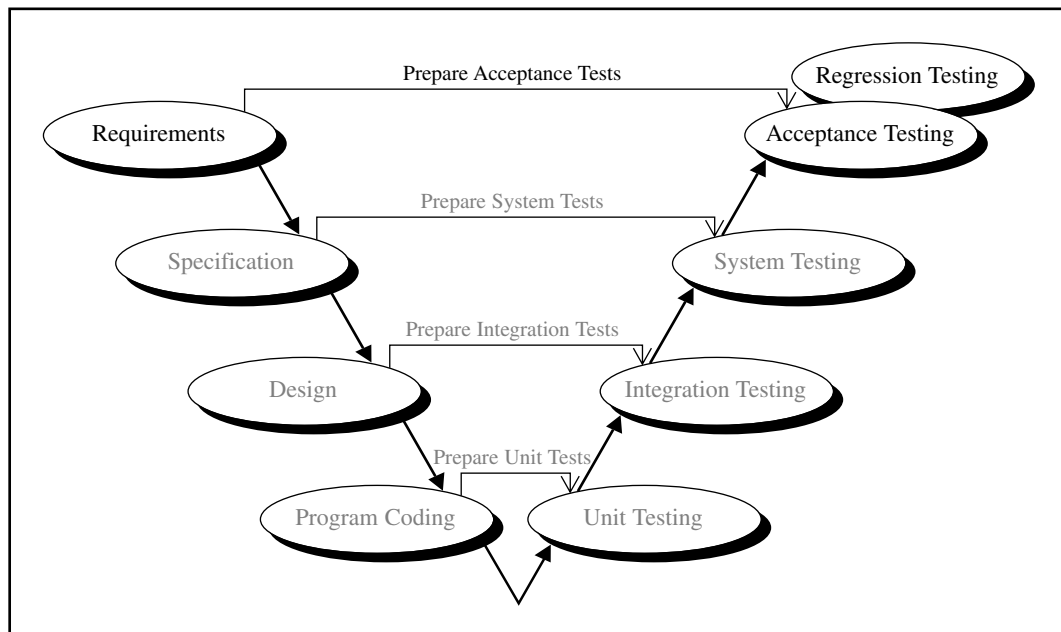


Figure 18.2. The ATT view of the V Model.

reports being created by the QA Analyst following inspection of the automated *test logs*. Defects were then automatically relayed to the developers by means of an integrated e-mail facility. The same system was used to track and report on the progress of defect correction, and ultimately, following successful retest (in effect, rerunning the original automated test), the defect could be closed.

18.4 Roles and Responsibilities

The roles and responsibilities defined for the ATT directly reflect the nature of the testing project. The relatively small number of staff involved in the testing process has a number of implications for overlap of responsibilities, with individual members of staff in effect performing more than one role within the project.

Because of the strong quality culture supported within ADP, the ATT was set up and administered by members of the Quality Group. This contrasts with those organizations (such as the British Library – Chapter 14, or Reuters, Chapter 15) in which the testing projects are administered from within an IT group.

The following ATT testing roles and responsibilities are described in subsequent sections:

- ▲ Product Development Manager
- ▲ Development Project Manager
- ▲ Senior QA Analyst
- ▲ QA Analyst.

Product Development Manager

The role of Product Development Manager is analogous to the role of *Testing Manager* in a traditional testing project (see Chapter 4 and Appendix A). Where there are differences between the roles, these are mainly due to the position of the ATT within the Quality Group combined with the small number of staff involved in the testing process.

The Product Development Manager is responsible for ensuring that the ATT produces the required products (including the fully Acceptance Tested AUT), as well as any incidental artifacts (such as the *Test Plan* and *Test Specification* documents) to the required level of quality and within acceptable timescales (as specified in the ATT Testing Procedures and ATT Test Plan documents, respectively).

The Product Development Manager is also responsible for reviewing the *Requirements Specification* document for the AUT to determine the resources required for testing and to identify any specific risks that may affect the testing process (such as possible late delivery of the AUT) and for producing the ATT Test Plan document. The Product Development Manager is also responsible for ensuring that the ATT Testing Procedures document is produced by the Senior QA Analyst and that the process described in the document is followed by the ATT.

Due to the relatively small size of the ATT, the Product Development Manager also performs the role of “*Independent Test Observer*.” In effect, the Product Development Manager quality-assures all aspects of the testing project, being involved in the review of testing artifacts and reviewing the results of the Acceptance Test to ensure correct procedures were followed (this also involves assuring the suitability of the development and test process employed by the developers of the AUT). The use of an automated testing tool by the ATT further reduces the need for independent observation of testing, since the testing tool does not suffer from human error during testing, and the results of testing are logged-in with complete fidelity for subsequent analysis.

The Product Development Manager reports to the IT Director and is reported to by the Senior QA Analyst. The Product Development Manager also liaises with the Development Team Leader via the Development Project Manager for the purposes of determining the timescales for delivery of the AUT for Acceptance Testing and of verifying that the development team employs appropriate quality procedures in the development and testing of the AUT.

Development Project Manager

The Development Project Manager is responsible for managing the development aspects of the AUT from an ADP perspective and for monitoring the progress of the development team against agreed-on plans. In effect, the Development Project Manager provides an interface between the developers and the other ADP staff involved in the Acceptance Testing project.

The Development Project Manager reports to the IT Director, is reported to by the developers, and liaises with the Product Development Manager and the ATT.

In practice, this role has proven to be very beneficial to the testing process, providing a single point of contact for all ATT members to contact the *Development Team*.

Senior QA Analyst

The Senior QA Analyst is responsible for the day-to-day activities of the testing team. The specific responsibilities of the Senior QA Analyst include:

- ▲ Set-up and maintenance of the testing project filing system
- ▲ Reviewing the ATT Requirements Specification document for the AUT in conjunction with the QA Analysts to determine the testability of the software requirements and to identify any errors and/or omissions and any specific resources needed for testing
- ▲ Producing the ATT Testing Procedures document, ensuring that the other members of the ATT are aware of the document and its contents and follow the defined test process
- ▲ Tasking the QA Analysts and monitoring their progress against agreed-on plans
- ▲ Reporting testing progress and issues to the Product Development Manager at agreed-on intervals
- ▲ The installation, maintenance, and updating of the test automation tools
- ▲ Liaison with the test automation tool vendor for requests for support, training, mentoring, and consulting
- ▲ The planning and organization of training, mentoring, and consulting services from the test automation tool vendor.

The Senior QA Analyst reports to the Product Development Manager, is reported to by the QA Analysts, and liaises with the Development Project Manager for the purposes of determining the likely dates for delivery of the AUT for testing.

QA Analyst

The QA Analyst role within the ATT combines aspects of the traditional *Test Analyst* and *Tester* roles described in Part 1 of this book (see Chapter 4 and Appendix A).

Because of the use of a testing tool to support the testing process, the QA Analyst is primarily responsible for the design, implementation, and execution of the *automated test scripts* and their component *Verification Points* (analogous to *Test Cases* in traditional manual testing). Where it is necessary to perform manual testing, the QA Analyst also has responsibility for the design and implementation of manual *Test Scripts* and *Test Cases*.

The specific responsibilities of the QA Analyst include:

- ▲ Reviewing the ATT Requirements Specification document in conjunction with the Senior QA Analyst
- ▲ Assisting the Senior QA Analyst in generating the ATT Testing Procedures and ATT Performance Testing Specification documents
- ▲ Producing an ATT Business Thread document describing the business functionality of the AUT for the purposes of Thread Testing
- ▲ Designing and implementing the automated Test Scripts and Verification Points
- ▲ Designing and implementing manual Test Scripts and Test Cases as required

- ▲ Designing and implementing Test Data Sets as required
- ▲ Setting up and performing the *Skim Test* on new releases of the AUT to ensure that it is of sufficient quality to undergo full Acceptance Testing
- ▲ Setting up the hardware, software, and data required to conduct the testing as specified by the ATT Testing Procedures document (for Functional Testing) and the ATT Performance Testing Specification document (for Performance Testing)
- ▲ Setting up the automated software testing tool prior to testing and executing the automated Test Scripts and subsequently reviewing the test results²
- ▲ Executing the manual Test Scripts (most capable automated tools provide facilities to support the integration of manual and automated testing – see Reference 25, for example)
- ▲ Reviewing the automated *Test Log* to identify any defects and generating a Defect Report for each observed defect
- ▲ Rerunning the appropriate automated tests to verify the correction of defects
- ▲ The generation of the *Re-use Pack* (this is largely accomplished by the routine use of the automated testing tool, which allows tests to be recorded against one release of the AUT and then simply rerun against a later release to identify any defects in the new release)
- ▲ The recovery of the test bed in the event of failure (again, the Rational Suite testing tool provides facilities for automatically recovering from system failure and allowing testing to continue)
- ▲ The backup and archival of all testing documentation and materials, including the automated test repository (containing all of the automated testing artifacts, such as Test Scripts and Verification Points, and Test Logs) and the test bed (that is, a copy of the hardware configuration, test data, and AUT).

The QA Analyst reports to the Senior QA Analyst.

18.5 Testing Phases

As described earlier in this chapter, the development of the AUT followed an incremental approach (see Reference 8, for example). Although iterative development methods typically produce software systems that are of appropriate quality within acceptable timescales, the more frequent build and release cycle has serious consequences for the testing process in terms of a much higher volume of testing; in addition to testing the most recently implemented functionality, it is also necessary to conduct thorough Regression Testing to ensure that the functionality implemented in earlier releases has not been adversely affected by the development completed for the current release.

²One major advantage of automated testing tools is that they can perform unattended testing of the AUT, allowing tests to be run overnight or over the weekend, and allowing the complete set of results (recorded in a Test Log file) to be inspected after testing has finished.

The result is that the ATT was faced with a very challenging testing load, both for Acceptance Testing the frequent releases and for ensuring that thorough Regression Testing was performed. Furthermore, as development of the AUT proceeded, the Regression Testing load would grow at each release because of the need to test the cumulative functionality of the AUT from earlier releases.

In order to keep up with the demanding incremental release schedule and the challenging testing load, the Senior QA Analyst proposed the adoption of an automated software testing solution. The use of automation in the Acceptance and Regression Testing is the principal difference between the ATT interpretation of these testing phases and the classic view presented in Part 1 of this book.

In order to maximize the efficiency of the Acceptance Testing process, a *Skim Testing* approach was employed to verify the suitability of a new release of the AUT to undergo Acceptance Testing.

The following testing phases are described in subsequent sections:

- ▲ Skim Testing
- ▲ Acceptance Testing
- ▲ Regression Testing.

Skim Testing

At each incremental delivery of the AUT, in order to ensure that the developers had provided a version of the software of sufficient quality to undergo full Acceptance and Regression Testing, the ATT performed Skim Testing. This pretest activity ensured that time, effort, and cost were not wasted in setting up and conducting a full Acceptance Test against a copy of the AUT that was of unacceptable quality.

The Skim Testing process involved an initial installation of the AUT on a specially configured PC containing test data. A standard set of manual tests stored in a Re-use Pack (see Appendix I, for example) was executed against the AUT to verify the correct operation of a number of key business threads (or business scenarios).

If the Skim Test was successful, the AUT would proceed to full Acceptance Testing. If not, the AUT would be rejected and returned to the developers with a defect report documenting the problems identified during the test.

ADP's Skim Testing is analogous to the Crown Management's technique of Crash Testing (see Chapter 16 – Case Study 3: Crown Quality Assurance Group). In practice, this simple pretesting check of the AUT has been consistently proven to save time and effort on the part of the testing team and make the testing process more effective and efficient.

Acceptance Testing

Using the Rational Suite testing product (Reference 25), the ATT was able to conduct Acceptance Testing to verify the suitability of the functionality implemented in the most recent release of the AUT while recording the steps performed in the test. In subsequent releases, the existing functionality of the AUT could be rigorously tested by simply replaying the tests recorded in previous Acceptance Tests.

The functional aspects of the Acceptance Tests were based on the ATT Business Thread document, which specified a number of business scenarios that users would employ in their normal use of the AUT (see Section 18.6). Traditional design techniques were used in the design of the Acceptance Tests, including:

- ▲ Thread Testing and State Transition techniques, which were employed in designing the Test Scripts and Test Cases (which were subsequently implemented as Test Scripts and Verification Points in the testing tool)
- ▲ *Boundary Analysis* and *Equivalence Partition* techniques, which were employed in the design of the handcrafted data for exercising boundary conditions and error-handling functionality.

The Performance Testing aspects of the Acceptance Tests were based on the functional test scenarios. During Performance Testing a simulation technique was employed that used a number of virtual users to represent the higher demand for system resources that would occur in the live system. The automated testing tool provided integrated support to allow the functional Test Scripts to be re-used in performance testing and to be assigned to the virtual users, who concurrently executed the business threads. The results of performance testing were automatically written to a test log, which detailed the load on system resources, such as processor utilization, network loading, and memory utilization, allowing subsequent analysis of the results of the performance test. In addition to ensuring the acceptable performance of the AUT with realistic numbers of users, performance testing was used to provide estimates of the minimum acceptable specification for the client computers to support the AUT.

Acceptance Testing also verified the correct operation of the database aspects of the AUT and included explicit tests for checking record locking and deadly embrace situations.

Following completion of successful Acceptance Testing of each incremental release, a process of formal sign-off of that version of the AUT was performed with the developers.

Regression Testing

Since the AUT dealt with the business-critical application of payroll management and was to be delivered to ADP clients as a managed solution, it was essential that the software be reliable, robust, and fit for purpose. Because of these requirements, the ATT had to ensure that rigorous Regression Testing of each new release was performed.

For maximum confidence, it was decided it would be necessary to completely Regression Test the AUT at each release. In a traditional Regression Test, because of the drive for efficiency in the testing process, the Test Analyst designs the tests by considering the scope of the changes to the AUT and analysis of the likely impact on the other areas of the functionality of the application. However, with such an approach there is always some risk that the changes to the AUT will have had some unforeseen effect on the software that selective testing will miss.

The Regression Testing employed by the ATT was achieved by the re-use of the automated test scripts that had been created during the earlier Acceptance Tests.

This approach provided an elegant solution to ensuring thorough and rigorous testing of the AUT, as well as managing the demanding testing load that the incremental development of the AUT caused.

Additionally, the ability of the automated testing tool to perform unattended testing meant that the tests could be executed overnight or over a weekend – saving time and effort, but also allowing more testing to be conducted in the same time that a manual approach would have permitted – increasing the quality of the testing and resulting in greater confidence in the correctness of the AUT.

In order to manage the complexity of the Regression Test suite,³ the ATT exploited the “shell” facility provided within the Rational Suite testing tool. This allowed all of the automated test scripts (and their component Verification Points) associated with a particular release of the AUT to be assembled into a test shell to be executed together. Each test shell was then assembled into a larger Regression Test shell, which allowed each of the component shells (and their Test Scripts) to be executed sequentially, providing a simple and effective means of scheduling and executing the complete Regression Test (see Figure 18.3).

Following the execution of the Regression Test shell, the automated test tool creates a comprehensive Test Log detailing the results of testing and highlighting defects and areas for further investigation. The Test Log is structured in a hierarchical manner reflecting the hierarchy of test shells and Test Scripts. This allows defects to be tracked down quickly and easily by the QA Analyst responsible for reviewing the results of testing.

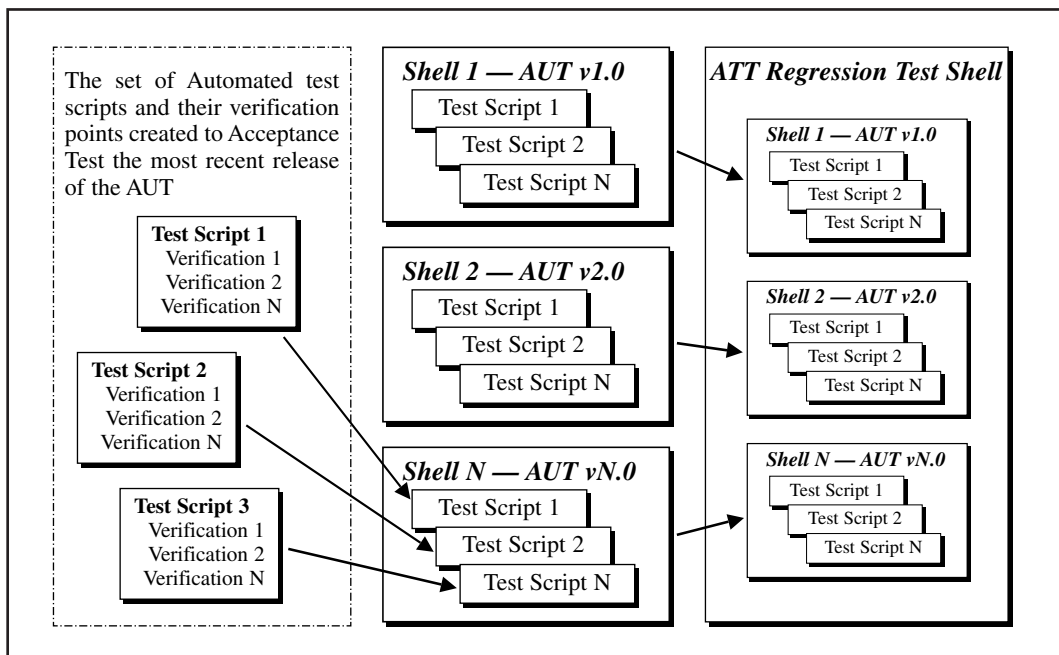


Figure 18.3. Organization of ATT Regression Testing Suite.

³In this context, “test suite” refers to the total collection of Test Scripts that were produced as a result of Acceptance Testing. As a direct consequence of the incremental development approach for the AUT, the test suite would grow in size each time there was a new release of the software.

18.6 Artifacts

The following artifacts are employed by the Acceptance Testing Team in the process of testing the AUT:

- ▲ The ATT Requirements Specification document – generated by the ADP Client Implementation Team, which employed its knowledge and experience of its prospective clients’ business processes to specify the requirements of the system
- ▲ The ATT Test Plan document – describing the testing tasks, their timescales, milestones, dependencies, and risks
- ▲ The ATT Testing Procedures document – this document is equivalent to the traditional Test Specification document described in Part 1 of this book and in Appendix D. The sections covered in this document include:
 - *Introduction* – providing a description of the background and purpose of the functional testing and a list of references to related documents (such as the ADP QA Responsibilities manual)
 - *QA Testing Strategy* – providing a description of the purpose of testing and an overview of roles and responsibilities, testing approach, and scope of functional testing
 - *Responsibilities* – providing the list of tasks to be completed by the ATT during Acceptance Testing and the member of the ATT responsible
 - *Functional Testing* – providing a specification of the functional areas of the AUT to be tested
 - *Acceptance Criteria* – specifying the criteria for delivery of the AUT for testing, defining the categories of severity of defects and priority for correction of defects, and the levels of severity and priority under which the AUT will be deemed acceptable following testing and review of the results.
- ▲ The ATT Performance Testing Specification document – this document is an adjunct to the Acceptance Test Team Testing Procedures document that provides the specification for the performance testing aspects of the Acceptance Testing Project. The sections provided in this document include:
 - *Introduction* – providing a description of the background and purpose of the performance testing and a list of references to related documents (such as the ADP QA Responsibilities manual)
 - *Test Hardware* – specifying precise details of the hardware required to conduct the performance testing of the AUT and of the networking infrastructure
 - *Test Software* – providing a specification of the operating system, networking software, and details of the AUT, as well as a list of additional software required for the purposes of conducting stress and interoperability testing (such as e-mail or word-processing software that could be resident in memory at the same time as the AUT)
 - *Test Data* – describing the need for adequate volumes of data representative of the live data to be used by the AUT and specific hand-crafted data required for exercising boundary and error conditions

- ▲ ATT Business Thread document – a document created by analysis of the software requirements for the system and knowledge of the manner in which the end users employ the AUT to support their business requirements. The Business Thread document describes scenarios to be used in Thread Testing the AUT (see Chapter 3)
- ▲ Automated Test Scripts and Verification Points – planned, designed, and created using the Rational Suite testing tool
- ▲ Manual Test Scripts and Test Cases (where required) – integrated into the Rational Suite testing tool
- ▲ Test Log – automatically generated by the testing tool during the process of executing the automated tests and available after testing for the QA Analyst to review
- ▲ Re-use Pack—for Skim Testing as well as any manual tests required during Acceptance and Regression Testing.

The use of an automated testing tool meant that there was no need for the ATT to produce Test Result Record forms, since in effect these were provided within the tool. Similarly, it was not necessary to produce a Test Summary Report, since the testing tool provided extensive facilities to automatically generate a wide variety of management reports.

18.7 Process Improvement

Because of the size and short duration of the testing project, the ATT did not adopt a formal process improvement program.

However, information on certain aspects of the testing process was collected and used to improve the efficiency of testing during the course of the project:

- ▲ During the testing process, data were collected on the numbers of defects detected and the rate at which they were corrected and signed off. This enabled the ATT to ensure that correction of defects was being completed effectively and that there would be no retesting bottlenecks
- ▲ Testing coverage metrics were also collected, showing what percentage of the AUT had been covered during testing. This information was used to help plan and refine subsequent testing tasks and could be obtained by running predefined reports provided within the Rational Suite testing tool.

Part 3

The Appendices

This section contains a set of standard testing document templates, proformas, and checklists, and a number of appendices that expand on topics described in passing in the main body of the book.

The standard testing document templates, proformas, and checklists are also available from the following URL: us.cambridge.org/titles/052179546X so that they can be used immediately without modification, or customized to reflect the particular requirements of any organization (such as a corporate style, branding, or documentation standard).

Specifically, the appendices contain:

- Appendix A—Terms of Reference for Testing Staff (one-page summary sheets documenting the terms of reference for staff involved in the testing process)
- Appendix B – Testing Guides (one-page summary sheets documenting the key aspects of each testing phase)
- Appendix C – Test Plan Document Template





-
- Appendix D – Test Specification Document Template
 - Appendix E – Test Script Template
 - Appendix F – Test Result Record Form Template
 - Appendix G – Test Log Template
 - Appendix H – Test Certificate Template (for those testing activities that require formal demonstrable proof of the acceptance of the tested software)
 - Appendix I – Re-use Pack Checklist (to assist in setting up and managing testing re-use packs)
 - Appendix J – Test Summary Report Template
 - Appendix K – Equivalence Partition Example (providing a worked example of equivalence partition analysis)
 - Appendix L – Boundary Analysis Example (providing a worked example of boundary analysis)
 - Appendix M – State Transition Example (providing a worked example of state transition analysis)
 - Appendix N – Automated Testing Tool Selection Criteria (providing a list of criteria to be used in the evaluation of automated testing tools and a scoring scheme)
 - Appendix O – Usability Testing Overview
 - Appendix P – Testing Process Health Check (providing a set of criteria and a process for auditing testing projects)
 - Appendix Q – The Testing of Object-Oriented Software (discussing the particular issues facing staff involved in the testing of object-oriented software).



Terms of Reference for Testing Staff

A.1 Introduction

This appendix provides recommendations for the terms of reference (TORs) for the following staff involved in software testing:

- ▲ *Testing Manager*
- ▲ *Test Team Leader*
- ▲ *Test Analyst*
- ▲ *Tester*
- ▲ *Independent Test Observer.*

For each TOR, the following information is documented:

- ▲ Responsibilities
- ▲ Reporting and Liaison
- ▲ Typical characteristics
- ▲ Appropriate references for further information.

Each TOR appears in the following pages of this appendix.

This appendix also describes the role of a typical *Testing Program* Board (see Chapter 4) for organizations interested in introducing such an entity into their testing program.

It is intended that the individual TORs be provided to the appropriate staff on testing projects, either as photocopies or electronically, as memory aids.

Testing Manager

Responsibilities

The Testing Manager is responsible for:

- 1. Setting up and resourcing new testing projects
- 2. Tasking one or more Test Team Leaders and monitoring their progress against agreed-on plans
- 3. Ensuring that (where appropriate) the development teams follow the *Unit* and *Integration* testing approach documented within the testing process
- 4. Assuring that (where appropriate) software development conducted by a third party follows good development and testing practice
- 5. Formally representing the organization during the co-signing of the Test Certificate (Appendix H) with the representative of a third party responsible for development of software
- 6. Maintenance of the testing program filing system
- 7. Reporting progress and issues to senior management at agreed-on intervals
- 8. Liaising with Independent Test Observers to ensure good practice has been followed in testing.

Reporting and Liaison

Figure A.1 shows the lines of reporting and liaison for the Testing Manager.

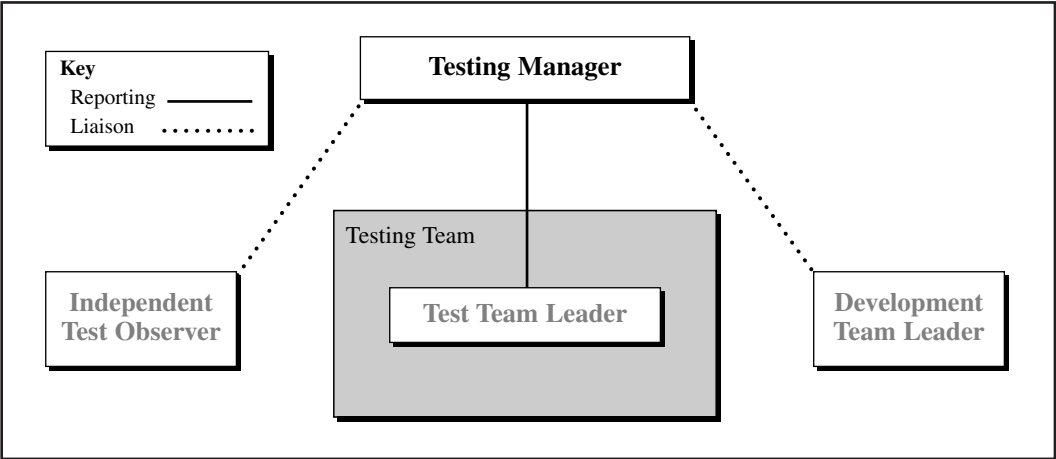


Figure A.1. Lines of reporting and liaison for the Testing Manager.

Characteristics of a Testing Manager

The characteristics of a Testing Manager include:

1. Good project management skills
2. Good communications skills, including negotiation skills and diplomacy
3. Good appreciation of the software development lifecycle and the need for testing
4. Excellent project planning and monitoring skills across multiple projects and staff
5. Capability of working under pressure and being able to motivate staff.

References

1. Chapter 4 – The Management of Testing
2. Reference 3 (PRINCE 2 Manual)

Test Team Leader

Responsibilities

The Test Team Leader is responsible for:

1. Generating the Test Plan (Appendix C) and *Test Specification* documents (Appendix D)
2. Tasking one or more Test Analysts and monitoring their progress against agreed-on plans
3. Tasking one or more Testers and monitoring their progress against agreed-on plans
4. Liaising with Development Teams (e.g., to determine availability date of the *AUT*)
5. Liaising with the Independent Test Observer, and User and Operations representatives
6. The set-up and maintenance of the testing project filing system
7. Reporting progress and issues to the Testing Manager at agreed intervals
8. Performing other ad hoc tasks as specified by the Testing Manager
9. (Potentially) managing multiple concurrent test plans across multiple *AUTs*.

Reporting and Liaison

Figure A.2 shows the lines of reporting and liaison for the Test Team Leader.

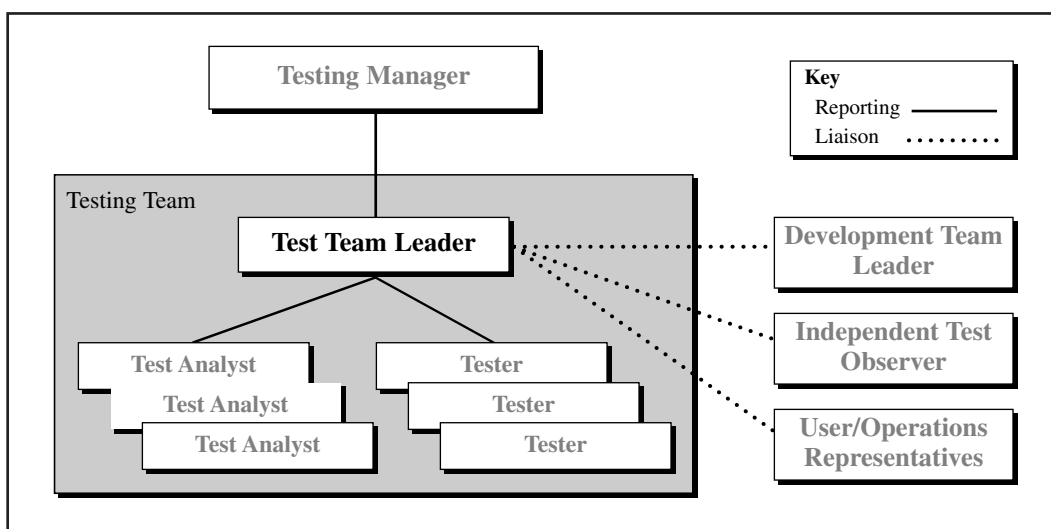


Figure A.2. Lines of reporting and liaison for the Test Team Leader.

Characteristics of a Test Team Leader

The characteristics of a Test Team Leader include:

1. Good communications skills, including negotiation skills, diplomacy, and a sense of humor
2. Good appreciation of the software development lifecycle and the need for testing
3. Good project planning and monitoring skills
4. A pragmatic and practical approach
5. Capable with working under pressure and being able to motivate staff
6. Experience of automated test management tools desirable.

References

1. Chapter 4 – The Management of Testing
2. Reference 3 (PRINCE 2 Manual)

Test Analyst

Responsibilities

The Test Analyst is responsible for:

- 1. Assisting the Test Team Leader in generating the Test Specification and *Test Plan Documents*
- 2. Performing testing tasks as specified by the Test Team Leader
- 3. Assisting the Test Team Leader with functional analysis of the *AUT*
- 4. Defining the Test Requirements (with respect to the *Functional Specification*)
- 5. Designing and implementing *Test Cases* and *Test Scripts* (Appendix E)
- 6. Designing and implementing *Test Data Sets*
- 7. Briefing the Tester for the AUT prior to testing
- 8. The generation of *Re-use Packs* (Appendix I)
- 9. Backup and archival of all testing documentation and materials
- 10. Completion of the Test Summary Report (Appendix J).

Reporting and Liaison

Figure A.3 shows the lines of reporting and liaison for the Test Analyst.

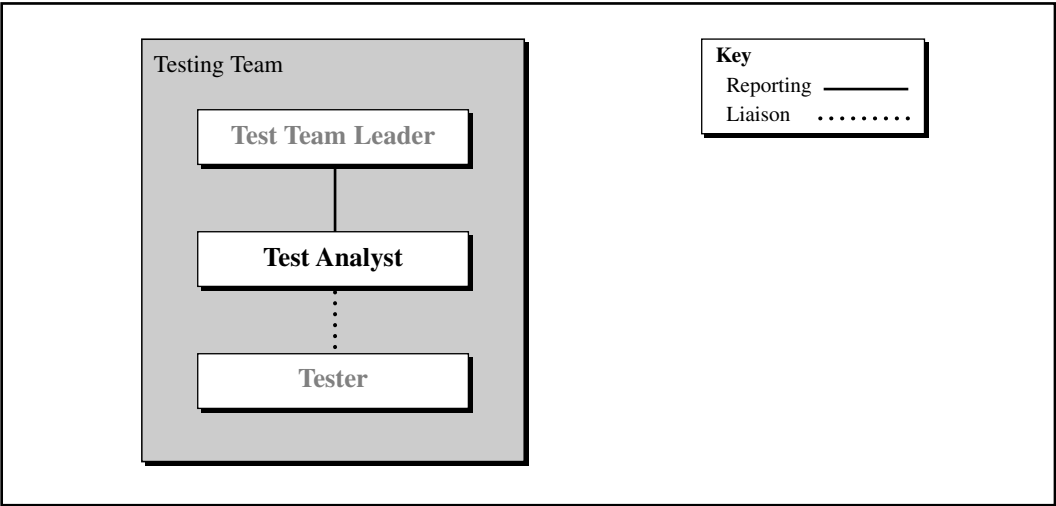


Figure A.3. Lines of reporting and liaison for the Test Analyst.

Characteristics of a Test Analyst

The characteristics of a Test Analyst include:

1. Good team player with good communications skills
2. Good appreciation of the software development lifecycle and the need for testing
3. Good experience with testing and the testing process
4. A concise and structured way of working
5. Capability of completing tasks independently to an agreed plan
6. Capable of working under pressure
7. Experience with automated testing tools desirable.

Reference

1. Chapter 4 – The Management of Testing

Tester

Responsibilities

The Tester is responsible for:

- 1. Performing testing tasks as specified by the Test Team Leader
- 2. Executing Test Scripts (Appendix E)
- 3. Observation and documentation of test results on *Test Result Record Forms* (Appendix F)
- 4. Maintenance and retention of Test Result Record forms
- 5. Identification and logging of any observed faults
- 6. Creation of tests for retest of fault fixes
- 7. Set-up and initialization of the test bed
- 8. Backup and archival of the test bed
- 9. Recovery of the test bed in the event of failure.

Reporting and Liaison

Figure A.4 shows the lines of reporting and liaison for the Tester.

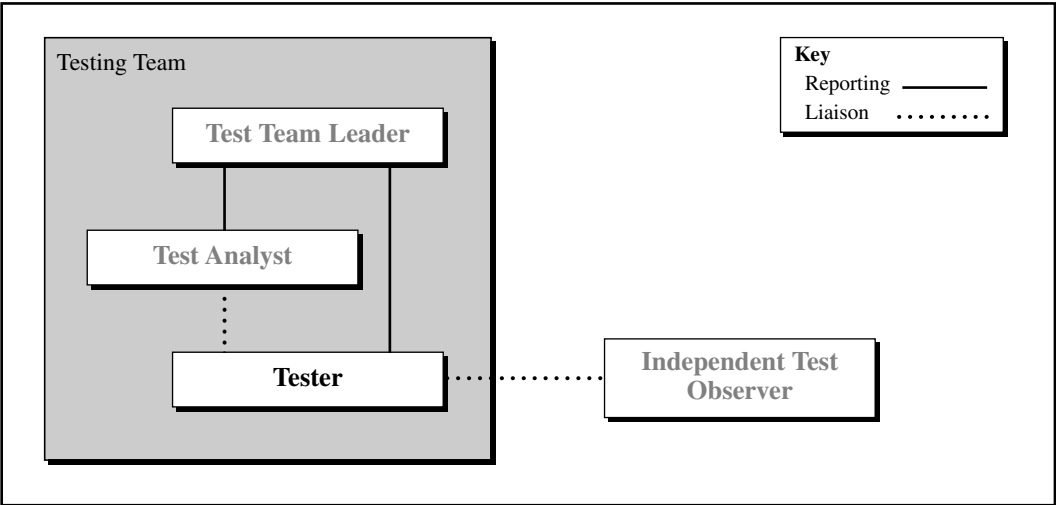


Figure A.4. Lines of reporting and liaison for the Tester.

Characteristics of a Tester

The characteristics of a Tester include:

1. Good team player
2. Good appreciation of the need for testing
3. A concise and structured way of working
4. Capable of completing tasks independently to an agreed plan
5. Capable of working under pressure
6. Experience with automated testing tools desirable.

Reference

1. Chapter 4 – The Management of Testing

Independent Test Observer

Responsibilities

The Independent Test Observer should be a staff member capable of providing independent observation of the testing project (such as a member of a *Quality Assurance (QA)* group if one exists in the organization). The Independent Test Observer is responsible for:

1. Attending the testing of the AUT
2. Formally witnessing that the Tester correctly completes all Test Scripts and Test Cases
3. Signing the appropriate section of the Test Result Record Form (Appendix F)
4. Liaising with the Testing Manager to report any problems observed during the testing process (or *Testing Project* Board if appropriate).

Reporting and Liaison

Figure A.5 shows the lines of reporting and liaison for the Independent Test Observer.

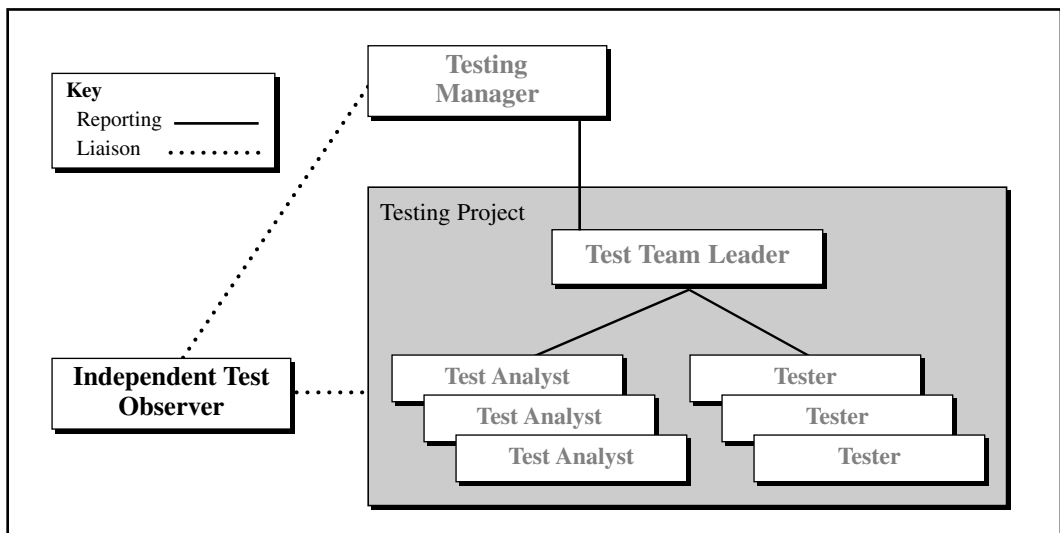


Figure A.5. Lines of reporting and liaison for the Independent Test Observer.

Characteristics of an Independent Test Observer

The characteristics of an Independent Test Observer include:

1. Capable of genuinely independent assessment of the testing project and testing activities
2. Good communications skills, including assertiveness, diplomacy, and negotiation skills
3. Good appreciation of the need for testing
4. Good appreciation of the need for independent verification of the testing process
5. Good understanding of quality and the quality process.

References

1. Chapter 4 – The Management of Testing
2. Reference 3 (PRINCE 2 Manual)
3. Reference 14 (ISO 9001 and 9003)

Testing Program Board

Responsibilities

Some organizations (such as those following the PRINCE project management approach) may decide to introduce the notion of a Testing Program Board into their testing process.

A Testing Program Board represents the user and developer interests of individual testing projects at senior managerial level (see Reference 3). The Testing Program Board is likely to be composed of senior decision-makers within the organization and is responsible for the commitment of resources to the project (such as User Representatives).

The level of seniority of managers required to fill the roles will depend on such factors as the budget and the importance of the testing program. The Testing Program Board members' responsibilities are additional to their normal tasks, and the use of "management by exception" (that is, keeping the Board members informed of events, requiring joint decision-making only at key points in the project) is recommended.

The members of the Testing Program Board typically fulfill the following roles:

- ▲ The Executive, who is ultimately accountable for the success of the testing program, and for ensuring that individual testing projects are conducted efficiently, balancing the demands of the business, user, and developers/suppliers
- ▲ The Senior User, who is responsible for ensuring that the results of the testing projects yield a system that is fit for purpose, meets the user's needs, and falls within the constraints of the business case for the AUT. This role represents the interests of all staff (including operators) who will use the AUT and has the authority to commit user resources (such as user representatives for Acceptance Testing)
- ▲ The Senior Supplier, who is responsible for ensuring that proposals for testing (such as test plans and the scope of testing) are realistic in terms of achieving the results required by the users within the cost and time parameters for which senior management is accountable. This role represents the interests of staff members designing, developing, procuring, or maintaining the AUT.

The term "Testing Program Board" is synonymous with the PRINCE 2 term "Project Board" (see Reference 3 for further details).

Reporting and Liaison

Figure A.6 shows the lines of reporting and liaison for the Testing Program Board:

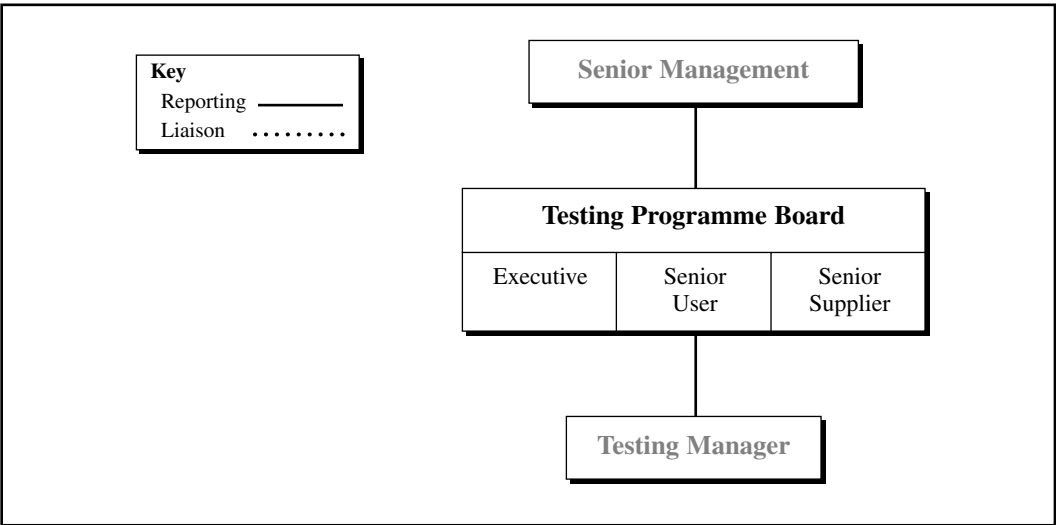


Figure A.6. Lines of reporting and liaison for the Testing Program Board.

Reference

- 1. Reference 3 (PRINCE 2 Manual)

B

Testing Guides

B.1 Introduction

This appendix contains a number of one-page testing guides that can be used as a memory aid for staff involved in the various phases of software testing.¹ Full details of each of the testing phases covered in this appendix are provided in Chapters 5–11.

In particular, the guides are intended to be used by *Testers* (whether they are members of the *Development Team* (in *Unit* and *Integration Testing*), of the *Testing Team*, or of the QA Group).

This appendix contains the following testing guides:

- ▲ Unit Testing Guide
- ▲ Integration Testing Guide
- ▲ System Testing Guide
- ▲ Systems Integration Testing Guide
- ▲ User/Operations Acceptance Testing Guide
- ▲ Regression Testing Guide.

For each testing phase, the following summary information is provided:

- ▲ Purpose – a brief description of the purpose of this testing phase
- ▲ Approach – listing the high-level approach to conducting this testing phase
- ▲ Inputs – listing the inputs to this testing phase
- ▲ Testing Techniques – listing the testing techniques appropriate to this testing phase
- ▲ Outputs – listing the outputs from this testing phase
- ▲ Quality Considerations – listing a number of quality issues that should be considered within this testing phase
- ▲ References – further sources of information on this testing phase.

It is intended that the individual testing guides be provided to the appropriate staff on testing projects, either as photocopies or electronically for reference purposes.

¹The testing guides will be of particular use to Testers.

Unit Testing Guide

Purpose of Unit Testing

The purpose of *Unit Testing* is to produce reliable program units that meet their requirements. The process is primarily intended to identify errors in program logic.

Approach

1. Review Unit Test Plan.
2. Execute and check all Unit Test Scripts and complete the Test Result *Record Form* (Appendix F).
3. Ensure that each completed Test Result Record Form is countersigned by the *Independent Test Observer*.
4. Complete the Unit *Test Log* during the course of the testing phase (Appendix G).
5. On completion of Unit Testing, provide the *Development Team Leader* with completed Test Result Record Forms, the Unit Test Log, and the Unit Test Summary Report (Appendix J).

Inputs

1. Unit Testing Plan (from the Development Team Leader)
2. Unit Test Scripts and Test Cases (as designed by the Test Analyst)
3. Blank Test Result Record Forms (from Appendix F of the Testing Framework document)
4. Any relevant *Re-use Packs* from previous Unit Testing (from the project file)

Testing Techniques (see Chapter 3)

1. *Functional Testing* against the requirements for the unit under test
2. *Static Testing* (such as code review and code walk-through)
3. *White Box Testing* (the structure of the unit should be known to the developer)
4. *State Transition Testing* (particularly where it is possible for the unit to be in a number of different states and for negative testing of the unit)
5. *Nonfunctional Testing* (where appropriate; e.g., for Performance, Stress, or Reliability Testing of the unit)

Outputs

1. Fully tested units
2. Revised Test Cases (where appropriate)

3. Archived test data
4. Completed Test Result Record Forms
5. Completed Unit Test Log
6. Completed Unit Test Re-use Pack (Appendix I)
7. Completed Unit Test Summary Report (Appendix J)

Quality Considerations

1. The use of independent observation of unit tests and traceability of testing documentation (such as signed-off Test Result Record Forms) is desirable.
2. Where a particular unit demonstrates serious defects on testing (and on retesting), consider deleting the existing unit and rewriting it (statistics show that units [and modules] that have demonstrated poor quality during Unit and Integration Testing are extremely likely to continue to demonstrate defects in live running).
3. Ensure that adequate contingency is included in the development plans for both testing and retesting.

References

1. Chapter 5 (Unit Testing)
2. Chapter 3 (Testing Techniques)
3. Chapter 4 (The Management and Planning of Testing)
4. Glossary

Integration Testing Guide

Purpose of Integration Testing

The purpose of *Integration Testing* is to demonstrate that the modules which comprise the *Application Under Test (AUT)* interface and interact in a stable and coherent manner prior to *System Testing*.

Approach

1. Review Integration Test Plan.
2. Execute and check all Integration Test Cases and complete the Test Result Record Form (Appendix F).
3. Ensure that each Completed Test Result Record Form is countersigned by the *Test Observer*.
4. Complete the Integration Test Log (Appendix G).
5. On completion of Integration testing, provide the *Development Team Leader* with completed Test Result Record forms, Integration Test Log, and the Integration Test Summary Report (Appendix J).

Inputs

1. Integration Testing Plan (from the Development Team Leader)
2. Integration Test Script and Test Cases (as designed by the Test Analyst)
3. Blank Test Result Record Forms
4. Re-use Packs from the Unit Testing phase (from the project file)

Testing Techniques (see Chapter 3)

1. *Functional Testing* using *Black Box Testing* techniques against the interfacing requirements for the module under test.
2. *Nonfunctional Testing* (where appropriate; e.g., for performance or reliability testing of the module interfaces)
3. Where appropriate, some Functional Testing against relevant intermodule functionality (again using Black Box Testing techniques).

Outputs

1. Fully tested and integrated modules.
2. Revised Test Cases (where appropriate)
3. Archived test data
4. Completed Test Result Record Forms
5. Completed Integration Test Log

6. Completed Integration Test Re-use Pack (Appendix I)
7. Completed Integration Test Summary Report (Appendix J)

Quality Considerations

1. Where a particular module demonstrates serious defects on testing (and on retesting), consider re-writing the module (statistics show that modules which have demonstrated poor quality during Integration Testing are extremely likely to continue to demonstrate defects in live running).
2. For reliability testing, consider how representative the test environment is of the live system (e.g., has the volume of background network traffic been simulated accurately?).

References

1. Chapter 6 (Integration Testing)
2. Chapter 3 (Testing Techniques)
3. Chapter 4 (The Management and Planning of Testing)
4. Glossary

System Testing Guide

Purpose of System Testing

The purpose of *System Testing* is to establish confidence that the *Application Under Test (AUT)* will be accepted by the users (and/or operators) and to avoid errors in the live system that will have an adverse impact on the business from a functional, financial, or image perspective.

Approach

1. Review System Test Plan.
2. Execute and check all System Test Cases, and complete the Test Result Record Form (Appendix F).
3. Ensure that each completed Test Result Record Form is countersigned by the *Independent Test Observer*.
4. Complete the System Test Log (Appendix G).
5. On completion of System Testing, provide the *Test Team Leader* with completed Test Result Record forms, the System Test Log, and the System Test Summary Report (Appendix J).

Inputs

1. System Testing Plan (from the Test Team Leader)
2. System Test Script and Test Cases (from the Test Analyst)
3. Blank Test Result Record Forms
4. Re-use Packs from the Unit and Integration Testing phases (from the project file)

Testing Techniques (see Chapter 3)

1. *Black Box Testing* against high-level system requirements
2. *Thread Testing* against the high-level business requirements for the AUT
3. *Nonfunctional Testing* (such as Volume, Stress, and Performance Testing)
4. *Static Testing* (e.g., review of system documentation, such as user manuals)

Outputs

1. The tested system
2. Revised Test Cases (where appropriate)
3. Archived test data

4. Completed Test Result Record forms
5. Completed System Test Log
6. Completed System Test Re-use Pack (Appendix I)
7. Completed System Test Summary Report (Appendix J)

Quality Considerations

1. Has good development process/methodology been employed?
2. Is the hardware/operating system/network software mature and robust?
3. Is the quality/quantity of the test data adequate?
4. Where appropriate, has it been possible to simulate appropriate load/stress/volume that the live system will be exposed to?

References

1. Chapter 7 (System Testing)
2. Chapter 3 (Testing Techniques)
3. Chapter 4 (The Management and Planning of Testing)
4. Glossary

Systems Integration Testing Guide

Purpose of Systems Integration Testing

The purpose of *Systems Integration Testing* is to provide confidence that the *Application Under Test (AUT)* is able successfully to interoperate with other specified systems and does not have an adverse effect on other systems that may also be present in the live operating environment, or *vice versa*.

Approach

1. Review Systems Integration Test Plan.
2. Execute and check all Systems Integration Test Cases, and complete the Test Result Record Form (Appendix F).
3. Ensure that each Completed Test Result Record Form is countersigned by the *Test Observer*.
4. Complete the Systems Integration Test Log (Appendix G).
5. On completion of Systems Integration Testing, provide the *Test Team Leader* with completed Test Result Record Forms, the Systems Integration Test Log, and the Systems Integration Summary Report (Appendix J).

Inputs

1. Systems Integration Testing Plan (from the Test Team Leader)
2. Systems Integration Test Script and Test Cases (from the *Test Analyst*)
3. Blank Test Result Record forms
4. The Re-use Packs from the *Unit*, *Integration*, and *System Testing* phases (from the project file)

Testing Techniques (see Chapter 3)

1. *Black Box Testing* against high-level system requirements (and specifically, those requirements addressing interoperability issues)
2. *Thread Testing* against the high-level business requirements for the AUT (again, focusing on interoperability requirements)
3. *Negative Testing* and *Error Guessing* to uncover any unanticipated problems (such as contention for system resources)

Outputs

1. The fully tested and integrated system
2. Revised Test Cases (where appropriate)
3. Archived test data

4. Completed Test Result Record Forms
5. Complete Systems Integration Test Log
6. Complete Systems Integration Test Re-use Pack (Appendix I)
7. Complete Systems Integration Test Summary Report (Appendix J)

Quality Considerations

1. Systems Integration Testing must take place in the live environment (using a separate test environment should be considered only if there are pressing reasons not to use the live system and if the alternate environment can *accurately* represent the live system).
2. Consider tool use to monitor system resource usage (e.g., to identify contention for resources).
3. Ensure that the Test Analyst is both experienced and knowledgeable about the AUT and the live system (this is particularly valuable in respect of negative testing).

References

1. Chapter 8 (Systems Integration Testing)
2. Chapter 3 (Testing Techniques)
3. Chapter 4 (The Management and Planning of Testing)
4. Glossary

User/Operations Acceptance Testing Guide

Purpose of Acceptance Testing

The purpose of *Acceptance Testing* is to confirm that the system meets its business requirements and to provide confidence that the system works correctly and is usable before it is formally “handed over” to the user (either the *end user* or the *operations user*).

Approach

1. Review Acceptance Test Plan.
2. Execute and check all Acceptance Test Cases and complete the Test Result Record Form (Appendix F).
3. Ensure that each Completed Test Result Record Form is countersigned by the *Test Observer*.
4. Complete the Acceptance Test Log (Appendix G).
5. On completion of Acceptance Testing, provide the *Test Team Leader* with completed Test Result Record Forms, the Acceptance Test Log, and the Acceptance Test Summary Report (Appendix J).

Inputs

1. Acceptance Testing Plan (from the Test Team Leader)
2. Acceptance Test Script and Test Cases (from the *Test Analyst*)
3. Blank Test Result Record forms
4. Re-use Pack from the *Systems* and *Integration Testing* phases (from the project file)

Testing Techniques (see Chapter 3)

1. *Black Box Testing* against high-level system requirements
2. *Thread Testing* against the high-level business requirements for the AUT
3. *Static Testing* (e.g., review of system documentation, such as user manuals)

Outputs

1. The User accepted system
2. Revised Test Cases (where appropriate)
3. Archived test data
4. Completed Test Result Record Forms
5. Completed Acceptance Test Log

6. Completed Acceptance Test Re-use Pack (Appendix I)
7. Completed Acceptance Test Summary Report

Quality Considerations

1. Re-use of System Test materials (e.g., System Test Cases) wherever possible
2. User representatives chosen with appropriate experience and skills

References

1. Chapters 9 (User Acceptance) and 10 (Operations Acceptance)
2. Chapter 3 (Testing Techniques)
3. Chapter 4 (The Management and Planning of Testing)
4. Glossary

Regression Testing Guide

Purpose of Regression Testing

The purpose of *Regression Testing* is to ensure that the Application Under Test (AUT) still functions correctly following modification or extension of the system (such as user enhancements or upgrades or following new builds or releases of the software).

Approach

1. Review Regression Test Plan.
2. Execute and check all Regression Test Cases and complete the Test Result Record Form (Appendix F).
3. Ensure that each Completed Test Result Record Form is countersigned by the *Test Observer*.
4. Complete the Regression Test Log (Appendix G).
5. On completion of regression testing, provide the Test Team Leader with completed Test Result Record Forms, the Regression Test Log, and the Regression Test Summary Report (Appendix J).

Inputs

1. Regression Testing Plan (from the Test Team Leader)
2. Regression Test Script and Test Cases (from the Test Analyst)
3. Blank Test Result Record forms
4. Re-use Packs from previous testing phases (from the project file)

Testing Techniques (see Chapter 3)

1. *Black Box Testing* against high level system requirements
2. *Thread Testing* against the high level business requirements for the AUT
3. *Nonfunctional Testing* (such as *Volume*, *Stress*, and *Performance Testing*), where appropriate.

Outputs

1. The tested system
2. Revised Test Cases (where appropriate)
3. Archived test data
4. Completed Test Result Record forms
5. Completed Regression Test Log
6. Completed Regression Test Re-use Pack (Appendix I)

7. Completed Regression Test Summary Report

Quality Considerations

1. Re-use of Test Cases from earlier testing phases (and in particular from System Testing) is key.
2. Test expertise must be employed to anticipate the impact of the effect of enhancements/extensions to the AUT and its existing functionality.
3. Consider the role of automated testing tools in performing Regression Testing.

References

1. Chapter 11 (Regression Testing)
2. Chapter 3 (Testing Techniques)
3. Chapter 4 (The Management and Planning of Testing)
4. Glossary



Test Plan Document Template

C.1 Introduction

This appendix contains a template for a *Test Plan Document*, guidance on its use, and where appropriate, examples of the typical contents of such a document. This template may be used as the basis of a Test Plan Document for a particular test phase.

As described in Chapter 4 (The Management and Planning of Testing), each testing project must generate a Test Plan as one of the documents produced prior to the commencement of the testing activities.

Where text appears within the body of the template in angle brackets (< >), this denotes a place marker, which must be replaced with the appropriate information for a particular testing phase.

Text that appears in *italic* type is provided for information only (such as illustrative examples), and should not appear in the final Test Plan Document.

Where references to “Appendices” appear within this appendix, these refer to other appendices in this book. The term “Annex” is used to refer to supplementary sections within this template.

The principal approach to be used in writing a Test Plan Document is to keep it brief and simple. Make best use of the supporting information presented in this book (such as the other relevant template appendices), as well as pertinent material from the appropriate testing phase chapters.

An electronic version of the template is available from the following URL: <us.cambridge.org/titles/052179546X> (or, assuming the role has been implemented, a copy may be obtained from the *Testing Manager* or equivalent role).

TEST SPECIFICATION DOCUMENT FRONT SHEET

Document Information {to <Client> Doc Standard}	
Project ID:	<Project ID> <i>The unique ID for this testing project</i>
Document Ref:	<Doc Ref> <i>A unique document reference for this document</i>
Testing Phase:	<Testing Phase> <i>The testing phase (e.g., Unit Test)</i>
AUT Title:	<AUT Title> <i>The definitive title of the application under test</i>
Date:	<Date> <i>The date this document was completed</i>

Distribution	
Copy Number	Recipient
1.	<activity leader> <i>That is, Development Team Leader for Unit and Integration Testing, Test Team Leader for System Test, and so on</i>
2.	<Test Analyst> <i>That is, the person designing and developing the Test Cases</i>
N.	<Date> <i>The date this document was completed</i>

Review & Approval	
Issue	<Issue status> <i>Issue status of the document (e.g. Draft, v1.0)</i>
R&A Number:	<R&A reference> <i>The reference to the final approving review</i>
Author	<name of the author> <i>The person who wrote this document</i>
Author Signature	<signature of author>
Approval (PM)	<name of the Project Manager>
Approval (PM) Signature	<signature of the Project Manager >
Approval (QA)	<name of the Quality Assurance representative>
Approval (QA) Signature	<signature of the Quality Assurance representative>

CONTENTS

- 1 Introduction**
 - 1.1 Background**
 - 1.2 Structure of the Document**
 - 1.3 References**
- 2 Test Approach and Constraints**
 - 2.1 Introduction**
 - 2.2 Test Objectives**
 - 2.3 Test Constraints**
- 3 Risks and Dependencies**
 - 3.1 Introduction**
 - 3.2 Risks**
 - 3.3 Dependencies**
- 4 Test Assumptions and Exclusions**
 - 4.1 Introduction**
 - 4.2 Assumptions**
 - 4.3 Exclusions**
- 5 Entry and Exit Criteria**
 - 5.1 Introduction**
 - 5.2 Entry Criteria**
 - 5.3 Exit Criteria**
- 6 Testing Project Controls**
 - 6.1 Introduction**
 - 6.2 Roles and Responsibilities**
 - 6.3 Training Requirements**
 - 6.4 Problem Reporting**
 - 6.5 Progress Reporting**

Annex A – Test Plan

1 Introduction

1.1 Background

This document provides the plan for completing the testing activities required for the <Testing Phase> testing of the <AUT Title>.

This section should contain a brief background and history of this testing project, which should include:

- (a) *a brief description of the purpose of the AUT*
- (b) *definitive version information for the AUT*
- (c) *any other relevant supporting information*

1.2 Structure of the Document

This document is structured in the following manner:

- (a) Chapter 2 describes the approach to testing, including the objectives of the project and any constraints.
- (b) Chapter 3 describes the risks and dependencies associated with the testing project.
- (d) Chapter 4 lists the test assumptions and exclusions.
- (e) Chapter 5 documents the entry and exit criteria for testing.
- (f) Chapter 6 lists the project controls to be employed in the testing project.
- (h) Annex A provides a <type of plan> plan of the testing project *{such as a Man Power and Milestone plan or a Gantt Chart. If a Project Management or Planning tool is being used on the project, it may be possible to obtain a hard copy of the plan to insert into the Annex}.*

1.3 References

A numbered list of documents referenced within this document. For example, <list of references to Overall Project Documents>

- (1) <Project authorization document>
- (2) <Overall Project Plan>
- (3) <Quality Assurance Plan>
- (4) <Configuration Management Plan>
- (5) <Relevant Policies referenced within this document>
- (6) <Relevant Standards referenced within this document>
 <List of references to items specific to this testing project>
- (7) <AUT Title> Functional Specification (version N.N), <date>
- (8) <AUT Title> Design Specification (version N.N), <date>
- (9) <AUT Title> User's Guide (version N.N), <date> (if appropriate)
- (10) <AUT Title> Operations Guide (version N.N), <date> (if appropriate)
- (11) <AUT Title> Installation Guide (version N.N), <date> (if appropriate).

2 Test Approach and Constraints

2.1 Introduction

This chapter describes the overall approach that will be used in testing the AUT and any constraints that may apply. These will be used at a high level to support the planning of the testing project.

This chapter also documents the use of any particular testing tool (such as an automated record/playback tool) and its details (version number, release information, etc.), or project support tool (such as a project planning or project metrics tools).

2.2 Test Objectives

This section lists the high-level objectives of the testing project. *For example, in Regression Testing a particular AUT, the objectives might include "The Regression Testing process will specifically consider just the functionality of the AH51 module modified during the extensions to the system, and any interfaces between this module and other related modules."*

It will also be appropriate to list any exclusions. For example, in Regression Testing a particular AUT, the objectives might include "The regression testing will not address any load or performance issues."

2.3 Test Constraints

This section lists any constraints that may apply to the testing project. *For example, in Regression Testing a particular AUT, the objectives might include "Regression testing must be complete (including correction of errors and retesting) by the Nth of October 2005 for scheduled release to beta test client site."*

3 Risks and Dependencies

3.1 Introduction

This chapter documents the risks and dependencies that apply to this testing project. These will be used at a high level to support the planning of the testing project.

3.2 Risks

This section documents risks that may have a deleterious impact on the testing project. The occurrence, impact, and likelihood of these risks must be reviewed during the course of the testing project, and procedures for mitigation/recovery must be documented.

The level of rigor employed in the process of identifying risks will depend on a number of factors, including:

- (a) *the business criticality of the AUT*
- (b) *safety criticality of the AUT*
- (c) *whether the AUT is to be used in-house or supplied to a client.*

Although typically the process of identifying risks to the testing project is conducted in an informal manner, relying on the experience of the testing staff (and in particular the Test Team Leader/Development Leader), a number of tools and methods are available (see Reference 11[of this book] for example).

For example, in Regression Testing a particular AUT, the following risk might have been identified: "It is possible that the latest version of the operating system that was planned to be used for testing the AUT will not be available from the manufacturer in time for the planned testing start date. In the event that this occurs, testing will proceed using the current version of the operating system software, with a subset of Regression Tests to be rerun as and when the new version of the operating system becomes available."

3.3 Dependencies

This section documents any events that must take place as a prerequisite to any of the planned events within the testing project. The progress of these dependencies must be reviewed during the course of the testing project to determine their impact on the testing project timescales.

Dependencies can be categorized as:

- (a) *internal – a dependency originating within the testing project.*
- (b) *external – a dependency originating outside of the testing project.*

For example, in Regression Testing a particular AUT, an internal dependency might be "The start of the testing process by the Tester is dependant on receipt of the Test Script and Test Cases from the Test Analyst."

An external dependency might be "The testing start date is dependent on the delivery by the software development team of version N.N of the AUT."

4 Test Assumptions and Exclusions

4.1 Introduction

This chapter provides greater detail about the testing process and in particular the assumptions that have been made for the purposes of the testing project, precisely what issues will be addressed by the testing, and what issues will not be addressed.

4.2 Assumptions

This section lists the assumptions that have been made for the purposes of the testing project. It also lists the specific aspects of the AUT to be tested.

For example, in Regression Testing a particular AUT, the following assumptions have been made:

- (a) *The testing project will test:*
 - *the functionality of the AH51 module*
 - *the interface between module AH51 and AH52*
 - *the interface between module AH51 and AH53.*
- (b) *The database server will be live during the entire testing process.*
- (c) *Access will be available to the test rig outside of normal working hours.*

4.3 Exclusions

This section lists the items that will not be included in the testing project.

For example, in Regression Testing a particular AUT, the following exclusions have been made:

- (a) *The testing project will not test:*
 - *performance aspects of the AH51 module*

- load aspects of the AH51 module
- the interface between the AH51 module and the AH50 module
- (b) *It is not necessary for a User Representative to be present during testing*
- (c) *Testing tool support will not be necessary for this testing project.*

5 Entry and Exit Criteria

5.1 Introduction

This chapter documents the criteria that must be satisfied before testing can begin as well as the criteria to be satisfied in order for testing to finish. This information is recorded to provide a clear and unambiguous means of determining when to start or stop testing.

5.2 Entry Criteria

This section lists the entry criteria for the testing project.

For example, in Regression Testing a particular AUT, the following entry criteria have been specified:

- (a) *The test environment has been set up (and tested).*
- (b) *The amended (to reflect changes to the AUT) Requirements Specification Document has been received.*
- (b) *The final copy of the amended AUT has been received from the Development Team and has been installed in the test environment.*
- (c) *The latest version of the operating system required for testing has been installed in the test environment.*
- (d) *The test project has been appropriately resourced.*

5.3 Exit Criteria

This section lists the exit criteria for halting the testing process.

For example:

- (a) *no outstanding defects*
- (b) *observed defects below an agreed-on profile, such as:*
 - *no critical defects, and less than N severe defects*
 - *N hours testing time without observing a defect*
 - *a complete run (or rerun) through the test script with no defects*
- (c) *some budgetary criterion, such as:*
 - *the assigned testing budget has been exceeded*
 - *the assigned testing effort had been exceeded*
- (d) *some scheduling criterion, such as:*
 - *testing must stop to allow planned delivery of AUT*
 - *testing must stop to allow planned development of AUT (e.g., Unit Testing must be complete by a certain date to allow integration of the AUT to begin)*
- (e) *testing must stop because the AUT is unfit for this level of testing; that is, the AUT is of such poor quality that further testing would be a waste of effort).*

6 Testing Project Controls

6.1 Introduction

This chapter documents the staff members involved in the testing project and their roles and responsibilities. It also describes the manner in which problems and progress will be reported.

The precise details for each of the following sections will need to be tailored to match the particular testing phase (see the appropriate testing phase chapter of this book) and the specific testing requirements for the AUT (see Test Specification Document, Appendix D).

6.2 Roles and Responsibilities

This section specifies the names, roles, and responsibilities of the staff involved in the testing project.

For example, for a System Testing project, the following roles would need to be documented:

- (a) *Test Team Leader – Carol Jenkins*
- (b) *Test Analyst – Martin Stemp*
- (c) *Tester – Chris Dodsworth*
- (d) *User Representative – Angela Jones*
- (e) *Independent Test Observer – Lesley Williams.*

6.3 Training Requirements

This section documents any particular training needs for the staff involved in the testing.

For example, the Test Analyst and Tester may require training with the AUT to gain the familiarity with the user interface to improve their effectiveness during the design and execution of the tests.

6.4 Problem Reporting

This section documents the manner in which defects observed during testing are recorded and reported to the appropriate staff.

For example, all defects observed during System Testing will be recorded on the appropriate Test Result Record Form (see Appendix F). All such defects will be reviewed during the Post Test Review Meeting, where the Test Team Leader and the Development Team Leader will discuss the scope of remedial work to the AUT, timescales for correction, and any test case(s) that need to be rerun as a result of the remedial work.

If a test-management tool or a defect-reporting tool is being used within a particular testing project, the tool details and the process by which it will be used should also be documented here.

6.5 Progress Reporting

This section documents the frequency, scope, and detail of testing project progress reporting and describes the staff involved in the process.

For example, the Test Team Leader will issue a weekly progress report, which will include:

- (a) *planned versus actual progress*
- (b) *revised estimates and their justification*
- (c) *any outstanding problems and issues.*

The report will be issued to:

- (a) *Testing Manager*
- (b) *Test Analyst*
- (c) *Tester*
- (d) *Independent Test Observer.*

The Test Team Leader will convene a progress meeting every other week to report on the testing project progress. The following staff should be invited:

- (a) *Testing Manager*
- (b) *Development Team Leader*
- (c) *the User Representative*
- (d) *the Independent Test Observer.*

A Test Plan

This annex contains a <type of plan> plan of the testing project {such as a Man Power and Milestone plan or a Gantt Chart. If a Project Management or Planning tool is being used on the project, it may be possible to obtain a hard copy to insert into the Annex}.

As a minimum, the <type of plan> should include the following items:

- (1) *task names*
- (2) *timescales for each task*
- (3) *risks and dependencies*
- (4) *brief deliverable names*

The V Model (Chapter 4) can be used to provide useful guidance during the planning aspects of the testing project.

D

Test Specification Document Template

D.1 Introduction

This appendix contains a template for a *Test Specification Document*, guidance on its use, and where appropriate, examples of the typical contents of such a document. This template may be used as the basis of a Test Specification document for a particular test phase.

As described in Chapter 4 (The Management and Planning of Testing), each testing project must generate a Test Specification as one of the documents produced prior to the commencement of the testing activities.

Where text appears within the body of the template in angle brackets (< >), this denotes a place marker, which must be replaced with the appropriate information for a particular testing phase.

Text that appears in *italic* type is provided for information only (such as illustrative examples), and should not appear in the final Test Specification Document.

Where references to “Appendices” appear within this appendix, these refer to other appendices in this book. The term “Annex” is used to refer to supplementary sections within this template.

The principal approach to be used in writing a Test Specification Document is to keep it brief and simple. Make best use of the supporting information presented in this book (such as the other relevant template appendices), as well as pertinent material from the appropriate testing phase chapters.

An electronic version of the template is available from the following URL: <us.cambridge.org/titles/052179546X> (or, assuming the role has been implemented, a copy can be obtained from the *Testing Manager* or equivalent role).

TEST SPECIFICATION DOCUMENT FRONT SHEET

Document Information {to <Client> Doc Standard}	
Project ID:	<Project ID> <i>The unique ID for this testing project</i>
Document Ref:	<Doc Ref> <i>A unique document reference for this document</i>
Testing Phase:	<Testing Phase> <i>The testing phase (e.g., Unit Test)</i>
AUT Title:	<AUT Title> <i>The definitive title of the application under test</i>
Date:	<Date> <i>The date this document was completed</i>

Distribution	
Copy Number	Recipient
1.	<activity leader> <i>That is, Development Team Leader for Unit and Integration Testing, Test Team Leader for System Test, and so on</i>
2.	<Test Analyst> <i>That is, the person designing and developing the Test Cases</i>
N.	Project file

Review & Approval	
Issue	<Issue status> <i>Issue status of the document (e.g. Draft, v1.0)</i>
R&A Number:	<R&A reference> <i>The reference to the final approving review</i>
Author	<name of the author> <i>The person who wrote this document</i>
Author Signature	<signature of author>
Approval (PM)	<name of the Project Manager>
Approval (PM) Signature	<signature of the Project Manager >
Approval (QA)	<name of the Quality Assurance representative>
Approval (QA) Signature	<signature of the Quality Assurance representative>

CONTENTS

- 1 Introduction**
 - 1.1 Background**
 - 1.2 Scope**
 - 1.3 Structure of the Document**
 - 1.4 References**
- 2 Test Requirement**
 - 2.1 Introduction**
 - 2.2 Test Philosophy**
 - 2.2.1 Overview*
 - 2.2.2 Functional Areas*
 - 2.2.3 Test Result Categories*
 - 2.2.4 Exclusions*
 - 2.3 Test Environment**
 - 2.3.1 Overview*
 - 2.3.2 Hardware*
 - 2.3.3 Software*
 - 2.3.4 Test Data*
 - 2.4 Staff Roles and Responsibilities**
 - 2.5 Test Identification**
 - 2.5.1 Test Scripts*
 - 2.5.2 Result Reporting*
 - 2.5.3 Acceptance Criteria*
 - 2.5.4 Test Error Clearance*
 - 2.5.5 Test Documentation*
- 3 Test Procedure**
 - 3.1 Introduction**
 - 3.2 Pre-Test Activities**
 - 3.2.1 Test Environment*
 - 3.2.2 Test Timescales*
 - 3.2.3 Test Liaison*
 - 3.2.4 Test Script Preparation*
 - 3.3 Conducting the Test**
 - 3.3.1 Test Execution Procedure*
 - 3.3.1 Filing of Completed Test Result Record Forms*
 - 3.4 Post-Test Activities**
- Annex A – Test Cases**

1 Introduction

1.1 Background

This document provides the specification for the testing activities required for the <Testing Phase> testing of the <AUT Title>.

This section should contain a brief background of this testing project, which should include:

- (a) *a brief description of the purpose of the AUT*
- (b) *definitive version information for the AUT*
- (c) *any other relevant supporting information.*

1.2 Scope

This document contains both the requirement for and the specification of the <AUT Title> <Testing Phase> test.

This section should contain any specific information on the scope of this testing (for example, if the testing activity does not address Nonfunctional Testing of the AUT, this should be stated).

1.3 Structure of the Document

This document is structured in the following manner:

- (a) Chapter 2 documents the <Testing Phase> test requirement for conducting <Testing Phase> testing on the <AUT Title> system.
- (b) Chapter 3 describes the high-level activities comprising the <Testing Phase> test.
- (c) Annex A contains a list of the Test Cases to be employed in testing <AUT Title>.

1.4 References

A numbered list of documents referenced within this document. For example, <list of references to Overall Project Documents>

- (1) *<Project authorization document>*
- (2) *<Overall Project Plan>*
- (3) *<Quality Assurance Plan>*
- (4) *<Configuration Management Plan>*
- (5) *<Relevant Policies referenced within this document>*
- (6) *<Relevant Standards referenced within this document>*
<List of references to items specific to this testing project>
- (7) *<AUT Title> Test Plan Document (version N.N), <date>*
- (8) *<AUT Title> Functional Specification (version N.N), <date>*
- (9) *<AUT Title> Design Specification (version N.N), <date>*
- (10) *<AUT Title> User's Guide (version N.N), <date> (if appropriate)*
- (11) *<AUT Title> Operations Guide (version N.N), <date> (if appropriate)*
- (12) *<AUT Title> Installation Guide (version N.N), <date> (if appropriate).*

2 Test Requirement

2.1 Introduction

This chapter documents the test requirement for conducting <Testing Phase> testing on the <AUT Title> system.

Testing will be against the Functional Specification (Reference 1), plus any other test instructions specified within this document.

The structure of this chapter is as follows: Section 2.2 describes the philosophy behind the <Testing Phase> test process; Section 2.3 describes the requirements for the <Testing Phase> test environment; and Section 2.4 describes the test procedures to be used during the <Testing Phase> testing.

2.2 Test Philosophy

2.2.1 Overview

This section provides a brief overview of the purpose and goals of this testing activity.

The author should consider cutting and pasting selected sections from the "Overview" section of the appropriate Testing Phase Chapter of this Framework Document (e.g., Chapter 5, Unit Testing).

The Overview should also contain a statement describing what the basis of developing the tests will be. For example, The individual <Testing Phase> tests will be derived from the Functional Specification Document (Reference 1).

Finally, the Overview should state on what version of the AUT the testing will take place. For example, Acceptance Testing will be performed on the final delivered system (vN.N).

2.2.2 Functional Areas

This section describes the general areas of functionality that will be tested as part of the <Testing Phase> test; *For example:*

- (a) *user interface, including Help system*
- (b) *on-line query and maintenance facilities*
- (c) *error handling*
- (d) *remote operation of the system.*

2.2.3 Test Result Categories

This section describes the test result categories that can be assigned as the result of conducting individual Test Cases; *for example:*

There are five categories of test results, of which four represent an error state (that is, a deviation of the observed result from the expected result). The categories are as follows:

- (a) *Pass – the observed test result conforms to the expected result.*
- (b) *Acceptable – the observed result indicates that the system differs from the agreed specification but is acceptable, requiring no change to the AUT, but requiring a change in the Functional Specification.*
- (c) *Tolerable – the observed result is incorrect; the AUT is workable and will be accepted, but the fault must be rectified within an agreed-on time period.*
- (d) *Intolerable – the observed result is incorrect and the fault must be corrected before the AUT passes this testing phase.*

- (e) *Test Error – the observed result is correct, but the expected result is incorrect (e.g., a typographical error in the test script).*

On completion of each test, one of the above test result categories will be assigned to the Test Case by the *Tester* and recorded in the Test Result Record form.

2.2.4 Exclusions

This section lists any areas or aspects of the AUT that will not be considered for testing within this testing project; *for example:*

The User Acceptance Test will exclude testing of the system administration functions, since these will be tested during the Operations Acceptance Testing.

2.3 Test Environment

2.3.1 Overview

This section provides a brief overview of the test environment and should include:

- (a) *where the testing is to take place (i.e., which <Client> Site/Department)*
- (b) *which computer facilities (e.g., standalone PCs, <Client> Network)*
- (c) *any other relevant supporting information.*

2.3.2 Hardware

This section provides definitive information on the hardware to be used for this testing activity; *for example:*

The acceptance test will be conducted using the following hardware:

- (a) *a twin node DEC VAX cluster consisting of:*
 - (i) *VAX 6410 of 32Mbytes memory*
 - (ii) *VAX 8550 of 48Mbytes memory*
- (b) *VT220 terminal*
- (c) *HP Laserjet 4Si printer.*

2.3.3 Software

This section provides definitive information on the software to be used for this testing activity (including operating system details and any ancillary system details, such as networking software); *for example:*

The acceptance test will be conducted using the following software:

- (a) *<AUT Title> (<version information>)*
- (b) *VAX VMS operating system (v 5.4)*
- (c) *Oracle RDBMS (v 6.0).*

2.3.4 Test Data

This section specifies the data that will be used for the purposes of the testing, as well as any exceptions or exclusions; *for example:*

The acceptance test will be conducted using live data. No items of handcrafted data will be used during the test.

2.4 Staff Roles and Responsibilities

This section specifies the staff members who will be expected to attend the <Testing Phase> test, the roles that they will perform, and their responsibilities.

Unless there are significant departures from the Roles and Responsibilities outlined in the appropriate testing phase chapter of this book (such as Chapter 5, Unit Testing), cutting and pasting that section, or providing a reference to the appropriate section, will be sufficient.

If there are any exceptions or exclusions to the standard roles and responsibilities, these should be stated. For example, during the User Acceptance Test, the User role will be represented by a committee of five users.

2.5 Test Identification

2.5.1 Test Scripts

Each test will have a script describing how that test will be conducted and the expected result (Annex A provides a list of the individual Test Cases). In particular, the script will contain the following information:

- (a) a unique test identifier
- (b) a description of the purpose of the test
- (c) a brief description of the state of the AUT prior to the test (*e.g., for the purposes of this test, the Tester will begin the test with the AUT in the Start Up screen. The working database will be loaded, and the cursor will be in the Login Name field of the Login Dialog Box*).
- (d) the precise and unambiguous steps required to execute the test
- (e) a brief description of the expected results.

See Appendix E for a Test Script template.

2.5.2 Result Reporting

The test results will be recorded on a Test Result Record Form. This form will record the following information:

- (a) *the AUT title and version*
- (b) *the testing phase* (e.g., Unit Test)
- (c) *the date of the test*
- (d) *a unique test identifier*
- (e) *the time of execution of each Test Case*
- (f) *the observed test result*
- (g) *the assigned Test Result Category* (as documented in Section 2.2.3 of this document)
- (h) *a Test Error description* (in the event that the assigned Test Result Category is Test Error)
- (i) *the signature of the Tester and Test Observer* (plus any other signatures appropriate to the particular testing phase).

See Appendix F for a Test Result Record Form template.

2.5.3 Acceptance Criteria

This section documents the frequencies of the Test Result Categories (see Section 2.3.3) that are considered acceptable for the AUT to pass this testing phase; *for example*:

The AUT will be deemed acceptable and the <Testing Phase> Test Certificate signed when:

- (a) *There are no outstanding intolerable faults.*
- (b) *There are less than 5 tolerable faults.*

This section may also consider a number of general criteria that could be satisfied in order to accept the AUT:

- (a) *Test Requirements – has it been demonstrated that all the requirements for the AUT have been tested?*
- (b) *Test Coverage – has it been demonstrated that all “parts” of the software have been exercised during testing (including exception-handling and error-handling routines)?*
- (c) *Test Case Metric – how many Test Cases have been planned, designed, implemented, executed, and passed or failed? This is a useful metric to collect to measure the progress of the testing activity*
- (d) *Defect Detection Metric – has the rate of defect detection been plotted, and has the rate of detection of defects leveled off? This is a reasonable indication that the majority of the defects have been detected (however, caution must be exercised, since tester fatigue can produce a similar result).*

2.5.4 Test Error Clearance

Where errors have been observed and recorded during testing, this section specifies the process by which correction of the errors is to be achieved; *for example*:

For each observed error that requires further work to the AUT or the Functional Specification, the User Representative, Test Team Leader, and Development Team Leader will formally agree the following:

- (a) *The scope of rework and timescales for correction*
- (b) *The Test Case (or Test Cases) required to be rerun following correction.*
- (c) *Any such rework made to the <AUT title> system will be made under strict change-management procedures.*

2.5.5 Test Documentation

This section describes the documents that will be generated in support of the test activity. These documents are as follows:

- (a) *The Test Script and component Test Cases*
- (b) *Test results in the form of completed Test Result Record forms*
- (c) *A Test Report briefly summarizing the test results and any outstanding actions*
- (d) *A Test Certificate to formally record that the AUT has passed the testing (see Appendix H for a Test Certificate template).*

3 Test Procedure

3.1 Introduction

This chapter describes how the <Testing Phase> test will be performed.

This chapter is structured as follows: Section 3.2 describes the activities that must be completed before testing can begin; Section 3.3 describes the activities that take place during the <Testing Phase> test; Section 3.4 describes the activities that take place after the <Testing Phase> test.

3.2 Pre-Test Activities

This section describes the activities that must be completed prior to the testing of the AUT.

3.2.1 Test Environment

This section describes what must be done in setting up the test environment for the testing. *For example:*

The Test Team Leader will contact the System Manager to ensure that the live system will be available for the testing and that live system data are available for the Acceptance Test.

3.2.2 Test Timescales

The <AUT title> <Testing Phase> test start date will be agreed-on in consultation with <staff involved in the testing> (see the *Roles and Responsibilities* section of the appropriate *Testing Phase* chapter for further details).

The period allowed for the completion of the test will be <time period> (as determined from the *Testing Plan*).

It may also be appropriate to specify how much time will be allowed for retest following correction of errors. Although it may be difficult to provide an estimate, one rule of thumb would suggest that retest should take less time than the original testing period.

3.2.3 Test Liaison

This section documents the requirements for liaison with other staff; *for example:*

The Test Team Leader will liaise with the Development Team Leader to agree the delivery date of the AUT for testing.

The Test Team Leader will liaise with the Test Observer to ensure that he or she can attend the testing.

The Test Team Leader will liaise with the User Representative to ensure that he or she can attend the testing.

3.2.4 Test Script Preparation

The Test Analyst will, with reference to the Functional Specification for the AUT and any other specific instructions, prepare a Test Script and component Test Cases.

3.3 Conducting the Test

3.3.1 Test Execution Procedure

The Tester will execute the individual Test Cases comprising the <Testing Phase> test in the order in which they appear in the Test Script.

For each Test Case, the Tester will follow the instructions specified in the Test Script for executing that Test Case.

Following the completion of each Test Case, the Tester will complete a Test Result Record Form, assigning a Test Result Category for that Test Case by observation and interpretation of the test result and signing the Test Result Record form for each Test Case.

The Independent Observer will oversee this process to ensure that the Tester follows the correct procedure and to witness this fact by signing the Test Result Record Form for each Test Case.

3.3.2 Filing of Completed Test Result Record Forms

Completed Test Result Record forms will be provided to the Test Team Leader for filing and subsequent review at the Post-Testing Review.

3.4 Post-Test Activities

This section describes the activities that must be completed following the testing; *for example:*

Following the completion of the <Testing Phase> test, a review meeting will be convened with the objective of:

- (a) *reviewing the Test Result Record Forms*
- (b) *determining whether the AUT passes the <Testing Phase> test.*

If any test observations require further work on the AUT or its Functional Specification, the following will be agreed on:

- (a) *the scope of further work and the timescales for correction*
- (b) *the Test Case (or Test Cases) that need to be rerun.*

A Test Certificate will be issued to show formally that the AUT has passed its <Testing Phase> test.

Backup and Archival of the Test Assets (AUT, Test Data Set(s), Testing Procedures, Test cases, description of Test Rig) shall take place.

A summary will be written of lessons learned for use in future testing phases.

Annex Test Cases

This annex contains a list of the Test Cases to be designed, implemented, and executed during the testing of <AUT>.

For each Test Case, the following information is documented:

- (1) *the unique identifier for the Test Case*
- (2) *a reference to the requirements this Test Case will test*
- (3) *a brief description of the Test Case*
- (4) *any other supporting information.*

E

Test Script Template

E.1 Introduction

This appendix contains a *Test Script* template that may be copied and used to record the information necessary for conducting each test comprising a particular testing phase.

A completed example of a Test Script (Figure E.1) is also included for guidance.

For each test, a separate Test Script will need to be completed by the *Test Analyst*. Each Test Script contains the following sections:

- ▲ Project ID – the unique project identifier
- ▲ AUT Title – the definitive title of the *Application Under Test* (front sheet only)
- ▲ AUT Version – the definitive version information for the AUT (front sheet only)
- ▲ Testing Phase – the title of the phase of testing being conducted (e.g., *Unit Test*, *Integration Test*, *System Test*) (front sheet only)
- ▲ Date of Test – the planned start date of testing (front sheet only)
- ▲ Test ID – the unique identifier for the test
- ▲ Purpose of Test – a brief description of the purpose of the test, including a reference, where appropriate, to the requirement that is to be tested (consider providing references to the requirements specification, design specification, user guide, operations guide and/or installation guide), as well as any dependencies from or to other Test Scripts/*Test Cases*
- ▲ *Test Environment* – a brief description of the environment under which the test is to be conducted (may include a description of the state of the AUT at the start of this test, details regarding the platform or operating system, as well as specific information about data used in this test)
- ▲ Test Steps – concise, accurate, and unambiguous instructions describing the precise steps the *Tester* must take to execute the test, including navigation through the AUT as well as any inputs and outputs
- ▲ Expected Result – a brief and unambiguous description of the expected result of executing the test.

Where text appears in angle brackets (< >), this denotes a place marker, which must be replaced with the appropriate information for a particular testing phase. For example, <Client> should be replaced by the name of your own company or organization.

Where the Test Script template provided in this appendix does not precisely match the requirements of a particular testing project, the *Test Team Leader* should obtain an electronic copy of this template and modify it as required.

An electronic version of the template is available from the following URL: us.cambridge.org/titles/052179546X (or, assuming the role has been implemented, a copy may be obtained from the *Testing Manager* or equivalent role).

<CLIENT> TEST SCRIPT

(front sheet)

Project ID			
AUT Title		Version	
Testing Phase		Date of Test	

Test ID	
Purpose of Test	
Test Environment	
Test Steps	
Expected Result	

<CLIENT> TEST SCRIPT

(continuation sheet)

Project ID	
------------	--

Test ID	
Purpose of Test	
Test Environment	
Test Steps	
Expected Result	

LIME TELECOMS TEST SCRIPT*(front sheet)*

Project ID	P1234A		
AUT Title	Market*Master (M*M)	Version	v1.1
Testing Phase	User Acceptance	Date of Test	8/31/2000

Test ID	P1234A/PM5.5/1
Purpose of Test	To ensure that: ▲ M*M can be run ▲ the user can log in successfully.
Test Environment	Windows 2000 Start Up window. No other applications should be running. Hand crafted "login data" is held in C:\test-dir\login.dat.
Test Steps	Invoke the M*M application using the Windows 2000 Start menu. In the login dialog box, the Tester should: ▲ left click into the "User Name" field and enter "testuser" ▲ left click into the "Password" field and enter "password" ▲ left click on the "OK" button.
Expected Result	On completing the above steps, the M*M application should start up. Once started, the M*M Start Up screen should be displayed (The Title Bar contains the legend "M*M Start Up Screen.")

*Page 1 of 1 Pages***Figure E.1.** Completed Sample Test Script.

F

Test Result Record Form Template

F.1 Introduction

This appendix contains a *Test Result Record Form* template that may be copied and used to record the result of the individual *Test Cases* comprising a particular test phase.

The Test Result Record Form contains the following sections:

- ▲ Project ID – the unique project identifier
- ▲ AUT Title – the definitive title of the *Application Under Test* (front sheet only)
- ▲ AUT Version – the definitive version information for the AUT (front sheet only)
- ▲ Testing Phase – the title of the phase of testing being conducted (e.g., *Unit Test*, *Integration Test*, *System Test*) (front sheet only)
- ▲ Date of Test – the date on which the test was performed (front sheet only)
- ▲ The names of the *Tester* and the *Test Observer*
- ▲ Test ID – the unique identifier for the test
- ▲ Time of Test – the time at which each individual Test Case was started
- ▲ Observed Test Result – a brief textual description of the result of the test
- ▲ Test Result Category – the *Test Result Category* assigned to this test (as specified from the *Test Specification* Document for this testing phase – see Appendix D)
- ▲ Test Error Description – a brief textual description (if appropriate) of the observed *Test Error*
- ▲ Tester Signature – the signature of the *Tester*, recording his or her participation in the test, observation of the result of the test, and agreement to the documented test result
- ▲ Test Observer Signature – the signature of the *Independent Test Observer*, recording the observation of the result of the test and his or her agreement to the documented test result.

With the exception of the last item, these sections of the Test Result Record Form will be completed by the Tester in consultation with the Independent Test Observer and any other staff attending the testing phase in an official capacity (such as the *User Representative* in *User Acceptance Testing*).

Where text appears in angle brackets (< >), this denotes a place marker, which must be replaced with the appropriate information for a particular testing phase. For example, <Client> should be replaced by the name of your own company or organization.

Where it is necessary to include other signatures on the Test Result Record Form, an electronic copy of the master version of this form should be obtained from the CD accompanying this book or the Test Manager (or equivalent role) and modified as required.

An electronic version of the template is available from the following URL: us.cambridge.org/titles/052179546X (or, assuming the role has been implemented, a copy may be obtained from the *Testing Manager* or equivalent role).

<CLIENT> TEST RESULT RECORD FORM (front sheet)

Project ID			
AUT Title		AUT Version	
Testing Phase		Date of Test	

Test ID		Time of Test	
Observed Test Result			
Test Result Category			
Test Test Error Description			
Tester Signature			
Test Observer Signature			

<CLIENT> TEST RESULT RECORD FORM

(continuation sheet)

Project ID			
AUT Title		AUT Version	
Testing Phase		Date of Test	

Test ID		Time of Test	
Observed Test Result			
Test Result Category			
Test Test Error Description			
Tester Signature			
Test Observer Signature			



Test Log Template

G.1 Introduction

This appendix contains a template for a *Test Log*, which may be copied and used to record the activities and events comprising a particular test phase. An electronic version of the template is available from the following URL: us.cambridge.org/titles/052179545X (or, assuming the role has been implemented, a copy may be obtained from the *Testing Manager* or equivalent role).

The purpose of a Test Log is to document the chronological record of all relevant details of a testing project (in effect, a Test Log is a diary of all relevant events that have occurred during the testing project).

The Test Log forms part of the formal documentation associated with the testing project and will be filed at the completion of the testing project.

The Test Log template presented on the following pages is based on that found in Reference 1, with additional information from Reference 15, plus extensive feedback obtained firsthand in implementing Testing Frameworks for a number of clients (see the Case Study section of this book).

Two different Test Log pages are presented on successive pages:

- ▲ Test Log front sheet
- ▲ Test Log continuation sheet.

The information that needs to be filled in on the Test Log includes:

- ▲ Project ID – the unique project identifier
- ▲ AUT Title – the definitive title of the *Application Under Test* (front sheet only)
- ▲ AUT Version – the definitive version information for the AUT (front sheet only)
- ▲ Testing Phase – the title of the phase of testing being conducted (e.g., *Unit Test*, *Integration Test*, *System Test*) (front sheet only)
- ▲ Date of Test – the date on which the testing project was initiated (front sheet only)
- ▲ Overview – a brief overview of the testing activity, including what is being tested and by who, and any other appropriate details relating to the testing activity (front sheet only)
- ▲ Activities and Event Entries, which include:
 - *Date* – the date on which the Activity/Event took place
 - *Time* – the time at which the Activity/Event took place

- *Activity/Event* – a brief description of the testing activity/event being recorded.

Where text appears in angle brackets (< >), this denotes a place marker, which must be replaced with the appropriate information for a particular testing phase. For example, <Client> should be replaced by the name of your own company or organization.

(continuation sheet)

Project ID	
------------	--

[illegible]



Test Certificate Template

H.1 Introduction

This appendix contains a specimen test certificate, which may be customized and used to formally record the successful completion of a specific test phase for a particular *Application Under Test (AUT)*.

Although it is unlikely that a test certificate will be used to formally record the successful completion of the earlier phases of the testing process, in any formal acceptance of an AUT, and in particular involving a third-party developer or supplier, the use of such a certificate should be mandatory.

When customizing the specimen test certificate template for a particular AUT under a specific testing phase, the following information should be considered for inclusion within the test certificate:

- ▲ The title of the AUT
- ▲ The Test Phase name
- ▲ A Statement of Acceptance describing the circumstances of the test, its date (and optionally, its location), and what the AUT has been tested against
- ▲ The signature of the staff member accepting the tested AUT (must be a senior level, particularly for acceptance of third-party developed systems or products). The signature should be dated.
- ▲ The signature of the *Independent Test Observer* (this item may not be appropriate for an *Acceptance Test* where the system has been supplied by a third-party organization). The signature should be dated.
- ▲ The signature of the Supplier or Developer (these need not be the representatives of a third-party organization—for example, the AUT may have been developed in-house, with a suitably senior representative of the development project signing). The signature should be dated.

Where text appears within the body of the certificate in angle brackets (< >), this denotes a place marker, which must be replaced with the appropriate information for a particular testing phase. For example <Own Organization> should be replaced by the name of your own company or organization.

Once the test certificate has been customized, it should be printed on good-quality paper (particularly when a third-party supplier or developer will co-sign the certificate). If it is appropriate, company branded or letterheaded paper should be considered.

An electronic version of the template is available from the following URL: <us.cambridge.org/titles/052179546X> (or, assuming the role has been implemented, a copy may be obtained from the *Testing Manager* or equivalent role).

<AUT TITLE> <TESTING PHASE> TEST CERTIFICATE

It is hereby agreed that:

The delivered <AUT Title> (version <version details>) as supplied by <Name of Supplier or Developer> under contract to <Own Organization> conforms to its functional specification (<Definitive Reference to the Functional Specification Document>).

for and on behalf of <Own Organization>

Signature: Dated:

Name: <Name of Own Organization Representative, e.g., Senior User (see Chapter 4)>

Role: <Own Organization Representative Job Title>

Optionally, where a Quality Assurance Representative is involved

for and on behalf of <Own Organization>

Signature: Dated:

Name: <Name of Independent Test Observer, e.g., QA Rep. (see Chapter 4)>

Role: <Own Organization QA Representative >

for and on behalf of <Name of Supplier or Developer>

Signature: Dated:

Name: <<Name of Supplier or Developer Representative>

Role: <Job title of Supplier or Developer representative>

I

Re-use Pack Checklist

I.1 Introduction

This appendix contains guidance on ensuring that the re-use of tests and testing materials is achieved between testing phases and testing projects. It includes:

- ▲ How and when to create a *Re-use Pack*
- ▲ A list of the typical contents of a *Re-use Pack*
- ▲ Guidance on using *Re-use Packs*.

I.2 Creating a Re-use Pack

At the end of any testing phase, the *Test Team Leader* (or *Development Team Leader* during *Unit* and *Integration Testing*) should collect copies of a number of items generated during the testing project into a single file (a simple manila folder is usually sufficient for this purpose). This file is termed a *Re-use Pack*.

The *Re-use Pack* will be archived with the rest of the project files and can be retrieved subsequently on request from the *Testing Manager* (or equivalent role).

The contents of the *Re-use Pack* can be used by other testing projects in order to reduce the effort and cost associated with testing and to improve the consistency and quality of testing.

Examples of the use of *Re-use Packs* include:

- ▲ During subsequent testing phases of the same *Application Under Test (AUT)* (such as re-using some *System tests* during *Acceptance Testing*)
- ▲ In *Regression Testing* modifications to the same *AUT* (such as either rerunning a complete *System Test* or selected elements of the *System Test*)
- ▲ Where the *AUT* is similar or related to another *AUT* (such as employing existing integration test material when an additional module is added to an existing suite of applications).

I.3 Contents of a Re-use Pack

A typical Re-use Pack should contain:

- ▲ A paper copy of the *Test Specification document*
- ▲ A paper copy of the *Test Log*
- ▲ A paper copy of the *Test Script* (and any revised Test Cases)
- ▲ A paper copy of the Test Guide used during the testing
- ▲ A paper copy of the Test Summary Report document
- ▲ A floppy disk of the appropriate format containing the above
- ▲ A floppy disk copy of the test data and, in particular, any handcrafted data (if technically feasible) or a reference to an electronically archived copy of the data
- ▲ General (see Section I.4) and specific instructions (written by the *Test Team Leader* prior to archival) on using the Re-use Pack.

I.4 How to Use the Re-use Pack

The following steps should be employed in using the Re-use Pack within the context of a new testing project:

- ▲ Discuss the availability of existing appropriate testing material with the Testing Manager and obtain a copy of the Re-use Pack.
- ▲ Review the contents of the Re-use Pack for its relevance to the current testing requirement (in particular, read the specific instructions for using the Re-use Pack).
- ▲ Identify any tests and testing materials that can be re-used for the current testing project.
- ▲ Incorporate these tests and testing materials into the current testing project.
- ▲ Where possible, use the electronic versions of the tests and testing materials to save time and effort.
- ▲ On completion of the testing project, restore the initial state and contents of the Re-use Pack and return it to the Testing Manager for storage.

J

Test Summary Report Template

J.1 Introduction

This appendix contains a Test Summary Report template, which may be copied and used as the basis of a Test Summary Report for a particular testing phase. The purpose of a Test Summary Report is to summarize the result of a particular testing phase and to provide the basis for subsequent improvement of the testing process.

As described in Chapter 4 (The Management and Planning of Testing), each testing project must generate a Test Summary Report as one of the documents produced at the conclusion of any given testing phase.

The principal approach to be used in writing a Test Summary Report is to try to produce a short, succinct document. Make best use of the supporting information presented in this book (such as the other relevant templates), as well as pertinent material from the appropriate testing phase chapters.

Where text appears within the body of this template in angle brackets (< >), this denotes a place marker, which must be replaced with the appropriate information for a particular testing phase.

Text that appears in italic type is provided for information only (such as illustrative examples), and should not appear in the final Test Summary Report.

An electronic version of the template is available from the following URL: <us.cambridge.org/titles/052179546X> (or, assuming the role has been implemented, a copy may be obtained from the *Testing Manager* or equivalent role).

TEST SUMMARY REPORT FRONT SHEET

Document Information {to <Client> Doc Standard}	
Project ID:	<Project ID> <i>The unique ID for this testing project</i>
Document Ref:	<Doc Ref> <i>A unique document reference for this document</i>
Testing Phase:	<Testing Phase> <i>The testing phase (e.g., Unit Test)</i>
AUT Title:	<AUT Title> <i>The definitive title of the application under test</i>
Date:	<Date> <i>The date this document was completed</i>

Distribution	
Copy Number	Recipient
1.	<activity manager> <i>That is, the Testing Manager</i>
2.	<activity leader> <i>That is, Development Team Leader for Unit and Integration Testing, Test Team Leader for System Test, and so on</i>
N.	Project File

Review & Approval	
Issue:	<Issue status> <i>Issue status of the document (e.g. Draft, v1.0)</i>
R&A Number:	<R&A reference> <i>The reference to the final approving review</i>
Author	<name of the author> <i>The person who wrote this document</i>
Author Signature	<signature of author>
Approval (PM)	<name of the Project Manager>
Approval (PM) Signature	<signature of the Project Manager >
Approval (QA)	<name of the Quality Assurance representative>
Approval (QA) Signature	<signature of the Quality Assurance representative>

CONTENTS

- 1 Introduction
 - 1.1 Background
 - 1.2 Structure of the Report
 - 1.3 References
- 2 Overview
- 3 Variances
- 4 Assessment
- 5 Results
- 6 Evaluation
- 7 Summary of Activities

1 Introduction

1.1 Background

This document provides the Test Summary for the <Testing Phase> testing of the <AUT Title>.

This section should contain a brief background and history of this testing project, which should include:

- (a) *a brief description of the purpose of the AUT*
- (b) *definitive version information for the AUT*
- (c) *any other relevant supporting information.*

1.2 Structure of the Report

- (a) Section 2, Overview, provides a high-level overview of the significant events and activities documented during the <Testing Phase> testing of the <AUT Title>.
- (b) Section 3, Variances, records any variances of the artifacts from their design specification (as documented in the <Testing Phase> Test Specification Document [Reference <reference number from Section 1.3 of this report>]) or as documented within the overall Testing Process
- (c) Section 4, Assessment, provides a brief assessment of the comprehensiveness of the testing process for the <Testing Phase> Testing of the <AUT Title>.
- (d) Section 5, Results, provides a summary of the results of the <Testing Phase> Testing of the <AUT Title>.
- (e) Section 6, Evaluation, provides an overall evaluation of the testing process, including any observed problems and/or limitations.
- (f) Section 7 summarizes the major testing activities and events for the <Testing Phase> Testing of the <AUT Title>.

1.3 References

A numbered list of documents referenced within this document; for example, <list of references to Overall Project Documents>

- (1) *<Project authorization document>*
- (2) *<Overall Project Plan>*
- (3) *<Quality Assurance Plan>*
- (4) *<Configuration Management Plan>*
- (5) *<Relevant Policies referenced within this document>*
- (6) *<Relevant Standards referenced within this document>*
<List of references to item specific to this testing project>
- (7) *<AUT Title> <Testing Phase> Test Plan document <Ref> (version N.N), <date>*
- (8) *<AUT Title> <Testing Phase> Test Specification document <Ref> (version N.N), <date>*
- (9) *<AUT Title> <Testing Phase> Test Script document <Ref> (version N.N), <date>*
- (10) *<AUT Title> <Testing Phase> Test Result Record Forms <Ref> (version N.N), <date>*
- (11) *<AUT Title> <Testing Phase> Test Log <Ref> (version N.N), <date>*

2 Overview

This section provides a high-level overview of the significant events and activities documented during the <Testing Phase> testing of the <AUT Title>.

This section also specifies the scope of the testing (what was and what was not tested) and the test environment details (including the hardware, software, and data used in the testing; for example:

The Acceptance Test for the SuperSoft software (v1.1, 2001) was begun on January 4, 2001, and completed on February 21, 2001. During this Acceptance Test only the Customer Services Module was considered.

The testing was conducted on a Pentium III processor with 64 Mbytes of memory running Microsoft Windows 2000 SPI. Data set DS:301 was used to test the AUT.

3 Variances

This section is used to record any variances of the artifacts from their design specification (as documented in the Test Specification document or as documented within the overall Testing Process; for example:

Conditions observed during the course of testing resulted in the design of additional Test Cases to explore concerns regarding the Customer Details input routine. A number of additional defects were identified as a result, which were subsequently corrected and successfully retested. The additional Test Cases were appended to the end Test Script for the Acceptance Test and listed as TP12/AT1/TS1/TC:35 to TC:46.

4 Assessment

This section provides a brief assessment of the comprehensiveness of the testing process for the completed testing phase against the test objectives and constraints specified in the Test Plan document.

Where code coverage measurements have been made, the results should also be included in the section.

This section also identifies any aspects of the AUT that were not tested as thoroughly as planned (due to insufficient time or resources); for example:

All Test Cases were executed with the exception of Test Cases TP12/AT1/TS1/TC:14 to TC:18 (testing the Customer Database Details Access routines), which were omitted because of challenging testing timescales combined with the need to further investigate concerns associated with the Customer Details input routine.

All priority 1 Test Cases associated with the Customer Database Details Access routines were executed successfully, providing sufficient confidence to omit the Test Cases with lower priority.

5 Results

This section provides a summary of the results of the <Testing Phase> Testing of the <AUT Title>. This section identifies all resolved issues and summarizes the details of their resolution. It also lists all outstanding issues; for example:

Three of the Test Cases TP12/AT1/TS1/TC:06 to TC:08 revealed problems with the logic of the New Customer Input routine. The developers corrected the observed defects, and the amended code passed the retest. However, it was not possible within the time scales of the testing task to Regression Test the associated Add New Customer Details to Database routine. This issue remains outstanding and should be observed following installation and use in the live environment.

6 Evaluation

This section provides an overall evaluation of the testing process, including problems and limitations. The evaluation is based on the observed results of testing and how well they matched the evaluation criteria listed in the Test Specification Document for this testing phase.

If available, an assessment of the risk of failure of problem areas of the AUT should be included. This may be qualitative (following a low, medium, and high scale, for example) or quantitative if it is possible to produce probability of failure values; for example:

The Customer Services Module of the SuperSoft application (v1.1, 2001) underwent comprehensive testing, with only five defects being observed, all three of which were associated with the New Customer Input routine.

Additional Test Cases were designed and executed to explore the Customer Details input routine. Following correction and retesting it is believed this routine will have a high degree of reliability in use.

Since a number of problems were observed with the New Customer input routine and only perfunctory retesting was possible because of challenging testing timescales, it is thought that there will be a medium likelihood of failure in use.

Similarly, the lack of time for Regression Testing the associated Add New Customer Details to Database routine may cause issues in use. However, because of the success of the initial testing of this routine, it is thought that the likelihood of failure of this routine is low.

7 Summary of Activities

This section provides a summary of the major testing activities and events for the testing phase. It also summarizes testing resource information, such as total staffing levels, total testing time, time spent in analysis and design, and so on.

It is useful to include planned and actual data where possible.

The information recorded in this section will depend on what information is collected during the progress of the testing. Such information may well form part of any metrics program used as part of the testing process; for example:

Test Start Date: January 4, 2001

Test End Date: February 12, 2001

Item	Planned	Actual
Staff Levels	5	5
Test Design Effort	2.5	3.0
Test Execution Effort	3.5	3.5
Retest Effort	1.0	1.0
Test Management and Reporting	0.5	0.75
Etc.		

K

Equivalence Partition Example

K.1 Introduction

This appendix contains a worked example illustrating the testing technique of *Equivalence Partitioning* described in Chapter 3, which the *Test Analyst* can use to select specimen data as part of the test design process for the *Application Under Test (AUT)*.

K.2 The Testing Problem

The specification for a software system for validating expenses claims for hotel accommodation includes the following requirements:

- ▲ There is an upper limit of \$70 for accommodation expense claims
- ▲ Any claims above \$70 should be rejected and cause an error message to be displayed
- ▲ All expense amounts should be greater than \$0 and an error message should be displayed if this is not the case.

K.3 Analyzing the Testing Requirements

To support the process of analyzing the above requirement, it is useful to graphically show the partitions and their boundaries, and to state the ranges of the partitions with respect to the boundary values (see Figure K.1).

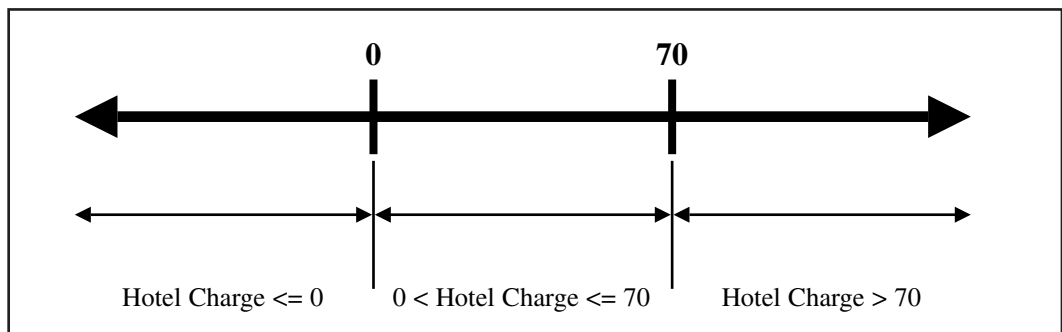


Figure K.1. Graphical view of Partitions.

K.4 Designing the Test Cases

The next step is to design the *Test Cases* by drawing up a table showing the Test Case ID, a typical value drawn from the partition to be input in the Test Case, the partition it tests, and the expected output or result of the Test Case (see Table K.1). This process can be repeated to select further representative data for use within the Test Cases generated by the Test Analyst.

Table K.1. Test Case Table

Test Case ID	Hotel Charge	Partition Tested	Expected Output
1	50	0 < Hotel Charge <= 70	OK
2	-25	Hotel Charge <= 0	Error Message
3	89	Hotel Charge > 70	Error Message

L

Boundary Analysis Example

L.1 Introduction

This appendix contains a worked example illustrating the testing technique of *Boundary Analysis* described in Chapter 3, which the *Test Analyst* can use to select specimen data as part of the test design process for the *Application Under Test (AUT)*. This example uses the same testing problem as that presented in Appendix K, Equivalence Partition Example.

L.2 The Testing Problem

The specification for a software system for validating expense claims for hotel accommodations includes the following requirements:

- ▲ There is an upper limit of \$70 for accommodation expense claims.
- ▲ Any claims above \$70 should be rejected and cause an error message to be displayed.
- ▲ All expense amounts should be greater than \$0, and an error message should be displayed if this is not the case.

L.3 Analyzing the Testing Requirements

To support the process of analyzing the above requirement, it is of benefit to show graphically the boundaries, and to determine the boundary values and the significant values either side of the boundaries (see Figure L.1).

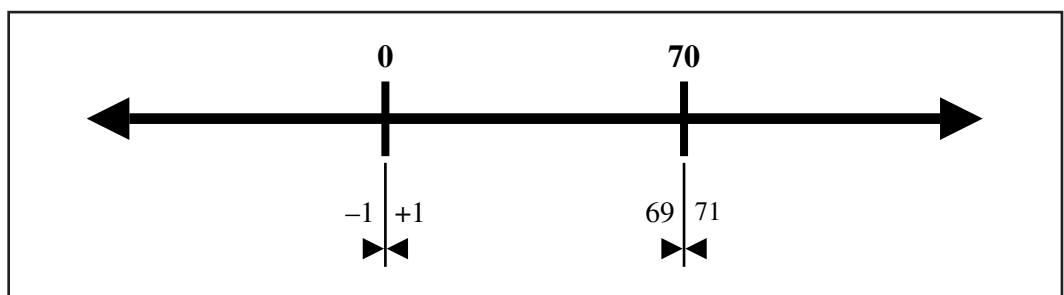


Figure L.1. Graphical view of boundaries.

L.4 Designing the Test Cases

The next step is to design the *Test Cases* by drawing up a table showing the Test Case ID, the values about and on the boundary to be input for the test, the boundary it tests, and the expected output or result of the Test Case (see Table L.1).

These values for hotel charges can now be used by the Test Analyst to design an effective series of effective Test Cases to test the boundary values.

Table L.1. Test Case Table

Test Case ID	Hotel Charge	Boundary Tested	Expected Output
1	-1	0	Error Message
2	0		Error Message
3	1		OK
4	69	70	OK
5	70		OK
6	71		Error Message

M

State Transition Example

M.1 Introduction

This appendix contains a worked example illustrating the testing technique of State Transition analysis described in Chapter 3, which the *Test Analyst* can use to select specimen data for use in testing the *Application Under Test (AUT)* as part of the test design process.

M.2 The Testing Problem

This example describes part of the specification for the software controlling the operation of a water pump:

- ▲ The pump can be in one of three states: Isolated, Ready, or Running. The pump cannot start (that is, move to the Running state) if it is Isolated
- ▲ Opening the water valve will move the pump from the Isolated into the Ready state. Closing the water valve when the pump is Ready will return it to the Isolated state
- ▲ Pressing a Start button when the pump is in the Ready state will start the pump and move it into the Running state. Pressing a Stop button when the pump is Running will stop the pump and move it into the Ready state.

M.3 Analyzing the Testing Requirements

To support the process of analyzing the above requirement, the Test Analyst draws a State Transition diagram to show graphically the states, their transitions, and the events causing those transitions (see Figure M.1).

M.4 Designing the Test Cases

By inspecting the State Transition diagram it is possible to identify a number of *Test Cases* to verify the requirements (that is, *Positive Testing*) for the pump control software:

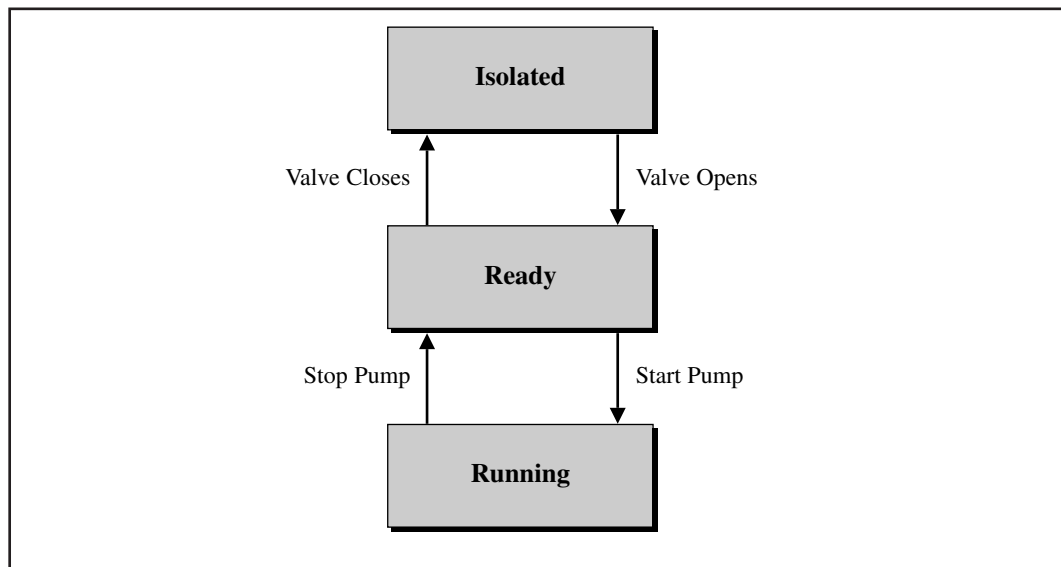


Figure M.1. States, transitions, and events.

- ▲ Test Case 1. With the pump in the Isolated state, does the pump move to Ready on opening the Valve?
- ▲ Test Case 2. With the pump in the Ready state, does the pump move to Isolated on closing the Valve?
- ▲ Test Case 3. With the pump in the Ready state, does the pump move to Running when the Start button is pushed?
- ▲ Test Case 4. With the pump in the Running state, does the pump move to Ready when the Stop button is pushed?

Similarly, by inspecting the State Transition diagram it is also possible to identify a number of Test Cases to check the operation of the system outside of the stated requirements (that is, *Negative Testing*); for example:

- ▲ Test Case 5. What happens if the pump is Running and an attempt is made to close the Valve (typically catastrophic for a real pump)?
- ▲ Test Case 6. What happens if the Start button is pushed when the pump is in the Isolated state?
- ▲ Test Case 7. What happens if the Start button is pushed when the pump is already in the Running state?
- ▲ Test Case 8. What happens if the Start and Stop buttons are pushed simultaneously when the pump is in the Running state?

The ability to visualize the operation of the AUT using the State Transition technique is a very effective way of identifying Test Cases and is particularly effective for Negative Testing purposes.



Automated Testing Tool Selection Criteria

N.1 Introduction

This appendix contains a list of criteria to be used in evaluating *automated software testing tools* and a suggested approach to conducting the evaluation. A simple scoring scheme with weightings is described, which can be used to score each of the evaluation criteria. The values for weightings and criteria scores may be modified through experience and use.

An evaluation summary checklist can be found at the end of this appendix, which may be copied and used in evaluating testing tools.

This appendix is available in electronic form so that the evaluation criteria and template can be customized to match more closely the evaluation requirements of the reader. See the following URL: <us.cambridge.org/titles/052179546X>.

N.2 Scoring Scheme

In conducting a review or evaluation of testing tools, the following approach should be employed:

- ▲ Review the evaluation criteria and assign a weighting to each of them:
 - 1 for *essential criteria*
 - 0.75 for *important criteria*
 - 0.5 for *desirable criteria*
 - 0 for *nonapplicable criteria*

- ▲ When evaluating the tool, consider each criterion and score it as follows:
 - 1 if the tool satisfies the criteria fully
 - 0.75 if the tool largely satisfies the criteria
 - 0.5 if the tool partially satisfies the criteria
 - 0 if the tool does not satisfy the criteria
- ▲ On completing the review, multiply the score for each criterion by its weighting and total the scores. The resulting value can be used to compare different automated testing tools.

The following sections list the evaluation criteria, which are grouped under the following high-level headings:

- ▲ Support for Testing Phases and Techniques
- ▲ Support for Test Management
- ▲ Tool Technical Issues
- ▲ Tool Defect Analysis Facilities
- ▲ Tool Documentation Issues
- ▲ Tool Integration Issues
- ▲ Tool Usability Issues
- ▲ Supplier Issues
- ▲ Training and Consultancy Issues
- ▲ Contractual and Financial Issues
- ▲ Miscellaneous Issues.

N.3 Support for Testing Phases and Techniques

In selecting a testing tool, it is important that the tool be capable of supporting the key testing phases the organization is involved in.

Does the testing tool provide support for the following testing phases?

- ▲ Functional Testing
- ▲ Regression Testing
- ▲ Installation Testing
- ▲ Configuration Testing
- ▲ Maintenance Testing
- ▲ Network Testing
- ▲ Performance Testing (including Load, Stress, Volume, and Scalability testing)
- ▲ Multiuser Testing.

N.4 Support for Testing Management

Management of the testing process is key to successful, effective, and efficient testing. It is important that any testing tool support effective management of testing, providing good support for the typical testing management tasks.

Does the testing tool provide the following test-management facilities?

- ▲ Support for multiple users with varying user privileges
- ▲ The ability to record test plans or integrate seamlessly with a project planning tool
- ▲ The ability to integrate seamlessly with a Requirements Management tool in order to represent, manage, and store test requirements
- ▲ The ability to design Test Scripts and Verification Points against requirements (and for this information to be updated automatically when requirements change)
- ▲ The ability to integrate seamlessly with a Defect Management and change request tool to allow the creation, monitoring, reporting, and resolution of defects
- ▲ The ability to run predefined reports on all aspects of the testing process
- ▲ The ability to customize existing and create new reports.

N.5 Tool Technical Issues

In selecting a testing tool, you need to consider how well the tool facilities match your testing requirements (for example, does the tool support testing against the programming language you are developing in, or does the tool perform on your selected operating system?).

Does the tool provide the following technical capabilities?

- ▲ The ability to record and play back test procedures
- ▲ The ability to work with a particular development language or environment (e.g., C++, Powerbuilder, Visual Basic, Java, Delphi, Oracle)
- ▲ The ability to test object container technologies (e.g., OCXs, VBXs, Data Windows)
- ▲ The ability to test Hidden Objects (i.e., nonvisual objects)
- ▲ The ability to test the attributes of Objects (Visible, Focus, Grayed, etc.)
- ▲ Object-oriented recording/playback of scripts
- ▲ The option of using low level or coordinate-based recording/playback of scripts
- ▲ Clarity/simplicity of the scripting language
- ▲ The ability to edit scripts manually
- ▲ The ability to play back scripts in W98/NT/W2000 without modification
- ▲ The ability to trap Unexpected Windows and act on them
- ▲ The ability to store all information in a single repository
- ▲ The ability to integrate with other tools (e.g., CASE Tools)
- ▲ The ability to update/maintain test cases/test procedures by example
- ▲ Cross-platform support (e.g., mainframe and PC testing)
- ▲ The ability to recover automatically from GPFs/crashes and continue to test.

N.6 Tool-Defect Analysis Facilities

Does the testing tool provide the facilities for inspecting and analyzing defects?

- ▲ The ability to view the Test Log following testing or at a later date
- ▲ The ability to determine which Test Scripts and Verifications passed or failed
- ▲ The ability to examine the first failure and subsequent failures
- ▲ Context-sensitive facilities for viewing defects
- ▲ The ability to enter a defect into the defect-management system
- ▲ Full traceability from the original requirement the test was based on through the point of reporting the defect and on through the defect-tracking facilities
- ▲ The ability to update/maintain test cases/test procedures by example (e.g., to update master data with observed data).

N.7 Tool Documentation Issues

Irrespective of how technically good the testing tool is, the lack of clear and comprehensive documentation may compromise how successfully you use the tool.

Some manufacturers now provide complete tool documentation on CD to save shipping bulky manual sets and to protect the environment. Such on-line documentation can provide a powerful means of accessing information, allowing the user to search for specified information. Also check whether manuals can be ordered as a free option.

Does the testing tool support the following documentation issues?

- ▲ Does the testing tool provide adequate installation documentation?
- ▲ Does the testing tool provide a User Manual?
- ▲ Does the testing tool provide Tutorial information?
- ▲ Does the testing tool provide Quick Start information (such as “try it” sheets)?
- ▲ Does the testing tool provide Trouble Shooting information?
- ▲ Does the testing tool provide a Scripting Language Manual?
- ▲ Is the documentation robust, durable, and of good quality?
- ▲ Is the documentation clear, unambiguous, and usable?
- ▲ Are additional copies of specified documentation available on request?
- ▲ Does the documentation provide further references to source material (e.g., definitive testing books, conference proceedings, testing technical reports)?
- ▲ Does the testing tool provide adequate contact information (e.g., Help desk telephone number, Web Site address, e-mail address)?

N.8 Tool Integration Issues

Integration is an increasingly important issue as senior software development managers appreciate the need to provide good communication among all the members of their projects, from Analysts to Developers to Testers.

IT professionals are increasingly looking for cross-development lifecycle tool support rather than the much less flexible standalone point solutions from single technology vendors.

Also, in considering the level of integration a given testing tool supports, beware of clumsy solutions where the user is expected to open the source tool (such as a Requirements Management tool), save the information held in that system into an intermediate file form, after which they have to run up the target tool (such as the testing tool), and then load the information into that tool. Such solutions are notoriously unreliable, and in practice maintaining the currency of the information such integrations provide is at best difficult, and at worst, results in the Tester working with out-of-date information, jeopardizing the success of the testing project.

In evaluating a testing tool, the following Tool Integration issues should be considered:



- ▲ Does the tool have an integrated Requirements Management facility to ensure that each of the requirements for the Application Under Test can be verified?
- ▲ Does the tool fully integrate with Design/Visual Modeling tools to allow automatic generation of test scripts based on design information (e.g., automatic generation of code stubs and test harness code for Unit Test, automatic generation of Integration Tests from Sequence Diagrams, automatic generation of Boundary and Partition Test data from design information)? For Performance testing tools, is it possible to generate automatically Test Scripts from design information before any code has been written and execute them against the Server logic to test the architecture and scalability of the proposed software?
- ▲ Does the tool have an integrated Defect Management facility to ensure that defects are automatically reported to the staff who need to know about them (such as the Developer, Project Manager, QA Manager). Does the Defect Management facility provide traceability to the original requirement the defect is associated with?
- ▲ If the testing tool is a functional testing product, does it have a seamless integration into other testing tools, such as:
 - *reliability testing tools*
 - *code coverage tools*
 - *low-level Performance Testing tools (to identify code bottlenecks)?*
 - *high-level Performance Testing tools (such as Load, Stress, Volume, Scalability Testing tools)?*
- ▲ Does the tool provide seamless integration into a complete cross-lifecycle Software Development and Testing Process, where requests for Help within the tool take the user directly to context-sensitive Process information, and where the Process provides explicit advice on how to perform specified tasks within the testing tool?

- ▲ Does the tool provide seamless integration into Configuration Management tools, allowing each artifact generated during testing (such as Test Scripts, Verification Points, Reports) to be under rigorous configuration management?

N.9 Tool Usability Issues

Any testing tool should provide facilities that directly support the process by which Testers typically test software in a natural and intuitive manner. The tool may be very powerful, but if the User Interface to the tool is poor, the user may not be able to realize the full potential of the tool.

This section considers the usability/learnability issues associated with the testing tool (it is important to note that the criteria presented here represent only a sample of the usability issues that would need to be addressed in a full usability review):

- ▲ Does the tool adhere to user interface standards (e.g., Windows on PC platforms). For example, does the tool use a standard “File, Edit, <tool specific menus>, Help” menu structure?
- ▲ Is the tool consistent in its use of menus and toolbar buttons? For example, is the “New Doc” toolbar button icon standard,  or is it some form of custom icon? Consistency is key in ensuring rapid familiarization with the GUI.
- ▲ Where other tools integrate with the tool, do the other tools employ the same style of menu and toolbar buttons (or identical toolbar buttons where the function is the same) and is the look and feel the same (for example, try comparing the “Open” toolbar button icons between “integrated” tools to see if they are the same)?
- ▲ Are the toolbar button icons standard and easily understood?
- ▲ Does the tool provide consistent shortcut key access to its facilities?
- ▲ Is the tool simple, intuitive, and easy to understand and use (i.e., can you easily guess what action to take next in most situations)? Beware tools that have “modal” menus – that is, menus where the menu items change with the state of the tool – such tools will be difficult to learn how to use.
- ▲ Does the tool provide good user help facilities:
 - *is there a Help menu providing Help topics?*
 - *is there a Help button on the toolbar?*
 - *do the toolbar buttons have Tooltips (the MS yellow pop-up Help labels)?*
 - *are there Help buttons providing context-sensitive help in dialog boxes?*
 - *does pressing the F1 key provide context-sensitive help?*
 - *is there a “point and click” help button ()? As a test of “attention to detail,” find out what happens if you click on this button and then select the button again (i.e., Help on Point and Click Help)*

- ▲ Where text is used in the tool, is it legible and free of technical terms or jargon?
- ▲ Does the tool allow the user to customize its look and feel?
- ▲ Does the tool allow the user to configure its facilities to match user preferences?
- ▲ Does the tool have good choice of color for background and text (i.e., is there good contrast)?
- ▲ Is the tool attractive and enjoyable to use?

N.10 Supplier Issues

It is essential that you assure yourself about the size and financial stability of the tool vendor. The best tool in the world will be worthless if the supplier business fails. Similarly, small suppliers with few technical consultants may be unable to provide you with adequate technical support, mentoring, consulting, or training.

Small vendors with poor finances may also have difficulties in performing adequate research and development, have problems keeping to promised release schedules, and suffer quality problems with releases. Another problem facing such organizations is the threat of takeover, with no guarantee that the new owners will continue to support the old product range.

Think carefully before adopting tools from point solution vendors because you may experience integration issues if you need to use other tools in your software analysis, design, and testing process. Consult with the other stakeholders in your organization to find out what tools they use for analysis, design, change management, and configuration management to determine if the suppliers of those tools also provide integrated testing tools before making a decision.

Do not be overly influenced by suppliers who claim they have the best-selling tool. A recent customer survey by *.EXE Magazine* showed that the five most important tool selection criteria were reliability, features, performance, ease of use, and quality of documentation. Market leadership was rated as least important – in fifteenth place.

In evaluating a testing tool, the following supplier issues should be considered:

- ▲ Is the supplier financially and commercially sound?
- ▲ Is the supplier part of a larger IT company providing a range of IT services?
- ▲ Have you dealt successfully with the supplier in the past?
- ▲ Does the supplier have a Web site that provides:
 - *company information?*
 - *product-release news?*
 - *technical papers on the testing tool?*
 - *help and tips on the testing tool use?*
 - *sample/re-usable scripts?*

▲ Help desk issues:

- *Is there a Help desk available for user support calls?*
- *Is the Help desk UK based/European based/U.S. East Coast based/U.S. West Coast based (this may affect availability of the service)?*
- *Does the Help desk provide 24-hour support?*
- *Is there an e-mail address for support information?*
- *When you need support, how quickly does the Help desk respond?*
- *When you need support, does the Help desk understand your problem?*
- *When you receive support, is the Help correct and useful?*

N.11 Training and Consultancy Issues

Despite what some vendors may tell you, no competent testing tool can be used without some level of user training. Even if you purchase a tool with comprehensive on-line tutorial material, the users will still have to find the time and resources to make use of the tutorial. In practice, training of staff is essential; otherwise lack of investment in training will cause a greater loss of investment when staff members struggle to use the tool effectively.

Similarly, it can be very cost effective to put aside budget for mentoring and consulting in order to show staff how to best use the tools and to allow regular “health checks” to ensure continued effective use.

For these reasons, the following testing tool training and consulting issues should be considered:

- ▲ Will the supplier install the testing tool at your site and provide introductory information (such as a half-day walk-through of the tool)?
- ▲ Does the supplier offer introductory training? If so:
 - *Can you obtain a copy of the prospectus?*
 - *What is the cost (per person/per day)?*
 - *Is there a minimum number of attendees?*
 - *Can the training be performed at your site?*
- ▲ Does the supplier offer advanced training (this is often a good discriminator)?
- ▲ Does the supplier offer testing fundamentals training (another good discriminator of the depth and quality of the training available)?
- ▲ Does the supplier offer Software Development and Testing Process training (a particularly important criterion for organizations with a mature approach to development and testing)?
- ▲ Does the supplier offer mentoring and skills-transfer consulting?
- ▲ Does the supplier offer tool consultancy?
- ▲ Are there third-party organizations offering tool training/consulting and are they certified by the supplier before they are able to provide training (this is a very good criterion to show widespread use and acceptance of a particular tool)?

N.12 Contractual and Financial Issues

This section addresses a number of financial and contractual issues that must be considered. Some of these issues may be difficult to resolve, but must be considered in evaluating a particular tool (such as estimating the value of the tool in terms of the benefits of using or not using the tool).

Although it makes good business sense to inquire about discounts, be careful not to let obstinacy get in the way of obtaining a product that will be of benefit to your organization. As incredible as it may seem, there have been many occasions on which staff members involved in the purchase of tools have walked away from a purchase just because the supplier is unable to reduce the purchase price by, in some cases, ten dollars!

On the other hand, you should also be very wary of suppliers who will suddenly slash the price of their product as soon as they hear that one of their competitors is involved. You will have to work with the supplier following your purchase (perhaps for training and mentoring as well as for ongoing support), so it is worth considering the business ethics of any supplier who was perfectly happy to charge you \$X one day and then suddenly charge you half that amount just a few days later for exactly the same product (while presumably still making a profit). This is not a good basis for a successful continuing business relationship, and suppliers who indulge in such activities are almost certain to find ways of recouping the discount at a later stage; otherwise their business would be unsustainable.

The following contractual and financial issues should be considered:

- ▲ Does the price/pricing structure of the testing tool represent good value (the resolution of this issue may require cost-benefit analysis)?
- ▲ Is the testing tool priced on a per license basis?
- ▲ Are there any discounts for purchasing multiple licenses?
- ▲ Are they fixed or floating licenses?
- ▲ Does all of the tool functionality come in the “basic package,” or do additional facilities cost extra (e.g., does the basic tool support Network Testing, or is this a separate additional purchase)?
- ▲ How is multiuser testing priced (e.g., is it based on the number of “virtual users” that are required)?
- ▲ What is the cost of maintenance, and does it include upgrades?
- ▲ What does maintenance provide?
- ▲ If priced separately, what is the cost of upgrades?
- ▲ How expensive is training (basic and advanced)?
- ▲ What is the cost benefit of purchasing and using the tool (the resolution of this issue may require cost benefit analysis)?

N.13 Miscellaneous Issues

The following issues are associated with obtaining further information on the testing tool and its use:

- ▲ Is there a User Group for the tool?
- ▲ Is the User Group independent of the supplier?
- ▲ How vigorous is the User Group?
 - *How often do they meet?*
 - *Do they publish proceedings?*
 - *Do they hold seminars? Is there a mailing list?*
 - *Do they have a point of contact for testing tool issues/advice?*
 - *Do they have a Web page?*
- ▲ Is there any independent documentary support for the testing tool?
 - *Are textbooks available on the tool?*
 - *Are there technical reports/journal papers on the tool?*
- ▲ Annual conference:
 - *Is there an annual tool-specific conference?*
 - *Where is it held?*
 - *Are the proceedings available?*

Testing Tool	Version	Date of Evaluation	Evaluator Name

Criteria	Weighting (0–1)	Score (0–1)	Result
Support for Testing Types			
Functional Testing			
Regression Testing			
Installation Testing			
Configuration Testing			
Maintenance Testing			
Network Testing			
Performance Testing (Load, Volume, Scalability)			
Multiuser Testing			
Subtotal			
Support for Test Management			
Support for Multiple Users			
Test Planning facilities			
Test Requirements facilities			
Design of Test Procedures/Test Cases			
Defect Management			
Predefined Reports			
Customization/Creation of new Reports			
Subtotal			
Tool Technical Issues			
Record and Playback facilities			
Required language/environment			
Object Container technology testing			
Testing of “hidden objects”			
Testing attributes of objects			
OO recording and playback			
Low-level/coordinate recording & playback			
Clarity/Simplicity of Scripting language			

Criteria	Weighting (0–1)	Score (0–1)	Result
Tool Technical Issues <i>(continued)</i>			
Manual editing of Scripts			
Playback in W95, W98, NT, W2000			
Trap Unexpected Windows			
Repository-based			
Integration with other tools			
Update/Maintenance of tests			
Cross-Platform Testing – Mainframe/PC			
Automatic recovery from GPF/Crashes			
Subtotal			
Tool Defect Analysis Facilities			
Viewing the Test Log			
Determining Test Procedure/Test Case failure			
Examining failures			
Context-sensitive defect analysis			
Entering defect reports			
Full traceability of original requirements			
Update/Maintenance of Tests			
Subtotal			
Tool Documentation Issues			
Installation documentation			
User Manual			
Tutorial information			
Quick Start information			
Trouble Shooting information			
Scripting Language Manual			
Robust, durable, and good-quality documentation			
Clear, unambiguous, and usable documentation			
Additional copies of documentation			
References to source material			
Adequate contact information			
Subtotal			

Criteria	Weighting (0–1)	Score (0–1)	Result
Tool Integration Issues			
Fully Integrated Requirements Management			
Fully integrated with Visual Modeling tool			
Fully Integrated Defect Tracking			
Full Integration with other testing tools			
Integration with Development & Testing Process			
Integration with Configuration Management tool			
Subtotal			
Tool Usability Issues			
User Interface standards			
Consistent Menus and Toolbar buttons			
Standard Toolbar Icons between Integrated Tools			
Standard Toolbar buttons used within the tool			
Consistent Shortcut key access			
Simple, intuitive, and easy to use			
Help facilities			
Legible and jargon-free text			
Customize Look and Feel			
Configurable user preferences			
Good choice of color and contrast			
Attractive and enjoyable to use			
Subtotal			
Supplier Issues			
Financially and commercially sound			
Part of larger IT company			
Dealt successfully with supplier in past			
Supplier Web site			
Help desk			
		Subtotal	

Criteria	Weighting (0–1)	Score (0–1)	Result
Training and Consultancy Issues			
Install and introduce tool			
Introductory training			
Advanced training			
Testing Fundamentals training			
Software Development & Testing Process training			
Mentoring and skills transfer			
Tool consulting			
Third party training/consulting			
Subtotal			
Contractual and Financial Issues			
Estimated value of Tool			
Pricing Scheme			
Multiple license discount			
Fixed/Floating licenses			
Packaged functionality			
Multiuser pricing			
Cost of maintenance			
Maintenance features			
Cost of upgrades			
Cost of training			
Cost benefits			
Subtotal			
Miscellaneous Issues			
User Group			
User Group independence			
Vigorous User Group			
Documentary support			
Tool conference(s)			
Subtotal			
Total Score			



Usability Testing Overview

0.1 Introduction

This appendix provides a brief overview of the process of usability testing.¹ For a comprehensive treatment of this topic see Reference 9.

Specifically, this appendix describes roles and responsibilities of staff, test design, and test and post-test activities. It also provides a graphic view of the organization of a typical usability test.

0.2 Roles and Responsibilities

The following roles are applicable to usability testing:

- ▲ *Test Team Leader.* The usability testing project is managed on a day-to-day basis by the Test Team Leader, who is also responsible for liaising with all other staff involved in the testing.
- ▲ *Test Analyst.* The usability testing scenarios are devised by the Test Analyst with reference to available usability requirements for the *Application Under Test (AUT)* and in close collaboration with the *Development Team Leader*. This collaboration is essential for identifying specific areas of concern within the AUT user interface (UI) that may need to be investigated
- ▲ *User Representatives.* Usability testing is conducted by a number of User Representatives (users). For purposes of obtaining accurate statistical results, the more users that can be persuaded to become involved, the better. As a general rule, ten users is the minimum number that should be involved in a testing project.
- ▲ *Test Observer.* The testing is remotely monitored by a Test Observer, who monitors all sources of information from the test (video, audio, and the captured AUT UI images) and who has computer-aided support for subsequent analysis of the information
- ▲ *Development Team Leader.* The Development Team Leader may optionally sit with the Test Observer in order to observe the test.

¹This appendix does not discuss the role of psychometric usability tests (Reference 11) or review of graphical user interfaces (Reference 9).

- ▲ *Independent Test Observer.* It is important that the testing process (particularly the analysis and conclusions) be monitored by an Independent Test Observer, who is involved in formally witnessing the usability test.

In practice, it is likely that one person will take on the Test Team Leader, Test Analyst, and Test Observer roles. However, the User Representative, Development Team Leader, and Independent Test Observer must be distinct roles.

0.3 Usability Test Design

The aim of usability testing is to confirm that the AUT meets its usability requirements as stated in the functional specification for the application.

The Test Analyst is responsible for inspecting the usability requirements and designing the individual Test Cases that will comprise the usability test. The Test Analyst will also liaise with the Development Team Leader and may base a number of tests on usability issues raised during these discussions.

0.4 Usability Testing

Usability testing is conducted in a formal manner under controlled conditions. All information from the test, including video footage of the User Representative interacting with the AUT, the image of the AUT itself, and any audio information – such as User Representative comments, is captured on videotape for subsequent review and analysis. Figure O.1 shows the typical organization of a usability test.

The usability test is organized and managed by the Test Observer, who during the usability test will observe and monitor the test remotely.

During the usability test, the User Representative will be expected to follow the Test Script generated by the Test Analyst. The only assistance provided to the User Representative will be on-line Help (if the AUT supports such a facility), the user manual, or Help documentation (if available). Depending on the usability requirements for the AUT, the on-line Help or paper-based documentation itself may be the focus of the usability test.

For the purposes of statistical analysis, at least ten User Representatives should perform the usability test. The User Representatives should be drawn from the same user group (that is, they should have broadly similar IT experience and should all have similar business skills and experience appropriate to the AUT that is the focus of the usability test).

0.5 Post-Test Activities

Following usability testing the Test Observer will review the information collected during the usability test in order to determine if the AUT meets its usability requirements.

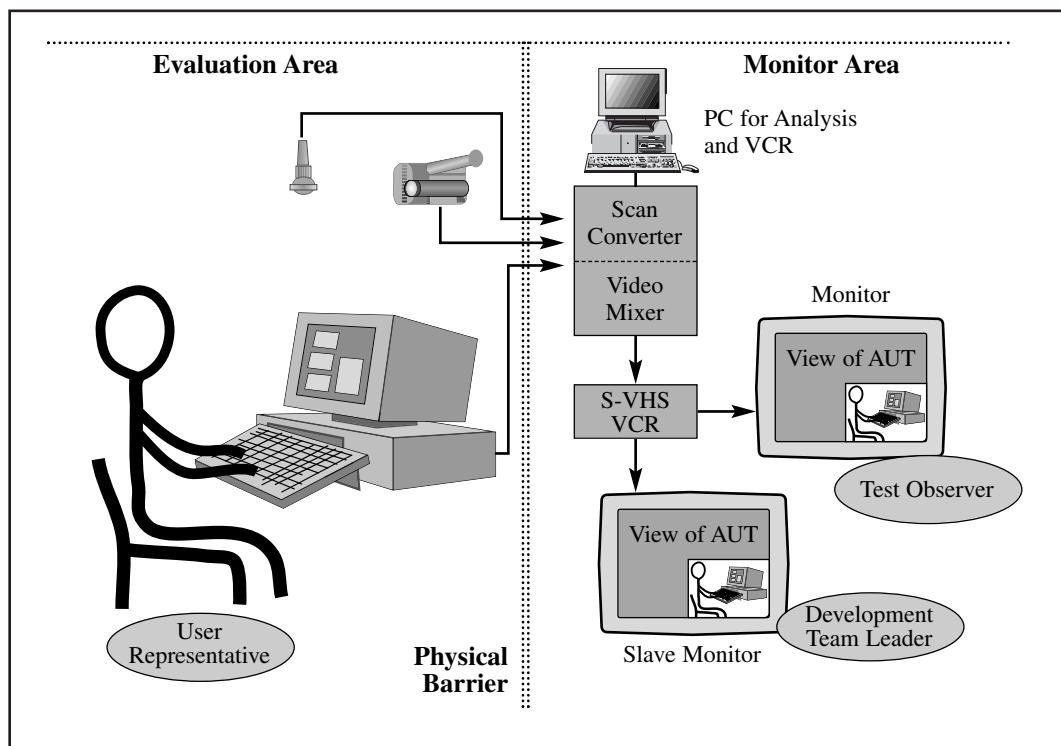


Figure O.1. Organization of a typical usability test.

Computer-aided statistical techniques are used to analyze the video footage of the usability test, which may be recorded on sophisticated professional videocassette recorder equipment or using digital recording technology.

The usability test results are documented in a report, which summarizes the results of the usability test and highlights any usability issues. A brief summary video, which highlights the usability issues documented in the report, may also be produced to provide visual feedback to the developers.

The usability report (and summary video, where available) is presented to the Development Team Leader at a Post-Usability Test meeting. The Test Team Leader will also present a summary of the results of the usability test and any usability issues to the Development Team Leader.

P

Testing Process Health Check

P.1 Introduction

This appendix contains a list of criteria to be used in performing a testing process health check and a suggested approach to conducting the task.

The testing process health check could be used in a number of ways:

- ▲ To baseline the current state of testing within an organization prior to introducing a formal testing process
- ▲ To monitor the progress of the adoption of a formal testing process and to identify areas that were satisfactory and areas that could be improved
- ▲ As part of a process-improvement program (see Chapter 13 – Improving the Testing Process).

The criteria used in the testing process health check are grouped under the following high-level headings:

- ▲ Testing and the Software Development Process
- ▲ Testing Process
- ▲ Roles and Responsibilities
- ▲ Testing Phases.

These groups and their individual criteria are described in full in the following sections respectively. Section P7 contains an optional scheme for scoring the various criteria if a more quantitative approach is required.

A checklist is provided at the end of this appendix, which can be copied and used as a memory aid during the testing process health check. This check list is also available from the following URL: us.cambridge.org/titles/052179546X.

P.2 Terminology

The term “subject” is used to denote the person, project, department, or organization that is the target of the testing process health check. It could refer to a third-party organization whose testing process needs to be assessed or it could be used internally within the host organization.

The term “auditor” is used to denote the person performing the testing process health check.

P.3 Testing and the Software Development Process

This section contains criteria addressing high-level software development and testing issues:

- ▲ Does the client have a formally documented process for software development, which incorporates testing issues?
- ▲ Does the client consider test requirements during all phases of the software development process? For example, are the system requirements reviewed to verify that they can be tested easily?
- ▲ Does the client have plans for software development projects, and can they show you examples of such plans? Do the plans explicitly show the points in the development at which testing will take place? That is, is testing performed at all stages of the development, and is adequate time assigned to testing?
- ▲ Does the client organization have a Quality System (such as ISO9001 or TickIT), and is it integrated into the software development and testing process?
- ▲ Does the client have a formally documented process-improvement scheme (such as CMM or SPICE)?
- ▲ Does the client make use of software metrics (that is, does the client collect and use metrics)? Are metrics collected with regard to testing and used to improve the process (e.g., improved estimating of testing timescales)?
- ▲ Who is responsible for ensuring that software development and testing standards are adhered to? For example, does the client have a QA Group or a Testing Group with a QA Manager or Testing Manager?
- ▲ How does the client manage requests for change/enhancement/defect (bug) fixes? How are these requests tracked, how are they verified, and how are they signed off? For example, is there some form of tool support for this process?
- ▲ Does the client develop and test software under rigorous Configuration Management?

P.4 Testing Process

This section contains criteria addressing client testing process issues:

- ▲ Does the client have a formally documented testing process (such as a Testing Strategy or a Testing Framework document)?

- ▲ Does the client have formally documented detailed plans for testing (such as a Test Plan document), and do the plans incorporate contingency for retesting defects?
- ▲ Does the client understand and make use of the V Model in planning the various testing phases (see Chapter 4)?
- ▲ Does the client have re-usable templates for standard testing documentation that all testing projects must use, including:
 - a *Test Plan Document*?
 - a *Test Specification Document*?
 - *Test Scripts*?
 - *Test Result Record forms*?
 - *Test Logs*?
 - *Test Certificates*?
 - a *Test Summary Report*?
 - *other documentation defined within the client process*?
- ▲ Does the client make re-usable templates for standard testing documentation widely available to all projects/groups engaged in testing within the organization in an easily usable form (e.g., as sheets in a manual that may be photocopied, on CD, via an Intranet system)?
- ▲ Does the client enforce the use of templates for standard testing documentation within projects/groups engaged in testing within the organization?
- ▲ Does the client have reproducibility of testing? (i.e., is the client confident that a given test environment could be recreated and the test rerun with the same result)?
- ▲ Does the client conduct the testing process under rigorous Configuration Management?
- ▲ Does the client have formally documented *Test Result Categories* and acceptable frequencies for defects?
- ▲ Does the client testing process formally support re-use (e.g., are testing projects responsible for compiling Re-use Packs and for making use of Re-use Packs compiled by previous testing projects)?
- ▲ Does the client make use of *automated testing tools* within its testing process to reduce timescales and save effort?
- ▲ Does the client ensure adequate test coverage (i.e., does the client ensure that all parts of the *Application Under Test (AUT)* are tested)?
- ▲ Does the client follow a process improvement program within the testing process?
- ▲ Does the client collect and use metrics within their testing process? If so, what metrics does the client collect, and how are they used in the testing process?

P.5 Roles and Responsibilities

This section contains criteria addressing the issues of staff roles and responsibilities within the client testing process:

- ▲ Does the client have formally documented roles and responsibilities for the staff involved in the testing process, such as:
 - *Testing Manager?*
 - *Test Team Leader?*
 - *Test Analyst?*
 - *Tester?*
 - *Independent Test Observer?*
- ▲ Are the reporting/liaison lines clearly defined for each role?
- ▲ Who is responsible for ensuring that testing is managed correctly:
 - *within the client organization?*
 - *within each project?*
- ▲ Who has responsibility for ensuring *independent observation* of testing (such as a QA representative or staff member not directly involved in the software development and testing)?

P.6 Testing Phases

This section contains criteria addressing the issues of testing phases within the client testing process:

- ▲ What formally documented testing phases does the client software development and testing process contain, and for each phase is there a specification of:
 - *what is tested within that phase?*
 - *who is responsible for managing that phase?*
 - *who is responsible for performing the testing?*
 - *who is responsible for independently observing the testing?*
 - *which testing techniques should be used?*
 - *the test result categories?*
 - *the starting and halting criteria?*
 - *the formal documentation that is needed (e.g., Test Scripts, Test Logs)?*
 - *the inputs to and outputs from the phase?*
- ▲ Does the client consider obtaining Re-use Packs (see Appendix I) from previous testing phases in order to save time and effort within the current testing phase?
- ▲ Does the client consider the requirements for and plan a testing phase during the corresponding phase of the software lifecycle (e.g., does the client plan *Acceptance Testing* during the Requirements phase)?
- ▲ Does the client consider re-use at the end of each testing phase (e.g., by compiling a Re-use Pack of the materials created during the testing phase – see Appendix I)?

P.7 Optional Evaluation Scheme

Although the overall result of conducting a testing process health check is to provide a qualitative view of the suitability of the testing process employed by the client, it may be useful under certain circumstances to take a quantitative approach to the evaluation.

For example, the auditor conducting the evaluation may find it useful to flag certain criteria or results that he or she feels need emphasis, or clients themselves may specify the use of such an approach in conducting the project.

Under such circumstances, the following scheme can be employed:

- ▲ Review the criteria and assign each a weighting, placing this value in the appropriate cell of the check list provided at the end of this appendix:
 - *1 for essential criteria*
 - *0.75 for important criteria*
 - *0.5 for desirable criteria*
 - *0 for non-applicable criteria*
- ▲ When evaluating the client testing process, consider each criterion and score it as follows:
 - *1 if the testing process fully satisfies the criterion*
 - *0.75 if the testing process largely satisfies the criterion*
 - *0.5 if the testing process partially satisfies the criterion*
 - *0 if the testing process does not satisfy the criterion*
- ▲ On completing the evaluation, multiply the score for each criterion by its weighting and total the scores.

This scheme must be used with some caution, since it may be difficult to interpret the precise meaning of any particular score a given client testing process may achieve.

Client	Department	Date of Testing Audit	Auditor Name

Criteria	Weighting (0–1)	Score (0–1)	Result
Testing and the Software Development Process			
Formally Documented S/W Development Process			
Test Requirements at All Phases of S/W Development			
Integrated S/W Development & Testing Plans			
Quality System in S/W Development & Testing			
Process Improvement in S/W Development & Testing			
Use of Metrics in the Development Process			
Responsibility for S/W and Testing Standards			
Management of Change Requests			
Use of CM in S/W Development			
Subtotal			
The Testing Process			
Formally Documented Testing Process			
Formally Documented Testing Plans			
Understanding and Use of the V Model in Planning			
Standard Testing Documentation			
Availability of Standard Testing Documentation			
Reproducibility of Testing			
Use of Rigorous CM in Testing			
Formally Documented Test Result Categories			
Support for Re-use			
Use of Automated Testing Tools			
Adequate Test Coverage			
Use of a Testing Process Improvement Program			
Use of Metrics in the Testing Process			
Subtotal			

Criteria	Weighting (0–1)	Score (0–1)	Result
Roles and Responsibilities			
Formally Documented Roles and Responsibilities			
Clearly Defined Reporting/Liaison Lines			
Management Responsibility for Client Organization			
Management Responsibility within Testing Project			
Responsibility for Independent Test Observation			
Subtotal			
Testing Phases			
Formally Documented Test Phases			
Access to Re-use Packs from Previous Testing Phases			
Test Planning at All Phases of S/W Development			
Re-use of Testing Materials			
Subtotal			
Miscellaneous Issues			
Subtotal			
Total Score			



The Testing of Object-Oriented Software

Q.1 Introduction

Irrespective of the technology employed in the development of a software system, any testing process will involve a number of common activities that must be performed by staff with appropriate skills and completed in a particular order to ensure successful testing. This is the essence of a repeatable, predictable, reusable process.

Where an organization has a requirement to test a number of different applications, each of which may have been developed using different tools, techniques, and languages, rather than emphasizing the differences between the applications and their implementation technologies, it is much more beneficial to look for the similarities.

Where differences that impact on the testing process are observed, these exceptions should be documented and the testing solutions that are adopted to address them incorporated into the process as supplementary activities for future use.

Thus, the starting point for determining how to approach the testing of any new or novel technology should be to assume that the testing will fit into the currently used process and not throw current best practice away in favor of creating a new process from first principles.

This appendix discusses the particular issues involved in testing object-oriented (OO) software systems (Reference 36) and makes a number of proposals for how to approach this activity. Specifically, Section Q2 briefly reviews the process of object-oriented software development, Section Q3 discusses the impact of OO on the management and planning of testing, and Section Q4 examines the impact of OO on the design of tests.

Q.2 Object-Oriented Development

Object orientation is one of many approaches available for developing computer applications. In the same way that high-level procedural programming languages

provided developers with significant benefits over assembler-based approaches, object-oriented programming (OOP) provides further benefits over procedural programming (see Reference 36, for example).

OO is fundamentally different from procedural programming in the way it views process and data. In a procedural application, data and process are separate elements; data is passed to procedures that manipulate the data and that pass it back when the processing is complete. In an object-oriented language, data and process are associated in programming entities termed “objects,” which are responsible for controlling access to and manipulation of their own data. When developing an application, OO provides a simple, “one-to-one” mapping between the entities that are to be modeled or represented in the domain of interest and their programming counterparts, the objects.

Another significant difference between OO and a procedural style of development is that of re-use (see Reference 37, for example). In writing OO applications, developers are encouraged to re-use previously created objects in order to save time, effort, and cost. This approach can also improve the quality of the delivered software by allowing developers to re-use tried and tested objects, whose modular nature combined with their well-defined formal interface make it a straightforward task to integrate the objects into the application.

OO includes not just the implementation technology required to support objects but also encompasses the philosophy of object-oriented development, which incorporates particular techniques for requirements capture, analysis, and design (see Reference 6, for example). The particular approach to application development employed in OO software projects is one of the main areas of impact that OO has on the testing process.

The Unified Modeling Language (Reference 6), or UML, is the foremost of the OO analysis and design methods and has largely achieved de facto standard status. UML includes a number of processes and notations for obtaining and representing information about the domain of interest. Use Cases (Reference 7), for example, provide a concise approach to obtaining and agreeing on the requirements of the AUT by supporting a formal dialog between user representative and analyst. The result is one or more simple diagrams and some associated text, which succinctly describes some specific interaction between the user and the system.

A number of software development processes, such as the iterative techniques (see Reference 39, for example) and rapid prototyping methods (Reference 12) are particularly well suited to use with object orientation, and further enhance the claimed productivity of OO development. Iteration breaks the traditional *waterfall* development model typically associated with procedural programming down into smaller steps – termed “iterations” (see Reference 39, for example). The result of each iteration is a complete deliverable, which can be tested to ensure it meets the particular requirements for that iteration. In this way, testing begins earlier in the development process, and a higher volume of testing can be performed during the course of the overall project.

The next section discusses the impact of object orientation on the process of test management and planning.

Q.3 Impact of OO Development on Test Management and Planning

Although an effective means of improving the productivity of developers (Reference 36), object orientation, in combination with iterative development techniques (Reference 39), Rapid Prototyping (Reference 12), and object re-use (Reference 37), provide a massive challenge to QA staff in terms of a significantly larger testing load as compared with more traditional development techniques.¹

This increased testing load presents the biggest impact of OO on the management and planning of testing. Although test management must always be as efficient as possible, the relatively higher volumes of OO code produced in shorter timescales means that test managers must focus on the means of increasing testing effectiveness.

Iterative Development

An iterative approach to development and testing can be viewed in a very positive light, since it brings testing further forward into the development process and permits more testing to be completed. The result of this process is to help manage risk and ensure a higher-quality deliverable.

However, in addition to earlier and more frequent testing, iterative development means that there will also be a significantly higher *Regression Testing* requirement. At the completion of each iteration it will be necessary to verify that the changes made during the current iteration have not had an adverse effect on the functionality implemented during earlier iterations. Furthermore, the amount of time and effort spent on Regression Testing will grow with each iteration because there is progressively more of the AUT to Regression Test.

Under such circumstances, it is essential that as much re-use of previously developed *Test Scripts* and *Test Cases* be employed. *Test Analysts* and *Testers* must be made aware of the need to ensure that Test Scripts are developed with re-use in mind and that these Test Scripts are rigorously copied and filed for future use. *Re-use Packs* are one means of helping to administer such an approach (see Appendix I).

Automated testing tools should also be considered to improve re-use in testing OO software, as well as to save testing time and effort and increase confidence in the quality of the software. The so-called record and playback testing tools (see Reference 22, for example) are very well tailored to testing object-oriented software, and use “object-recognition” techniques to identify objects in the AUT and to interrogate their properties. At the end of each iteration, the Test Analyst designs the set of Test Scripts to be used to verify that the requirements have been met for

¹Many advocates of OO would argue that object re-use assists the tester by allowing the developer to re-use tried and trusted objects and components. However, for complete confidence in the software, it is still necessary to ensure thorough testing.

that iteration. As these Test Scripts are executed, the automated testing tool is used to record the process, allowing the tests to be replayed in Regression Testing subsequent iterations. As the project proceeds, a comprehensive Regression Testing suite is built up that can be run overnight or over the weekend to perform unattended testing of the AUT.

Although a powerful technique for saving time, effort, and cost, some caution must be employed in introducing automated testing tools, since their use will suffer from an initial learning curve. Their introduction and adoption must be planned and managed, and organizations must persevere with their use to gain the benefits that an automated approach can bring. Although there are reported cases of organizations gaining benefits (that is, experiencing savings in time, effort, and cost over equivalent manual testing) after just two uses of such tools (see Reference 17), typically, organizations will need to use such tools three or four times to recoup their investment and begin to save time, effort, and money.

Chapters 16 and 18 (the Crown and ADP case studies) provide examples of how an automated testing approach can be used successfully to reduce the testing load and improve the quality of the AUT.

Rapid Prototyping

Rapid prototyping techniques provide users with an opportunity to see and use early trial versions of the AUT. This is a very powerful approach to managing the user expectations of the AUT and to obtaining rapid feedback regarding any mismatch between the user requirement and the implementation of the AUT. The difficulty for the testing process is that such approaches make it very much easier for users to ask for enhancements or changes to the AUT following exposure to the prototype software, with the concomitant issues of the need to update the requirements appropriately.

Where the AUT is being developed using rapid prototyping techniques, it is particularly important that *Test Managers* and Test Analysts have complete and full access to the most up-to-date requirements, and that they are able to assess the implications of changes to the requirements to the testing activity. To ensure rigor and efficiency, a formal process for managing change requests and documenting and maintaining requirements must be followed.

Tool support for managing the requirements in an OO development and testing project must be considered (see Reference 19, for example). Furthermore, the tool must be shared by Analysts, Developers, and Testers, and any changes to the requirements information must be automatically communicated to each of the roles involved in development and testing.

A key means of increasing efficiency is to ensure that the requirements management tool integrates with the testing tool to allow the Test Managers and Test Analysts to view the requirements for the AUT within the test tool, and to plan and design appropriate tests against those requirements. The new generation of development and testing tools that seamlessly integrate requirements, analysis, design, and testing facilities provide a powerful means of unifying the different roles involved in development while optimizing the performance of the individual roles such as that of the Tester (see References 23 and 25, for example).

If correctly integrated into the development and testing process, Rapid Prototyping can be a very effective means of managing the expectations of the users, preventing unexpected complaints at Acceptance Testing from users who are unhappy with the look and feel of the AUT or who are dissatisfied with its behavior or performance.

Object Re-use

The philosophy of re-use that object orientation promotes allows applications to be quickly built from libraries of existing objects. In addition to the benefits of reduced time and effort in development, the use of objects that are “tried and tested” means that the developer has a higher confidence in the quality of the resulting AUT. Although the majority of these objects are available from commercial object libraries, many organizations are becoming increasingly aware of the benefits of encouraging their developers to write software with re-use in mind.

The increase in re-use of existing objects and modules that OO promotes provides both benefits and difficulties for Testers. Although re-use means that more complex applications can be created more quickly by re-using libraries of predefined objects, the tester is faced with the challenge of how to manage the testing of the much higher volumes of code produced by such an approach. On the other hand, the opportunity for developing the AUT from trusted library objects should result in a product of higher quality.

As with any testing issue, the tester should be looking to exploit the benefits of OO while carefully managing its challenges. Automated testing has already been discussed in the previous section as a very effective means of managing the higher productivity that OO brings.

The use of effective configuration management (see Reference 20, for example) is a powerful approach to managing the testing of re-usable objects. Using such an approach, one may associate re-usable tests with the re-usable objects, allowing efficient testing of such objects by automatically incorporating those tests into the existing test suite.

As particular objects are used more and more frequently in the development of applications, it becomes possible to have increasing confidence in their quality as they become trusted objects. In such cases, it may be necessary only to test the integration of such objects or modules with other objects or modules when they are incorporated into the AUT, rather than having to test the functionality of the object itself.

Q.4 Impact of OO on Test Design

Those aspects of the testing process that employ *Black Box Testing* techniques are largely unaffected by the use of object-oriented programming because Test Cases designed using such techniques are constructed with no knowledge of the internal structure of the AUT or its components. Thus, the main area of testing that OO impacts on is Unit Testing, where the programmer will design Test Cases based on

knowledge of the structure and function of the code, and to a lesser extent, on *Integration Testing*.

Where developers have followed a UML approach in designing and implementing the AUT, this process can also be exploited by Test Analysts in designing effective and efficient tests. For example, Use Cases employed in requirements capture can be exploited in the design of realistic and effective *Thread Tests*.

The impact of OO in the testing phases is described in the following sections.

Unit Testing

The modularity and encapsulation of objects combined with their well-defined interfaces means that the process of Unit Testing an object is more straightforward than testing a corresponding unit in a procedural language.² In effect, the object acts as its own test harness – allowing the tester to invoke its functionality or inspect its state by making the appropriate message calls to the object.

Where UML is being used in the development of the AUT, supplementary information, such as that provided by the OCL (object constraint language; see Reference 38), may also be used by the Test Analyst in boundary analysis and partition design techniques.

The code-generation facilities of some of the OO design (or CASE – computer-aided software engineering) tools (such as Reference 34) can also be used to simplify the process of Unit Testing by allowing the objects that the unit under test interacts with to be created (albeit in a very basic form) so that the interactions between the adjacent objects can also be validated.

Where an integration exists between an OO design tool and an automated testing tool (such as that provided by Reference 25), it may also be possible to generate automatically Unit Tests based on the design information, providing very significant benefits in terms of time, effort, cost, and quality.

Integration Testing

The greatest impact of object-oriented development on the process of Integration Testing is obtained from the design process, where various UML diagrams (such as the collaboration diagrams) can be used by the Test Analyst to support the process of test design. Such diagrams describe the interactions between well-defined aggregations of objects (analogous to modules in an Integration Testing sense) and can form the basis of tests to verify:

- ▲ The correct invocation of one module from another interoperating module
- ▲ The correct transmission of data between modules

²Although “experienced programmers” have always strived to write well-encapsulated modular code with well-defined interfaces in whatever language or technology they have used, OO helps to enforce these principles on all developers. It is of course still possible to abuse the technology and produce poor code using OO.

- ▲ Nonfunctional issues, such as the reliability of the interfaces between modules.

As with Unit Testing, where an integration exists between an OO design tool and an automated testing tool, it may also be possible to automatically generate Integration Tests based on the design information.

Acceptance Testing

Use Cases have a major benefit of helping to ensure that there is no misunderstanding between the user and analyst regarding the requirements for the AUT – in essence, bridging the understanding gap between users who are expert in their domain but are typically IT-naïve and analysts who have IT expertise but are likely to be unfamiliar with the domain.

The resulting Use Cases provide a succinct description of the interaction between the user and the AUT, and include not just the “correct” use of the system but also the alternate courses of action (such as actions leading to failure or error conditions).

Thus Use Cases can provide a rich source information for the design of *Acceptance Tests*, describing both the navigational aspects of the Test Script (what the user does when interacting with the system) as well as the specific verifications needed in the Test Cases (what the user expects to see as a result of their actions).

Acceptance Tests can frequently be directly derived by inspection of relevant Use Cases, and often much of the language used in the Use Case can be copied into the Test Script (particularly where an OO CASE tool is being used).

Alternate flows (such as error conditions) can also be quickly produced by simply copying the Test Script describing the typical interaction of user and AUT and customizing it to reflect the steps involved in the alternate flow.

As with Unit Testing and Integration Testing, where the project team is using an integrated support tool for development and testing (such as that provided by Reference 25), it may be possible to view automatically the Use Cases from within the testing tool so that tests can be planned, designed, and executed against those Use Cases.



References

1. Hetzel, B. *The Complete Guide to Software Testing*, 2d ed. QED Information Sciences Inc, 1988.
2. Myers, G. J. *The Art of Software Testing*. John Wiley, New York, 1979.
3. CCTA. *PRINCE 2: Project Management for Business*, 4th ed., 1996.
4. *British Library Testing Course*. Ref: P6188A/T0, AMSL, 1997.
5. CCTA. *IT Infrastructure Library*. Gildengate House, Norwich, December 1993.
6. Booch, G., and J. Rumbaugh. *Unified Method for Object-Oriented Development*. Documentation Set v0.8. Rational Software Corporation, Santa Clara, CA, 1995.
7. "Getting Started: Using Use Cases to Capture Requirements." *Journal of Object-Oriented Programming*, September, 1994.
8. Krushten, P. *The Rational Unified Process*. Addison-Wesley, Reading, MA, 1998.
9. Jenny Preece et al. *Human Computer Interaction* Addison-Wesley, 1994.
10. *The Windows Interface Guidelines for Software Design*. Microsoft Press, 1995.
11. Kirakowsky, J. "The Software Usability Measurement Inventory: Background and Usage." In Taylor and Frances, *Usability Evaluation in Industry*, UK, 1996.
12. DSDM Consortium. *DSDM v2*. Tesseract Publishing, December 1995.
13. *SQA System Documentation*. Rational Software Corporation, Cupertino, CA, 1998.
14. *BS EN ISO 9001: 1994*. International Standards Organisation, 1994.
15. *Software Testing – Part 1: Vocabulary*. BS 7925-1:1998, BSI, August 1998.
16. *IEEE Standard for Software Test Documentation*. IEEE Std. 829-1998, IEEE, December 1998.
17. Graham, D., and M. Fewster. *Automating Software Testing*. Addison-Wesley, Reading, MA, 1999.
18. Gilb, T. *Principles of Software Engineering Management*. Addison-Wesley, Reading, MA, 1988.
19. *Rational RequisitePro Documentation*. Rational Software Corporation, Cupertino, CA, 2000.
20. *Rational ClearCase Documentation*. Rational Software Corporation, Cupertino, CA, 2000.
21. *Rational ClearQuest Documentation*. Rational Software Corporation, Cupertino, CA, 2000.
22. *Rational TeamTest Documentation*. Rational Software Corporation, Cupertino, CA.
23. *Rational PerformanceStudio Documentation*. Rational Software Corporation, Cupertino, CA.
24. *Software Systems and Their Development*. Reference: M301. The Open University, Milton Keynes, UK, 2000.
25. *Rational Suite TestStudio Documentation*. Rational Software Corporation, Cupertino, CA.
26. Boehm, B. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, NJ, 1981.
27. "European Systems and Software Initiative Process Improvement Experiment." Project 27581, www.cordis.lu/esprit/src/stessi.htm.
28. Bach, J. "James Bach on Risk-Based Testing." *Software Testing and Quality Engineering Magazine*, Vol. 1, No. 6, Nov./Dec. 1999.
29. *CRAMM User Guide*. Insight Consulting, Walton-on-Thames, UK, 1998.
30. "Ris3: Professional Risk Management Software," Line International Ltd, Bristol, UK, 2000.
31. Grady, B. R., and D. L. Caswell. *Software Metrics*. Prentice-Hall, Englewood Cliffs, NJ, 1987.
32. Schein, E. *Organizational Psychology*, 2d ed. Prentice-Hall, Englewood Cliffs, NJ, 1970.
33. *Complexity Estimating Software*. McCabe Associates, MD, 2000.
34. Quatrani, T. *Visual Modeling with Rational Rose and UML*. Addison-Wesley Longman, Reading, MA, 1998.
35. Paulk, M. C., et al. *Key Practices of the Capability Maturity Model – Version 1.1*. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.
36. Rentsch, T. *Object-Oriented Programming*. SIGPLAN Notices, Vol. 17, No. 9, September 1982.
37. Meyer, B. *Reusability: The Case for Object-Oriented Design*. IEEE Software, March 1987.
38. Muller, P.-A. *Instant UML*. Wrox Press Ltd., Birmingham, UK, 1997.

39. Boehm B. W. "A Spiral Model of Software Development and Enhancement." *Computer*, IEEE, May 1988.
40. *IEEE Standard for Software Test Documentation*. IEEE Std. 1002-1998, IEEE, 1998.
41. Gilb, T. and D. Graham. *Software Inspection*. Addison-Wesley, Reading, MA 1993.
42. Kit, E. *Software Testing in the Real World*. Addison-Wesley, Reading, MA 1995.
43. Beizer, B. *Software Testing Techniques*. Van Nostrand Reinhold, New York, 1990.
44. *Rational Quantify Documentation*. Rational Software Corporation, Cupertino, CA.
45. "British Telecommunications Verification, Validation & Testing Handbook." *British Telecommunications plc*, Issue 3, 1997.
46. *The Rational Unified Process*. Rational Software Corporation, Cupertino, CA.
47. *VisualTest Documentation*. Rational Software Corporation, Cupertino, CA.
48. *Rational Purify Documentation*. Rational Software Corporation, Cupertino, CA, 2000.



Glossary

Introduction

The glossary contains terms and phrases used in this book that require further explanation or definition. These terms and phrases appear in *italic type* the first time they are used in each chapter. The definitions conform to standard testing terms as specified in testing standards documents (see References 15, 16, and 40).

Acceptance Test Certificate

A document (typically 1-page long) used to formally record the successful completion of the *Acceptance Test* for a particular *Application Under Test (AUT)*. The certificate is signed by a senior representative of the organization accepting the AUT and the organization delivering the AUT (see Appendix H).

Acceptance Testing

Formal testing conducted to enable a User, customer, or other authorized entity to determine whether to accept a system or component.

This testing phase is used to test the system as a whole to ensure that it meets its stated business requirements and that it performs within defined constraints. This testing phase takes place after *System Testing* has been completed (also see *User Acceptance Testing* and *Operations Acceptance Testing*).

Application Under Test

The application or program that is the focus of the testing process. Application Under Test is abbreviated *AUT*, and is synonymous with *System Under Test (SUT)* in this book.

AUT

The abbreviation of *Application Under Test* (i.e., the application or program that is the focus of the testing process). AUT is synonymous with *System Under Test (SUT)* in this book.

Automated Testing

The use of software to control the execution of tests, the comparison of their observed results to predicted results, the setting up of test preconditions, and other test control and test reporting functions.

Automated Testing Tools

Software tools, such as Reference 13, that automate the testing process in order to reduce effort, timescales, and cost in the testing of software.

Automated Test Script

A series of instructions, in a computer-readable form, that allows an *Automated Testing Tool* to reproduce the navigation and *Verification Points* programmed into the tool to perform testing of the *AUT*. An Automated Test Script is analogous to a manual *Test Script*, with Verification Points analogous to manual *Test Cases*.

Behavioral Testing

See *Black Box Testing*.

Black Box Testing

Test Case selection based on an analysis of the *specification* of the *component* without reference to its internal workings. That is, testing where the Test Cases are designed without any knowledge of how the *AUT* is constructed. The Test Cases are derived from a knowledge of the *AUT*'s external behavior only.

Boundary Analysis

A *Test Case* design technique in which Test Cases are designed that include representatives of *Boundary Values* (see Chapter 3 and Appendix L).

Boundary Value

An input value or output value that is on the boundary between *Equivalence Classes* or at a significant distance either side of the value.

Boundary Value Analysis

See *Boundary Analysis*.

Build Verification Testing

A testing technique that verifies the navigational structure of the *AUT*. Typically, it is performed depth first; each menu and its component menu items (and submenus) are invoked to verify that they run some specific aspect of the *AUT*'s functionality or open a dialog box. In the latter case, further navigation is performed to verify that dialog box buttons invoke *AUT* functionality or open further dialog boxes. Build Verification Testing is synonymous with *Smoke Testing*.

Business Process Testing

A testing technique that verifies a typical business use of the *AUT* (e.g., the process of setting up a new user account in a banking system). Business Process Testing takes a typical scenario involving user interaction with the *AUT* and implements this as a test. Use Cases (see Reference 7) are often cited as a particularly good source of Business Process Tests.

Business Scenario

The combination of navigation through the *AUT* and the invocation of specific functionality to support a particular business use of the *AUT*. For example, in a banking application, the collection of steps involved in creating a new account could be considered to be a business thread.

Business Threads

See *Business Scenario*.

Configuration Testing

See *Installation Testing*.

Commercial Off-the-Shelf (COTS)

Commercially available software products/tools that can be purchased from vendors. For example, Microsoft Word and Rational TeamTest are examples of COTS products.

Compatibility Testing

Testing verifies that when the *AUT* runs in the *Live Environment* its operation does not impact adversely on other systems, and vice versa. Also see *Interoperability Testing*.

Component Testing

See *Unit Testing*.

Configuration Testing

See *Installation Testing*.

Crash Testing

A testing technique used to determine the fitness of a new build or release of an *AUT* to undergo further, more thorough testing. Crash Testing seeks to discover if the *AUT* is reliable and robust or if it “crashes” during routine use. In essence, Crash Testing is a “pretest” activity that could form one of the acceptance criteria for receiving the *AUT* for testing. Also see *Skim Testing*.

Development Environment

The IT environment (including the hardware, software, data, and support infrastructure) used by the *Development Team* to support the development of the *AUT*. Typically, the early phases of testing (such as *Unit Testing*) are conducted on the development environment. Also see *Live Environment* and *Test Environment*.

Development Team

The project team with responsibility for developing the *AUT*.

Development Team Leader

The staff member responsible for leading the team developing the *AUT*. Liaises with the *Test Team Leader* to report on the progress of the development of the *AUT* and likely dates for delivery of the *AUT* for testing purposes.

Domain Testing

See *Equivalence Class Testing*.

End Users

The staff members who will use the delivered *AUT* in support of their work. Also see *Operations Users*.

Equivalence Class

A set of the *Component's* input or output domains for which the *Component's* behavior is assumed to be the same from the *Component's Specification*. An equivalence class may also be termed an *Equivalence Partition*.

Equivalence Class Testing

A *Test Case* design technique for a *Component* in which *Test Cases* are designed to execute representatives from *Equivalence Classes*. This form of testing may also be termed *Domain* or *Partition Testing*.

Equivalence Partition

See *Equivalence Class*.

Equivalence Partition Testing

See *Equivalence Class Testing*.

Error Guessing

A *Test Case* design technique where the experience of the Tester is used to postulate what faults might be present in a *Component* or *AUT* and to design tests specifically to expose them.

Fault Recovery Testing

A form of testing to verify that following an error or exception, the *AUT* is restored to a "normal" state, such as the initial state of the *AUT* when it is first executed. This form of testing may also be used to verify the successful roll back and recovery of the data used or manipulated by the *AUT*.

Functional Requirements

Traditionally, those requirements for the *AUT* that have been specified within the *Functional Specification* Document for the *AUT*. Also see *Nonfunctional Requirements*.

Functional Specification

A document that describes in detail the characteristics of the *AUT* with regard to its intended capability.

Functional Testing

The process of testing the *AUT* to verify that it meets its *Functional Requirements*. These requirements are typically recorded in a *Requirements Specification* document. (also see *Nonfunctional Testing*).

Glass Box Testing

See *White Box Testing*.

Incremental Testing

A testing technique used in *Integration Testing*, where untested modules are combined with the module under test incrementally, with the functionality of the module (plus the added modules) being tested (retested) as each module is added.

Independent Test Observer

A staff member who is independent from the *Development* and *Testing* teams, who observes the testing process and is involved in signing off the *Test Result Record forms* for each *Test Case*.

Installation Testing

A testing technique used to validate that the *AUT* is able to run correctly on a variety of delivery platforms of various specifications (in terms of hardware, operating system, and software characteristics).

Integration Testing

Testing performed to expose faults in the interfaces and in the interaction between integrated components. This testing phase is used to identify problems with communications between modules or units and with resource contention and to verify the system design. This testing phase takes place after *Unit Testing* but before *System Testing* and may also be termed *Link* or *Module Testing*.

Interoperability Testing

Interoperability Testing verifies that when the *AUT* is operating in the *Live Environment* it is able to communicate successfully with other specified systems (such as invoking, passing data to, or receiving data from another system). Also see *Compatibility Testing*.

IT

An acronym for the computing term *Information Technology*.

IT Systems Administrator

The staff member within an organization with responsibility for administering the corporate *IT* facilities and for performing the day-to-day operations of the system (such as installation of software, upgrade of operating system software, backup, and archive and preventive maintenance).

IT Systems Manager

Synonymous with the term *IT Systems Administrator*.

Link Testing

A testing phase used to identify problems with communications between modules or units and with resource contention and to verify the system design. This testing phase takes place after *Unit Testing* but before *System Testing* and may also be termed *Integration* or *Module Testing*.

Live Data

The actual data used by an organization to support its business activities. This is distinct from any test data that may have been generated to represent the Live Data for the purposes of testing.

Live Environment

The actual *IT* environment (including the hardware, software, data, and support infrastructure) used by an organization to support its business activities. This is distinct from any testing environment that may have been set up to represent the live environment for the purposes of testing. Also see *Development Environment* and *Test Environment*.

Load Testing

A *Nonfunctional Testing* technique that determines the *AUT*'s ability to cope with sudden instantaneous peak loads (such as the sudden attempt by all users of a system to log on simultaneously) (as opposed to the sample operating conditions that are likely to have been used during development of the *AUT*). See also *Performance*, *Stress*, *Volume*, and *Usability testing*, and Chapter 3, Testing Techniques].

Millennium Testing

A testing activity used to identify and test applications to assess their capacity to operate and restart without disruption resulting from the processing of times/dates from before the year 2000 date change until February 29, 2000 and beyond, and where necessary, to modify systems to support the above objective. Millennium Testing is synonymous with Year 2000, Y2K, and Date 2000 testing.

Module Testing

A testing phase that is used to identify problems with communications between modules or units and with resource contention and to verify

the system design. This testing phase takes place after *Unit Testing* but before *System Testing* and may also be termed *Integration* or *Link Testing*.

Negative Testing

Testing aimed at demonstrating that the software does not behave in a manner prohibited in its *Specification*.

Nonfunctional Requirements

Within the context of testing, *Nonfunctional Requirements* refers to the testing of those aspects of the *AUT* not specified within the *Functional Specification* document for the *AUT*. For example, *Performance*, *Volume*, *Stress*, and *Usability testing* are examples of Nonfunctional Testing (also see Chapter 3, Testing Techniques, for further details of these Nonfunctional Testing techniques).

Nonfunctional Testing

The testing of requirements that do not relate to functionality. Literally, the testing of those aspects of the *AUT* that are not explicitly specified in the Functional Specification document (although in practice, issues of *Performance* and *Usability* may be well defined in exemplary examples of such documents). Non-functional testing includes *Performance*, *Stress*, *Volume*, *Reliability*, *Security*, and *Usability testing* (also see *Functional Testing*).

Operations Acceptance Testing

A form of *Acceptance Testing* conducted from the *Operations* perspective (i.e., involving representatives from the Operations staff) (also see *Acceptance Testing*).

Operations Representative

A nominated member of the operations community for the *AUT* who will be expected to participate in or to nominate other operations staff to participate in the *Operations Acceptance Testing* of the system. (It is important that the Operations Representative be a genuine user of the system and not, for example, a manager who manages *Operations Users* of the system.)

Operations Users

The staff members who will execute the administrative aspects of the delivered software system in support of the *End Users*. For example, typical tasks performed by the Operations Users include setting up User access and privileges, backup and archival of system data and system recovery.

Partition Testing

See *Equivalence Class Testing*.

Performance Testing

Testing conducted to evaluate the compliance of a *Component* or *AUT* with specified performance requirements. Performance Testing is a particular type of *Nonfunctional Testing* that measures the speed of execution and response of the *AUT* under typical working conditions with the intention of determining that these characteristics will meet the User's requirements (expectations) for the *AUT*. See also *Stress*, *Volume*, and *Usability testing* and Chapter 3, Testing Techniques).

Pseudocode

A form of structured English used by programmers in developing the program code to represent the functionality, structure, and flow of the application. Frequently used in *Static Testing* techniques (such as code review and code walkthrough; see Chapter 3, Testing Techniques).

QA

An acronym for the term *Quality Assurance*.

Rapid Application Development

A software development approach that emphasizes the rapid delivery of the software under development. Such approaches typically engage the *End Users* in the development process as early as possible by the use of prototypes and mock-ups of the system. The approach employs stringent planning and management using an iterative development style with strictly observed milestones and budgets (see Reference 12, for example).

RAD

See *Rapid Application Development*.

Regression Testing

A testing technique in which the *AUT* is tested following modifications or extensions to ensure that the overall functionality of the system has not been compromised. Regression Testing may also take place following changes to the environment in which the system is resident (e.g., changes to the Operating System or to hardware).

Reliability Testing

A *Nonfunctional Testing* requirement that verifies the *AUT* is robust and reliable in normal use, ensuring, for example, that the *AUT* does not suffer from catastrophic failures (such as a GPFs on Windows platforms) or from memory leak problems.

Requirements Specification

A document that records the requirements for a particular application or software system. These are traditionally obtained from the intended Users of the system by an analyst using some form of requirements capture method (such as Use Case modeling – Reference 7). The Requirements Specification document is synonymous with the *Software Requirements Specification* document.

Re-use Packs

Documents that allow *Testers* to quickly and easily rerun some or all the tests performed during an earlier testing phase. Where modifications have been made to a system, Re-use Packs can speed up the process of *Regression Testing*. Similarly, Re-use Packs can enable selected tests to be re-run in later testing phases should they be required (see also Appendix I, Re-use Pack Checklist).

Safety Testing

A testing technique intended to validate that the *AUT* or *SUT* meets its requirements in terms of safety of operation. This technique is particularly appropriate for testing Safety Critical Systems.

Security Testing

Testing focused on ensuring the *AUT* and/or its data are accessible only to authorized *Users*. Security tests can be designed, implemented, and executed within any testing phase, but are typically conducted at *System Testing*.

Simulation

The representation of selected behavioral characteristics of one physical or abstract system by another system. In testing terms, that is the use of a computer program to represent elements of the testing environment that are not available during testing. For example, if the *AUT* is being developed and tested on a standalone workstation but will be delivered on a networked system, it may be necessary to simulate typical network traffic during testing (also see *Test Harness*).

Simulator

A software system or computer program responsible for providing *Simulation* services in support of the testing of an *AUT*.

Skim Testing

A testing technique used to determine the fitness of a new build or release of an *AUT* to undergo further, more thorough testing. In essence, a “pretest” activity that could form one of the acceptance criteria for receiving the *AUT* for testing (also see *Crash Testing*).

Smoke Testing

A testing technique that verifies the navigational structure of the *AUT*. Typically performed depth first, each menu and its component menu items (and submenus) are invoked to verify that they run some specific aspect of the *AUT*’s functionality or open a dialog box. In the latter case, further navigation is performed to verify that dialog box buttons invoke *AUT* functionality or open further dialog boxes. Smoke Testing is synonymous with *Build Verification Testing*.

Software Requirements

See *Requirements Specification*.

Specification

See *Requirements Specification*.

SRS

Common abbreviation of *Software Requirements Specification*.

Static Testing

The inspection of the *AUT* in isolation (that is, not in a running state). Static Testing typically takes place early in the development of the *AUT* and involves techniques such as code review and code walkthrough (also see Chapter 3, Testing Techniques).

Stress Testing

Testing conducted to evaluate a system or *Component* at or beyond the limits of its specified requirements. That is, a particular type of *Nonfunctional Testing* that examines the ability of the *AUT* to cope with instantaneous peak loads with the aim of identifying defects which appear only

under such adverse conditions. Also see *Performance*, *Volume*, and *Usability testing* and Chapter 3, Testing Techniques].

Structural Testing

See *White Box Testing*.

Systems Integration Testing

An optional testing phase (typically for software systems which have a significant requirement to work or interoperate with other software systems) that is used to ensure that the *AUT* can interoperate successfully with any other software systems with which it needs to communicate, and to ensure that the *AUT* is compatible with other systems that may be running in the same environment. Systems Integration Testing takes place before *Acceptance Testing* and after *System Testing* (under certain circumstances, System Integration and System Testing may be performed together).

System Testing

The process of testing an integrated system to verify that it meets specified requirements. That is, a testing phase that is used to test the *AUT* as a complete system to identify any functional problems not found in *Unit Testing*, to assess *Nonfunctional requirements* such as *Performance* and *Usability*, and to establish confidence that the system will be accepted by the Users. System Testing takes place after *Integration Testing* and before *Acceptance Testing*.

System Under Test

The application or program that is the focus of the testing process. System Under Test is abbreviated *SUT* and is synonymous with *Application Under Test (AUT)* in this book.

SUT

The abbreviation of *System Under Test* (i.e., the application or program that is the focus of the testing process). *SUT* is synonymous with *Application Under Test (AUT)* in this book.

Test Analyst

A member of the Testing Team responsible for inspecting the *Requirements Specification* document for the *AUT*, analyzing the test requirements, designing the *Test Cases*, and drafting the *Test Scripts* (see Appendix A for complete *TORs* of the staff members involved in the testing process).

Test Automation

See *Automated Testing*.

Test Case

A specific test intended to verify one particular aspect or requirement of the *AUT*, and which will contain a set of input values, execution preconditions, and predicted outcomes and objective for the test. A *Test Script* may contain one or more Test Cases.

Test Data

The data used during testing to exercise the *AUT*. To provide complete confidence in the results of testing, the test data should be *Live Data*;

however, there are many reasons why it may not be appropriate to use Live Data (e.g. where the data contains confidential data or information relating to security issues, or where the testing may threaten the integrity of the Live Data). Under these circumstances, it may be possible to use a copy of the Live Data. If neither of these options is available, it may be necessary to generate handcrafted data. Handcrafted data can also be introduced into the Live or copied data in order to explicitly test some boundary or error condition in the AUT. Also see *Live Data*.

Tester

A member of the *Testing Team* responsible for conducting testing of the AUT by following a *Test Script* (see Appendix B for complete *TORs* of the staff members involved in the testing process).

Test Error

A specific *Test Result Category*. The Test Error result indicates that the associated *Test Case* failed, but that the AUT performed correctly; that is, the Test Case itself was in error (e.g., a typographical error in the Test Case).

Test Harness

A program used to test a portion of the AUT (such as a module or unit) by representing other interoperating elements of the AUT. For example, during *Unit Testing*, it may be necessary to represent other units that provide or receive data from the unit under test (also see *Simulation*).

Testing Environment

The IT environment (including the hardware, software, data, and support infrastructure) used by the *Test Team* to support the testing of the AUT. Also see *Development Environment* and *Live Environment*.

Testing Manager

A staff member responsible for managing the entire testing process within an organization (see Appendix A for complete *TORs* of the staff members involved in the testing process).

Testing Program

The complete organizational specification for an organization engaged in performing formal, rigorous testing of software systems (see Chapter 4, *Management of Testing* for details).

Testing Project

A project whose goal is verify the suitability of the AUT for its intended purpose

Test/Testing Team

A dedicated team responsible for performing the higher phases of testing of the AUT (typically, *System Testing* and above). The Testing Team is managed by a *Test Team Leader* and contains one or more *Test Analysts* and one or more *Testers*.

Test Log

A chronological record of all relevant details of a testing activity (Appendix G provides a template for a Test Log).

Test Plan Document

One of the artifacts generated by the *Test Team Leader* during a testing project. The Test Plan document is used as the basis of project management control throughout the testing process and contains information specifying the approach to testing and any constraints, risks, and dependencies; the test assumptions and exclusions; test entry and exit criteria, project controls; and a plan for completing the testing (including a contingency for any retest activities). Appendix C contains a Test Plan Document template.

Test Repository

The means of organizing all of the testing assets (such as *Test Cases* and *Test Logs*, etc.) employed by a project using an *Automated Testing Tool* during a testing project.

Test Result Category

A member of a set of results that can be applied to the outcome of a particular *Test Case*. For example, typical Test Result Categories include:

- (a) Pass – the observed test result conforms to the expected result.
- (b) Acceptable – the observed result indicates that the *AUT* differs from the agreed specification but is acceptable, requiring no change to the system but a change to the functional specification.
- (c) Tolerable – the observed result is incorrect; the *AUT* is workable and will be accepted, but the fault must be rectified within an agreed time.
- (d) Intolerable – the observed result is incorrect and the fault must be corrected before the *AUT* passes this testing phase.
- (e) Test Error – the observed result is correct, but the expected result is incorrect.

Test Result Record Form

A form used to record the results of executing a particular *Test Case*. A template copy of a Test Result Record Form can be found in Appendix F.

Test Rig

A flexible combination of hardware, software, data, and interconnectivity that can be configured by the *Test Team* to simulate a variety of different *Live Environments* on which an *AUT* can be delivered.

Test Script

A prescriptive document describing in detail how a test is to be conducted, the inputs and the expected behavior of the item being tested. A Test Script contains a number of *Test Cases*.

Test Specification document

One of the artifacts generated by the *Test Team Leader* during a testing project. The Test Specification Document provides the specification for the testing process required for formal verification of the *AUT*. Appendix D contains a Test Specification Document template.

Test Team

The team responsible for conducting the testing process on the *AUT* (see also Chapter 4, Management of Testing and Appendix A, Terms of Reference for Testing Staff).

Test Team Leader

A member of the Testing Team responsible for managing the day-to-day tasks of the *Testing Team* (see Appendix A for complete *TORs* of the staff members involved in the testing process).

Thread Testing

A testing technique used to test the business functionality or business logic of the *AUT* in an end-to-end manner, in much the same way a *User* or an operator might interact with the system during its normal use.

Tool Support

The use of an *Automated Testing Tool*, such as Reference 13, to support the testing process in order to reduce effort, timescales, and cost in software testing.

TOR

Terms of Reference (see Appendix A, Terms of Reference for Testing Staff).

Usability Testing

A particular type of *Nonfunctional Testing* that focuses on the ease of use of the *AUT* from the perspective of the *End User*. Issues to be considered in Usability Testing include the intuitiveness, consistency, and clarity or understandability of the user interface. See also *Performance*, *Stress*, and *Volume testing*.

User

See *End User*.

User Acceptance Testing

A form of Acceptance Testing conducted from the *User* perspective (i.e., involving User representatives) (also see *Acceptance Testing*).

User Representative

A nominated member of the User community for the *AUT* who will be expected to participate in or to nominate other Users to participate in the *User Acceptance Testing* of the system (it is important that the User Representative be a genuine User of the system and not, for example, a manager who manages Users of the system).

V&V

An abbreviation of *Verification* and *Validation*. This is the process of confirming that the *AUT* meets its requirements and has been developed using best practice/formal process.

Validation

"Are we building the product right?" The process by which it is confirmed that the *AUT* has been developed in line with best practice and applicable standards. Also see *Verification*.

Verification

"Are we building the right product?" The process by which it is confirmed that the *AUT* meets its requirements. Also see *Validation*.

Verification Point

An element of an *Automated Test Script* (that is, a script which can be run using an *Automated Testing Tool*) that verifies some aspect of the functionality of the *AUT*. A Verification Point in an automated Test Script is analogous to a *Test Case* in a manual *Test Script*.

Volume Testing

A *Nonfunctional Testing* technique that determines the *AUT*'s ability to cope with large volumes of data (as opposed to the sample data that is likely to have been used during development of the *AUT*). See also *Performance*, *Stress*, and *Usability testing* and Chapter 3, *Testing Techniques*).

Waterfall Development

A particular approach to software development that views the process as a series of linear steps each of which (in the strictest sense of the term) must be completed before the following step can begin. The steps traditionally include requirements, specification, design, and implementation.

White Box Testing

Testing where the *Test Cases* are designed with the knowledge of how the *AUT* is constructed. White Box Testing may also be termed *Glass Box Testing* (also see *Black Box Testing*).



Index

- Acceptance Testing, 38, 73–79
 - in case studies, 138–39, 151–52, 162, 175–76
 - of object-oriented software, 283
- adoption of testing processes, 107–11
- artifacts, *see* documentation
- automated software testing, 18–19
- automated testing tools
 - in case study, 169
 - in Regression Testing, 90, 91, 93
 - selection criteria for, 261–74
- Automatic Data Processing (ADP) Limited case study, 119, 167–68
 - documentation in, 178–79
 - management and planning of testing in, 169–71
 - testing phases in, 174–77
 - testing requirements of, 168–69
 - testing roles and responsibilities in, 171–74
- behavioral testing, 17
- Black Box Testing, 17
 - Integration Testing as, 53, 54
 - Operations Acceptance Testing as, 81
 - Regression Testing as, 89–90
 - Systems Integration Testing as, 65
 - System Testing as, 59
 - User Acceptance Testing as, 73
- Boundary Analysis, 20, 257–58
- British Library case study, 115–16, 121–28
- Business Specification documents (BSDs), 145, 152–53
- case studies
 - Automatic Data Processing Limited, 119, 167–79
 - British Library, 115–16, 121–28
 - Crown Quality Assurance Group, 117–18, 143–54
 - Reuters Product Acceptance Group, 116–17, 129–41
 - Wine Society, 118–19, 155–65
- Client Representatives, 149–50
- commercial off-the-shelf software (COTS), 17, 156
- communications, 101
- Compatibility testing, 23
- Compatibility Testing (Systems Integration Testing), 65
- complexity of applications, 100
- computed metrics, 98–99
- confidentiality, 24–25
- Configuration/Installation Testing, 22
- configuration management (CM), 42, 55
- configurations, testing of, 22
- Cost/Effort measurement, 100
- Crash Testing, 147, 152
- Crown Quality Assurance Group case study, 117–18, 143
 - documentation in, 152–53
 - management and planning of testing in, 145–47
 - process improvement in, 153–54
 - testing phases in, 150–52
 - testing requirements in, 144–45
 - testing roles and responsibilities in, 147–50
- data integrity, testing of, 24–25
- data requirements
 - for Integration Testing, 55
 - for Regression Testing, 91–92
 - for Systems Integration Testing, 67
 - for System Testing, 60–61
 - for Unit Testing, 47–48
 - for User Acceptance Testing, 75
- defect detection effectiveness percentage (DDE), 101–2, 154
- defect removal effectiveness percentage (DRE), 102, 154
- defects
 - error guessing for, 17–18
 - measurement of, 100
 - tracking, 42–43
 - “zero defect software,” 11
- Developer Representatives, 149
- development phases, in V Model, 40
- Development Project Managers, 172
- Development Team Leaders, 35–37
 - Integration Testing under, 55–56
 - Unit Testing under, 45, 48, 49
- difficulty of software development, measurement of, 101

- documentation
 - for automated testing tools, 264
 - automated testing tool selection criteria, 261–74
 - in case studies, 127–28, 152–53, 163–64, 178–79
 - Operations Acceptance Testing of, 81
 - Re-use Packs, 247–48
 - in Reuters Product Acceptance Group case study, 140–41
 - Test Certificates, 245–46
 - testing of, 23
 - testing process health check, 279–83
 - Test Logs, 241–44
 - Test Plan documents, 211–19
 - Test Result Record Forms, 237–39
 - Test Scripts, 231–35
 - Test Specification documents, 221–30
 - Test Summary Reports, 249–54
 - User Acceptance Testing of, 73
- document templates, *see* templates
- Equivalence Partitioning, 19–20, 255–56
- error guessing, 17–18
- European Software Systems Initiative, 143
- Exploratory Testers, 34
- Fault Recovery Testing, 23
- Flood Testing (Volume Testing), 26
- Functional Specification documents (FSDs), 146, 153
- functional testing techniques, 19–22
- general testing techniques, 16–19
- Graphical User Interface (GUI) Acceptance Testing, 131, 132
 - testing staff for, 134–35
- Hawthorne Effect, 102
- help facilities, testing of, 23
- human resources, *see* testing staff
- Independent Test Observers, 32–33, 189–90
 - in Acceptance Testing, 38
 - in case studies, 126–27, 136
 - in Integration Testing, 36, 56
 - in Regression Testing, 92
 - in Systems Integration Testing, 37, 68
 - in System Testing, 37, 61
 - in Unit Testing, 48–49
 - in User Acceptance Testing, 76
- inputs
 - to Integration Testing, 57
 - to Operations Acceptance Testing, 86
 - to Regression Testing, 94
 - to Systems Integration Testing, 69
 - to System Testing, 63
 - to Unit Testing, 50
 - to User Acceptance Testing, 77–78
- installation, testing of, 22
- Integration Testing, 35–36, 53–58
 - in case studies, 127, 137–38
 - of object-oriented software, 292–93
 - testing guide for, 200–201
- Interoperability Testing, 23
- introduction of testing processes, 107–11
- Intrusive Testing, 20–21
- Invitations to Tender (ITTs), 169
- iterative development, 289–90
- legacy applications, testing of, 17
- Link Testing (Integration Testing), 53, 137–38
- Load Testing (Stress Testing), 25
- maintenance of testing process, 111–12
- management of testing, 41–42
 - in case studies, 122–24, 132–33, 145–47, 157–59, 169–71
 - of object-oriented software, 287–93
- metrics, 97–99
 - proposal for metrics set, 104–6
 - setting up and administering metrics programs, 102–4
 - use within testing process of, 99–102
- Millennium Testing, 89
- modules, 53
- Module Testing (Integration Testing), 53
- Negative Testing, 9, 16–17
- nonfunctional testing techniques, 22–26
- object-oriented software, 287–93
- operating systems, 89
- Operations Acceptance Testing, 38, 81–87
 - testing guide for, 206–7
 - User Acceptance Testing distinguished from, 73
- Operations Representatives, 81–85
- organization of testing, 28–34
- outputs
 - from Integration Testing, 57–58
 - from Operations Acceptance Testing, 86–87
 - from Regression Testing, 94–95
 - from Systems Integration Testing, 70
 - from System Testing, 63–64
 - from Unit Testing, 50–51
 - from User Acceptance Testing, 78

- Performance Testing, 23–24
- personnel, *see* testing staff
- phases of testing, 34–39
 - in case studies, 127, 136–39, 150–52, 161–63, 174–77
 - testing process health check of, 272
- pilot studies, 110
- planning, 27–28
 - in case studies, 122–24, 132–33, 157–59, 169–71
 - in Crown Quality Assurance Group case study, 145–47
 - of Integration Testing, 56
 - management of test requirements in, 41–42
 - of object-oriented software testing, 289–91
 - of Operations Acceptance Testing, 85
 - organization of testing in, 28–29
 - of Regression Testing, 93
 - role of risk in, 43–44
 - of Systems Integration Testing, 68–69
 - of System Testing, 62–63
 - of Unit Testing, 49
 - of User Acceptance Testing, 76–77
 - V Model role in, 39–41
- platforms, changes of, 89
- Positive Testing, 9, 16–17
- primitive metrics, 98
- PRINCE project management methodology, 122, 145, 158, 159, 164
- Product Development Managers, 172
- Project Managers, 159–60
- QA Analysts, 173–74
- QA Team Managers, 147–48
- Quality Assurance Representatives (QARs), 32
 - in Acceptance Testing, 38
 - in Integration Testing, 36
 - in Systems Integration Testing, 37
 - in System Testing, 37
 - in Unit Testing, 35, 49
- quantitative measurements, *see* metrics
- Random Testing, 21
- Rapid Application Development (RAD)
 - approach, 156
- rapid prototyping techniques, 290–91
- Rational Suite (SQA) testing product, 169
- Reference Database (RDB) Acceptance Testing, 131, 134–35
- Regression Testing, 38, 89–95
 - automated software testing in, 18–19
 - in case studies, 139, 152, 162–63, 176–77
 - testing guide for, 208–9
- Reliability Testing, 24
- Requirements Specification documents, 16
- re-use of objects, 291
- Re-use Packs, 31, 35
 - checklist for, 247–48
- Reuters Product Acceptance Group case study, 116–17, 129
 - documentation in, 140–41
 - management and planning of testing in, 132–33
 - testing phases in, 136–39
 - testing requirements in, 130–32
 - testing roles and responsibilities in, 133–36
- reviews of testing processes, 111–12
- risk, role of, in test planning and management, 43–44
- Security Testing, 24–25
- Senior QA Analysts, 173
- size of applications, 99–100
- Skim Testing, 175
- software component testing, *see* Unit Testing
- software metrics, 98
- software testing process
 - automated, 18–19
 - definitions and purposes of, 8–10
 - see also* testing; testing process
- staff, *see* testing staff
- standards, 13
- State Transition Analysis, 21, 259–60
- Static Testing, 22
- stopping testing, 39
- Stress Testing (Load Testing), 25
- structural testing, 17
- Supplier Representatives, 161
- Systems Integration Testing, 37, 65–70
 - in case study, 137–38
 - System Testing distinguished from, 59
 - testing guide for, 204–5
- System Testing, 36–37, 59–64
 - in case studies, 137–38, 150–51
 - testing guide for, 202–3
- templates
 - for automated testing tool selection criteria, 261–74
 - Re-use Pack checklist, 247–48
 - for Test Certificates, 245–46
 - testing process health check, 284–85
 - for Test Logs, 243–44
 - for Test Plan documents, 212–13
 - for Test Result Record Forms, 239–40
 - for Test Scripts, 233–36
 - for Test Specification documents, 222–30
 - for Test Summary Reports, 250–54

- Test Analysts, 31–32, 188–89
 - in case studies, 125–26, 135, 148–49
 - management of test requirements by, 41
 - Operations Acceptance Testing by, 82, 84
 - QA Analysts as, 173–74
 - Regression Testing by, 92, 93
 - User Acceptance Testing by, 74, 76
- Test Automation Analysts, 33–34
- Test Automation Architects, 33
- test case design effectiveness percentage (TDE), 102
- Test Certificate template, 246
- Testers, 32
 - in case studies, 126, 136, 148–49, 158
 - Operations Acceptance Testing by, 84
 - Regression Testing by, 92
 - for System Testing, 61
 - User Acceptance Testing by, 76
 - testing, 7–8, 11–12
 - automated software testing, 18–19
 - definitions and purposes of, 8–10
 - error guessing in, 17–18
 - functional techniques of, 19–22
 - general techniques of, 16–19
 - nonfunctional techniques of, 22–26
 - of object-oriented software, 287–93
 - standards for, 13
 - stopping, 39
 - techniques of, 15–16
 - verification and validation in, 10
 - White Box and Black Box, 17
- Testing Consultants, 160–61
- testing guides, 197–209
- Testing Managers, 30, 111, 122–25, 184–85
- testing process
 - guides for, 197–209
 - health check of, 279–85
 - improving, 97–98
 - introduction and adoption of, 107–11
 - maintenance of, 111–12
 - management of test requirements in, 41–42
 - metrics used in, 98–106, 99–104
 - organization of, 28–29
 - phases of, 34–39
 - planning of, 27–28
- testing process documents, 112
- Testing Program Boards, 30, 194–95
- testing staff
 - in case studies, 124–27, 133–36, 147–50, 159–61, 171–74
 - for Integration Testing, 55–56
 - for Operations Acceptance Testing, 84, 85
 - organization of, 28–29
 - for Regression Testing, 92–93
 - roles and responsibilities of, 29–34
 - for Systems Integration Testing, 67–69
 - for System Testing, 61–63
 - terms of reference for, 183–95
 - testing process health check of, 284–85
 - for Unit Testing, 48–49
 - for Usability Testing, 275–76
 - for User Acceptance Testing, 75–77
- Test Log template, 241–44
- Test Plan documents, 30–31, 211–30
- Test Planners, 135
- Test Result Record Forms
 - Integration Testing documented on, 36
 - template for, 237–39
 - Tester's responsibility for, 32
 - Unit Testing documented on, 35
- Test Scripts
 - template for, 233–36
 - Test Analyst's responsibilities for, 31
 - Tester's responsibility for executing, 32
- Test Specification documents, 31, 221–30
- Test Summary Report template, 249–54
- Test Team Leaders, 30–31, 41, 186–87
 - in case study, 125
 - for Integration Testing, 56
 - for Operations Acceptance Testing, 83–85
 - for Regression Testing, 92, 93
 - for Systems Integration Testing, 65, 67, 68
 - for System Testing, 59, 61–62
 - for Unit Testing, 49
 - for User Acceptance Testing, 76
- Thread Testing, 22, 81, 168
- Unified Modeling Language (UML), 288
- units, 46
- Unit Testing (software component testing), 35, 45–51
 - in case study, 137–38
 - of object-oriented software, 287–93
 - testing guide for, 198–99
- Usability Testing, 25–26, 275–77
- User Acceptance Testing, 38, 73–79
 - Operations Acceptance Testing distinguished from, 81
 - testing guide for, 206–7
- user interfaces
 - Graphical User Interface (GUI) Acceptance Testing, 131, 132
 - Usability Testing of, 25
- User Representatives, 36, 37
 - in case study, 158, 160
 - in Operations Acceptance Testing, 85

- in Systems Integration Testing, 68
 - in System Testing, 62
 - in Unit Testing, 49
 - in User Acceptance Testing, 73–77
- User Representative Testers, 158, 160
- User Testing, 139
- validation, 10, 41
- verification, 10
- V Model
 - in case studies, 123–24, 132, 145, 146, 170–71
 - Integration Testing in, 53–54
 - Operations Acceptance Testing in, 81–82
 - planning role of, 39–41
 - Regression Testing in, 90
 - risk identified in, 43
 - Systems Integration Testing in, 65–66
 - System Testing in, 59–60
 - Unit Testing in, 46
 - User Acceptance Testing in, 73–74
- Volume Testing (Flood Testing), 26
- Web sites on testing, 12
- White Box Testing, 17, 45
- Wine Society case study, 118–19, 155–56
 - documentation in, 163–64
 - management and planning of testing in, 157–59
 - process improvement in, 164–65
 - testing phases in, 161–63
 - testing requirements in, 156–57
 - testing roles and responsibilities in, 159–61
- “zero defect software,” 11