Software Testing:
An ISEB Intermediate Certificate

Brian Hambling and Angelina Samaroo

The official textbook for the ISEB Intermediate Certificate in Software Testing covers the testing fundamentals to help newly qualified software testers learn the skills needed to take them to the next level. The book includes worked examples and sample questions from the examination, self-assessment exercises and also provides a structured revision exercise for those who have attended a course.

- Testing fundamentals
- Reviews
- Testing and risk
- Test management
- Test analysis

The only official textbook for the ISEB qualification.

About the authors
Brian Hambling has experienced software development from a developer’s, project manager’s and quality manager’s perspective in a career spanning over 30 years.

Angelina Samaroo CEng is an instructor for the ISEB Foundation and Intermediate Courses in Software Testing.

The authors are both experienced ISEB examination setters and markers.

Building on the success of their best-selling Foundation book, Hambling and Samaroo take software testers to the next level of professional testing expertise with this perfectly targeted text.
Software Testing

An ISEB Intermediate Certificate
The British Computer Society

BCS is the leading professional body for the IT industry. With members in over 100 countries, BCS is the professional and learned Society in the field of computers and information systems.

BCS is responsible for setting standards for the IT profession. It is also leading the change in public perception and appreciation of the economic and social importance of professionally managed IT projects and programmes. In this capacity, the society advises, informs and persuades industry and government on successful IT implementation.

IT is affecting every part of our lives and that is why BCS is determined to promote IT as the profession of the 21st century.

Joining BCS

BCS qualifications, products and services are designed with your career plans in mind. We not only provide essential recognition through professional qualifications but also offer many other useful benefits to our members at every level.

BCS Membership demonstrates your commitment to professional development. It helps to set you apart from other IT practitioners and provides industry recognition of your skills and experience. Employers and customers increasingly require proof of professional qualifications and competence. Professional membership confirms your competence and integrity and sets an independent standard that people can trust. Professional Membership (MBCS) is the pathway to Chartered IT Professional (CITP) Status.

www.bcs.org/membership

Further Information

Further information about BCS can be obtained from: BCS, First Floor, Block D, North Star House, North Star Avenue, Swindon SN2 1FA, UK.

www.bcs.org/contact
Software Testing
An ISEB Intermediate Certificate

Brian Hambling
Angelina Samaroo
# Contents

Figures and tables vii

Authors ix

Acknowledgements xi

Preface xii

Introduction 1
  - ISEB software testing certificates 1
  - The Software Testing Intermediate Certificate 2
  - The Intermediate Certificate syllabus 2
  - The Intermediate Certificate examination 5
  - Relationship of the book to the syllabus 5
  - How to get the best out of this book 6

1 Testing fundamentals 8
  - Background 8
  - Introduction to testing fundamentals 8
  - Relevant Foundation topics 10
  - Application domains 11
  - Software development models, test levels and test types 21
  - Testing, maintenance and change 30
  - The Fundamental Test Process 32
  - Scenario analysis 34
  - Summary 36
  - Example examination questions 36

2 Reviews 44
  - Background 44
  - Introduction to reviews 44
  - Principles of reviews 46
  - Types of review 48
  - Performing successful reviews 54
  - Summary 58
  - Example examination questions 59

3 Testing and risk 66
  - Background 66
  - Introduction to testing and risk 66
  - Risk fundamentals 68
<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Risk management</td>
<td>73</td>
</tr>
<tr>
<td>Risk-based testing</td>
<td>79</td>
</tr>
<tr>
<td>Summary</td>
<td>80</td>
</tr>
<tr>
<td>Example examination questions</td>
<td>81</td>
</tr>
<tr>
<td>4 Test management</td>
<td></td>
</tr>
<tr>
<td>Background</td>
<td>86</td>
</tr>
<tr>
<td>Introduction to test management</td>
<td>86</td>
</tr>
<tr>
<td>Test policy, test strategy, test plans</td>
<td>88</td>
</tr>
<tr>
<td>Entry and exit criteria</td>
<td>95</td>
</tr>
<tr>
<td>Estimating techniques</td>
<td>99</td>
</tr>
<tr>
<td>Test monitoring</td>
<td>103</td>
</tr>
<tr>
<td>Incident management</td>
<td>108</td>
</tr>
<tr>
<td>Summary</td>
<td>111</td>
</tr>
<tr>
<td>Example examination questions</td>
<td>112</td>
</tr>
<tr>
<td>5 Test analysis and design</td>
<td></td>
</tr>
<tr>
<td>Background</td>
<td>119</td>
</tr>
<tr>
<td>Introduction to test analysis and design</td>
<td>119</td>
</tr>
<tr>
<td>Approaches to testing</td>
<td>121</td>
</tr>
<tr>
<td>Fundamentals of test analysis and test design</td>
<td>122</td>
</tr>
<tr>
<td>Test design techniques and test types</td>
<td>127</td>
</tr>
<tr>
<td>Test coverage</td>
<td>135</td>
</tr>
<tr>
<td>Scenario analysis</td>
<td>138</td>
</tr>
<tr>
<td>Summary</td>
<td>140</td>
</tr>
<tr>
<td>Example examination questions</td>
<td>141</td>
</tr>
<tr>
<td>6 The examination</td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>146</td>
</tr>
<tr>
<td>The Intermediate Certificate examination</td>
<td>146</td>
</tr>
<tr>
<td>Scenario analysis</td>
<td>150</td>
</tr>
<tr>
<td>Planning examination revision</td>
<td>151</td>
</tr>
<tr>
<td>Taking the examination</td>
<td>153</td>
</tr>
<tr>
<td>Summary</td>
<td>153</td>
</tr>
<tr>
<td>A Mind maps of the main chapters</td>
<td>155</td>
</tr>
<tr>
<td>B Definition, interpretation and examples of K levels</td>
<td>160</td>
</tr>
<tr>
<td>Index</td>
<td>163</td>
</tr>
</tbody>
</table>
Figures and tables

Figure I   A mind map of the syllabus topics 4
Figure 1.1 Typical mainframe configuration 14
Figure 1.2 The client–server application domain 15
Figure 1.3 Web-based domain 18
Figure 1.4 The V life cycle 22
Figure 1.5 The iterative life cycle 22
Figure 1.6 Testing activities in the V life cycle 24
Figure 1.7 Testing activities in the iterative life cycle 24
Figure 1.8 The Fundamental Test Process 32
Figure 1.9 The Fundamental Test Process in the V life cycle 39
Figure 1.10 The Fundamental Test Process in an iterative life cycle 40
Figure 2.1 Progressive reviews 54
Figure 3.1 The risk management process 74
Figure 3.2 Risk analysis matrix example 77
Figure 3.3 Populated 7 × 7 risk analysis matrix example 78
Figure 4.1 Test management documentation hierarchy 89
Figure 4.2 ABC Ltd quality management system 95
Figure 4.3 Basic work breakdown structure for a testing project 101
Figure 4.4 Incident management process 109
Figure 5.1 The test analysis and design process 123
Figure 5.2 Scenario analysis example 139
Figure A.1 Chapter 1 Testing fundamentals mind map 155
Figure A.2 Chapter 2 Reviews mind map 156
Figure A.3 Chapter 3 Testing and risk mind map 157
Figure A.4 Chapter 4 Test management mind map 158
Figure A.5 Chapter 5 Test analysis and design mind map 159

Table 2.1 Review reporting form 56
Table 3.1 Risk mitigation strategy example 79
Table 5.1 The benefits and pitfalls of test design techniques 132
Brian Hambling has been a software professional for over 30 years. His initial experience in real-time software development in the RAF led to a period in project management for radar simulation development with Marconi Radar Systems. From there he went to Thames Polytechnic (now the University of Greenwich) as Head of the Systems and Software Engineering teaching group for three years before starting a career as an independent consultant. It was at this stage in his career that he became more involved with testing and the broader discipline of quality management. As a Lead TickIT Auditor Brian gained experience of how quality management systems are implemented, and he also became involved in training TickIT auditors, both for their initial qualification and at a more advanced level for their continuing development. In 1997 he accepted the challenge of training testing professionals for deployment as testing consultants for ImagoQA, and as a result of his success in that role, he became ImagoQA’s Technical Director in 1998. He continued in that role until ImagoQA was acquired by Microgen UK plc, and he stayed with Microgen to integrate the testing business into their broader IT services portfolio. Since leaving Microgen at the end of 2004 Brian has concentrated on his work in two related areas: for ISEB he has taken on the role of Chief Examiner for software testing qualifications; for the Open University he has continued in the distance teaching role that has occupied much of his spare time for nearly 20 years. More recently he has become increasingly involved in course development for the postgraduate programmes in Computing and Technology, the promotion of Open University courses overseas through OU Worldwide and managing a consultancy programme with a major multinational company.

Angelina Samaroo started her career in the Defence sector, working on the Tornado ADV, where she was exposed to all aspects of the software development life cycle. In 1995 she was awarded Chartered Engineer status by the Royal Aeronautical Society.

Early in her career she took an interest in developing staff, beginning with training new engineers in the testing team, then moving to managing the training of new engineers across the company, to the standards laid down by the IEE (now the IET).

In 1999 she joined the commercial sector, and once again became involved in staff development. She became an instructor for the ISEB Foundation Course in Software Testing and then managed the business unit dealing with training of both internal consultants and external clients. Her team
Authors

developed the first course to be accredited by ISEB, for the Practitioner Certificate in Software Testing, early in 2002.

Angelina has also instructed delegates in other aspects of testing, such as unit testing, user acceptance testing and managing testing projects, in the UK, Europe, America and Australia.

She currently works for Pinta Education Limited.
Acknowledgements

The authors acknowledge the significant efforts of Matthew Flynn, Suzanne Peart and Rachael Kavanagh of the BCS and ISEB in bringing this book to fruition and making the qualification a worldwide success.
Preface

The book covers the Intermediate certificate syllabus, explaining what the syllabus means and what needs to be known and understood for the examination. Each chapter relates to one section of the syllabus and includes self-diagnostic questions to provide the reader with a quick guide to how much preparation they might need for that section. Each chapter ends with sample questions in the style of the Intermediate examination. Guidance is given on how to prepare for the examination and on examination technique.
Introduction

ISEB SOFTWARE TESTING CERTIFICATES

The Information Systems Examination Board (ISEB) of the British Computer Society awards qualifications in a very broad range of subject areas, of which software testing is one of the most popular. Software testing currently has three levels of certification: Foundation, Intermediate and Practitioner.

The ISEB Foundation Certificate was first offered in 1998 and became an immediate success. It was followed in 2002 by a Software Testing Practitioner Certificate. Demand for the Practitioner Certificate was initially lower than that for the Foundation certificate for a variety of reasons: not every Foundation holder would wish to proceed to the much more advanced Practitioner level qualification, especially if they were not professional testers; many employers, at the time, did not feel the need for more than a very few testers with the more advanced qualification; most significant of all, many of those who wished to proceed to Practitioner found the transition particularly challenging.

The updating of the Foundation syllabus in 2005 inevitably created some inconsistencies between the new Foundation syllabus and the older Practitioner syllabus, and this situation served to emphasise the need for an update to the Practitioner syllabus. As a result, in 2007 ISEB decided to update the Practitioner syllabus, taking the opportunity to make some other changes, of which three were particularly significant:

- The Practitioner syllabus was split into two separate syllabuses to cover each of the two main Software Testing Practitioner disciplines: test management and test analysis.
- The new Practitioner syllabuses were structured around objective measures of required levels of understanding (K levels, which are introduced on page 3 and explained in Appendix B).
- Some changes were made to reflect new topics introduced at Foundation and to bring the Practitioner syllabuses more in line with the state of testing in 2007.

These were necessary changes but they could not solve the problem of the distance between the Foundation and the Practitioner levels of qualification. So one further change was recognised as necessary to make the transition from the Foundation to the Practitioner level more achievable and more attractive. This important change could not be accomplished by Practitioner syllabus updates, so ISEB decided to create a new Intermediate level qualification to bridge the gap between Foundation and Practitioner.
ISEB introduced the Intermediate Certificate in Software Testing in 2007 to coincide with the release of the new Practitioner syllabuses, with the aim of providing the desired ‘bridging’ qualification between Foundation and Practitioner. The Intermediate qualification was designed to give candidates an opportunity to tackle a more advanced examination without having to assimilate a significant amount of new material. The Intermediate Certificate is a prerequisite for those seeking a Practitioner qualification.

In this introductory chapter we explain the content and structure of the Intermediate Certificate syllabus and its associated examination and provide an insight into the way the book is structured to support learning in the various syllabus areas. Finally we offer guidance on the best way to use this book, either as a learning resource or as a revision aid.

THE SOFTWARE TESTING INTERMEDIATE CERTIFICATE

Like the first level Foundation Certificate, the Intermediate Certificate provides broad coverage of the whole discipline of software testing. Unlike the Foundation, the Intermediate Certificate is aimed at those who seek a more practical treatment of the core material.

The Intermediate syllabus contains a number of learning objectives related to the analysis of testing situations, with the intention that examinations will test a candidate’s ability to apply the key ideas in a practical and realistic setting. For this reason, the Intermediate examination is mainly set at the K4 level. A full explanation of K levels is contained in Appendix B, but Foundation Certificate holders will be familiar with levels K1 to K3, all of which were used in the Foundation examination. The K4 level requires candidates to be able to analyse information so that the response made to a situation is specific to that situation. In examination terms, this means that candidates will be asked to analyse scenarios and answer questions directly related to a scenario.

The authors of the syllabus have aimed it at Foundation Certificate holders, which makes the certificate accessible to those who are or who aim to be specialist testers, but also to those who require to take their more general understanding of testing to a higher level; it should therefore be appropriate to project managers, quality managers and software development managers. Although one specific aim of the Intermediate Certificate is to prepare certificate holders for the Practitioner level of certification, the Intermediate Certificate has sufficient breadth and depth of coverage to stand-alone as the next step up from the Foundation certificate.

THE INTERMEDIATE CERTIFICATE SYLLABUS

Syllabus content and structure
The syllabus is broken down into five main sections, each of which has associated with it a minimum contact time that training courses must include within any accredited training course:
Testing fundamentals (3 hours)
Reviews (4 hours)
Testing and risk (3 hours)
Test management (4 hours)
Test analysis (4 hours)

The relative timings are a reliable guide to the amount of time that should be spent studying each section of the syllabus.

Each section of the syllabus also includes a list of learning objectives that provides candidates with a guide to what they should know when they have completed their study of a section and a guide to what can be expected to be asked in an examination. The learning objectives can be used to check that learning or revision is adequate for each topic. In the book, which is structured around the syllabus sections, we have presented the learning objectives for each section at the beginning of the relevant chapter, and the summary at the end of each chapter confirms how those learning objectives have been addressed.

Finally, each topic in the syllabus has associated with it a level of understanding, represented by the legend K1, K2, K3 or K4; these are explained in detail in Appendix B.

- Level of understanding K1 is associated with recall, so that a topic labelled K1 contains information that a candidate should be able to remember but not necessarily use or explain.
- Level of understanding K2 is associated with the ability to explain a topic or to classify information or make comparisons.
- Level of understanding K3 is associated with the ability to apply a topic in a practical setting.
- Level of understanding K4 is associated with the ability to analyse a situation and make reasoned judgements about testing in that specific situation.

The level of understanding influences the level and type of questions that can be expected to be asked about that topic in the examination.

Questions in the Intermediate examination are at the K level associated with the syllabus topic(s) covered by the question.

**Syllabus map**

The syllabus can usefully be viewed as a mind map, as shown in Figure I.
Figure I  A mind map of the syllabus topics
In this representation the main sections of the syllabus, corresponding to chapters in the book, provide the first level of ordering. The next level provides the breakdown into topics within each section. In most cases the syllabus breaks topics down even further, but the level of breakdown is omitted from the diagram for clarity. Figure I enables the entire syllabus to be viewed and is potentially useful as a tracking mechanism to identify visually which parts of the syllabus need most attention and which parts you feel are well understood. By recognising the relative strengths and weaknesses by topic within sections it is easier to understand the nature and extent of the weakness. There is relatively little new theoretical content in the Intermediate syllabus, so the main aspect to be assessed is your confidence in analysing scenarios and applying the ideas from each topic to the scenario.

**THE INTERMEDIATE CERTIFICATE EXAMINATION**

The Intermediate Certificate examination consists of 25 multiple choice questions. All questions are set in the context of scenarios; an examination includes five scenarios, each with five questions.

More detail about the question style and about the examination is given in Chapter 6. Example questions, written to the level and in the formats used in the examination, are included within each chapter to provide plenty of analysis practice.

**RELATIONSHIP OF THE BOOK TO THE SYLLABUS**

This book has been written specifically to help potential candidates for the ISEB Intermediate Certificate in Software Testing to prepare for the examination. Like its predecessor, *Software Testing: An ISEB Foundation* (2007), the book is structured to support learning of the key ideas in the syllabus quickly and efficiently for those who do not plan to attend a course, and to support structured revision for anyone preparing for the exam, whether or not they have attended a training course.

The book is structured into chapters that mirror the sections of the syllabus so that you can work your way through the whole syllabus or select topics that are of particular interest or concern. The structure enables you to go straight to the place you need, with confidence that what you need to know will either be covered there and nowhere else, or that relevant cross references will be provided.

Each chapter of the book incorporates the learning objectives from the syllabus and identifies the required level of understanding for each topic. Each chapter includes self-assessment questions to enable you to assess your current knowledge of a topic before you read the chapter; at the end of each chapter there are examples of scenario-based questions to provide practice in answering typical examination questions. Answers are provided
for all questions, and the rationale for the correct answer is discussed for all practice questions.

Chapter 6 explains the Intermediate Certificate examination strategy and provides guidance on how to prepare for the examination and how to manage the examination experience to maximise your own performance.

HOW TO GET THE BEST OUT OF THIS BOOK

This book is designed for use by different groups of people. If you are using the book as an alternative to attending an accredited course you will probably find the first method of using the book described below to be of greater value. If you are using the book as a revision aid you may find the second approach more appropriate. In either case you would be well advised to acquire a copy of the syllabus (available from www.bcs.org.uk) and a copy of the sample examination paper (available from ISEB) as reference documents, though neither is essential, and the book stands alone as a learning and revision aid.

Using the book as a learning aid

For those using the book as an alternative to attending a course the first step is to familiarise yourself with the syllabus structure and content by skim reading the opening sections of each chapter where the learning objectives are identified for each topic. You may then find it helpful to turn to Chapter 6 and become familiar with the structure of the examination and the types and levels of questions that you can expect in the examination. From here you can then work through each of the five main chapters in any sequence before returning to Chapter 6 to remind yourself of the main elements of the examination.

For each chapter begin by attempting the self-assessment questions at the beginning to get initial confirmation of your level of confidence in the topics covered by that chapter. This may help you to prioritise how you spend your time. Work first through the chapters where your knowledge is weakest, attempting all the exercises and following through all the worked examples. Read carefully through the chapters where your knowledge is less weak but still not good enough to pass the exam. You can be more selective with exercises and examples here, but make sure you attempt the practice questions at the end of the chapters. For the areas where you feel strong you can use the chapter for revision but remember to attempt the practice questions to positively confirm your initial assessment of your level of knowledge. Every chapter contains a summary section that reiterates the learning objectives, so reading the first and last sections of a chapter will help you to understand how your current level of knowledge relates to the level required to pass the examination. The best confirmation of this is to attempt questions at the relevant K level for each topic; these are provided in the book. The ability to analyse situations and answer questions at the K4 level is the single most important part of preparation for the Intermediate examination. There is
no level of knowledge that will substitute for this practical ability to analyse
scenarios and apply the ideas explained in the book.

**Using the book as a revision aid**

If you are using this book for final revision, perhaps after completing an
accredited course, you might like to begin by using a selection of the example
questions at the end of each chapter as an initial assessment of where you
are in relation to what is needed to pass the exam; this will provide you with a
fairly reliable guide to your current state of readiness to take the examination.
You can also discover which areas most need revision from your performance
in this initial analysis, and this will guide you as you plan your revision.

Revise first where you feel weakest. You can use the opening sections of
each chapter, containing the learning objectives and the self-assessment
questions, together with the summary at the end of each chapter to further
refine your awareness of your own weaknesses. From here you can target
your studies very accurately.

You can get final confirmation of your readiness to take the real examina-
tion by taking the sample examination paper provided by ISEB.

Whatever route you take it should lead you to a confident performance in
the examination and we wish you good luck with it.

**Answers**

Answers to chapter Checks of Understanding, Example Examination Ques-
tions, Exercises and Self-assessment Questions can be found at the end of
the relevant chapter.
1 Testing fundamentals

BACKGROUND

The ISEB Foundation Certificate syllabus covered six main areas:
- Fundamentals of testing
- Testing throughout the software life cycle
- Static techniques
- Test design techniques
- Test management
- Tool support for testing

The Intermediate Certificate follows a similar structure, covering five main areas:
- Testing fundamentals
- Reviews
- Testing and risk
- Test management
- Test analysis

The topic headings reflect a slightly different organisation of ideas in theIntermediate Certificate but with broadly similar scope to the Foundation Certificate. At Intermediate level the topics are addressed at a higher cognitive level than at Foundation, for example by considering how test design techniques are selected rather than focusing on using the techniques themselves.

The Intermediate Certificate extends the testing fundamentals in the Foundation syllabus by considering the testing challenges associated with particular types of application, particular software development life cycles and particular development methods. To support this aspect of the syllabus a new topic is introduced. The section on application domains introduces four broad groups of application types and characterises them in terms of their attributes and the testing challenges they pose.

INTRODUCTION TO TESTING FUNDAMENTALS

Learning objectives

The learning objectives for this chapter are listed below. You can confirm that you have achieved these by using the self-assessment questions on page 9, the ‘Check of understanding’ boxes distributed through the text, and the example examination questions at the end of the chapter. The chapter summary will remind you of the key ideas.
We have given a K number to each topic to represent the level of understanding required for that topic; for an explanation of the K numbers see Chapter 6 and Appendix B.

**Review of the Foundation Certificate content**

- Review the main principles and themes from relevant areas of the Foundation syllabus, all of which are considered part of the required knowledge for this syllabus. (K1)

**Application domains**

- Describe the similarities and differences between typical application domains. (K2)
- Identify and explain the testing challenges associated with these application domains. (K2)
- Analyse a situation to determine the testing challenges present in that scenario. (K4)

**Testing in the life cycle**

- Recognise and explain the relationship between testing and development. (K2)
- Identify other processes with which testing interfaces during development. (K1)
- Explain the relationships between debugging, initial testing during development, confirmation testing and regression testing. (K2)
- Explain how testing fits into sequential and iterative life cycle models. (K2)
- Describe the testing challenges associated with each life cycle and explain how these challenges can be met. (K2)
- Analyse a situation to identify the SDLC model(s) in place and select appropriate testing activities to fit with the situation and the life cycle(s) in place. (K4)

**Fundamental Test Process**

- Recall the Fundamental Test Process and explain how it may be deployed in different situations and within different life cycle models. (K2)

**Self-assessment questions**

The following questions have been designed to enable you to check your current level of understanding of the topics in this chapter. Note that they are not representative of the questions you will see in an Intermediate examination; examples of these are given at the end of the chapter. These self-assessment questions are designed to enable you to assess your knowledge of the topics covered in this chapter before you attempt any scenario-based questions.

The answers to the self-assessment questions are provided on page 37.
**Question SA1**
Which of the following is the best description of a client–server application?

A. The application makes use of a web front end.
B. There is at least one computer that provides information to other computers.
C. There is one computer that provides information to dumb terminals.
D. The application(s) run on a single computer.

**Question SA2**
Which of the following best describes a difference between a sequential development model and an iterative one?

A. A sequential development model will always take longer than an iterative model, for a similar sized project.
B. A sequential development model guarantees that the customer will like the product, while the iterative model does not.
C. A sequential development model involves the users in testing mostly at the end of development, while the iterative model concentrates on the user perspective throughout development.
D. A sequential development model requires significant regression testing, while an iterative model does not.

**Question SA3**
Which of the following best describes the purpose of the Fundamental Test Process?

A. To identify the amount of testing that should take place at each test level in both sequential and iterative development models.
B. To identify the activities required to conduct testing regardless of the test level or development life cycle.
C. To identify the risks for a project and the activities required to manage these risks.
D. To identify the skills required in order to conduct testing effectively.

**RELEVANT FOUNDATION TOPICS**
This chapter revisits the following topics from the Foundation syllabus:
- Development life cycles
- Fundamental Test Process
- Test levels
- Test types
- Maintenance testing

We will look at the topics above from a K4 perspective, analysing situations in order to understand the relevant development life cycles and test processes.
in place, with a view to identifying expectations and challenges associated
with their use. We will cover these on pages 21–35.

APPLICATION DOMAINS

The topic of application domains has been included in the Intermediate
syllabus to increase awareness of the testing challenges, not just of the func-
tional and non-functional aspects of an application, but also of the underly-
ing technical architecture on which the application may reside.

What is an application domain?

The term ‘application domain’ has been used in the Intermediate syllabus to
identify a collection of applications with similar characteristics. Four applica-
tion domains have been specified: PC-based; mainframe; client–server; web-
based. Each of these domains implies a set of characteristics that will affect
testing, and the characteristics of each domain contrast with those of the
other domains.

In this chapter we describe the application domains and discuss their
most significant characteristics from a testing point of view. From here we
can identify the kinds of risk associated with each domain. Each domain is
described in terms of its technical characteristics and the testing challenges
that it generates. This is followed by the analysis of scenarios in order to
recognise which application domain or domains are present and to identify
the associated testing challenges.

In case you are concerned at this point, please do not worry. It will not be
necessary for you to have a detailed understanding of the technical aspects of
testing any specific application or application domain for the Intermediate
Certificate, but it is essential to understand the key differences between
domains and how these might affect testing.

We will now give you a high-level description of the technical characterist-
ics of these domains, as required by the syllabus.

The PC-based domain

Technical characteristics of the PC-based domain

PC-based applications are those that are designed to run on a single PC or
Macintosh (or any other stand-alone desktop, laptop or notebook architec-
ture, down to and including all those mobile devices that use a Windows or
similar operating system). Examples are Microsoft® Office, games software,
accounting packages and the like. The key characteristic of the PC-based
domain is that applications run on a relatively small general purpose com-
puter with a relatively simple operating system.

You should be aware that this does not entirely differentiate the PC-based
architectures from those used in other domains, because PC-based hard-
ware has a very wide scope of applications from large servers to stand-alone
machines. Stand-alone office machines, for example, will typically have medium-speed processors but limited networking capability, while other PC architectures may be used as network servers. Software written for stand-alone PCs may have a sophisticated user interface, while specialised software will be needed to enable PCs to act as web servers. At the server end of the PC spectrum applications will overlap with client–server systems and web-based systems. Even mainframe systems may have similar characteristics to a large array of PCs custom designed to solve a specific problem. This need not concern us because we will focus entirely on the nature of the PC-based domain as we have defined it.

As far as the syllabus and this book are concerned the testing of PC-based domains addresses the unique aspects of the single stand-alone PC typical of home users or simple office systems.

Testing challenges of the PC-based domain

The testing challenges associated with the PC-based domain include the fact that applications may need to be tested for:

- compatibility with both PC and Macintosh architectures and with a variety of operating systems;
- the ability to work alongside other applications;
- resource requirements, to ensure that they are not ‘resource hungry’, taking up more system resources than expected or desired.

The limited PC architecture offers relatively little support to testers wanting to run sophisticated tools or create complex test environments, especially if the application under test is using most of the system resources.

As we said earlier, PC-based hardware has a very wide scope of applications from stand-alone machines to front-end clients and back-end servers. We will discuss these application domains next.

EXERCISE 1 – PC-BASED APPLICATION

A software house is creating a new game, called MyEcho. It is designed for children under 16, to teach them about their carbon footprint. It starts off by taking the player through a typical day at school, at the weekend and on holiday. The player interacts by nominating the activities and modes of transport etc. At each step a carbon cost is given. The game is intended to be multi-player, but can be played stand-alone as well.

Which of the following is a challenge of testing that the game works correctly in stand-alone mode?

A. Testing that the game can be played on PCs with specified system requirements.
B. Testing that the carbon footprint calculations are correct.
C. Testing that the game is easy to play.
D. Testing that the game can be linked with other gamers.
The mainframe domain

**Technical characteristics of the mainframe domain**

The mainframe domain was the earliest to emerge and it has evolved over the decades, but many of its key characteristics remain the same. Early examples were typified by:

- being physically large;
- using ‘dumb’ terminals that input and access data, but do not process data;
- having significant processing capability, in terms of volume and speed of processing of data;
- being used as general purpose computing ‘engines’ running a variety of applications.

These characteristics are still often present but modern mainframes tend to be more specialised and have the following characteristics:

- High availability
- High security
- High scalability
- Limited user interface
- Very robust hardware and fault tolerance at hardware and software levels

Mainframe applications are commonly business critical, hence the need for high availability and security. Mainframe systems typically timeshare a central (host) computer, containing most of the intelligence of the system. Mainframe architectures may also utilise multiple virtual machines (i.e. multiple operating systems on a single hardware platform). Mainframe systems were originally designed to run applications in batch form (one at a time) using dumb terminals; interactive terminals and transaction-based applications are now more common. Hardware and software scalability is essential in most cases to enable growth, but there is also a need for backwards compatibility to enable critical legacy applications to be supported.

The mainframe category has become blurred because mainframe machines may be migrated for use as a server in a client–server system and some mainframe machines have been replaced by supercomputers (typically an array of powerful machines). Mainframe architectures are also now often used as web servers or as application servers with a web server to facilitate access via the internet.

**Testing challenges of the mainframe domain**

Mainframe systems may not always support a graphical user interface, so testing may need to be via a command line or other textual interface. This makes tests more labour intensive to set up and run. The large and complex operating systems used can make the test set up complex, and test
environments may also be complex and may require very large databases or co-ordination between users in dispersed locations. There are many testing challenges, including:

- creating the required test environments, which may be large and complex;
- creating and maintaining large regression test suites to support significant maintenance changes over a long time frame may be expensive and time-consuming;
- creating the large-scale test data that may be required;
- virtual machines must be tested for non-interference at the data and processing levels;
- testing for robustness and fault tolerance may require specialist skills;
- scalability testing may require costly extra hardware;
- security testing may require specialist skills and test data.

**EXERCISE 2 – MAINFRAME APPLICATION**

A bank is migrating its data from its current mainframe computers, to a more modern system, allowing improved data and business continuity, as part of its implementation of a disaster recovery programme. The data migration is starting with the retail arm of the bank, and will move to its investment arm, followed by other divisions. The retail banking side deals with individual and business depositors, and loans (including home loans).
Which of the following would be most important when testing the mainframe aspects of this phase of the data migration?

A. Testing that the home loan repayments are correctly calculated.
B. Testing that the integrity of the data on home loans remains intact.
C. Testing that individual customers can continue accessing their accounts online.
D. Testing that investments can continue to be made.

The client–server domain

Client–server domains are made up from a collection of ‘client’ computers that request services from a ‘server’ computer. In this context a ‘client’ is any computer that uses services from a ‘server’ and a ‘server’ computer is a provider of services.

The client–server architecture can be compared with the mainframe architecture, with its single time shared computer. A simple client–server architecture is depicted in Figure 1.2.

![Figure 1.2 The client–server application domain](image)

Technical characteristics of the client–server domain

- Clients and servers are remote from each other and connected by a network.
- Clients may be simple ‘dumb’ terminals, with no functionality of their own and acting solely as communication between client and server.
- Clients may carry out local data processing – they can be characterised as ‘fat’ or ‘thin’, depending on the amount of data processing being carried out – a PC is a typical ‘fat’ client.
- Clients are active components that initiate requests and wait for a response; they may be all the same or there may be differences between clients for particular users. Different clients may also have access to different services.
- Servers are passive and wait for and respond to requests.
• Servers are associated with the resource used to provide the service; this could be a large database, the internet, a group of applications, a collection of files, or any other service.

• A server is designed to manipulate the source effectively and efficiently and to respond to requests, but may not have display or other user interface functionality.

• A server would normally provide services for more than one client, with clients sending requests to the server for specific services.

• Communication between clients and servers is managed by using a network protocol that provides a standard template for all communications.

• Client–server systems are tier structured, with the server as one tier and the clients as a separate tier or tiers. It is feasible to have more than two tiers and three-tier architectures are actually quite common. For example, a three-tier architecture could have client terminals as the first tier, application servers to process data for clients as the second tier, and database servers to manage data for the applications as the third tier. This can be continued more or less indefinitely to \( n \)-tier architectures. One major advantage of increasing the number of tiers in a client–server architecture is that the tiers separate out layers of processing to better balance the load on individual servers, but the price of this is that there is more network traffic and therefore greater load on the network.

• Client–server systems are ‘always on’ with both clients and servers running continuously and the servers awaiting some kind of ‘event’ generated by a client. This is often described as an ‘event-driven’ architecture.

Client–server architectures have some important characteristics that make them very attractive for some kinds of applications, standing between the power and relative inflexibility of the mainframe architecture and the flexible but relatively vulnerable PC architecture.

Key advantages of client–server architectures over PC or mainframe architectures include:

• improved usability – by separating out the user interface into the client computers the usability can be enhanced at a local level;

• greater flexibility – servers or clients can be updated without affecting overall system operation;

• scalability – the architecture allows additional servers or clients to be added relatively easily;

• greater interoperability – the connections between systems can be managed at the server level.

On the negative side, traffic congestion on the network is always a potential problem. The advantage of separating out data and applications to servers
also has the disadvantage that the servers become critical and any breakdown will affect the entire system.

**Testing challenges of the client–server domain**

- Testing a client–server system requires the creation of ‘events’ to exercise the event-driven architecture. This involves setting up or simulating client events, such as a request for data from a database, and then checking that the system responds correctly to each event.
- The requirement that client–server systems run continuously gives rise to new problems such as memory leaks and other dynamic effects that may cause degradation of performance over time.
- Since client–server systems contain multiple computers and networks, testing may become complicated in terms of the number of combinations of system components and interfaces to be tested.
- As for mainframes, security and performance must be given due consideration.

As a result of all these factors tests can become more complicated to set up, execute and check than for simpler architectures. Regression testing can be more difficult to automate because it can be harder to manipulate system behaviour at a local level (e.g. creating a ‘server busy’ condition). New kinds of defects can arise (e.g. communication problems) and performance and scalability become key characteristics that need testing.

One example of more complex testing arises because tests originating at a client (requesting a service) may fail for a large number of reasons. The test will need to ensure that all the possible responses to the request are tested, which will almost certainly mean simulating some or all of them at the client end.

**EXERCISE 3 – CLIENT–SERVER APPLICATION**

A client–server system has a single server providing access to a database of customer records. There are six clients in different departments, each requiring the data to be presented in a different form, so each client manipulates the data for presentation. The system requires access codes at the client sites to prevent unauthorised access to customer records.

Which of the following options best describes a testing challenge for the client–server aspects of the system?

A. Data requests will need to be generated from each client terminal to test the server’s response.
B. Six testers will be required to operate each terminal simultaneously.
C. The integrity of the data stored on the database will need to be tested.
D. System testing will require a fully defined specification to be provided upfront.
The web-based domain

A web-based application is one that provides its services via a website. In some respects a web-based application is a client–server system, with a local terminal and its web browser considered as the (thin) client and the web server as the server. In this case the network is the internet and there can be a very large population of clients.

Figure 1.3 illustrates the web-based domain.

![Diagram of web-based domain](image)

**Figure 1.3 Web-based domain**

Web-server architecture is mainly concerned with protocol and information retrieval interfaces on the host system. The web-server architecture uses a high-level communication layer called HyperText Transfer Protocol (HTTP); HTTP uses a reliable network transport layer called Transmission Control Protocol/Internet Protocol (TCP/IP) as the method of communication over a network.

Web-based applications are identified as a separate domain because the use of the internet as a network poses new challenges, of which the scale of the underlying network and the increased challenges of security are two fairly obvious examples.
Technical characteristics of the web-based domain

- Web applications generate documents in standard formats, such as HTML/XHTML, which are supported by all common browsers, making it very easy to update applications on the server without impact on clients.
- Web pages are generally delivered as static documents but web-form elements embedded in the page mark-up (the HTML that defines how the page is presented) can be used to make the user experience interactive.
- Web servers can be dynamically updated (e.g. when database content changes) so that user access to servers extracts the latest updated information. This is common for applications that incorporate news updates (e.g. www.bbc.co.uk).
- Problems can occur if users customise display settings or use inconsistent implementations of HTML.
- Increasingly, rich functionality at the client end is moving these applications to a ‘thick’ client–server. Most are three-tiered, with the browser as the user tier, dynamic web content technology (such as ASP) as a middle tier, and a database as the third tier.

There are some important consequences of these technical characteristics.

For example:

- Web-based applications are essentially client–server applications running on a very large network, so they share the characteristics of client–server configurations.
- Browser-based applications use a desktop as the client with a web server as server (possibly also with a complete client–server architecture behind it).
- Dynamically generated content of web pages and dynamic database connections mean that each access to a web page may generate different results.
- Java/ActiveX content may require applets to be executed in the client environment.
- Remote application servers may sit between web servers and database servers and these may be complex.
- Some applications will bypass the web server to download directly (e.g. by using DCOM).

Testing challenges of the web-based domain

Web-based applications have all the challenges of client–server systems. In addition, they pose new problems:

- The server(s) are even more remote from the clients, which may cause issues when trying to initiate events at different tiers.
- The clients are remote from each other and may be geographically widespread.
• Client hardware architectures may have a wide variation in capability.
• Client architectures may be using a variety of operating systems and browser versions, which may increase the scope of testing.
• Increased performance testing will be required.
• Increased security testing will be required.
• Increased accessibility testing may be applicable (www.w3.org) – accessibility refers to the ability of people with disabilities to use a service.
• Increased usability testing will be required.

EXERCISE 4 – WEB-BASED APPLICATION

A global investment bank is upgrading its systems to allow its traders to access information on sales and profit performance. They will be able to access their personal performance figures, as well as those for the whole team. The system already allows traders from each country to access their own figures via an intranet. Now the information will be made available via a web interface, and is intended to allow traders to measure their performance on a global scale.

Which of the following best describes a testing challenge of the web aspects of the upgrade?
A. Testing that the traders enter their figures accurately.
B. Testing that the systems calculate individual and team performance accurately.
C. Testing that the figures are available on demand.
D. Testing that bonuses match performance.

CHECK OF UNDERSTANDING CU1

Match the testing challenge on the left to its most appropriate application domain on the right.

<table>
<thead>
<tr>
<th>Testing challenge</th>
<th>Application domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generating network traffic</td>
<td>PC-based</td>
</tr>
<tr>
<td>Testing for different browsers</td>
<td>Mainframe</td>
</tr>
<tr>
<td>Operating system compatibility</td>
<td>Client–server</td>
</tr>
<tr>
<td>Robustness testing</td>
<td>Web-based</td>
</tr>
</tbody>
</table>

Analysing application domains

Of course real application domains are seldom, if ever, limited to one of the models we have just described. Two or more of the domain models may be
in use in a real situation, and the analysis of the scenario will need to identify the dominant features so that testing decisions are targeted at those features.

**A TYPICAL SCENARIO**

Let us return to our game *MyEcho* that you first met in Exercise 1 on page 12.

There we saw that the game is multi-player. It can be played with other users, usually these days via a web interface. This means that we must test not just that the game can be successfully played on individual machines, but also that it can be linked with other gamers. This may require, for instance, the creation of avatars to represent the gamer online. Thus, there are two application domains present: stand-alone and web-based. Testing the avatar functionality must also include checking the ability to not just create the avatar on the user machine (a stand-alone aspect), but also to upload it for use online (a web-based aspect).

Later we will extend the analysis to include software development life-cycle models and development approaches, which are the other main factors to be considered in selecting a testing approach in a given scenario.

**SOFTWARE DEVELOPMENT MODELS, TEST LEVELS AND TEST TYPES**

**Software development life cycles**

The Foundation syllabus included an outline of the two generic life cycles: sequential, using the V life cycle as an example, and iterative. The V life cycle, shown in Figure 1.4 in one of its many possible manifestations, is often used as an ideal model because of the very structured way that development and testing are related.

This is an idealised model in many respects and is seldom replicated in practice. It provides a clear and simple way to describe the relationship between testing and development, by showing that each development activity has an associated testing activity, and by demonstrating that tests can and should be specified at the same time as the associated development deliverable is created. Why? Because the activity of specifying tests is an excellent way of testing the quality of the development deliverable. Any problems with creating tests signal similar problems with the next step in development and these problems are better sorted out early. A full implementation of this model would have reviews at every stage and test specifications created before module coding begins.

In practice this model has some fundamental problems, most important of which is the fact that the whole model depends on the quality and completeness of the requirements, since both the testing and development activities stem from this single deliverable. Requirements are notoriously difficult to
define and usually do not remain stable throughout a software project, so this model is usually compromised in practice.

The iterative model (Figure 1.5) takes the idea that requirements are seldom stable to its logical conclusion and removes the dependence on stable requirements by not attempting to define them fully at the beginning of the project. Instead it identifies a set of objectives for the project and prioritises these. The development is then split into a series of short timeboxes, during which a subset of the requirements is developed and tested, so that the project progresses as a series of short development cycles, each focused on the highest priority requirements not yet developed.
This model also has its drawbacks. The very light verification process has the consequences that specifications and systematic testing do not often feature very highly. Validation of functionality, often by customers, is carried out at each iteration. Typically, the lack of detailed requirements limits the use of systematic test design techniques (in particular those requiring detailed specifications, such as equivalence partitioning). In addition, the focus on subsets of requirements can impair the overall design.

Examples of iterative development models are: Rapid Application Development (RAD), Prototyping, Agile, Dynamic Systems Development Methodology (DSDM).

This brief overview of testing in alternative life cycles shows that each model has its benefits and drawbacks. Practical testing, like practical development, involves compromise and choice.

The focus of the Intermediate Certificate is to consider how testing processes can be integrated most effectively with development processes to minimise the inherent problems and maximise the opportunities to identify and remove defects.

**Test levels and test types**

Test level is simply a way of referring to any group of test activities that are organised and managed together.

In the V life cycle, test level relates to the testing associated with each development work product, such as component testing or integration testing. In this case the idea of level is naturally associated with the point in the life cycle where testing occurs.

In an iterative life cycle test levels would occur at the end of each timebox, where all the functionality developed in that timebox is tested together and with functionality developed in earlier timeboxes.

Test type refers to test activities directed towards achieving a particular test objective, so functional testing and performance testing are both examples of test types. Any test type can occur at any test level and may occur at more than one test level. Functional testing is likely to occur at every test level, whereas performance testing is more likely to occur at some test levels than at others e.g. more at system test level than at component test level.

**Testing activities in the life cycle**

The relationship between testing, development and other disciplines within the overall development life cycle is a complex one. In this section we will consider the relationship between testing and some of the many alternative ways of developing software to provide a framework for assessing how this relationship can be optimised. We will also consider how software testing relates to the other key disciplines that interface with development.

First of all, we return to the familiar V life cycle, where the relationship between testing and development is simple and consistent. For every development work product there are two corresponding test activities: one to
determine the quality of the work product itself, including its testability, and one to specify the testing to be carried out on the software defined by the work product. When the software is constructed a third test activity executes the tests. This is shown for a single work product in Figure 1.6.

![Figure 1.6 Testing activities in the V life cycle](image)

Each level in the V life cycle will have a similar relationship between work product, software artefact and test activities, but the details of the test specifications will be different for each level.

In an iterative life cycle with time boxed development the picture is rather different. Each timebox represents a complete ‘life cycle’ from requirements through to software product but for a limited set of requirements. The relationship between development and testing is shown in Figure 1.7.

![Figure 1.7 Testing activities in the iterative life cycle](image)
Figure 1.7 represents a single timebox; a complete project would consist of a sequence of timeboxes, each with a new set of requirements. Note that testing is continuous and related to validating timebox requirements.

**LIFE CYCLE CHALLENGES**

**V model**
- Defining requirements fully upfront.
- Managing change requests throughout the life cycle.
- Getting developers and testers interested and involved in reviews early.
- Balancing cost of writing tests against a test basis that may be changed downstream.
- Gaining acceptance of the system by the end users.

**Iterative model**
- Planning without fully defined requirements.
- Designing tests without complete specifications.
- Establishing a baseline for regression testing.
- Sourcing staff with strong technical and business domain skills.
- Keeping items under configuration management.

*Life cycle related testing challenges*

The testing challenges generated by alternative life cycles can now be identified as a set of inherent risks that will need to be accounted for in any test plan.

In a sequential life cycle a significant challenge is the management of change. The dependence on complete and correct requirements is a problem that, in practice, leads to change throughout the life cycle as new requirements emerge. This means that test plans must try to estimate and allow for this unknown activity and its impact on testing. Another difficult area is the unknown quality of the work products because lower quality will generate a greater burden of confirmation and regression testing as well as causing delays while defects are corrected. The issue of quality is a serious one. A project team’s familiarity with a set of requirements or the best way to create an application to meet those requirements will be key factors in determining the quality of work products; an inexperienced team or one working in a new application environment can be expected to generate lower quality in the sense that work products will contain more defects.

In an iterative life cycle the challenges are mostly associated with the timeboxed approach, both because of the limitations that are placed on the time available for testing and because the product is often built with minimal documentation (usually outline requirements). The idea of building test specifications in advance from detailed work product specifications is often not tenable. One response to this problem has been an approach known as
test-driven development. This is a method of software design in which the test procedures for the iteration are written before the code is created. This has the advantage at this stage of enabling better software design by understanding what it should do from a user perspective. The code is then created and tested against the test procedures already written, and released once the tests have been passed. This is one innovative way in which the inability to construct detailed test scripts from specifications has been tackled.

Whatever the nature of the life cycle, there will be some testing challenges that generate inherent risk for the test strategy to address. In practice the ‘pure’ V life cycle is as uncommon as the ‘pure’ iterative life cycle. Both are idealised models, which can be tailored to meet the challenges of particular projects.

**CHECK OF UNDERSTANDING CU2**

1. Which development model relies on requirements being captured as fully as possible at the very start of the project?
2. Which development model relies on user validation throughout?
3. Give two benefits of the V model.
4. Give two benefits of the iterative model.
5. Give two testing challenges of the V model.
6. Give two testing challenges of the iterative model.

**Testing and development**

So far we have considered ‘vanilla-flavoured’ development, or development in the absence of any particular development environment. The V life cycle, for example, assumes that requirements are functionally decomposed and modules of software constructed to meet each decomposed function in some kind of top down approach. Is software actually constructed that way? Not very often.

In practice, software is constructed using technology to support and, wherever possible, automate the development processes in a development environment. Development methodologies are many and varied but they each bring their own advantages and disadvantages from a testing perspective. We will here consider three alternatives that broadly represent the main ways in which development is made more productive and less risky; these examples can then serve as a template for assessing the impact of development methodologies and technology on the testing activities.

Our examples will be:

- object-oriented development;
- development using reused software components;
- construction of systems from commercial off-the-shelf (COTS) software.
Object-oriented software development

Object-oriented software development was created as a means to maintaining software quality, as systems increased in complexity. A key feature in object-oriented development is the organisation of the code into discrete classes that contain both data structures and the procedures that operate on them, as an alternative to defining procedures and data structures separately. The classes then have limitations on how they can be used and manipulated, leading to the core concept of information hiding.

Information hiding means that data in a system can only be accessed via specific access methods provided by the system. In other words the tester cannot arbitrarily amend data fields to set up tests.

Testing of object-oriented systems therefore entails harnessing the object-oriented development system to testing by constructing ‘test objects’ that utilise the access methods of objects in the system and encapsulate the test scripts within the test object’s own methods.

This is one relatively simple example of the impact of development methodologies, especially those supported by technology, which may demand from the tester a more detailed understanding of the underlying development methodology and technology.

Software reuse

Software reuse involves the storing and reuse of software components that perform tasks likely to be required in other projects. When a requirement for the task arises the existing component is reused in preference to constructing a new component. There are a number of advantages to this, perhaps the most important being reduced effort and the reduced risk in using a ‘tried and tested’ component. From a testing perspective, however, this approach creates new challenges.

- A reused component will have been tested in its original environment, which is unlikely to be identical to the environment into which it will now be introduced.
- It is also possible that the reused component may need to be modified to fit the exact requirements of the new task. Thus the modifications will also need to be tested and monitored to ensure that they do not equate to a new piece of code, requiring more extensive testing than at first thought.

ARIA 5 – A CASE OF POOR TESTING?

The destruction of Ariane 5 after about 40 seconds of its maiden flight on 4 June 1996 is well documented. The cause was a failure of the flight control system, leading to swivelling of the solid booster and engine nozzles, which in turn caused the vehicle to veer off its intended flight path and trigger self-destruction.
The flight control system failure was caused by a software defect but investigation showed that no software failed to perform its intended function. A module reused from the earlier Ariane 4 vehicle generated a software exception that led to the failure of both inertial reference systems (SRIs) on board and transmission to the on-board computer of diagnostic data that, when interpreted as flight data by the on-board computer, caused the demand for nozzle movements that led to the destruction of the vehicle. The reused module functioned correctly throughout but had never been exposed to the countdown and flight-time sequence and the trajectory of Ariane 5; that of Ariane 4 was different.

The testing of Ariane 5 was conducted at four levels, with each level checking what could not be checked at the previous levels. This standard approach was conducted rigorously but did not include a test that the SRI would behave correctly when being subjected to the countdown and flight-time sequence and the trajectory of Ariane 5. The test was not performed because the specification for the SRI did not contain the Ariane 5 trajectory data as a functional requirement.

This is a greatly simplified version of the official report’s findings but it demonstrates that even reused software that functions correctly can be the cause of a serious failure that rigorous testing based on the requirements may still fail to predict. In this case the best tool for discovering the predictable but not obvious limitations of the reused software would have been reviews; reviews were conducted but did not discover the limitations.

Bad testing? That would be a harsh judgement, but the failure could have been averted by testing if the right tests had been designed.

Reuse of software generates new and different testing challenges that require the tester to consider carefully what risks may arise from the proposed reuse and what potential problems need to be considered in designing tests.

Implementing COTS

The main advantage of COTS is that it provides a complete solution to a business problem, usually built by domain experts and used by many other organisations. The confidence this gives in the effectiveness and reliability of the application is potentially very valuable.

In reality, however, COTS may provide a complete or a partial solution to a particular set of requirements. In most cases the fit between a COTS solution and the user community’s requirements is not exact and some modification is required. Organisations may be able to modify their business processes to fit the application as built, but this will necessitate careful testing of the application in its new process environment. Commonly, the COTS software is itself modified to meet the needs of the organisation. In this case, the functionality of the modified software will need to be tested as well as its fit within the new process environment.
The main challenge here is that, even though a COTS application may have been thoroughly tested in a number of different implementations, it is unlikely that any of these will be identical to that now envisaged. Any mismatch between the new environment and those encountered before can throw up anomalous behaviour and identifying where in the application this might occur is not an easy task. A carefully planned and designed test of the application in its new environment is therefore needed. This is an exercise that has some similarities with defining a regression test suite.

In testing COTS-based systems the following are usually required by the customer:

- Acceptance testing of the package in its new environment, if unmodified.
- Where modified by the supplier, acceptance testing of original functionality and integration testing of modifications with original functionality.
- Where modified by the buyer, acceptance testing of original functionality, full testing of modifications, and integration testing of modifications with original functionality.

Of course, it is expected that the supplier will conduct full testing of all software supplied.

CHECK OF UNDERSTANDING CU3

1. In what type of development could information hiding pose a challenge in testing?
2. Identify a potential benefit of reusing software.
3. Identify a potential drawback of reusing software.
4. Identify two challenges of COTS-based development.

Testing and other project processes

Software development projects involve a myriad of overlapping and interconnecting processes. Overall there will be a project management process, the purpose of which is to ensure that the project is completed successfully and within the constraints of the project. Project management will determine how much resource is dedicated to testing and when (even if) testing will happen. Within project management the risk management discipline will identify and quantify the risks that will be a major driver of the testing effort. The relationship between testing and project management is such that project management enables testing to take place, while testing provides information about quality and remaining risk enabling the project manager to make informed decisions on software release.

Managing change involves various processes, such as change management, configuration management and requirements management. These all need to interface with testing so that changes can be incorporated into
the testing plan. Any change will require an assessment of impact that may lead to variations in the test plan to test new requirements, test corrections after defect corrections or regression test after changes have been incorporated. Changes to hardware or business processes may also invalidate testing already completed, so all processes that involve change should be linked with testing so that any change can be reviewed.

Other areas that support the project, such as technical support and technical writing, will also need an interface with testing. Manuals will need to be included in testing at some stage in the project to ensure that testing correctly reflects the user procedures described in them. Technical support may be involved in building or supporting test environments and software builds for testing.

In an iterative life cycle the processes may not be so distinct. The team members may take on different roles at different times, blurring the distinctions between roles such as project manager, configuration manager and tester.

CHECK OF UNDERSTANDING CU4
List three other project processes that interact with the testing process.

TESTING, MAINTENANCE AND CHANGE

Testing and change

Testing, if it is effective, will find defects and thus generate change to the software and the requirements.

In a sequential life cycle, in which tight control is maintained over work products at every stage, we will need to recognise four distinct activities:

- Debugging, though not strictly testing at all, will be the first level of defect removal. Debugging will normally be done by the development team and typically will not be documented; neither will the debugged code be under project version control at this stage (though the development team should be providing local version control). Debugging is preparation for testing, seeking to bring software to a state where formal testing can begin.

- Initial testing is the execution of the tests specified in the appropriate test specification and is part of every test level. The initial tests will typically uncover a number of defects.

- Confirmation testing is the testing carried out after defects have been corrected to ensure that the corrected module now passes the initial test(s) that it failed at the first pass.

- Regression testing is the testing carried out to check that changes such as defect corrections have not made any unexpected and unwanted changes to any other area of the system.
Any change to a system will generate new or modified initial tests and associated confirmation and regression tests, which is one good reason for minimising change to requirements.

In an iterative life cycle the control of individual changes may be less formal within a timebox, allowing multiple revisions to code and test throughout the timebox. At the end of the timebox, however, the configuration of the software and the tests carried out on the software will need to be brought under control. Once a timebox has completed a controlled version of the product will be created. Future timeboxes, then, will generate regression tests to ensure that earlier tested versions of the product have not been compromised, and may require confirmation testing if earlier tested versions of the product are subsequently modified in later timeboxes.

CHECK OF UNDERSTANDING CU5
What is the difference between debugging and confirmation testing?

Testing and software maintenance

Maintenance is defined as everything that happens to a software product after its initial development, so most of a software product’s life is in maintenance of one kind or another. Maintenance can vary from simple defect correction arising from user routine reporting, through to emergency defect correction when something goes seriously wrong in a critical area, to major enhancements of core functionality. If a maintenance activity is large and costly enough it will most probably be managed as a new project, but usually maintenance work will involve making relatively small changes to the system or product.

The problem with small changes is that they do not necessarily imply small impact on the software, so the first task is to try to discover exactly what the likely impact is, by studying the documentation if any has survived or by examining the code. Once the impact is understood a test plan can be constructed to address the risks associated with making the required change(s). This will try to restrict testing to areas where some impact is expected or at least considered possible, but in many cases this will not be feasible and the whole system may need to be tested. Since maintenance involves change, the testing regime will need to include testing of any new functionality and regression testing of potentially impacted areas. The regression tests may need to incorporate non-functional as well as functional testing. A risk-based strategy is usually the best approach in this situation and the risks will be determined by an impact analysis.
SOFTWARE TESTING

THE FUNDAMENTAL TEST PROCESS

The most important tool the tester has for coping with the many variations and challenges of life cycles, development environments and the like is the Fundamental Test Process that was introduced in the Foundation syllabus.

Test processes

The Fundamental Test Process is a set of steps showing the core activities of testing in any context; it is a reminder that we omit any of the five stages shown in Figure 1.8 at our peril.

Figure 1.8  The Fundamental Test Process

The Fundamental Test Process is clearly linear, though with some important feedback loops.
The process begins with planning; this includes setting up the monitoring and control mechanisms that will be needed to ensure the process is effective. Most importantly, test planning includes the setting of test objectives and the setting of the criteria by which we will judge testing to be complete. The completion criteria reflect the level of acceptable risk while the feedback loops are included to ensure that testing is always directed at the test objectives and judged by progress towards the completion criteria. Setting the criteria at the beginning of testing ensures that we are not intimidated into inappropriate criteria when pressure may be mounting at the end of the process.

The process continues by identifying the key stages of defining test conditions and test cases, designing test scripts, executing them and determining whether the test has passed or failed, after which the status of testing against the completion criteria is assessed and decisions taken about the next step. The feedback loops enable the scope of testing to be adjusted according to the evidence provided by test results and any residual risks.

The Fundamental Test Process should be applied at every level of testing, so we would expect to see evidence of every stage in this process in a project test plan but also in each level test plan, though some activities will be more prominent than others at different levels (thus we would not expect to see a full test plan at unit testing).

In the following sections we consider how the Fundamental Test Process relates to the life cycles and development methodologies we have considered in earlier sections.

The Fundamental Test Process and the V model

Implementation of the Fundamental Test Process in the V life cycle is the most straightforward; a single process extends from test planning at the requirements stage to evaluation of exit criteria at acceptance and this embodies all the test phases, so the test planning step is the project test planning step. The Fundamental Test Process also applies to every test level, with test planning for that level initiated at the appropriate level on the left-hand side of the V and evaluation of exit criteria happening at the appropriate level on the right-hand side of the V. In this way each test level is planned, designed, executed and evaluated in its own right, and achievement of its exit criteria is a prerequisite for entering the next test level. The complete set of level entry and exit criteria will be documented in the project test plan. Failure to meet exit criteria requires a decision at the appropriate test level and also at the overall project level, so that risks are managed at the test level but any risk remaining from a test level is taken into account at the project level.

The Fundamental Test Process and the iterative model

The iterative model poses a different process challenge. In this life cycle there are no hierarchical levels but a sequence of timeboxed developments. Here the Fundamental Test Process applies to the project as a whole and
will include the overall structure of timeboxes, with their generic entry and exit criteria. Phases such as test analysis and design, and implementation and execution, may be combined to reflect the testing approach adopted, but the evaluation of testing against criteria to determine when testing is complete will still be a key feature at the end of the project. In this case the Fundamental Test Process will be applied in exactly the same form to each timebox, but the final details of entry and exit conditions may be deferred until just before the timebox begins, when the objectives of the timebox are known.

The Fundamental Test Process is a very important part of preventing the testing in an iterative life cycle degenerating into a secondary or overhead activity under time pressures, and it is in this kind of situation that a well-defined process adds most value.

The Fundamental Test Process and development

The variations in approach brought about by the use of particular development methods may have a significant impact on individual test activities or the scope and content of test levels, but we would not expect the Fundamental Test Process to be overridden or significantly modified. For example, the incorporation of reused components may make the component testing level less significant but this may be balanced by a more extensive integration testing level. Both levels would still be planned and implemented using the Fundamental Test Process as a template.

In the case of COTS the variation may be greater. If COTS is being used to provide a complete application, testing may be effectively reduced to two levels: a brief system test to confirm functionality, and some kind of acceptance test that determines the correctness and effectiveness of the COTS application with business processes, especially if the latter have been modified. The acceptance activity may be extensive and may be expected to find defects, so the careful planning and management of the test process, at both levels and as a project, will be of major importance to the success of the COTS implementation. The main point here, though, is that the testing should still use the Fundamental Test Process as a template for the activity.

CHECK OF UNDERSTANDING CU6

1. Draw a representation of the V model, showing where the Fundamental Test Process fits in.
2. Repeat the above for an iterative development model.

SCENARIO ANALYSIS

The first task of the specialist tester is to analyse not only the application domain, the specific application under test and the product risks but also
the life cycle, the development environment, and any other factor that will impact on the nature and extent of testing required to reduce the risk to an acceptable level. Each of these factors needs to be taken into account. You also need to consider the overall scenario by assigning weights to the various factors so that you can come to a balanced view of the scenario as a whole. From this balanced view you can identify the most significant factors for selecting and deploying the optimum testing approach.

In exam questions the scenarios will clearly signal the various factors but you will need to draw from them your own balanced view. The questions will provide choices for the way to handle the testing challenges with some explanation of the reasoning so that you can select the right response with the right reason and reject incorrect responses or those for which the reason given is not valid.

Exercise 5 should help to clarify how scenario analysis works.

**EXERCISE 5 – A TYPICAL SCENARIO**

An online book selling company is enhancing its offerings to include sourcing out-of-print books, for specialist interest groups (SIGs). It is already providing access to books on aviation, and is about to start on bird watching, with other topics to be added later on demand. The system includes an invitation to current SIG members to register their interest. On registration, customers are sent an instruction sheet on how to use the system. Development is carried out using Agile methods, in line with company standards.

The development team intends to adapt the software used for aviation books for use in sourcing bird watching books. In addition, the site will now include links to countrywide SIGs.

Which two of the following are challenges most likely to arise specifically out of the life cycle in use?

i. Testers may not be able to review the specification for the bird watching book offerings fully at the start of development.

ii. Testers may not fully understand how the enthusiasts are likely to use the site.

iii. Testers may not become involved in the creation of the instruction sheet on how to use the system.

iv. A defect found and fixed at iteration $n$ may become a defect again at iteration $n + 1$ because of changing requirements.

A. i, ii
B. i, iii
C. iii, iv
D. ii, iv
SUMMARY

In this chapter we have reviewed the relevant testing fundamentals from the Foundation syllabus and extended them to provide a greater awareness of the challenges in testing particular software development life cycles and development approaches. We have also introduced the main application domains and examined the challenges in testing these. We have reviewed the Fundamental Test Process and considered how it applies to the two main life-cycle models that we have examined: the sequential V life cycle and the iterative life cycle.

The chapter has also introduced the kind of scenarios and scenario-based questions from which the Intermediate Certificate examination is constructed and provided some initial practice in analysing scenarios and tackling scenario-based questions.

A mind map of the content of the chapter is in Appendix A.

EXAMPLE EXAMINATION QUESTIONS

A website building company (WBC) specialises in building websites for clients with existing database-driven systems. Their clients include those in the real estate, recruitment and mailing list businesses.

The company has been asked to build a new website for a major recruitment agency (RA) that has offices around the world. The agency has a database of over 5,000 firms, and holds CVs on over 100,000 candidates. Access to the current site is by permission level, with firms having access to one part, candidates another, and in-house staff another.

WBC uses object-oriented methods and the Agile model for development, and prides itself in the quality of its products. It uses its standard COTS product, and customises it for each client. However, it has been taken over recently, and many of the development staff have left. It is currently undertaking its own recruitment drive to find new staff.

RA is concerned that its corporate branding is retained, and that its staff and customers instantly identify with the new website, while enjoying a fresh online experience. However, it is keen to get the new site up and running as quickly as possible, and expects to double its firm and candidate list in the next year.

WBC will build the system in two iterations: front-end development followed by integration with the back-end.

E1

Which of the following are testing challenges associated with the integration of the new website with the back-end database?

i. Testing that the new website reflects RA’s branding.

ii. Testing that the new website retains the current access levels.
iii. Testing that the integrity of the data on firms and candidates is not compromised.

iv. Monitoring the number of hits to the new website.

A. i and ii
B. i and iii
C. ii and iii
D. ii and iv

**E2**
Which of the following is a testing challenge associated with the development methodology?

A. Performance testing to cope with the doubling of clients.
B. Information hiding, necessitating the creation of test objects.
C. Navigation testing of the new website.
D. Training new staff in reviewing specifications quickly.

**E3**
Which of the following steps in the Fundamental Test Process is most likely to be reduced in this scenario?

A. Planning of how long testing of the iteration for the back-end integration will take.
B. Design of tests required for the back-end integration.
C. Running tests on the changes to the back-end interface after integration.
D. Evaluation of suitability of the website to RA’s needs before moving on to the back-end integration.

**E4**
Which of the following levels of testing would RA be recommended to perform on WBC’s core product prior to the customisations?

A. Full testing on the core product.
B. Acceptance testing on the core product.
C. Regression testing after customisation.
D. Unit testing on the customisations.

**ANSWERS TO SELF-ASSESSMENT QUESTIONS (ON PAGE 10)**

SA1. B
SA2. C
SA3. B

**ANSWERS TO CHECKS OF UNDERSTANDING**

**CU1 (on page 20)**
In the table below we have reordered the entries in the right-hand column to match the most appropriate application domain to the testing challenge.
Testing challenge | Application domain
--- | ---
Generating network traffic | Client–server
Testing for different browsers | Web-based
Operating system compatibility | PC-based
Robustness testing | Mainframe

Note that in testing, we would aim to test all aspects of a system relating to each application domain present. Here we have looked at what our focus would be when testing against each application domain.

CU2 (on page 26)

1. The V model requires requirements to be fully captured at the start of a project.
2. The iterative model (e.g. Agile development) relies on continual user validation of product functionality during the project.
3. Two benefits of the V model: tests are created as soon as specifications are written, thus defects in the specifications can be identified before coding begins; model shows explicitly the levels of testing to be undertaken at each stage of development.
4. Two benefits of the iterative model: provision is made for requirements to be changed by the customer, thus reducing the risk that the end product may not meet their needs; the customer has early visibility of the working product, allowing them to highlight deficiencies as the product is being built, again reducing the risk of rejection at release.
5. Two testing challenges of the V model: tests created at the start of development may need to be changed downstream; validation at the end of development may lead to customers not fully accepting the system.
6. Two testing challenges of the iterative model: the frequency of requirements changes may lead to an onerous amount of regression testing; lack of detailed specifications may lead to poor test design, thus key defects may be missed.

Here we have listed some key benefits and challenges associated with linear and iterative software development models. Hopefully you have added some more of your own.

CU3 (on page 29)

1. Object-oriented software development makes use of information hiding, which may create a testing challenge.
2. A potential benefit of reusing software is that less effort may be required for testing.
3. A potential drawback of reusing software is that the reused component may need to be modified in order to integrate it into a new environment. This could cause an increased need for testing, which may not be obvious, leading to defects being missed.

4. A challenge of COTS-based development is similar to that of reusing software, it may not work as expected in its new environment. Another is identifying whether defects identified lie within the customised software or in the original functionality.

CU4 (on page 30)
Project processes interacting with the testing process include:
- configuration management
- technical authoring
- technical support
- risk management
- project management.
(Note that we have given more processes than the question has asked for.)

CU5 (on page 31)
The difference between debugging and confirmation testing is that debugging is the process of removing a defect, while confirmation testing is the process of confirming that the fix has been applied correctly.

CU6 (on page 34)
1.
Here we see that we should undertake all steps of the Fundamental Test Process, from planning and test design through to test closure at every level of testing in the V model. Note that we are not suggesting that the full process must be followed at the lower levels of testing. However, there must be some element of planning, test design etc.

2.

Figure 1.10  The Fundamental Test Process in an iterative life cycle

Here we see that the Fundamental Test Process should apply throughout each iteration. Note that the absence of fully defined specifications may make test design difficult, but the method used should be documented.

ANSWERS TO EXERCISES

Exercise 1 (on page 12)

The correct answer is A.

Answer B is incorrect because the calculation of the carbon footprint is associated with testing the application itself, not the underlying technical architecture on which the application resides.

Answer C is incorrect because testing whether the game is easy or not, once again depends on the requirements of the application itself, not the application domain.

Answer D is incorrect because linking the game to other players would not be associated with the game being used in stand-alone mode.
Exercise 2 (on page 14)
The correct answer is B.

Answer A is a tempting option, since it is possible that the calculation algorithms may become corrupted on the migration. However, it is not as good as B, because it is considering just one aspect of data integrity, while answer B considers all of the data.

Answer C is relevant in testing, but is not part of mainframe data testing, and thus is incorrect.

Answer D should be a clear distracter. The first phase will deal with the retail arm of the business, and the scenario gives no information on how investments are made. Thus this is incorrect.

Exercise 3 (on page 17)
The correct answer is A.

Answer B is incorrect because the number six refers to the number of client machines, not the number of testers. What will be of interest is the number of requests that can be handled from the servers simultaneously. Typically, a performance test tool will be used to generate the requests, not testers.

Answer C is incorrect because this is a challenge of testing the database itself, not the client–server part of the system.

Answer D is incorrect because this is a challenge associated with testing any system, not specifically any particular part of it (and sadly, we’re aware that we often don’t have these, whatever we’re testing!).

Exercise 4 (on page 20)
The correct answer is C.

Answer A is incorrect because the system as described makes no mention of checking accuracy of data entered.

Answer B is incorrect because testing of the calculations would form part of the testing of the functionality of the application itself, not its use online.

Answer D is incorrect because the scenario gives no indication that bonuses are calculated in the system. In addition, this calculation, if it existed, would form part of the application testing, as for answer B.

Exercise 5 (on page 35)
The correct answer is D.

The scenario presents a number of factors:

- The application is online (so web-based).
- It will use Agile (iterative) methods.
- It involves adapting an existing application and extending it to include new functionality (links to SIGs).

The question, however, focuses on the life cycle model in use. In other words, it is asking for you to identify two challenges most likely when using an iterative model. Let us look at each possible option in turn.
i. This suggests that specifications will be available and that testers will need to be able to review them fully. This is more applicable to the use of the V model, but for now let’s leave it in as an option, since it is not impossible in an iterative model.

ii. This is looking at the end-user perspective and is applicable in iterative developments, so this can be a possible answer.

iii. This may be useful, but would be a challenge in any life-cycle model, therefore we can rule it out. This is because the question asks for challenges specific to the iterative model.

iv. This is very likely in iterative development, where requirements are expected to change significantly.

We have now ruled out option iii as a possible answer.

We need to rule out another. Option i is quite unlikely when compared with options ii and iv. Therefore, we can rule out option i, leaving us with options ii and iv.

**ANSWERS TO EXAMPLE EXAMINATION QUESTIONS**

**E1 (on page 36)**

The correct answer is C: activities ii and iii.

i. Testing that the new website reflects RA’s branding is part of the functional testing of the website, it is not associated with integration.

Monitoring the number of hits to the new website is also not part of integration testing of the front and back ends.

**E2 (on page 37)**

The correct answer is B.

Answer A is about performance testing, which is independent of the development methodology.

Answer C is about navigation testing. As for performance testing, this is independent of the development methodology.

Answer D should be a clearly incorrect answer. This is outside the scope of test execution.

**E3 (on page 37)**

The correct answer is B.

Answer A, test planning, should not be reduced.

Answer B, test design, relies on detailed specifications. These are unlikely to be available in an Agile environment.

Answer C, test execution, is a given and should not be reduced.

Answer D, evaluation of the functionality against the user needs is a key consideration in testing in an Agile environment.
E4 (on page 37)

The correct answer is B.

Answer A, full testing of the core product should be carried out by WBC, not RA.

Answer C refers to regression testing after customisation. The question asks for testing required before customisation.

Answer D refers to unit testing on the customisation. This would be carried out by WBC, not RA. In addition, as for answer C, the question relates to the core product before customisations.
Figure A.1  Chapter 1 Testing fundamentals mind map
The review process
Objectives and roles
Inputs and outcomes
Principles of reviews
Preparing for reviews
Setting up a review
Conducting a review
Following up a review
Assessing the effectiveness of a review
Using the assessment to improve review performance

Figure A.2 Chapter 2 Reviews mind map
Figure A.3  Chapter 3 Testing and risk mind map
Figure A.4 Chapter 4 Test management mind map
Chapter 5 Test analysis and design mind map

**Approaches to testing**
- Static testing
- Dynamic testing

**Scenario analysis**

**Test coverage**
- Principles of test coverage
  - Test coverage measures
  - Coverage measures and risk
- Benefits and pitfalls of coverage measures
  - Practical coverage measurement

**Fundamentals of test analysis and test design**

**Test design techniques and test types**
- Principles of test design techniques
  - Specification-based techniques
  - Structure-based techniques
  - Experience-based techniques
- Benefits and pitfalls of test design techniques
- Criteria for selecting test design techniques
- Test types
Appendix B

Definition, interpretation and examples of K levels

K1 level – knowledge

Definition
Recall appropriate, previously learned information to provide factual answers.

Typical verbs
Recall, select, define, identify.

Examples from testing
Which of the following is the correct definition of . . . ?
Select the correct statement about regression testing from . . .
Identify the best reason for testing early from the following . . .

K2 level – comprehension

Definition
Understand the meaning of information and be able to make comparisons.

Typical verbs
Explain, differentiate, summarise, explain.

Examples from testing
Which of the following correctly describes the main difference between regression testing and confirmation testing . . . ?
Which of the techniques in list A is most appropriate to each of the applications in list B . . . ?

K3 level – application

The K3 level is not used in the Intermediate syllabus or examinations.

K4 level – analysis

Definition
Break down information into parts, examine information and select relevant parts, understand the structure of information.
Definition, interpretation and examples of K levels

Typical verbs

Compare, contrast, discriminate, prioritise, relate.

Examples from testing

Which of the test strategy options is most appropriate for the (scenario) project . . .?
Which review technique would be most appropriate for this (scenario) situation?
Which of the actions below should be highest priority at the present (scenario) stage of this (scenario) project . . .?
  In these examples (scenario) indicate where information from an associated scenario would determine the appropriate answer.

Higher K levels

Levels K5 (synthesise) and K6 (evaluate) are used in Practitioner level syllabuses and examinations but are not used at the Intermediate level.
  Chapter 6 provides further examples of questions at each level.
Index

application domains
analysing 20–21
check of understanding 20
client-server domain
configuration 15
exercise 17
generally 15
technical characteristics 15–17
testing challenges 17
definition 11
mainframe domain
configuration 13
exercise 14–15
technical characteristics 13
testing challenges 13–15
PC-based domain
exercise 12
technical characteristics 11–12
testing challenges 12
risk and 71–72
web-based domain
configuration 18
generally 18
technical characteristics 19
testing challenges 19–20
Ariane 5 case study 27–28
automated scripting techniques 126
black-box techniques
see specification-based techniques
boundary value analysis 74–75
checks of understanding
reviews 48, 53
risk, testing and 70, 73
testing fundamentals 20, 26, 29, 30, 31, 34
test management 94, 102
client-server application domain
configuration 15
generally 15
technical characteristics 15–17
testing challenges 17
commercial off-the-shelf software (COTS)
Fundamental Test Process and 34
implementing 28
COTS
see commercial off-the-shelf software
coverage analysis
definition 137
coverage measurement
practice, in 138
data-driven testing 126
decision coverage 136
decision table testing 129
decision testing 130
detailed work breakdown structure 100
document hierarchy
test management 88–91
DSDM
see Dynamic Systems Development Methodology
Dynamic Systems Development Methodology (DSDM) 23
dynamic testing and analysis 122
economic risks 68–69
entry and exit criteria
test management
exercise 97–98
generally 95–103
management and 96–97
objectivity, importance of 95–96
worked example 98
equivalence partitioning 129
error guessing 131
examination questions
reviews 59–61
risk, testing and 81–82
test management and design 141–142
testing fundamentals 36–37
test management 112–114
examinations
see examination questions; Software Testing Intermediate Certificate
exercises
reviews
assessment 57–58
inspections 51
management reviews 52
teaching reviews 50
walkthroughs 49
risk
product risk 70
project risks 70–71
risk analysis 78
testing fundamentals
client-server application 17
mainframe application 14–15
PC-based application 12
scenario analysis 35
web-based application 20
test management
entry and exit criteria 97–98
estimation 102–103
test monitoring/test control 106
test policy, test strategy, test plans 94
experience-based techniques 131
exploratory testing 131
Foundation Certificate
see ISEB Foundation Certificate
FPA
see function point analysis
function point analysis (FPA) 101
Fundamental Test Process
COTS and 34
generally 32–33
iterative life cycle and 33–34, 40
software development and 34
V life cycle and 33, 40
group consensus estimation 100
IEEE 829 - 1998 test plan topics 93
incident management
exercise 111
mechanisms 110–111
process 108–110
informal reviews 52
inspections
generally 50–51
Intermediate Certificate
see Software Testing Intermediate Certificate
intuition-based estimation 100
ISEB Foundation Certificate
generally 1, 2
syllabus 8
ISEB Intermediate Certificate
see Software Testing Intermediate Certificate
ISEB software testing certificates 1–2
iterative life cycle
Fundamental Test Process and 33–34, 40
generally 22
risk and 72
testing activities 24–25
testing challenges 25–26
test levels 23
keyword-driven testing 126
K levels
generally 146–150
K1 - knowledge
generally 160
sample question 148
K2 - comprehension
generally 160
sample question 149
K3 - application 149
K4 - analysis
generally 160–161
sample question 149
K5 - synthesis 161
K6 - evaluation 161
The official textbook for the ISEB Intermediate Certificate in Software Testing covers the testing fundamentals to help newly qualified software testers learn the skills needed to take them to the next level. The book includes worked examples and sample questions from the examination, self-assessment exercises and also provides a structured revision exercise for those who have attended a course.

Building on the success of their best-selling Foundation book, Hambling and Samaroo take software testers to the next level of professional testing expertise with this perfectly targeted text.

Testing fundamentals

Reviews

Testing and risk

Test management

Test analysis

The only official textbook for the ISEB qualification.

About the authors

Brian Hambling has experienced software development from a developer’s, project manager’s and quality manager’s perspective in a career spanning over 30 years.

Angelina Samaroo CEng is an instructor for the ISEB Foundation and Intermediate Courses in Software Testing.

The authors are both experienced ISEB examination setters and markers.