USER ACCEPTANCE TESTING
BCS, THE CHARTERED INSTITUTE FOR IT

BCS, The Chartered Institute for IT champions the global IT profession and the interests of individuals engaged in that profession for the benefit of all. We promote wider social and economic progress through the advancement of information technology science and practice. We bring together industry, academics, practitioners and government to share knowledge, promote new thinking, inform the design of new curricula, shape public policy and inform the public.

Our vision is to be a world-class organisation for IT. Our 70,000 strong membership includes practitioners, businesses, academics and students in the UK and internationally. We deliver a range of professional development tools for practitioners and employees. A leading IT qualification body, we offer a range of widely recognised qualifications.

Further Information
BCS, The Chartered Institute for IT,
First Floor, Block D,
North Star House, North Star Avenue,
Swindon, SN2 1FA, United Kingdom.
T +44 (0) 1793 417 424
F +44 (0) 1793 417 444
www.bcs.org/contact
List of figures and tables viii
Authors x

INTRODUCTION
What this book is about 1
The role of UAT 2
The costs and benefits of UAT 5
The value of UAT – the reasons we need to do it 6
Stakeholders – who this book is for 10
How to get the best from this book 11
Checklists 12
Case studies 13

1. THE IMPORTANCE OF UAT 15
What is UAT? 15
Why test ISs? 17
Business vulnerability 19
The UAT process 21
From UAT to service delivery 24
UAT and contracts 26
Stakeholders in UAT 29
Chapter summary 31

2. BUSINESS REQUIREMENTS 34
Business requirements 34
Business intent and user expectations 38
Acceptance criteria 39
The requirement types 40
Prioritising business requirements 42
The relationship between business requirements and UAT 43
The relationship between development and UAT 45
Scope of UAT 46
Building a test basis for UAT 48
Chapter summary 54

3. TESTING BASICS FOR UAT 56
What is testing? 56
Test types 58
Testing processes 62
Test-case design techniques 68
LIST OF FIGURES AND TABLES

Figure 0.1  An information system 3
Figure 0.2  Life cycle of an IS 4
Figure 1.1  End-to-end testing of a transaction in an online retail system 17
Figure 1.2  The process of UAT 22
Figure 1.3  The transition to live use 25
Figure 2.1  A sequential development life cycle 45
Figure 2.2  An iterative development life cycle 46
Figure 3.1  Functional testing 59
Figure 3.2  Business process end-to-end testing 61
Figure 5.1  The IS life cycle 94
Figure 5.2  A map of transitions 94
Figure 5.3  Preparation for transitions 95
Figure 5.4  UAT as transition 97
Figure 6.1  The cost and value of a system 106
Figure 7.1  The test design hierarchy 123
Figure 8.1  The life cycle of a test 146
Figure 8.2  Example of an incident report 154
Figure 8.3  Example status report contents 157
Figure 9.1  Process for deciding go/no go for release 160
Figure 9.2  UAT completion report format 167
Figure 10.1  UAT completion report outline 173
Figure 10.2  Post-UAT analysis report 174
Figure C.1  The training process 204

Table 2.1  Business requirements 35
Table 2.2  The requirements document 36
Table 2.3  Excelsior requirement 37
Table 2.4  Use case 53
Table 3.1  Test conditions table 63
Table 3.2  Test cases 64
Table 3.3  Test script 65
Table 6.1  Absence requirements 109
Table 6.2  ATM use cases 112
Table 6.3  Expenses use cases 115
Table 7.1  Login requirements 124
Table 7.2  Test condition matrix 126
Table 7.3  Login use case 129
Table 7.4  Simple login test script 134
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 7.5</td>
<td>A contract data input sheet</td>
<td>141</td>
</tr>
<tr>
<td>Table 8.1</td>
<td>Example of a test schedule</td>
<td>149</td>
</tr>
<tr>
<td>Table 8.2</td>
<td>Example of a test log</td>
<td>152</td>
</tr>
</tbody>
</table>
Pauline van Goethem is a freelance UAT and IT training consultant. She has 17 years’ experience of change management, testing and training, helping to deliver IT and change management projects in industries as diverse as financial services, energy and utilities and TV and social media. She is a certified tester and Scrum Master and has been employed as a project manager, business analyst, UA tester, UAT manager, end user trainer and UAT trainer. She started her own implementation support company in 2006, specialising in offering training and UAT consultancy and services. Pauline is a member of the International Software Testing Qualification Board (ISTQB) Glossary review team.

Brian Hambling has spent nearly 40 years in the IT industry in a wide variety of development, testing and quality management and project management roles. His first task as a software professional was to acceptance test flight software on behalf of the RAF and he has maintained a strong interest in the achievement of software quality throughout his career. He has been an active member of the testing community through the BCS Special Interest Group in Software Testing, at BCS Professional Certification (formerly ISEB) as Chair of the Software Testing Examination Board, and at the ISTQB as a contributor to examination processes and as an examiner.
This is a book about user acceptance testing (UAT) in its many forms and the uses to which it is put. It draws together many strands of material about testing, project management, quality management, team behaviour and other relevant pieces of the complete UAT experience and weaves them into a strong and reliable lifeline for the novice UA tester or stakeholder.

The book has been written to meet the needs of three disparate groups of people. The first of these groups is those who are directly involved in the UAT exercise. For this group we aim to provide a practical and fairly complete guide to the testing of information systems that contain software. As the subtitle indicates, we have adopted a step-by-step approach to enable them to acquire the necessary terminology (jargon) and basic principles as they learn about the practical challenges of UAT and how to deal with them. Within this group we include not only those asked to do the testing (usually end-users of the system) but also those who will have commissioned the system and the testing (sponsors) and those who will be expected to deliver the expected benefits (business managers). The book addresses this group as a whole, but also identifies the specific challenges and provides a practical guide for each of the subgroups within it.

The second group is the professional testers or developers who have been asked to support UAT, or who may simply wish to better understand why UAT is both important and challenging. For this group we explain what kind of support may be needed and why, and we explain where UAT fits within the overall context of structured testing and development life cycles.

The third group is made up of those professionals for whom UAT is an essential ‘tool of the trade’; project managers, quality managers and test managers for example. For this group we seek to place UAT within the overall disciplines of testing, quality management and project management.

Above all the book is intended to explain and explore the significance of an exercise that is commonly neglected in books about testing and often overlooked in books about project management and quality management, yet which is a bridge between these disciplines and, in some respects, is an essential practical expression of each of them.
WHAT THIS BOOK IS ABOUT

Information systems (ISs)

Generally speaking we acquire software to enable us to achieve something of value to us, such as playing a game, making our first million on the stock exchange, or making our business run more efficiently. In some cases software can achieve its purpose without human involvement (except to press a button), but in most cases software interacts with people in a way that achieves a desired result. For example an air traffic control system might provide information to air traffic controllers about where aircraft are, where they are going, how fast and so on. The air traffic controller makes decisions about where each aircraft should go to ensure they are all safely separated and the system relays this information to the pilots of the aircraft. So the purpose of the overall system is to keep aircraft safe and to do that it needs suitably trained people, the air traffic control software, some hardware and some organisation (for example to provide communications between air traffic controllers and pilots). Although this is quite a complex example, it has all the characteristics of an IS. What it most importantly demonstrates is that software is not something that operates in isolation. What users are interested in is not just whether the software does what it should do but whether the system as a whole achieves its purpose and whether they, as users, will be able to operate the system effectively (and without undue stress).

Characteristics of a system

A system is a collection of interdependent components that interact according to a plan to achieve a specific goal.

The key thing to remember about a system is that ‘the whole is greater than the sum of the parts’. So a team focused on a goal can achieve much more than the team members could achieve individually if they were not a team.

Interdependence means that every part is vital.

Systems only behave like systems when they have a clear purpose.

Characteristics of an IS

An IS is a system comprising humans, computers, organisation, processes and a single purpose – the business intent.

It is the business intent that makes this collection a system. The components are interdependent, so neither the computers nor the humans can achieve the business intent on their own. Organisation and processes manage the interactions.

When we build or test an IS we must consider all the components, and the interactions, and the common business intent. Figure 0.1 provides a schematic view of an IS for a business.
In a business context we would have business processes that involve human beings using information to add business value, such as a supermarket checkout assistant adding up all the retail values of the items purchased by a customer, preparing a bill for the customer to pay, accepting payment and providing a receipt. All of this could be done manually, but this is a process that is usually automated for reasons of efficiency and for the advantage of having all the details of each transaction available as information to be processed in making other decisions such as when to restock. In this example the effectiveness of the overall process would probably be measured by how quickly and accurately a single customer’s shopping can be handled. Here again the whole process is measured, not just the part the software does, because that is the process that adds value to the business. From a user perspective a checkout operator will be sitting at a checkout for perhaps hours at a stretch. To them it will be vitally important that the efficient operation of the checkout is achievable routinely and without heroic effort. If the scanner refused to scan some items, the conveyor broke down at regular intervals or the bills produced were even occasionally incorrect, the impact on the user would undoubtedly be increased stress.

So this book is about testing ISs and not just software, and it is also about considering all the perspectives of all the people involved in building, operating and using the system to achieve its expected business benefits.
Testing ISs

When we set up a UAT exercise we are testing a system and not just a piece of software, which means we are really interested in knowing whether the overall IS works. If the system does not work as we expect, there could be a number of reasons, and among these is the possibility that the software does not do what it should. But the software could be working perfectly and some other aspect of the system could be at fault.

To make this as clear as possible we need to take a brief look at how ISs are created – not in detail but as a high-level view of what happens to an IS from the first idea through to the realisation. This high-level perspective is usually called a life cycle.

The life cycle of an IS is simply a model of everything that happens to it from the time it is first envisaged to the time it is retired. Figure 0.2 shows a simple schematic view of what an IS life cycle might look like.

What this life-cycle model describes is a process that begins with requirements (which are the means of describing what the system needs to do), proceeds through a vague and shadowy phase called development (which we will only delve into occasionally and only if we absolutely need to understand something about it) before reaching UAT as the final stage before the system is ‘released’ into service. We will take a closer look at how requirements are arrived at and documented in Chapter 2.
One key feature of this life cycle is the balance of time, effort and money spent on it. The part of the life cycle that usually gets most of the attention is the initial development, even though that accounts for only about 20 per cent of the total spend. Most of a system’s life is spent in being used, maintained, modified and repaired. The better the system is at the end of development, the lower that large chunk of life-cycle cost is likely to be – and the ‘gatekeeper’ between development and the rest of the life cycle is UAT. Effective UAT can avoid excessive maintenance costs (for example repairing defects), reduce the cost of modifications (for example to improve the system’s usability) and prolong the life of the system, so giving a better return on the original investment in development.

UAT is the final frontier between initial development and the rest of the system’s life, a frontier beyond which could be a world in which your new IT system makes your life easier, saves you time and always turns out great results – or a world where using your IT system is a nightmare in which nothing functions, the system seems determined to prevent you completing your work and the outputs are never what you wanted.

Anyone accustomed to using IT systems knows that these two ‘parallel universes’ are actually very close together and that predicting which of those worlds you will be transported to when your system is declared operational is not easy. That is the role of UAT – to provide a glimpse of the future (just) in time to avert a disaster if that is what you foresee and to give the users the best possible start with using the system.

THE ROLE OF UAT

UAT is a test of an IS from the perspective of the users and other stakeholders for whom it has been built or acquired. UAT is not just a test of the software, nor of the functionality, performance, reliability and security, nor of the usability of the system. That does not mean UAT is not concerned with any or all of those areas, but that its primary concern and absolute focus is on whether or not the system can deliver the expected business benefits when operated by its designated users. All of the specialist areas mentioned will have, or certainly should have, been tested during development. None of those tests, however, answers the question ‘Is the system fit for purpose?’ UAT answers that question unequivocally by testing it against the reason it was built or acquired, using the eventual end-users as testers and, as far as possible, utilising the user documentation prepared to support their use of the system. This will entail not just exercising the system in some random or even structured way; it will entail using the system to enable business processes that deliver value using realistic (or actual) scenarios, data and operations. UAT is the nearest we get to ‘the real thing’ without actually taking the risk of releasing the system into service, and it is the exercise that will enable us to make a rational judgement about whether to take the risk of releasing it into service.

THE COSTS AND BENEFITS OF UAT

UAT is an expensive exercise. As well as the hard costs of actually doing the testing, we have to factor in the additional time that development resources are tied up before roll-out of the system allows them to be released to other projects.
USER ACCEPTANCE TESTING

The largest components of the direct cost of UAT are:

- The cost of diverting end-user staff from their normal work to plan and carry out UAT. As well as salary costs there is the loss of whatever revenue the staff would have generated or the delay in realising that revenue.
- The cost of training or familiarisation for the UAT team (and subsequently for all end-users).
- The cost of test environments for UAT.

A UAT exercise with a team of six staff that takes two weeks to complete might have direct costs of around £6,000–£10,000 for salaries alone.

What do we have to offset this cost?

THE VALUE OF UAT – THE REASONS WE NEED TO DO IT

In case you are not convinced of the need to go to the time and expense of a UAT exercise, we will take a brief look at the world of ISs and what can go wrong with them.

Reason 1: risk management – avoiding expensive failures

HIGH-PROFILE IT FAILURES

US trading systems, unable to cope with the number of transactions, fail causing panic among sellers.
On Friday 17 October 1987, after a number of Securities and Exchange Commission (SEC) investigations of insider trading and the London stock market closing due to severe weather conditions, it looked like a sustained period of growth was going to come to a sudden halt for the US stock market. On Black Monday the crash started in Far Eastern markets and, after a weekend of worrying about their shares, investors sold stock in a mass exodus of the market, generating a flood of sell orders and crashing trading systems. As a result more than 20 per cent was wiped off the value of the US stock market in a single day.

A new computer system means 500,000 UK passports cannot be issued on time.
In the summer of 1999 the UK Passport Agency brought in a new Siemens computer system without sufficiently testing it or sufficiently training staff on how to use it. At the same time an unusually high demand for passports was driven by a change in the law, which meant that children under the age of 16 required their own passport when travelling abroad. As a result the Home Office had to pay millions in compensation for missed holidays and staff overtime.

A memory leak in the London Ambulance Service’s new emergency dispatch system leaves 46 people dead.
Before 1992, office staff decided what ambulances should be dispatched where and it seemed that there were efficiencies that could be achieved by using a
computerised system. Just a few hours after it was deployed problems began to arise with the new system. The root cause of the main breakdown of the system was a memory leak in a small portion of code. The system retained memory about incident information on the file server even after it was no longer needed. As with any memory leak, after enough time, the memory filled up and caused the system to fail. The next day, the Ambulance Service switched back to a part-manual system, and shut down the computer system completely when it stopped working altogether eight days later. In those nine days the lives of up to 46 people might have been saved had an ambulance been able to get to them in time.

Microsoft rushes to release a flawed games console to stay ahead of the competition. The Xbox 360 games console was released by Microsoft in November of 2005, just ahead of Nintendo’s PlayStation. It quickly became clear that the Xbox was subject to a number of technical problems and failures; a series of glowing red lights flashing on the face of the console, later nicknamed the ‘Red Ring of Death’, being the most well known. It was not until 5 July 2007 that Microsoft published an open letter recognising the console’s problems, as well as announcing a three-year warranty for every Xbox 360 console that experienced the general hardware failure. The design issues were alleged to be the end results of management decisions and inadequate testing resources prior to the console’s release.

Not all ISs end in high-profile failure; many are highly successful and trouble free over a long life, but it is nevertheless true that the statistical likelihood of failure for an IS is relatively high.

In 1995 The Standish Group published a report based on research into a large number of failures in software systems. It was called the CHAOS report (The Standish Group has never revealed what the acronym stands for, only conceding that a few select people in the organisation know) and it contained some frightening statistics, for example that only 16.2 per cent of IS projects were completed on time and on budget. Worse still, completion often required significant watering down of the original specification for these systems.

The statistics relate to system failures and these are not necessarily related to UAT, but the CHAOS report analysed their statistical evidence and identified key factors that contributed to failure. These factors point to problems in some aspects of system development that UAT can throw some light on, such as lack of user involvement and poorly defined requirements.

The original CHAOS report was published quite a long time ago, but The Standish Group has repeated its research and published updated results over a number of years. The results have gradually been improving with the introduction of agile project models and a greater awareness of the issues that cause projects to fail. The 2011 edition of the CHAOS report found that 37 per cent of all projects succeeded and Jim Johnson, Chairman of The Standish Group, stated, ‘This year’s results represent the highest success rate in the history of the CHAOS research. We clearly are entering a new understanding of why projects succeed or fail.’ Although the results show a significant improvement on the original 1995 figures, it is a sobering thought that, even with these
unprecedented results, 63 per cent of projects in the study did not succeed. Although other researchers have criticised The Standish Group for providing little or no context to its figures, for instance by not distinguishing between projects where going over the deadline means failure and those where an reasonable overrun can be easily tolerated, nevertheless the CHAOS report presents a very useful list of the key factors that contribute to project failures, the knowledge and application of which can benefit any project. We shall explore some of these factors and how we can learn from them in Chapter 1.

Avoiding high-profile disasters or even low-profile mishaps is not solely about performing effective UAT of course. By the time we get to UAT the seeds of failure have not only been sown, they may well have already germinated; but well-planned UAT may still be able to highlight the potential for disaster before it actually happens. This is an important reason, but not the only one, for doing UAT.

**Reason 2: confidence building – achieving expected business benefits**

If we have acquired some expensive software or some software designed to do something really important for our business, we need to be sure before we accept it that we have minimised the risk of it not doing the job we acquired it for. This is not just a repetition of disaster avoidance, but a form of ‘good housekeeping’ that ensures we have addressed all the things that could go wrong when we start to use the system. Many of these may be only small things, but they can add up to a problem in service, and some of these smaller issues may not have been addressed by the testing done during development, which was designed to ensure the system meets its specification rather than ensuring that the users will be able to use it effectively. In Chapter 3 we will show how risk-based testing can be used to minimise the risk.

We will also need to be sure that the people who use the software will get the expected productivity or other benefits and that the business will get the efficiency or other improvements we intended. To do this we need to put the system to use in a realistic environment and work through some examples of situations in which it is expected to add value, such as streamlining processes or reducing the cost of production. This kind of testing will be based on an understanding of how the system will be used to gain business benefits and setting up a realistic ‘trial’ to see that it is capable of delivering those benefits. We explain in Chapter 6 how we can set up this kind of test.

**Reason 3: assessment – getting the business processes right**

As well as making sure that the system will not only do what it is supposed to do but will also be usable by our staff, we need to give the end-users an opportunity to exercise the system in whatever way they have agreed to work to make it effective. Users will need to check system behaviour as part of the overall business processes, using it in the way that has been agreed. We will not only treat the system as an IS and test its behaviour when the intended business transactions are passed through it, but we will also test that end-users can operate the system in the way that is needed to make those processes work effectively.

One further benefit of UAT is that the act of determining a system’s fitness for purpose necessarily involves comparing the system’s behaviour with user expectations. If the
comparison is done systematically and formally, then the primary ‘result’ of UAT may be to accept or not to accept the system. A valuable secondary result, though, is that the comparison will have generated a detailed understanding of how well the system matches expectations. This comparison can be made the basis of an assessment of the system’s current capability and value to the business, which in turn can be used to underpin a strategy for future development or enhancement. It enables us to know exactly what the strengths and weaknesses of the system are and to assess value for money and potential for improving business value from the investment in development.

**Reason 4: preparation – assessing readiness for service**

Finally we want to be sure that the software is fit to release into the business and that the business is ready to make use of it, so we will need to assess readiness. After we have completed the testing based on risks, benefits and usability, we may have discovered some problems that need to be resolved, some defects that need to be corrected or some other aspect that needs to be addressed (such as training). The combination of these things will have an impact on when a system will be completely ready for the users to take on. This aspect of readiness is about how robust the system is in operation, how closely it fits expectations, how many ‘workarounds’ have been found necessary and a host of other factors that may affect whether or not the system is ready to deliver value to the business. This is usually a matter of discussion and negotiation, and results in a decision to release the software. Chapter 9 will explore the mechanisms we can put in place to make that decision as objective as possible. If we consider this aspect of UAT well in advance (as we should), we can define a set of acceptance criteria that will be used to come to a decision about release and so avoid the conflicts that can arise at this stage.

One important thing to say in this context is that UAT is distinct from training, familiarisation and all other aspects of roll-out. It has its own set of objectives that would be seriously jeopardised if UAT was carried out by unprepared, untrained end-users who were unfamiliar with the system. Having said that, however, the experience of UAT will be a learning exercise that is second to none and the experience gained during UAT can be captured and utilised in training, familiarising and preparing the end-user population for using the system effectively – including dealing with any issues raised by UAT for which ‘workarounds’ have been defined.

**THE ENIGMA OF SOFTWARE**

It is always difficult to pin down the cost of software, but it is true that the case usually made for acquiring software – that it saves staff costs – has been shown to be invalid in most cases. Software is expensive to acquire and operate over its complete life cycle from inception to disposal.

A second key characteristic of software is its inflexibility. We imagine software to be very elastic, so that we can change it at will, but this is far from the truth. While it is very easy to change a line of code (in the sense that code can be updated at almost no cost because it is electronic and lives on a machine that can alter its contents instantly), it is very difficult to change the behaviour of a software application in a way that has no negative consequences and meets all our expectations. Software
is actually very inflexible as a tool for achieving a business purpose. Once installed the cost and impact of modification can be, and often is, very high.

Thirdly, software is not visible. It has no moving parts and so has nothing that we can see operating. When it stops working it does not ‘break’ in the way that physical systems do. The computers usually keep operating and doing something; they just stop doing what we want them to. This can be very frustrating especially if we find that the software is not malfunctioning at all, but just responding to some unexpected input.

These three characteristics make software a difficult medium to work with and a difficult business tool to operate unless it is working exactly as we expect it to.

The enigmatic nature of software as a medium makes the risk associated with deploying ISs containing software significant and UAT is the best mechanism we have for managing that risk for the user community. We have already stated that the main reason why UAT is a worthwhile exercise is that it can help to avoid some of the costs of dealing with failures and of modifying the software after its acceptance. The potential costs of failure can be judged from the case study examples, but the cost of failure is invariably higher than expected in a business context because of the ramifications of failure in the tightly knit relationship between business processes and ISs.

There is also one other major benefit from UAT. The skills and newly acquired experience, combined with the invaluable experience that stakeholders bring with them to UAT, have enormous potential value in preparing for and achieving a painless roll-out. Beyond that, the insights gained by those spending time in testing a system can be harnessed to enhance future business process development, training and support. There is potentially a massive benefit to be reaped from the experience of UAT for the individuals concerned and for the business as a whole.

STAKEHOLDERS – WHO THIS BOOK IS FOR

This book is aimed at the groups of people who together comprise the principal stakeholders of UAT:

- sponsors (typically owners or senior managers who have decided to acquire software of some kind to enhance their business);
- business managers, who will be responsible for ensuring the IS delivers the expected business benefits;
- end-users, who will be most interested in the way they are able to interact with the system to achieve their tasks;
- professional developers and testers, who will be concerned with providing the most effective support for UAT;
- managers of processes, practices and standards, who will take an interest in making UAT a cost-effective process using best practice.
In each of these groups there may be some who have no experience of testing at all and for whom an assignment to a UAT team is a daunting prospect.

If you are in this category then this book is for you, and it speaks to you very directly:

- by using non-technical language wherever possible;
- by assuming no prior knowledge of testing at all;
- by identifying what you need to do in a step-by-step fashion;
- by providing case studies, examples and exercises for you to practise at least some of the tasks.

If you are a sponsor and are in the process of acquiring a new IS, the book will identify the role of UAT in ensuring you get the system you are expecting, but it will also help you to understand how UAT fits into your overall planning and what benefits it brings in terms of training and supporting the inevitable change to working practices that new ISs invariably bring.

If you are a business manager who will be responsible for the effective operation of the system in achieving business benefits, the book will explain how those benefits are defined and measured and how you can ensure UAT is shaped to provide you with the information you need to prepare for the system’s release into service in your part of the business.

If you are an end-user tasked with testing an IS, the book will provide you with all the tools and techniques you will need to be an effective UA tester.

If you are in the group of ‘interested professionals’ who may need to support UAT, you will find in this book a clear exposition of how and where the world of UAT interfaces with the work of professional developers and testers and how best to manage that interface and support the UAT activity.

Finally, if your role involves managing and enhancing practice of some kind as a test manager, quality manager or project manager, you will find the book useful as a manual of key techniques, methods and tools. Quality managers wanting to improve UAT practices, for example, will find here a comprehensive review of UAT from which best practice can be distilled and then customised to fit your organisational culture, size and budget.

**HOW TO GET THE BEST FROM THIS BOOK**

The two main premises of this book are:

1. We can and must make UAT effective to reduce the risk and cost of underperforming ISs.
2. The benefits of UAT outweigh the costs.

To explain why and how these results can be achieved the book has been structured as described below.
The first three chapters provide all the background and justification for a step-by-step approach to UAT:

- Chapter 1 – The Importance of UAT;
- Chapter 2 – Business Requirements;
- Chapter 3 – Testing Basics for UAT.

The next two chapters are dedicated to the practical issues of creating, shaping, training and deploying an effective UAT team:

- Chapter 4 – The UAT Team;
- Chapter 5 – UAT as Transition.

The remaining chapters explain the step-by-step approach in practical detail:

- Chapter 6 – Preparing for UAT – Planning;
- Chapter 7 – Test Design for UAT;
- Chapter 8 – Implementing the Tests;
- Chapter 9 – Evaluating the System;
- Chapter 10 – Life after UAT.

We sincerely hope that all readers of the book will find it interesting and rewarding enough to want to read it from cover to cover. However, the first encounter with any book of this kind needs some kind of plan to extract the maximum value in the minimum time to provide a starting point from which a more detailed exploration of the content can begin. The initial route through the book naturally depends on your starting point.

All readers are strongly advised to read Chapters 1–3 as general background and as preparation for the practical approach later in the book.

Those responsible for managing a UAT team should read Chapters 4 and 5, and all those selected to take part in UAT will gain an understanding of the skills needed and the style of operation for a UAT team by reading these two chapters.

All readers are encouraged to read Chapters 6–10 although some chapters will be more relevant to some groups than to others. Within these chapters all of the stakeholders are considered and information of particular importance to one group or another is labelled appropriately to guide your initial reading.

**CHECKLISTS**

Whatever approach you take to reading the book you will be guided through all the key steps in UAT through Chapters 6 – 10. Your reading should help you to understand what the key steps are and why they are ordered as they are.
To help you in applying what you learn from the book, you will find a set of checklists for all the major steps in UAT at Appendix A. The purpose of the checklists is to present the key steps in a logical order, uncluttered by explanation, but also indicate where you can find supporting material in the book. We hope you will find these useful when you come to apply the step-by-step approach to your projects.

**CASE STUDIES**

Throughout the book we will make use of case studies, examples and exercises. The case studies are used to provide documented real-life examples of the things we are talking about; the examples will give you practical guidance on how to use the techniques, methods and approaches we present; the exercises will provide you with opportunities to try out some of the techniques.

In addition to these in-text components we end each chapter with a few multiple-choice questions that you can use as a check that you have understood the key ideas in the chapter and we also provide you with questions that you might like to think about. These raise some of the issues and challenges that you might face as a user acceptance (UA) tester and give you an opportunity to think about them before you have to deal with any real challenges in a project context. We have provided answers to all the exercises in Appendix B so you can check your answers against ours. The more open-ended questions at the end of the chapters have no correct answer, but we have included some of our own thoughts on these questions that you can read if you want to. We make no claims about these being the best responses to the questions, but they are the ones that we have developed from our experience.

Finally, before we progress on to Chapter 1 and take the first step in introducing the UAT process, we are going to introduce you to a small imaginary company called HMF that is acquiring an IS. We will use the HMF case study mainly to wrap around the more significant examples we provide, especially in Chapters 6–9, to provide continuity of ideas and hopefully to help you see how information is built up at each step of the approach for use in later stages.

**CASE STUDY – HMF AND THE EXCELSIOR SYSTEM**

HMF is a web-based insurance company offering low-cost insurance solutions for a broad range of domestic and commercial risks. HMF has rapidly developed its product range by acquiring smaller specialised insurance providers and integrating their specialised products into its own platform.

HMF has recently conducted a rationalisation programme to ensure that staff numbers and skills are optimal for the integrated product range and to harmonise processes across the business. The rationalisation exercise revealed that accounting and human resource (HR) practices had become fragmented after multiple acquisitions and integrations, and a decision was taken to develop an IS to support common accounting practices across the business and, as a secondary objective, to enhance HR process integration. At the heart of the IS a new software package, known as Excelsior, was specified to run on existing HMF servers.
The Excelsior accounting system was commissioned as a custom-built accounting solution based on a collection of standard accounting modules configured in an architecture that supports full integration of all the modules’ services and also enables selected HR applications to be incorporated. The development will be incremental to allow basic functions to become well established before adding further complexity and the first incremental delivery is now ready for UAT. At the current stage of development the accounting functions available will be:

- general accounting;
- purchase orders;
- contracts;
- payments.

In addition the following HR services will be available:

- absence (to book days off, sickness and other absence and manage requests for absence for a team);
- expenses (to enable submission of expenses forms and receipts for payment and to approve expenses related to a team);
- changing contact and bank details and updating personal details held by HR;
- internal purchase orders (request purchases and approve purchases related to a team, marking a purchase order as fulfilled);
- training (request training, take online training, record training attendance);
- administration (support and managerial functionality);
- jobs (search for, list or apply for vacancies, approve applications related to a team).

Each module has a unique title related to Excelsior, for example Excelsior – Training.

Every service is listed on the main menu on the Excelsior home page but access is restricted for some modules. Users are granted access to services based on their role-based login profile. Where access is denied, a link appears that allows users to request access from the appropriate person.

We hope you find HMF and Excelsior, and all the case studies, examples, exercises and the questions at the chapter ends, helpful in bringing the ideas to life and enabling you to gain some insight into what is happening during a real UAT exercise.
The introduction presented some general concepts that help to get a better understanding of UAT and introduced examples of high-profile project failures that, if not caused by UAT, were certainly not prevented by it. Chapter 1 provides an overview of UAT, its purpose and its relationship to an implementation project and the people who take part in it. You will find out why UAT is different from other types of testing and yet often uses the same processes – one of which is the fundamental test process. Finally you will discover what the different types of UAT are, who the stakeholders of UAT are and what each role has to offer to and to gain from the UAT process.

**Topics covered in this chapter**

- What is UAT?
- Why test information systems?
- Business vulnerability
- The UAT process
- From UAT to service delivery
- UAT and contracts
- Stakeholders in UAT

**WHAT IS UAT?**

UAT stands for user acceptance testing and is commonly used to refer to the end-user software testing carried out prior to a new information system (IS) being introduced to an organisation. The primary objective of UAT is to ensure the new system does what it set out to do and meets the requirements the business has of it. Here is a definition of the term UAT taken from the ISTQB Glossary of Testing Terms:

Formal testing with respect to user needs, requirements, and business processes, conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.
Three aspects of this definition are important and will drive what we do in preparing and implementing UAT:

1. UAT requires ‘formal testing’, which means that tests should be designed and executed in a structured way that provides objective evidence of the acceptability or otherwise of the system.

2. The definition speaks of testing with respect to ‘user needs, requirements, and business processes’. It does not mention any particular specification document but it does draw attention to what users need and it goes beyond testing software to include business processes.

3. The definition speaks of satisfying ‘acceptance criteria’, which define what is acceptable to the users.

We will explore all three of these key ideas in the book and we will be developing our UAT around acceptance criteria and formal testing of user needs and business processes.

Although other testing takes place before UAT is carried out, UAT is unique in that it is the first time the completed system is tested by users against user needs. Testing during development is based on ensuring that what is delivered matches the corresponding technical specification, but the fit between an IS and its business users is more intimate and it is the unique user perspective that can identify the problems of fitness for purpose in a business context. Without it a system that has been tested by developers and professional testers, and which has passed every test, can still fail at the first hurdle of real use.

That is why UAT needs to include ‘end-to-end’ testing that exercises complete business processes, including the computer system, rather than testing the computer system in isolation (Figure 1.1). For example in an online retail system, a test of the ordering process must check that an order placed by a customer is accepted and that it triggers identifying, packaging and shipping of the correct product. But these processes must trigger other processes to complete a transaction, for example that the order generates and sends a correct invoice and that payment received is correctly matched with the invoice to close the transaction. Although the scenario is simple, this is a fairly complex set of interacting processes and there is also a time lapse between the initial order and the payment of the invoice, so a full end-to-end test of the transaction will need to follow...
an order through the system and over time to ensure all the necessary interactions
happen correctly and produce a successful result.

**Figure 1.1 End-to-end testing of a transaction in an online retail system**

![Diagram of end-to-end testing process]

**WHY TEST ISs?**

Software is one of the greatest examples of human creativity and endeavour. The growth in what can be achieved with computers and software in recent decades has been phenomenal and the trend is likely to continue. Software and software-based systems such as IS are becoming ever more endemic in business life. As the complexity and subtlety of the relationship between IS and business increase, so the impact of failure, by which we mean any mismatch between what an IS actually does and what we need it to do, becomes greater.

This increase in the risk of using an IS must be balanced by increased attention to risk management. There are many facets of risk management but, in the world of software and IS, testing remains one of the key risk management disciplines. As software has become more complex, testing has had to become more effective at identifying problems and potential problems so that they can be resolved before a system fails. As we will explain in Chapter 3, where the sophistication of development activity
has increased, testing has also had to become more sophisticated. Why? Because with greater complexity comes greater risk of human mistakes in design and in implementation; because greater integration between human activity and the systems that enable it brings more opportunities for mismatch; and because the many essential characteristics of ISs (such as reliability, usability and security) all need to be assured before the system can be trusted to deliver its services in a real business situation.

The increased dependence of business on IS has also made the relationship between systems and business processes more critical to business success, and in many cases the desired behaviour of an IS cannot be adequately described in technical specifications. Most testing during development is based on ensuring that what is delivered matches the corresponding technical specification, but the fit between an IS and its business users is more intimate and more critical than the fit between craftspeople and their tools; it is rather more like the fit of a garment that is ‘made to measure’ so that an individual can work comfortably and effectively in it all day and every day. Whether that kind of fit has been achieved is not something that can be determined by anyone other than the user. That is why UAT is different from all other kinds of testing and also why users must take part in it. It is the unique user perspective that can identify the problems of fitness for purpose in a business context and, without it, a system that has been tested by developers and professional testers, and which has passed every test, can still fail at the first hurdle of real use.

**REASONS WHY WE NEED TO TEST**

The fact that we live in an imperfect world is, at the most basic level, why we may have trouble creating a perfect IS and why it is important to test them.

There may be many reasons why requirements do not exist at all or are not complete and up to date:

- It is often hard for the sponsor or end-users to imagine what their requirements are going to be. Ideas about what the system should become change over time and in the context of development.
- Even when the requirements are clear in the minds of the sponsor or end-user, they may have trouble communicating the requirement. What is written in the requirements document may therefore not reflect what was intended.
- If the requirements reflect what was intended by the sponsor or end-user, the developers may interpret requirements incorrectly because they will bring their own assumptions to the process.
- Even if correctly imagined, communicated, written and interpreted, developers may make mistakes when writing the code that creates the functionality described in the requirements because they are human and therefore fallible.

All of these have been cited as significant issues in studies of why systems fail.
Buisness Vulnerability

The relationship we have with IS in business is a unique one. In most cases the system does not just make our lives a little easier; it actually carries on some or all of our business processes in a way that may be difficult or even impossible to do manually once the system is in place. The intimate relationship between businesses and the ISs that serve them makes us particularly vulnerable when a system is first introduced or when it is changed or updated. The term ‘business critical’ is often used for systems that have a significant impact on the way a business operates and delivers its services. To be clear, this is not just about avoiding high-profile failure; it is about ensuring that each and every time we initiate a new IS or change the relationship between a business and its IS, we take care that the new relationship will be as we expected and planned it to be.

The UAT process is the last, and arguably the most important check, before an IS is rolled out. ISs are becoming more and more embedded in business processes and in delivery of business value and, as a result, they are also becoming more complex. The combination of the complexity and business criticality of ISs leaves little margin for error.

The Chaos Report – The Standish Group

The Standish Group was formed in 1985 by a group of IT professionals whose aim was to collect case information on real-life IT failures in order to improve IT project success rates. The original CHAOS report, published in 1994, examined 8,380 projects in order to determine whether they were successful and, if not, what the causes of the failures were. The report found the following percentages of projects were successful, unsuccessful or impaired:

- Successful projects completed on time, within budget, containing all the features and functions initially specified – 16 per cent.
- Challenged projects completed and working, but over budget, over time and offering fewer features and functions than initially specified – 53 per cent.
- Impaired projects cancelled at any point during the development cycle or not used post-completion – 31 per cent.

The report also identified the reasons for the failures, listed from the most often occurring to the least often occurring:

1. incomplete requirements;
2. lack of user involvement;
3. lack of resources;
4. unrealistic expectations;
5. other;
6. lack of executive support;
7. changing requirements and specifications;
8. lack of planning;
9. did not need it any longer;
10. lack of IT management;
11. technology illiteracy.

The 2011 edition of the CHAOS Manifesto, an annual report from The Standish Group that examines trends in software project success, found that in a marked improvement on the 1994 figures 37 per cent of all projects succeeded, 42 per cent of projects were challenged and the remaining 21 per cent were considered a failure. Notably, however, the majority of projects are still challenged or impaired according to their findings.

Not all the reasons listed are relevant to UAT, but those that are relevant will be at the forefront of our thinking when planning and executing UAT activities.

The massive disruption to NatWest Group customers following a routine update to its software is an example of what can happen when systems do not perform as expected in a business-critical environment such as banking. The introduction of a new baggage-handling system for Heathrow Airport’s new Terminal 5 is another recent example.

THE HEATHROW TERMINAL 5 OPENING

In March 2008 Willie Walsh, CEO of British Airways (BA), gave evidence to the House of Commons Transport Committee about the problems with the opening of London Heathrow Airport’s Terminal 5. In his evidence he said:

... we had compromised on the testing regime as a result of delays in completing the building programme and the fact that we compromised on the testing of the building did impact on the operation of T5 in the first few days after its opening.

In the first five days of operation the cost to BA of the operational problems was estimated at £16 million and a total of 23,205 bags were ‘misconnected’. The extensive delays experienced by passengers, the sight of the unfinished buildings and the problems caused by delays and unprepared staff also caused embarrassment to BA, the British Airports Authority (BAA) and the UK government (House of Commons Transport Committee 2008). One of our main aims in this book is to explain why these kinds of outcomes happen and how they can be avoided.

Meticulous planning, use of highly experienced and competent practitioners and good project management are all essential to success, but they are not enough. We need
to know exactly what will happen from the point when the first real user logs on to a system, and no amount of time and money spent on modelling, problem solving, focus groups, contingency planning or any other discipline will tell us what we need to know as effectively as a well-planned, well-structured and well-designed UAT. The immediate cost of failure (£16 million in a few days in BA’s case) and the ongoing impact on a business of even a small interruption in a key service, such as paying bills through a bank, have a huge impact – significant enough to cause even large companies to fail.

It is important to understand at this stage that although UAT has so far been described as the activity that can verify the usefulness of the new IS and save a project from going horribly wrong, UAT may not be perceived as such by others. UAT is sometimes perceived as an expensive evil that interrupts ‘business as usual’, diverts important staff, and costs a disproportionate amount of time and money. The number of high-profile failures associated with releasing software systems that were not ready ensures that most organisations make at least some attempt at an acceptance test, often by pressing reluctant users into an activity in which they feel well outside their ‘comfort zone’. Does this have to be the case? Can we make UAT an exercise that is independent of the IT community that built the system yet in partnership with it to achieve the best possible result? The premise of the book as a whole is that we can.

Before we embark on UAT we need to be sure that the cost and effort are worthwhile and we also need to understand the activity well enough to do the best job with the fewest resources. If we understand the activities of UAT well enough to carry them out effectively, the parties involved in UAT should be well prepared, the testing should prove the usefulness of the system and little testing time should be wasted or subject to delay.

THE UAT PROCESS

The process we will describe in this book will take you through each stage of achieving successful UAT in a step-by-step fashion. We will first explain enough background to enable you to see how the parts of the process fit together and we will provide you with tools and techniques that you will need to do your testing. Next we will describe how to recruit and prepare an effective UAT team. Finally we will take you through the stages of a UAT project, again step by step, so that you can apply all the tools and techniques at the right time and in the right way. The basic process is shown in Figure 1.2 and it is based around the fundamental test process as defined in the ISTQB Glossary so that you can compare it with other testing processes you may read about elsewhere.
Figure 1.2 The process of UAT

Recruit/Train UAT Team

Set Up/Plan

Design Tests

Implement Tests

Report/Evaluate

Decision Making

System Release

Follow Up

Requirements

Design Process

Test Environment

Stakeholders

System

Stakeholders

Rework/Delay

End-user Training

Support

Roll-out Plan

User Guide

Workarounds
The fundamental test process (FTP) has five stages.

### The fundamental test process (FTP)

The ISTQB Glossary describes the FTP as comprising five test stages:

1. test planning and control;
2. test analysis and design;
3. test implementation and execution;
4. evaluating exit criteria and reporting;
5. test closure activities.

Each stage represents a number of key tasks that need to be carried out in order to complete the formal testing as described in the ISTQB definition of UAT. The stages also represent the order in which these tasks are carried out, although we will see that in real life we will probably need to move between these stages, especially when additional functionality is identified that needs to be tested or faults are repaired that need to be retested. The FTP can be applied to any testing level or to an overall testing project.

For our UAT process we will add an additional stage at the beginning for the unique element of UAT of recruiting and forming a UAT team – unique because UAT is conducted by end-users rather than test professionals; it does not simply follow on as one more testing stage for the professional testers to do. UAT team recruitment, development and training are explained in Chapter 4.

The second stage aligns with the FTP’s ‘test planning and control’ stage; it entails setting up the UAT project, identifying its purpose and goals, ensuring that we have a sound basis for testing, and planning all the remaining stages. The key input is requirements, which can be problematic, and Chapter 2 explains how we ensure we have the right requirements to work from. Chapter 6 explains in detail what we have to do at this stage.

The third stage aligns with the FTP’s ‘test analysis and design’ stage; it entails generating tests from the requirements, using well-defined processes that we will explain in Chapter 3. The output from this step will be a complete set of tests that we need to implement to achieve our overall goal; this is explained in detail in Chapter 7.

The fourth stage aligns with the FTP’s ‘test implementation and execution’ stage. At this point we need the system itself and any test environments needed to enable all the tests to be successfully executed. As we test we will generate data about the status of the system that we can report to stakeholders and we report any problems we find to the development team for investigation and, if appropriate, correction. This is explained in detail in Chapter 8.

Next comes a step that aligns with the FTP’s ‘evaluating exit criteria and reporting’ stage. In our UAT process this is the point at which we gather together all the information from all the testing to determine whether the acceptance criteria have been met and, if not, how far short the system falls and what could be done about any problems. This vital information enables the final acceptance decision to be taken and guides the follow-up from UAT to system deployment. This is explained in detail in Chapter 9.
The final step aligns with the FTP’s ‘test closure activities’ stage and extends it to consider the implications of what was found in UAT and what was decided about deployment. Valuable experience and knowledge gained in UAT can now be used to identify any ‘workaround’ solutions needed to overcome shortcomings of the system, guide end-user training and the user documentation needed to support users, shape the kind and level of technical support needed, and plan deployment mechanisms and timescales. This is all explained in Chapter 10.

<table>
<thead>
<tr>
<th>THE MANY FACES OF UAT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Contract acceptance testing</strong></td>
</tr>
<tr>
<td>Testing to determine whether contractual conditions have been met, so UAT is based on the requirements defined in the original contract for a system acquired from a third-party supplier.</td>
</tr>
</tbody>
</table>

| **Factory/site acceptance testing** |
| For systems requiring installation after build, there may be a factory acceptance test to establish that the system meets requirements on completion of factory build before installation. Installation may have its own contractual conditions and there may be stage payments for completion of each stage, especially if installation involves shipping overseas. In this case factory acceptance will normally be based on requirements in the contract and site acceptance will be based on achieving the same set of requirements in an installed location. |

| **Alpha/beta testing** |
| When requirements are difficult to define or deliberately open-ended, a conventional contract acceptance may not be possible. In this case some form of ‘testing in use’ may be done. Alpha testing would be conducted at the developers’ premises and beta testing would be at the customers’ premises. The actual testing may be based on specific activities but more often is left to the users’ discretion. For obvious reasons this is seldom used for custom-built systems, but may be used for releases of commercial products to the marketplace. |

| **Field testing** |
| Field testing is needed for systems that are deployed when in use, such as a control system for fire services. The system elements that are used in deployed locations are tested in their deployed environment to ensure that they are fit for purpose and sufficiently robust, and that communications with base facilities are effective. |

| FROM UAT TO SERVICE DELIVERY |
| The importance of testing throughout the development of an IS should not be underestimated and, as The Standish Group has shown, there is a strong argument for |
getting the eventual users involved as early as possible and for keeping them engaged throughout development. Some systems are built this way, but many more are not – and however we build a system, at some point it has to be transitioned from development to use. It is at this transition that most problems with a system become visible (though the root cause may have occurred much earlier). The choice is between unearthing the problems before and during the transition or living with them after the transition. Given that, for most systems, the transition involves some kind of handover from development to users that involves money or some other form of exchange, we need a mechanism for making the transition.

Figure 1.3 shows the transition phase at the end of development. It may have several steps, but UAT will be a critical step that will enable others. The transition may take many forms in different organisations and the UAT step will have to be shaped to the overall transition process, so UAT can also take many forms.
The tested system will then move on through other steps in transition that might involve running a pilot project to ensure that the system is capable of meeting the business need on a small scale before roll-out. At some point users will need training and familiarisation with the new system. Roll-out may need to be phased to reduce the impact on existing systems and processes. There may be many factors that can affect the transition mechanism, some of which are outlined below. The choice of transition mechanism will affect the nature of UAT and the way we plan it.

**MANAGING THE TRANSITION TO LIVE USE**

**Pilot projects**

A pilot project is a scaled-down implementation, usually in a controlled environment and with specially selected staff. The aim is to demonstrate that the system is ready for implementation and to identify any problems that might affect a full-scale roll-out.

**Phased transitions**

Following a successful pilot a phased roll-out can be used to gradually increase the scale of operations. The aim in this case is to provide an opportunity to confirm what was found in the pilot and to manage any issues that arise related to scale, for example logistics.

**Training and familiarisation**

All users will need training before roll-out is complete. This may be incremental with a phased implementation. Users will also need to become familiar with the system following training so that they do not lose the knowledge and skills acquired during training and so that they are reasonably proficient when the system goes into live operation. This is a separate exercise from UAT, although UAT will certainly enhance the system knowledge of the testers. If UAT is used for training or familiarisation, the effect is not only to slow down the testing but also to compromise the quality of testing because testers not trained on the system will be likely to make errors in designing and/or executing tests.

**UAT AND CONTRACTS**

In the development of an IS we typically see requirements evolve throughout development, so that the system delivered to UAT is typically rather different from that specified in the initial requirements specification.

Where a contractual agreement exists, however, the contract is normally based on an agreed requirements specification, and that specification will have been defined at the beginning of the contract and, therefore, also at the beginning of the project. Even if some allowances are made through the contractual mechanism for changes, it is almost inevitable that the contractual requirements (on which contractual acceptance must be based) will not completely describe the evolved business and user expectations.
at the end of the project. UAT must therefore embed contract acceptance within a wider UAT exercise.

We need to make a distinction between the acquisition of an IS and the development of an IS. We speak of developing an IS when we build the system in-house using our own resources so a contractual relationship is not necessary (although some organisations may use an internal contract to enable one part of the business to buy services from another). Even though the relationship between sponsor, business managers, end-users and developers may be relatively formal, the possibility of coming to a final agreement about what is an acceptable outcome always exists. If an IS is acquired from a third party, or if a third party is instrumental in developing and delivering it, the scope for flexibility in the end result is very limited.

FOUR WAYS TO ACQUIRE SOFTWARE FOR AN IS

There are at least four different ways we can acquire software:

1. Build the system in-house (with our own resources).
2. Outsource the development to a third party (who could be offshore).
3. Acquire a commercial off-the-shelf (COTS) solution configured to our specific needs.
4. License software for our use (software as a service).

An IS may use any of these to acquire the software part of the system and this will have an effect on how UAT is performed.

**Build in-house**

Within this category of systems, requirements may be documented before development begins or they may have evolved incrementally during development. In extreme cases, requirements may never have been documented at all or not documented in any detail. In all these cases some work will need to be done to match what has been captured in requirements with current user expectations.

The main advantage of building software in-house is that the key stakeholders – developers, sponsor(s), manager(s) and end-users – should all be available for consultation.

**Outsource**

A system acquired via outsourcing development will most likely have been designed and developed in a place remote from the location where it will be used. The requirements in this case may have been written by the acquirer, the developer or in some form of collaborative effort, but the likelihood is that they will be similar to the requirements we would get from an in-house development. The development process may also be similar to that used in-house.
Where the similarity ends is in the availability of information and the accessibility of key stakeholders. Sponsor, manager(s) and end-users should all be available, but developers will be off-site, making communication about requirements, testing and other aspects of the development activity more difficult. Essentially this acquisition is similar to in-house development but communication channels will be potentially less reliable, communication lines will typically be longer and information may be harder to get and take longer to acquire.

**COTS**

COTS software covers a wide range of systems from simple productivity tools to sophisticated decision support systems. COTS solutions are already defined and built in modular form so implementation involves selecting modules, configuring individual modules, where this is feasible, and installation. In this case there is no access to the requirements from which the modules were built and the purchaser (sponsor) must make any changes necessary to make the solution work by changing business processes.

UAT for a COTS system is, in some respects, very simple because the modules will typically be in use in other organisations and will have been in use for some time. There should be very few, if any, issues with the way the software functions. On the other hand, the fit with the business processes is likely to be less than ideal and assessing how well the solution will work in the business context and how well users will be able to adjust to its approach will be much larger concerns. UAT in this case is much more about assessing a solution than about testing software.

**Software as a service**

Software as a service (SaaS) may be used to provide services that the purchasing organisation does not wish to develop or host on its own systems. The service is provided under a licensing arrangement and is usually a `standard` service that will be in use in many organisations; as a result, it should be stable and reliable in use. In some cases the service can be customised to generate changed or additional functionality to fit a particular business need.

In the case of SaaS solutions the main focus of UAT will need to be the fit of the solution to the organisation’s business processes.

Whenever we do business with a third party we enter into some form of contract and, where a contract is in place, contract acceptance will be essential to enable completion of the contract and any payment associated with that event. However, contract acceptance does not fulfil all the objectives of UAT so we will need to ensure that UAT includes an agreed contract acceptance. That does not mean that UAT will be limited solely to contract acceptance; we will still need to plan to achieve all the other outcomes that give us benefits from UAT.
STAKEHOLDERS IN UAT

There are many stakeholders in an implementation, drawn from the business, its staff and its customers, the development organisation, any suppliers and potentially a host of others. In an ideal world all these stakeholders would be identified, consulted and informed of progress at every stage. In the real world this is hardly ever the case, but we still need to take account of the expressed needs of at least some stakeholders.

For the purposes of UAT the most important stakeholder groups will be:

- the sponsor, which means the person or group that commissioned the system (and defined the business intent);
- the manager(s) who will be responsible for delivering expected business benefits from the IS implementation;
- the end-user(s) who will actually operate the system;
- developers and other technical staff who are responsible for implementation and who will support the UAT effort.

Each of these stakeholder groups has a unique viewpoint that will influence UAT and each group has a set of responsibilities. Each group will be expected to deliver certain information and services and each will have information needs to enable it to carry out its role successfully.

Sponsors

The sponsor is the person who signs the cheques. In a small company the sponsor will typically be the owner; in a larger company a sponsor will typically be an executive at some level who has a budget to acquire software for the organisation. In both cases the sponsor’s interest is in getting value for money by ensuring that the acquired software is fit for purpose, usable by the people for whom it has been acquired and meets the acceptance criteria.

The sponsor will normally initiate the risk analysis on which test prioritisation is based and subsequently be the person who makes the release decision. Unless the company is very small (sole trader or partnership), the sponsor is unlikely to be the user who will conduct the tests but may be involved in planning the UAT exercise.

The sponsor’s main concern is with achieving the expected business benefits, so their focus of attention will be on identifying potential risks and barriers to success. Following risk analysis the sponsor will be most interested in defining test scenarios that will exercise the software in a realistic environment and take some measurements that give comfort in the performance of the software and its effectiveness in driving the key business success parameters, such as revenue, profit and cost.

Business managers

Business managers are those who will be commissioning the software and delivering the expected benefits so they will need reassurance that the software is usable and
capable of achieving the claimed benefits. For this group the scope of interest is likely to be wider than simple fitness for purpose and the tests defined will need to enable the business processes designed around the software to be exercised by users in realistic scenarios. The scenarios will be based on the success criteria set by the sponsor and will ensure that users interact with the software and the business processes at a realistic level over a realistic time frame.

An example might be to carry out a few days’ transactions, using saved data from real processing using the existing system, to ensure that the results are better than those achieved when the transactions were first exercised. This kind of exercise would need the original transaction details to be saved and stored, and it would need the complete system to be set up, with users suitably prepared, to repeat the set of transactions with the new system and measure the results. This is an exercise that may go beyond UAT but will necessarily include a UAT component that can be designed to achieve the desired test coverage and measure outcomes against acceptance criteria. The overall exercise, a form of trial or pilot, can achieve effective UAT as well as a benchmark of business performance for the new system.

There are also other management roles that will be involved in the UAT exercise. One of the most important of these may be a quality manager, who would be primarily interested in the overall quality of the delivered system, or a test manager, who would be responsible for the overall quality and effectiveness of testing and would advise and guide the UAT team to enable them to achieve the best possible UAT with the resources available.

**End-users**

End-users are those who will directly interact with the system, either inside the organisation (staff) or outside the organisation (suppliers and customers). So a group of end-users is not necessarily a homogeneous group with a single set of expectations; there may be multiple viewpoints and sets of expectations to be satisfied. Since end-users directly interact with the system it is clearly appropriate for this group to carry out the actual testing at the user interface and a team made up of as many end-user viewpoints as possible should be formed for that purpose. The primary role of this team will be to assist in designing the tests and then to execute them and report on the results. A vital secondary role, however, is to raise any issues or insights that arise as they carry out the tests so that the user perception of the system is as fully explored and documented as possible.

For end-users the nature of testing will be in a practical context, so test cases should replicate realistic scenarios and use realistic data to enable the testers to operate the software as it will be utilised to achieve the IS’s overall purpose.

End-users of software may be drawn from any specialism or none. They may have lots of previous computer experience or none. Their common characteristic is that they will be expected to use the new system to achieve their own business targets. Depending on the level of previous experience with computers and especially with systems of the kind being tested, users can be encouraged to identify scenarios that will entail use of the software to achieve a particular business outcome. Those with significant experience will be able to suggest scenarios of interest in testing, such as those that will
be particularly challenging for the system, those that will be particularly prone to errors and those that will involve high levels of user interaction. These scenarios will ensure that the testing is based on real business activity, while other tests can be defined to provide adequate test coverage and ensure that all the higher risk areas of operation are adequately exercised.

While end-users are expected to make up the body of a UAT team, it is important that business managers and sponsors are aware that the UAT exercise should also involve exercising their expected interaction with the system. In a well-balanced UAT team all three roles will have some part to play directly in the UAT exercise, but even if this is not possible the dependencies between the roles need to be well understood.

**Developers and technical support staff**

Although the developers’ main work should be completed by the time UAT begins and the work of technical support will not yet have started in earnest, these two groups can make a vital contribution to successful UAT. Developers have an important role in helping UA testers to gain familiarisation with the built system and they will be assessing any incident reports raised during UAT to determine what remedial work may be required. Getting this work done promptly can have a significant impact on the time it takes to complete UAT. Technical support staff will be responsible for managing any test environments put in place for UAT and for keeping the system under test in a serviceable and usable condition. Without them the whole exercise could grind to a halt before it gets properly started.

**CHAPTER SUMMARY**

The main purpose of this chapter was to provide an overall view of what UAT is about at the broadest level. It has provided a basic understanding of what UAT is and outlined its unique role and purpose in preparation for the content covered in the rest of the book.

The definition of UAT identified the need for formal testing and the importance of acceptance criteria and of understanding the users’ real needs rather than relying on a specification written at the beginning of the development project. The basic concept of the business requirement has been introduced and this will be expanded in Chapter 2, but we need to remember that business requirements represent the reason a system is being constructed and this may be quite different to what was originally captured in the requirements specification.

We have shown where UAT fits into the overall process of implementing ISs, where UAT fits into the sequence of testing (regardless of the model the project follows), who carries out UAT and whose needs are primarily considered during UAT. We have also identified how UAT may be affected by the way a system has been built or acquired.

We have considered the evidence that ISs often fail at introduction or after change and we have examined the characteristics of ISs that make them vulnerable to potential mishaps. An analysis of the CHAOS report provides surprising and interesting information about how closely ISs reflect the original business
USER ACCEPTANCE TESTING

requirements, and important conclusions can be drawn about the value of keeping stakeholders involved throughout the development process.

The way UAT is carried out to avoid damaging failures represents UAT as a form of risk management, and UAT also has an important role to play in building user confidence, assessing the readiness of the system for deployment, and preparing for roll-out.

Finally we have identified the key stakeholders in UAT and their roles and we have outlined the main areas of cost associated with UAT.

After reading Chapter 1 you should understand the purpose and the benefits of UAT, and who is involved in UAT, and understand and justify its costs.

What have you learned?

Test your knowledge of Chapter 1 by answering the following questions. The correct answers can be found in Appendix B.

1. In what way does UAT differ from other testing carried out during development?
   A. UAT takes longer to do than other kinds of testing
   B. UAT must be carried out by end-users
   C. UAT evaluates the system against the requirements specification
   D. UAT can be informal testing as long as it is done by users

2. Which three of the following are used as the basis of UAT?
   A. Technical specifications
   B. User needs
   C. Requirements specification
   D. Acceptance criteria
   E. Business processes
   F. Tests developed for system testing
   G. Automated tests designed by the developers

3. Why is it vital for users to carry out UAT? Select three options.
   A. Because they have more time available than developers and testers
   B. Because they are not experts on the technical performance of the system
   C. Because the users’ ability to use the system as a tool to achieve business benefit is critical to success
   D. Because only users will be able to recognise some situations in which the system’s behaviour may be counterproductive
   E. Because developer testing has less value than UAT
   F. Because the system must be evaluated against user needs and use actual business processes.
Some questions to consider (our responses are in Appendix B)

1. What questions would you ask if your organisation asked you to carry out UAT on a new piece of software that is being introduced to support the sales activity in your business?

2. How would you react if your boss told you that the development project for which you will be doing UAT is running late and he wants you to do UAT in parallel with the developers completing the development?

3. Why not just get professional testers to do UAT? After all they have experience of formal testing and know all the techniques.
INDEX

acceptance
decision-making, 159–161
flowchart, 160
acceptance criteria
absence of, 163
acceptance and, 161
definition, 39, 105
evaluation, 166
examples, 39–40
identification, 101–103
prioritisation, 196
progress against, 155–156
purpose, 40
actors, 52, 53, 112
alpha testing, 24
Ariane 19, 47–48

business processes
capturing, 52–53
fitness for purpose and,
8–9, 18
prioritisation of testing, 43
testing, 8–9, 47, 60–61
business requirements
accuracy of, 18, 42
behavioural requirements, 41–42
characteristics, 49–50
compilation, 114
environmental requirements, 42
examples, 35–38, 50
functional requirements, 41
informational requirements, 41
meaning, 34, 39
prioritisation of testing, 42–43
qualitative requirements, 41–42
quantitative requirements, 41–42
review, 48–50, 107–109, 188, 194
in test design hierarchy, 122
types, 40–42
UAT and, 43–44
see also requirements specification
business vulnerability, 19–21
BVA see boundary value analysis
case study, 13–14
business requirements, 37–38, 188
incident reporting, 154
post-UAT analysis report, 174
requirements review, 109, 194
status report, 157
test conditions, 126–127
test schedule, 148–149
test scripts, 135–138
use cases, 115–119
user stories, 111–112, 194
change control, 119
CHAOS report, 7–8, 19–20, 31–32
commercial off-the-shelf (COTS) software, 28, 46, 193
communications plan, 95
component-based development, 46
costs
of failure, 21
impact of UAT on, 5
in-service, 96
of UAT, 5–6
value and, 105–106
COTS see commercial off-the-shelf coverage
determination of, 68, 165
as emergency-release criteria, 164–165
meaning, 60, 61–62, 67
progress against, 156
data input sheets, 141–142
defects
criticality, 102, 201
defect analysis, 172, 175
management, 119–120
outstanding, 40
post-implementation correction, 176
progress against, 156
testing fixes, 47, 147
in tests, 61
delays
estimating, 199
measuring, 40, 155, 156
developers, as UAT stakeholders, 10, 11, 29, 31
development
component-based, 46
deliverables, 100–101, 103
in-house, 27
independence of UAT, 43–54
iterative, 45–46
outsourced, 27–28
relationship with UAT, 43–44, 45–46
sequential, 45
edge cases, 113–114, 129–130
emergency-release criteria, 163–165
end-to-end testing, 16–17, 47, 60–61, 131
end-users
role, 75
training, 26, 82, 83–84, 96, 175, 203
as UAT stakeholders, 10, 11, 29, 30–31
entry criteria, 103–104
environmental requirements, 42
EP see equivalence partitioning
equivalence partitioning (EP), 68–69, 129
factory acceptance testing, 24
failures
CHAOS report, 7–8, 19–20, 31–32
examples, 6–8
reasons for, 19–20
familiarisation, 26
FAQs see frequently asked questions
field testing, 24
fitness for purpose, 8–9, 18
formal testing, 56–58
forming, storming, norming, performing, 86–87
frequently asked questions (FAQs), 175
frozen test basis, 48
functional requirements, 41
functional test design techniques, 58
functional testing, 58–59
functionality, 102
fundamental test process (FTP), 21–22, 62
Heathrow Terminal, 19, 20, 21
high-level plans, 106
high-level test schedules, 145–146
IM see incident management
implementation, 176
see also release; roll-out
in-house development, 27
incident management (IM), 78, 120
incidents
incident reporting, 151, 154, 196
severity, 155
information systems (ISs), 2–3
acceptance, 159–169
business vulnerability and, 19–21
characteristics, 2
development, 27
life cycle, 4–5, 93–95
software acquisition for, 27–28
testing, 4–5, 17–18
informational requirements, 41
input data, 128, 138
integration testing, 47, 57–58
International Software Testing Qualifications Board (ISTQB), 16
IS see information systems
ISTQB see International Software Testing Qualifications Board
iterative development, 45–46
lessons learned, 175
live environment data, 139
London Ambulance Service, 6–7, 83–84
managers
role, 75
as UAT stakeholders, 10, 11, 29
metrics, 155
Microsoft Xbox 360, 7
negative testing, 132–133, 197
output data, 128
outsourcing of development, 27–28
performance, 132, 139
personality
analysis, 81, 82
skill vs, 80
UAT team selection, 80, 81, 82, 192
phased transitions, 26
pilot projects, 25, 26
planning
checklist, 180–181
importance, 99–100
training, 82, 203–209
for transitions, 95
UAT, 77, 99–120
post-conditions, 64, 65, 128
post-testing summary, 156
post-UAT actions, 175–177, 184–185
post-UAT reporting, 171–175
preconditions, 64, 65, 128, 138
prioritisation
acceptance criteria, 196
business processes, 43
business requirements, 42–43
testing, 71–72, 102, 105, 145, 198
usability, 43, 46
progress
against acceptance criteria, 155–156
against coverage, 156
against defect counts, 156
against schedule, 155
qualitative requirements, 41–42
quantitative requirements, 41–42
readiness for service, 9
regression testing
meaning, 47, 68, 133
scheduling and, 133, 147
test scripts for, 67
release
checklist, 184
criteria for, 102–103, 163–165
emergency criteria for, 163–165
final decision, 169
go/no-go flowchart, 160
interested parties, 200
readiness for, 9
recommendations, 168–169
risk of, 161, 162–163, 165, 166, 201
reports
incident reports, 151, 154
post-UAT, 171–175
status reports, 156
test summary reports, 168–169, 171–175
UAT completion reports, 166, 167, 168, 173
requirements see business requirements
requirements specification (RS)
accuracy of, 42, 44, 107
contract and, 26
enhancing, 51–53
evaluation, 48–50
format, 36–37
meaning, 35, 40, 189
requirements-based test cases, 70
reviews
benefits, 73
of business requirements, 48–50, 107–109, 188, 194
checklist, 107, 108
conduct, 108
definition, 72
follow-up, 108
preparation for, 107–108
process, 72
of requirements specifications, 48–50
review team, 107, 190–191
RIAD, 78
risk
management, 6–8, 10, 17–18, 32
of release, 161, 162–163, 165, 166, 201
risk-based testing, 71–72, 129, 162
roll-out
phased, 26
strategies, 175–176
transition from development, 25
RS see requirements specification
SaaS see software as a service
scenarios
extreme, 130–131
linked, 130, 131
test cases, 130–131
use cases, 52, 53
sequential development, 45
site acceptance testing, 24
software
acquisition methods, 27–28
characteristics, 9–10
software as a service (SaaS), 28
software under test (SUT), 147
sponsors
role, 75
as UAT stakeholders, 10, 11, 29, 31
stability, 163–164
stakeholders
groups involved, 10–11, 29–31
interviewing, 110
UAT team relationship, 75–76
‘stand up’ meetings, 90
Standish Group, 7–8, 19–20, 101
status reports, 156, 157
streamlining, 148
structural testing, 59–60
system testing, 47, 48
systems, 2
task-based training, 85, 208
technical specifications, 72
test basis
developing, 48–49, 107–119
meaning, 48, 58, 189
test cases
business process-based, 71
change control, 119
design, 128–133
examples, 64
meaning, 63
perspective, 133
requirements-based, 70
scenarios, 130–131
in test design hierarchy, 122–123
test scripts and, 135
user interface-driven, 71
test conditions
examples, 63–64, 66, 124–125, 126–127
identification, 124–128
meaning, 63
risk analysis, 128
in test design hierarchy, 122–123
test data
data input sheets, 141–142
input data, 128, 138
live environment data, 139
meaning, 138–139
output data, 128
test design
checklist, 182–183
definition, 122
hierarchy, 122–123
test development process (TDP), 62–67
test environments
availability, 145–146
identification, 106
test execution
checklist, 183
process, 151
test levels, 45, 57–58
test logging
examples, 152–153
meaning, 119–120, 151, 155
test procedure specifications see test scripts
test reporting, 172
test schedules
detailed, 146–150
examples, 148–149
high-level, 145–146
implementation, 150–155
importance, 144–145
meaning, 144
prioritisation, 145, 198
progress against, 155
streamlining, 148
test scripts
allocation, 150
change control, 119
design, 133–138
examples, 65–67, 135–138, 140–141
execution, 151
format, 140–142
functional testing, 59
level of detail, 133–134
meaning, 63
sequencing, 130, 131, 196
test cases and, 135
in test design hierarchy, 122–123
test summary reports
conclusions, 168–169
content, 171–175
test-case design techniques, 68–70, 129
testing
alpha testing, 24
beta testing, 24
coverage, 60, 61–62, 67–68, 164–165
in cycles, 131
end-to-end, 16–17, 47, 60–61, 131
factory acceptance testing, 24
field testing, 24
formal testing, 56–58
functional testing, 58–59
meaning, 56–58
negative, 132–133, 197
processes, 62–67
risk-based, 71–72, 129, 162
site acceptance testing, 24
structural, 59–60
system testing, 47, 58
test types, 58–62
unit testing, 47, 57
usability testing, 43, 47, 132
white-box, 59–60
see also UAT
tests, life cycle of, 146
time
criticality of release, 102
required for testing, 199
traceability, 67, 123
training
der- user, 26, 82, 83–84, 96, 175, 203
new starters, 175
task-based, 85, 208
UAT, 82–85, 191, 202–209
training consultants, 203
transitions
phased, 26
planning for, 95
in system life cycle, 93–95
to live use, 24–26
UA testers
availability, 146
role, 78–79
UAT (user acceptance testing)
approaches, 70–72
completion reports, 166, 167, 168, 173
contracts and, 26–28
coordinators, 77–78, 79
costs, 5–6
COTS software, 28, 46, 193
definition, 15, 31
efficiency, 60–61, 123
entry criteria, 103–104
evaluation of results, 162, 165–166
as ‘event’, 96, 98
exclusions, 47, 133
formal nature, 56–58
high-level plans, 106
in-house nature, 27
independence of, 43–44
issues arising, 151, 155
meaning, 15–17
objectives, 15, 57, 103, 105
outsourced software, 27–28
perceptions, 21, 190–191
personnel see UA testers; UAT teams
planning, 77, 99–120
prioritisation, 70–71, 102, 105, 145, 198
process, 21–24, 98
progress, 155–156
project initiation checklist, 178–180
reasons for, 17–18
relationship with development, 43–44, 45–46
as risk-management tool, 7–8, 10, 17–18, 32
role, 5
scope, 46–48
software as a service, 28
stopping, 161–162
strategy see UAT strategy
timescales, 187, 199
training see UAT training as transition, 93, 96, 97
value of, 6–10
UAT strategy
defining, 103, 105–106
test schedules and, 145
UAT teams
conflict within, 88–89
formation, 85–86, 89–90, 208
life cycle, 86–88
meaning, 76–77
personality and, 80, 81, 82, 192
responsibilities, 76, 87–88
roles, 77–79
rules, 90–91
skills, 79–80
stakeholders and, 75–76
team leaders, 87–88, 91
team selection, 80–82, 187–188, 192
training, 82–85, 191, 202–209
working environment, 89–90, 192–193
working patterns, 90, 192–193
UAT training
content, 207–208
entry criteria, 206
inputs, 206–207
objectives, 203, 205
outputs, 209
planning, 82, 203–209
process, 202, 204
team formation and, 85–86, 208
timing, 205
UK Passport Agency, 6, 99–100
unit testing, 47, 57
usability
definition, 43, 164
determination, 164
prioritisation, 43, 46
usability testing, 43, 47, 132

use cases
definition, 52
definition, 52
edge cases, 113–114,
129–130
examples, 53, 112–114,
115–119
purpose, 52, 112

user acceptance testing see UAT

user expectations
capturing, 38, 51–52
stakeholder interviews, 110
testing, 46

user interfaces, test cases for,

user processes, capturing, 110

user stories
capturing, 111
test cases for,

workarounds, 40, 175

working environment, 89–90,
192–193

working patterns, 90, 192–193
USER ACCEPTANCE TESTING
A step-by-step guide
Brian Hambling and Pauline van Goethem

Every information system requires user acceptance testing before going live. This is by far the most common IT activity undertaken by non-IT staff. This book is a hands-on manual for both testers and non-specialists to plan and carry out effective acceptance tests. It gives the reader a structured, step-by-step approach and identifies ways of making the process as simple and cost-effective as possible.

The book also explains the significance of UAT as a bridge between the disciplines of testing, project management and quality management.

Contents include: the importance of UAT, business requirements, testing basics, the UAT team, preparation and planning, test design, test implementation, system evaluation, life after UAT.

- Step-by-step approach
- Invaluable for test and quality managers and anyone involved in UAT
- Pragmatic and robust guidance
- Hands-on manual for non-specialists

To date there has been very little written or provided to help UA testers. This book comfortably fills that void and helps UA testers understand their role and the many tasks they have to undertake.

Geoff Thompson,
Consultancy Director at Experimentus and Chair of the UK Testing Board

ABOUT THE AUTHORS
Brian Hambling and Pauline van Goethem have nearly 60 years’ combined experience in the IT industry in a wide variety of development, testing and project management roles. Brian has been Chair of the Software Testing Examination Board at BCS and an examiner at the International Software Testing Qualification Board (ISTQB). Pauline is a member of the ISTQB Glossary review team.

This book fills a gap in my bookshelf, a volume I wish I had years ago. This is a step-by-step guide that is clear and praiseworthy.

Peter Morgan,
Testing Practitioner

You might also be interested in:

Information Technology

Cover photo: Shutterstock, © ZGFotography

ISBN 9781780171678