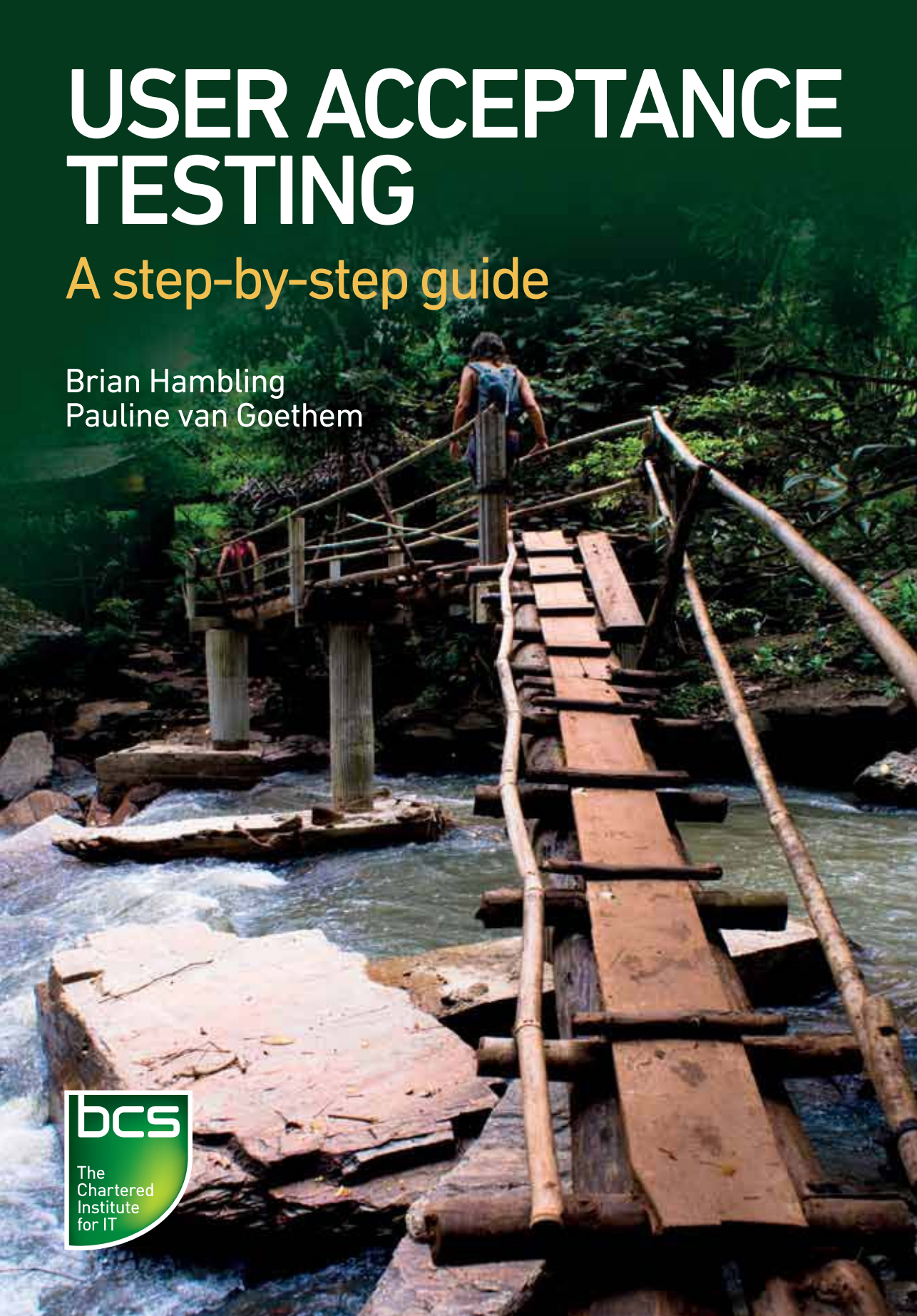
USER ACCEPTANCE TESTING

A step-by-step guide

Brian Hambling
Pauline van Goethem



bcs
The Chartered Institute for IT

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

USER ACCEPTANCE TESTING
A STEP-BY-STEP GUIDE

Brian Hambling, Pauline van Goethem

bcs
The Chartered Institute for IT

© BCS Learning & Development Ltd 2013

All rights reserved. Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted by the Copyright Designs and Patents Act 1988, no part of this publication may be reproduced, stored or transmitted in any form or by any means, except with the prior permission in writing of the publisher, or in the case of reprographic reproduction, in accordance with the terms of the licences issued by the Copyright Licensing Agency. Enquiries for permission to reproduce material outside those terms should be directed to the publisher.

All trade marks, registered names etc. acknowledged in this publication are the property of their respective owners. BCS and the BCS logo are the registered trade marks of the British Computer Society, charity number 292786 (BCS).

Published by BCS Learning and Development Ltd, a wholly owned subsidiary of BCS, The Chartered Institute for IT, First Floor, Block D, North Star House, North Star Avenue, Swindon, SN2 1FA, UK.
www.bcs.org

Paperback ISBN: 978-1-78017-167-8
PDF ISBN: 978-1-78017-168-5
ePUB ISBN: 978-1-78017-169-2
Kindle ISBN: 978-1-78017-170-8

British Cataloguing in Publication Data.
A CIP catalogue record for this book is available at the British Library.

Disclaimer:
The views expressed in this book are of the author(s) and do not necessarily reflect the views of the Institute or BCS Learning and Development Ltd except where explicitly stated as such. Although every care has been taken by the authors and BCS Learning and Development Ltd in the preparation of the publication, no warranty is given by the authors or BCS Learning and Development Ltd as publisher as to the accuracy or completeness of the information contained within it and neither the authors nor BCS Learning and Development Ltd shall be responsible or liable for any loss or damage whatsoever arising by virtue of such information or any instructions or advice contained within this publication or by any of the aforementioned.

Typeset by Lapiz Digital Services, Chennai, India.
Printed at CPI Antony Rowe Ltd, Chippenham, UK.

CONTENTS

LIST OF FIGURES AND TABLES

AUTHORS

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.

INTRODUCTION

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.

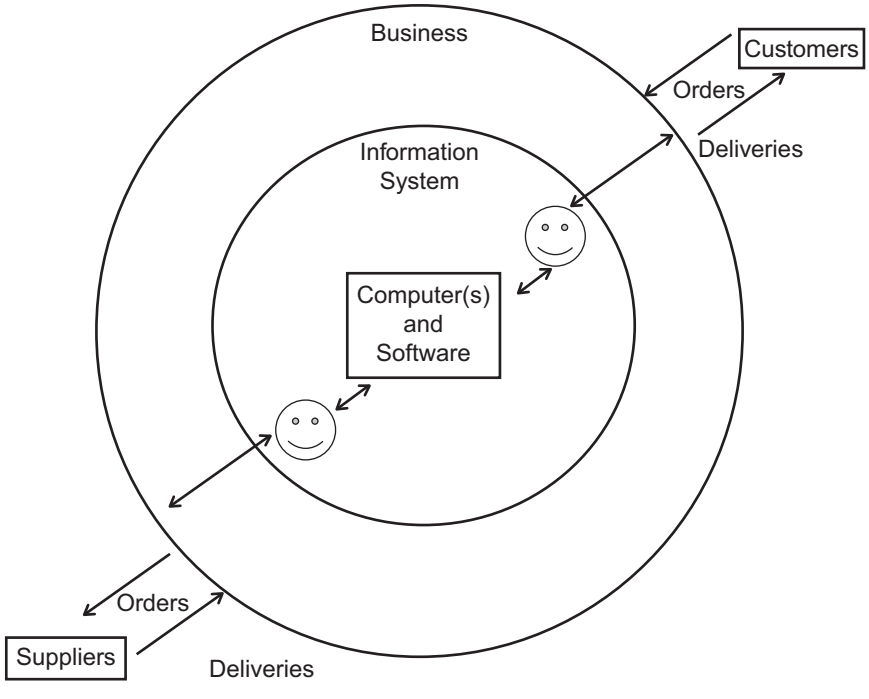
Characteristics of an IS

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.

Figure 0.1 An information system



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

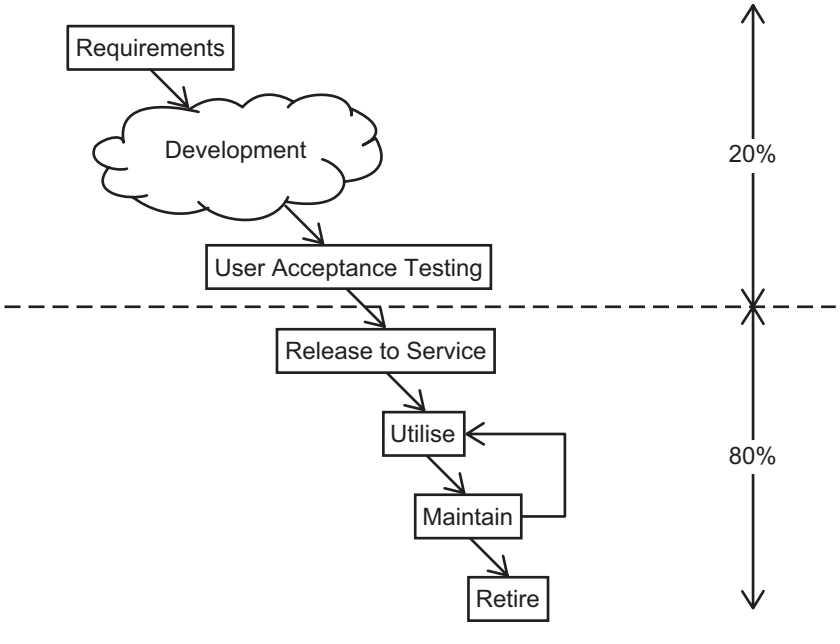
When we set up a UAT exercise we are testing a system and not just a piece of software, which means we are really interested in knowing whether the overall IS works. If the system does not work as we expect, there could be a number of reasons, and among these is the possibility that the software does 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.

Figure 0.2 Life cycle of an IS



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, anticipates your needs, 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.

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