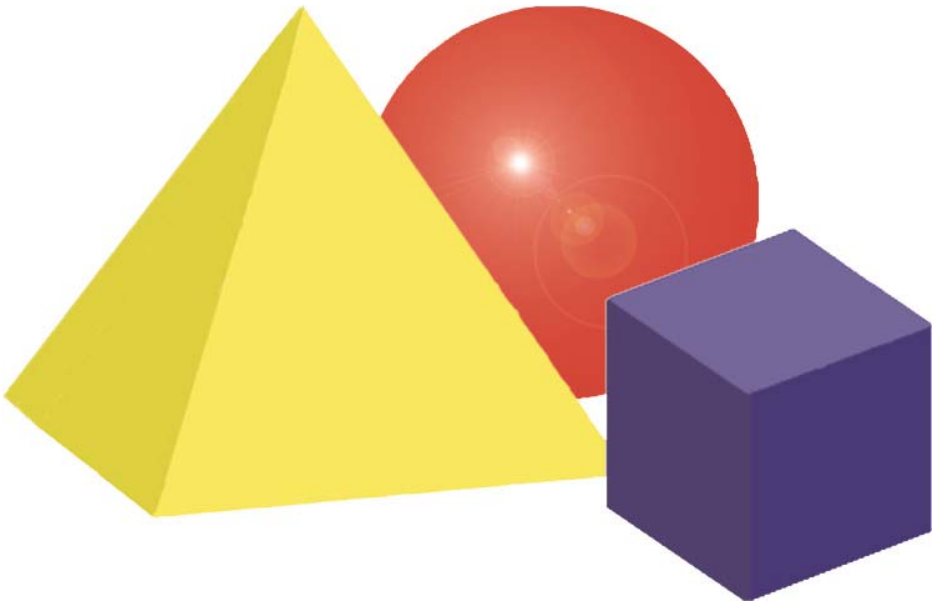
Professional TESTER

Number 18
April 2004
ISSN 1742-8742

THE MAGAZINE FOR TESTING PROFESSIONALS £4



Having problems solving the software testing puzzle?

Whether it's Weblogic or Websphere, Tibco or MQ, Oracle or DB2**, nmqa** have the skills and experience to solve your testing problems.

For the most flexible, innovative and high quality software testing solutions, think:

n m q a

intelligent testing

For more information on this and other services, please visit www.nmqa.com

11-15 betterton street
covent garden
london
WC2H 9BP

tel: 0207 470 8818

email: info@nmqa.com

intelligent testing

Doing it our way

In the last issue of* Professional Tester *I suggested that Software Testing as an activity and a profession still has a lot to prove, as evidenced by the generally poor quality of software in use. Some readers have suggested that I am blaming the wrong group and this is true in many ways. However my point was not that testers are responsible for poor products but that testing has not yet achieved its aim of solving the problems caused in development projects by others.

This is even more true when applied to the almost universally abysmal usability of public-facing web applications. In this area, testing has not only failed to provide good enough solutions: it has, so far, failed to even begin addressing the problem. In this issue we and our excellent contributors intend to offer assistance to readers to improve matters; not only web usability but appropriate user interfaces for applications of all kinds.

This kind of testing is different from the activities in which most testers have expertise and it is possible to argue that it is not testing at all, but rather *assessment*. Our regular columnist Felix Redmill made a convincing case in a recent issue *(PT 15)* that testing should include assessment but for the moment we will leave this matter to him and other thinkers on it; our contributors this time have been asked to approach user interfaces and usability from a tester's point of view, and provide practical ideas which can be applied using the skills testers already possess, and they have risen to this challenge.

An interface which is readily learnable and usable by inexperienced users, of the SUT and similar systems, will lead to fewer errors being made by all users. Just as we examine the interfaces between components carefully to prevent unexpected input to those components, we should consider the user interface as a source of software failure of all kinds.

Edward Bishop
Editor

Vocabulary for a new era

This year's new buzzwords, as heard at EuroStar and ICSTEST

IT governance	*high level management using information business can understand (ie amounts of money) and verbal instructions so that it's not necessary to understand the trivial technicalities of doing the job itself*
alignment	*extra work for testers to produce the monetary figures needed for governance (see above)*
classical testing	*old-fashioned, cumbersome techniques that are not as good as radical new ideas. Better still do them though, it's too risky not to*
experience-based testing	*guessing*
pattern-based testing	*guessing*
know-how	*not knowledge but "direct applicable problem-solving knowledge for productive tasks". What's the difference? We don't know*
collaboration	*good communication between business, development and testing. Now why didn't we think of that?*

Professional TESTER

Number 18 • April 2004

News

Conferences 4
Developments 5
MarketWatch 5

User interface testing

Making it easy 6
Edward Garson's holistic approach to good design
Usability and reusability 10
Chris Ambler discusses scripted usability tests
For the record 14
Sarah Saltzman on using automation in usability testing
Not a matter of opinion 18
Edward Bishop's suggests an objective usability testing method
Through other's eyes 30
How Tim Edmonds gained insight to accessibility issues

Articles and features

Test library 8
The quarter's new books
Venerable and vital 9
Geoff Quentin revisits the V-Model
Performance testing survey 2004 16
In preparation for the next issue, we ask for your views
Changing gear 24
Mike Lucas with a regression testing case study
Ghost in the machine 26
John Kent continues his series on test automation
Using risk as the basis of test planning . . . 28
Can some risk of failure be tolerated? Felix Redmill thinks so

The theme for the next issue, in July 2004, will be **Performance Testing**. This is clearly the fastest-growing type of testing in terms of the numbers of testers working on it, and the one best supported by automation, but there are many tales of high-value systems which passed testing and still failed under real load. As well as the results of Embarcadero's survey (see page 16 of this issue) we will have a range of articles on performance testing, from first principles to using highly advanced tools. Ideas and opinions in letter form are, as always, invited. If you would like to contibute an article please download our guidelines from ***www.professionaltester.com*** and let us know what you have in mind before starting to write.

Professional Tester *is published quarterly by Test Publishing Ltd, 73 West Road, Shoeburyness, Essex SS3 9DT. Tel 01702 294491; Fax 01702 299040. Email editor@professionaltester.com.*

News: conferences

Edward Bishop reports direct from the fifth ICSTEST, in Düsseldorf

One expects to learn of new developments and ideas at ICSTEST, but this time the most striking aspect was the emphasis placed by many of the speakers on established, "traditional" good practice: independent test teams, formal definition and early validation of requirements, frequent reviews and inspections facilitated by trained moderators, and empowering testers to influence development. So this ICSTEST was less radical than the last, but as always there were excellent talks by speakers with a great deal of experience. This time, many of them were drawn from the automotive, aerospace and space transportation industries, and the approaches they took to the special challenges faced gave food for thought on how the lessons they have learned might be applied to the business systems on which most testers, and *PT* readers, are working.

Dr Lawrence E Day of Boeing has been involved in every kind of lifecycle but explained what seemed to be a largely traditional, disciplined approach to creating a unified management structure and common development methodology, with responsibility for project success and delivery time given to small development teams (comprising development, testing and QA staff) rather than to management. Day described how he sold this idea to senior management by bringing them into inspections where they see people from the different groups achieving mutual understanding of the development process quickly, understanding and working to implement verbal directions (all of which are carefully documented) and providing the information senior management needs to work at a "governance" level.

Professor Mike Holcombe of the University of Sheffield explained his current research into new evidence on the factors affecting testers, their wellbeing, and the quality of the work they do. He quoted some frightening statistics from various sources, eg "bugs in software cost £3,250 per user annually", "23% of users have their work disrupted by software failures al least once per day", "20% of users spend at least one hour per day fixing problems caused by poor software" and "67% of software companies have purchased [testing] tools and never used them". He concluded that developing business models will affect software development and also the interface between testing and design, and that testing needs to adapt to handle change - an important factor in dynamic business – better.

5th ICS TEST®
INTERNATIONAL CONFERENCE ON SOFTWARE TESTING

The testing/business relationship theme was continued by several subsequent speakers including **Errol Rodericks** of Mercury Interactive and **Theresa Lanowitz** of Gartner, both of whom focused on "aligning" IT to the needs of the business – not by analyzing requirements to build and test systems, but by analyzing the systems against business criteria, ie money earned and saved; this, at last, threw some light upon Mercury's new product range which includes their test tools as only part of a range of monitoring and measuring solutions, the top level being a "business dashboard" containing instruments showing key business performance indicators. The central message seemed to be "IT cannot carry on as it has been, business has now recognized the importance of applications and is about to get far more involved". Depending on your point of view, there are two possible reactions to this news: (i) IT is entering a new era of maturity which will enable it to achieve the same levels of efficiency and quality as manufacturing industry; or (ii) the barbarians are at the gates. Of course, it's one thing for business to believe it can manage software development and maintenance projects better than technical personnel; proving it is another.

Test use cases are not test cases but a formalized way of describing tests by capturing interactions between the tester and the system under test. **Vincenzo Cuomo** of ST Microelectronics introduced us to the format and how it can be used as the first step in introducing UML into a test process. He finished by expressing his hope that the forthcoming UML 3.0, and UML-based modelling tools such as Rational Rose, will include more support for testing.

Dr Adam Kolawa of Parasoft was one of the most entertaining speakers because he did not shy away from controversy and obviously believes in his message - that many defects in software have a root cause in the code and can be prevented - with passion. Here are some quotes: "We believe we can test bugs out of software but this is not so, as we can learn by looking at other industries"; "unit testing is now at an automatic, white box level and should not involve any people"; "I went to an organization in India which was supposed to be at CMMI level five. They had one PC for static analysis of code for the programmers to use if they wanted to. I told them in my opinion they are at level one"; "there are two ways of implementing [code quality standards]; the fascist way and the nice way. Either way the developers will be squeaking". He was not the only speaker to advocate fully automated unit testing, but made the most convincing case for it.

We returned to an "agile" development environment with **Andreas Kornstädt** of IT-WPS, but not as we previously knew it. His approach involves giving the tests to the developers to describe the system they should build ("to avoid haggling later about the meaning of requirements"), the creation of tests by users, and the use of the open source tool *Fitnesse* which allows tests to be defined, and JUnit-style red light/green light results displayed, in simple HTML tables by users and testers with no Java knowledge. His explanation, full of interest, enthusiasm and practical information, showed how the "test first" philosophy can actually be made to work; I am certain that most of the people in the room will have downloaded and begun playing with Fitnesse on their return to their workstations.

ICSTEST always takes place in Germany and the next will be in April 2005. However some of the speakers from Düsseldorf, plus additional ones, will also be at the smaller events taking place later this year: ICSTEST-NL in Utrecht/Soestduinen in June, ICSTEST-UK in London in October, and ICSTEST-E in Bilbao in November. For full details see ***www.icstest.com.***

News: developments

Market still looks healthy • new product from Segue • success for Seapine and Compuware

Gang of 4 Centre4 Testing is a new recruiter focusing solely on and with deep knowledge of contract testing, having been formed by a team of people who have worked in the UK testing industry for many years *(see picture: Ryan Hannigan [left], Michelle Richardson, Josephine Beavitt and Tony Wells)*, which has already enjoyed early success by winning preferred supplier agreements with several blue-chip organizations. Director Ryan Hannigan is looking for experienced recruitment consultants to join the team, who will also be using their expertise to provide the latest information for our new regular feature *MarketWatch*, below.

www.centre4testing.com • 0870 850 3434

Making the big things happen Major testing consultancy Tescom has confirmed that Neil Goodall is its new European managing director. He was formerly banking programme director for Post Office Ltd and took responsiblity for what was possibly 2003's largest and most successful project, introducing online banking services into 17,500 Post Office retail outlets across the UK. Neil will be sharing some lessons learned from this project with *PT* readers in the next issue.

www.tescom-intl.com
+ 44 (0)207 022 6700

Knitting with silk Segue Software's *Silk* test management, defect tracking, functional test execution, load testing and performance monitoring tools have been integrated with a new centralized control, repository and reporting interface, *SilkCentral*. The modules can be purchased and installed individually and the central component can also collect metrics from non-Segue products.

www.segue.com • +44 (0)1189 657721

Are you watching, Gervais? Seapine Software has won the 2004 Jolt Productivity award for project management with its defect tracker *TestTrack Pro*, and its automated test execution tool *QA Wizard* took the testing tools award. The Jolts are awarded by the US magazine *Software Development*.

www.seapine.co.uk • +44 (0) 1344 297613

Company's three Compuware's internal quality assurance organization has achieved SW-CMM level three certification. The TIC is responsible for testing integration between Compuware products.

www.compuware.co.uk • +44 (0)1753 444 444

Please send press releases, news etc to press@professionaltester.co

MarketWatch with Centre4 Testing

The contract software testing market is perhaps the earliest indicator of trends for the rest of our industry. Every quarter we ask our customers key questions designed to tap their knowledge and experience and improve our view of what is happening and may happen.

Market sentiment

Compared to last quarter, 87% of contractors and test managers feel more confident about prospects for the sector.

Whilst there is overwhelming confidence in the sector today, it would seem that there are distinct pockets of activity and there are a number of high profile projects recruiting large teams of testers. This may have been caused by the downsizing trend of recent years creating a need for teams to be built from scratch rather than backfilling roles ad hoc.

Whereas in the past many contractors enjoyed contracts that extended so often that they appeared to work with companies for longer periods than their permanent staff counterparts, the trend today is to hire contractors for distinct time slices.

Jobs advertised

Contractors tell us that jobserve.com™ is currently their favourite source of vacancies. 1,410 contract vacancies were advertised in the week prior to 23rd April 2004. Of course, the same roles may have been advertised by multiple recruiters.

Rate of pay

Following the laws of economics, as demand for contractors rises slowly, so do remuneration rates. There's an inevitable time lag between the shift in client's preconceptions, where they have grown accustomed to a highly competitive market, and contractors for whom the telephone is perhaps ringing a little more than of late.

In the late nineties, as we embarked upon year 2000 projects, clients adopted a stance towards day rates rather than hourly rates which had previously been the norm. With a "professional day" typically defined as being 8 hours, overtime is paid only in exceptional circumstances, enabling clients to budget more accurately and avoid project cost creep.

Our methodology for reporting rates is based upon the amount actually paid to contractors. This removes the discrepancy of a wide range of margins applied by recruitment agencies and consultancies who then sell on the contractor to the end client.

Test analyst:	£268 per day
Test manager:	£384 per day

These same test analysts seeking new contracts today are looking for upwards of £280 per day, an increase of 4.5% plus.

Market skills

The top three skills in demand by our clients today are:

1. J2EE
2. Oracle
3. retail banking experience

Next time we'll include a special focus on the banking and finance sector. If you have ideas for the questions we should ask or measurements we should make, or would like to be included in our polls, please contact us at marketwatch@centre4testing.com.

Making it easy

Edward Garson, senior consultant for Dunstan Thomas Consulting, opens our look at user interface testing

Traditional defect testing involves mastering a range of different types of testing and being assiduous in the pursuit of excellence. The best testers tend to be exceedingly detail oriented and able consistently to find, exactingly reproduce and communicate defects back to stakeholders. It is a distinctly rigorous discipline. There is usually no question as to whether exhibited behaviour constitutes a defect case or not.

On the other hand, usability issues are a totally different beast. Usability transgressions usually go wholly unnoticed by rafts of testers who are understandably focused on the functional behaviour of software.

IT professionals who take a conscious decision to challenge elements of the user interface of a system have the opportunity to excel through the provision of value over and above that which is expected of traditional development roles. Generally speaking, usability is misunderstood and applied by the development community at large. The opportunity to improve systems in a very tangible fashion exists and can feel very rewarding. Furthermore, most development shops do not have a usability expert on hand and generally do not place much emphasis on this aspect of product development.

Armed with some basic principles and the right tools, savvy and self-motivated IT professionals can make a big impact on the quality of the systems they work on by challenging elements of the user interface. But user interface design is as much an art as it is a science: it has the potential to be extremely contentious.

Usability testing

The amount of time and effort spent testing user interfaces will naturally be a function of the perceived business value of a system. Usability experts recommend allocating 10 per cent of the overall project budget to usability. However this is unrealistic for most organisations.

True usability testing is the domain of usability specialists. Bona fide testing on large projects involves setting up a dedicated usability lab where session participants are videotaped and may be observed through one-way glass. Participants are given a prescribed number of tasks to perform. The manner in which participants go about completing tasks – such as the navigation choices they make – reveals important information about the usability of various presentation components. Participants are encouraged to verbalise their thought processes to yield further insights during the evaluation.

In his classic tome on usability, *Usability Engineering,* Jakob Nielsen says "There are several methodological pitfalls in usability testing…" He acknowledges that there are problems inherent to "real" usability testing. The factors that contribute to this include disparities in the individual skills of test participants and large margins of error attributed to interpreting test results. As he puts it, "For usability engineering purposes, one often needs to make decisions on the basis of fairly unreliable data…"

The net result is that "real" usability testing should be a verifying and tweaking process when viewed in the context of a holistic approach to software development. This is because instituting changes at this point that are any more significant than tweaking can be a very expensive prospect. Usability testing should be an integral part of the development process, irrespective of the methodology being used.

Testing user interfaces - either casually or seriously as described - demands knowledge of how to design them in the first instance. Understanding this process is integral to being a good UI designer.

Use cases: an invaluable foundation

A very close relationship exists between use cases and user interfaces. It should therefore come as no surprise that use cases contribute to the design of user interfaces. A use case is a sequence of actions a system performs that yields an observable result of value to a user. Use cases describe goals of the system in a story-like fashion between the user and the system.

They stipulate the bare essentials in terms of data and interactions necessary to achieve the functionality described by the use case. In doing so, they facilitate deriving simple user interfaces.

More discrete components within a system may be considered after having settled on the basic user interface. These components are born from requirements elicited during the analysis phase. The requirements are often grouped together into cohesive and meaningful units known as use cases. Use cases provide an invaluable foundation for arriving at the right user interface.

The use case diagram in figure 1 contains two use cases, *Purchase item* and *Browse catalog*. Use case diagrams provide an overview of the system as a whole, while individual use cases describe coarse-grained functionality.

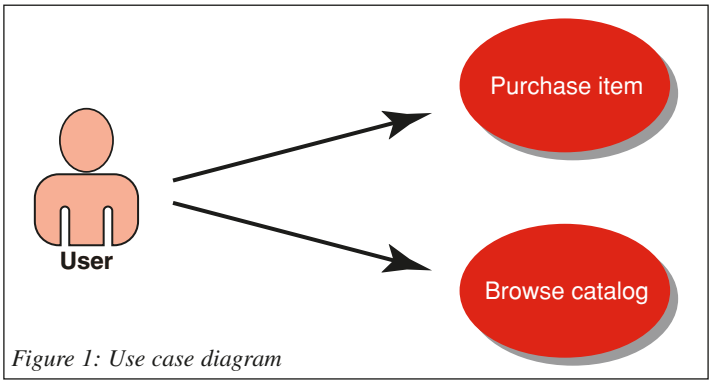


Figure 1: Use case diagram

When designing smart-client user interfaces (ie desktop applications), this broad view of the system is important in considering the foundational, application-level user interface. There exist only a few application-level user interfaces deemed familiar to users on the Windows platform. The vast majority of business-oriented, OLAP-style applications fall into one of these categories. They are:

1 Windows Explorer style user interfaces, with a hierarchical tree-view on the left and context-sensitive user interface elements on the right

2 Outlook-style user interfaces with a number of different "modes" and a task-centric philosophy

3 Multiple document interface (MDI) style applications with a parent window containing one or more child windows (considered a power user's interface)

4 Microsoft Excel-style user interfaces with separate individual tab sheets

The first step in designing or evaluating a user interface is to first consider the big picture. What foundational user interface style is best suited to meet the requirements of the use cases when considered as a whole? Choose one of the above, unless special circumstances dictate an alternative. Of course this does not apply to the design of a thin-client user interface.

Use cases stipulate the bare essentials in terms of data and interactions necessary to achieve the functionality described by the use case. In doing so, they facilitate deriving simple user interfaces.

Use case driven user interface design

Use cases are ideally generic with respect to implementation. There is ideally no indication of how functionality is rendered (eg by a web browser, handheld device or thick client application - or indeed, all three).

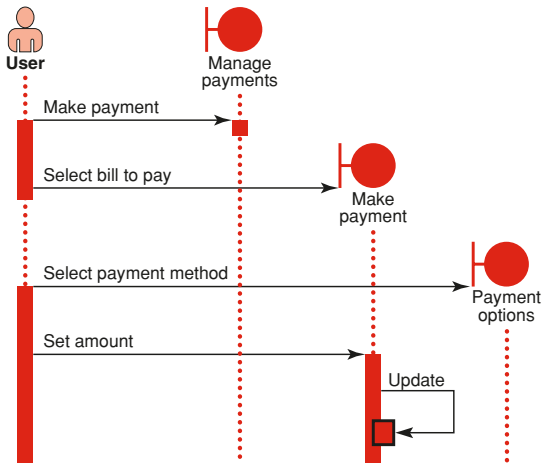


Figure 2: Sequence diagram (showing only boundary classes)

Each use case is supported by a use case document which describes the interactions between the user and the system in an implementation-agnostic manner. The use case should be focused solely on the minimum information required at each step and the workflow, which in the context of user interface design may be viewed as a kind of constraint.

The notion of generic use cases becomes important when testing user interfaces, because the use case is the definitive source of "the minimum of information" required to achieve functionality. Good UIs at a minimum require no more information than that outlined by the use case. Use cases should be completely focused on what is required of the user and the system, not how it will be achieved.

A major component of testing user interfaces involves verifying that how the user interface renders functionality marries up with what is required by the use case. Although this may seem obvious, a surprising number of flawed user interfaces introduce extraneous steps or implicitly require more data than is required to achieve functionality. These transgressions should be caught first, before concentrating on lower-level details such as the layout of discrete presentation components.

Designing good user interfaces is all about facilitating how the user specifies what is required by a use case in as straightforward and task-oriented manner as possible. The steps required to achieve this will partially depend on the device rendering the user interface. This is where use case realisation comes into play.

Use case realisation

Suppose both a web browser and handheld device interface were required of the Purchase item use case. There will of course be significant differences in the user interface between the two implementations. However, the basic use case remains the same in that the same basic data and sequence is still required.

A use case realisation is the implementation of a use case in a specific instance. They exist to support the design process of internal software components, their collaborations and most importantly the user interface. Use case realisations play a pivotal role in the initial design and prototyping of user interfaces.

Use case realisations are developed into one or more sequence diagrams, each representing a scenario through the use case. Sequence diagrams visually depict the interactions that take place between the user and the system in the course of realising functionality for a discrete scenario.

These diagrams are a great starting point for determining elements of the user interface. Decisions such as how to segregate data, the number of presentation elements required and the order in which they are shown starts to become clear during this activity.

Inductive user interfaces

Inductive user interface design has been well received in the usability community. Inductive user interfaces leave users in no doubt as to what they should do at any point in the system. These UIs are focused and have clear and purposeful intent. They are distinctly uncluttered and guide users toward accomplishing tasks. Contrast these user interfaces which expect some knowledge on the part of the user about how to use the system, with inductive user interfaces, which in spirit do not. Deriving inductive user interface designs can be facilitated by following the design steps previously discussed, especially with respect to navigation.

The prototyping phase should reveal hard data about the decisions we made during the design phase such as how usable the proposed navigational structure truly is. This begs the question of how best to prototype user interfaces in support of this activity.

Paper prototyping: more than meets the eye

Paper prototyping is overall the single most effective tool for prototyping user interfaces. Paper prototyping is far more sophisticated than most people realise and should absolutely not be dismissed due to its apparent simplicity.

Benefits of paper prototyping include the fact that it is extremely cheap and easy to learn. It is possible to create and test more user interfaces faster than with any other technique. Paper prototyping also mitigates the apprehension that some users feel during "real" usability testing: some users subconsciously feel that they are in fact being tested, not the user interface. They feel "stupid" if they are unable to complete a task that is attributable to a poor user interface; this can skew test results. Playing computer with mock-up screens on paper reduces these apprehensions and better data is obtained.

Paper prototyping can also be turned on its head. Printouts (or paper mockups) of existing user interface elements can be made in order to test them objectively against proposed improvements or alternative designs. What is learned from these user interface tests can be used to lend credibility to the case for improving them. In this manner, 'real' user interfaces get tested against mockups on a level playing field.

It is important to prototype user interfaces 'early and often'. Several iterations should be performed to get the user interface right. Testing it with a number of different stakeholders is critical to ensure that the user interface meets the requirements of disparate user types.

Micro usability

The following fundamental principles outline the basic characteristics of excellent user interfaces. They can be used as a basic guide to actively improve the usability of user interfaces in conjunction with other resources.

Simplicity

Simplicity is the foundation of all great user interfaces. Efforts to reduce complexity or ambiguity are always good. Great user interfaces reduce complexity and guide users

toward accomplishing difficult tasks. A good example of this is the Rules Wizard in Microsoft Outlook.

Platform standards

Rich-client applications (as opposed to thin-client web interfaces) should follow the user interface design guidelines for the platform for which they are intended. For Windows applications, the bible is the *Official Guidelines for User Interface Developers and Designers*. Equivalent documentation exists for the Macintosh platform.

The fundamental principle is that applications should fit seamlessly into the environment in which they live. An application running on the Windows platform should feel like any other standard Windows application.

When testing for platform standards, rigorously apply the guidelines to the user interface. Flag any deviations as usability transgressions and cite the appropriate documentation.

Consistency

Consistency is a very important element in user interface design. It is better for related things to work in the same way throughout the user interface than to be inconsistent, even if it is slightly flawed. Users learning systems tend to try doing the same things to invoke similar functionality: have them succeed.

Education

User interfaces sometimes disable widgets that invoke actions that cannot be performed under certain conditions. The problem with this is that it may not be evident to the user how to change the state of the application in order to be able to invoke the desired action. It is sometimes appropriate to enable a widget that invokes an action that is inappropriate in a given context, so long as an informative message appears when the user does so. This improves the 'learnability' of the application.

This problem is hidden by the fact that the people who develop the system are expert in it and so they do not view this as a bug; in fact, they usually don't notice this at all. The ability to put oneself in the mindset of a first-time user is required to improve the UI from this perspective, which is not always easy.

Informative error messages

Many software professionals are satisfied when an error message is correctly generated when testing error conditions. However, good error messages are the exception rather than the rule.

All error messages should be composed of an informative reason why the error occurred in very simple terms, and it is often helpful to include a suggestion as to how the error may be rectified.

It is interesting to note that error messages are almost invariably authored by the same person who wrote the code that generates them in the first place. Larger software projects should have a single person responsibile for writing all error messages, to achieve consistency across the system.

When testing the usability of error messages, play the role of a first-time user. Does the error message make it absolutely clear what went wrong? Would you know precisely what to do to achieve your original intent?

Conclusion

The best way to learn about user interface design is to study good user interface designs and read up on the subject. There exists a vibrant and active usability community that readily share and disseminate their expertise on the web. It is possible to glean a lot of useful information this way. They put a lot of hard work into discerning what works and what doesn't. This information is shared with the community: it is there for the taking. The rigorous approach described in this article should yield a high degree of confidence that your user interface will be successful. This is because first and foremost it will meet requirements and hopefully do so in as straightforward a manner as possible. PT

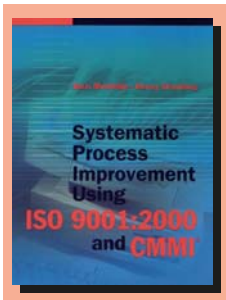
Test library

This quarter's new books about testing or of interest to testers

The Capability Maturity Model
Carnegie Mellon University
Software Engineering Institute
Addison-Wesley, ISBN 0-201-54665-7

Excellent descriptions of performing assessments, identifying opportunities, progressing improvement and what process maturity means to both customer and supplier make this an essential reference. Published before CMMI so covers only CMM v1.1.

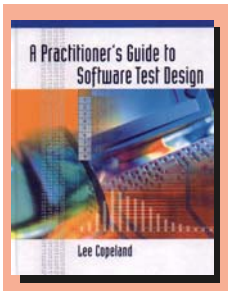
Relevance to testing: high, but generic



Systematic Process Improvement Using ISO 9001:2000 and CMMI
Boris Mutafelija and Harvey Stromberg
Artech House, ISBN 1-58053-487-2

Brings together the various models with very helpful emphasis on the synergy between them, getting roles and responsibilities well defined, and the need for full and continuous management support.

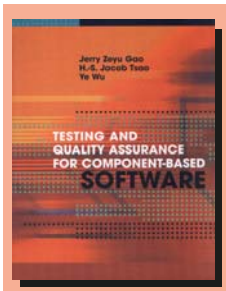
Relevance to testing: high, but indirect. No advice specifically for testers



A Practitioner's Guide to Software Test Design
Lee Copeland
Artech House, ISBN 1-58053-791-X

A very practical book of applicable techniques from one of testing's best-known names. Readable, friendly style, clear layout, a few jokes, and useful practice exercises. Recommended.

Relevance to testing: very high



Testing and Quality Assurance for Compenent-Based Software
Jerry Zeyu Gao et al
Artech House, ISBN 1-58053-480-5

A general testing textbook, but with examples and explanations drawn from component-reuse setups, eg EJB, COM+, CORBA etc. This doesn't cause much change to the basic testing theory which comprises much of the content.

Relevance to testing: very high

Venerable and vital

MD of QBIT and founder chairman of the BCS SIGIST Geoff Quentin explains the origins and revisits the importance of the model on which virtually all software testing is based

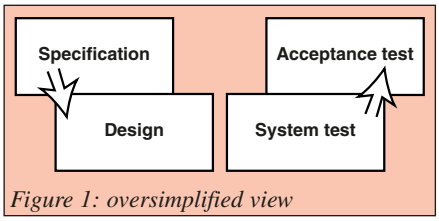


Figure 1: oversimplified view

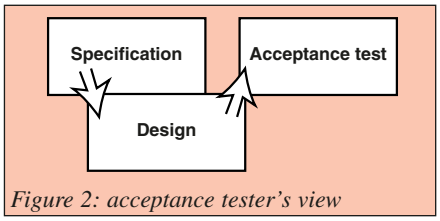


Figure 2: acceptance tester's view

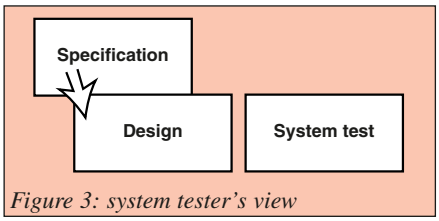


Figure 3: system tester's view

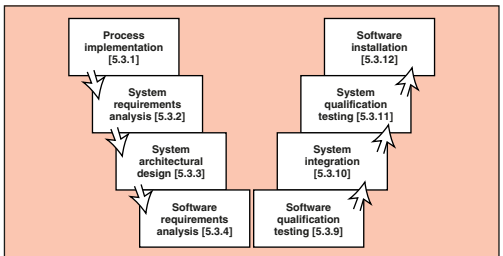


Figure 4: system tester's view, based on ISO 12207

The original V-Model was called the U-Model – and the U was on its side. It can be found in the book *Testing in Software Development (Ould and Unwin, CUP, 1986).*

As a trainer I found this difficult to draw on a white board so I drew it upright as a V and used this to introduce software testing to thousands in the UK, Asia and Australia. The diagram was the basis of much of my teaching and by the end of a three-day course my drawing of it would look like a Jackson Pollock painting.

The V-Model is now widespread and even found in ISO 12207 *System Development Processes* to illustrate the way later testing activity should be developed from the earlier requirements-gathering activities. Not all testers like the V-Model and some popular presentations and books are openly disdainful of it. This is a strange opinion and can be safely ignored by any organisation that aspires to process maturity above level one.

Care is needed with the V-Model as it is often presented in far too simplistic a fashion, diminishing its value. Figure 1 shows a poor version easily discredited by detractors and promoters of other models. First it is essential to separate the views of the acquirer and the developer and view the work from each in turn. Thus the view of the acceptance tester is as shown in figure 2. This can now be balanced by the view of the system tester which is shown in figure 3. The V-Model now makes sense to both the acquirer (user) and developer and both the acceptance and system testers draw test material such as test plans and test scripts from the specification and design documents.

The project manager has the job of deciding the degree of overlap between the testing work done at system level and that done at acceptance. In an ideal world the acceptance testers may simply concentrate on ensuring that the delivered system integrates properly within all the existing systems and rely on the system testers to have tested all the specified functionality.

The system tester view of the V-Model based on the activities of ISO 12207 is in figure 4. The references in the form [n.n.n] can be used to find the activities shown described in the standard.

It will be seen that system integration is not a testing activity as such and this is very sensible and practical. It will also be seen that the system tester may draw on information from many documents created during the early stages of the development process and this is essential for success. Finally it will be apparent that the system tester expects the software to have been checked before delivery to system test. The empirical testing within ISO 12207 is referred to as qualification testing indicating that the system needs to be qualified before migration to the next activity.

The conclusion to be drawn from all this is that the system tester needs to be present at reviews of requirements and design documents throughout the development process and that the strategy for the system test should be based on the way that management have established the project at process implementation and documented it in the Project Initiation Document (PID). This is all very sound and accepted best practice. The V-Model is a mature and sensible model for all involved in the qualification and acceptance testing of both software and systems with a software component. PT

Usability and Reusability

Chris Ambler, Head of Testing Architecture and Strategies at Newell & Budge, on usability testing with scripts

In today's fast moving technological climate, it is safe to say that the only constant is change. Technology moves nearly as fast as people's ideas and new ways of doing things are created every day. In the software world, this means that Graphical User Interface (GUI) front ends are always under review and change. A changing GUI can have a massive impact on the way the business is run. Users that cannot use their 'tools' will find other more efficient (but less effective) ways of carrying out their daily tasks. As testers, this causes a number of problems when it comes to writing re-usable usability scripts. A balance must be achieved between 're-inventing the wheel' and utilising old assets every time there are changes to the front end. During development, the main goal of a usability test is usually 'diagnostic' and used to find out what is performing correctly and what is not working well, so that development can continue with what is working and the developers and system testers can fix what needs more work. The earlier this is done and the more iteratively, the better.

Once development and system testing is complete, it can be passed to the users/testers. The most important thing to remember at this stage is usability testing is not designed for functional diagnostic testing, it is more of a verification process to ensure that users can complete a task successfully and it is fast enough to satisfy them, the paths they take are perceived to be efficient enough for them and that they do not have any problems or get confused anywhere. It ensures that the application or system ergonomic qualities and overall end–to–end interaction is satisfactory and follow the Jacob Nielson usability heuristics. These heuristics are:

1 **Visibility of system status:** The system must always keep the user informed about what is going on. This is done by feedback within reasonable period of time. From a testing point of view, it is necessary to understand, define and quantify 'reasonable'.

2 **Match between the system and the real world:** The system must speak the users' language, using familiar words, phrases and concepts. It must avoid using system-oriented 'jargon'. It is necessary to follow real-world conventions as much as possible, making information appear in a natural and logical order.

3 **User control and freedom:** It is a fact of life that users will often make mistakes. This creates the necessity for needing a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended procedure. It is important that the application supports 'undo' and 'redo' facilities.

4 **Consistency and standards:** Users should not have to wonder whether different words, situations, or actions mean the same thing. The specified platform conventions need to be followed, allowing a user to be comfortable with the 'look and feel' of the system.

5 **Error prevention:** Even better than good error messages is a careful design, which prevents a problem from occurring in the first place. When error messages occur, they need to be necessary, or alternatively be avoided by better design.

6 **Recognition rather than recall:** Objects, actions, and options need to be visible. The user should not have to remember information from one part of the dialogue or process to another. Instructions for use of the system should be visible or easily retrievable whenever the user requires them.

7 **Flexibility and efficiency of use:** The system needs to be as flexible as possible for differing levels of users. It is sometimes useful if experienced users can tailor frequent actions. Accelerators, unseen by the novice user, may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users.

8 **Aesthetic and minimalist design:** Dialogues should not contain information, which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility to the user.

9 **Help users recognise, diagnose, and recover from errors:** Error messages should be expressed in plain language and contain no codes (unless as part of an error message that may help the troubleshooting process). They must also precisely indicate the problem, and constructively suggest a solution.

10 **Help and documentation:** Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any information should be easy to search, focused on the user's task. It must list concrete steps to be carried out, and not be too large.

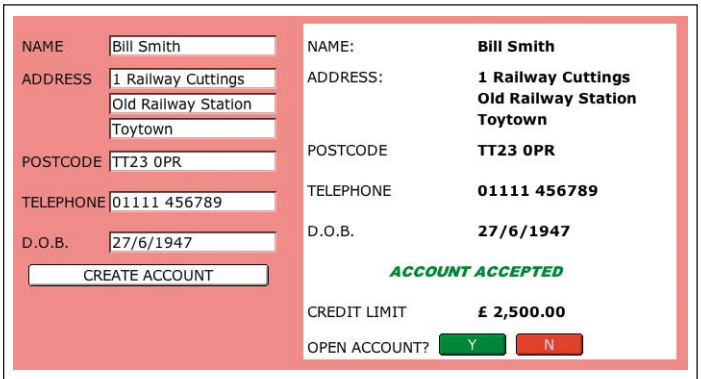


Figure 1: Account creation dialog

Let's look at a very simple example. A sales system has a dialog for creating a customer account (figure 1).

A system test for this screen would go into the detail of each input field size, check the

formats, validate the credit checking interface, check the screen outputs etc. and would need a very detailed, step by step system test script with detailed expected results. Any changes that are made to the screen will need detailed changes to the test script and test data. This will have major impact on configuration management and version control.

As usability testing is defined at a higher level, a process scenario can be created and each of the usability heuristics can be assessed during the process. This usability test script would look like the one shown in figure 2.

This test has raised a number of 'potential' defects. These defects may or may not be important to the users, given the context in which they are working. I am sure every reader will come up with a different set of 'issues' with this example!

Usability testing needs to be more 'subjective' rather than 'objective'. This creates the need for the tester to understand the process under test and be able to improvise the additions to the screens (as long as they are documented). The major changes will occur in the test data sets to ensure repeatability and again, the experienced user/tester needs to control these and document all changes.

If a change was made to the account creation screen, for example an 'ANNUAL SALARY' field was added, this would not change the usability test script as the scenario described has not changed. The only change would be to add a test data item to the test data set that would allow the user/tester to enter an annual salary. This is only an extra column to a test data spreadsheet or test data list and can be controlled easily.

It is impossible to completely 'future proof' usability test scripts because there may be more complex changes to the screen or changes to the process itself. The best you could ever hope for is to design them in a way that allows changes to be made easily. This is achievable as long as the key rules are remembered:

- usability testing is not about proving functionality: it's about verifying tasks
- iterative testing is best – knowledge of the business processes is key
- test scripts should be built around scenarios
- communication between users, testers and developers is paramount
- the secret is not to have detailed scripts
- test data needs to be updated and managed

Developers have been writing code and testing it against scenarios for years, so it is a tried and trusted technique and if the relationship is good enough why not communicate with the developers and look at linking the development changes to the usability test changes? Working with the developers and the users on the required changes to the GUI allows the tester to both understand the changes and inform the users (and developers) what the impact of those changes will be. It is possible to link these changes to business risk and business priorities, but that is another story… PT

USABILITY TEST SCRIPT – CREATE ACCOUNT **TESTER:** _____

Process Scenario:

The salesperson enters the customer details on the left hand side of the screen. On completion of the details, the 'CREATE ACCOUNT' button is clicked. The system then populates the box on the right, carries out a credit check on the applicant and accepts or rejects the applicant. If the 'ACCOUNT ACCEPTED' is shown, a credit limit is displayed and the 'OPEN ACCOUNT' 'Y' and 'N' buttons are available. If 'ACCOUNT REJECTED' is shown, then no credit limit is displayed and the 'OPEN ACCOUNT' 'Y' button is greyed with only the 'N' available. The salesperson then clicks the relevant 'Y' or 'N' button

Test Scenario:_____ Test Data Set: TDS001 Application Version: v2.01.a Environment: Test07

Expected Result : Successful account creation

Heuristic	Comments	Pass/Fail
1 Visibility of system status	There is no visible means of showing the system status other than the usual 'timer'	FAIL
2 Match between the system and the real world	All the terms used are consistent and have real world meanings (although D.O.B. could be open to misinterpretation)	PASS
3 User control and freedom	There is no obvious 'emergency exit'. There is no way to exit the screen until the process is complete, either successfully or unsuccessfully.	FAIL
4 Consistency and standards	Consistency with the rest of the application is acceptable	PASS
5 Error prevention	There is no error prevention available to this screen	FAIL
6 Recognition rather than recall	Recognition and recall is acceptable	PASS
7 Flexibility and efficiency of use	The screen is flexible and efficient in accordance with its use	PASS
8 Aesthetic and minimalist design	The screen is not 'cluttered' and all information shown is pertinent to the process under test	PASS
9 Help users recognise, diagnose, and recover from errors	Errors can only be identified at the end of the process at 'ACCOUNT REJECTED'	FAIL
10 Help and documentation	There is no help documentation	FAIL

Figure 2: Usability test script

e-testing® consultancy

Mercury Tool Specialists

e-testing is an independent specialist IT consultancy with a total focus in software testing - consultancy, resourcing, training and offshore managed services.

We are currently hiring Mercury Tool Specialists to work with clients across Financial Services, Telecommunications, Banking and Retail Sectors.

e-testing Consultancy can provide you with the opportunity to build a stimulating career working on different platforms, technologies and exciting software projects.

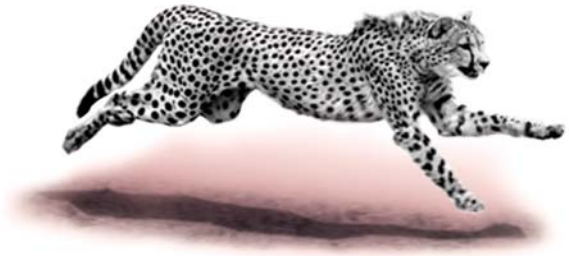
To apply, email your CV to our resourcing team at mercury@etesting.com or complete the on-line registration form by visiting our website www.etesting.com

e-testing® Consultancy
Kinetic Centre,
Theobald Street
Borehamwood,
Hertfordshire, WD6 4PJ

Tel: +44 (0) 20 8387 1701
Fax: +44 (0) 20 8387 1706
Web: www.etesting.com

CONSULTING | RESOURCING | TRAINING | OFFSHORE

StarBase Incisive solutions with integrity



Automated Testing. Are you maximising the benefits?

CHANNEL ALLIANCE PARTNER
MERCURY INTERACTIVE

StarBase are a Mercury Interactive top tier business partner

StarBase's comprehensive suite of solutions support Mercury Interactive's ever expanding range of Automated Testing and Application Management tools. Our Consulting services include:

Load/Performance Testing

"The Combination of StarBase testing expertise and CIMA application design knowledge assured a successful project"
Guy Gaskins, Chartered Institute of Management Accountants

Automated Functional Testing

"Within Functional Testing, StarBase have proved themselves to be much more than a service provider, they have evolved with us to be a valuable extension of our IT department".
Matt Cowdrey, AON

StarBase's unique Advanced LoadRunner Training

"Very enjoyable course with useful methodologies. We will be adopting a number of the suggested working practices"
Richard Green, HSBC Bank

To maximise the benefits from your Testing requirements call 0208 905 1120 or visit www.starbase.co.uk

For the record

Sarah Salzman, technology support manager for Compuware UK and Ireland, explains her view of usability testing and how how test automation tools can help with it

As testers we know why quality assurance is important. We understand that coding errors or glitches in the way one component interacts with another can bring down an entire application and result in an organisation losing thousands of pounds. Therefore it's not surprising that a great deal of testing effort is put into making sure applications are reliable, robust and will perform well under strain. However it shouldn't be forgotten that even the most resilient application, containing code written using best practices, could be perceived as failing if it is not designed in the way it works for the users. At the end of the day what's the point in building an application if people find it too difficult to work with? Most organisations understand this. In fact ease of use is often the first thing on a customer requirements list; however it's also a factor that is often neglected during the development cycle.

This is where usability testing can help, because it ensures applications are tested with the user in mind. Essentially, it is about looking at software design and trying to ensure applications are designed in a way that makes them easy to learn and use. This might seem like an obvious thing to do, but many development projects do not include any usability testing. This is understandable as both development and testing teams are under more pressure than ever to go live with applications, however if businesses want to ensure that they see the benefits of investing in and developing new applications; this is an area they simply cannot overlook.

If you need convincing on usability testing then simply consider the fact that we all come in different shapes and sizes. Our physical and psychological make up will impact the way we use a computer application. In some cases, culture and background may also be influences to consider. Maybe, most importantly, the experience of end users using an application may vary dramatically and this will without doubt impact how easy or difficult they find it to navigate and use. For example, an experienced computer user may decide to tab through the different fields in a CRM system, whereas a novice may choose to use the mouse. It is these types of differences in human to computer interaction which can be captured and analysed in usability testing. Subsequently trends can be identified in relation to how people want to interact with the application and these trends can be then fed back to the development teams who are designing the user interface.

How do you carry out usability testing?

Opinions may vary on how to carry out usability testing, but to me the starting point should always be to find a group of users who are inexperienced with the application. At the very least you should bring in users who have the same experience levels of those people that will be using the application in the production environment.

You should also try to ensure the user group reflects the ultimate user base of the application in relation to factors such as sex, age, and background. However, in many cases it may be difficult to predict the attributes of the end users of the application being developed and therefore in this situation it is important to use a diverse range of users in the usability testing process.

Once you have your sample user base, you should analyse the functional specification for the application and pick a range of standard tasks end users will be carrying out in the application on a regular basis. If you use manual testing methods it is at this stage that you would ask your user group to carry out the tasks. As testers you would try to note down how users are approaching the task and how they are navigating through the application. Basically from start to finish you would note down the steps the user takes within the application to carry out and complete the set task. The problem with this is that is virtually impossible for you to note down every single interaction the user has with the application and even if you can this is very time consuming, especially if you are using a large user group.

An alternative method could be to create test cases for the tasks you have chosen to base your usability testing on and then use a test automation tool. Understandably, you may be wondering how you use a tool such as this in usability testing, as the traditional use for these tools is to repeatedly test whether different functionality within the application works. Well, the trick is to adapt your testing tool and use it in record mode. As users carry out the test cases or tasks you have given them, the test automation tool will record every interaction they have with the application. The data recorded by the testing tool will be collected within a test script, which can then be analysed.

Testers should look for patterns in how the application is being navigated and utilised. Were there parts of the task that took users longer to complete than expected? If so, why was this the case? Is it because they found it hard to find a particular function in the application? Did they go to the wrong part of the application? Was key information not prominent enough? By analysing the test scripts and asking these types of questions, testers can highlight problematic areas in the design of the application and ask developers to refine or rectify these issues.

When do you carry out usability testing?

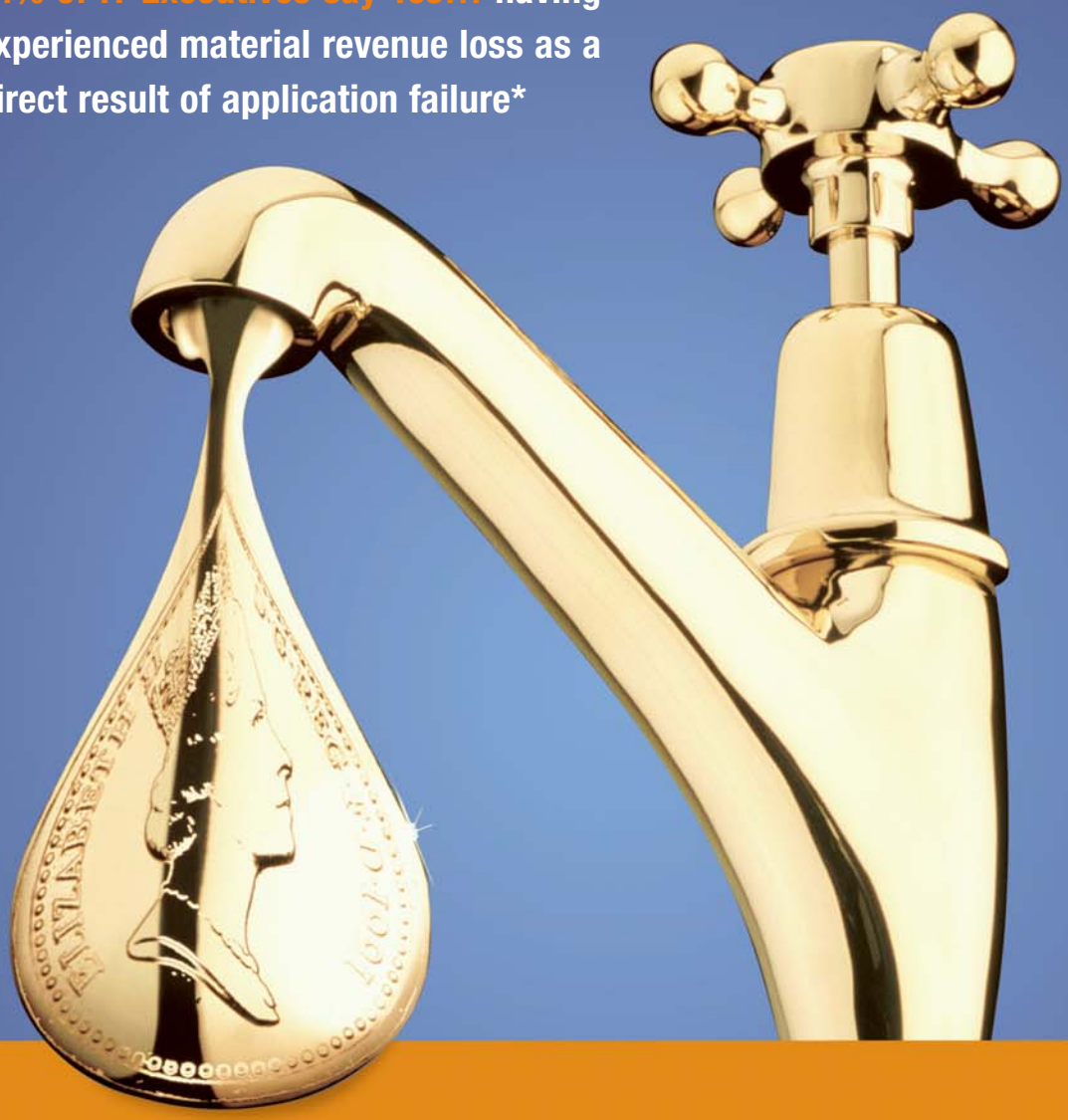
Usability testing should be started as early in the development cycle as possible. You should try to ensure that it is completed once a prototype of the application is built. It should then be repeated when major sections of the application are complete and again when the entire application is finished.

In addition, if the application is going to be maintained and updated then it is advisable to carry out usability testing before upgrades are rolled out to ensure changes to the way people interact with the application can be fed into the design of the upgrade.

Although usability testing takes time, money and resource, an understanding of how users interact with the systems we build is invaluable. Everyone involved in the development and deployment of an application should remember this and also never forget that the usability of a system will make or break the uptake of it. PT

Are your applications leaking money?

64% of IT Executives say Yes… having experienced material revenue loss as a direct result of application failure*



In today's competitive environment, IT departments need to release increasingly rich feature sets across complex distributed infrastructures. To reduce the risk of costly errors, analysts such as Forrester Research and Patricia Seybold, recommend an Automated Software Quality (ASQ) solution.

To learn how your software projects can be a third less expensive** and to download your ASQ information pack with Patricia Seybold white paper, visit:

www.compuware.co.uk/money

* Forrester Research - 2003 ** Patricia Seybold Group - 2003

COMPUWARE®
www.compuware.co.uk

Performance testing survey 2004

We invite all Professional Tester readers to participate in Embarcadero Europe's informal survey about application and performance testing for enterprise systems. Results and analysis will appear in the July issue

Q1 Which of the following best describes your organisation's primary business or industry? (tick one):

- ❍ IT services
- ❍ retail
- ❍ government
- ❍ financial
- ❍ insurance
- ❍ utilities
- ❍ pharmaceutical
- ❍ other: _____

Q2 How many people currently work in performance management or QA engineering/management at all locations throughout your organisation? (tick one)

- ❍ zero: not currently staffed
- ❍ 1 or 2 people
- ❍ 3 to 5 people
- ❍ 6 to 10 people
- ❍ more than 10 people
- ❍ I'm not sure

Q3 In your opinion which of the following issues concerning application and performance testing are affecting your organisation? (tick all that apply)

- ❍ testing solutions we have looked at do not enable collaborative involvement of professionals representing multiple IT disciplines and lines of business
- ❍ we lack time to perform proper, comprehensive application and performance tests
- ❍ performance testing is difficult to implement and thus difficult to justify
- ❍ it's difficult to customise tests to obtain precise answers to specific questions and conditions
- ❍ it's difficult to justify costs and maintenance fees, particularly for smaller testing projects
- ❍ it's difficult to demonstrate return on investment in performance and application testing
- ❍ other (please give details)

  _____
  _____
  _____

Q4 What method/approach do you currently use most for application and performance testing? (tick one)

- ❍ manual approach (scripts)
- ❍ automated approach
- ❍ we don't do application or performance testing (please state why not):

  _____
  _____
  _____

Q5 Have there been occasions when you would have liked to have conducted application and/or performance tests, but did not?

- ❍ no
- ❍ yes, because:
  - ❒ the cost was thought too high
  - ❒ insufficient time was available
  - ❒ the small size of the project made it difficult to justify the investment
  - ❒ an appropriate solution could not be found
  - ❒ other (please give details)

  _____
  _____
  _____
  _____

Q6 In your opinion, how aware is your executive management team of your critical application and performance testing issues? (tick the best answer)

- ❍ fully aware, with approved budget and sponsorship
- ❍ fully aware, but affected by limited budget
- ❍ somewhat aware
- ❍ not aware
- ❍ I'm not sure how aware they are

WIN A WIFI HOTSPOT DETECTOR!

To thank readers for contributing to this survey, Embarcadero Europe is giving away this year's must-have gadget: the pocket-sized Smart ID 802.11b and 802.11g wifi detector, which enables you to find out if you are in a hotspot without turning on your laptop! Everyone who completes the survey will automatically be entered into the prize draw and the first ten selected at random on 21st June 2004 will receive a detector.

EMBARCADERO EUROPE.

Q7 What features would be included in your **ideal** application and performance testing solution? (tick all that apply)

- ❍ integrated relational repository
- ❍ real-time, team-based collaborative testing
- ❍ facilitation of goals-based testing
- ❍ single console for managing all aspects of the testing cycle: environment setup, test creation, test execution, test analysis
- ❍ open, standards-based architecture
- ❍ performance analysis
- ❍ simple script record/replay
- ❍ comprehensive script paramatisation/randomisation
- ❍ wizard-driven Interface for key tasks
- ❍ quick installation with auto-configuration
- ❍ usage-based pricing

Q8 In your opinion, what is the most important feature or function that an application and performance testing tool should have? (please specify)

_____

_____

_____

Q9 Are you planning to purchase an application and performance testing tool? (tick best answer)

- ❍ in the next 1 to 3 months
- ❍ in the next 4 to 6 months
- ❍ in the next 7 to 12 months
- ❍ in the forseeable future, but timeframe is not yet decided
- ❍ not in the forseeable future
- ❍ I'm not sure

Q10 Please tell us about yourself and your organisation. This information will not be included in the survey and will not be shared with any other party.

Name: _____

Job title: _____

Organisation name: _____

Postal address: _____

_____

_____

_____

Daytime telephone: _____

Email: _____

Embarcadero Europe offers a complete set of application and data lifecycle management solutions that help leading companies build, optimise, test, and manage their critical data, database, and application infrastructure. Please indicate whether and how you would like to receive information about our award winning products.

- ❍ Extreme Test (application and performance testing)
- ❍ DT/Studio (data integration)
- ❍ ER/Studio (data modelling)
- ❍ DBArtisan (database administration)
- ❍ Rapid SQL (database development)
  - ❒ I prefer to receive this information by post
  - ❒ I prefer to receive this information by email
  - ❒ no, please do not send me any information

Please return this form (photocopies are acceptable) to Emma Estrada, Embarcadero Europe, Thames House, 17 Marlow Road, Maidenhead SL6 7AA, UK. You may also complete the survey online at www.embarcadero.com/survey.html

Not a matter of opinion

QBIT's web testing course presenter and PT editor

Edward Bishop believes web usability testing must be objective

For public-facing web applications which are intended to attract and retain users, good usability is as critical as correct functionality.

This article suggests a method, designed for use by testers carrying out functional testing of a public-facing website, to improve usability at the same time as, and as part of the same activity as, assuring correct functionality. It may also be of use in some non-public-facing applications, eg intranets and extranets. However usability of these is often considered to be of less importance because poor usability can be compensated for by user documentation and training and because the users generally have no or less choice as to whether or not to use the application.

What is web usability?

BS 7925 defines usability testing as *testing the ease with which users can learn and use a product*. IEEE 610 defines usability as *the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.* Both of these are hopelessly vague.

For a list of similar (and mostly similarly useless) definitions Google for "usability definition". The problem with nearly all of them is their reliance on the relative term "ease" (or similar alternatives such as "user-friendly").

Although many meanings can be applied to these terms, there are two main possibilities: "easy" can mean "straightforward" or it can mean "efficient". This article is about the former, ie how we as testers can help to increase the ease with which users of a web site can discover how to operate it correctly.

The so-called "usability standard", ISO 9241, is almost exclusively concerned with the latter meaning; it, and most other non-testing sources, describe what most people (including the standard itself) call "ergonomics" rather than usability. Poor ergonomic design in software, generally speaking, is obvious and detected early in development. Even if this is not the case and it falls to users to complain that they are being required to repeat tasks or do manual work unnecessarily, such defects are usually easy to fix. However ISO 9241 does give us the best definition of usability so far: *the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.*

A third school of thought says that usability is "whatever the user wishes to define it as". While this is obviously an admirable sentiment, it seems of little practical use.

When our ultimate aim is to increase usability by testing, it is better to take the negative view and define what usability is not; that is, to define a usability failure. For a website, this is one of three occurrences:

1 The user makes a mistake

2 The user becomes unsure what to do, or whether what they have done is correct

3 The user misunderstands the meaning of information imparted by the site.

Heuristics are for designers, not testers

Many of the most often suggested approaches to usability testing involve compring the design features of the site under test to advice given by various commentators on usability, also known as "usability gurus". The most famous of these are Jacob Nielsen (see *www.useit.com*), Vincent Flanders (*www.webpagesthatsuck.com*) and Steve Krug (*www.sensible.com*). These sites, and the interesting books written by these and similar authors, contain examples of good and bad usability and advice on which navigation and other design features tend to cause usability problems. Nielsen, in particular, has listed general principles (he calls them "rules of thumb" or "heuristics") for user interface design. All web designers should read at least some of this material.

Now, imagine yourself as a tester speaking to a designer/developer: "I've seen this feature on your site and Nielsen says you shouldn't have done it. His explanation of course is a general rule of thumb (or another guru gives a specific example which refers to a different site and is set in a different context), but as far as I can tell with my limited knowledge, understanding and practical experience of web design you're in the wrong and should be criticized and made to redo it the way I think it should be". The likely effect on the tester-builder relationship, project morale and the final product is obvious.

Keep your opinions to yourself

The second type of usability testing is often carried out informally thoughout the development process, and usually does more harm than good. It's natural for people to form opinions on the site; the subject may range from specific details ("users won't understand what the wording of that link means") to broad generalizations ("they shouldn't have designed the navigation like that, they should have done it more like my favourite site"). These opinions, especially the broader ones, are valuable early in development, and should be taken into account by those making decisions.

Once those decisions have been made however, further expressed opinion becomes damaging because it undermines the work already done and gives rise to change - with all the loss and risk that implies - for no clearly-defined or traceable reason.

Persons who have been exposed to the design of the site, or who have knowledge of the underlying business processes, cannot predict what users will understand or do. To do so would require the ability to "pretend" that one does not know what one does know. In many years working on web sites – right from their earliest days – I have never met anyone who can do this, from the most intuitive programmer, to the most experienced interface designer, to the most analytical and methodical tester. Their knowledge of the project makes them "tainted" and their opinions on usability worthless. Even so, their hunches may in fact be correct; but there is no way to check this.

Managers need to draw a line much earlier in projects beyond which the opinions of tainted persons, including themselves, are discounted and discouraged. After this point has been

Cresta Testing – accident prevention through test driven development

Ensure that you conduct the testing of your mission critical systems early in the project lifecycle through Cresta's Structured System Testing Methodology™ (SSTM™). Reduce risk and cost through the customisable Processes, Tools and Techniques which allow you to effectively manage and execute the complete lifecycle of testing within a project.

Cresta Training – where accidents should happen

Fully exploit your investment by turning WinRunner & LoadRunner into "wealthware" through educating staff in testing, quality processes and effective test automation. Cresta provides a broad range of renowned software quality and testing courses, including ISEB Certification in Software Testing, Mercury Interactive TestDirector, QuickTest Pro, WinRunner and LoadRunner.

For more details, contact Karen Espley on +44 (0)870 1600 333, email info@cresta.net or visit www.cresta.net/testing_training.php.

cresta
Proving the difference

www.cresta.net

CHANNEL ALLIANCE PROGRAM
MERCURY INTERACTIVE

reached, change should be allowed only if based on empirical evidence. Otherwise, the final design is likely to be influenced mainly by those persons with the loudest voices and/or most manipulating personalities. The dreadful end results of this syndrome can be observed at any of thousands of hard-to-use websites, many owned by very large organizations.

Usability trialling is not a test technique

One well-known way of gathering that empirical evidence is to introduce untainted persons in order to observe them using the site and ask their opinion. Some of the usability gurus advocate this, and much advice on various ways to carry out such an exercise (often called "user testing" or "usability trialling") can be found at *www.usability.gov* and *www.usableweb.org*.

The exercise can be very enlightening and interesting. However from the testing point of view there are several fatal, and unavoidable, flaws:

1 There's no way to know if the abilities of the subjects are typical of real users
2 Trials can't be done until late in development when change probably means very expensive rework
3 Any rework carried out as a result must then be retested (see 1)

The last is the most serious. The trial is likely to report that a large proportion of subjects made mistakes, encountered difficulty or expressed dislike of certain features or points in navigation of the site. However the subjects are far less likely to agree on what change could be made to improve matters – it must be remembered that they do not have a design or IT background (they could not be representative of users otherwise) and have been given only a few minutes to think about the site. Their opinions, although they may give useful insight, must not be used as a design.

So instead the findings are given to the site designers and they are asked to act upon them. But they have already tried to produce the best and most usable design they could: their first reaction may well be to say "despite the trial's findings, we think this part of the site is already as good as it can be" – in which case, there was no point carrying out the trial.

Otherwise, the designers must now think of even better ideas, or else readopt alternative ideas they had previously rejected. In either case, there is no way of knowing whether the new version is better than the old; the only chance of finding that out is to repeat the usability trial with new, untainted users. That will probably give rise to another, similar set of results suggesting shortcomings, perhaps in the same places as before, for reasons which are unknown: perhaps the subjects have less or different abilities than the first group, perhaps they are judging the features relatively and picking out the "worst" rather than the "bad", or perhaps the designers' first attempt at a given feature was in fact more usable. So we now have a situation where two expensive trials have been done, and perhaps more are needed, with no certainty of conclusive results at any time.

Test 15c: To show that a bill of value more than the current balance of account cannot be paid

Entry criteria

- function 15 'Pay a bill' and function 3 'Check balance' released for testing

Actions

1 log on to use home banking
2 obtain the balance of the account to which you have logged on
3 attempt to pay a bill of value exceeding the balance of the account
4 check the balance of the account again

Exit criteria

- balance checked successfully at least once
- bill payment details displayed for confirmation reflect user input correctly
- bill payment can be confirmed

Expected outcomes

- after confirming bill payment, information that bill cannot be paid appears
- the displayed balance of the account remains unchanged

Figure 1: Nongranular, requirements-based web test script

The best way to do usability testing is to do testing

If we as testers are ever going to help to provide more usable sites to users, a test method is needed that

1 is more objective and does not depend upon opinion
2 can be done by testers without the use of untainted subjects
3 can be started earlier and continued for longer, improving usability gradually throughout development
4 is economical enough for use in web projects with short timescales and frequent change of requirements and design for reasons other than usability issues.

Objectives 3 and 4 can be achieved by performing usability testing not only in parallel with but as part of the same activity as functional testing. The key to being able to do this in the test design phase: web testing scripts should provide opportunites for testers to detect usability issues. This means writing a script whose input section (often called "actions" in web testing) describes what the person executing the test should do to exercise the functions but avoids mention of how they should go about it.

The very granular scripts used for non-web testing, featuring step-by-step instructions, confirmation of the expected outcomes at each step, and individual traceability of every step to a business objective, are not appropriate for web testing. Users of a public web application will not have such guidance and will probably not be prepared to invest much effort in learning how to operate the interface. Writing such a script assumes that the developers and testers can predict the navigation choices and other inputs made by the user, and it is the unpredictable deviations from this prediction made by real users that give rise to severe usability failure and which testing should aim to detect.

Nongranular scripts such as that shown in figure 1 give exactly the same benefit to analytical testing: the scripts describe the system from the test analyst's understanding of the requirements, thus providing a check that the developers' understanding is the same, as well as helping to guide the developers in directing and prioritizing the development effort with the intention that all the tests will be passed first time. However they can also provide effective usability testing during the empirical phase which granular scripts cannot.

The starting point for a script should, of course, be a functional test objective. The most difficult part of writing a nongranular script is getting the correct level of detail. This is a skill learned quickly with practice. The script shown in figure 1 is a good example but was not written like this at the first attempt; rather, it is the result of a refinement process. Start by writing the steps as they come to mind, and then pass through them looking critically for ones which can be simplified because they give unnecessary detail or eliminated because they are not essential to the test objective. Incidentally, a similar process can be applied to the entry criteria for each test, enabling testing to be done earlier, when fewer items have been released.

If you ignore these opportunities, it's not just software you should be testing

£ATTRACTIVE + EXCELLENT BENEFITS + TRAINING + RAPID CAREER PROGRESSION

We are the world's largest and fastest growing independent IT performance assurance consulting group, with offices in 13 countries and over 550 consultants, including more than 100 in the UK. Our comprehensive, unbiased range of testing solutions – from specific expertise to full outsourcing – use automated testing tools across all platforms, applications, networks and operating systems over a range of sectors including financial services, telecoms, defence, public sector, retail and leisure as well as healthcare.

TESTING PROFESSIONALS ALL LEVELS — ANALYSTS, SENIOR TESTERS, TEST TEAM LEADERS & TEST MANAGERS

We have very ambitious plans for expansion in the UK, where we expect to grow by 94% and increase headcount by 60% in 2004 alone, and seek equally ambitious people who would thrive on working on high level projects, both locally and as part of international teams.

You will need a minimum of two years' relevant experience, good client and consultative skills, the personality and flexibility to thrive as part of a team in a global organisation as well as the ability to learn quickly and the drive to get things done. We are particularly interested in people whose background includes any of: Peoplesoft/HR Programs; SMS; MMS; WAP; LoadRunner; WinRunner; Rational Tools; Java; Biztalk; .NET; Set Top Box/Digital TV.

Our exceptional client base, variety of new projects and range of sectors mean the opportunities for personal and professional growth with Tescom are immense – and we are dedicated to rapid promotion from within.

So why not join us? And test yourself to the full. Please send your CV to jobsuk@testcom-intl.com or post it to Tescom Direct, Tescom International, 21-22 Great Sutton Street, London EC1V ODN. **www.tescom-intl.com**

TESCOM

AUSTRALIA FRANCE GERMANY ISRAEL SINGAPORE UNITED KINGDOM UNITED STATES

Objectives 1 and 2 are achieved by recording every usability failure experienced during testing, in exactly the same way that a functional failure is raised. This requires a certain amount of discipline from the persons executing the tests and recording results; however it is not necessary for these persons to be untainted.

The script shown in figure 1 was created from its test objective, which was based on a description of a function, which was derived from business requirements. All this was done before any design work had taken place and so the script does not contain any assumptions about *how* the user will access that function. Such a script can be executed by its author so that in very small projects all the testing can be done by one person. Very often in web projects early screen designs are available at the same time as or even before requirements and this may enable more detailed scripts to be created which do refer to specific screens and objects by name. A discussion of which type of script is best is not in the scope of this article, but it should be understood that scripts of the second type contain an implicit assumption that the user – who does not have access to these names – will recognize the purpose and meaning of the screens and objects. To check this assumption such scripts must be executed by someone other than their author; this means that in small projects all the testing could be done by two people, sharing the test design and execution work equally. In either case, however, during the empirical phase more people can be enlisted if necessary to execute scripts.

The point of writing scripts without guidance as to how to proceed is to make the knowledge and ability of the text executors irrelevant. Obviously proficient web users, or those with a knowledge of the system under test or its underlying business processes, will tend to experience fewer usability failures. However when they do, it is very likely that less proficient or knowledgeable users will experience the same failure. A long list of errors made by inept users is of little value; those users would probably experience similar difficulties on any site, and in any case their ability is beyond our control. A shorter list of errors and other usability failures recorded by experienced testers is an accurate guide to where the most severe usability faults lie; addressing these will lead to improved usability for users of all proficiency levels.

Failure is a fact, not a feeling

When executing a test, from time to time the tester will experience a usability failure; one of the three types listed above. The tester must be familiar with this list and sufficiently disciplined to be aware of and record each failure just as they would a functional failure or anomaly.

The first type, "the user makes a mistake", will typically include failures such as:

- user navigates to a page which was not expected
- user revisits a previously-viewed page inadvertently
- user enters invalid data to a form and is asked to amend

Failures suggesting these might be reported by the tester, respectively, as comments such as:

"When I clicked the "My account" link I expected to get a page that would let me log on and was surprised to see that it meant an explanation of the types of account available"

"I selected "recent transactions" from the dropdown list when trying to check if my standing orders were active for step 4. I didn't expect that it would lead back to the page showing what I'd done in this session, to which I'd already been at step 2"

"I didn't notice that I could not choose a password less than 6 characters long and my first choice was rejected"

These are obviously most likely to be noticed the first time a tester runs a given script, but sometimes testers will find by accident later in the script, when executing the script again (eg for compatibility testing), or when executing another script that they have been operating under a misconception and so experienced usability failure previously:

"I've just realized that up until now I've been going round the long way by selecting 'music' then 'CDs' then 'popular music' then '70s' when I could have just selected 'Great 70s Pop' from the dropdown on the front page. I ignored this link before because I thought it was just the title of an album being promoted"

The second type, "the user becomes unsure what to do, or whether what they have done is correct", might manifest itself as follows:

- user is not sure which link to click to see information they want
- user does not know what they should enter in a field
- user is uncertain whether they have entered a user transaction or not

which could be experienced and recorded by the tester as, respectively,

"At step 4 (order a 128MB DIMM) I was unsure whether I should select "computer upgrades" or "computer components"

"When entering my account number at step 3, I wondered whether or not I should include the dashes"

"When the 'track your order' screen appeared after I submitted the order form and stated 'order not yet picked' I was not sure whether this meant I had placed my order successfully or not"

The third type, "the user misunderstands the meaning of information imparted by the site", is the rarest; however if it occurs after deployment it can sometimes have severe business impact. It is usually one of the following situations:

- user selects incorrect product based on what they think the content means
- user proceeds incorrectly in the real world based on what they think the content says
- user misinterprets information or instructions on the site as advice

These are not usually reported directly by the tester; like the user, he or she can make mistakes of this type and remain unaware of the fact. However, they are sometimes prevented indirectly by the results of functional testing with a usability element; for example when examining a back-end database to ensure that transactions have been recorded correctly, it might be noticed that a tester has ordered a product which does not fit the criteria in the test script. More often, the tester realizes that they have been operating under a misconception and records something like:

"I've just realized that the colour of the model car is not shown in the photo but in that little box underneath. I've been ordering the wrong colours", or:

"I've just been surprised to find out that the 'buy this share' button appears on every page. On most of the ones I've looked at it's been scrolled off the bottom. I thought we were using it to recommend certain shares to users and might have chosen those shares as a result"

This last example actually occurred after deployment of a share dealing site and affected some users. It could not have been detected by error-guessing techniques such as checklists of site heuristics, and it is unlikely that inexperienced users performing a trial would have noted it. However usability testing of the type described in this article would have stood a chance of detecting it.

They all count

It is vitally important that testers are aware and bear in mind throughout testing that:

1 They must record every usability failure, even if they feel it is a result of their own lack of concentration or knowledge; users will be affected by exactly the same factors

2 Failures must be recorded even if the tester feels he or she could or should have noticed and recorded them earlier; testers, like users, will notice and deduce more about the site over time

3 One of the objectives of this method is to eliminate opinions and base change on empirical evidence. A tester may believe that a feature of the site will cause problems to some users and should be changed; however there is no way of knowing whether they are right in this or not. In contrast, when an error is recorded by a tester he or she is attesting to the fact that a usability failure occurred. It is conceivable (but would seem, intuitively, unlikely) that a tester with a strongly-held opinion might introduce a failure report designed to support that opinion.

If these guidelines are followed test execution will produce a potent deliverable: a stream of failures experienced during realistic use of the site, indicating precisely those points where it is important to improve usability. These must not be ignored by development; "we think this was caused by the tester, or an unlucky coincidence, and we don't want to change it" is not an acceptable resolution. Every failure recorded should cause a change to the interface, and if necessary the underlying system, which attempts to prevent that or similar failure happening again, during testing or after deployment; and that change should not be a reversion to a previous state which, by definition, has also been shown to give rise to usability failure. Often the change will be as simple as rephrasing or increasing visual prominence of text or an object; occasionally it may involve rethinking the grouping of information and form fields into pages or the navigation paths and methods offered to the user.

Sometimes a change will be made which is ineffective or actually makes the same or another usability failure more likely; if so, these failures will come to light as testing proceeds. The aim is not to attempt to maximize usability of the product in one "big bang", but to improve it gradually as testing and development proceeds, in the same way and at the same time as functionality is improved.

Because the functional tests have been prioritized by their relationships to functions and thus to business objectives, the more important faults should come to light earliest in the testing, and as testing and rework proceeds the frequency and criticality of faults reported should gradually lessen. More importantly, there will be more time to work on the faults associated with critical and/or popular functions; if change leads to further failure reports there will tend to be time for further change and further testing until failure reports related to the changed items cease. As the project nears its close, faults found during execution of the lower-priority tests will have less time to reach stability; so the success of this approach depends upon accurate prioritization of the functional tests.

Summary

Using this method usability changes made to the site are based on empirical evidence – the fact that a usability failure was observed – not subjective opinion. The evidence is gathered during the activity which must be carried out anyway: functional test execution. The number of faults detected, and the chance of detecting those which are hard to find, is increased due to repetition of navigation and other actions and the fact that testers have more time to think about their use of the site and are relieved of the very difficult mental exercise of trying to pretend they are a user. The best use is made of the testing time available since usability testing starts at the same time as functional test execution and is automatically prioritized by its association with functional testing.

The main disadvantage of this method is that the nongranular test scripts do not lend themselves to automated execution; or, to put this another way, automated testing cannot help with usability. However it might be possible to gain the main advantage of automated testing – faster and more reliable functional regression and maintenance testing – by creating an automated test suite near the end of the first phase of test execution when the frequency of usability failure reports has fallen and further major changes to the user interface become less likely. PT

Changing gear

Regional technology manager Mike Lucas explains how Compuware helped to implement regression testing of a constantly-changing product

The business environment of the automotive industry is a dynamic one, with changes like the ending of the European "block exemption" looking set to transform already complex relationships between manufacturers, dealers and customers. Anyone providing software to this industry needs to update their product frequently; it is vital they can do so without disrupting existing functionality.

Compuware customer Kerridge Automotive Systems has been able to keep up and move ahead of competitors by applying the latest automated testing techniques to its dealership product, Autoline.

Taking quality seriously

Kerridge has been in business for a quarter of a century and uses Autoline, a comprehensive dealership management system, to take care of all business processes. It is developed using Kerridge's bespoke programming language, KCML. The Autoline product is continually being updated in a six-monthly development cycle by the central in-house quality assurance (QA) team. Testing is a constant process, with programmers taking responsibility for thoroughly unit-testing their own work, before passing 'frozen' code on to the dedicated test team for independent integration testing prior to release.

Kerridge's manual testing (following any change) focuses on the modules that have been modified and on the surrounding functional area - the area most likely to be affected by the change. However, software developers know only too well that there is a small but significant chance that a change will affect an apparently unrelated area of the application. Such knock-on effects can only be discovered by regression testing which involves re-testing the entire product prior to releasing it for general use. Carried out manually, this would have been an extremely expensive and time-consuming task.

As Autoline's customer base had grown internationally, the question of regression testing became more pressing. Now that they were selling into many more countries, the cost of correcting problems after release would be much greater than when they had only three or four markets, so they needed to do everything possible to identify and correct any problems before distributing the software. Consultants in each country re-test the software after carrying out localisation and translation and before implementation at clients' sites, but Kerridge wanted to be certain these local consultants had a robust core to work on. This also applied to the importer management systems and fleet management software that Kerridge provides to the automotive industry, as both these systems also use Autoline as their development base.

Creating Test Conditions

At the time when regression testing was becoming an issue, the Kerridge QA team recognised that automated testing tools could make an end-to-end test feasible, both technically and commercially. They selected a suite of tools including Compuware's QARun for functional testing and QADirector for advanced test management.

Good object level recognition of an applications components (edit controls, tree views etc) is fundamental to achieving robust regression test packs. Kerridge used the tools to analyse and generate objects and test scripts that are both efficient and easily understandable. Recording a test script is a simple two-step activity. The first step is to navigate the required test path creating and naming objects as required. The second is to record the test script.

The Kerridge testing experts already had a clear idea of what they wanted to do. Rather than undertake a small-scale pilot, they put the tools to work in earnest. Careful thought was given to the processes that every dealership in every country would do every day, and a list of 140 basic processes, such as taking cash from a customer or selling a part at 10% discount, was produced. The plan was to put together a suite of tests that would allow Kerridge to carry out end-to-end testing of all these processes whenever anything in the system changed. Having documented their test conditions, the team began to use the tools to create the scripts that would execute the tests. After an initial training session to make sure everyone understood what was achievable, they were able to get to work.

End-to-end testing

The team has now built up a 'test catalogue' consisting of the 140 basic scripts. It has also created a standalone replica of a typical customer system in the field as the basis for running the tests – a test-bed which can be restored after each test run. Now, whenever there is a new version of the software, whether for a bug fix or a major release, the tests can be run quickly and the results automatically checked. The only significant human effort required is to check the results for highlighted exceptions, where the actual results do not match the expectations, and to follow up on these cases. There is virtually no overhead involved in increasing the volume of test data. Because the test is almost entirely automated, the only extra resource needed to build up the volume is disk space.

It is planned to expand the test catalogue gradually to cover more functions, Kerridge are determined to retain a modular test design based on simple modules. Rather than have huge scripts that test extensive processes, they will keep them short and simply assemble a big test sweep from multiple scripts. This philosophy helps to keep the overhead of script maintenance under control. Kerridge always recognised that there would be work involved in maintaining automated tests, but believed the benefit would justify the effort, and have now been proved right. A process is now in place for keeping the scripts up to date. In most cases all that will be necessary is to re-record a single script because there's an extra field on a screen or the order of fields has changed.

International testing

The test catalogue is distributed to all of Kerridge's operations and distributors worldwide as a basis for their own testing. Larger operations may decide whether to automate their tests; some are already in touch. Each operation could use automated testing tools to record its own scripts using the localized versions of the screens and then test the various levels of customisation: country specific, franchise specific, and bespoke. These different levels of customisation make local testing complex, so automation could be very valuable.

Autoline is a versatile product, but one that must be constantly revisited it to make sure it meets current market needs. Kerridge is now able to take a proactive approach to testing all these changes and avoid "being caught on the back foot". PT

Ghost in the Machine

John Kent, MD of Simply Testing Ltd, continues his series.

Part 4: Advanced test automation architectures

A test automation architecture is the same as a software architecture but built in the language of the test tool. But what is a software architecture?

What is an architecture?

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them" –Software Architecture in Practice, Bass et al, (Addison-Wesley 1997)

So, a software architecture is the structure of the source code and the relationship between the components. In civil engineering, the architecture is the overall structure of the building. The architecture provides the space for the functional use of the building and ultimately is the major factor affecting how successful it is at fulfilling its purpose.

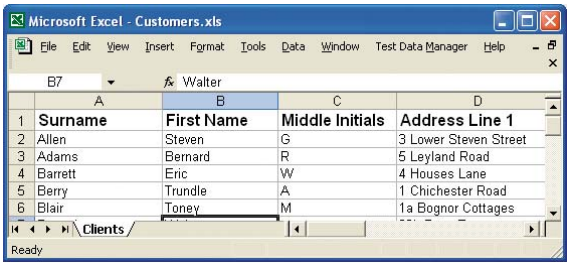


Figure 1: Simple 'data-driven' data

We have already seen that scripted test automation consists of code and is really a product of software engineering, therefore it should have a good architecture to be successful. Little wonder that test automation does not get very far when left to junior programmers or testers when they have some spare time. No matter how hard-working these people may be it is too much to expect them to create a good architecture. They are often too busy with the low level detail to be able to see the overall structure and do not have the software engineering experience to design architectures. They are builders, not architects.

A good test automation architecture provides structures for logging and error reporting and allows recovery to be incorporated (dropping failed tests, navigating the system to a base point and continuing with the next test). It will have libraries of reusable functions in much the same way as any other software system. Most importantly of all its structure will minimise the code maintenance overhead and thus it is the starting point for success with test automation.

Advanced test automation architectures are really the logical place to go when you take a software engineering approach to building test automation. So how do advanced automation architectures differ from the basic data driven approach?

In order to illustrate this, let's first look at some data used in data driven automation (see figure 1). In this example of Excel test data, every line (except the heading line) has customer details in it. Column 1 is the surname, column 2 the first name etc.

In the test tool there will be a function which repetitively reads in this data and inputs it into the 'Add customer' windows or screens. Most basic data driven architectures for more complex systems involve many different data files for the different types of business functions and many automation programs to input the data.

One of the main disadvantages of the basic data driven approach is that the automation pumps data into the system in an unnatural way. Most regression tests are not about repetitively inputting similar data into the user interface; rather they seek to fully exercise all of the business functionality of the system under test in a realistic way.

Automated testers have recognised this for a while now and some provide a solution for it by adding in another layer to the automation which specifies the order in which data is used from the various files. For example in an insurance system, the extra layer would select the customer data to add, then select the insurance policy data to add for that customer and so on. This however becomes a mess architecturally.

What the advanced architectures do is take (at least some) of the navigation and actions out of the test programs and put them into the test data. The test data becomes the script—the sequence of actions to be followed. It tells the automation code what to do and in what order. Advanced architectures allow the test analyst some choice about what the sequence of events should be. The level of choice is dependent upon how sophisticated the architecture is.

Figure 2 illustrates data from an advanced approach. Note that if there is a hash at the beginning of the left hand column this means that the line is a comment and not test data. Lines beginning `Supplier_Add` or `Stock_Item_Add` are the actual test data which will be read by the automation code. The format of the file is very different to that in figure 1 because the meaning of the data in each column is dependent upon what type of line it is and this is defined in the first column. The comment lines give the format so that the test analyst knows what each column represents. For example in the seventh line, column four is the 'Name' field because this is a `Stock_Item_Add` format. Thus the test analyst can choose the order of the actions when creating the Excel data.

#Supplier_Add	Sup Name	Description	Type	External Flag
Supplier_Add	SC_SUP00	SC-SUP00 Supplier Company		OFF
Supplier_Add	SC_SUP01	SC-SUP00 Supplier Company		ON
#Loc_Add	Loc Name	Description	Corp	Corporate Y/N
Loc_Add	SC-SXX-1	Test LO Stock Room only	ON	OFF
#	SI Name	Item Description	Name	External Flag
Stock_Item_Add	SC_STCK00	CABLE	COAX	Test SI
Stock_Item_Add	99	PUBLICATION	BOOK	Wind in the …
#	SI Name	Description	type1-01	Quantity
Stock_Loc_Add	SC_STCK00	SC-SXX-1	1-01	10
Stock_Loc_Add	99	SC-SXX-1		10

Figure 2: Advanced architecture data

When the automation runs, a driver program reads through the data and calls the function specified in column 1 passing it the data for that particular line. For example there will be a function `Stock_Item_Add` which will have been written by the test programmer (scripter) in order to perform all of the actions

required to add a stock item. These functions are what are known as *wrappers*.

Automation wrappers

Wrapper is an OO term that means some software that is used to interface with an object. In programming terms you call the wrapper if you wish to use the object. You can't call the object directly.

In test automation, wrappers are programs or functions written in the language of the test tool which perform discreet automation tasks as instructed by the test data and actions. They provide the interface between the test and the system under test's user interface. In our previous example there are four wrappers – `Stock_Item_Add`, `Loc_Add`, `Supplier_Add` and `Stock_Loc_Add`. These are business level wrappers – they 'wrap' business functions and were written by the test programmer (scripter). You will be able to see that the example in figure 2 is similar to the data driven approach but the first column in the test data of each line tells the automation which wrapper to call.

There are two distinct types of advanced architecture. Figure 2 shows a *business object level architecture*. In this type of automation architecture, one automation function or program is written in the test tool for each type of business task. These functions are the wrappers for the business tasks.

Usually a functional decomposition of the system is the first step in building this type of architecture. In a functional decomposition, the basic business functions of the system are defined. Then the wrappers for each business task are programmed (scripted) and the data format for each task is created.

Test data in business object level architectures is at the business language level and therefore understandable by end users, which is a great advantage.

#Screen					E&xit		Fo&rward					
#	EditBox	POLREFIN	EditBox	GRPREF	Button	cmdBUTO	Button	cmdBUTTWO_DATA				
#	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA				
ZB01N				Scheme			CLICK					
#Screen									Employed contribution		Contracting out of SE	
#	EditBox	SCHNAM	EditBox	CLIREFIN	E	POLREFIN	EditBox	SCHINQ_D	CheckBox	chkEEE_D	CheckBox	chkSRP_D
#	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA
ZB15N					CHECK	AD012083			SETON		SETON	
#Screen												
#	EditBox	SCHINQ_D	EditBox	CLINAM	e	PRDNAM	EditBox	SCHNAM	e	CLISUR_D	EditBox	CLIFOR_D
#	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA
CL12N										Kent		John
#Screen												
#	EditBox	SCHINQ_D	EditBox	CLINAM	ListBox	PRDNAM	ListBox	SCHNAM	ACT	PCDTWO	ACT	PCDONE
#	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA
CL02N						PWF03				4TP		GU21

Figure 3: Screen\window level advanced architecture data

Screen/window level architectures

In this architecture there is one test program that deals with each screen/window in the system—it acts as a wrapper for that screen/window. It handles all of the input and output (getting expected results) for its window or screen. See figure 3 for an example of the data. Again lines with a hash in the left hand side are comments used to show the test analyst the format of the data for that screen/window. The other lines are the test data which is passed to the automation wrappers for that particular screen. The first column of the test data represents the screen that the data refers to and thus the format of the line is dependent upon what the screen name is in column one. Also note that, as this a test of a GUI system, for each user interface object there is an action and a data field. Thus the test analyst can specify actions like 'Check' that the object contains the data equal to 'Smith' or even '`CheckEnabled`'.

One of the biggest advantages of screen/window level architectures is that all of the user interface objects can be made available to the test analyst in Excel and thus the analyst has complete control over what the navigation and actions should be, rather than being dependent upon what the test automation programmer (scripter) has put into the wrapper—as with business object level architectures.

Next issue: New ways to create automation code

PT

Using risk as the basis of test planning

Consultant and trainer Felix Redmill says there is such a thing as tolerable level of failure

If we have learned one thing in software development, it is that we must get the specification right. More project and system problems arise from the gaps, inaccuracies, and ambiguities in specifications than from any other cause. Yet, specification problems recur in project after project. The fact is that producing and maintaining good specifications is not trivial. Acquiring the necessary information is difficult, analysing it is a particularly problematic task that is often omitted, verifying the results is tedious, and expressing the requirements in clear, grammatically-correct language is impossible for many.

The test of a lesson being learned is evidence of change in the thinking process as well as in the way of doing, and the recurring lesson should by now have influenced our approach to everything. Yet, not only do deficient specifications continue to plague projects, but casual observation suggests that software engineers have not transferred the lesson into their own lives. They seem to take no more care in specifying the requirements for meetings with friends, journeys through unfamiliar parts of the country, and picnics for their children, than do other members of society who have not had the benefit of their salutary experiences.

Perhaps the preparation of specifications, and planning in general, do not come naturally to us. It seems that our heuristic approach is to identify a goal and to take intuitively defined steps towards it, rather than to plan the route in detail. In life we are usually oblivious to the inefficiency of this strategy because, in the main, it seems to work, and when it doesn't, the consequences are mostly small and not far-reaching. For example, having not clearly specified their meeting place, two people wait for each other in different parts of a train station, but they laugh it off later; a traveller doesn't plan his journey and becomes lost, but arrives only an hour late and is relieved at the outcome; the ice and sandwiches are forgotten but the picnic goes ahead anyway, and everyone enjoys the cake and says that the drinks were really cool enough.

In simple situations, intuition gets us by, inefficiently but mostly without catastrophe. But in development projects and other non-trivial endeavours, intuition is not enough. In well managed projects, the fact that specification and planning do not come naturally is compensated for by the provision of training, structured methods of working, quality assurance, and other means. Because planning needs to be based on a specification, difficulty in planning is often an indication of specification deficiencies. Yet, the opportunity that this presents for specification clarification or improvement is frequently ignored, with useless or misleading plans being produced rather than specifications being revised.

The importance of specification is as great in testing as at other stages of software development projects. Without clarity of what is required, test planners cannot carry out their task with confidence; they cannot plan the most effective testing. Yet in many (perhaps in most) instances, vague specifications are accepted without challenge. The culture of testers seems to be to accept the inadequacies of others rather than to challenge them, even though the quality of their own work is compromised.

This point was made by the writer of a letter published in a recent issue of *Professional Tester (issue 16, October 2003).* The writer claimed to be confused, having read 'some amazingly ridiculous things', and provided two examples of them. The first was 'the requirement is for 80% statement coverage', and the letter writer asked which 20% of the statements do not need to be tested, and whether this could be thrown away and a 20% discount obtained on the cost of the application. The second example was, 'a package could make do with a medium level software quality', in response to which the letter writer asked how much is saved by accepting medium rather than high quality, and what additional saving would be made by settling for low quality. Both the original expert's statement and the letter writer's question assume a correlation between quality and cost, but it is worth noting that inferior programmers and slack management are almost certain to produce very costly low-quality software!

The letter writer called on experts to desist from making such statements and to provide a sound basis for test planning. The trouble is, though, that this is hard to come by, and the silence of testers suggests that it isn't thought to be needed. There isn't a universally agreed basis for test planning, and it doesn't seem to be missed. Perhaps a part of the cause lies in the fact that, although students are sometimes (but not always) taught testing, they are almost never taught how to plan it.

Returning to the quoted statements, are they indeed amazingly ridiculous, or are they founded on some element of good sense but just inadequately expressed? If the latter, why do the experts who write them use forms of expression that first baffle and then amuse readers, instead of creating statements that ring true? They raise doubts about authorship and editing in the field of testing, and about expertise and professionalism too. They also raise questions about test planners and testers who, although admitting to not knowing how to interpret the statements, seldom question them. But is there some rationale behind such statements? If the authors understood it, surely they would express it and not leave the statements apparently groundless.

Without justifying their conclusions, experts define requirements for 80% statement coverage rather than, say, 40% or 90%, and for "medium-quality" rather than "high-" or "low-quality" software. Perhaps they sense something that they don't understand. If testing is time-consuming and costly, and cannot be carried out to the same extent on all software, then preference might usefully be given to the software that matters most. Risk is implicated. Indeed, it is often mentioned. But how it forms the basis of test planning is not made clear. Yet, if the risks arising from system or software failures were understood – identified, analysed and understood, and not merely guessed at – it would be possible for stakeholders to use them as a basis for deciding what likelihood of failure they were prepared to tolerate from a system, or from particular subsystems. How such decisions may be translated into test plans is the subject of the remainder of this article.

Risk is a function of two variables: the probability that a defined undesirable event

will occur, and the potential consequences if it did. In the context of software that has not yet been tested, it is possible to determine failure's consequences but not easy to estimate its probability. Thus, although we speak of 'risk', in the present context, the single factor, consequence, is proposed as the basis of test planning. The consequences are likely to be different for the various stakeholders. They may involve financial loss, safety or security breaches, loss of goodwill, or mere inconvenience, and each of these may be more or less serious, depending on the circumstances. Their identification, including its difficulties, has been addressed in previous articles in this column and will not be explained here. In some cases, the consequences of failure may be assessed for the system as a whole; in others, it may be possible to determine them for subsystems - but this needs to be done with caution, for the effects of interactions between software items can easily be overlooked. But whatever the reference point, if the consequences are categorised according to severity in the context of the system's use, the categories may be used to inform test planning.

Suppose we define four consequence categories, or classes, Class 1 being trivial, Class 4 being catastrophic, and Classes 2 and 3 forming categories between the two. The software in question would then be accorded the class of its highest-class consequence. So we know one of the components (consequence) of risk but not the other (probability). We also know that we can reduce risk by reducing either the consequence or the probability of a given type of failure. Assuming the potential consequences in each case to be fixed, because they are linked to the system's objectives, risk reduction must be achieved by reducing the probability of failure. It is true that failure may be caused by operator error, the probability of which should be reduced by design, documentation, training, supervision, and other means, but these issues are beyond the scope of this article. What we are concerned with here is reducing the probability of failure due to software faults - by reducing the number of faults and by finding and removing those that would result in the greatest consequences. (Of course, it would be preferable to produce better software in the first place, but that is another matter. The subject now is testing rather than development.)

The logic that we are following leads to the proposition that testing would be most effective if Class 4 software were tested more thoroughly than Class 3 software, Class 3 more thoroughly than Class 2, and so on. In other words, testing should be risk-based. But the process depends on the classes being predicated on properly identified and analysed risks and not supposition. Nor should it stop there. It requires the methodical development of test programmes, one for each software class. Naturally, test cases must be designed specifically for individual items of software, but the general plan, based on the context of system use, and also on the intention to put most effort into trapping the faults with the greatest potential consequences, should be designed such that there is progression from the basic test level to the most rigorous, ie:

Test programme 3 = Test programme 4 – something;

Test programme 2 = Test programme 3 – something;

Test programme 1 = Test programme 2 – something.

Test programme 4 needs to be designed for providing confidence that the probability of failure of Class 4 software is reduced to a tolerable level. For example, in the context of safety-critical systems, it may include substantial static analysis as well as formal proof of correctness with respect to a mathematically based specification. Other test programmes are then reductions on programme 4. Test programme 1 may in some cases call for no more than a level of black-box testing - and it could include a requirement for '80% statement coverage'. But here the basis for making it is clearly stated, and the requirement is seen to be not ridiculous but a purposeful attempt to achieve cost-effective testing appropriate to the circumstances.

But what is a tolerable level of failure, and how can we know when we have achieved it? By judgement (which is always subjective and always needs to be justified) we can define a failure rate that we would consider tolerable in the circumstances. A continuous-operation example is, 'no more than one failure per ten thousand hours of operation', and an on-demand example is, 'no more than one failure per thousand uses of the application'. But we cannot prove achievement in advance. We can never be certain. We must seek to increase confidence, and we do this by increasing the rigour of testing in proportion to the judged acceptable failure rate (based on the severity of the consequences of failure).

This approach relies on the intuitively plausible assumption that a more rigorous test process achieves better results. However, finding bugs depends not only on the test plan but also on the way in which it is implemented, and on whether the faults found are corrected and the software re-tested. Importantly, it depends on the designers, programmers, and their managers. If sows' ears are produced in the first place, good testing will not make silk purses of them.

If the version of risk-based testing described here seems a good idea, then let us not ignore the assumptions made in executing it. Determining the various consequences of failure can be difficult, and we can easily overlook or underestimate some of them. Further, linking consequences to individual subsystems through a cause-and-effect chain, based on top-down decomposition, may implicitly assume the independence of subsystems from each other, but, unless design has included partitioning, this is unlikely to be valid. Then, the confidence that we derive from our testing is strongly linked to our starting assumptions. Although is not difficult to devise test plans and cases, we cannot be sure that they will uncover the faults that need to be found, and our final confidence that they have done so is related to our initial assumption that they will. Notwithstanding these and other assumptions that we might make, the approach is methodical; it is preferable to basing testing only on intuition.

Returning to the statements that our letter writer referred to as ridiculous, it is now apparent that their use implies an intuitive recognition that testing is most cost-effective when based on risk. But intuitive recognition does not amount to understanding. The statements are not couched in terms of risk and they do not escape ridicule. We need an understanding of the subject of risk so that risk-based testing can be well done. PT

Through other's eyes

Independent testing specialist Tim Edmonds closes our issue on user interface testing with his take on the very first thing designers and testers should consider: accessibility

My sixteen years of testing experience have been mainly at the integration, system and user acceptance levels, with the emphasis firmly on functional testing. Indeed it is significant that on many projects on which I have worked the source documents on which testing was based were called 'Functional Specifications'. On some projects I tested non-functional attributes such as performance and security, but when working with mainframe and PC based applications any usability issues were generally confined to matters such as ensuring consistency in the user interface, and the availability of appropriate help text and error messages.

Testing web-based systems changed all that, with known target user environments being replaced by an open environment. One consequence of this was a need to test how accessible the functionality was under different configurations of operating systems and browsers. Another was the lack of a standard for the user interface. As I began to test the functionality of web applications I soon became painfully aware of some usability issues; these included inconsistent methods for carrying out similar tasks, over-lengthy navigation paths, unnecessarily complex screens for carrying out apparently simple tasks, and needless validations. For e-commerce applications the implications were serious.

I began to discuss these problems with developers, to build usability into my test plans, and to report usability issues as software faults. This was still very limited in scope and did not include any formal consideration of broader accessibility issues, such as the aspects of website use that have most impact on those with physical impairments.

The realities of accessibility

My awareness of web accessibility was brought into focus following an incident last October which put my left leg in a thigh-length plaster for several months. During that time I was confined to a world that consisted of three downstairs rooms at home, although the internet meant that I could still communicate with the outside world via email and the web. However, space was limited by a make-shift arrangement of furniture that was necessary, and my PC had to be positioned such that its TFT screen was a metre away and slightly above eye level. Even after I had adjusted the brightness setting to its maximum, the quality of the display from this viewpoint was significantly reduced compared to normal desktop use. Effectively I was accessing the internet as someone with visual impairment and so I had to make adjustments in the way that I configured and used the PC.

With a lengthy spell off work in prospect I wanted to fill my time usefully, and an opportunity soon arose when a locally-based charity asked for volunteers. My offer of help was accepted, and I was asked to do a job which involved making telephone calls to various other charities, not-for-profit organisations and related commercial companies. First I needed to research these organisations to find out what they did and to obtain a telephone number and contact name – a task ideally suited to the web.

It was this work that quickly brought home to me some realities of website accessibility, and I began to look at the pages that I visited wearing a usability testing hat. The same applied to my regular internet usage – some of my favourite sites were less easy to use in my changed circumstances. What follows are observations on some of my web accessibility experiences during my incapacity. From them I suggest some resources to use in addressing these issues when testing web applications.

Size matters

One of the first things that I did to address my problem of distance from the PC display was to set the view text size to 'Larger' in my browser (Internet Explorer 6). Immediately I was struck by the different responses of websites to this change, so to magnify the effect further I then set the size to 'Largest'. A few sites coped well, with all or most of the significant text being enlarged and more readable, as I expected. However, on several sites the increase in the size of text had an adverse effect on the rest of the page, typically causing text to overlap graphics, text boxes, borders or other text, with consequent difficulties in reading. On a few sites the text size did not change at all and so small text remained too small to read.

That was only the beginning. Further investigation of my browser's accessibility capabilities revealed that it could be configured to ignore the font sizes and styles specified on web pages. This solved some problems but often revealed additional inconsistencies and errors, thereby opening up a further range of testing possibilities. The lesson for me as a tester was clear – the use of different text sizes was an accessibility issue which previously I had not considered specifically in my website testing, but which I would certainly include now.

The colour is magic

My view of the screen meant that my ability to distinguish between colours was reduced and, as I soon discovered, on many websites the colour combinations used could result in text magically disappearing into the background. For example, a visit to a local authority site revealed a combination of a dark green background with dark text colours which made navigation a nightmare, regardless of the size of the text. Here and on several other sites the colour choice seemed to be influenced by adherence to a 'corporate' colour scheme that was probably designed for purposes other than web pages. Testers should be prepared to point out this kind of conflict – those responsible for company design criteria may not be aware of implications like this and should welcome feedback.

I found that the easiest pages to read were those with black text on a white background – straightforward and effective – but any dark colour on a light background was also fine. Light text on a dark background was usually readable, but the choice of font then became significant – thin spidery typefaces tend not to work well in these circumstances. Some people with visual impairment find white-on-black the most readable and one charity site offered this as an alternative to its standard black-on-white.

My investigations of the browser's accessibility options revealed a configuration option to ignore the colours specified on web pages. However, as I found with text size, this was sometimes only a partial solution that served inadvertently as a 'test technique' to highlight other issues.

A picture is worth a few words

Another potential block to my view of website contents was the use of graphics and video clips. Of course both can be effective

media on websites, but they can also intrude and become counterproductive when used as self-indulgent embellishments. For example, several of the sites that I visited used an image as a tiled background such that text positioned over it was very difficult to read, particularly when text and image colours were similar.

Animated graphics can also cause difficulties. Sometimes they merely distract, but in two cases there were hyperlinks behind animated graphic controls, making it a challenge to read the text on them that described their function. One site included this 'feature' on most of their pages so that navigation was rather like playing an arcade game of 'hit the hyperlink'. Admittedly this was a youth charity, whose typical users would be more visually and digitally adept than me, but they made it as difficult as possible for me to find out how to contact them.

A picture may be worth a thousand words, but sometimes you need a few words as well as, or instead of, pictures. Images on web pages can (and should) have some 'alternative text' associated with them, to be displayed if the image is not present for any reason. I configured my browser to disable graphics. This showed where no alternative text had been provided and, where the image was also a link, it meant that there was no indication of where the link went. Many visually impaired users access the web using text-to-speech browsers, which will render alternative text audibly, so lack of text effectively hides the message in the image, whether or not it has a hyperlink associated with it. An extreme example was a site that on entry provided a page consisting only of graphical images with no alternative text, so with graphics disabled in the browser there was no text at all displayed on the page! So here is another testing suggestion – try running some web test scripts with the browser graphics disabled. Similarly, if a website makes extensive use of video then try running some tests with video disabled.

Some guidelines and resources

What I have described above are a few examples of real web accessibility issues that can be used to feed into formal or informal usability testing. This is, of course, a selection based on personal experience with impaired vision and merely represents the tip of the iceberg. For example, what about colour blindness, sensitivity to flashing screens, and deafness? Fortunately there are many resources available that can be used to plan and focus the test effort:

The World Wide Web Consortium (W3C) under its Web Accessibility Initiative (WAI) provides a set of Web Content Accessibility Guidelines which can be used as input to web page design and testing. The current draft of this document is available at *www.w3.org/wai*. There is a wealth of other useful material here, including information about evaluating websites for accessibility and alternative browsers for special needs.

There are various statutory regulations that have a bearing on accessibility. For example, in the UK the 1995 Disability Discrimination Act, Part III – Access to Goods and Services, makes it unlawful for service providers to discriminate against disabled people in certain circumstances. For more information see *www.disability.gov.uk/dda/#part3*.

In the USA Section 508 requires that Federal agencies' electronic and information technology is accessible to people with disabilities. There is a great deal of information available at *http://www.section508.gov*.

Jakob Nielsen's Alertbox columns, published on his website at *www.useit.com/alertbox*, are a rich source of information about usability issues in general and include several covering accessibility topics.

Tools and automation

Using different browser configurations for text, colour and graphics is effectively adding additional test environments. So, in the same way that you might test technological accessibility under different versions of operating systems or browsers, you can run tests under alternative browser settings. Fortunately testing conformance to guidelines or regulations is a task well suited to automation, and there are resources available through the internet to help with this. For example, the WAI website has a section with links to many different measurement tools, and the Section 508 website contains links to tools and resources for implementing conformance.

Whether or not you have a mandatory compliance requirement to measure and improve accessibility, the Bobby website at *http://bobby.watchfire.com* is a good starting point. This measures against either the WAI Web Content Accessibility Guidelines or Section 508, and awards a 'Bobby approved' icon for websites that prove a given level of conformance. If you are not sure about how accessible your website is, then Bobby offers a free service that provides accessibility feedback on a given URL (single page only) and this is well worth trying.

Tools like Bobby are particularly suited to situations such as that where a 'corporate style' transferred to the web page gives accessibility problems. In these cases a report from a tool based on a recognised standard or set of guidelines offers an objective way to inform and convince management that it is worth taking a different approach to the website.

Feedback and a word of caution

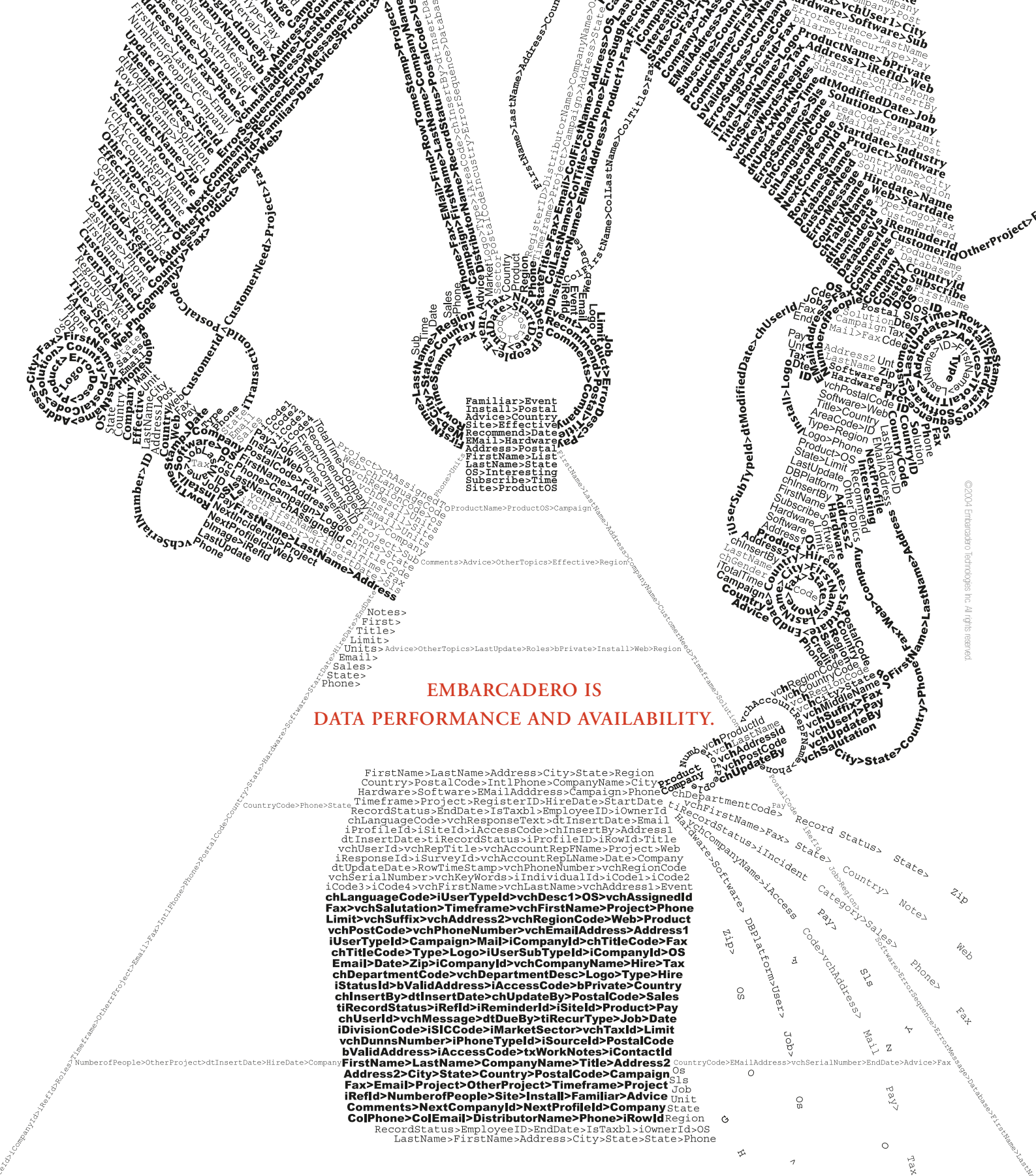
The work that prompted this article was concerned mainly with charity sites and 'caring' organisations. Whereas I would just abandon my use of a poorly accessible commercial website, in these cases I took the trouble to give some feedback. So, where there was an easily reproducible error or usability problem (and if I could locate the contact details) I sent an email 'incident report' describing it. On revisiting some of these sites when writing this article I was pleased to see that several of them have made changes and I can 'close' the incident reports! They include the local authority site with the indistinguishable text and background colours, which now displays a very readable dark green text on a light yellow background. Alas, the site that opens with the text-free page when graphics are disabled remains unchanged and just as unusable.

At the other end of the spectrum, there were some sites that stood out because they were clear and easy to use, so I took the time to tell them so. For an example of what can be done to make a website clear and accessible, take a look at the Bobby-approved site *www.sense.org.uk*.

However, a word of caution is necessary here. A website that displays a *Bobby approved* logo shows that it meets a certain level of conformity to a set of guidelines, as reported by a tool analysis of the underlying HTML. However, it does not mean that there are no accessibility problems. For example, on another approved site I found the Bobby logo and the accessibility features were on the home page, but instead of being positioned at the top left of the page they were at the bottom right hand corner, reached only by scrolling and easy to miss. One of the features offered was a 'text only' option. Selecting it initiated a conversion utility to render the page and this took several minutes to complete – a process that was repeated for each link selected. Furthermore, the resultant pages were the utility's literal interpretation of the original HTML, including any duplicate links and unnecessary information.

So, by all means use conformance tools as part of testing – they have a valuable role to play – but develop some additional accessibility tests of your own. Users need testers on their side and improving accessibility for users with special needs will, after all, improve usability for everyone.

Author's note: I am grateful to all the developers and test practitioners who have taken the time to discuss usability and accessibility issues with me, and particularly to Isabel Evans who, amongst other things, introduced me to Bobby. PT



You've just gone live with a new mission-critical application. Everything went well. The big system integrator left their invoice. But suddenly it's all come to a grinding halt. Money is evaporating while anxious colleagues and bosses wait for you to troubleshoot the problem, then test the system you've brought back to life. This could have been avoided. With Embarcadero's Performance Centre and Extreme Test, you can proactively optimise the databases and verify application availability and performance, ensuring system uptime and reducing costs. To learn more, contact Chris Birks on +44 (0)1628 684443, email **chris.birks@embarcadero.co.uk**, or visit **www.embarcadero.com/performance**.

EMBARCADERO
TECHNOLOGIES®
we make data work™