

# Incorporating Usability Testing into Formal Software Testing Models

James Christie



UNIVERSITY  

---

*of*  
ABERTAY DUNDEE

School of Computing & Creative  
Technologies

MSc Information Technology 2007

INCORPORATING USABILITY  
TESTING INTO FORMAL  
SOFTWARE TESTING MODELS

James Duncan Christie  
0504217

# University of Abertay Dundee - Permission to Copy

Author: **James Duncan Christie**

Title: **Incorporating Usability Testing into Formal Software Testing Models**

Qualifications: **MSc IT**

Year: **2007**

- (i) I certify that the above mentioned project is my original work.
- (ii) I agree that this dissertation may be reproduced, stored or transmitted, in any form and by any means without the written consent of the undersigned.

Signature .....

Date .....

# Abstract

## **The Problem**

This project is aimed at the gap between usability engineering, as envisaged by the Human Computer Interface (HCI) profession, and software engineering (SE). Specifically, it is aimed at the historical gap between usability testing and formal software testing, reasons for the gap and prospects for improving the situation, and means by which software test managers can improve matters on individual projects.

## **Methodology**

The project was carried out in three phases.

- 1 Literature review.
- 2 Company visits.
- 3 Analysis of research.

## **Major Findings**

- 1 In the 1960's and 1970's SE evolved following dysfunctional models, initially the Waterfall Model and then also Structured Methods. These were implicitly built on the mistaken assumption that SE was an orderly process, akin to civil engineering. Usability was a major victim of this failure.
- 2 The HCI profession lacked sufficient understanding of the reality of software development, thus contributing to HCI being marginalised by SE.
- 3 The problem is not that usability testing needs to be incorporated into SE testing. HCI needs to penetrate the whole development lifecycle, rather than be confined to performing summative evaluation at the end of the project when it is too late to make a difference.
- 4 It is widely recognised that HCI has not had sufficient penetration into SE. This dissertation argues that the situation is even worse than most academics have acknowledge. It is still common to find highly experienced and competent SEs with no knowledge of HCI or usability.

## **Conclusion**

Traditional development models and practices may not prevent the development of usable applications, but they make the job very much harder.

Promising possibilities are offered by Object Oriented and Agile development techniques. However, neither is specifically geared towards usability, and the key issue remains whether

the organisation is committed to developing usable applications. Without that commitment new techniques and technology will just be used to develop bad systems more efficiently.

However, even if the organisation is not committed to usability, forceful test managers can make a significant difference to the usability of applications, provided they have the knowledge and skills required to influence fellow managers.

# Table of Contents

<b>University of Abertay Dundee - Permission to Copy</b> .....	ii
<b>Abstract</b> .....	iii
<b>The Problem</b> .....	iii
<b>Methodology</b> .....	iii
<b>Major Findings</b> .....	iii
<b>Conclusion</b> .....	iii
<b>Table of Contents</b> .....	v
<b>Table of Diagrams, Figures, Charts</b> .....	viii
<b>Introduction</b> .....	1
<b>Methodology</b> .....	3
<b>Historical Development of Software Engineering and the Effects on Usability</b> .....	4
The Waterfall Model and its implications for usability .....	4
The requirements almost invariably do change .....	4
Project management and governments .....	6
The effect of external contracts .....	10
Summarising the effect of the Waterfall .....	12
Responses to the Waterfall .....	12
The influence of Development Methodologies .....	12
The survival of the engineering paradigm? .....	15
Where does this leave software testing? .....	17
<b>Usability Engineering and its Relationship with Software Engineering</b> .....	19
A Lack of precision, consistency and detail within HCI .....	19
Limited understanding of commercial Software Engineering .....	22
The HCI profession lacked awareness of how software applications were developed ..	23
Preece - lifecycle models, testing and requirements .....	24
Nielsen - over-ambitious claims for HCI .....	26

Mayhew - lack of integration with SE .....	28
Fallacious belief in the separability of the interface from the function of applications ..	30
Conceptual objection .....	33
Architectural objection .....	34
Managerial (organisational) objection .....	36
Inappropriate reliance on usability testing .....	37
Weak requirements and the problem of deriving the design .....	39
Lack of penetration of HCI into SE .....	43
Research on how HCI practitioners work together .....	43
Results of questionnaire .....	44
<b>The Way Ahead</b> .....	47
<b>Conclusion - What Can Test Managers Do to Improve Matters?</b> .....	52
<b>References</b> .....	57
<b>Bibliography</b> .....	67
<b>Appendices</b> .....	75
<b>Appendix A - Questionnaire</b> .....	76
<b>Appendix B - Nielsen's Ten Usability Principles</b> .....	80
<b>Appendix C - Defect Management</b> .....	82
<b>Appendix D - ISO13407 &amp; ISOTR18529 "Human-Centred Design Processes and their activities"</b> .....	83
ISO13407 - "Human-Centred Design Processes for Interactive Systems", International Organisation for Standardisation 1999 .....	83
ISOTR18529 - "Human-Centred Lifecycle Process Descriptions", International Organisation for Standardisation 1999 .....	86
<b>Appendix E - Usability Testing</b> .....	87

# Table of Diagrams, Figures, Charts

Figure 1 - Waterfall Model with limited iteration (Royce, S07, 1970)	5
Figure 2 - V Model (from Marick, S25, 2000)	17
Figure 3 - Preece's simple interaction design model (Preece, U04, 2002, p186)	24
Figure 4 - Mayhew's Usability Engineering Lifecycle (Mayhew, U122, 2006)	28
Figure 5 - Star Model (Hartson and Hix, U65, 1989, p52)	30
Figure 6- Nielsen's "scenario" model (Nielsen, U59, 1994, section 2.1)	53
Figure 7 - Usability Maturity Model ISO TR 18529 Human-centred Design Lifecycle (Process for Usability website, U108, 2003)	86
Table 1 - Human-Centred Lifecycle Process Descriptions (Earthy, U109, 2000, p14).	86
Table 2 - Different usability testing types and techniques, (Preece, U04, 2002, p347)	87

# Introduction

The aims of this dissertation are as follows;

- 1 to review the historical and current relationship between the Human Computer Interface (HCI) and Software Engineering (SE) professions, with specific regard to testing,
- 2 to identify further work that is required to improve the relationship, specifically to incorporate usability testing effectively into SE testing models,
- 3 to identify practical means by which SE test managers can improve the situation on projects that are threatened by inadequate communication between the HCI and SE disciplines.

The research soon revealed that the second aim was misconceived. There are two aspects to the problem.

Firstly, there is no such thing as a software testing model. There are differing techniques, and differing philosophies, but when one considers what is done in practice there are no testing models, only adjuncts of development models. Testing is essentially a response to development methods and is dictated by them.

Secondly, the focus on usability testing in the aims of this dissertation, and also historically within the HCI profession was based on a rather narrow assumption of how usability testing can make a significant difference to the usability of applications.

As will be explained, usability testing, where it has taken place within SE, has been forced back into the final stages of projects, where it is summative evaluation, i.e. checking the application against its requirements. In practice, this often amounts to no more than documenting the usability problems, with no hope that they will be fixed.

If usability testing is to be effective, it has to entail formative evaluation, i.e. shaping the design before it is finalised. This requires different tactics and techniques from simply testing.

Usability requires the adoption of user centred design principles throughout the development process.

These are separate issues, but they are related. In particular, the lack of independence of testing as a discipline has meant that trust in usability testing has been misplaced. SE testers have often been isolated and ineffective, a little respected community within SE. HCI inadvertently joined them in their ghetto. Also, misconceptions about the effectiveness of usability testing have perhaps made HCI too passive in the face of the dominance of the SEs in the software development process.

The two issues run as a theme through this dissertation. Sometimes they appear to be the cause of problems, but it would be more accurate to regard them as symptoms of a malaise afflicting the way that software applications have been developed. To understand this it is

necessary to go back to the early days of IT and explain the historical development of SE and HCI.

This dissertation will not discuss in any detail the techniques of usability testing. These are described in outline in Appendix E.

# Methodology

This project was conducted in three stages.

- ✘ Literature review and analysis to establish the extent to which academics and practitioners have identified the problem and sought to resolve it.
- ✘ Interviews and questionnaires (see Appendix A) with commercial SE testers and HCI professionals.
- ✘ Original work, building on the literature review, interviews and questionnaires identifying ways that test managers can try and effect improvements on individual projects where usability has not been given sufficient consideration.

23 organisations were invited to complete questionnaires. Eight were returned.

- ✘ UK insurance company
- ✘ Australian insurance company
- ✘ UK retail bank
- ✘ UK electronics manufacturer
- ✘ 2 UK IT services companies
- ✘ UK testing consultancy
- ✘ UK government department

The names of these organisations will not appear in this dissertation. The sample is so small that if the names were to appear it would be difficult to quote from the responses, or say anything about the type of organisation from which the response had come.

# Historical Development of Software Engineering and the Effects on Usability

SE is a young profession and, within that, testing has always been an immature and little respected member. Testing has not developed coherently in its own right, but in conjunction with, and as a response to, changes in SE development models. Testing has always been an adjunct of the development model, and understanding the evolution of testing is a matter of understanding how SE projects have evolved.

During the 70's and 80's there were many powerful polemics from academics and practitioners explaining how traditional approaches were ineffective and the profession **must** change. Even now some of the most fundamental advice of these writers has not permeated the mainstream - despite its highly practical nature. The main target of their frustration was the Waterfall Model.

## **The Waterfall Model and its implications for usability**

The history of SE lifecycle models can reasonably be presented as the evolution of the Waterfall Model and the reactions against it.

SE lifecycle models were slow to evolve. Until the end of the 1960s SE was essentially a practical exercise with the minimum of theoretical and academic underpinning. Of all the articles and books consulted for this dissertation the earliest useful example dates from 1970. All of the sources considering the nature of SE post-date this by at least three years.

This 1970 source is the famous paper by Royce (S07). It is a strange example of how perceptive and practical advice can be distorted and misapplied. Using the analogy of a waterfall, Royce set up a straw man model to describe how SE developments had been managed in the 1960s. He then demolished the model.

The Waterfall assumes that a project can be broken up into separate, linear phases and that the output of each phase provides the input for the next phase, thus implying that each would be largely complete before the next one would start. Although Royce did not make the point explicitly, his Waterfall assumes that requirements can, and must, be defined accurately at the first pass, before any significant progress has been made with the design.

The problems with the Waterfall, as seen by Royce, are its inability to handle changes, mainly changes forced by test defects, and the inflexibility of the model in dealing with iteration. Further flaws were later identified, but Royce's criticisms are still valid.

## **The requirements almost invariably do change**

Royce makes the point that testing is the first stage of the project where outputs are unpredictable, i.e. not the result of rigorous analysis. Yet, this is near the end of the project and if the outcome of testing is not what was predicted or wanted, then reworks to earlier stages

can wreck the project. The assumption that iteration can be safely restricted to successive stages therefore doesn't hold. Such reworking could extend right back to the early stages of design, and is not a matter of polishing up the later steps of the previous phase.

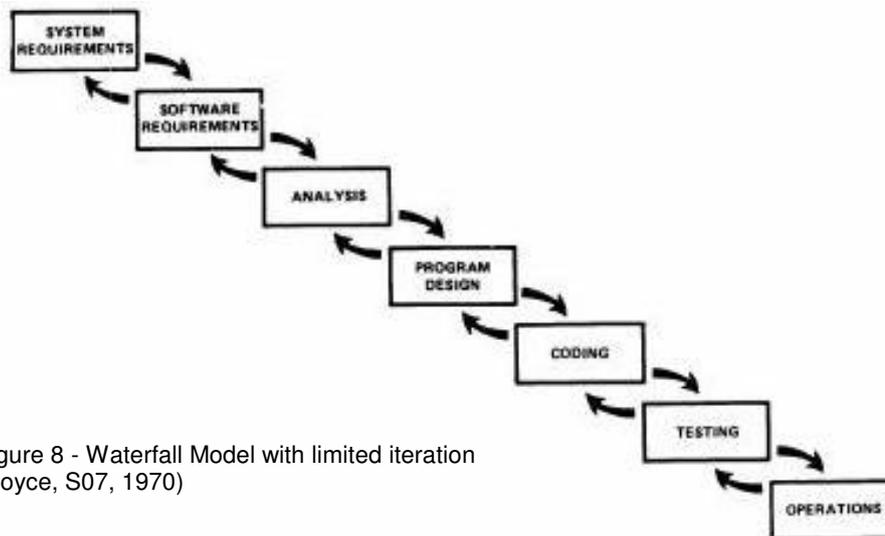


Figure 8 - Waterfall Model with limited iteration (Royce, S07, 1970)

Although Royce did not dwell on the difficulty of establishing the requirements accurately at the first pass, this problem did trouble other, later writers. They concluded that not only is it impossible in practice to define the requirements accurately before construction started, it is wrong in principle because the process of developing the application changes the users' understanding of what is desirable and possible and thus changes the requirements.

Brooks (S27, 1987) famously argued this point in his paper "No Silver Bullets".

*"The truth is, the client does not know what he wants. The client usually does not know what questions must be answered, and he has almost never thought of the problem in the detail necessary for specification. ... **in planning any software-design activity, it is necessary to allow for an extensive iteration between the client and the designer as part of the system definition.***

*..... **it is really impossible for a client, even working with a software engineer, to specify completely, precisely, and correctly the exact requirements of a modern software product before trying some versions of the product.***"

Boehm (S37, 1988) elaborates, and makes criticisms of the Waterfall that are highly relevant to the usability issue.

*"A primary source of difficulty with the Waterfall model has been its emphasis on fully elaborated documents as completion criteria for early requirements and design phases. For some classes of software, such as compilers or secure operating systems, this is the most effective way to proceed. However, it does not work well for many classes of software, particularly interactive end-user applications. Document-driven standards have pushed many projects to write elaborate specifications of poorly understood user interfaces and decision*

*support functions, followed by the design and development of large quantities of unusable code.”*

Sommerville in the latest edition of his classic textbook, “Software Engineering” (S55, 2004, p69) in discussing lifecycle models also relates the problem directly to usability in stating that *“parts of the system, such as the user interface, which are difficult to specify in advance, should always be developed using an exploratory programming approach”*, as opposed to the Waterfall.

## **Project management and governments**

So if the Waterfall was originally depicted as a strawman in 1970, and if it has been consistently savaged by academics, why is there still debate about it? Why does a contemporary, respected book like Hallows (S13, 2005, p84) still defend it?

The title of Hallows book provides part of the answer; *“Information Systems Project Management”*. The Waterfall was shaped by project management requirements, and it therefore facilitates project management.

S50 Raccoon (S50, 1996, p88) uses a revealing phrase, describing the Waterfall. *“If the Waterfall model were wrong, we would stop arguing over it. Though the Waterfall model may not describe the whole truth, it describes an interesting structure that occurs in many well-defined projects and it will continue to describe this truth for a long time to come. I expect the Waterfall model will live on for the next one hundred years and more”*.

Note the key phrase *“an interesting structure that occurs in many well-defined projects”*. The Waterfall allows project managers to structure projects neatly, and it is good for **projects not applications!**

It is often argued that in practice the Waterfall is not applied in its pure form; that there is always some limited iteration and that there is invariably some degree of overlap between the phases. Defenders of the model therefore argue that critics don't understand how it can be used effectively.

Before elaborating on these points it should be noted that Royce's model **does** assume limited iteration - between successive stages. He implies that this is a workable approach provided that the iteration takes a project back only to the previous stage, and doesn't have to extend to earlier stages. Also, his depiction of the Waterfall does **not** assume no overlap between phases.

Kruchner (S51, 2004) says *“nowadays, the waterfall is rarely 100 percent waterfall”*. Hallows (S13, 2005, p84) goes further in his guide to project management. He argues that changes are time-consuming and expensive regardless of the model followed. He dismisses the criticism that the Waterfall doesn't allow a return to an earlier stage by saying that this would be a case of an individual project manager being too inflexible, rather than a problem with the model.

An interesting feature of Hallows' defence is that it did not appear in the 1st edition (S58, 1998). However, seven years later Hallows felt impelled to make a strong defence for the continued survival of the Waterfall. This suggests that the influence of the Waterfall might be waning, but that the project management community will hang onto it for some time yet.

As evidence, consider these current quotes from commercial websites.

*"The pure waterfall performs well for products with clearly understood requirements or when working with well understood technical tools, architectures and infrastructures."* Business ESolutions (S17).

*"Provided the desired project outcomes can be clearly defined from the outset, (the waterfall) approach (i.e. the Waterfall) is swift and effective."* PA Consulting Group (S14).

*"Waterfall development makes it easy to keep your project under control. ... it's relatively easy to estimate, and it allows for greater ease in project management since plans aren't constantly being revised. The waterfall development methodology is popular with new managers because it facilitates project estimation."* Doll (S16, 2002 but available still on the UK Builder website).

All of these sources provide warnings about the weaknesses of the Waterfall. Doll in particular makes sound and balanced criticisms about the problems with quality and the difficulties in proving the design is correct. She makes a perceptive comment that *"while the waterfall methodology is easy on the manager, it can be grueling for developers."*

In spite of these caveats, It would be naive to underestimate the seductive nature of phrases like *"swift and effective"* and *"easy to keep your project under control"* and *"performs well for products with clearly understood requirements"* (and what new project manager does not intend to ensure the requirements are clearly understood?).

However, the defence mounted by Hallows and the other sources is highly questionable. The Waterfall is fundamentally flawed because it ignores the reality of SE;

- ✘ the requirements almost invariably **do** change in mid-stream, especially for user interfaces, and
- ✘ the politics of project management, and contractual constraints, make unplanned returns to earlier stages extremely difficult.

Hallows requires further examination. He claimed that any difficulty in returning to an earlier stage would be caused by inflexible project management. This is naive. The problem is not solved by better project management, because project management itself has contributed towards the problem; or rather the response of rational project managers to the pressures facing them.

The dangerous influence of project management had been recognised by the early 1980s. Practitioners had always been contorting development practices to fit their project management model, a point argued forcibly by McCracken & Jackson (S10, 1982). *"Any form of life cycle is a project management structure imposed on system development. To contend*

*that any life cycle scheme, even with variations, can be applied to all system development is either to fly in the face of reality or to assume a life cycle so rudimentary as to be vacuous.”*

In hindsight the tone of McCracken & Jackson’s paper, and the lack of response to it for years is reminiscent of Old Testament prophets raging in the wilderness. They were right, but ignored.

The symbiosis between project management and the Waterfall has meant that practitioners have frequently been compelled to employ methods that they knew were ineffective. This is most graphically illustrated by the UK government’s mandated use of the PRINCE2 project management method and SSADM development methodology. These two go hand in hand. They are not necessarily flawed, and this dissertation does not have room for a discussion of their merits and problems, but they **are** associated with a traditional approach such as the Waterfall.

The National Audit Office’s statement (S19, 2003, p5) that *“PRINCE requires a project to be organised into a number of discrete stages, each of which is expected to deliver end products which meet defined quality requirements. Progress to the next stage of the project depends on successfully meeting the delivery targets for the current stage. The methodology fits particularly well with the “waterfall” approach.”*

The NAO says in the same paper (p3) that *“the waterfall ... remains the preferred approach for developing systems where it is very important to get the specification exactly right (for example, in processes which are computationally complex or safety critical)”*. This advice is still current at the time of writing. The public sector tends to be more risk averse than the private sector. If auditors say an approach is “preferred” then it would take a bold and confident project manager to reject that advice.

This official advice is offered in spite of the persistent criticisms that it is never possible to define the requirements precisely in advance in the style assumed by the Waterfall model, and that attempting to do so is possible only if one is prepared to steamroller the users into accepting a system that doesn’t satisfy their goals.

The UK government is therefore promoting the use of a project management method partly because it fits well with a lifecycle that is fundamentally flawed because it has been shaped by project management needs rather than those of software development.

The history of the Waterfall in the USA illustrates its durability and provides a further insight into how it will survive for some time to come; its neat fit with commercial procurement practices.

As noted by Fayad et al (S62, 2000, p115) the US military was the main customer for large-scale software development contracts in the 1970s. *“Since then, virtually all software engineering literature has concentrated explicitly and implicitly on the model of DoD contract software development.”* The Department of Defense (DoD) insisted on formal and rigorous

development methods, and Fitzgerald (S12, 1996, pp8-9) argues convincingly that the pressure from the DoD influenced the techniques used by SEs.

A specific DoD requirement was that software contractors had to comply with the 1975 Standard DOD-STD-2167 (S61, 1975) and Directive 5000.29. These did not explicitly mandate the Waterfall, but their insistence on a staged development approach, with a heavy up-front documentation overhead, and formal review and sign-off of all deliverables at the end of each stage effectively ruled out any alternative to the Waterfall. The reasons for this were quite explicitly to help the DoD to keep control of procurement.

Curiously, the standard did mandate the use of *“applicable human factors engineering principles”* (S61, p23), but without setting the requirement clearly into the context of the lifecycle, and while the DoD were effectively insisting on the use of the Waterfall, this gesture towards HCI was no more than window dressing.

DOD-STD-2167A (1988) relaxed the requirements and allowed iterative developments. However, it did not forbid the Waterfall, and the standard’s continued insistence on formal reviews and audits that were clearly consistent with the Waterfall gave developers and contractors every incentive to stick with that model.

The damaging effects of this were clearly identified to the DoD at the time. A report of the Defense Science Board Task Force (S46 abstract, 1987) released just a few months before the updated DOD-STD-2167A criticised the effects of DOD-STD-2167 and complained that the draft version of the update *“does not go nearly far enough . ... it continues to reinforce exactly the document-driven, specify-then-build approach that lies at the heart of so many DoD software problems”*, (p38).

The Task Force’s report made clear that usability was one of the victims of this approach. It states, as quoted by Glass (S29, 2006, p19).

*“The hardest part of the software task is the setting of the exact requirements. ... Misjudgements abound. ... (A) common error is the mis-imagination of how user interfaces would work. In our view, the difficulty is fundamental. We believe that users cannot ... accurately describe the operational requirements for a substantial software system without testing by real operators in an operational environment, and iteration on the specification. The systems built today are just too complex for the mind of man to foresee all the ramifications purely by the exercise of analytic imagination.”*

However, the report is realistic in acknowledging that *“evolutionary development plays havoc with the customary forms of competitive procurement, ... and they with it. Creativity in acquisition is now needed”*. (Glass, S29, p21).

The report provides such creativity (S46 abstract, 1987, p4); *“for major new software builds, we recommend that competitive level-of-effort contracts be routinely let for determining specifications and preparing an early prototype ... The work of specification is so crucial, and yet its fraction of total cost is so small, that we believe duplication in this phase will save*

*money in total program cost, and surely save time. After a converged specification has been prepared and validated by prototyping, a single competitively-bid contract for construction is appropriate .”*

The Task Force contained such illustrious experts as Frederick Brooks, Vic Basili and Barry Boehm. These were reputable insiders, part of a DoD Task Force, not iconoclastic academic rebels. They knew the problems caused by the Waterfall, they understood that the rigid structure of the model provided comfort and reassurance that large projects involving massive amounts of public money were under control. They therefore recommended appropriate remedies .They were ignored. Such was the grip of the Waterfall nearly 20 years after it had first been criticised by Royce.

The DoD did not finally abandon the Waterfall till Military Standard 498 (MIL-STD-498) seven years later in 1994, by which time the Waterfall was embedded in the very soul of the SE profession.

### **The effect of external contracts**

As the DOD-STD-2167A report noted (see Glass, S29, in previous section) evolutionary development played havoc with traditional procurement practices, which fitted much more comfortably with the Waterfall.

This raises the question of whether external suppliers are subjected to greater, and different pressures, than internal IT developers, and whether this leads to poorer usability.

This dissertation argues that external suppliers are far less likely to be able to respond to the users real needs, and the research supports this. The contractual relationship between client and supplier reinforces the rigid project management style of development.

Newberry, a US Air Force officer, (S82, 1995) explains the need to deliver mind-numbing amounts of documentation in US defence projects. *“DOD-STD-2167A imposes formal reviews and audits that emphasize the waterfall model and are often nonproductive “dog and pony shows.” The developer spends thousands of staff-hours preparing special materials for these meetings, and the acquirer is then swamped by information overload.”*

This is a scenario familiar to any SE who has worked on external contracts, especially in the public sector. Payments are tied to the formal reviews and dependent on the production of satisfactory documentation. The danger is that supplier staff become fixated on the production of the material that pays their wages, rather than the real substance of the project.

Lewis & Rieman (U104, 1994, ch2.2) claimed cynically, but plausibly, that external suppliers are much less likely than internal SEs to take Participatory, or User-Centred, Design on board. *“At bottom, your objective as a commercial developer may really not be to improve the quality of somebody else's work life, but rather to make money for yourself. So you don't have the right goal to begin with.”*

There have been relatively few academic papers discussing whether external suppliers deliver poorer usability. Artman (U81, 2002, pp61-62) argued that this is the case.

*“Procurers should require usability. However, they seldom do. ... If the contract does not contain explicit requirements for usability, it is generally one of the first considerations to be cut if time or finances are constrained. When time is short, the contractor fulfils only those requirements specified in the procurement agreement.”*

Holmlid & Artman (U80, 2003, p1) looked at this issue in three projects where usability was a major initial concern, and found evidence of *“an asymmetry often assumed in system development, that the procurer should understand, or more specifically, adapt to the supplier. We think that this assumed asymmetry is the problem, rather than the symptom.”*

In practice, in the projects examined this meant that the customers found themselves being forced to accept more linear, less responsive, less iterative development processes than they wished, or expected, with damaging results for usability.

This is consistent with the findings of Grudin in a series of papers in the 90s. He argued (U85, 1996, p170) that *“users and developers are in distinct organizations. They are typically geographically separated, face legal barriers to communication, and may find it impractical to renegotiate a contract to exploit user feedback.”*

Grudin also argued (U84, 1995, p64) that *“contract compliance is based on conformance to the written specifications. Any deviation from the specification is to be avoided: Programmers have been known to code nonfunctioning procedures described in miswritten specifications to avoid jeopardizing compliance. ... Usability requirements in contracts for interactive systems can be as vague as ‘the system shall present information succinctly’ or ‘the system shall be easy to operate efficiently.’ Some contracts require that prototypes be demonstrated to users yet preclude or discourage changes based on feedback from the demonstrations.”*

An interesting observation from Grudin (U85, p169) was that whereas most of the industrial HCI researchers in the USA came from a background of developing PCs and consumer products, the Scandinavian participatory design techniques (discussed later) came out of the in-house development context that is prevalent in Europe.

In 1995 a workshop was held by Association for Computing Machinery's Special Interest Group on Computer-Human Interaction to consider the challenges of introducing usability to US government developments. In addition to the general problems faced by all IT projects the participants concluded (U87, Winkler & Buie, 1995, p35);

*“Historically, very few (tender invitations) have specified HCI features or usability activities beyond such generalities as ‘Motif compliant GUI’ or ‘user friendly interface’. Bidders hesitate to increase their proposed price by including features and activities that ... (the invitation) does not mandate, and contractors may hesitate to add them even after winning, for fear of gaining a reputation for cost increases and overruns.”*

They found a further problem that *“many contracts require that team members have degrees in hard science or engineering; psychology doesn't count. This increases the difficulty of building a team with sufficient HCI/usability expertise.”*

Effective usability engineering must require iteration, which in turn must require flexible contracts with estimates and costs being revised repeatedly. This is a frightening prospect for both sides; with the supplier scared of being committed to a job which cannot be sized effectively, and the client equally scared of being tied into a contract with no cost cap, and no easy exit without writing off the work already paid for.

Nevertheless, as noted earlier, the UK public sector is still firmly wedded to the waterfall method and this must surely bias contracts against a commitment to usability.

## **Summarising the effect of the Waterfall**

The significance of this for usability was that the development process followed by default ensured that the following invariably happened.

- ✘ Requirements were not adequately specified and changes to requirements could not be handled easily. As requirements affecting the interface are particularly difficult to get right, usability was badly affected.
- ✘ Iteration was severely limited, which made it difficult to produce a satisfactory interface.
- ✘ Testing was too late and rushed to be effective, and there was little opportunity, or will, to correct those usability issues that were detected. This will be expanded upon in “Where does this leave software testing?”, (page 17).

These combined to deliver applications which were of low quality, and in particular with poor usability. The Waterfall model may never have enjoyed intellectual credibility, but it has proven astonishingly resilient in practice, and has proven highly effective in thwarting the aspirations of HCI professionals, and indeed users. None of the respondents to the questionnaire for this dissertation denied using the Waterfall. It is still very much one of the tools of the SE trade.

The Waterfall may have resulted in bad systems, but its massive saving grace for companies and governments alike was that they were developed in projects that offered at least the illusion of being manageable! This suggests, as Racoon stated (S50, 1996, p88) that the Waterfall may yet survive another hundred years.

## **Responses to the Waterfall**

### **The Influence of Development Methodologies**

It is important to remember the distinction between lifecycle models and development methods. The terminology varies depending on the time and the writer, but the distinction essentially remains constant; as described by Boehm (S37, 1988, p61).

*“The primary functions of a software process model are to determine the order of the stages involved in software development and evolution and to establish the transition criteria for progressing from one stage to the next” whereas “a method’s primary focus is on how to navigate through each phase”.*

Logically, and ideally, it should be only the development methods that are relevant to usability. However, it is a major argument of this dissertation that the lifecycle used has a profound influence on usability, but only in a negative sense. The Waterfall has had a significant, damaging effect on usability, but if one dispenses with this model and uses more appropriate, iterative models then usability is more easily achieved, but is certainly not inevitable.

If one turns one’s attention from the lifecycle models to development methods then a similar pattern becomes clear. The Waterfall arose in part from the wish of the SE profession to see software development as being essentially the same as construction engineering. The history of development methods reflects this desire that SE should be a more formal, scientific and rational profession.

We have seen that project management was distorting the lifecycle model. Did this mislead academics into believing that what they saw happening represented what *should* be happening?

Critics of traditional SE approaches would argue that academics did misunderstand the nature of software development, regarding it as a more precise and orderly process than it really was.

The academics saw practitioners managing projects with a structured, orderly process that superficially resembled civil engineering or construction management, and that fitted their perception of what SE **ought** to be.

Academics built conceptual models on this misunderstanding. This was damaging in that it reinforced development practices which were dysfunctional, certainly from the usability perspective. The misconceived conceptual models were used as the basis for development methods which could be adopted readily because they were consistent with existing development practices, centred around the concept of a Systems Development Lifecycle, that was phased, structured and manageable - essentially the Waterfall Model.

It would be highly desirable if SE were a more rational, predictable and controllable process, but the truth is that it is not. The search for the key to understanding SE as just such a process has frequently been compared to a man searching for his wallet in the dark under a lamppost (e.g. Curtis et al, S57, 1987, p96); not because he dropped it there, but because that’s where the light is.

Fitzgerald argued in a very perceptive paper (S09, 1994, p2) that when academics were trying to build a conceptual model for SE, they worked backwards from what they observed being done in practice in order to construct their model. Referring to Floyd (S52, 1987), Fitzgerald says *“operating from an ontological position of realism, systems development is conceptualised as a linear model, relying on phases ordered in time and leading to*

*increasingly-formal defining documents until the software system eventually emerges”.*

Ironically, Floyd was trying to provide a conceptual basis for contemporary methods of SE as a precursor to advocating a paradigm shift towards user-centred design, but the observation by Fitzgerald is still valid.

Fitzgerald backs up his argument by citing the influential Swedish academic Börje Langefors (S53, 1973) who (p2) *“conceptualised systems development as a rational and scientific process”*. Fitzgerald also cites other authorities who shared his own insight, such as Oleron (S54, 1991) and Argots (S56, 1986).

Fitzgerald goes on to argue (p6) that *“systems development in practice is not an orderly systematic phased process, rather it happens ‘all at once’ (DeGrace & Stahl, S32, 1993). **In the Waterfall life-cycle the what of requirements specification is strictly separated from the how of design.** Yet, as Peters (S33, 1981) bluntly puts it: ‘one cannot state a problem without some rudimentary notion of what the solution should be’. Shemer (S31, 1987) suggests a jigsaw analogy. He argues that the ultimate design is not achieved in a rational top-down manner. Rather, information is obtained in random order; some design is done bottom-up by guessing at a local pattern, and, simultaneously, some design is done top-down by following an apparent pattern. A study by Zelkowitz (S34, 1988) lends validity to this, reporting that 50 percent of design activity occurs in phases other than the design phase of the waterfall life-cycle.”*

This is a vital passage. The contrast and the links between the “what” and the “how” is fundamental to the usability problem that is the concern of this dissertation. The issue being considered in Fitzgerald’s paper is the contrast between the logical functions of the system and the technical implementation. SE’s do not move naturally and smoothly from abstract requirements, devoid of any preconceptions about the implementation, to a coherent design. Yet that is what they are encouraged to do by traditional methods.

Bansler & Bødker (S47, 1993, p169) argue that structured development methods reduce humans to being just another component ; *“people are made into objects, simply perceived as ‘system components,’ comparable with tools, machines, and raw materials. In Structured Analysis one does not distinguish between the way people act and the way machines function, they are all alike from the method’s point of view.”*

This sheds light on the way formal consideration of physical design can be pushed aside during systems analysis. This frequently includes usability issues, because the implicit assumption has been that these are questions for the physical design - by which time the users are excluded. They had their say during requirements definition.

The focus during a traditional, structured development was on the function, an objective concept, rather than quality, which is inherently more subjective and harder to specify. Function was a matter of logical relationships, data flows and business rules. It was emphatically not a matter of how the user interacted with the application.

It is clear that usability is always likely to be a victim of traditional methods. However, to complicate matters Bansler & Bødker (S47, p182) argue that *“what happens in practice is that experienced designers-instead of following the rules and procedures of Structured Analysis-pick and choose, ... and integrate them into their own design processes. .... The designers (therefore) avoid or compensate for some of the method’s deficiencies and limitations, especially with respect to describing the user-interface and the human-computer interaction.*

The situation is therefore more complex than saying that the Waterfall and its associated traditional methods are based on unsound assumptions and are damaging in practice. That may well be strictly true, but it would be more accurate and constructive to argue that they don’t correspond precisely to what is done at the coal face. Practitioners recognise the flaws with the models and methods, and try to mitigate them.

### **The survival of the engineering paradigm?**

Although most academic observers realised that existing practices were producing poor quality systems it was largely assumed that this was because these practices were insufficiently rigorous and methodical. Although the academics unanimously denounced the construction engineering paradigm in its lifecycle manifestation, i.e. the Waterfall, they did not seem to question whether this paradigm was really appropriate for development methods.

The academics were therefore largely unsuccessful in attempting to wean the SE profession of the Waterfall, but they were more successful in trying to promote formal, structured development methods. They had been working against the grain of the SE profession in attacking the Waterfall. When they advocated structured methods they had been working with the grain.

This is not to say that structured methods were adopted enthusiastically and universally. They were not. However, they were accepted in principle, and there was a willingness to try and adopt them. The process was slow and painful for the SE profession; but the end was not seriously questioned.

It seems plausible to argue that SE focussed on trying to make the whole development process more engineering like, and missed the point about usability.

Fitzgerald argued somewhat cynically (S11, 1995, p6) that the move towards more formal methods led to goal displacement.

*“There are a number of reasons underpinning the goal displacement phenomenon. Perhaps, the main one is the fact that development is a complex stressful process (Wastell & Newman, S83, 1993), and any rational prescriptive mechanism - as a methodology generally is - which purports to provide some comfort and reassurance that this complexity can be addressed, will be seized upon by management and developers alike. There is little concern as to whether the methodology acts more as a placebo than a panacea, and a subtle conspiracy takes place*

*with developers and management taking comfort from the fact that a rational methodological approach is being followed.*

*... The development of the actual system becomes almost an afterthought ... as developers concentrate on blind and slavish adherence to the methodology instead."*

SEs had to adopt formal methods to appear professional and win business. They may not have really believed in their efficacy, but it was reassuring to be able to follow an orderly process. They saw the role of these methods as providing commercial advantage rather than building better applications.

Commercial advantage was unquestionably one of the main reasons behind the increasing popularity of the Carnegie Mellon Software Engineering Institute's Capability Maturity Model Integration (CMMI) accreditation scheme.

Yourdon (S64, 1998, p57) commented that *"the CMM doesn't indicate whether an organisation has the right process; it simply indicates whether the IT organisation has some kind of process that's well-defined, repeatable, measure, and in a constant state of evolutionary improvement. But this says nothing about the overall needs of the enterprise: In theory, one could imagine a level-5 SEI organisation with a near-perfect development process but which still fails to meet the needs of the organisation."*

It is interesting to note that Yourdon, one of the proponents of Structured Methods, was taking a nuanced view on the need for structure and formality by the late 90s, especially in relation to usability.

(S63, 1997, p145). *"The ... team needs to agree on which processes will be formalised - perhaps source code control and change management, and, hopefully, requirements management - and which processes will be carried out in a completely ad hoc basis, e.g. user interface design."*

In S64 (p58) he asks; *"does it make sense, (the CMM critics) ask, to focus so much of our effort on formalising and institutionalising a software process in an industry where all the rules change so quickly and radically? ...."*

*The SI model has been largely influenced by the needs of the US Defenso Department, which desperately needs to improve its track record for developing large, complex systems; and it's also true that the SEI-CMM is based on a classical quality control paradigm of defining and standardising a process, then measuring it to reduce variance, and then applying incremental improvements."*

Yourdon argues that business is fast-moving and chaotic, and IT must be nimble enough to support that. However, the choice between undisciplined anarchy and rigid, bureaucratic management is a false one. SEs are accustomed to working pragmatically; doing what they have to do - what works for them. This means flexibility in the early stages of the development process, and greater formality in the testing and implementation stages.

Yourdon continues (p59); *“Silicon Valley has become painfully aware of the consequences of shipping buggy, low-quality software products to a mass marketplace; consequently many of these companies have instituted, very formal processes for testing, quality assurance, and configuration management of their product releases. But they retain a great deal of flexibility at the front end of the software process, where product requirements and features are discussed, debated and constantly changed in an innovative environment.”*

However, the engineering paradigm is not dead. The claims are still heard, and they are usually unconvincing, like a shoe shop regarding the shoe horn as a step forward in footwear retailing. Shoes can be forced onto feet more quickly and efficiently. The customers aren't any more likely to be leaving the shop with the right pair of shoes.

Nevertheless, there are interesting and possibly useful variations. In the November 2006 edition of the BCS magazine “IT Now” an article by Briton (S71) argued in somewhat contentious style that SE should mimic civil engineering more closely. This is backed up by a deeper, more thoughtful and persuasive paper on Briton's consultancy website (S72, 2006) which argues that SE **does** still have much to learn from civil engineering, particularly in the way that it moves from requirements to an outline design, which can be verified at an early stage. As we shall see later, this is a topic of vital concern to usability.

## Where does this leave software testing?

The title of this dissertation was misconceived. There is no such thing as a software testing model. There are differing techniques, and differing philosophies, but when one considers the history of SE and lifecycle models, it becomes clear that in practice there are no testing models, only adjuncts of development models. Testing is essentially a response to development methods and is dictated by them.

Marick (S25, 2000) essentially makes the same point; *“most software testing models are bad ones. They're bad because they're mere embellishments on software development models. People think hard and argue at length about how to do software development. They invent a model. Then they add testing as an afterthought.”*

This criticism applies specifically to the V Model.

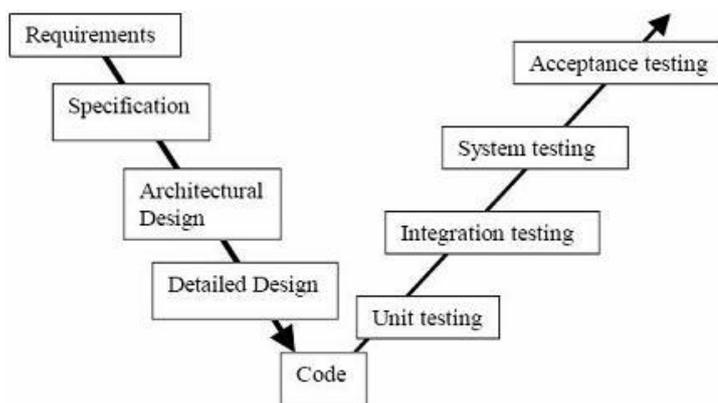


Figure 9 - V Model (from Marick, S25)

In reality, the V Model is no more than a different perspective on the waterfall to help testers mitigate some of the damage that the model causes.

The V Model assumes that developers will be using the Waterfall. Testers must get involve at an early stage, in order to plan testing, and influence the development. When requirements are being agreed, the parallel testing activity of planning the acceptance testing takes place. When the system specification (i.e. the system design) is being written, the system testing is being planned.

Pyhäjärvi et al (S49, 2003, p2) reported that *“in the field of software testing, the V-model is the state-of-the-art taught on practically every course on testing”*.

Certainly the British Computer Society is wedded to the V Model. The testing standard, BS7925, written by the BCS, and the syllabus of the BCS’s ISEB Foundation Certificate in Software Testing assume that the only lifecycle models are the Waterfall and the V model, which, arguably, are essentially the same thing.

A further Interesting illustration of the lack of influence that SE advances have had on testing is provided by the BCS’s Special Interest Group in Software Testing. SIGIST focusses on the detail of testing techniques, with no apparent awareness of the bigger picture of how different lifecycle models and development methods enable or constrain effective testing.

The SIGIST website presents usability testing as just another form of non-functional testing. It is treated as a standalone technique. There is no consideration of how, or whether, it can be incorporated into the wider testing exercise. Nor is there any discussion of the need to ensure usability is built into the earlier stages of development.

In truth, handling usability testing in this way is no more than damage limitation. However, HCI academics and practitioners seem to have been complicit in this, as we shall discuss later on.

# Usability Engineering and its Relationship with Software Engineering

## **A lack of precision, consistency and detail within HCI**

Usability was meanwhile evolving in parallel, but with insufficient understanding of the reality of software development. From the perspective of this dissertation, the question is less a matter of how usability testing has developed, and more a question of how HCI professionals and usability engineering have adapted uneasily to changes in SE, and how their work has done little to inspire confidence in sceptical SEs. Indeed, the focus on “usability testing” in the title of the dissertation as originally envisaged now looks misplaced.

Attempting to construct a narrative of the development of HCI reveals one of its features; the discipline’s lack of unity and cohesion.

In trying to understand the development of an idea or profession it is common to define generations, or paradigms. A cursory examination of HCI’s history reveals three or more generations. There seems little agreement about what these generations are, or whether they should be based on the physical attributes of interfaces, the underlying ideas motivating innovators, or the context of use.

Grier (U133, 2005, p1) defines the generations in terms of the interfaces.

- ✘ 1st generation was punch card input.
- ✘ 2nd was the command based interface (e.g. DOS).
- ✘ 3rd (and current) is the graphical user interface, “*most often instantiated as a WIMP (Window, Icon, Menu, Pointer)*”.

Jacob (U135, 2006, p1) also defined three generations so far, but not the same set.

- ✘ 1st generation was the command-line interface.
- ✘ 2nd was direct manipulation.
- ✘ 3rd was the graphical user interface (the current state).

Bødker (U121, 2006, p1) defined the generations in terms of the context of use and the underlying principles.

- ✘ 1st generation was as defined by Bannon (U134, 1991) who said that it focussed on “*the individual user working on a computer system (and) totally neglects the importance of coordination and cooperation between work processes that is necessary in many work situations.*”
- ✘ 2nd generation saw the introduction of usability to the workplace with the focus on groups working with a collection of corporate applications.
- ✘ 3rd generation concerns the different challenges of introducing it to applications to be used by the public and at home, with the focus returning to individuals.

Nielsen (U03, 1993, p50) confusingly numbers his generations from zero. He had reached 6 generations by 1993, and it is interesting to note that he thought that the profession was on the verge of moving beyond the WIMP paradigm, whereas Grier and Jacob thought that was still the current position, 12 years later.

- ✘ Generation 0 describes direct access to the hardware, i.e. no interface, up to 1945.
- ✘ Generation 1 was batch programming; again no interface, from 1945 to 1955.
- ✘ Generation 2 was on-line command line interfaces, from 1955 to 1965.
- ✘ Generation 3 was mainframe full-screen, hierarchical menus for commercial in-house applications, from 1965 to 1980.
- ✘ Generation 4 was WIMP, from 1980 to 1995
- ✘ Generation 5 was envisaged as being non command-based interfaces, to run from 1995.

Nielsen showed he'd lost touch with what the mass of the SE profession were doing when he defined these generations. His historical analysis may be correct in his categorisation, and in the sequence of his narrative, but he misses what is actually happening at each stage of the development of IT. He comments on the leading edge, as if it is the mainstream.

He states that generation 3 was the era of commercial, mainframe, on-line applications, and that generation 4 was characterised by the takeover of offices by desktop PCs. This is reasonable till one considers the dates. Generation 3 ran from 1965 to 1980, and generation 4 from 1980 to 1995. Nielsen is describing the leading edge in each case, and presenting it as if it is the norm.

The period 1980 to 1995 saw the mass introduction of essentially generation 3 interfaces, perhaps with generation 4 technology, as almost the entire western white collar workforce changed working patterns and got dumb terminals or PCs on their desks.

Nielsen then misreads what would happen from the mid-90s onwards when he sees generation 5 providing ubiquitous computing to everyone. In reality, the great development of this period was the internet, and though there was a massive extension of computer ownership and use, this was using interfaces Nielsen described as being generation 4.

In the 21st century mainframe applications were still being developed using "obsolete" mainframe technology such as CICS/Cobol, simply because some corporations see it as effective for the job they want to do. A quick check of internet recruitment sites shows that people are still being hired for such work.

The lack of cohesion and clarity extended to the methods and techniques that were developed in HCI. Similar phrases recur throughout the literature, but the same terms can mean different things when used by different writers; and differing terms can refer to identical concepts.

For instance, livari (U125, 2006, p185) acknowledges that the same group of HCI practitioners have been called "user centred designers" and "usability engineers", but argues that these are

essentially different concepts, the former being creative and the latter more formal and rigorous.

On the other hand, ISO13407, "Human-centred design processes for interactive systems", (U07, 1999) is cited as the prime reference for User-Centric Design by both Antilla et al (U129 2002, p4) and Pekkola et al (U126, 2006, p21). This illustrates how fuzzy HCI can be.

User-Centric Design is not mentioned in ISO13407. "Human-centred design processes" in ISO13407 is synonymous with "User-Centric Design", in the eyes of some experts at least.

In addition to Human-centred/User-Centric Design, we have Participatory (or Participative) Design and Co-operative Design. Schuler (U127, 2006) describes Participatory Design as *"the general philosophy of empowering the end user through co-design"*. Co-operative Design, according to Bannon (U131, 1995, p67) is *"a concept that needs to be, and is being, fleshed out."* It seems unlikely that it ever was, given the findings of Pekkola et al (U126, 2006, p121), see below.

Bannon continues, making the crucial point from the SE perspective; *"what can practising professional designers take from these studies and apply, if there is no clear method being defined?"*. Unfortunately Bannon does not specify the studies to which he's referring, but they are clearly the work of Kyng and others at Denmark's Aarhus University in the early 1990s, (e.g. U134, U136, U137).

It is beyond the scope of this dissertation to tease out the differences, similarities and overlaps between these different terms and concepts. The point is that few within the SE profession would take the trouble to do so either, especially when it is hard to discern clear techniques amidst these concepts.

The lack of consistency, detail and precision within HCI (as perceived by SE) has been a serious drawback for a profession with a pressing need to be accepted and respected by SE, a profession that was meanwhile trying to make itself more formal, rigorous and objective. HCI would have seemed inconsistent and imprecise to those SEs who were interested. They would have been further concerned by the inability of HCI to establish exactly how its practices should hook into SE methods.

This is implied by the observation of Antilla et al (U129, 2002, p4) that ISO13407 is *"not bound to any specific techniques or methods. ISO13407 cannot be regarded as a theory. It represents the prevailing and generally accepted understanding of how usable products are created in the community of usability engineers and scientists"*. This is not to dismiss ISO13407, far from it. The approach it takes is probably correct. A detailed, prescriptive approach would be inappropriate. However, as it stands it is unattractive to any SEs who are looking for a route map to guide them through unfamiliar territory.

The point is made more explicitly by Pekkola et al (U126, p121) as recently as August 2006. *"Currently, there is a gap between methodologies addressing user participation and information systems development (ISD) methodologies. Roughly, user-oriented*

*methodologies (e.g. participatory design, user-centric design, co-operative design and JAD) discuss how to involve users in the systems design. Yet they do not explicitly point out the relevant phases of the ISD process nor how exactly the user involvement should take place.”*

Iversen et al (U130, 2004, p171) describe the result picturesquely. *“25 years ago brave ‘sailors’ set out for ‘Utopia’ in system development. They attempted to give end users a voice in design of computer supported work places. This journey had a strong impact on following generations of research in HCI. Some researchers adapted the co-operative design approach, others reshaped it to fit local requirements and changing times, and later generations romanced about returning to the Utopian ideals. In spite of these efforts, co-operative design has had little impact on the industrial development of IT artefacts outside research in HCI and co-operative design.”*

Bannon (U131, 1995, p67) in commenting on Kyng’s Aarhus studies notes that *“the actual framework within which the design process is conducted is not usually one with the constraints---time, money, effort, contractual obligations---of commercial software development.”*

This observation by Bannon is key to understanding the attitude of the more thoughtful, receptive side of SE to HCI. The more blinkered side of SE has no view on HCI because it is lamentably ignorant about the subject (see the discussion of the questionnaires, page 44), but the receptive element, even if it recognises that HCI offers a wealth of insight, experience and advice, is frustrated and disappointed by a lack of practical awareness of the realities of commercial SE and the needs of its practitioners.

## **Limited understanding of commercial Software Engineering**

HCI’s lack of awareness extends to the work of some of the profession’s most respected authorities; Nielsen (U03, 1993), Mayhew (U02, 1999), Hartson and Hix (U65, 1989 and U105I 1993), and Rubin (U01, 1994). This is a serious concern, not just because they are highly influential within HCI, but also because they are likely to be the first sources consulted by SEs trying to find out about HCI. The result is likely to be scepticism and disappointment, and a reluctance to take on board the vital lessons that these authorities **do** have.

This failing manifested itself in two broad trends.

- ✘ A failure to understand the needs of SE practitioners and how applications were developed.

This was combined with a naive and ill-informed assumption that if applications were developed following user centred design principles then that would resolve the problem. This resulted in HCI professionals being marginalised and ignored by the SE profession.

- ✘ A misguided belief that interfaces could be separated from the functional heart of applications.

HCI academics assumed that the detailed functions of applications could be left to the technical developers, and the HCI practitioners could meanwhile perfect the interface iteratively. The resulting interface could be grafted on to whatever functional application the software engineers produced.

There was inappropriate reliance on usability testing, at the expense of other usability techniques, as though testing alone could make a significant difference to the usability of applications; an assumption which was largely dependent on the belief that function and interface were separable.

Tied in to this, was a failure to come to terms with the move from requirements to design in SE; a failure which they shared with many SE academics and authorities. This problem, combined with poor or missing usability requirements, added to the pressure to rely on usability testing at the end of projects.

## **The HCI profession lacked awareness of how software applications were developed**

HCI academics have tended towards a partial and naive understanding of how software applications are developed. Authorities like Preece, Nielsen, Hartson and Hix, and Rubin provide excellent and thought provoking expositions of usability issues, but look out of their depth when they try to analyse problems from an SE perspective. Much of their commentaries are either truisms or superficial. They then move back onto the more comfortable territory of HCI techniques. This has left them floundering and criticising on the sidelines, unable to influence the action on the field, and largely ignored by the combatants.

Gulliksen & Boivie (U10, 2001, p8) commented, following a workshop attended by academics looking at the integration of usability into SE. *“Some of the participants argued that presently, there are no textbooks on usability that cater to the needs and background knowledge of software engineers. Is it so, perhaps, that few within the HCI community have enough knowledge about software engineering to address the issues and problems with usability that pertain to engineering? In the HCI community, usability is the major aspect within software engineering, requiring special attention, expertise and methods based on, for instance, psychology and ethnography. For the software engineer, usability is one aspect out of many that must be taken into account.”*

Preece et al (U117, 1994, p359) say that *“SE models are primarily oriented towards the development of large software systems with a focus on system functionality. By contrast, the field of HCI has established user-centred design because it recognises the importance of frequent user testing using informal representations as well as computer-based prototyping”*

They are quite correct in their comments about SE. However, whilst HCI was thinking in terms of supplanting SE with its own preferred models and methods it was doomed to remain on the margins. They needed to change SE, and that required a greater commitment to learning about SE and understanding its problems than HCI experts seem prepared to invest. I will demonstrate this by reference to what are probably the main works by four of the leading lights of HCI; Preece, Rubin, Nielsen and Mayhew. Their narrow perspective, their excessive optimism, and the unwordliness of their views on SE surely contributed towards the huge potential value of HCI being ignored by any SEs who took the trouble to find out about it.

### **Preece - lifecycle models, testing and requirements**

*“Interaction Design”* by Preece et al (U04, 2002) has become a staple of university HCI courses in the last few years. Yet it shows a poor awareness of SE. Preece and her colleagues discuss SE lifecycle models, concentrating on the Waterfall model, but have clearly not read Royce’s paper (S07, 1970). *“The waterfall model was first proposed in 1970”* (U04, p187), and *“some feedback to earlier stages was acknowledged as desirable and indeed practical after this lifecycle became widely used... But the idea of iteration was not embedded in the waterfall’s philosophy”* (p188).

The Waterfall was described not “proposed” by Royce, whose paper assumed that iteration did take place. Iteration was **not** added afterwards. Royce made it clear that *“design iterations are never confined to the successive steps”* (S07, p330).

Preece et al provide their own attempt at a lifecycle model, which they claim (U04, 2002, p186) *“has its roots in ... SE and HCI lifecycle models”*.

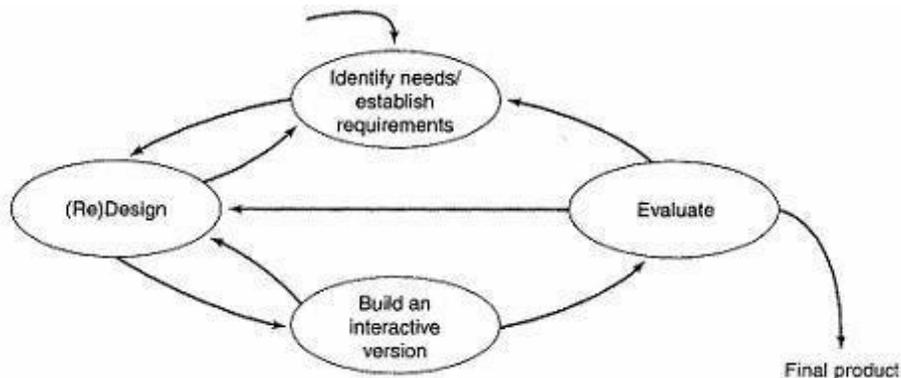


Figure 10 - Preece's simple interaction design model (U04, 2002, p186)

Even allowing for their admission that this is a simple model, it is clearly lacking in some respects. It doesn’t answer one of the main criticisms of the Waterfall Model. The model assumes no evaluation takes place until an interactive version has been built, i.e. till the stage when ones hopes to have a finished product. This is a dangerously wasteful approach. Preece et al say *“the only factor limiting the number of times through the cycle is the resources available”*. That may be true, but the comment betrays considerable naivety. The resources available are always likely to be constrained in practice, and the pressure to reduce the

number of iterations will be huge on a commercial SE project. If resources are wasted because evaluation doesn't take place during the requirements and design stages then the number of full iterations will be reduced with a resulting loss of quality.

The treatment of evaluation by Preece et al in U04 is also weak. They propose a framework for usability evaluation of products, called DECIDE (pp348-356). The context makes it clear that this is meant to cover software applications. Their sole reference to the field of SE is a paper by Basili et al (S84, 1994), which goes back to first principles to establish a conceptual basis for software metrics. It is not concerned with software testing. Selecting it in isolation, and not even referring to the huge body of literature on SE testing, smacks of tokenism.

Preece does not mention the need to integrate the usability evaluation with the wider software testing exercise, let alone provide any analysis of the issues raised by that need.

The DECIDE framework itself is a decidedly naive and simplistic view of software testing.

- 1 **D**etermine the goals.
- 2 **E**xplore the questions to be answered.
- 3 **C**hoose the type of evaluation.
- 4 **I**dentify practical issues to be addressed.
- 5 **D**ecide how to deal with ethical issues.
- 6 **E**valuate and interpret the results.

There is no mention of the application's requirements in DECIDE. Goals are seemingly plucked out of thin air; there is no advice about relating them to the goals to which the developers have been working, or proactively ensuring that the developers' goals are the same as the testers'.

An example of a clearly stated goal is given as "*check that the evaluators have understood the users' needs*" (U04, p348). This is a fine example of the basic SE testers' error of attempting to define objectives for their testing (which should be based on the objectives of the development) and coming up with pointless platitudes which describe what the testers are going to do, e.g. "testing will check if the system is working".

An alternative "clear" goal is "*identify the metaphor on which to base the design*". This is something that the designers have to do; the testers need to know what was chosen. Are the authors assuming that the testers are doing the design? Are they assuming the testers are proactively involved during the design stage, working hand in hand with the designers? They offer no guidance on the point. The questions raised are left hanging.

The DECIDE framework is frequently cited by HCI researchers and students (e.g. Lesso, U160, 2005, p64) looking for some guidance on planning testing. I have found no trace of SEs citing it when they were looking for guidance on usability testing.

The detachment of Preece from SE is further illustrated by their discussion of requirements. Their view of the subject bears little relation to SE orthodoxy. E.g. Preece defines user requirements as capturing "*the characteristics of the intended user group*" (U04, p207).

Sommerville (S55, p118) defines them in standard SE terminology as *"statements, in a natural language plus diagrams, of what services the system is expected to provide and the constraints under which it must operate"*.

Preece produces surprising examples of "data requirements". There is no set, agreed definition within SE but few would argue with this definition from the Carnegie Mellon Software Engineering Institute (S73, 2006, p35); *"a data requirement is a product requirement that specifies mandatory [types of] data that must be manipulated by the product"*. A more detailed, commercial description is that they *"are the foundation of any software project. They ensure that information and business rules maintained and enforced by an application actually represent your organization's needs."* (S74, 2006).

Preece's examples are (U04, p209) *"the system must have access to the price of products"* and *"the system will need access to temperature readings"*. It could be argued that Preece's examples are deliberately general and high-level, but they are so vague that they would not qualify as data requirements at all in the judgement of an SE.

Is price to be stored gross of VAT, or net? Is temperature to be measured in centigrade or fahrenheit, what is the level of precision, the acceptable range? The temperature requirement is supposedly for a system to control a nuclear reactor where detail and precision would be paramount. Comparison with a real SE data requirements statement reflects badly on Preece's understanding. This is a genuine example from an **outline** business requirements document produced by an insurance company which took part in the survey for this dissertation. The document (which cannot be formally cited for reasons of confidentiality) details all the different types of premium data and the definition of each.

*"Transaction Paid Premium - the premium that is charged to the insured, net of Insurance Premium Tax, gross of commission, and net of all discounts and loading"*.

The unfair impression that experienced SEs will get as they read Preece's work is that the authors are operating out of their field of expertise, are parochial, and offering muddled thinking instead of useful insight. Inevitably experts in one field when they are learning about another will judge the material by what they already know. If the new learning material deals accurately and fairly with subjects that the student is already familiar with, then it will be trusted on matters that are entirely new to the novice.

### **Nielsen - over-ambitious claims for HCI**

Even the acknowledged HCI guru Jakob Nielsen has made overstated claims for usability engineering methodology. In *"Usability Engineering"* (U03, 1993, p5) he claimed that the four reasons thought to have the highest responsibility for SE cost overruns were *"all related to usability engineering"*.

These reasons were;

- ✘ scope creep,

- ✘ inadequate requirements definition by SE analysts,
- ✘ users' lack of understanding of their own requirements,
- ✘ poor communication between users and SEs.

These are fundamental problems, which have plagued the SE profession over the years. They apply also to software applications that have no user interface, e.g. batch financial processing applications that can reach a level of size and complexity to dwarf the more familiar and visible on-line applications. Portraying these problems as usability problems is looking at the problems of SE whilst wearing blinkers.

Nielsen also defines a usability engineering lifecycle model. Nielsen tends to labour the point that he is not a graphic designer, and his work lacks the simple, explanatory diagrams favoured by most other writers. His lifecycle model is as follows (U03, p72).

- “1 *Know the user.*
  - a *Individual user characteristics*
  - b *The user's current and desired tasks (task analysis)*
  - c *Functional analysis*
  - d *The evolution of the user and the job.*
- 2 *Competitive analysis*
- 3 *Setting usability goals*
  - a *Financial impact analysis*
- 4 *Parallel design*
- 5 *Participatory design*
- 6 *Coordinated design of the total interface*
- 7 *Apply guidelines and heuristic analysis*
- 8 *Prototyping*
- 9 *Empirical testing*
- 10 *Iterative design*
  - a *Capture design rationale*
- 11 *Collect feedback from field use”*

From the perspective of SE, Nielsen's model seems unclear, impractical and of limited value. Certainly, it seems to have had less impact than much of his other work. Everything in the model is viewed from the perspective of the user, but this is the ordinary user; the individual who will be pressing keys and clicking the mouse.

Where is the big picture; an awareness of what the application is trying to do, why it is being built?

Nielsen does not convey the importance of the organisation's goals and functions. There **is** an activity called functional analysis, but this is not an analysis of what the organisation needs to do; it is an analysis of **how** the user works. Task analysis (1b, a synonym for “*knowing the user's current and desired tasks*”), is merely the flip side of the same coin, i.e. **what** the user's

task entails, as opposed to how it is done. This approach is appropriate if the user is a consumer; it is misplaced, or rather incomplete, if the users are employees carrying out tasks set by their employer.

Grudin (U85, 1996, p170), discussing efforts to integrate usability engineering into formal SE methods (e.g. Lim & Long, U164, 1994) commented that these efforts “*identify organizational, task, and interaction levels, where “task” refers to a high-level work activity and “task analysis” is an organizational task analysis. In contrast, much human-computer interaction has focused on cognitive task analyses of low-level activities such as cutting and pasting text.*”

The focus Nielsen had on the individual user in 1993 is of greater value now, in the era of e-commerce, but in the 80s and early 90s such micro-vision would have contributed to the marginalisation of HCI.

Nielsen is surely providing a mirror image, rather than a genuine correction, of one of SE's great failings, the focus on the organisation and not the user.

### Mayhew - lack of integration with SE

Mayhew (U02, 1999) offers a more interesting and useful perspective by setting the question of usability engineering and testing more clearly in the context of a development lifecycle.

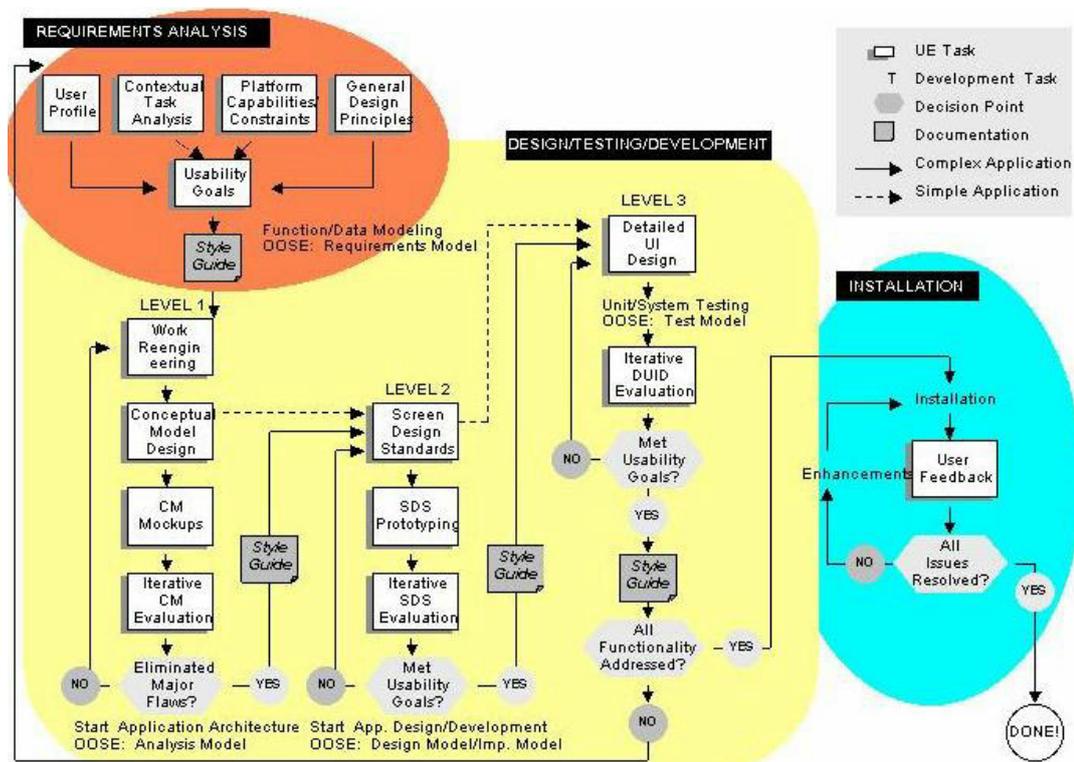


Figure 11 - Mayhew's Usability Engineering Lifecycle (U122, Mayhew, 2006)

Mayhew has a sufficiently good understanding of SE to realise that her work could not supplant SE methods. Her Usability Engineering Lifecycle (see figure 4) is envisaged as running in parallel to the SE lifecycle. On her own admission she does not tackle the question

of integrating usability engineering into the SE lifecycle, (p17). *“The exact overlapping of Usability Engineering and SE tasks is not completely clear. The most appropriate relationship between them depends on various project-specific factors”.*

The most obvious stumbling block concerns requirements definition. In the UEL these are gathered in a “contextual task analysis” and they are viewed only from the user’s perspective. The wider organisational perspective is ignored.

Mayhew correctly states that contextual task analysis is not the same as SE systems analysis (p68) but beyond saying that there’s some overlap she doesn’t say how they should be integrated. Indeed, (p17) she says *“exactly how to integrate them to avoid duplication of effort and yet produce the intended and necessary output of both is a topic for future work. In the meantime I simply adapt how I integrate UE tasks with an overall product development methodology on a project-by-project basis.”*

This still seems to be the current approach. The website for her consultancy (U122) still uses the same UEL diagram from her 1999 book, and offers to tailor it to the development practices of each client. This may be pragmatic and commercially prudent, but it hardly suggests a great conceptual advance is on offer.

However, integration of the UEL with SE would surely be required for Mayhew’s fundamental principles to be realised. As Gellner & Forbrig note, (U119, 2003, p75), there would be a substantial overhead in implementing the UEL alongside a conventional SE structure.

*“Fulfilling the requirements of this lifecycle will lead to a complex organizational structure. ... These requirements are hard to fulfill for a small or middle sized company. A whole staff is necessary to manage the various tasks. Mayhew’s lifecycle is directed to usability experts or deciders in a bigger environment that want to install an all-including usability department.”*

In practice, this overhead surely weighs heavily against the widespread introduction of Mayhew’s ideas.



*possibility of a single user interface for multiple applications (or vice-versa), and faster interaction with the user. These advantages ... have led many in the pursuit of a general purpose UIMS for building and running virtually any style of user interface."*

The Seeheim Model (U141, 1985), defined the essential features of a UIMS. As described by one of its developers (U103, Green, 1985, 206-9) it consisted of;

- ✘ presentation component, *"responsible for screen management"*,
- ✘ dialogue control component, *"manages the dialogue between the user and the application"*
- ✘ application interface model, *"is the user interface's view of the application"*.

There were other versions of UIMSs and models, but they were essentially similar, at least to the level of detail required for this discussion.

The technology of UIMSs is not relevant to this discussion. Nor are the wider merits of this approach, or its relevance to commercial SE. What concerns us is the effect that this approach had on the HCI profession.

Unsurprisingly, the principle of separability was seen as offering a great chance for HCI to introduce usability to SE, or rather to bypass SE practitioners and gain direct access to the users.

*"One of the major ways to achieve improved international usability is to separate the interface and the system's functionality"* wrote Nielsen (U03, 1993, p251), referring to Edmonds (U55, 1991).

Dialogue independence was seen as being the key way of linking the application and its interface, whilst enforcing segregation between them.

Hartson (U66, 1989, p63) wrote; *"Dialogue independence means that design decisions that affect only the user interface are isolated from those that affect the application's structure and computational software. Dialogue independence is crucial for easy modification and maintenance of a user interface. .... **dialogue independence requires that the external dialogue and computational part are developed separately as much as possible by the dialogue developer and the application programmer, respectively.**"*

Sotirovski and Kruchten (U142, 1995, p61) described the goal of dialogue independence as being *"to make the core applications and the screen presentation as independent as possible. This lets the interface and its application undergo independent development cycles while preserving mutual compatibility."* This concept of independent development cycles was seen as liberating HCI from the constraints and compromises that would be inevitable if they were fully integrated into SE development projects.

Foley (U143, 1991, p292) developed the idea of independence with a revealing analogy. *"There is a useful analogy and symmetry between UIMS concepts and those found in database management systems (DBMS) ... Just as a DBMS manages data, a UIMS manages*

*the interface. A DBMS hides the user from details of storage organizations, just as a UIMS hides the programmer from details of interaction devices and techniques.”*

SEs would regard the interface and detailed functionality as being intimately interwoven. The more detached relationship between the application and the database seems very different to SE eyes, and Foley’s analogy is helpful to understand just how important and fundamental HCI saw dialogue independence as being for the profession’s future.

Dialogue independence meant that the continued separation of the HCI profession and practices from SE was not only possible, but it was positively desirable for the future health of the HCI profession. This alignment of instinct and theory was immensely seductive.

Hartson & Hix (U65, 1989, p1) argued that this was the opportunity for HCI to create a new role. *“The role of a dialogue developer, whose main purpose is to create quality interfaces, is a direct result of the dialogue independence concept.”*

(pp15/16) *“The dialogue developer is a human factors specialist concerned with design, implementation, and evaluation of the form, style, content, and sequencing within human-computer interfaces. The dialogue developer’s needs and constraints are different from those of the programmer. The dialogue developer is involved in the entire system lifecycle, including task analysis and system requirements specification. During design and implementation of the dialogue, the dialogue developer uses an understanding of psychology and human factors principles to build and iteratively evaluate and refine an interface that supports effective human-computer communication. Often, dialogue independence allows modifications to be made quickly, so that the evaluation and revision cycle can begin again. Unlike the programmer, the dialogue developer must be sensitive to cognitive needs of the end-user. The dialogue developer role is a cross between a behavioural scientist and a systems analyst.”*

This contrast between the dialogue developer and the programmer is very revealing. The argument seems based on the false premise that the roles are equivalent, but in separate spheres. A more accurate contrast would have been between SE systems analysts and dialogue designers. It could be argued that Hartson and Hix are using “programmer” as a catch-all term for SEs. However, the claim that the dialogue developer’s role incorporates some of the systems analyst’s role, and the implication that programmers are not involved in the full system lifecycle suggests that the writers are using the word “programmers” correctly and that they see SE as being essentially a technical role operating on functionality that is hidden from the user.

Mayhew (U02, 1999, p483) argues the need for a user interface developer role, distinct from SE programmers and from the designers of the interface. Proving usable interfaces is not merely a matter of defining the requirements correctly by usability engineers; the construction of the interface itself requires different people with different skills from those of the SE programmers.

Despite the enthusiasm with which HCI seized on the apparent opportunities offered by separability the objections to separability ultimately proved impossible to ignore. They fall into three categories, and amount to a convincing rejection of the idea. The idea was flawed conceptually. It was flawed technically, ignoring the bicausal relationship between the architecture of an application and its user interface, and it was flawed managerially and organisationally.

## Conceptual objection

Usability is a matter of interaction between the user and the whole application, not just between the user and interface. The interface and the application are tightly intertwined, and the assumption that the user can, or should, be insulated from the internal workings of the application is mistaken.

Cooper (U26, 2004, p23) in discussing the difference between interface design and interaction design, says *"'Interface' suggests that you have code over here, people over there, and an interface between them. It implies that only the interface is answerable to the users' needs. The consequence of isolating design at the interface level is that it licenses programmers (i.e. SEs) to reason like this: 'I can code as I please because an interface will be slapped on after I am done'"*

In surveying the literature covering this area Kuutti and Bannon (U97, 1993, p264) comment in relation to a series of studies by Bannon & Bødker (U145, 1991 & U151, 1990), de Montmollin (U152, 1991), Thomas & Kellogg (U147, 1989), Göransson et al (U153, 1987).

*"Over the past few years, a number of people have pointed to the neglect of the task context in which human-computer interaction is taking place, and an undue emphasis on lower level interface issues. They argue that much of the HCI world pays too little attention to the underlying functionality of the computer applications and their usability in everyday work contexts."*

Donald Norman is quoted in an interview with Rheingold in Laurel (U154, 1990, p6), asking; *"What's wrong with interfaces? The question for one. The interface is the wrong place to begin. It implies you already have done all the rest and want to patch it up to make it pretty for the user. That attitude is what is wrong with the interface ....In the future I want less emphasis on "interfaces" and more on appropriate tools for the task."*

Göransson et al (U153, 1987, p136) put their finger on the nub of the problem, even before the idea of separability had taken hold; *"the interface can not be seen as a separate function, but must be specified from a comprehensive analysis of the work situation, and evaluated in the same context."* Usability requires an analysis of how the whole application will interact with the job the user's trying to do in a particular context.

Berry (U139, 2000) illustrate this argument using the analogy of an iceberg; *"over the years my colleagues and I have come to believe that the aspects embodied in the look of an interface contribute about 10% and those that are driven by the feel contribute about 30%. The user*

*model plays the major role, contributing about 60% to overall usability.*" The 60% represented by the user model is the invisible functionality of the application, behind the interface.

User models define, according to Berry, *"what the user is trying to accomplish, or in other words, the user's task goals. ... these goal-related aspects represent the semantics. They convey meaning in the conversation between the user and the system. The user model provides an understandable and cohesive framework of concepts that users can relate to, and that enables users to accomplish their tasks."*

This concept of semantics, the exchange between user and application leads us to the second objection to separability.

## **Architectural objection**

The architectural objection has two strands. The more obvious one is that separability ignored the bicausal relationship between the architecture of an application and its user interface. Usability requirements affect the architecture, which in turn have an impact on usability.

However, the other main strand of objection is interesting because it was acknowledged right from the start by the proponents of separability. This was the question of semantic feedback, i.e. the need for an application to provide feedback on user input, e.g. error messages and validation results.

If control over the semantic feedback is moved into the interface layer then that introduces functional complexity and reduces dialogue independence. If the control is left within the application then that increases the communication between the interface and the application and makes it harder to develop them separately, and again dialogue independence is reduced.

Further, the increased communication between the layers introduces *"adaptability problems"* (Evers, U56, 1999) making changes more difficult, and thus removing one of the main benefits claimed for dialogue independence.

*"The functionality that crosses the application-interface boundary becomes hard to adapt: if the functionality changes, modifications will have to be made to different locations in the software system (problems of scattering and propagation of changes). Changes to the user interface influence the application and vice versa.*

*The adaptability problems described in this paper are in my opinion fundamental problems of user interface architectures that are based on the separation of application and user interface."*

Hartson (U66, 1989, p65) recognised the problem of semantic feedback and argued the need for a balance between loading all the semantic complexity into either the interface or the application.

*"You can violate dialogue independence by having too much or not enough display and computation in the input component. The best separation (between interface and application)*

*achieves just the right amount of power in the dialogue. Achieving that ideal separation is neither always easy, nor in fact always possible."*

It is significant that all of Hartson's examples in his paper are based on office systems applications; word processing and spreadsheet applications. There is no consideration of the implications of semantic feedback for dialogue independence in commercial processing applications which need to perform calculations, processing and complex cross-validations between input fields before passing feedback to the user. With such applications it is impossible to provide adequate feedback by greater use of pull-down menus or simple validations at the interface.

Hartson's concession that ideal separation is not always possible does not go far enough. The problem is deeper than he acknowledges.

In a 2001 edition of IEEE Software focussing on usability, Juristo et al (U114, p21), in their introduction to the edition, seemed sufficiently sure that separability had had its day to say; *"ensuring a certain degree of usability based on the intended user's work practices is very important when designing a good system concept or metaphor and must be integrated early in the design process. Interaction design can greatly affect the application's overall architecture. If you consider usability too late in the life cycle, there is no time left to really make a difference—you can't just toss it in at the last minute, any more than you could a good database schema."*

Much of the academic consideration of the implications of separability has used the example of the "undo" function, e.g. Bass & John (U11, 2003), Folmer et al (U155, 2003) and Pyla et al (U54, 2003), which supports the observation by Grudin (U86, 1991, p150) that HCI is dominated and influenced, in the USA at least, by experts from companies that make shrink-wrapped applications, rather than bespoke commercial applications. However, the architectural problem applies with equal force to bespoke applications.

Usability concerns reach deep into the heart of bespoke applications. Response times are an important concern for usability, and SEs have to plan and design applications carefully to produce acceptable results. The technical constraints and usability requirements can be in conflict, forcing compromise.

To hark back to semantic feedback, if the application has to perform complex processing and validations that combine input data with existing data before providing feedback then consideration has to be given to whether the application should provide feedback piecemeal, or check the transaction as a whole before providing feedback. It can be infuriating if the user is asked to correct a string of minor errors before having the entire transaction rejected for some fundamental reason, e.g. an insurance claim being rejected because the renewal premium had not been paid.

However, if the application has to access and compare data from several database tables over a network the delay between the user hitting enter and receiving feedback may be

unacceptable, and it may be preferable to give the user snappier feedback on individual problems as they are detected. It is impossible to generalise on these points other than to say that such considerations have significant implications for the architecture of the application and making the correct choices for each case requires an understanding of the specific users. This can't be achieved if the internal architecture is regarded as a black box, separate from the interface.

### **Managerial (organisational) objection**

Separability of the interface from the application may have been flawed conceptually and technically, but it was immensely attractive to HCI because it seemed to liberate them from SE. However, this illusory freedom actually meant isolation and impotence.

As Bannon (U146, 1997) wrote, *"paradoxically, this reification of the interface as the domain of HCI has actually marginalised its impact"*.

Holmlid (U79, 2002, pp18/19) noted the increased *"isolation of HCI and usability activities to the interface"* and that it *"is likely that many HCI-practitioners will find themselves trapped in activities concerning the presentation layer, at the worst doing cosmetic work"*. One of the reasons he gave was the increased use of development models based on separate layers. Interestingly, the more recent examples given by Holmlid of layered models (Nunes & Cunha, U156, 2002, and van Harmelen et al, U157, 1997) make no mention of a fundamental split between the interface designers and the SEs. They assume that the HCI professionals will be involved in designing the whole application. This suggests that the damage HCI did to itself by over-enthusiastic adoption of separability had already been done, but was being recognised.

However, in 2004 Bass et al (U16, p1) saw it as a continuing problem. Separability had influenced SEs, effectively persuading them that usability was not their concern. *"For the past twenty years, software architects have treated usability primarily as a problem in modifiability. That is, they separate the presentation portion of an application from the remainder of that application. ... Separating the user interface from the remainder of the application is now standard practice in developing interactive systems. Treating usability as a problem in modifiability, however, has the effect of postponing many usability requirements to the end of the development cycle where they are overtaken by time and budget pressures. If architectural changes required to implement a usability feature are discovered late in the process, the cost of change multiplies. Consequently, systems are being fielded that are less usable than they could be."*

Separability ignored the reality of development projects. Testing is a cinderella discipline, which if it is to be effective, has to break out of its ghetto at the end of projects and influence requirements and design as early as possible. There is always a danger that testing will be squeezed in the final dash for implementation. Nothing that could be done earlier should ever be scheduled for that late, frantic time.

Dialogue independence and separability meant that the integrated interface and application could be tested only at a late stage when it was too late to remedy fundamental flaws. In effect, the HCI profession was volunteering to join SE testers in their traditional ghetto because of their confidence that separability would allow them to produce high-quality interfaces.

As Read (U22, 2003, p2) has correctly noted, serious usability issues can be the product of a large number of relatively trivial annoyances. In the pressurised circumstances of a software development project these would not be sufficient in themselves to justify a postponement of implementation, but their cumulative effect in operation could be very detrimental to the user.

Lewis & Rieman (U104, 1994, appendix M.2) memorably savaged the idea that usability professionals could hold themselves aloof from the application design, calling it *“the peanut butter theory of usability, in which usability is seen as a spread that can be smeared over any design, however dreadful, with good results if the spread is thick enough. If the underlying functionality is confusing, then spread a graphical user interface on it. ... If the user interface still has some problems, smear some manuals over it. If the manuals are still deficient, smear on some training which you force users to take.”*

The separability fallacy was all the more damaging because it created the dangerous illusion that architectural advances had solved the problem, in principle at least. All that was required was to introduce usability engineering to the development of the interface, and usability testing to the completed application. It took until well into the 90s for the HCI professionals to recognise what should have been obvious when the idea was first mooted in the late 80s.

## **Inappropriate reliance on usability testing**

There are inherent limits to usability testing, and this was always recognised by its proponents. However, of greater interest to this dissertation is the way that its influence has been weakened by over-reliance on it. When a technique becomes fashionable, yet imperfectly understood, there is a danger that it will be used poorly and inappropriately.

Rubin (U01, 1994, p27) lists four reasons why usability testing won't guarantee that a product is usable.

- ✘ Testing is an artificial situation.
- ✘ Test results don't prove the product works.
- ✘ Participants are unlikely to be fully representative of the user population.
- ✘ Testing may not be the best technique to use.

These four reasons are simply pragmatic statements of reality, rather than dramatic concessions. Of particular interest is the fourth one.

*“There are many techniques intended to evaluate and improve products .... For example, in some cases it is more effective both in terms of cost, time, and accuracy to conduct an expert*

*evaluation of a product rather than test it. This is especially true in the early stages of a products when gross violations of usability principles abound.*

*However, in spite of these limitations, usability testing, when conducted with care and precision, for the appropriate reasons, at the appropriate time in the product development lifecycle, and as part of a an overall user-centred design approach, is an almost infallible indicator of potential problems and the means to resolve them.”*

It is clear that even one of the most respected experts on usability testing saw it as only one weapon in the armoury of HCI professionals. However, in the context of software development projects it became not just the weapon of choice, but all too often the only weapon, and one that had often been devalued or distorted when SEs have tried to adopt it.

Dicks (U58, 2002, p28) writes sceptically about what SEs sometimes call usability testing. *“It is fairly standard practice to perform tests against ... online systems at the conclusion of a development cycle. ... Some documentation groups have begun to add a few usability criteria to the list of things they check and to refer to these final tests as usability tests. This gives everyone involved a cheap, easy way to say that they did usability testing, but it is a subversion of the entire idea of usability. While human factors experts can perform heuristic evaluations that uncover some usability problems, the kind of checklist testing done by managers, editors, or testers at the conclusion of a development process should not be called usability testing. To do so cheapens the whole concept of usability, which requires that we test with “real users.”*

Gellner & Forbrig (U77, 2001, p1) reached a similar conclusion, that laymen were misusing “usability testing”. Their comments were specifically directed at smaller organisations. However, since their definition of “smaller” was organisations who didn’t have usability engineers or usability labs, they were clearly talking about the great majority of SE development projects.

*“Usability evaluation often happens in the last phase of the development cycle ... where ergonomic laymen practise these evaluations. There, usability testing is considered part of the testing-phase. Functional and logical errors will be fixed certainly in that situation, whereas a user interface that misses the target will not be changed.”*

This problem has been exacerbated by a tendency to confuse usability testing with usability design. Interface design is emphatically not the same as interaction design, and it is the latter which has the greatest significance for usability. Interaction design reaches deep into the heart of the application.

Ferre et al, writing from the SE perspective said (U113, 2001, p24); *“a system’s usability depends on the interaction design. Therefore, we must deal with system usability throughout the entire development process. Usability testing alone is not enough to output a highly usable product, because usability testing uncovers but does not fix design problems. Furthermore, usability testing has been viewed as similar to other types of software quality assurance*

*testing, so developers often apply the techniques late in the development cycle—when major usability problems are very costly, if not impossible, to fix. Therefore, it is crucial to evaluate all results during the product development process, which ultimately leads to an iterative development process. A pure waterfall approach to software development makes introducing usability techniques fairly impossible.”*

So did an inappropriate emphasis on usability testing, rather than usability engineering (or interaction design), perversely support the Waterfall? That is a reasonable conclusion to reach when you consider the managerial objection to separability (see previous section, and especially U114, Juristo et al).

HCI and SE had inadvertently conspired to relegate usability to the margins at the end of projects, when the only testing that could be done was summative testing. As Read (U22, 2003, p2), see page 37, argued, and every SE knows, when the pressure is on, applications will be implemented with large numbers of “cosmetic” defects uncovered late in the project with appalling results for usability.

## **Weak requirements and the problem of deriving the design**

This dissertation has been critical of HCI’s lack of awareness of commercial SE, but it can be argued that SE academics have also had a shaky grasp of the current realities of commercial IT development. The focus of academics of both stripes has understandably been on the glamorous vanguard of the profession, rather than the massed ranks of the “poor bloody infantry” slogging along behind.

This poor understanding of what’s happening in the outside world is illustrated by this quote from the famed SE methods expert Yourdon (S66, 2006, ch6, Endnote3) referring to the surprise many firms had when they assessed their software inventory as part of the Y2K problem.

*“Many organizations that sincerely believed they were using only “packaged” software from third-party vendors discovered, to their amazement, that the user community was continuing to operate home-grown systems ... that had been developed 15-20 years earlier.”*

The SEs who ran these applications were not in the slightest amazed. They knew their organisations had ancient systems. Yourdon’s comment is revealing because it suggests that while his attention was fixed on more interesting and advanced problems he was unaware of the mess out there in the real world, and also that he did not appreciate that the SEs at the coal face were better informed than he was.

One key field where the HCI academics have struggled to understand what is going on and to make a worthwhile contribution is the process of moving from usability requirements to concrete, practical SE design decisions.

Cooper (U26, 2004, p208) puts the problem bluntly.

*“At least four large companies that I work with have a long history with usability professionals. The companies decided to invest in usability. They hired professionals who built labs, performed their studies, identified likely problem areas, and made a series of guesses about how to improve things. The programmers diligently made changes to their programs, **and not much happened**, except that the programmers had worked a lot harder. After a few cycles of this, the programmers simply gave up, and so did most of their managers. ....*

*I have seen well-respected usability professionals blandly take potshots into the dark. .... When their rigorous, scientific method uncovered a problem area, they would lapse into the most amateurish babble about solutions.”*

Bass & John (U12, 2000, p171) make much the same point, but in less entertaining style; *“The CHI community advocates design processes that bring usability considerations to early design decisions, but most of these processes stop short of explicitly mapping usability quality to specific software design patterns. This leaves the mapping to a software engineer who may not know much about usability, to a usability specialist who may not know much about software engineering, or to a multi-disciplinary team whose members have difficulty communicating. In other words, the mapping may often remain incomplete, or implicit in architectural decisions and thereby unexamined.”*

Folmer and Bosch (U60, 2004, pp2-3) write that the SE development process has three elements to it.

- ✘ Extract the requirements
- ✘ Realise the requirements in a design
- ✘ Assess the resulting product

They argue (p2) that SEs can't handle usability effectively because of fundamental problems in the design stages. It is impossible to perform any effective assessment of the Architectural Design.

*“SEs in general have few techniques available for predicting or realising the quality attributes of a software system before the system itself is available. Most engineering disciplines provide techniques and methods that allow one to assess and test quality attributes of the system under design. ....*

*Some of the more popular techniques such as user testing, heuristic evaluation and cognitive walkthroughs can be used during several stages of development, however, there are no assessment techniques that focus on assessment of usability during the early stages of design (e.g. software architecture design).”*

When the architectural, or outline, design is being carried out, the realisation and assessment activities are done badly. In particular, realisation is effectively missing from Architectural

Design. Designers have no way of translating usability requirements into an application architecture that will satisfy these requirements.

Folmer & Bosch, in a separate paper (U158, 2004, p12) identified a specific problem with usability requirements. They are traditionally weak; *“specified such that these can be verified for an implemented system. For example: ‘new users should require no more than 30 minutes instruction’. However, assessing an architecture for such a requirement is difficult because such requirements can only be measured when the system has been completed.”*

This is a familiar problem for SEs. When confronted with requirements like this, and taking account of the difficulty in performing early assessments and of converting the requirements into a design, it is common for SEs to effectively ignore such weak usability requirements.

Objective testing requires measurable targets. If usability requirements, and thus targets, are aspirational, imprecise, and therefore subjective, then they pose little challenge to developers. During the early stages of testing (i.e. unit, integration, and system testing) the developers and SE testers can subjectively assess the application against these imprecise targets, and mark the test as “passed”.

By the time that the real users have the chance to test the application in the realistic conditions of user acceptance testing the project has picked up momentum, the end is in sight, and there is tremendous pressure on the users not to rock the boat by raising usability problems that are glibly dismissed as cosmetic.

Weak requirements will be complied with if possible. They may even influence the design. However, if there is any conflict between such a requirement and a firmer requirement, or with the project budget or schedule, or if it seems too abstract to influence the design then the requirement will be ignored.

This may seem irresponsible, or even unprofessional, but it is entirely rational and predictable behaviour on the part of the SEs. Their careers live and die by their ability to implement applications on time, to budget and to an acceptable level of quality. It is naive to expect them to respond altruistically to weak requirements.

Gulliksen et al (U159, 1999, p13) commented on this; *“In order for the systems developers to appreciate the importance of (usability) requirements, special measures may be needed. One example was using a bonus system as an incentive for the systems developers to meet the requirements.”*

This raises the question of how SEs take requirements and move forward to build an outline (or architectural design). Folmer & Bosch alluded above (U60) to the difficulty that SEs have in realising requirements in a design. This leads us on to one of the guilty secrets of SE. Translation of requirements into design is not simply a difficult task; it is a fundamental problem for SE. It is not a problem specific to usability requirements, and it was never resolved in the structured methods that were dominant for so long.

In the discussion of the historical development of SE we saw how experienced designers were selective in their use of structured techniques, specifically their use of the output from analysis; doing what worked, rather than sticking rigidly to the rules (Bansler & Bødker, S47, p182) see page 15.

Fitzgerald et al (S01, 2002, pp28-29) provide a fascinating account of the underpinning of structured methods. They cite articles by Ward (S75, 1991; S76, 1992; S77, 1992), one of the leading developers of structured methods, which admit that structured methods were neither based on empirical research nor subjected to peer-review. Fitzgerald (S01, p28) also quotes Constantine, another founder of the structured approach; “(the early) *investigations ... were no more than noon hour critiques*”. (S78, 1977). Fitzgerald goes on to say “*The authors relied on intuition rather than real-world experience that the techniques would work*”.

One of the main problem areas for structured methods was the leap from the requirements to the design. Fitzgerald et al say, (S01, p29), referring to Colter (S79, 1982); “*the creation of hierarchical structure charts from data flow diagrams is poorly defined, thus causing the design to be loosely coupled to the results of the analysis. Coad & Yourdon (S80, 1991) label this shift as a ‘Grand Canyon’ due to its fundamental discontinuity.*”

The solution to this discontinuity, according to the advocates of structured methods, was supposed to be an avalanche of documentation to help SEs move carefully from the current physical system, through the current logical system to a future logical system and finally a future physical system.

Not surprisingly, given the massive documentation overhead, and SEs propensity to pragmatically tailor and trim formal methods, this full process was seldom followed. What was actually done was more informal, intuitive, and opaque to the HCI profession.

An interesting strand of research was pursued by HCI academics such as Curtis et al (S40, 1988) and Robbins et al (S39, 1998). They attempted to identify the mental processes followed by successful software designers when building designs. Their conclusion was that they did so using a high-speed, iterative process; repeatedly building, proving and refining mental simulations of how the system might work. Unsuccessful designers couldn’t conceive working simulations, and fixed on designs whose effectiveness they couldn’t test till they’d been built.

Curtis et al (S40, 1988, p1272) wrote; “*Exceptional designers were extremely familiar with the application domain. Their crucial contribution was their ability to map between the behavior required of the application system and the computational structures that implemented this behavior. In particular, they envisioned how the design would generate the system behavior customers expected, even under exceptional circumstances.*”

Robbins et al stress the importance of iteration (S30, 1988, p3); “*The cognitive theory of reflection-in-action (Schoen, S41, 1992) observes that designers of complex systems do not conceive a design fully-formed. Instead, they must construct a partial design, evaluate, reflect on, and revise it, until they are ready to extend it further.*”

Glass (S29, 2006, p34) in discussing such studies makes the point that *“people who are not very good at design ... tend to build representations of a design rather than models; they are then unable to perform simulation runs; and the result is they invent and are stuck with inadequate design solutions”*

This work shows that iterative, prototyping lifecycle models went with the grain of successful design, whereas unsuccessful designers replicated the rigidity of the Waterfall model. Strangely, the relevance of this research to the gap between SE and HCI does not seem to have been pursued by the mainstream of HCI.

Instead, the response of HCI to the difficulties in gaining acceptance with SE was to look for solace in the prospect of technological advances providing the answer to usability problems.

To summarise the involvement of HCI with SE, we are looking at a situation where the usability requirements are still frequently poorly specified or absent; SEs have no reliable, objective way of moving from whatever requirements there might be to an architectural design; and usability remains largely consigned to the margins, the interface where it has effectively become a matter of aesthetics and cosmetic concerns.

It is no surprise that even when HCI managed to shake off the damaging illusion that technical advances such as UIMS and dialogue independence would come to their rescue, it has had great difficulty engaging effectively with SE. Separability allowed HCI to stay aloof and largely irrelevant, to join SE in placing undue reliance on usability testing, and thus allowed SE to ignore for years the fundamental issues affecting usability.

## **Lack of penetration of HCI into SE**

### **Research on how HCI practitioners work together**

This dissertation has explained reasons why HCI has had little effective influence within SE. The arguments are backed up by a paper by Kazman et al (U17, 2003). Their study surveyed 33 SE's and 63 HCI practitioners who considered themselves to be in the software development profession.

Their results show a lack of mutual knowledge and respect, which is worse on the SE side. Not only do the two sides fail to communicate as professions, they do not communicate effectively at the level of individual projects; the two sides frequently giving different names to the same activities. Where communication does take place it is usually too late to be effective, (U17, p507-8).

*“SE's and HCI practitioners are not keeping informed about changes being made in each other's development processes. This could explain why SE's are often forced to make design decisions that affect the user interface without consulting HCI practitioners. .... One HCI practitioner said ‘designers have to work around the architectural decisions of the SW*

engineers'. Another respondent, commenting on compromises between SE and HCI professionals said 'none. (Software engineering) always wins. ' ...

78% (47 of 60) indicated that they corresponded with the SE's (for the first time) during the testing or release phases of the software - i.e. far too late to fix usability problems economically and with minimal user impact. A mere 3% (2 of 60) claimed that they worked with SEs during the specification phase of the project."

These figures are bad, but they come from organisations where HCI professionals **are** involved in SE developments. Many more organisations do not have even that token involvement from HCI.

Educating SE students in HCI at university has had little effect. As Seffah (U161, 2004, p89) reported; "while many developers may take a course of basic concepts of user interface design such as task analysis or heuristic evaluation, few of them have an understanding of the complete UCD toolbox at a level that allows them to incorporate it into the whole software development lifecycle". Scrutiny of the sort of standard texts they would have been studying helps to explain why these SEs cannot see how to incorporate HCI. It is not simply a matter of failing to understand all the tools.

## **Results of questionnaire**

The results from the questionnaire for this dissertation reveal little evidence of HCI penetrating the mainstream of SE. Of the eight questionnaires returned, three were from organisations with a specific interest in HCI (a usability testing consultancy, a manufacturer of electronic public-facing equipment, and the web manager for a government department). The remaining five were from experienced SE professionals. Two working for large insurance companies (one UK, and one in Australia), one for a UK bank, and two for large UK IT services suppliers.

The respondent from the UK insurer, a senior SE with 20 years commercial IT development, regards usability as a matter of complying with legislation to make applications accessible to everyone; "Not sure what is meant by usability. On our internal web developments we have to be aware of how visually impaired people can use the applications and we had to make a number of changes in order to conform to the Disability Discrimination Act."

His response to a question about HCI was to say it's "not a term I'm familiar with."

The respondent from the Australian insurer, a senior developer with 15 years experience, had never come across any HCI professionals or usability initiatives, but commented; "requirements etc are often developed by those who perceive they know how the system is used and not by those who are actually are using it and hence changes can often make the system less usable."

On the subject of considering usability in developments she said; "only if it is really referred to in the requirements .. i.e. that this must perform like this... "

The respondent from the UK bank, a senior developer with 20 years experience had never heard of usability engineering or HCI. This was not a question of terminology; discussion of his response made it clear he had never encountered the concept.

The responses from the two IT suppliers were intriguing. They both loyally expressed confidence that usability was handled properly, but their detailed follow up answers failed to support the claims.

One, a senior project manager with 25 years experience, said that usability was always considered, and that it was a standard form of non-functional testing. He was talking in the context of contract development of applications for external clients. However, when asked to comment on whether he thought that non-functional requirements and testing were handled adequately he wrote; *“experience suggests that most organisations do not consider these aspects until too late in the development lifecycle”*. He also said that his company does not use HCI professionals (internal or external) and has not undertaken any usability initiatives.

The respondent from the other IT supplier, a senior SE tester with 10 years experience of contract developments for external clients, admitted that usability was *“not an area in which I have been directly involved”*, and that he didn't know of any special initiatives relating to usability. He believed that usability was handled properly, but admitted that he'd never been involved in a project where there was any usability testing, even though all of his experience had been on interactive systems. This company does employ HCI professionals. However, an examination of this organisation's public website makes it clear that HCI professionals services are sold separately from normal SE services, the implication being that it is a premium, “add-on” service, an interpretation whose accuracy was conceded by the respondent.

The respondent from the equipment manufacturer was the only one to confirm that their organisation does employ HCI professionals, and has been able to integrate them with SE with some success. This company is not representative though, because it makes and sells software and hardware for kit that will be used by the general public on a “walk up and use” basis. However, surprisingly HCI professionals are involved in specification, but not testing.

The only respondent to say that usability testing is performed as a matter of routine was the web manager for the government department. Usability testing is carried out once the development is essentially complete. Then, external HCI consultants and testers check it. This is inconsistent with iterative development. However, if problems are uncovered then they would be corrected, and they would keep iterating till they got it right. The website is purely informational (rather than processing input and output data). Also, usability and accuracy are essential, so the overhead of working this way does not as onerous as it would be on a commercial processing system.

These results support this dissertation's argument that HCI's penetration of SE has a long way to go. The only respondents to have any practical knowledge, or even awareness of HCI and

usability, were those working in fields where it was inescapable and ignorance is simply not an option. The conventional SEs who replied were almost completely unaware of the field.

The academics are aware that there is still a serious problem, but they are so detached from the realities of commercial SE that they do not realise that the bad situation they describe is actually even worse than they fear.

# The Way Ahead

Twenty two years ago, Gould & Lewis (U28, 1985) wrote a paper, *“Designing for Usability: Key Principles and What Designers Think”*, which has become a staple source for academics and students. It has been called a classic by Mayhew (U02, p4) and has been cited in 86 papers on ACM. A Google search on the authors and title returns 997 hits. Gould & Lewis stated three key principles of designing for usability, of which Mayhew said *“their ideas were quite revolutionary at the time”* (p4).

These principles were;

- ✘ early focus on users and tasks,
- ✘ empirical measurement (i.e. use of prototypes and simulations, with recording and analysis of the results),
- ✘ iterative design.

These principles are as valid now as they were then. They were seen as the way ahead in 1985, but penetration of HCI into SE over the last 20 years has been so poor that they remain the way ahead in 2007.

In principle, the answers are amazingly simple. In practice, they are fiendishly difficult to apply. The difficult historical relationship between HCI and SE suggests that the answers lie less in the technology and formal methods and more in the managerial, contractual and cultural issues that shape projects.

Technology offers useful tools and opportunities; methods offer guidance. Neither can make a significant difference if the management and the culture are wrong. Similarly, if the organisation is committed to usability then it can achieve good results regardless of the tools or methods.

Object-Orientation (O-O) offers opportunities to improve usability if a prototyping, iterative approach is chosen. However, it is perfectly possible to use O-O without prototyping or iteration. As Mandel wrote (U162, 1997, p240); *“when some programmers obtain more sophisticated technologies and tools, they then have new and exciting ways to build even worse user interfaces than before”*.

As an illustration, one of the respondents (from a large UK IT services company) to this dissertation’s questionnaire, said that he usually worked on developments using the O-O based Rational Unified Process, but had never been involved with questions of usability and had never seen usability testing performed.

Gulliksen & Goransson (U111, 2003, p29) argue that RUP is too focussed on the architecture and technical matters, that RUP’s use cases are opaque to users, and that HCI favours broadly defined use cases whereas SE prefers them to be much more specific. The following year Gulliksen et al, but no Goransson, (U168, 2004, p214) reached a rather less negative

conclusion when they reported on the experiences SEs had had combining HCI and RUP; *“even though the process does not in itself contain support for usability, the respondents have integrated usability activities into it rather than introducing another process.”*

There have been attempts to refine RUP to make it more amenable to usability, notably by Sousa et al (U36, 2005 & U167, 2005), but these do not yet seem to have had an impact.

Similarly, Agile methods seem to provide interesting opportunities to introduce usability to developments, notably Kent Beck’s Extreme Programming (S85, 2005). This requires fast movement by developers, with heavy, early user involvement and rapid production of successive prototypes and working applications. The testing emphasis is moved from the end of the development to the beginning, and functionality is delivered in small portions.

This offers obvious potential benefits to HCI (e.g. U29 Sharp et al, 2006, & U33, Holzinger et al 2005), who now see the chance to get involved in effective interaction design of applications. However, there is scepticism about this approach, based mainly on doubt about whether the developers have been really committed to HCI principles (U35, Jokela & Abrahamsson, 2004), on whether core techniques are generally applicable (S65, Stephens & Rosenberg, 2003), and a belief that the approach has been oversold, or rather mis-sold (S65 again, and U165, Ambler, 2006).

A criticism of XP directly relevant to usability comes from Constantine (U37, 2002, p5) who argues that iterative prototyping of the interface is too superficial an approach except for simple applications, and is no substitute for a deeper, more sophisticated understanding of the users and the design they require.

On the other hand, Meszaros & Aston (U34, 2006) claimed considerable success in combining formative usability testing (paper prototyping) with XP. This was for a fairly complex application calculating quotes for rail freight in Canada.

XP is not the only Agile method. Others are DSDM (S86, DSDM website), Scrum (S87, Schwaber & Beedle, 2002) and FDD (U166, Ambler, 2006). However, Rakitin (U91, 2005) has argued that Agile methods have a shared problem in that none of them work unless the right conditions are in place, but that if these conditions are present, then other, more traditional, methods would also work.

A common feature of all these Agile methods is that usability is not an up-front aspect. It is referred to in passing. They are of interest not because usability is necessarily dependent on methods such as these, but because they provide opportunities to those who are already committed to usability. It seems unlikely that there is any neat fix, or silver bullet, that can solve all usability problems. SEs who want to develop usable applications must select appropriate tools and methods for each situation rather than assuming that one set will always be applicable.

An alternative approach to the Agile movement is offered by Alan Cooper’s Interaction Design (ID), (U26, 2004). The essence of ID is that SEs must carry out a form of functional analysis

that seeks to understand the business problem from the perspective of the users, based on their personal and corporate goals, working through scenarios to understand what they will want to do. It strikes a balance between the old, flawed extremes of SE structured methods (which ignored the individual) and mainstream HCI (which paid insufficient attention to the needs of the organisation).

On the face of it ID seems inconsistent with Agile methods; the former requiring heavy up-front analysis, whereas the latter requires SEs to derive requirements on the fly with the users as each iteration is developed. However, Cooper and Beck seem to believe that it will be possible to develop a process that combines the essential features of ID and XP, see Nelson (U45, 2002). However, five years on there is no sign of this common process.

It is outwith the scope of this dissertation to assess the pros and cons of these methods and technologies, especially since none of them are individually either necessary or sufficient for producing usable applications. What should concern us is the framework within which any of these would have to operate.

There have been relatively few attempts to merge SE and usability lifecycle models, possibly because the approach would be misconceived. Long after the separability principle lost credibility Pyla et al (U54, 2003) defined a merged model. However, the underlying assumption was that the two spheres had distinct, valid lifecycle models and that the problem was essentially one of defining the communication links between them.

The conclusion of this dissertation is that new lifecycle models are unnecessary, and a distraction in that there is no obvious appetite for new models. Bundling up useful techniques and initiatives and trying to sell them in a "lifecycle model" will elicit only weariness and cynicism in SE practitioners. The lifecycle chosen is of vital importance, but in a negative sense. The wrong model, specifically the Waterfall, will make it very difficult to achieve usability. Selecting an appropriate model is verging on the necessary, but it is definitely not sufficient.

it is therefore inappropriate to look to HCI lifecycle models, e.g. Mayhew (U02, 1999) or Hartson and Hix (U65, 1989, p52). SE needs to incorporate HCI techniques, and co-opt usability specialists into projects. Thinking in terms of parallel lifecycles risks making the same mistakes that have seen HCI marginalised where they have been involved in SE developments, restricted to summative evaluation at the end of projects, without significant influence over the design.

HCI professionals must be prepared to sacrifice some of their impotent independence and integrate into SE. They have to persuade SEs that they are technical specialists whose presence is as important as the database administrators, systems analysts, tools specialists, architects, coders and testers who make up the typical SE project.

If SE is to make serious efforts to introduce usability then HCI should persuade them to buy into the existing international standards for user centred design; ISO13407, *"Human-centred*

*design processes for interactive systems*" (U07, 1999) and ISO18529 "*Human-centred lifecycle process descriptions*" (U108 & U109, 2000).

Between them these standards lay out the activities and the sequence of work that is required. ISO13407 is not highly prescriptive. It is indicative of the work required, rather than providing a precise route map.

It would be possible to quibble with some of the content. For instance, a literal reader would infer that the usability requirements are not used to generate a design, but are used only for evaluating the application. However, it would be a mistake either to approach the standard in such a literal manner, or to reject it because it does not provide idiot-proof instructions.

ISO18529 provides much greater detail for each activity, sets the development work more clearly into a lifecycle, and defines a Usability Maturity Model for assessing an organisation's ability to provide usability. See Appendix D for an expanded description of ISO13407 and ISO18529.

It seems reasonable to assume that ISO13407 can fit in with Agile methods, provided that there is real commitment to usability (see reference to U35, Jokela, on page 48), but it is less obvious that ISO13407 will easily fit conventional, iterative SE development methods. RUP has been the main area of research.

Bass et al (U14, 2003) studied that very point with reference to RUP. Their broad conclusion was that they were compatible. There were points of difference in emphasis and terminology (p3), but my interpretation of their work is that there would not be a problem for a team that had the right skills and genuine management support and commitment.

It seems less likely that the more detailed ISO18529 could be accommodated easily within a conventional SE development, though the detailed content would provide valuable guidance.

It is tempting to look to improved education of SEs for the answer to achieving usability in software applications. However, as Seffah argued (U161, 2004, p89), see page 44, a university introduction to HCI for student SEs has little practical benefit. This is consistent with the earlier discussion in this dissertation on HCI authorities such as Preece, Mayhew, Rubin and Nielsen. These are the very sources that the students are likely to be consulting, and it is not surprising that they cannot relate their studies to their practical experience. Any useful lessons they learn about HCI will be quickly forgotten once they arrive on SE projects led by the sort of experienced staff who returned the questionnaires for this project.

What is needed is a more radical, practical curriculum for student SEs, and it needs to be taught explicitly as an integral part of the SE curriculum, not as a separate, possibly optional addition. Students need to be challenged, and should be required to read some of the more controversial, and possibly less academic, material in order to shake them out of their complacency. Possible sources would be Alan Cooper's "The Inmates are Running the

Asylum” and papers, such as those by Folmer & Bosch (U60) which argue that software design is a less precise and scientific process than was formerly thought.

In conclusion, there are no straightforward answers to the question of incorporating usability testing into conventional software testing, and attempting to do that could lead to the mistakes made by the HCI community when they isolated themselves from SE and inadvertently promoted summative, rather than formative, usability testing.

The development of usable applications requires management and user commitment at **each** stage in the development process, and not just during testing; it requires the selection of an iterative lifecycle model and adoption of user centred design principles. In the long term, it will require SEs to change their mindset so that they apply usability techniques as a matter of course, in the same way that they follow change management, release management and program library management principles instinctively.

Trainees SEs are not born with a commitment to these disciplines. They learn from more experienced staff, and from experience that that is the way it has to be done. Likewise, SEs need to have a commitment to usability instilled in them at university, and then in the workplace, where there has to be consistent pressure from senior management and from users to ensure that applications are developed the right way, that projects are scheduled to allow iteration, that contracts are written in such a way as to permit iteration and require the right level of user involvement.

Williamson (U148, 1997, p13) suggested that *“one could argue that the HCI profession is transitory”*. He drew an analogy between HCI and fitness experts. There was a time when fitness was the province of specialists in nutrition and physiology. Now, the expertise of these specialists *“has been absorbed into our culture. In 50 years time it won't be necessary for the HCI Professional to be on hand to chastise the anoraks, or inform the arm wavers. Just as fitness regimes combining diet, exercise and mental strength are designed by Tennis coaches, Badminton coaches, Football coaches, Rugby coaches, etc., best HCI practice will permeate life ..... My conclusion, therefore, is that HCI's purpose, and consequently the future of its professionals, is to ensure that computer technology is absorbed into our culture and that they .... achieve transparency.”*

Ten years on Williamson would probably be disappointed. He **will** be proved right in the end, however. Commercial realities dictate that SE must get usability right.

However, developing usable applications will require, in the short to medium term, test managers who understand this, can insist on effective, early formative testing and can sway project managers and users. The final section will consider how test managers can do this.

# Conclusion - What Can Test Managers Do to Improve Matters?

The question of practical responses that test managers can employ to deal with projects where usability has not been considered seems to have received little attention. Crispin & House (S30, 2002, p241-7) do address it indirectly in the context of Extreme Programming. They advocate certain techniques for test managers committed to XP who find themselves working on conventional projects. These centre on trying to get their role redefined as Quality & Testing Manager, giving them the right to intervene in earlier stages to try and anticipate quality problems. As seen earlier, usability testing is fatally compromised if it is performed only at the end of the development. If it is to be effective, then the test manager **must** try to ensure it is deployed earlier, i.e. that it takes the form of formative evaluation (shaping and improving the product during the construction), rather than summative (checking whether the finished product meets the requirements).

This ties in with what has been seen in practice when test managers have spotted that the non-functional requirements were inadequate, and there were clearly informal criteria against which the application would be judged but which did not appear in the formal contract or requirements. These criteria are often quality factors, such as correctness, reliability, efficiency, security, maintainability, testability, flexibility, portability, reusability, interoperability and of course usability (McCall, S90, 1994). They dictate **how** the system will work, as opposed to **what** it will do.

Focussing on these quality factors can entail test managers checking requirements, discussing the implications with users and developers, insisting on greater clarity and detail, and suggesting that implicit assumptions are turned into explicit requirements. This can create initial friction and disagreement, because the short term impact means additional complications and work, but it ultimately makes the development less fraught since it helps prevent disagreements and disputes in the stressful period of acceptance testing. Such an approach can cut the work required over the full life of the project, and certainly helps improve the product and the developers' reputation.

Unfortunately, no trace has been found of academic studies of such pragmatic techniques. This is possibly because academics tend to be more interested in extending boundaries, and reaching for excellence, rather than looking at how practitioners are trying to advance from complete chaos to an acceptable level of disorganisation.

The Crispin & House approach is prima facie entirely consistent with the V Model of testing, which gives testers the right to intervene and shape requirements. However, this right is probably less honoured in practice than it should be. Again, this is an area of greater interest to commercial commentators than academics.

Rothman (S88, 2000 and S89, 1999) has written interesting articles arguing that test managers must adopt the sort of proactive approach that this dissertation suggests. There is

always a danger that testing will be neglected and have its budgets and schedules squeezed. Test managers therefore need to develop a deep understanding of what applications are meant to be doing, and how their organisations actually work, then use that knowledge to influence the outcomes they want. They also need to ensure that the requirements are derived and documented properly, with implicit assumptions about the application's functions and quality exposed so that a full, detailed and objective test plan can be produced.

A related issue, not mentioned by Rothman, is that the project must have a change management regime that allows controlled amendments to the requirements once the project is underway. If the testers continually unearth problems or gaps in the requirements, but the budget and schedule have already been set in stone then the result could be conflict. A proactive test manager should examine the change management rules and flag up the potential problems before they become real.

Another approach, which fits neatly with the Crispin/Rothman tactics, would be to lobby for the introduction of Nielsen's Discount Usability Engineering (DUE) method (U03, 1993, pp17-20, & U59 1994, section 2); a cut down version of his full usability engineering method, designed for just this scenario.

DUE uses three techniques for formative evaluation, i.e. to try and ensure that potential problems are identified early enough to improve shape the design.

✘ **Scenarios (i.e. prototypes)**

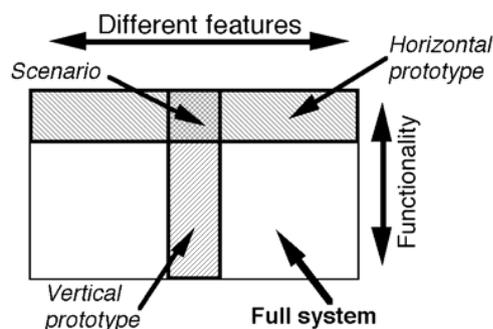


Figure 13- Nielsen's "scenario" model (U59, 1994, section 2.1)

This entails taking slices of the full projected application, either horizontal, which would take the full appearance of a particular layer of the application, without the detailed functionality behind it, or vertical, which would take the full functionality for a particular feature without any of the surrounding features. Each of these slices would represent a particular scenario that needs to be tested out, and cheap prototypes (perhaps just paper-based) would be built, tested and amended for each.

✘ **Simplified thinking aloud**

Thinking aloud requires users to work their way through a scenario, or prototype, explaining their thoughts as they go. Their words and actions are recorded, to provide evidence for the usability of the product.

## ✘ **Heuristic evaluation**

This involves inspecting the proposed interface for usability based on ten basic usability principles (see Appendix B).

Krug (U32, 2006, p137) provides what he claims to be an “*even more drastic*” version of DUE; Lost-Our-Lease Testing. In truth, Krug’s version focusses on the management of cut-down testing, rather than the techniques, so it complements, rather than competes with Nielsen’s DUE.

One of the respondents to the questionnaire (the testing consultancy) argued strongly that DUE provides a practical, effective response to the common situation where test managers find themselves on projects that are paying insufficient attention to usability. It requires considerable confidence and strength of mind on the part of test managers, but it is possible for them to introduce significant formative testing during the design stages, and prevent usability problems being fossilised in the finished design.

This is born out by the findings of Jones (U170, 2003), studying a project at the US Federal Government public health agency, Cooper (U171, 2005), studying the development of a medical decision support system, and Yao (U172, 2000), examining the development of a web-based reference application. All reported positively, saying the results were better than would otherwise have been achieved.

Cooper reported (p4) that “*DUE was shown to be an effective method for discovery of meaningful usability improvements at minimal cost and effort.*”

Jones wrote (p337); “*the benefits gained from DUE with relatively little cost have created opportunities for CDC’s Injury Center to conduct additional usability testing and heuristic evaluation ... The positive results also have opened dialogue about usability issues on several Injury Center web efforts and helped me secure more time to address usability on other projects.*”

Yao (p5); “*Formal usability engineering can be costly to any project budget but by employing DUE, a cost effective method which provides clear identification of problems is applied. It is simple in design, and in the real world, stands a better chance of being applied and reaping the rewards towards an improved product.*”

There are sceptics about DUE, however. Gray (U173, 1995, p2) wonders whether anyone but trained HCI practitioners can get acceptable results. “*While DUE may produce “bargain” interfaces, we should be careful that we do not simply get what we paid for. ...Shortcutting the time required to do (task) analyses may make interface design faster but the result is no bargain.*”

Mayhew (U174, 2001) agrees. “*My opinion ... is that very fast, very cheap DUE methods will not sufficiently reduce project risk and insure (required returns) .... Be prepared to invest time*

*and money ... with a highly structured, time-intensive program of proven methods and techniques by qualified experts.”*

The stance of the sceptics is essentially that full usability engineering is more effective than DUE, which ultimately provides poor value; i.e. best practice is better than good practice. This may be true, but it is unhelpful if the reality is a choice between good practice and bad practice. The purist stance also ignores the benefit reported by Jones above in using DUE to sell usability engineering to a sceptical organisation.

The consultancy which completed a questionnaire for this dissertation argued that DUE can be effective even in a traditional, waterfall development. The situation may be difficult. The outcome will almost certainly be worse than it should have been, i.e. if a less damaging lifecycle model had been chosen. However, a strong test manager can still carry out some effective usability testing and make a bad situation better than it could have been.

It is important that test managers remember that the Waterfall is a simplified representation of reality. A typical Waterfall project is less ordered, sequential and tidy than the model implies. There is usually an element of unpredictability, improvisation and even disorder. Test managers can make that work to their advantage, rather than letting it simply cause them problems.

Every Waterfall project has to allow for an element of iteration in the testing stage. The system is tested against the requirements, faults are found, fixes are applied by the developers, and the system is re-tested.

Test managers must try to ensure that the schedule allows for an adequate number of cycles, but less obviously, they must argue for an appropriate defect management process, with carefully defined defect severity and priority levels. This is crucial for usability. Appendix C contains a sample defect management process.

Typically, such a process would have four severity levels. Levels 3 and 4 would be defects for which there was a workaround. Level 4 defects would often be cosmetic. The exit criteria for each testing stage, and the acceptance criteria for the final product, should refer to these defect categories. A typical set of targets would be;

- ✘ No severity 1 or severity 2 defects
- ✘ No more than 10 severity 3 defects
- ✘ No more than 25 severity 4 defects

These thresholds, the definitions, and the process for assigning, reviewing and agreeing the categorisation of severities must be thrashed out and agreed at the start of the project.

There is a major implication for usability in that usability problems are usually assigned to levels 3 or 4, regardless of how serious they are, because there **is** a workaround. It is common for applications to be implemented with large numbers of outstanding, trivial defects. As Read wrote (U22, 2003, p2), see page 37, serious usability problems can be the result of a large

number of apparently trivial defects, which were not thought sufficiently serious to delay implementation, i.e. severity 3 and 4 defects.

A test manager who is keen to introduce usability testing surreptitiously, can define a defect management process that gives user management the right to bundle a group of severity 3 and 4 defects into a single severity 2 defect, which could effectively force the developers to fix them before implementation. Project and test managers sometime give users this right if they are concerned about agreeing acceptance criteria with large numbers of severity 3 and 4 defects. There is an obvious danger to the developers in doing so, but it can be a valuable way of building trust.

If the defect management process makes explicit allowance for the treatment of usability problems, and if users have the right to challenge severity classifications, and bundle minor defects into a single high-severity defect then that can help focus the mind of the developers on usability matters, and make it easier for the test manager to introduce the sort of formative usability evaluation that will help the developers to get it right earlier.

The counter-argument to all this is that making DUE work, and instituting a tough defect management regime, requires a strong commitment on the part of the project manager and the client, whom the test manager must win over. If they were disposed towards usability it is likely that they would have done something themselves. It is certainly possible that the test manager can gain their agreement, but active and strong commitment is perhaps asking too much.

Nevertheless, the test manager should always push. All professionally run SE projects require the test manager to write a Test Strategy at the start. The creation and agreement of this strategy must be the vehicle to force usability concerns into projects. It might not be possible for test managers to make the world a better place; but it is always possible to improve a small part. Visible proof of limited success for the test manager will make it easier to gain true commitment next time round.

# References

- S56 Agresti, W. (1986) "What are the new paradigms?" In Agresti, W., (Ed.). "New Paradigms for Software Development" (Tutorial). Washington, DC: IEEE Computer Society, pp 6-10.
- U165 Ambler, S. (2006) "Agile Modeling and eXtreme Programming (XP)". March 2006. Author is Practice Leader for Agile Development with the IBM Methods group, but this was written in his own right just before he joined IBM. [www.agilemodeling.com/essays/agileModelingXP.htm](http://www.agilemodeling.com/essays/agileModelingXP.htm) [accessed 13th January 2007]
- U166 Ambler, S. (2006) "Feature Driven Development (FDD) and Agile Modeling". January 2006. [www.agilemodeling.com/essays/fdd.htm](http://www.agilemodeling.com/essays/fdd.htm) [accessed 13th January 2007]
- U129 Anttila, H., Back, R., Ketola, P., Konkka, K., Leskelä, J., Rysä, E. (2002) "Combining Stepwise Feature Introduction with User-Centric Design". Turku (Finland) Centre for Computer Science TUCS Technical Report No 495, December 2002. [www.tucs.fi/publications/attachment.php?fname=TR495.pdf](http://www.tucs.fi/publications/attachment.php?fname=TR495.pdf) [accessed 10th December 2006]
- U81 Artman, H. (2002) "Procurer Usability Requirements: Negotiations in Contract Development". Proceedings of the second Nordic conference on Human-Computer Interaction (NordiCHI) 2002 . pp 61-70.
- U131 Bannon, L. (1995) "The Politics of Design: Representing Work" Communications of the ACM Volume 38 , Issue 9 (September 1995) pp 66-68.
- U134 Bannon, L. (1991) "From Human Factors to Human Actors: The Role of Psychology and Human-Computer Interaction Studies in Systems Design". Chapter in Greenbaum, J. & Kyng, M. (Eds.) "Design at work.: Cooperative Design of Computer Systems" Hillsdale: Lawrence Erlbaum Associates. pp 25-44. Picked up from [www.ul.ie/~idc/library/papersreports/LiamBannon/6/HFHA.html](http://www.ul.ie/~idc/library/papersreports/LiamBannon/6/HFHA.html) [accessed 11th December 2006]
- U146 Bannon, L. (1997) "Dwelling in the "Great Divide": The Case of HCI & CSCW". Chapter in Bowker, G., Gasser, L., Star, L., Turner, W. (eds.) "Social Science, Technical Systems and Cooperative work : Beyond the Great Divide". Lawrence Erlbaum Associates Inc, US.
- U145 Bannon, L., Bødker, S. (1991). "Beyond the Interface: Encountering Artifacts in Use". Chapter 12 in Carroll, J. (ed.) "Designing Interaction: Psychology at the human-computer interface". New York: Cambridge U.P.
- S47 Bansler, J., Bødker, K. (1993). "A reappraisal of structured analysis. Design in an organisational context", ACM Transactions on Information Systems, Vol. 11 No.2.
- S84 Basili, V., Caldiera, G., Rombach, H. (1994) "Goal Question Metric Approach". Marciniak, J. (editor) "Encyclopedia of Software Engineering", Vol 1, 1st edition. pp. 528-532, John Wiley & Sons, Inc.
- U11 Bass, L., and John, B. (2003). "Linking Usability to Software Architecture Patterns Through General Scenarios" The Journal of Systems and Software, Vol 66, Issue 3, 15 June 2003, Pages 187-197 [hdcp.org/Publications/BassJohnJSSpublished.pdf](http://hdcp.org/Publications/BassJohnJSSpublished.pdf) [accessed 25th April 2006]
- U12 Bass, L. and John, B. (2000) "Achieving usability through software architectural styles". Conference on Human Factors in Computing Systems, The Hague, 2000. <http://portal.acm.org/citation.cfm?doid=633292.633387> [accessed 25th April 2006]
- U14 Bass, L., Johns B., and Adams, R. (2003) "Communication across the HCI/SE divide: ISO 13407 and the Rational Unified Process". In J. Jacko and C. Stephanidis (Eds.) Human-Computer Interaction: Theory and Practice, Lawrence Erlbaum, Mahway, NJ.
- U16 Bass, L., John, B., Sanchez-Segura, M., Adams, R. (2004) "Bringing Usability Concerns to the Design of Software Architecture" 9th IFIP Working Conference on Engineering for Human-Computer Interaction and the 11th International Workshop on Design, Specification and Verification of Interactive Systems. Hamburg, Germany,

- July 11-13, 2004. [www.cs.cmu.edu/~bej/usa/publications/JohnEtAl-EHCI04dist.pdf](http://www.cs.cmu.edu/~bej/usa/publications/JohnEtAl-EHCI04dist.pdf) [accessed 25th April 2006]
- S85 Beck, K., Andres, C. (2005) *"Extreme Programming Explained"*. 2nd edition. Addison-Wesley.
- U139 Berry, D. (2000). *"The user experience - The iceberg analogy of usability"*. IBM developerWorks online magazine. 1st October 2000. [www-128.ibm.com/developerworks/library/w-berry](http://www-128.ibm.com/developerworks/library/w-berry) [accessed 18th December 2006]
- U121 Bødker, S. (2006). *"When Second Wave HCI meets Third Wave Challenges"*. Proceedings of the 4th Nordic conference on Human-Computer Interaction (NordiCHI) 2006, 14-18 October 2006
- U151 Bødker, S. (1990) *"Through the Interlace - a Human Activity Approach to User Interface Design"*. Lawrence Erlbaum, New Jersey.
- S37 Boehm, B. (1988). *"A Spiral Model of Software Development and Enhancement"*. Computer Magazine (IEEE Computer Society), May 1988, pp61-72. [doi.ieeecomputersociety.org/10.1109/2.59](http://doi.ieeecomputersociety.org/10.1109/2.59)
- S06 British Computer Society. (2001). Specialist Interest Group in Software Testing for BS7925. [www.testingstandards.co.uk](http://www.testingstandards.co.uk) [accessed 8th January 2007]
- S71 Britton, C. (2006). *"Take it to the bridge"*. IT Now magazine, British Computer Society, November 2006.
- S72 Britton, C. (2006). *"Making IT Application Design an Engineering Discipline"*. IT Modeller consultancy website, October 2006. [www.itmodeller.co.uk/docs/Engineering.pdf](http://www.itmodeller.co.uk/docs/Engineering.pdf) [accessed 17th November 2006]
- S27 Brooks, F. (1987). *"No Silver Bullet: Essence and Accidents of Software Engineering"*, Computer, Vol. 20, No. 4 (April 1987) pp. 10-19. [inst.eecs.berkeley.edu/~maratb/readings/NoSilverBullet.html](http://inst.eecs.berkeley.edu/~maratb/readings/NoSilverBullet.html) [accessed 7th August 2006]
- S17 Business eSolutions. (2002). *"Project Lifecycle Models: How They Differ and When to Use Them"*. Business eSolutions Website. [www.business-esolutions.com/isl.htm#purewaterfall](http://www.business-esolutions.com/isl.htm#purewaterfall) [accessed 29th November 2006]
- S44 Carnegie Mellon Software Engineering Institute, Capability Maturity Model Integration (CMMI) website. [www.sei.cmu.edu/cmmi/general/general.html](http://www.sei.cmu.edu/cmmi/general/general.html) [accessed 20th October 2006]
- S80 Coad, P., Yourdon, E. (1991). *"Object-Oriented Analysis"*. 2nd edition. Yourdon Press.
- S79 Colter, M. (1982). *"Evolution of the Structured Methodologies"*. In Cougar, J., Colter, M., Knapp, R. *"Advanced System Development Techniques"*. Wiley & Sons, New York, pp73-96.
- U37 Constantine, L. (2002). *"Process Agility and Software Usability: Toward Lightweight Usage-Centered Design"*. Constantine & Lockwood Ltd, University of Technology, Sydney. 2002. [www.foruse.com/articles/agiledesign.pdf](http://www.foruse.com/articles/agiledesign.pdf) [accessed 22nd September 2006]
- U26 Cooper, A. (2004). *"The Inmates Are Running the Asylum: Why High-tech Products Drive Us Crazy and How to Restore the Sanity"*. Que 2004.
- U171 Cooper, J. (2005). *"Discount Usability Evaluation of an Analytic Hierarchy Process Based Clinical Computer Application"*. Master's dissertation for Oregon Health & Science University's Department of Medical Informatics & Clinical Epidemiology.
- S30 Crispin, L. House, T. (2002). *"Testing Extreme Programming"*. Addison Wesley, 2002.
- S40 Curtis, B., Iscoe, N., Krasner, H. (1988) *"A field study of the software design process for large systems"*. Communications of the ACM, Volume 31, Issue 11 (November 1988), pp1268-1287.

- S57 Curtis, B., Iscoe, N., Krasner, H., Shen, V. (1987). *"On Building Software Process Models Under the Lamppost"*. International Conference on Software Engineering, Proceedings of the 9th international conference on Software Engineering, Monterey, California, 1987. pp96-103.
- S32 DeGrace, P., Stahl, L. (1993). *"The Olduvai Imperative: CASE and the State of Software Engineering Practice"*, Yourdon Press, Prentice Hall.
- U152 de Montmollin, M. (1991). *"Analysis and models of operators' activities in complex natural life environments"*. In J. Rasmussen & H. B. Andersen (Eds.) *"Human-Computer Interaction: Research directions in cognitive science"*. Lawrence Erlbaum, Hillsdale, pp. 95-112.
- S46 Defense Science Board Task Force On Military Software - Report (extracts), (1987) . ACM SIGAda Ada Letters Volume VIII , Issue 4 (July/Aug. 1988) pp35-46 doi.acm.org/10.1145/62162.62163 [accessed 20th October 2006]
- S60 Department of Defense Standard 2167A (DOD-STD-2167A), (1988). *"Defense System Software Development"*, US Government defence standard.
- S61 Department of Defense Standard 2167 (DOD-STD-2167). (1975) *"Defense System Software Development"*, US Government defence standard.
- S16 Doll, S. (2002). *"Waterfall development for new managers"*. Builder UK, on-line magazine and forum for SE developers. uk.builder.com/manage/project/0,39026588,20266893,00.htm [accessed 30th November 2006]
- S86 DSDM website www.dsdm.org/default.asp [accessed 13th January 2007]
- U109 Earthy, J. (2000). *"Usability Maturity Model: Processes"*, report, Lloyds Register & European Commission. Available at www.processforusability.co.uk/Usability\_test/html/downloads.html [accessed 30th November 2006]
- U55 Edmonds, E. (1991). *"The Separable User Interface"*. Academic Press, London.
- U56 Evers, M. (1999). *"Adaptability Problems of Architectures for Interactive Software"* Wisdom 99 13th European Conference on Object-Oriented Programming. Workshop on Interactive System Design and Object Models, 1999. math.uma.pt/wisdom99/papers/evers/evers.html
- S62 Fayad, M., Laitinen, M., Ward, R. (2000). *"Software Engineering in the Small"*. Communications of the ACM, Volume 43 , Issue 3 , March 2000, pp 115-118.
- U113 Ferre, X., Juristo, N., Windl, H., Constantine, L. (2001). *"Usability Basics for Software Developers"*. IEEE Software Magazine. January/February 2001 (Vol. 18, No. 1) pp. 22-29.
- S73 Firesmith, D. (2006). *"Engineering Safety-Related Requirements for Software-Intensive Systems "*. Carnegie Mellon Software Engineering Institute, 18th Annual Software Engineering Process Group Conference (SEPG), 2006. www.sei.cmu.edu/programs/acquisition-support/presentations.html [accessed 15th December 2006]
- S09 Fitzgerald, B. (1994). *"The Systems Development Dilemma": Whether to Adopt Formalised Systems Development Methodologies or Not?"*, in Baets, W. (Ed) Proceedings of Second European Conference on Information Systems, Nijenrode University Press, Holland, pp691-706.
- S11 Fitzgerald, B. (1995). *"A Descriptive Framework for Investigating Problems in the Application of Systems Development Methodologies"*. In Jayaratna, N. et al (Eds) Proceedings of Third Conference on Information Systems Methodologies, BCS Publications, Swindon, pp27-38.
- S12 Fitzgerald, B. (1996). *"Formalised Systems Development Methodologies: a Critical Perspective"*, Information Systems Journal, January 1996.

- S01 Fitzgerald B., Russo N., Stolterman, E. (2002). *"Information Systems Development - Methods in Action"*, McGraw Hill.
- S52 Floyd, C. (1988). *"Outline of a paradigm change in software engineering. In Computers and Democracy: A Scandinavian Challenge"*. ACM SIGSOFT Software Engineering Notes, Volume 13 , Issue 2 (April 1988). pp25-38
- U143 Foley, J. (1991). *"Future Directions in User-Computer Interface Software"*. Conference proceedings on Organizational Computing Systems, Atlanta. pp289-297.
- U60 Folmer, E., Bosch, J. (2004). *"Cost Effective Development of Usable Systems; Gaps between HCI and SE"* International Conference of Software Engineering 2004. Workshop Bridging the Gaps Between Software Engineering and Human-Computer Interaction.
- U158 Folmer, E., Bosch, J. (2004). *"Experiences with Software Architecture Analysis of Usability"*. 2004 paper produced as part of the Status (Software Architecture that Supports Usability), an Esprit project financed by the European Commission. [is.ls.fi.upm.es/status/results/publications.html](http://is.ls.fi.upm.es/status/results/publications.html) [accessed 7th January 2007]
- U155 Folmer, E., van Gurp, J., Bosch, J. (2003). *"A Framework for capturing the Relationship between Usability and Software Architecture"*. Software Process: Improvement and Practice, Volume 8, Issue 2. Pages 67-87. April/June 2003.
- U77 Gellner, M., Forbrig, P. (2001). *"Modelling the Usability Evaluation Process with the Perspective of Developing a Computer Aided Usability Evaluation (CAUE) System"*. Position paper in *"Usability Throughout the Entire Software Development Lifecycle - A Summary of the INTERACT 2001 Workshop"*. Official workshop for the International Federation for Information Processing (IFIP) working group 13.2 on "Methodologies for User Centred Systems Design", at the INTERACT 2001 conference in Tokyo, Japan, July 9, 2001. Available from [www.it.uu.se/research/publications/reports/2001-026/2001-026.pdf](http://www.it.uu.se/research/publications/reports/2001-026/2001-026.pdf). See U10. [accessed 9th November 2006]
- U119 Gellner, M., Forbrig, P. (2003). *"Extreme Evaluations – Lightweight Evaluations for Software Developers"*. INTERACT 2003 Workshop, Sep 1-2, 2003. Zurich, Switzerland. *"Closing the Gaps: Software Engineering and Human-Computer Interaction"*. pp75-80. [www.se-hci.org/bridging/interact/p75-80.pdf](http://www.se-hci.org/bridging/interact/p75-80.pdf) [accessed 7th December 2006]
- S29 Glass, R. (2006). *"Software Conflict 2.0: The Art and Science of Software Engineering"*. Developer Dot Star Books, 2006.
- U153 Göransson, B., Lindt, M., Pettersson, E., Sandblad, B., Schwalbe, P. (1987). *"The Interface is often not the problem"*. In Carroll, J., Tanner, B. (eds.) *"Human Factors in Computer Systems IV"*, North-Holland, New York. pp133-136.
- U28 Gould, J. and Lewis, C. (1985). *"Designing for usability: key principles and what designers think."* Communications of the ACM, March 1985. pp300-311. [www.research.ibm.com/compsci/spotlight/hci/p300-gould.pdf](http://www.research.ibm.com/compsci/spotlight/hci/p300-gould.pdf) [accessed 13th August 2006]
- U123 Gould, J. and Lewis, C. (1983). *"Designing for usability: key principles and what designers think."* Proceedings of the SIGCHI conference on Human Factors in Computing Systems, Boston, Massachusetts. pp50-53.
- U173 Gray, W. (1995) *"Discount or Disservice? Discount Usability Analysis - Evaluation at a bargain price or simply damaged merchandise?"*. Conference companion on Human factors in computing systems CHI '95. ACM Press.
- U103 Green, M. (1985). *"The University of Alberta User Interface Management System"*. Proceedings of the 12th annual conference on Computer graphics and interactive techniques, 1985. pp205-213.
- U133 Grier, R. (2005). *"Next generation human-computer interfaces."*. Proceedings of the Human Systems Integration Symposium, 2005. [www.aptime.com/hsi\\_publications.php](http://www.aptime.com/hsi_publications.php) [accessed 11th December 2006]

- U84 Grudin, J. (1991). *"Interactive Systems: Bridging the Gaps Between Developers and Users"*. Computer April 1991 (Vol. 24, No. 4) pp 59-69.  
doi.ieeecomputersociety.org/10.1109/2.76263
- U85 Grudin, J. (1996). *"The Organizational Contexts of Development and Use"*. ACM Computing Surveys. Vol 28, issue 1 (Mar. 1996), pp 169-171.  
doi.acm.org/10.1145/234313.234384
- U10 Gulliksen, J., Boivie, I. (2001). *"Usability Throughout the Entire Software Development Lifecycle - A Summary of the INTERACT 2001 Workshop"*. Official workshop for the International Federation for Information Processing (IFIP) working group 13.2 on "Methodologies for User Centred Systems Design", at the INTERACT 2001 conference in Tokyo, Japan, July 9, 2001. Available from [www.it.uu.se/research/publications/reports/2001-026/2001-026.pdf](http://www.it.uu.se/research/publications/reports/2001-026/2001-026.pdf) [accessed 9th November 2006]
- U159 Gulliksen, J., Boivie, I., Lantz, A. (1999). *"How to make User Centred Design Usable"*. Summary of the INTERACT '99 workshop. Edinburgh 1999.  
cid.nada.kth.se/pdf/cid\_72.pdf [accessed 7th January 2007]
- U111 Gulliksen, J., Goransson, B. (2003). *"Usability Design - integrating user-centred systems design in the software development process"*. Tutorial at HCI International 2003.  
[www.it.uu.se/research/hci/acsd/T27\\_Usability\\_Design\\_at\\_HCI\\_International\\_2003.pdf](http://www.it.uu.se/research/hci/acsd/T27_Usability_Design_at_HCI_International_2003.pdf) [accessed 30th November 2006]
- U168 Gulliksen, J., Boivie, I., Persson, J., Hektor, A., Herulf, L. (2004). *"Making a difference – a survey of the usability profession in Sweden"*. NordiCHI '04, October 23-27, 2004 Tampere, Finland
- S58 Hallows, J. (1998). *"Information Systems Project Management"*. 1st edition. AMACOM, New York.
- S13 Hallows, J. (2005). *"Information Systems Project Management"*. 2nd edition. AMACOM, New York.
- U66 Hartson, H. (1989). *"User-Interface Management Control and Communication"*. IEEE Software Magazine. Jan-Feb 1989, Vol 6 No 1. pp62-70.
- U65 Hartson, H., Hix, D. (1989). *"Human-Computer Interface Development: Concepts and Systems for Its Management"*. ACM Computing Surveys (CSUR) Volume 21, Issue 1, March 1989. pp5-92.
- U105 Hartson, H., Hix, D. (1993). *"Developing User Interfaces: Ensuring Usability Through Product and Process"*. Wiley Professional Computing.
- U79 Holmlid, S. (2002). *"Adapting users: Toward a theory of use quality"*. PhD thesis, Linköping University, Sweden. [www.ida.liu.se/~steho/publications/index.htm](http://www.ida.liu.se/~steho/publications/index.htm) [accessed 10th November 2006]
- U80 Holmlid, S. Artman, H. (2003). *"A Tentative Model for Procuring Usable Systems"*. 10th International Conference on Human-Computer Interaction, 2003.  
[www.nada.kth.se/~artman/Articles/Conference/HClint03.pdf](http://www.nada.kth.se/~artman/Articles/Conference/HClint03.pdf) [accessed 11th November 2006]
- U33 Holzinger, A., Errath, M., Searle, G., Thurnher, B., Slany, W. (2005). *"From Extreme Programming and Usability Engineering to Extreme Usability"*. Proceedings of the 29th Annual International Computer Software and Applications Conference, 2005 (COMPSAC'05)  
[csdl.computer.org/dl/proceedings/compsac/2005/2413/02/241320169.pdf](http://csdl.computer.org/dl/proceedings/compsac/2005/2413/02/241320169.pdf) [accessed 18th September 2006]
- U125 Iivari, N. (2006). *"Understanding the Work of an HCI Practitioner"*. Proceedings of the 4th Nordic conference on Human-computer interaction: changing roles. (NordiCHI) 2006. Oslo, Norway. pp 185-194.
- U07 ISO13407. (1999). *"Human-centred design processes for interactive systems"* International Organisation for Standardisation.

- U130 Iversen, O., Kanstrup, A., Petersen, M. (2004). "A Visit to the 'New Utopia' Revitalizing Democracy, Emancipation and Quality in Cooperative Design". Proceedings of the third Nordic conference on Human-computer interaction, (NordiCHI) Tampere, Finland. 2004. pp171-179.
- U35 Jokela, T., Abrahamsson, P. (2004). "Usability Assessment of an Extreme Programming Project: Close Co-operation with the Customer Does Not Equal to Good Usability". In "Product Focused Software Process Improvement" (Lecture Notes in Computer Science), pp 393-407. Springer Berlin.
- U170 Jones, C. (2003). "Lessons Learned from Discount Usability Engineering for the U.S. Federal Government". Technical Communication, (Journal of the Society for Technical Communication), Volume 50, Number 2, May 2003, pp232-246.
- U114 Juristo, N., Windl, H., Constantine, L. (2001). "Introducing Usability". IEEE Software Magazine. January/February 2001 (Vol. 18, No. 1) pp. 20-21.
- U17 Kazman, R., Gunaratne, J., Jerome, B. (2003). "Why Can't Software Engineers and HCI Practitioners Work Together?" In J. Jacko and C. Stephanidis (Eds.) Human-Computer Interaction: Theory and Practice, Lawrence Erlbaum, Mahway, NJ.
- S51 Kruchten, P. (2004). "Going Over the Waterfall with the RUP". IBM Rational Developer Works article, April 2004. [www-128.ibm.com/developerworks/rational/library/4626.html#N1023E](http://www-128.ibm.com/developerworks/rational/library/4626.html#N1023E) [accessed 19th November 2006]
- U32 Krug, S. (2006). "Don't Make Me Think!: A Common Sense Approach to Web Usability". 2nd edition. New Riders.
- U97 Kuutti, K., Bannon, L. (1993). "Searching for Unity Among Diversity: Exploring the 'Interface' Concept". Proceedings of the SIGCHI conference on Human factors in computing systems. 1993.
- S53 Langefors, B. (1973). "Theoretical Analysis of Information Systems", Auerbach, Philadelphia. 1973.
- U154 Laurel, B. (1990). (Ed.) "The Art of Human-Computer Interaction Design". Addison-Wesley, Reading.
- U160 Lesso, R. (2005) "An E-learning System for Basic Metrology". MSc thesis. Technical University of Denmark. [www.simet.gob.mx/elearning/Thesis/thesis.pdf](http://www.simet.gob.mx/elearning/Thesis/thesis.pdf) [accessed 10th January 2007]
- U104 Lewis, C., Rieman, J. (1994). "Task-Centered User Interface Design: A Practical Introduction". University of Colorado internet book, 1994. [www.hcibib.org/tcuid/tcuid.pdf](http://www.hcibib.org/tcuid/tcuid.pdf) [accessed 18th November 2006]
- U164 Lim, K., Long, J. (1994). "The MUSE method for usability engineering". Cambridge University Press, Cambridge, UK.
- U102 Löwgren, J. (1988). "History, State and Future of User Interface Management Systems". ACM SIGCHI Bulletin, Volume 20, Issue 1 (July 1988). pp32-44. [doi.acm.org/10.1145/49103.49105](https://doi.org/10.1145/49103.49105). [accessed 17th December 2006]
- S90 McCall, J. (1994). "Quality Factors", Marciniak, J. (editor) "Encyclopedia of Software Engineering", Vol 2, 1st edition. pp. 958-969, John Wiley & Sons, Inc.
- S10 McCracken, D., Jackson, M. (1982). "Life Cycle Concept Considered Harmful", ACM SIGSOFT Software Engineering Notes, Vol 7 No 2, April 1982. [doi.acm.org/10.1145/1005937.1005943](https://doi.org/10.1145/1005937.1005943) [accessed 25th July 2006]
- U162 Mandel, T. (1997). "The Elements of User Interface Design". John Wiley & Sons.
- S25 Marick, B. (2000). "New Models for Test Development" of Testing Foundations, March 2000. [www.testing.com/writings/new-models.pdf](http://www.testing.com/writings/new-models.pdf) [accessed 1st August 2006]
- U02 Mayhew, D. (1999). "The Usability Engineering Lifecycle", Morgan Kaufmann, San Francisco.

- U122 Mayhew, D. Deborah J. Mayhew & Associates website drdeb.vineyard.net/index.php [accessed 8th December 2006]
- U174 Mayhew, D. (2001). "Discount usability vs. usability gurus: A middle ground.". TaskZ.com, an InformationWeek.com website, "Executive Resource for User Centred Design". www.taskz.com/ucd\_discount\_usability\_vs\_gurus\_indepth.php [accessed 21st January 2007]
- U34 Meszaros, G., Aston, H. (2006). "Adding Usability Testing to an Agile Project". pp. 289-294, AGILE 2006 (AGILE'06). csdl.computer.org/dl/proceedings/agile/2006/2562/00/25620289.pdf [accessed 19th September 2006]
- S19 National Audit Office. (2003). "Review of System Development - Overview". www.nao.org.uk/intosai/edp/Audit%20Guides/Review%20of%20systems%20development%20-%20overview.pdf [accessed 28th July 2006 - still current guidance]
- U45 Nelson, E. (2002). "Extreme Programming vs. Interaction Design", Interview with Kent Beck & Alan Cooper. Fawcett Technical Publications website. www.fawcette.com/interviews/beck\_cooper [accessed 28th September 2006]
- S82 Newberry, G. (1995). "Changes from DOD-STD-2167A to MIL-STD-498". Crosstalk - the Journal of Defense Software Engineering, April 1995. www.stsc.hill.af.mil/crosstalk/1995/04/Changes.asp [accessed 23rd November 2006]
- U03 Nielsen, J. (1993). "Usability Engineering" Morgan Kaufmann, San Francisco, 1993
- U59 Nielsen, J. (1994). "Guerrilla HCI: Using Discount Usability Engineering to Penetrate the Intimidation Barrier" in Bias, R. G. and Mayhew, D. J. eds. "Cost Justifying Usability". Academic Press, Boston, 1994, 242-272.
- U156 Nunes N., Cunha, J. (2002). "Wisdom — A UML based architecture for interactive systems". 1 Taller Internacional de Modelado de Interfaces de Usuario (1st International Workshop for User Interface Models - TIMIU'2002), Denia, Spain, 2002.
- S54 Olerup, A. (1991). "Design approaches: a comparative study of information system design and architectural design".. The Computer Journal, Vol 34, issue 3, 1991, pp 215-224.
- S74 Optimex Inc. (2006). IT training providers. www.optimexinc.com/courses.asp?id=2601 [accessed 15th December 2006]
- S14 PA Consulting website. (undated). www.paconsulting.com/insights/managing\_complex\_projects/stright\_forward\_projects/Waterfall.htm [Accessed 29th November 2006]
- U126 Pekkola, S., Kaarilahti, N., Pohjola, P. (2006). "Towards formalised end-user participation in information systems development process: bridging the gap between participatory design and ISD methodologies". Proceedings of the ninth conference on participatory design: "Expanding boundaries in design", Volume 1. Trento, Italy, August 2006. pp21-30.
- S33 Peters, L. (1981). "Software Design: Methods and Techniques". Yourdon Press, New York. 1981.
- U141 Pfaff G., ten Hagen, P. (1985). "Seeheim Workshop on User Interface Management Systems", Springer-Verlag, Berlin.
- U09 Philip, R., Rourke, C. (2006). User Vision. "Beyond usability testing: user-centred design and organisational maturity - A Mercurytide white paper: 16th March 2006" www.uservision.co.uk/usability\_articles/usability\_ucd.asp [accessed 2nd May 2006]
- U04 Preece, J., Rogers, Y., Sharp, H. (2002). "Interaction Design", Wiley, New York.
- U117 Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., Carey, T. (1994) "Human-Computer Interaction". Addison-Wesley.

- S49 Pyhäjärvi, M., Itkonen, J. Rautiainen, K. (2003) *"Increasing Understanding of the Modern Testing Perspective in Software Product Development Projects"*. IEEE Proceedings of the Hawai'i International Conference on System Sciences, 2003.
- U54 Pyla, P., Perez-Quinones, M., Arthur, J., Hartson, R. (2003). *"Towards a Model-Based Framework for Integrating Usability and Software Engineering Life Cycles"*. Bridging the SE & HCI Communities Workshop. INTERACT (Ninth IFIP TC13 International Conference on Human-Computer Interaction). 2003
- U108 Process for Usability website. (2003). For ISO TR 18529 & Usability Maturity Model. [www.processforusability.co.uk/Usability\\_test/html/umm.html](http://www.processforusability.co.uk/Usability_test/html/umm.html) [accessed 30th November 2006]
- S50 Raccoon, L. (1997). *"Fifty Years of Progress in Software Engineering"*. SIGSOFT Software Engineering Notes Vol 22, Issue 1 (Jan. 1997). pp88-104.
- U116 Radle, K., Young, S. (2001). *"Partnering Usability with Development: How Three Organizations Succeeded"*. IEEE Software Magazine. January/February 2001 (Vol. 18, No. 1) pp. 38-46.
- U91 Rakitin, S. (2005). *"Agile Methods - Beyond the Hype"* in *"Food for Thought"* newsletter published by Software Quality Consulting. July/August 2005. [www.swqual.com/newsletter/vol2/no7/vol2no7.html](http://www.swqual.com/newsletter/vol2/no7/vol2no7.html) [accessed 14th November 2006]
- U22 Read, D. (2003). *"Those "Minor" Usability Annoyances"*. Developer Dot Star - A Web Magazine for Software Developers. [www.developerdotstar.com/mag/articles/PDF/DevDotStar\\_Read\\_Usability.pdf](http://www.developerdotstar.com/mag/articles/PDF/DevDotStar_Read_Usability.pdf) [accessed 9th August 2006]
- S39 Robbins, J., Hilbert, D., Redmiles, D. (1998). *"Extending Design Environments to Software Architecture Design"*, Automated Software Engineering, Vol. 5, No. 3, July 1998, pp261-290. [www.ics.uci.edu/~redmiles/publications/J003-RHR98.pdf](http://www.ics.uci.edu/~redmiles/publications/J003-RHR98.pdf)
- U140 Rosenberg, J., Hill, R., Miller, J., Schuler, A., Shewmake, D. (1988). *"UIMSs: threat or menace?"*. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Washington, DC. 1988. pp197-200.
- S88 Rothman, J. (2000) *"The Influential Test Manager"*. Software Testing and Quality Engineering magazine. March/April 2000. [www.jrothman.com/Papers/InfluentialTestManager.html](http://www.jrothman.com/Papers/InfluentialTestManager.html) [accessed 13th January 2007]
- S89 Rothman, J., Lawrence, B. (1999). *"A Pragmatic Strategy for NOT Testing in the Dark"*. Software Testing and Quality Engineering magazine. March/April 1999. [www.jrothman.com/Papers/Pragmaticstrategies.html](http://www.jrothman.com/Papers/Pragmaticstrategies.html) [accessed 13th January 2007]
- S07 Royce, W. (1970). *"Managing the Development of Large Software Systems"*, IEEE Wescon, August 1970
- U01 Rubin J. (1994). *"Handbook of Usability Testing – how to plan, design and conduct effective tests"*, Wiley, New York.
- S41 Schoen, D. (1992). *"Designing as Reflective Conversation with the Materials of a Design Situation"*. Knowledge-Based Systems. vol. 5, no. 1. pp3-14.
- U127 Schuler, D. (2006). *"Participatory Design"*. Paper on the website for Computer Professionals for Social Responsibility, October 2006. [diac.cpsr.org/cgi-bin/diac02/pattern.cgi/public?pattern\\_id=298](http://diac.cpsr.org/cgi-bin/diac02/pattern.cgi/public?pattern_id=298) [accessed 10th December 2006]
- U161 Seffah, A. (2004) *"Bridging the Educational Gap between SE & HCI: What Software Engineers Should Know"*. Workshop on Bridging the Gaps II: Bridging the Gaps Between Software Engineering and Human-Computer Interaction. International Conference on Software Engineering (ICSE) 2004. May 24-25, 2004, Edinburgh.
- U29 Sharp, H., Biddle, R., Gray, P., Miller, L., Patton, J. (2006). *"Agile Development - Opportunity or Fad?"*. CHI 2006, April 22–27, 2006, Montréal. [doi.acm.org/10.1145/1125451.1125461](https://doi.org/10.1145/1125451.1125461) [accessed 18th September 2006]

- S31 Shemer, I. (1987). "*Systems analysis: a systematic analysis of a conceptual model*". Communications of the ACM, June 1987, pp 506-512.
- S55 Sommerville, I. (2004) "*Software Engineering*". 7th Edition. Harlow: Pearson/Addison-Wesley.
- U36 Sousa, K., Furtado, E., and Mendonça, H. (2005). "*UPi: a software development process aiming at usability, productivity and integration*". Proceedings of the 2005 Latin American conference on Human-computer interaction CLIHC '05, October 2005. doi.acm.org/10.1145/1111360.1111368
- U167 Sousa, K., Furtado, E. (2005). "*From Usability Tasks to Usable User Interfaces*". ACM International Conference Proceeding Series; Vol. 127. Proceedings of the 4th international workshop on Task models and diagrams, Gdansk, Poland, 2005. pp103-110.
- U142 Sotirovski, D., Kruchten, P. (1995). "*Implementing Dialogue Independence*". IEEE Software magazine, November 1995. pp61-70.
- S87 Schwaber, K., Beedle, M. (2002). "*Agile Software Development with Scrum*". Prentice Hall.
- S65 Stephens, M., Rosenberg, D. (2003). "*Extreme Programming Refactored: The Case Against XP*". Apress.
- U147 Thomas, J., Kellogg, W. (1989). "*Minimizing Ecological Gaps in User Interface Design*", IEEE Software magazine, January 1989. pp. 78-86.
- U157 van Harmelen, M., Artim, J., Butler, K., Henderson, A., Roberts, D., Rosson, M., Tarby, J., Wilson, S. (1997). "*Object Models in User Interface Design*". A CHI 97 workshop. SIGCHI Bulletin, 29(4). pp55-62. acm.org/sigchi/bulletin/1997.4/harmelen.html [accessed 4th January 2007]
- S75 Ward, P. (1991). "*The evolution of structured analysis: Part 1 - the early years*". American Programmer, vol 4, issue 11, 1991. pp4-16.
- S76 Ward, P. (1992). "*The evolution of structured analysis: Part 2 - maturity and its problems*". American Programmer, vol 5, issue 4, 1992. pp18-29.
- S77 Ward, P. (1992). "*The evolution of structured analysis: Part 3 - spin offs, mergers and acquisitions*". American Programmer, vol 5, issue 9, 1992. pp41-53.
- S83 Wastell, D., Newman, M. (1993). "*The behavioral dynamics of information systems development: a stress perspective*". Accounting, Management & Information Technology, vol 3, issue 2, 121-148. 1993.
- U148 Williamson, A. (1997). "*The HCI Professional: a systemic individual?*". Interfaces (magazine of the BCS HCI Group, Autumn/Winter 1997. www.bcs-hci.org.uk/interfaces/interfaces36.pdf [accessed 3rd January 2007]
- U87 Winkler, I., Buie, E. (1995). "*HCI challenges in government contracting;a CHI '95 workshop*". ACM SIGCHI Bulletin, Volume 27, Issue 4 (October 1995). pp 35-37. doi.acm.org/10.1145/214132.214154
- U172 Yao, P, Gorman, P. (2000). "*Discount Usability Engineering Applied to an Interface for Web-based Medical Knowledge Resources*". American Medical Informatics Association 2000 Symposium. medicine.ucsd.edu/F2000/E001420.htm [accessed 21st January 2007.
- S66 Yourdon, E. (2006). "*Just Enough Structured Analysis*". Web book in Wiki form. www.yourdon.com/strucanalysis/wiki/index.php?title=Table\_of\_Contents [accessed 4th December 2006]
- S63 Yourdon, E. (1997). "*Death March - the Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects*". Prentice Hall International.
- S64 Yourdon, E. (1998). "*Rise and Resurrection of the American Programmer*". Prentice Hall International.
- S78 Yourdon, E., Constantine, L. (1977). "*Structured Design*". Yourdon Press, New York.

- S34 Zelkowitz, M. (1988). "*Resource utilisation during software development*". Journal of Systems and Software, 8, 1988. pp331-336.

# Bibliography

- S18 Adens, G. (2004) *"The Role of Risk in a Modern Software Development Process"*. Tassc Solutions technical paper/  
[www.tassc-solutions.com/downloads/The%20Role%20of%20Risk.pdf](http://www.tassc-solutions.com/downloads/The%20Role%20of%20Risk.pdf) [accessed 28th July 2006]
- U47 Agile Alliance website - [www.agilealliance.com](http://www.agilealliance.com)
- U43 Ambler, S. (2006). *"Agile Usability: User Experience Activities on Agile Development Projects"*. Author is Practice Leader for Agile Development with the IBM Methods group, but this was written in his own right just before he joined IBM.  
[www.agilemodeling.com/essays/agileUsability.htm](http://www.agilemodeling.com/essays/agileUsability.htm) [accessed 28th September 2006]
- U106 Anderson, J., Fleek, F., Garrity, K., Drake, F. (2001). *"Integrating Usability Techniques into Software Development"*. IEEE Software Magazine. January/February 2001 (Vol. 18, No. 1) pp46-53.
- U74 Antunes, H., Seffah, A., Radhakrishnan, T., Pestina, S. (2001). *"Unifying User-Centered and Use-Case Driven Requirements Engineering Lifecycle"*. Position paper in *"Usability Throughout the Entire Software Development Lifecycle - A Summary of the INTERACT 2001 Workshop"*. Official workshop for the International Federation for Information Processing (IFIP) working group 13.2 on "Methodologies for User Centred Systems Design", at the INTERACT 2001 conference in Tokyo, Japan, July 9, 2001. Available from  
[www.it.uu.se/research/publications/reports/2001-026/2001-026.pdf](http://www.it.uu.se/research/publications/reports/2001-026/2001-026.pdf). (See U10). [accessed 9th November 2006]
- S67 Avison, D., Fitzgerald, G. (2003). *"Where Now for Development Methodologies"*. Communications of the ACM, Volume 46, Issue 1 (January 2003), pp 79-82
- U46 Bankston, A. (2003). *"Usability and User Interface Design in XP"*. CC Pace (commercial consultancy). [www.ccpace.com/Resources/documents/UsabilityinXP.pdf](http://www.ccpace.com/Resources/documents/UsabilityinXP.pdf) [accessed 29th September 2006]
- U13 Bass, L. and John, B. (2003) *"Avoiding 'We can't change THAT!'"* Software Architecture and Usability. Tutorial materials presented at Conference on Human Factors in Computing Systems, CHI 2003 (Ft. Lauderdale, FL, April 5-10, 2003). & Bass, L. and John, B. (2003) *"Avoiding 'We can't change THAT!'"* Software Architecture and Usability. Tutorial materials presented at CHI 2004 (Vienna, Austria, April 24-29, 2004) <http://www.cs.cmu.edu/~bej/usa/publications.html> [accessed 1st May 2006]
- U15 Bass, L., John, B., Juristo, N., Sanchez-Segura, M. (2004). *"Usability-supporting Architectural Patterns"*. Proceedings of the 26th International Conference on Software Engineering, Edinburgh, May 2004.  
[http://hdcp.org/Publications/BassJohn\\_ICSE04.pdf](http://hdcp.org/Publications/BassJohn_ICSE04.pdf) [accessed 25th April 2006]
- U120 Benyon, D., Holland, S., Stone, D., Woodroffe, M. (1996). *"A student-centred approach to networked, multimedia courseware design"*. British HCI Group Symposium, The Missing Link: Hypermedia Usability Research & The Web, 1st May, 1996, Knowledge Media Institute, Open University, UK.  
[kmi.open.ac.uk/people/sbs/missing-link/Benyon.html](http://kmi.open.ac.uk/people/sbs/missing-link/Benyon.html) [accessed 7th December 2006]
- U63 Berkun, S. (2002). *"The list of reasons ease of use doesn't happen on engineering projects"*. Essay by consultant, formerly of Microsoft.  
[www.scottberkun.com/essays/essay22.htm](http://www.scottberkun.com/essays/essay22.htm) [accessed 5th November 2006]
- S26 Cabinet Office. (2000). *"Successful IT - Modernising Government in Action"*. [ogc.gov.uk/index.asp?docid=2632](http://ogc.gov.uk/index.asp?docid=2632) [accessed 1st August 2006]
- S15 Cantor, M. (2002). *"Software Leadership: A Guide to Successful Software Development"*. Addison-Wesley, NY. Appendix A.2, p153 & 162. Taken from [www.maxwideman.com/papers/linearity/waterfall.htm](http://www.maxwideman.com/papers/linearity/waterfall.htm) [accessed 31st July 2006]

- U132 Carmel, E., Whitaker, R., George, J. (1993). *"PD and Joint Application Design: a Transatlantic Comparison"*. Communications of the ACM, Volume 36, Issue 6 (June 1993). pp40-48.
- U89 Cohen, D., Lindvall, M., Costa, P. (2004). *"An Introduction to Agile Methods"*. Chapter in *"Advances in Computers"*, vol 62, ed. Zelkowitz, M. Elsevier.
- U48 Cody, N. (2004). *"Extreme Programming vs. Interaction Design"*. Blog. Not a formal source, but interesting thoughts.  
[www.primordia.com/blog/archives/2004/05/extreme\\_program.html#comments](http://www.primordia.com/blog/archives/2004/05/extreme_program.html#comments)  
 [accessed 30th September 2006]
- U107c Conallen, J. (2002). *"Building the User Interface: The Case for Continuous Development in an Iterative Project Environment"*. The Rational Edge, (IBM magazine) December 2002.  
[download.boulder.ibm.com/ibmdl/pub/software/dw/rationaledge/dec02/BuildingTheUserInterface\\_TheRationalEdge\\_Dec2002.pdf](http://download.boulder.ibm.com/ibmdl/pub/software/dw/rationaledge/dec02/BuildingTheUserInterface_TheRationalEdge_Dec2002.pdf) [accessed 27th November 2006]
- S68 Conboy, K., Fitzgerald, B. (2004). *"Toward a Conceptual Framework of Agile Methods: A Study of Agility in Different Disciplines"*. Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research. pp37-44.
- U38 Constantine, L., Lockwood, L. (2003). *"Usage-centered software engineering: An Agile Approach to Integrating Users, User Interfaces, and Usability into Software Engineering Practice"*. International Conference on Software Engineering, Proceedings of the 25th International Conference on Software Engineering, 2003.  
[portal.acm.org/citation.cfm?id=776931&dl=acm&coll=&CFID=15151515&CFTOKEN=6184618](http://portal.acm.org/citation.cfm?id=776931&dl=acm&coll=&CFID=15151515&CFTOKEN=6184618)
- U27 Cooper, A. (2003). *"About face 2.0 : the essentials of interaction design"*. Wiley, Chichester, 2003.
- U110 Cost Effective User Centred Design. (2002). The site describes simple user-centred methods recommended by the TRUMP project to improve the usability of end products and systems. Connected to Process for Usability.  
[www.usabilitynet.org/trump/index.htm](http://www.usabilitynet.org/trump/index.htm) [accessed on 30th November 2006]
- U96 Courage, C., Baxter, K. (2005). *"Understanding Your Users: A Practical Guide to User Requirements: Methods Tools and Techniques"*. The Morgan Kaufmann Series in Interactive Technologies.
- S35 Cramer, A., Alda, S. (2005). *"Software Development Process Models and their Impacts on Requirements Engineering"*. Bonn-Aachen International Center for Information Technology.  
[www.cs.uni-bonn.de/III/lehre/vorlesungen/SWT/RE05/slides/04\\_Software%20Development%20Process%20Models.pdf](http://www.cs.uni-bonn.de/III/lehre/vorlesungen/SWT/RE05/slides/04_Software%20Development%20Process%20Models.pdf) [accessed 21st September 2006]
- U75 Dabaaj, Y. (2001). *"An Evaluation Framework for Assessing the Capability of Human-Computer Interaction Methods In Support of the Development of Interactive Systems"*. Position paper in *"Usability Throughout the Entire Software Development Lifecycle - A Summary of the INTERACT 2001 Workshop"*. Official workshop for the International Federation for Information Processing (IFIP) working group 13.2 on *"Methodologies for User Centred Systems Design"*, at the INTERACT 2001 conference in Tokyo, Japan, July 9, 2001. Available from  
[www.it.uu.se/research/publications/reports/2001-026/2001-026.pdf](http://www.it.uu.se/research/publications/reports/2001-026/2001-026.pdf). See U10.  
 [accessed 9th November 2006]
- S21 Dale, C. (2005). *"The Underlying Problems with Prince2"*. eGov Monitor, 31st May 2005. [www.egovmonitor.com/node/1157](http://www.egovmonitor.com/node/1157) [accessed 29th July 2006]
- S22 Dale, C. (2006) *"Prince2 Weaknesses - the underlying problems with Prince2 (and related Project Management methodologies)"*. Business Transition Technologies - research paper. [www.btt-research.com/waterfall%20projects.htm](http://www.btt-research.com/waterfall%20projects.htm) [accessed 29th July 2006]
- S45 Defense Science Board Task Force On Defense Software - Report. (2000).  
[www.acq.osd.mil/dsb/reports/defensesoftware.pdf](http://www.acq.osd.mil/dsb/reports/defensesoftware.pdf) [accessed 20th October 2006]

- S48 DeMarco, T. (1979). *Structured Analysis and System Specification*. Prentice Hall, New York.
- U99 DeMarco, T., Boehm, B. (2002). *The Agile Methods Fray*. Computer Magazine (IEEE Computer Society), June 2002.
- U58 Dicks, R. (2002). *Mis-Usability: On the Uses and Misuses of Usability Testing*. ACM Special Interest Group for Design of Communication, Proceedings of the 20th annual International Conference on Computer Documentation, 2002.
- U57 Dlodlo, N., Bamford, C. (1996). *Separating Application Functionality from the User Interface in a Distributed Environment*. Proceedings of the 22nd EUROMICRO Conference, 1996. doi.ieeecomputersociety.org/10.1109/EURMIC.1996.546389
- U68 Dobos, H. (2002). *Separable User Interfaces and Interaction Controls*. Master's Thesis, University of Jyvaskyla,
- U115 Donahue, G. (2001). *Usability and the Bottom Line*. IEEE Software Magazine. January/February 2001 (Vol. 18, No. 1) pp31-37.
- U67 Evers, M. (1999). *A Case Study on Adaptability Problems of the Separation of User Interface and Application Semantics*, CTIT Technical Report TR99-14, Centre for Telematics and Information Technology, (1999).
- U53 Ferre, X. (2003). *Integration of Usability Techniques into the Software Development Process*. International Conference on Software Engineering (Bridging the gaps between software engineering and human-computer interaction), 2003.
- U76 Ferre, X. (2001). *Incorporating Usability into an Object Orientated Development Process*. Position paper in *Usability Throughout the Entire Software Development Lifecycle - A Summary of the INTERACT 2001 Workshop*. Official workshop for the International Federation for Information Processing (IFIP) working group 13.2 on "Methodologies for User Centred Systems Design", at the INTERACT 2001 conference in Tokyo, Japan, July 9, 2001. Available from [www.it.uu.se/research/publications/reports/2001-026/2001-026.pdf](http://www.it.uu.se/research/publications/reports/2001-026/2001-026.pdf). See U10. [accessed 9th November 2006]
- U41 Fitzpatrick, R. (2001). *Strategic Drivers of Software Quality: Beyond External and Internal Software Quality*. Proceedings of the Second Asia-Pacific Conference On Quality Software, 2001. (APAQS.01) [csdl.computer.org/dl/proceedings/apaqs/2001/1287/00/12870065.pdf](http://csdl.computer.org/dl/proceedings/apaqs/2001/1287/00/12870065.pdf) [accessed 26th September 2006]
- U62 Folmer, E., Bosch, J. (2003). *Usability Patterns in Software Architecture*. Proceedings of the 10th International Conference on Human-Computer Interaction Volume I. HCI2003, Crete, 2003. pp93-97.
- S24 Forsberg, K. Mooz, H. (1995). *Application of the 'Vee' to Incremental and Evolutionary Development*, NCOSE (National Council on Systems Engineering) — Fifth Annual International Symposium — St. Louis, July 23–26 1995
- U31 Fowler, S., Stanwick, V. (2004). *Web Application Design Handbook: Best Practices for Web-Based Software*. Morgan Kaufmann Publishers Inc.
- U82 Gentner, D., Grudin, J. (1990). *Why good engineers (sometimes) create bad interfaces*. Conference on Human Factors in Computing Systems Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people. Seattle, USA, 1990. pp277-282.
- U83 Gentner, D., Grudin, J. (1996). *Design Models for Computer Human Interfaces*. Computer, vol. 29, no. 6, pp. 28-35, June 1996.
- S28 Glass, R. (2003). *Facts and fallacies of software engineering*. Addison-Wesley, London/
- S20 Goodwin, C. (2001)/ *Development methodology in under 10 minutes*. Computer Weekly, 4th October 2001. [www.computerweekly.com/Article101748.htm](http://www.computerweekly.com/Article101748.htm) [accessed 29th July 2006]

- U78 Göransson, B. (2001). "A Usability Designer at Work". Position paper in "Usability Throughout the Entire Software Development Lifecycle - A Summary of the INTERACT 2001 Workshop". Official workshop for the International Federation for Information Processing (IFIP) working group 13.2 on "Methodologies for User Centred Systems Design", at the INTERACT 2001 conference in Tokyo, Japan, July 9, 2001. [www.it.uu.se/research/publications/reports/2001-026/2001-026.pdf](http://www.it.uu.se/research/publications/reports/2001-026/2001-026.pdf). (See U10). [accessed 9th November 2006]
- U136 Grønbaek, K., Kyng, M., Mogensen, P. (1993). "CSCW Challenges: Cooperative Design in Engineering Projects". Communications of the ACM, Volume 36, Issue 6 (June 1993). Special issue on graphical user interfaces: the next generation. pp 67-77.
- U86 Grudin, J. (1991). "Systematic Sources of Suboptimal Interface Design in Large Product Development Organizations". Human-Computer Interaction, 1991, Volume 6, pp. 147-196
- U88 Grudin, J. (2005). "Why CHI Fragmented". Conference on Human Factors in Computing Systems (CHI '05) extended abstracts on Human factors in computing systems. Portland, Oregon, USA, 2005. pp 1083-1084.
- S43 Guindon, R., Krasner, H., Curtis, W. (1987) "Breakdown and Processes During Early Activities of Software Design by Professionals". In: Olson, G. and Sheppard S. eds. "Empirical Studies of Programmers: Second Workshop". 1987. Norwood, NJ: Ablex Publishing Corporation. pp65-82.
- U93 Hartson, H., Hix, D. (1989). "Toward empirically derived methodologies and tools for human-computer interface development". International Journal of Man-Machine Studies. Vol 31, issue 4, 1989, pp 477-494.
- S81 Hazzan, O., Tomayko, J. (2004). "The Reflective Practitioner Perspective in Software Engineering". Conference on Human Factors in Computing Systems, CHI 2004. One Day Workshop - Designing for Reflective Practitioners. (Vienna, Austria, April 24-29, 2004)
- U100 Highsmith, J. (2000). "Retiring Lifecycle Dinosaurs". Software Testing & Quality magazine, July/August 2000.
- U101 Highsmith, J. (1997). "Messy, Exciting, and Anxiety Ridden: Adaptive Software Development". American Programmer, Volume 10, No. 1; January 1997.
- S69 Hughes, J. (1999). "Theory and Practice - the Practitioner's Dilemma?". Notes to accompany a talk to the Software Engineering and Information Systems Network for improved business processes, SEISN, 1999. [www.rdg.ac.uk/AcaDepts/si/sisweb8/seisn/archive.htm](http://www.rdg.ac.uk/AcaDepts/si/sisweb8/seisn/archive.htm) [accessed 5th December 2006]
- U112 Human-Centered Software Engineering Group (HCSE). (Undated). A multidisciplinary research group at Concordia University, Montreal. [hci.cs.concordia.ca/www/hcse/](http://hci.cs.concordia.ca/www/hcse/) [accessed 4th December 2006]
- U08 IBM Ease of Use website. (Undated). [www.ibm.com/easy](http://www.ibm.com/easy) [accessed 2nd May 2006]
- U135 Jacob, R. (2006). "What is the Next Generation of Human-Computer Interaction?". Workshop abstract. Conference on human factors in computing systems, CHI 2006. Montreal, Canada, April 2006.
- U61 Jokela, T. (2004). "Evaluating the user-centredness of development organisations: conclusions and implications from empirical usability capability maturity assessments". Interacting with Computers 16(6): pp1095-1132.
- U72 Jokela, T. (2005). "Certification of the User-Centredness of Development Organisation – A Way for Ensuring User Acceptance even before the Development of Software?". User-driven IT Design and Quality Assurance Conference, 2005.
- U107b Jones, W. (2002). "Distinguishing Between Application and Interface: An Alternative Approach to User Interface Development". The Rational Edge, (IBM magazine) December 2002.

- download.boulder.ibm.com/ibmdl/pub/software/dw/rationaledge/dec02/AlternativeApproachToUID\_TheRationalEdge\_Dec2002.pdf [accessed 27th November 2006]
- U30 Kaner, C. (2002). "XP, Iterative Development, and the Testing Community". StickyMinds.com Weekly Column, 21/10/02  
www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&ObjectId=5970 [accessed 13th August 2006]
- U137 Kyng, M. (1991). "Designing for cooperation: cooperating in design". Communications of the ACM, Volume 34, Issue 12 (December 1991) pp 65-73.
- U71 Koskela, J., Abrahamsson, P. (2004). "On-Site Customer in an XP Project: Empirical Results from a Case Study". European Software Process Improvement Conference (EuroSPI 2004), Trondheim, Norway, 10-12 November 2004, Dingsøy, T. (ed.), "Lecture Notes in Computer Science 3281", Springer, 2004
- U98 Krill, P. (2006). "Steve McConnell - Agile programming has fallen short". Infoworld Magazine, March 13th 2006.
- S59 Lattanze, A. (1997). "The Decentralized Computing Dilemma: How DoD Got into It and How We Hope to Deal with It". Crosstalk - the Journal of Defense Software Engineering, February 1997.  
www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1997/02/decentral.asp [accessed 24th November 2006]
- U40 Lauesen, S. (1998). "Usability Requirements in a Tender Process". Australasian Computer Human Interaction Conference, 1998.  
doi.ieeecomputersociety.org/10.1109/OZCHI.1998.732203
- S23 Lee, V. Schneider, H. Schell, R. (2004). "Mobile Applications: Architecture, Design, and Development". Prentice Hall.
- U169 Lethbridge, T. (2006). "Integrating HCI and Usability into Software Engineering". Presentation to CapCHI, the ACM SIGCHI, Ottawa Chapter, September 2006.  
www.site.uottawa.ca/~tcl/presentations [accessed 20th January 2007].
- U69 Lif, M., Göransson, B., Sandbäck, T. (2005). "Buying Usable - the User-Centred Procurement Process". User-driven IT Design and Quality Assurance Conference.
- U124 Löwgren, J. (1995). "Applying design methodology to software development". Proceedings of the conference on Designing interactive systems: processes, practices, methods and techniques. Ann Arbor, Michigan, United States. pp87-95.
- U24 Malotaux, N. (2006). "Optimising the Contribution of Testing to Project Success", Malotaux Consultancy. April 2006. www.malotaux.nl/nrm/pdf/EvoTesting.pdf [accessed 29th July 2006]
- U25 Malotaux, N. (2006). "Evolutionary Project Management Methods", Malotaux Consultancy. April 2006. www.malotaux.nl/nrm/pdf/MxEvo.pdf [accessed 29th July 2006]
- U138 Menlo Innovations. (Undated). Interesting papers about High-Tech Anthropology & Agile Teams, proprietary methods that use XP & RUP.  
www.menloinnovations.com/index.htm [accessed 14th December 2006]
- U18 Milewski, A. (2004). "Software Engineers and HCI Practitioners Learning to Work Together: A Preliminary Look at Expectations" 17th Conference on Software Engineering Education and Training (CSEET'04)  
http://doi.ieeecomputersociety.org/10.1109/CSEE.2004.1276509 [accessed 25th April 2006]
- U19 Milewski, A. (2003). "Software engineering Overlaps with Human-Computer Interaction: A Natural Evolution." Position Paper for the Workshop: Bridging the Gaps Between Software engineering and Human-Computer Interaction, International Conference on Software engineering, Portland, 2003  
http://www.se-hci.org/bridging/icse/accepted/10\_Milewski\_Monmouth.pdf [accessed 25th April 2006]

- S38 Misnevs, B., Daineko, D. (2001). *“Developing a New Testing Model”*. Programme Of The International Conference *“Reliability And Statistics In Transportation And Communication (Relstat’01)”* 15 October 2001. Riga, Latvia  
www.tsi.lv/Transport&Telecommunication/v31\_en/art04\_Misnevs.pdf [accessed 16th January 2007]
- U39 Moløkken-Østvold, K. Jørgensen, M. (2005). *“A Comparison of Software Project Overruns—Flexible versus Sequential Development Models”*. IEEE Transactions of Software Engineering, Vol. 31, No. 9, September 2005.  
csdl.computer.org/dl/trans/ts/2005/09/e0754.pdf
- U49 Nicholson, K. (2006). Scottish Executive. *“Usability in Government Websites”* Usability Professionals Association - Scottish branch presentation.  
www.scottishupa.org.uk/Scottish%20Executive%20SUPA%20presentation.pdf#search=%22%2Busability%20%2B%22scottish%20executive%22%22 [accessed 28th September 2006]
- S03 O'Regan, G. (2002). *“Practical Approach to Software Quality”*. Springer-Verlag New York.
- U163 Patton, J. (2002). *“Hitting the target: adding interaction design to agile software development”*. Conference on Object Oriented Programming Systems Languages and Applications archive. OOPSLA 2002 Practitioners Reports.  
doi.acm.org/10.1145/604251.604255 [accessed 13th January 2007]
- U107a Perrow, M. (2002). *“RUP and the User Experience: A Point / Counterpoint Exchange”*. The Rational Edge, (IBM magazine). December 2002.  
download.boulder.ibm.com/ibmdl/pub/software/dw/rationaledge/dec02/PointCounterP ointIntro\_TheRationalEdge\_Dec2002.pdf [accessed 27th November 2006]
- U50 Phillips, C., Kemp, E. (1996). *“Towards the Integration of Software Engineering and HCI Education - a Cross-Disciplinary Approach”*. 6th Australian Conference on Computer-Human Interaction (OZCHI '96), 1996.  
doi.ieeecomputersociety.org/10.1109/OZCHI.1996.560001
- U51 Phillips, C., Kemp, E. (1998). *“The Integration of HCI and Software Engineering”*. 1998 International Conference on Software Engineering: Education & Practice.  
doi.ieeecomputersociety.org/10.1109/SEEP.1998.707678
- U44 Preece, J., Rogers, Y., Sharp, H. (2002). *“Interaction Design - Using XP to Develop Context-Sensitive Adverts for the Web”*. Website accompanying book U04.
- U20 Pressman, R. (1997). *“Software engineering: A practitioner’s approach”*., Fifth Edition, McGraw Hill, New York.
- U144 Read, J., MacFarlane, S., Gregory, P. (2004). *“Requirements for the Design of a Handwriting Recognition Based Writing interface for Children”*. Proceeding of the 2004 conference on Interaction design and children: building a community, Maryland USA. 2004. pp. 81-87.
- U92 Robertson, J. (2002). *“Eureka! Why Analysts Should Invent Requirements”*. IEEE Software, Vol. 19, No. 4. July/August 2002. pp. 20-22
- U73 Rödiger, K. (2005). *“How to Teach User-Driven IT Design to Computer Science Students?”*. User-driven IT Design and Quality Assurance Conference, 2005
- S05 Rose L. (2006). Quality Engineering Manager, IBM. *“Myths and realities of iterative testing”* www-128.ibm.com/developerworks/rational/library/apr06/rose/index.html [accessed 2nd May 2006]
- U05 Rosson, M., Carroll, J.M. (2002). *“Usability Engineering: Scenario-Based Development of Human-Computer Interaction”*, Academic Press, London/
- S04 Royer, T. (1993). *“Software Testing Management – Life on the Critical Path”*. Prentice Hall.

- U52 Schneider, J., Vasa, R. (2006). *"Agile Practices in Software Development – Experiences from Student Projects"*. Australian Software Engineering Conference (ASWEC'06).
- S42 Schoen, D. (1983). *"The Reflective Practitioner: How Professionals Think in Action"*. Basic Books, New York.
- U128 Schuler, D., Namioka, A. (Eds). (1993). *"Participatory Design: Principles and Practices"*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- U23 Seffah, A. Andreevskaia, A. (2003). *"Empowering Software Engineers in Human-Centered Design"*. Proceedings of the 25th International Conference on Software Engineering 2003  
<http://portal.acm.org/citation.cfm?id=776909&coll=ACM&dl=ACM&CFID=74488562&CFTOKEN=63010766> [accessed 25th April 2006]
- U94 Software Reality. (Undated). A site providing sceptical coverage of Agile methods, by Matt Stephens (see S65). [www.softwarereality.com](http://www.softwarereality.com) [accessed 30th November 2006]
- U42 Spool, J., Porter, J. (2006). *"Agile Development Processes: an interview with Jeff Patton"*. User Interface Engineering (research & consultancy firm), September 2006. [www.uie.com/articles/patton\\_interview](http://www.uie.com/articles/patton_interview) [accessed 28th September 2006].
- U21 Sulaimain, S. (1996). *"Usability and The Software Production Life Cycle"*, ACM/SIGCHI CHI 96 Conference (Association for Computer Machinery Special Interest Group in Computer-Human Interaction)  
[sigchi.org/chi96/proceedings/doctoral/Sulaiman/ss\\_txt.htm](http://sigchi.org/chi96/proceedings/doctoral/Sulaiman/ss_txt.htm)
- U64 Stephanidis, C. (2001). *"User Interfaces for All: New perspectives into Human-Computer Interaction."* In C. Stephanidis (Ed.), *"User Interfaces for All - Concepts, Methods, and Tools"* (pp. 3-17). Lawrence Erlbaum Associates.
- U06 Stephanidis, C. (2003). *"Human-Computer Interaction : Theory and Practice, Part I, Volume 1"*, Lawrence Erlbaum Associates, 2003
- S70 Thomas, G. (2006). *"Aligning Development and Testing Lifecycles"*. Presentation to Software & Systems Quality Conference, London, 2006.  
[www.geocities.com/badgerscroft\\_com/presentations.htm](http://www.geocities.com/badgerscroft_com/presentations.htm) [accessed 7th December 2006]
- U71 Thoren, C. (2005). *"Approaches for inclusion of usability and accessibility in ICT procurements"*. User-driven IT Design and Quality Assurance Conference, 2005.
- U69 University of Chicago. (2002)/ *"Object Oriented Architecture: Patterns, Technologies, Implementations"*. Lecture on history of GUI techniques. 2002.  
[people.cs.uchicago.edu/~mark/51050/lectures/lecture.7/lecture.7.pdf](http://people.cs.uchicago.edu/~mark/51050/lectures/lecture.7/lecture.7.pdf) [accessed 6th November 2006]
- U149 van der Veer, G., van Vliet, H. (2003). *"A Plea for a Poor Man's HCI Component in Software Engineering and Computer Science Curricula; After all: The Human-Computer Interface is the System"*. Computer Science Education, Vol 13, no 3 (Special Issue on Human-Computer Interaction), 2003. pp 207-226.
- S36 van Veenendaal, E., Swinkels, R. (2002). *"Guidelines for Testing Maturity"*. Professional Tester, vol 3, issue 1, March 2002.
- U118 Wagner, H. (2004). *"Integrating usability into the software development ; Analysis of a software development process"*. University of Hamburg.
- S02 Watkins, J. (2001). *"Testing IT : An Off-the-Shelf Software Testing Handbook"* Cambridge University Press.
- U95 Wells, D. (2006). *"Extreme Programming: A gentle introduction"*. [www.extremeprogramming.org](http://www.extremeprogramming.org) [accessed 15th November 2006]
- S08 Yourdon, E. (1989). *"Modern Systems Analysis"*. Prentice Hall International/
- U150 Zhang, P., Carey, J., Te'eni, D., Tremaine, M. (2004). *"Integrating Human-Computer Interaction Development into SDLC: A Methodology"*. Proceedings of the Americas Conference on Information Systems, AMCIS 2004 Track on Human-Computer

Interaction Studies in MIS, New York, August 2004.  
[sigs.aisnet.org/sighci/amcis04/AMCIS\\_04\\_Zhang\\_etal\\_HCI\\_in\\_SDLC.pdf](http://sigs.aisnet.org/sighci/amcis04/AMCIS_04_Zhang_etal_HCI_in_SDLC.pdf) [accessed  
3rd January 2007]

# Appendices

## Appendix A - Questionnaire

<b>Organisation</b>		<b>Interviewee</b>		<b>Role</b>		<b>Date of Meeting</b>	
---------------------	--	--------------------	--	-------------	--	------------------------	--

Question	Yes	No	N/A	Explanation (if appropriate)
1a: Is your organisation accredited under the SEI CMMI scheme?				
1b: If so, at what level?				
2a: Does your organisation follow a project management method?				
2b: If so, what is it, or what are they?				
2b1 - Prince2				
2b2 - In-house method				
2b3 - Other (please explain)?				
3a: Does your organisation follow a development lifecycle model?				
3b: If so, what is it, or what are they?				
3b1 - Waterfall				
3b2 - Spiral				
3b3 - Star				
3b4 - RAD (please specify the particular model, e.g. DSDM)				
3b5 - Rational Unified Process (RUP)				
3b6 - Cleanroom				

Question	Yes	No	N/A	Explanation (if appropriate)
3b7 - Agile software development (please specify the particular model, e.g. Extreme Programming, SCRUM)				
3b8 - other (please specify)				
3b9 – Is the choice of lifecycle model influenced by whether usability is considered a significant factor in the development? E.g. is the waterfall model unlikely to be used if usability is a concern?				
4a: Does your organisation have an in-house IT function developing applications for your own staff?				
4b: Does your organisation have an in-house IT function developing e-commerce (b2c or b2b) applications <b>for your own organisation</b> ?				
4c: Does your organisation have an in-house IT function developing information websites <b>for your own organisation</b> which are targetted at the general public?				
4d: Does your organisation develop applications to be used by the staff of external clients?				
4e: Does your organisation develop e-commerce (b2c or b2b) applications for external clients?				
4f: Does your organisation develop information websites for external clients?				
5a: Do you believe that there is such a thing as a software testing model? If “no”, then go to 5d.				
5b: Does your organisation follow a model for software testing?				
5c: If so, how would you describe the model? Please tick all that apply.				
5c1 - V model				
5c2 - Requirements based				
5c3 - Statistics based				

Question	Yes	No	N/A	Explanation (if appropriate)
5c4 - BS7925				
5c5 - Six Sigma				
5c6 - JUnit				
5c6 - other (please specify)				
5d: If the answer to 5a was "no", is that because your testing practices are regarded as a integral part of the development model, and not as a testing model in their own right?				
6a: Do you have a testing strategy?				
6b: Does your testing maintain a distinction between functional and non-functional testing (ie, between <b>what</b> the system does and <b>how</b> it does it, between function & quality)?				
6c: Must test cases be traceable back to requirements?				
6d: Does the test strategy mandate that test cases can be written <b>only</b> if they can be traced back to a requirement?				
6e: Do you believe that non-functional requirements and testing are usually handled adequately?				
7a: Does your organisation carry out usability testing as part of software developments?				
7b: Is usability testing defined as a form of non-functional testing in the test strategy?				
7c: When testing is planned for an application, must "usability" (by whatever name) always be considered?				
7d: Do you believe that your organisation handles usability issues satisfactorily? Please give reasons for your answer?				
7e: Has the issue of usability been the subject of any special initiatives? Please explain.				

Question	Yes	No	N/A	Explanation (if appropriate)
8a: Does your organisation use HCI (Human Computer Interface) professionals, usability engineers or the equivalent? If the answer is "no" please go to Q9.				
8b: If so, are these internal staff?				
8c: Do they work on multi-disciplinary teams with the software developers?				
8d: Are HCI professionals or usability engineers involved at the specification stage?				
8e: Are HCI professionals or usability engineers involved at the testing stage?				
9a: Does your organisation use external software developers (commercial suppliers of bespoke software, not individual developers)?				
9b: If the answer to 9a was yes, do you believe that any of the answers previously given would be different for applications developed by external suppliers?				
9c: Do you believe that the use of external suppliers creates problems with testing that don't apply to in-house developments?				
9d: Do you believe that the use of external suppliers creates problems with usability that don't apply to in-house developments?				

# Appendix B - Nielsen's Ten Usability Principles

In *"Guerrilla HCI: Using Discount Usability Engineering to Penetrate the Intimidation Barrier"*, a chapter of Bias & Mayhew's book *"Cost Justifying Usability"* (U59, 1994, p242), Nielsen sets out the ten basic principles to be used for usability evaluation.

- ✘ Visibility of system status  
Users must get prompt, useful feedback about what is going on.
- ✘ Match between system and the real world  
The application should speak the users' language, rather than the developers, and should not use jargon, but restrict itself to a vocabulary and concepts familiar to the user. It should follow real-world conventions, presenting information appear in a natural and logical order.
- ✘ User control and freedom  
Users often choose the wrong functions, so they need clearly marked exits, allowing them to leave the unwanted state without an extended dialogue. The application should support undo and redo functions.
- ✘ Consistency and standards  
Users should never have to wonder whether different words, situations, or actions mean the same thing in different situations. Internal consistency within the applications is vital. Platform conventions should always be followed.
- ✘ Error prevention  
The application should try to prevent errors, rather than present error messages. It should either remove error-prone conditions or check for them and ask users for confirmation before they commit the action.
- ✘ Recognition rather than recall  
The application should minimise what the user has to remember by making objects, actions, and options visible. The user shouldn't have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
- ✘ Flexibility and efficiency of use  
Accelerators can speed up the interaction for the expert user so that the system can cater for both inexperienced and experienced users. These options shouldn't be visible to novices, or at least should not be presented in a way that would confuse them. Users should be allowed to tailor frequent actions.

✘ Aesthetic and minimalist design

Dialogues should contain no information which is irrelevant or rarely needed. Every extra unit of information in a dialogue distracts from the important and relevant information and diminishes its relative visibility.

✘ Help users recognise, diagnose, and recover from errors

Error messages should be expressed in plain language (with no codes), precisely indicate the problem, and constructively suggest a solution. SEs would argue that including codes makes error identification and resolution much simpler, but it must not be assumed that these give the user any direct benefit.

✘ Help and documentation

Ideally, the system should be capable of being used without documentation. However, help facilities and documentation are often necessary. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

## Appendix C - Defect Management

This is a typical, outline defect management process for an SE development. The process was written by the author, based on processes he wrote on commercial SE projects.

*Testers will assign defects to severity categories in the first instance. However, these will be reviewed by the Test Manager who, along with the Project Manager, will have the right to re-assign problems.*

*Each feature will have passed testing when the planned test cases have been performed with no failures. However, at the discretion of the Test Manager or Project Manager, test cases may be closed off with outstanding severity 3 or 4 problems if there is no viable solution or the cost outweighs the benefit.*

*All defects raised in the last day will be reviewed at the Daily Test Meeting, and the User Manager will have the right to change any of the Defect Severity Codes.*

<b>Defect Severity Codes</b>				
<b>Severity Codes</b>	<b>System Available</b>	<b>Severity of Function Degrade</b>	<b>Workaround Available</b>	<b>Business Impact</b>
<b>1</b>	No	Total	None	High
<b>2</b>	Yes	High - complete failure of a part of the application, or a significant part of the whole application is not working to specification.	None	Medium
<b>3</b>	Yes	Medium - minor part of the application is not working to specification.	Yes - but with difficulty	Low
<b>4</b>	Yes	Low - minor or cosmetic problem.	Yes	Low / None

*In addition, the Test Manager will give each defect a fix priority, dependent on the impact to the schedule.*

<b>Severity Fix Priority Levels</b>		
<b>Severity Fix Priority Level</b>	<b>Impact to Testing</b>	<b>Fix Turnround (unless TM requests diff't target)</b>
<b>A</b>	Severe impact to plan, cannot continue until a fix is supplied.	Immediate
<b>B</b>	Significant impact to plan, major areas of test cases awaiting fix to continue.	Next day, if notified by 13:00.
<b>C</b>	Medium / Low impact to plan, only single test case awaiting fix to continue.	Next cycle
<b>D</b>	Low / No impact to plan, test can continue without fix.	No default target

# Appendix D - ISO13407 & ISOTR18529 “Human-Centred Design Processes and their activities”

## ISO13407 - “Human-Centred Design Processes for Interactive Systems”, International Organisation for Standardisation 1999

### s5 Principles of human-centred design

Four characteristics of a human-centred approach.

*“a the active involvement of users and a clear understanding of user and task requirements,”*

*Helps developers understand the context of the task, and how users will use the product. This is obviously harder with web applications directed at the public, but there still must be some feedback from the public.”*

*“b an appropriate allocation of function between users and technology,”*

i.e., whether the technology or the user carries out specific tasks.

*“c the iteration of design solutions,”*

This is the usual worthwhile stuff. However, it does say;

*“Even in the ‘waterfall’ model, where there is a systematic top-down hierarchy of design decisions and the relationship between the stages generally precludes iteration between them, there can be extensive iteration within a stage.”*

*“d multi-disciplinary design.”*

Development teams must consist of a range of specialists.; the usual experts plus an HCI specialist.

### s6 Planning the human-centred design process

This stresses the importance of planning how, and where HC activities will be slotted in.

### s7 Human-centred design activities

Managers must decide which tasks will have to be changed to accommodate these activities and they must also schedule the project on the basis that HC activities will take place.

There are four human-centred activities.

*“a to understand and specify the context of use,*

*b to specify the user and organisational requirements,*

*c to produce design solutions,*

*d to evaluate designs against requirements.”*

Only the first applies solely to HCD. The others apply also to conventional developments. However, the content differs.

These activities are given in the order in which they appear in ISOTR18529, and are given the same reference numbers, i.e. HCD3 to HCD6.

HCD3 Specify the user and organisational requirements (s7.3)

NB - these are the user requirements as distinct from organisational requirements. In this HCD approach organisational requirements are what would normally be called "user requirements" in SE.

*"a required performance of the new system against operational and financial objectives,*

*b relevant statutory or legislative requirements, including health and safety,*

*c co-operation and communication between users and other relevant parties,*

*d the users' jobs (including the allocation of tasks, users' well-being and motivation),*

*e task performance,*

*f work design and organisation,*

*g management of change, including training and personnel to be involved,*

*h feasibility of operation and maintenance,*

*I the HC interface and workstation design."*

NB - the HC interface is just one (and the last listed) out of nine aspects of "user and organisational requirements". Effective user-centred design is much more than just the interface.

The requirements have to allocate the functions either to the users or the technology.

HCD4 Understand and specify the context of use (s7.2)

⇒ Understand the different types of users; experience, skills, education, preferences, habits etc.

⇒ Understand the tasks and the business goals.

⇒ Understand the environment; not just hardware and software, but also social, physical accommodation, the culture and the organisation.

Good, solid stuff. It is entirely consistent with Cooper's Interaction Design (U26, 2004).

HCD5 Produce design solutions (s7.4, iterate and evaluate till one gets it right)

The design solution builds on;

⇒ past experience/knowledge,

- ⇒ state of the art,
- ⇒ the context of use analysis.

Strangely, it doesn't mention the user and organisational requirements. Nor are these requirements mentioned in the 5 activities of design. This can only be an oversight.

There are 5 activities.

- a use your existing knowledge to build design proposals,
- b turn these into models, simulations, mock-ups, prototypes,
- c give these to the users and observe them trying to work with them - you can get experts to evaluate them if it's impractical to have genuine users trying to use them in a realistic way,
- d amend in the light of your observations, and iterate until your goals are met,
- e manage the iteration of the preceding 4 activities.

HCD6 Evaluate designs against requirements (s7.5, iterating back to HCD3 till one gets it right)

Evaluation should take place throughout the lifecycle.

- ⇒ to provide feedback,
- ⇒ to assess whether user & organisational objectives are being met,
- ⇒ to select the design option that best meets the requirements,
- ⇒ to elicit further and more detailed requirements from the users,
- ⇒ to monitor long term use of the product.

Projects require an evaluation plan (essentially a master test plan).

ISO13407 inadvertently draws attention to one of the conundrums of software development; how to move accurately and with confidence from a set of requirements to a design that meets those requirements.

A literal reading of the standard implies that the requirements are used to evaluate the design, and not to generate the design. That would be an impossibly wasteful approach, and is not intended by the creators of the standard. This is a pity because this whole area, i.e. the production of the initial architectural design is fraught with difficulty for SEs. In their commentary on ISO13407 (U09, 2006) Philip and Rourke correct the oversight without comment. This reinforces the point that ISO13407 should be regarded as outline guidance, and not a set of rigid instructions.

See Folmer & Bosch (U60, 2004), which discusses the difficulty in moving from usability requirements to an architectural design.

## ISOTR18529 - “Human-Centred Lifecycle Process Descriptions”, International Organisation for Standardisation 1999

The four activities HCD3 to HCD6 are the same as those in ISO13407. However, ISOTR18529 sets them more clearly and usefully in the organisational context. The additional activities are HCD2 for planning and managing the process, and HCD7 and HCD1, which recognise that once the new application is implemented there is a feedback into the IT strategy to refine the process. There is an iteration of HCD7 and HCD1 which is outside, and includes, the project level iteration of HCD3 to HCD6.

**Table 1 - Human-Centred Lifecycle Process Descriptions (Earthy, U109, 2000, p14)**

HCD.1	HCD.2	HCD.3	HCD.4	HCD.5	HCD.6	HCD.7
Ensure HCD content in systems strategy	Plan and manage the HCD process	Specify stakeholder and organisational requirements	Understand and specify the context of use	Produce design solutions	Evaluate designs against requirements	Introduce and operate the system
represent the end user	consult stakeholders	clarify system goals	identify user's tasks	allocate functions	specify context of evaluation	manage change
collect market intelligence	plan user involvement	define stakeholders	identify user attributes	produce task model	evaluate for requirements	determine impact
define and plan system strategy	select human-centred methods	assess H&S risk	identify organisational environment	explore system design	evaluate to improve design	customisation and local design
collect market feedback	ensure a human-centred approach	define system generate requirements	identify technical environment	develop design solutions	evaluate against system requirements	deliver user training
analyse user trends	plan HCD activities & manage HC activities	set quality in use objectives	identify physical environment	specify system and use	evaluate against required practice	support users
	champion HC approach			develop prototypes	evaluate in use	conformance to ergonomic legislation
	support HCD			develop user training		
				develop user support		

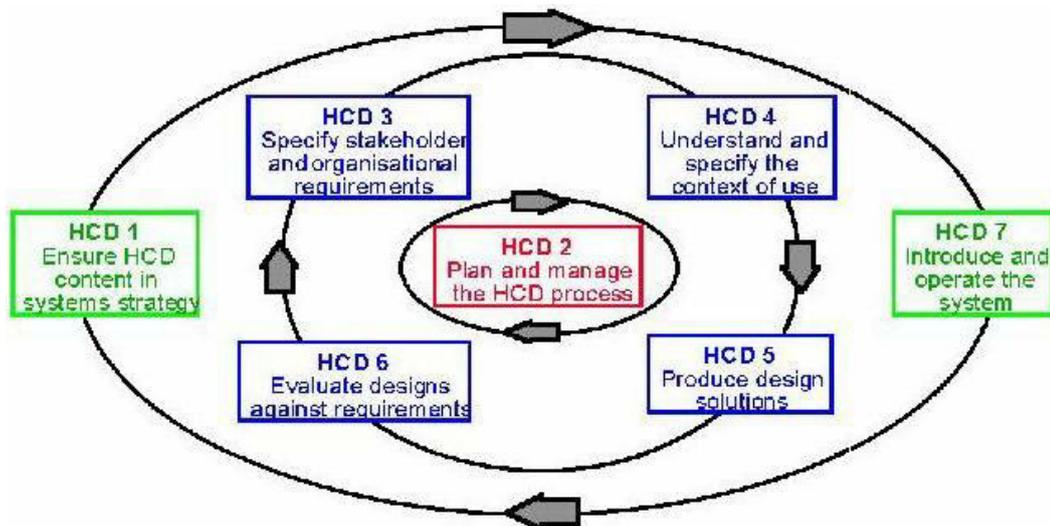


Figure 14 - Usability Maturity Model ISO TR 18529 Human-centred Design Lifecycle (U108, Process for Usability website, 2003)

## Appendix E - Usability Testing

Preece describes usability testing as comprising the following types and techniques, (Preece, U04, 2002, p347).

Table 2 - Types of Usability Testing				
Techniques	Quick & Dirty	Standard Usability Testing	Field Studies	Predictive
Observing users	Watching users in their natural environments.	Video & keystroke logging, which can be analysed to identify errors or identify common routes through the application.	More formal observation of users in their normal environment, i.e. ethnographic studies, where observers immerse themselves in the environment they are studying.	N/A
Asking users	Discussions with users and potential users individually or in groups.	User interviews and satisfaction questionnaires	Discussing observations with users. Interviewing users.	N/A
Asking experts	Providing critiques of the usability of a prototype	N/A	N/A	Early heuristic evaluation to predict the usability of an interface and shape the design.
User testing	N/A	Testing typical users on typical tasks.	N/A	N/A
Modelling users' task performance	N/A	N/A	N/A	Using models to predict usability of an interface, or compare performance times between versions.

For the purposes of this dissertation, usability testing is assumed to consist of standard usability testing and field studies. "Quick and dirty" evaluations are essentially an unstructured and poorly organised form of Discount Usability Engineering (DUE). Predictive Evaluation also overlaps to a large extent with DUE. The difference is that Predictive Evaluation does not include the DUE technique of "simplified thinking aloud", and the scenarios (or prototypes) of DUE are cheaper and less complex than is implied by the models of Predictive Evaluation.