VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

Department of Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 471 5328
Fax: +64 4 495 5232
Internet: Tech.Reports@comp.vuw.ac.nz

Methodologies for the Creation of Interactive Software

Judy Brown

Technical Report CS-TR-96/1

May, 1996

Abstract

This paper explores the methodologies for the development of interactive systems proposed by software engineers and human-computer interaction specialists. These two disciplines have proposed methodologies emphasizing different aspects of the design process.

The purpose of this paper is to present some of these methodologies and to identify their strengths and weaknesses. Methodologies are examined for their ability to facilitate working relationships between members of an integrated design team consisting of both SE and HCI specialists. Through a review of common SE and HCI methodologies, inadequacies of the education system in educating SE and HCI specialists about the roles of other specialists are identified. Differing value systems and work practices in the HCI and SE communities that block the creation of integrated methodologies are explored. Finally, tensions created by issues of power and control of the design process as conveyed through methodological descriptions are examined.

This paper calls for better methodologies that highlight and value the important contributions of both software engineering and HCI designers.

Author Information

Judy Brown is a lecturer in the Department of Computer Science at Victoria University Wellington. Her main areas of interest is : human-computer interaction.

1 Introduction and Definitions

In this paper methodologies proposed by software engineers and human-computer interaction specialists are compared. Since in many projects the individuals from these two disciplines must work together to produce a product, it was expected that methodologies should reveal the relevant points of contact between the disciplines. Although some methodologies in this review attempted to combine the methods from both disciplines into a coherent framework for product development, most did not. This survey of methodologies also revealed major differences in the type of methods that were used in each discipline. Further it was observed that each of the disciplines placed a different value on the development of methodologies per se.

Throughout this paper, the word **methodology** is used to refer to a process required for the development of a software system. Methodologies vary in breadth. They can be very broad, identifying the major steps involved from the conception of a software system to the final use of that system, or they can be more narrow and focus on a portion of this process. Methodologies also vary greatly in their depth. The amount of detail they provide when describing the design process varies considerably.

Each **step** in a methodology identifies a major task to be performed by the designer. Some methodologies advocate a specific **approach** to implementing a step such as : a data oriented approach, a functional approach or a object-oriented approach. Approaches can be clarified by methods. A method is a description of a technique for implementing a step. For instance, Booch describes a method for object-oriented analysis in [2].

Note that in this paper methods are contained within methodologies. A **method** is a technique for accomplishing a step in a methodology. A **methodology** is an integrated collection of methods. Methods can also stand on their own and not be part of a methodology.

Methods can at times be realized by the implementation of a software tool. Some methodologies advocate a particular approach and also provide tools to implement the approach. Sometimes these tools will be consolidated into an integrated environment and delivered as a complete system. In [3] Booth defines a methodology to be "a process that is ... supported by manuals, workshops, automated tools..." Booth points out that methods need not be computerized tools. They can be descriptions for new work practices. Many informal HCI methods can be described this way. Heuristic evaluation, the expert evaluation of an interface by an expert in HCI, is an example. Think aloud sessions, where users describe what they are doing while they are doing it, providing the designer with useful feedback, is another example. The spectrum for what can be classified as a method as wide. In this paper both informal and formal methods are considered.

Methodologies can be described in varying ways. Most methodologies are best described using diagrams showing transitions between stages. This sort of representation implies that one stage will derive input from a previous stage and produce some output useful for the next

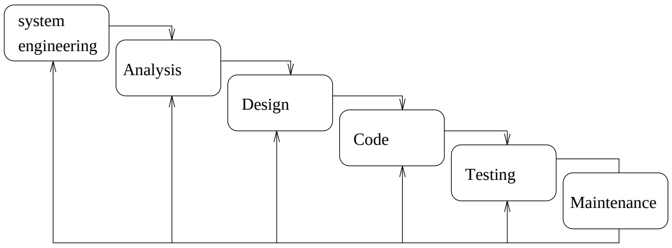

Figure 1: The Classic Life Cycle

step. Methodologies can also be depicted textually as a list of steps.

2 Review of Common SE Methodologies for Building Software Applications

Software engineering methodologies are general prescriptions for developing computer applications. In this part of the paper I review common software engineering methodologies a typical computer scientists might encounter in tertiary education courses. While some texts advocate a particular methodology others present a variety of approaches. Some approaches ignore the role of the HCI specialists, others slot HCI techniques into what is essentially a software engineering methodology.

- The Classic Life Cycle or Waterfall Methodology: The waterfall methodology (See Figure 1) identifies the major steps a software engineer takes to produce and support an application. This methodology is not a strictly linear process. Designers can return to previous steps and make modifications. Ideally however, the designer should not have to return to a previous step. Backtracking is an implicit admission that something was inadvertently missed during the first pass through that step, ie. that an omission or error was made. The major players in this process are : the systems analyst, software engineers, testers and the maintenance team. This methodology is presented in Pressman [15] and many other software engineering textbooks.

  In the original paper on the waterfall methodology [16] Royce more clearly describes the impact changes can have on preceding and succeeding steps. He also advocates the design phase includes the development of an "early simulation of the functional product". In his paper there is no mention of the special attention required for the development of the interface, but given the date of the paper, this is not surprising.

- Prototyping : With this methodology a sequence of prototypes is built until "a good understanding of the software requirements" is developed. See Figure 2. Presented in Pressman [15] this methodology is advocated when "the form that human-machine

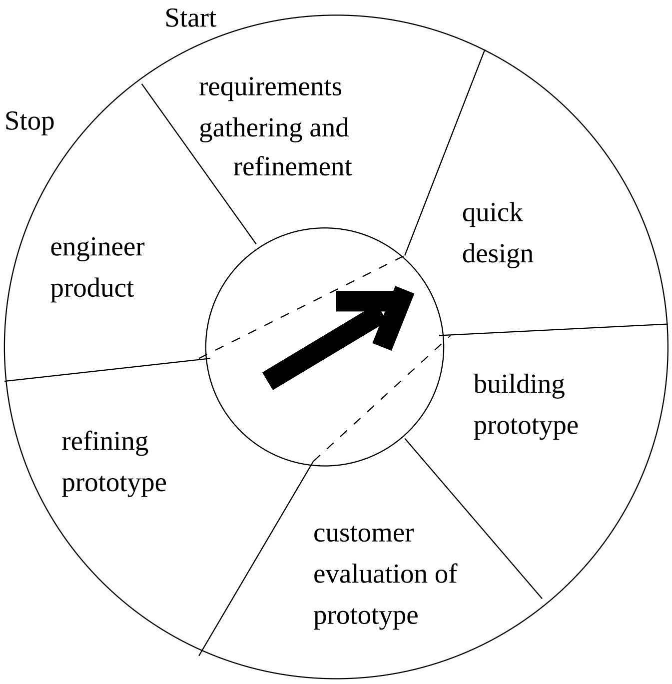

Figure 2: Prototyping

interaction should take is not clear". It recognizes that iterations of design steps are a normal part of the process. Via a prototyping approach, software engineers can develop a clearer idea of the product they are proposing to develop.

- The Spiral Model : See Figure 3. This methodology also presented in Pressman [15] highlights the role of the software engineer, the customer and managers. It presents the software engineering process in a business context where managers make decisions about the feasibility of a project, the resources allocated to it and the risks associated with developing it. At each cycle of the spiral, a fresh decision is made as to the purpose and value in doing another cycle. Customers are a necessary part of the process. They evaluate a series of prototypes developed by designers. Innately iterative, the spiral model encompasses both the rapid prototyping and waterfall methods while extending them. Steps in the waterfall method are subsumed by the 'engineering' step in this model.

  In the paper where the spiral model is proposed [1] Boehm is careful to make the design of the user interface a part of the spiral model. "If ... the user interface risks strongly dominate program development the next step may be an evolutionary development step." This leads to the development of a prototype that is a little closer to

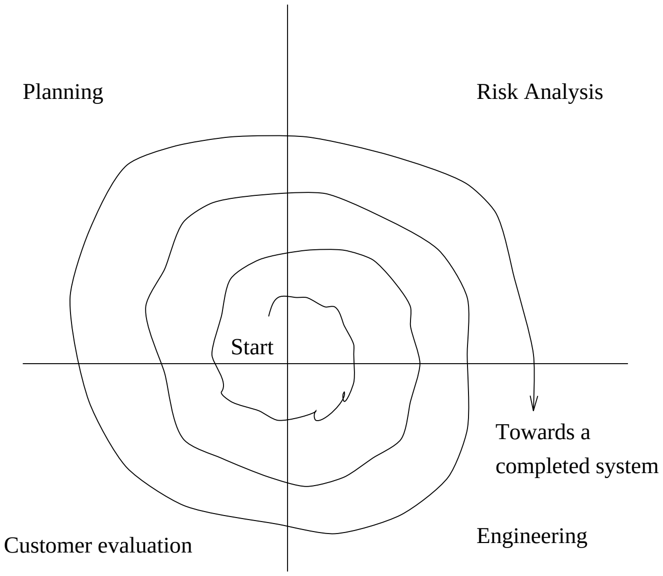

Figure 3: Spiral Model

an “operational software product”. He therefore recognizes that special attention be paid to the user interface without clearly specifying how this is done or without direct recognition of the role of the human-computer interaction specialist. His notion of the methodology required for user interface development is also somewhat different than prototyping as understood in the HCI community where the goal of the prototyping process is not necessarily to produce a piece of operational software.

- User Interface Design from a Software Engineer's Viewpoint : In [15] Pressman presents interface development as a totally separate activity from development of the functional part of a program. An interface is designed, prototypes are built and repeatedly assessed until the designer is satisfied that the interface is suitable for the user (See Figure 4). Pressman presents human-computer interface design as “one element of a larger topic that we have learned to call software design”. He admits that design methods for HCI are not widely used and gives very little detail as to the methods to apply for each step in the methodology he presents. User interface designers are seen as contributors to the requirements step of a larger software engineering development methodology.

- A Development Cycle with Emphasis on Testing : Pfleeger presents a methodology that emphasizes the testing steps required in software design (See Figure 5). Although not shown in this diagram, all of the steps are interconnected. The usual route through this process follows the circular path shown. The highly connected nature of this graph is necessary as “There is always the possibility that we will return to a previous step

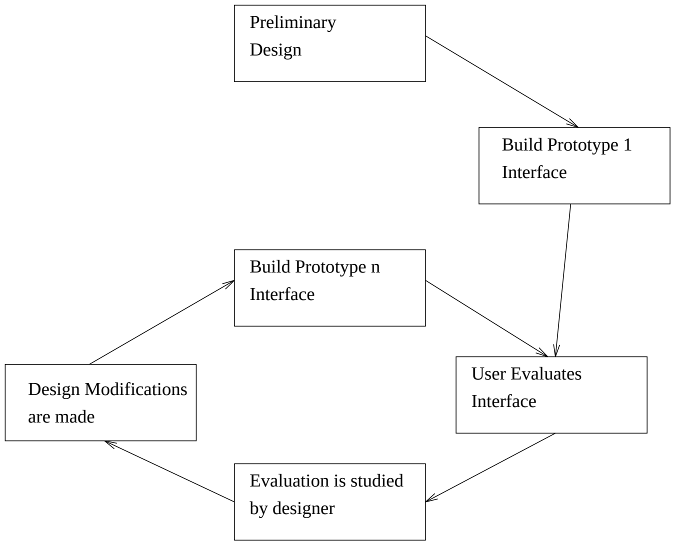

Figure 4: User Interface Design from a Software Engineer's Viewpoint

to revise or redefine something based on new information." Pfleeger's methodology is unique in that it identifies three different testing steps. No steps address the development of the user interface specifically. The team members she identifies in this methodology are : analyst, designer, programmer, test team and trainer.

- Engineering development life-cycle with added HCI awareness : Downton's book for systems designers with technical backgrounds proposes two methodologies that incorporate HCI techniques based on the composition of the team members in the design project [8]. The first methodology enhances a typical waterfall developmental methodology with HCI awareness (See figure 6). Here software engineers use simple cognitive or perceptual models, dialogue guidelines and informal evaluation techniques to replace an ad hoc or intuitive method for designing user interfaces.

- Engineering development life-cycle with added HCI practitioners and specialists : In Downton's second model (See Figure 7) the methodology of the software engineer is related to the methods of the HCI specialist. Task analysis, user modeling, formal interface specifications, dialogue design tools, formal evaluation techniques and standards for documents are used to produce useful interactive software. Downton proposes the methodology be adopted when HCI practitioners are part of the project design team. This view usefully slots many of the techniques of HCI into the software development methodology and depicts the HCI specialists and software engineering specialists as participants throughout the design process. In this methodology, the rapid prototyping technique is central to the development of the product's interface.

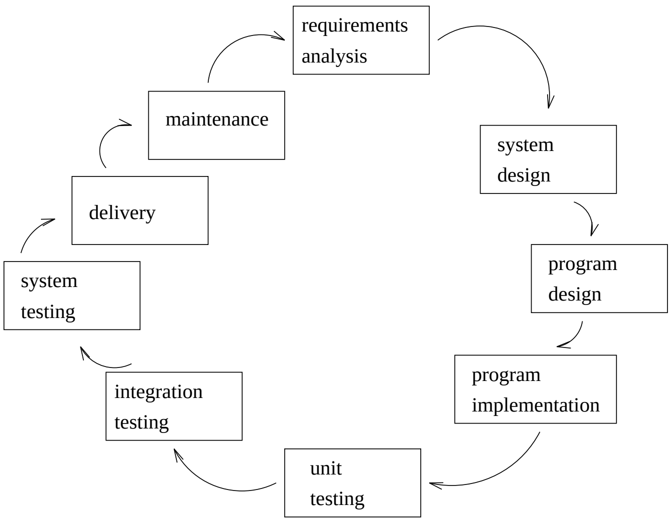

Figure 5: A Development Cycle with Emphasis on Testing

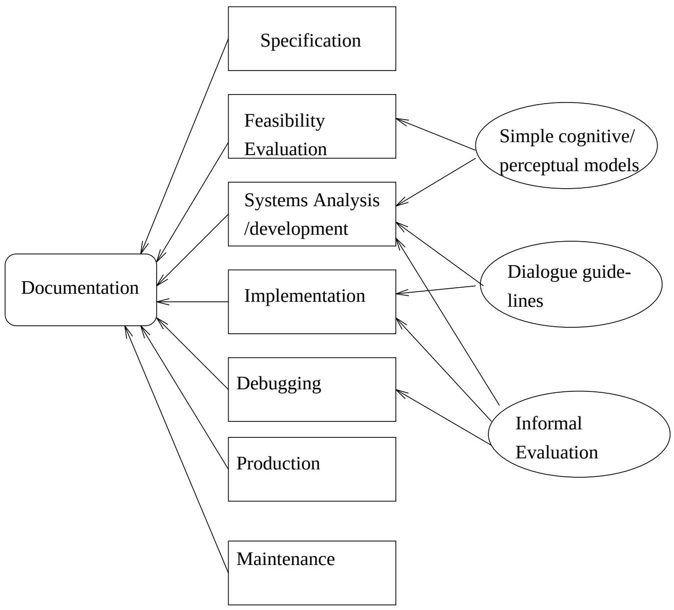

Figure 6: Engineering development life-cycle with added HCI awareness

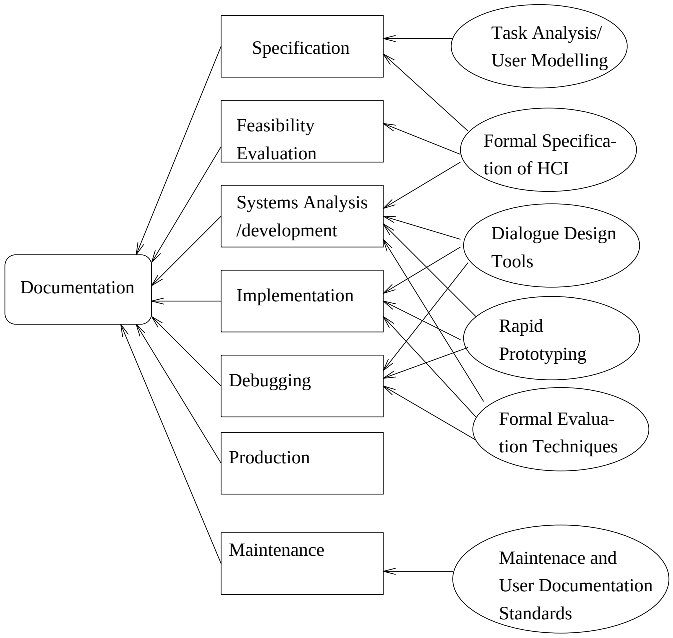

Figure 7: Engineering development life-cycle with added HCI practitioners and specialists

It is clear that there are vastly different opinions about the role of human-computer interaction specialists in software engineering texts. In [14] Pfleeger states that there is no special process for user interface design. For Pfleeger, interface requirements are indistinguishable from other requirements and are specified in the same document. This is not an adequate process for producing an interactive system.

In [15], Pressman presents many methodologies and gives more consideration to the development of the interface. Software engineers are not taught to rigidly adhere to any particular methodology. Pressman states in his introduction that "there is no need to be dogmatic about the choice of paradigms for software engineering." and suggests mixing paradigms at different points in the software engineering process depending on need. "A paradigm for software engineering is chosen based on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required." Presumably when an interface is a major portion of the project under development a software engineer would choose a methodology that gave the design of the interface due consideration such as the methodology in Figure 4. No guidance is given as to how the SE and HCI processes interrelate however.

Downton in [8] addresses the issue of user interface design integrated with the software engineering cycle most thoroughly. In his methodologies, methods from HCI to develop the user interface feed into the steps of the classic software engineering waterfall methodology.

3 Review of HCI methodologies for building User Interfaces

There is agreement within the HCI community that an iterative design technique is required for the development of user interfaces. The waterfall approach as in Figure 1 will not work for the development of user interfaces since a user interface cannot be specified without repeated testing with users. An experimental approach is necessary because there is not a sufficiently firm theory of human cognition and behavior from which a theoretically based interface design could be constructed.

In general the HCI community prescribes developing a prototype of an interface that is evaluated and then rebuilt and reassessed iteratively until the final interface has been designed. The user is seen as being integral to this process. Beyond this basic agreement to an approach, many issues are highly debated such as : the methods that are most effective or economical, the point at which these methods should be applied or the value of prescribing a methodology at all.

In this part of the paper I look at three major design approaches within the HCI community : user centered design, participatory design, and cognitive modeling. User centered design produces products **for** users, relying heavily on user feedback to produce a design. The participatory design produces products **with** users. Cognitive modeling emphasizes an understanding **of** users to produce a product. Of interest here is the emphasis on methodology within each of these approaches.

3.1 User Centered

3.1.1 USE Methodology

Wasserman et al in [17] and [9] advocates the use of a rapid prototyping methodology. His methodology is shown in Figure 8. Determining the user's characteristics is an important step in this methodology. Once the user's characteristics are well known, design begins. The main USE method for the specification of user interfaces is the development of state transition diagrams to describe the user's interaction with the computer. From an architectural point of view, the USE methodology makes a distinction between developing the user interface and developing the functional aspects of an application. This encourages developing rapid prototypes of the interface that can be iteratively assessed until a suitable interface has been designed. The step 'Build Functional Prototype' encompasses the steps from the software engineering waterfall model. In this methodology the user interface development process envelops the traditional software engineering process.

Although somewhat dated now, the USE methodology bears mentioning because of its maturity and breadth of coverage. In many ways it is typical of many user centered approaches today. It brings the user (rather than the customer) to the forefront as an important player in the design process. The user is the final user of the product. The customer is the purchaser of the product and may or may not be the user.

The user's characteristics and the tasks the user must perform are given due consideration. In many user-centered methodologies the user is also involved in the evaluation of the prototypes. It is not clear whether this is the case with the USE methodology. To specify a design in the USE methodology, state transition diagrams are used to describe the behavioral aspects of the program. Many other approaches have been proposed such as BNF grammars, event-drivers and by-example techniques. Architecturally, the USE methodology advocates a split between the user interface and the functional portion of the software. Other approaches (particularly dialogue driven approaches) advocate more software divisions than this. The USE methodology is an academic methodology which has not been used in industry because of the difficulty designers have with using the state transition diagrams and of its limitations in modeling asynchronous events common to bitmap graphical user interfaces which are currently preferred.

3.1.2 Usability Engineering

In [13] Nielson outlines 11 steps to a practical and economical process for user interface design. These are shown in Figure 9. This methodology was developed in industry and is fairly well known. Nielson's methodology is practical above all else. There is a major emphasis on

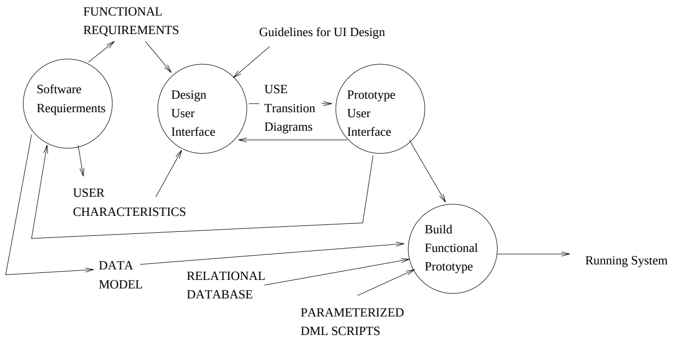

Figure 8: USE Methodology

knowing the user and the tasks the user must perform as well as having users test prototypes of a product under development.

Many of the methods in usability engineering are derived from other disciplines such as Psychology or Marketing. They tend to be a combination of qualitative and quantitative techniques. In general they are much less formal than the methods of software engineers. Specification of the user's characteristics is an example of a step which combines qualitative and quantitative approaches. Techniques such as surveys and questionnaires are often used to pinpoint some of the user's characteristics. Some of these characteristics can be quantified (such as the total number of colors a normal person can distinguish) whereas others are qualitative (such as a judgment on the learning style of a user). The designer summarizes this information in a report which defines the characteristics of the user and aids in the development of requirements, specifications and testing. Testing methods are usually more quantitative, frequently done in a lab situation, and the methods used are borrowed from psychology lab tests on humans.

The user and the interface designer are key players in this methodology. Nielson also sets out roles for managers and HCI testers. The relationship of the interface designer to the software engineer is not clarified in Nielson's methodology. This is a methodology clearly focused on the development of the interface of a product and probably designed with human-computer interaction specialists in mind.

A nice feature of usability engineering is that usability goals, which are set early on in the process, are used to define when the iterative design process is complete. These goals are generally quantifiable and measurable. One important outcome of the empirical testing phase would be to assess a product's progress towards the usability goals.

1. Know the user
   (a) Individual user characteristics
   (b) The user's current and desired tasks
   (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

Figure 9: Usability Engineering

3.2 Cognitive Modeling

3.2.1 Methodologies not Emphasized

The emphasis of cognitive modeling is to understand and model an activity as it is understood by the user. Cognitive modeling experts assert that if you can get this model right then you can use it to create a design which will be intuitive to the user. This approach is not focused on methods and does not advocate any particular methodology. Cognitive modeling advocates are interested in **why** users behave as they do or **why** one design is better than another.

Cognitive Scientists interested in modeling user behavior develop methods such as GOMS (Goals, Operators, Methods and Selection Rules). See [4] for a detailed description of GOMS. The designer lists the goals of the user, the primitive operators possible in a system (such as mouse clicks), the methods (ie sequences of primitive operators) required to achieve a goal and finally selection rules to decide between methods when more than one method can achieve the same goal. The GOMS based model can be analyzed and improved once developed. Bonnie John of Carnegie Mellon and Grey of George Mason University, are advocates of the GOMS approach and have used it to analyze real systems. This method is not yet accepted in industry.

3.2.2 Anti-Methodology

In [6] Carroll critiques methodologies. "We now see" he asserts, "how problem stages overlap, that there is no simple solution, that other non-design factors influence design and that the design team is multi-disciplinary." He states that a major shift has occurred from developing methodology to understanding designing rationale. Carroll is interested in why one design is better than another. Modern methods, such as Carroll's scenario-based methodology with psychological theory embedded to justify design, are methods by which design choices can be articulated. Other tools also described in [6] such as QOC (Questions, Options and Criteria) , DRL (Decision Representation Language) and IBIS (Issue-based Information System) also address the issue of design rationale over design process.

3.3 Participatory Design

3.3.1 No Particular Methodology

Participatory Design [PD] methodology advocates no particular methodology at all. In general, PD advocates do not see methodology as central to producing a good product. The more important issue is communication between users and designers. PD advocates work to describe the types of communication desired and methods which can help users and designers move closer to a better design. The focus in PD is much more on the quality of the communication which occurs between designers and users when a design is being worked on rather than on what particular steps to take when designing.

The role of the user as a peer in the design process is unique to participatory design. In [7] Clement describes the goal of participatory design to be to improve systems and empower workers. The method by which this is done involves the users as central actors in system development activities. In a review of major PD research projects, Clements reports that "several [PD] researchers noted that it was not the particular methods and techniques that were decisive [in system development], but a strong political focus on participation, communication and learning."

In PD design, methodology is not emphasized. However, PD methods do exist. Methods are seen more as a resource for designers to use as they deem appropriate and are not gathered into a coherent framework. They focus on the process by which users and designers are brought together to work on a design. PD methods which bring users and designers together span the entire development life cycle. Examples of these methods are : envisioning future solutions, contextual inquiries, games and prototyping. Although Muller [11] claims that PD techniques are applicable through the entire software development cycle, Carmel [5] claims these techniques are best for requirements definition phase noting that PD techniques appear to enhance creativity.

3.3.2 Emphasis on Communication

Some participatory designers are anti-methodology. Carmel, notes that "PD [advocates] criticize software design methodologies' ... claims of objectivity as well as the method of analysis through decomposition" [5] . Kensing in [10] believes that techniques [or methodologies] are not the answer. He claims that a more important focus is to develop models which enhance communication between users and developers. He classifies PD and software engineering techniques into 6 different communication enhancing categories. See Figure 10. Each box in this diagram represents a type of communication which needs to occur in order for a successful system to be developed. Knowledge is either abstract or concrete. Topics about which knowledge is required include : the present work situation, the new system and available technological options. Abstract knowledge is primarily the domain of the designer and concrete knowledge is generally the domain of the user. All of this knowledge is equally valued. Techniques which reveal knowledge in all 6 categories are required in design. Kensing notes that software engineering modeling techniques facilitate communication in only 2 of 6 required areas (categories 2 and 5) . Prototyping, a more promising technique, covers 3 of Kensing's 6 required categories of communication between user and designer (categories 3, 5 and 6 in the table).

PD design approach and methods have been used in many projects but are not in widespread use in industry.

	User's present work	New system	Technological Options
Abstract Knowledge	Relevant structures on user's present work (2)	Visions and design proposals (5)	Overview of Technological Options (4)
Concrete Experience	Concrete Experience with user's present work (1)	Concrete Experience with the new System (6)	Concrete Experience with Technological Options (3)

Figure 10: Kensing's Communication Enhancing Strategies

4 Conclusions

Projects which require a user interface are usually more difficult to build. In describing management tools for assessing project difficulty Pfleeger in [14] and Pressman in [15] rate user interface design projects as either a moderate or high level of difficulty by their nature. Myers describes why user interface systems are difficult in [12]. The development of user interfaces stresses the software engineering methodology and requires extra resources and expertise. In order for design teams to meet the challenge of developing software products with substantial user interfaces, an increased awareness of a development process that integrates user interface development and functional development is required.

This review shows that there is a basic fundamental difference between the approaches taken by software engineers and human-computer interaction specialists. Human-computer interface specialists are user-centered and software engineers are system-centered. This distinction is brought out by HCI experts who claim that software engineers do not put the user at the center of system development. HCI specialists have developed many techniques which attempt to do just this.

The approach of software engineers as reflected in their methodologies has many merits. Software engineering methodologies are good at modeling certain aspects of the problem domain. Formal methods have been developed to represent data, architectural, and procedural aspects of a software system. Software engineering approaches deal with managerial and financial issues well. Software engineering methodologies are useful for specifying and building the functional aspects of a software system. Other aspects which are useful for user interface design however, are missing.

Human-computer interface specialists show a better understanding of the user. They emphasize developing a deep understanding of user characteristics and a clear awareness of the tasks a user must perform. HCI specialists test design ideas on real users and use formal evaluation techniques to replace intuition in guiding design. This constant reality check improves the final product.

Some of the techniques which have been developed for interface design extend the type of modeling which can be done prior to the development of a software system and can contribute to producing a better requirements document and a better product.

- Profiles of user characteristics are useful additions to the requirements document. Descriptions, or models, of the user assist at guiding the development of both the interface and the functional portions of the software.
- Task descriptions can also be a useful part of a requirements document. Task descriptions are behavioral models. Behavioral models describe what actions the user must perform to get at a capability of the system.
- Prototypes, provide a visual model of the final system. Visual modeling represents how the system will appear to the user.

The contributions which HCI specialists can make to the development of a product are not well known at present, the fault lying at least in part with software engineering textbooks which ignore the contributions of the HCI community to the design process. Since HCI methodologies are not presented to software engineers it is easy to see how a software engineer may be unaware of the role of the HCI specialists and be unable to integrate his or her work practices with the HCI specialist. Texts, such as Downton's [8], although specifically designed for software engineers, are used in HCI courses offered by Computer Science departments and are not used in standard software engineering courses. HCI courses, when offered are not necessarily presented as part of the mainstream curriculum within Computer Science Departments.

Equally inadequate are HCI methodologies which do not clarify the role of the SE process relative to the HCI process. Neilson's usability engineering methodology is typical of HCI methodologies which leave this relationship ambiguous.

There are difficulties in attempting to combine methodologies however. Software engineers and human-computer interaction specialists do not agree on the value of methodologies per se. In line with a systems viewpoint, software engineers, who tend to be analytical by nature, have a preference for specifying a work strategy or methodology. HCI specialists, perhaps because of a need for flexibility in their work practice or because of a lack of maturity in their discipline, resist defining a process. No doubt this very different approach to doing work can cause difficulties when the two professional groups interact.

Another difficulty in combining methodologies of software engineers and HCI communities is the differing approach to methods employed by the communities. Software engineers tend to prefer formal methods and devalue informal methods. Formal methods are part of the HCI process but many informal methods are used regularly and valued as well.

Methodologies can be contentious in that they bring up power issues. Methodologies define who is in control of a process or who is central to a process. For instance, the methodologies proposed by human-computer interaction specialists usually leave the contribution of software engineers unspecified. This could imply a number of possible relationships between interface designers and software engineers. 1) Perhaps the interface is developed prior to (or after) the functional portion of the program. 2) Perhaps the functional and interface development are parallel activities. 3) Perhaps the HCI methodologies envelop the software engineering methodology. In contrast, the methodologies proposed by software engineers depict software engineers as central and HCI methods as contributing to the process but not changing it in any fundamental way.

There are many potential members in a team development of a software product : managers, marketing personnel, users, analysts, software designers, HCI specialists, testers, documentation personnel and style designers. Methodologies which acknowledge the contributions of all of these roles would assist at fostering good team relationships. Good methodologies can exacerbate or facilitate professional relationships. Good methodologies could clarify the roles of various professionals, identify when diverse team members are required to come together, and provide a means by which each group can contribute positively to a project. Good methodologies take all potential roles in the design process into account even though a full suite of roles may not be present for any particular project or some team members may be playing multiple roles. It is clear from reviewing the methodologies that the working relationship between the professional groups is not well defined. Good methodologies should describe the interactions with other working professionals. This is useful for developing a big picture of the work process and for fostering understanding between the members of the multi-disciplinary design team.

Methodologies are an idealized articulation of a process. In practice they are seldom perfectly realized. This does not detract from their value however. Methodologies convey the importance of various stages of a process to a community. They can be an aid to structuring work and can help to assess progress towards a final goal of a design team. They can identify key tasks in the development of a software product and define the boundaries of a task to be accomplished by a portion of a design team. They can help a team work together by clarifying the tasks of members of a team and by giving an overall picture of a process. Current methodologies are inadequate to this task.

References

[1] Barry Boehm. A spiral model of software developmnet and enhancement. *Computer*, pages 43–57, Sept 1987.

[2] Grady Booch. *Object-Oriented Analysis and Design with Applications*. The Benjamin/Cummings Publishing Company, Inc, 2nd edition, 1994.

[3] Paul Booth. *An Introduction to Human-Computer Interaction*. Lawrence Erlbaum Asociates, 1989.

[4] Stuart Card, Thomas Moran, and Allen Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, 1983.

[5] Erran Carmel, Randall Whitaker, and Joey George. Pd and joint application design : A transatlantic comparison. *Communications of ACM*, 36(4), June 1993.

[6] John Carroll and Thomas Moran. Introduction to this special issue on design rationale. *Human-Computer Iteraction*, 6:197–200, 1991.

[7] Andrew Clement and Peter Van den Besselaar. A retrospective look at pd projects. *Communications of ACM*, 36(4), June 1993.

[8] Andy et al Downton. *Engineering the Human-Computer Interface*. McGraw-Hill Book Company Europe, 1991.

[9] Wasserman et al. Developing interactive information systems with the user software engineering methodolgy. In Ronald Baecker and William Buxton, editors, *Human-Computer Interaction*. Morgan Kaufmann Publishers, Inc., 1987.

[10] Finn Kensing and Andreas Munk-Madsen. Pd : Structure in the toolbox. *Communications of ACM*, 36(4), June 1993.

[11] Michael Muller, Daniel Wildman, and Ellen White. Taxonomy of pd practices : A brief practitioner's guide. *Communications of ACM*, 36(4), June 1993.

[12] Brad Myers. Why are human-computer interfaces difficult to design and implement? Technical report, CCarnegie Mellon University, 1993.

[13] Jakob Nielsen. *Usability Engineering*. AP Professional, 1993.

[14] Shari Pfleeger. *Software Engineering*. Macmillan Publishing Co., 1987.

[15] Roger Pressman. *Software Engineering A practitioner's Approach*. McGraw-Hill Inc, 3rd edition, 1992.

[16] Winston Royce. Managing the development of large software systems: Concepts and techniques. *WESCON technical papers*, 14, 1970.

[17] A.I. et al Wasserman. Developing interactive information systems with the user software engineering methodology. *IEEE Transactions on Software Engineering*, 12(2), 1985.