CHAPTER V

5. HCI in the software process

5.1 Introduction
5.2 The software life cycle
5.3 Usability engineering
5.4 Iterative design and prototyping

5. HCI in the Software Process

- Software engineering provides a means of understanding the structure of the design process, and that process can be assessed for its effectiveness in interactive system design.
- Usability engineering promotes the use of explicit criteria to judge the success of a product in terms of its usability.
- Iterative design practices work to incorporate crucial customer feedback early in the design process to inform critical decisions which affect usability.
- Design involves making many decisions among numerous alternatives. Design rationale provides an explicit means of recording those design decisions and the context in which the decisions were made.

5.1 Introduction

Within computer science there is already a large sub-discipline that addresses the management and technical issues of the development of software systems – called *software engineering*.

One of the cornerstones of software engineering is the *software life cycle*, which describes the activities that take place from the initial concept formation for a software system up until its eventual phasing out and replacement.

5.2 The Software Life Cycle

One of the distinguishing characteristics of any engineering discipline is that it entails the structured application of scientific techniques to the development of some product. A fundamental feature of software engineering, therefore, is that it provides the structure for applying techniques to develop software systems.

The software life cycle is an attempt to identify the activities that occur in software development. These activities must then be ordered in time in any development project and appropriate techniques must be adopted to carry them through.

In the development of a software product, we consider two main parties: the customer who requires the use of the product and the designer who must provide the product.

5.2.1 Activities in the life cycle

The graphical representation is reminiscent of a waterfall, in which each activity naturally leads into the next. The analogy of the waterfall is not completely faithful to the real relationship between these activities, but it provides a good starting point for discussing the logical flow of activity. We describe the activities of this waterfall model of the software life cycle next.

Requirements specification

In requirements specification, the designer and customer try to capture a description of *what* the eventual system will be expected to provide. Requirements specification involves eliciting information from the customer about the work environment, or domain, in which the final product will function.

Requirements specification begins at the start of product development. Requirements are usually initially expressed in the native language of the customer.

Architectural design

The first activity is a high-level decomposition of the system into components that can either be brought in from existing software products or be developed from scratch independently. An architectural design performs this decomposition. It is not only concerned with the functional decomposition of the system, determining which components provide which services.

There are many structured techniques that are used to assist a designer in deriving an architectural description from information in the requirements specification.

Detailed design

The architectural design provides a decomposition of the system description that allows for isolated development of separate components which will later be integrated. For those components that are not already available for immediate integration, the designer must provide a sufficiently detailed description so that they maybe implemented in some programming language.

Coding and unit testing

The detailed design for a component of the system should be in such a form that itis possible to implement it in some executable programming language. After coding,the component can be tested to verify that it performs correctly, according to sometest criteria that were determined in earlier activities.

Integration and testing

Once enough components have been implemented and individually tested, they must be integrated as described in the architectural design. Further testing is done to ensure correct behavior and acceptable use of any shared resources.

Maintenance

After product release, all work on the system is considered under the category of maintenance, until such time as a new version of the product demands a total redesign or the product is phased out entirely. Consequently, the majority of the lifetime of a product is spent in the maintenance activity. Maintenance involves the correction of errors in the systems, which are discovered after release and the revision of the system services to satisfy requirements that were not realized during previous development.

5.2.2 Validation and verification

Verification of a design will most often occur within a single life-cycle activity or between two adjacent activities.

Validation of a design demonstrates that within the various activities the customer's requirements are satisfied. Validation is a much more subjective exercise than verification, mainly because the disparity between the language of the requirements and the language of the design forbids any objective form of proof.

Validation proofs are much trickier, as they almost always involve a transformation between languages. Furthermore, the origin of customer requirements arises in the inherent ambiguity of the real world and not the mathematical world.

5.2.3 Management and contractual issues

The life cycle described above concentrated on the more technical features of software development. In a technical discussion, managerial issues of design, such as time constraints and economic forces, are not as important. The different activities of the life cycle are logically related to each other.

In management, a much wider perspective must be adopted which takes into account the marketability of a system, its training needs, the availability of skilled personnel or possible subcontractors, and other topics outside the activities for the development of the isolated system.

A signed requirements specification indicates both that the customer agrees to limit demands of the eventual product to those listed in the specification and also that the designer agrees to meet all of the requirements listed.

From a technical perspective, it is easy to acknowledge that it is difficult, if not impossible, to determine all of the requirements before embarking on any other design activity.

5.2.4 Interactive systems and the software life cycle

The traditional software engineering life cycles arose out of a need in the 1960s and1970s to provide structure to the development of large software systems. In those days, the majority of large systems produced were concerned with data-processing applications in business.

These systems were not highly interactive; rather, they were batch-processing systems. Consequently, issues concerning usability from an end user's perspective were not all that important.

5.3 Usability Engineering

One approach to user-centered design has been the introduction of explicit *usability engineering* goals into the design process, as suggested by Whiteside and colleagues at IBM and Digital Equipment Corporation and by Nielsen at Bellcore.

Engineering depends on interpretation against a shared background of meaning, agreed goals and an understanding of how satisfactory completion will be judged. The emphasis for usability engineering is in knowing exactly what criteria will be used to judge a product for its usability.

The ultimate test of a product's usability is based on measurements of users' experience with it. Therefore, since a user's direct experience with an interactive system is at the physical interface, focus on the actual user interface is understandable.

The backward recoverability attribute is defined in terms of a *measuring concept*, which makes the abstract attribute more concrete by describing it in terms of the actual product.

The *measuring method* states how the attribute will be measured, in this case by the number of explicit user actions required to perform the undo, regardless of where the user is in the programming sequence.

The *worst case* value is the lowest acceptable measurement for the task, providing a clear distinction between what will be acceptable and what will be unacceptable in the final product.

The *planned level* is the target for the design and the *best case* is the level which is agreed to be the best possible measurement given the current state of development tools and technology.

5.3.1 Problems with usability engineering

The major feature of usability engineering is the assertion of explicit usability metrics early on in the design process which can be used to judge a system once it is delivered.

Set levels with respect to information on:

1. an existing system or previous version
2. competitive systems

3. carrying out the task without use of a computer system
4. an absolute scale
5. your own prototype
6. user's own earlier performance
7. each component of a system separately
8. a successive split of the difference between best and worst values observed in user tests

5.4 Iterative design and prototyping

The only way to be sure about some features of the potential design is to build them and test them out on real users. The design can then be modified to correct any false assumptions that were revealed in the testing.

This is the essence of *iterative design*, a purposeful design process which tries to overcome the inherent problems of incomplete requirements specification by cycling through several designs, incrementally improving upon the final product with each pass.

On the technical side, iterative design is described by the use of *prototypes*, artifacts that simulate or animate some but not all features of the intended system. There are three main approaches to prototyping:

- **Throw-away** The prototype is built and tested. The design knowledge gained from this exercise is used to build the final product, but the actual prototype is discarded.

- **Incremental** The final product is built as separate components, one at a time. There is one overall design for the final system, but it is partitioned into independent and smaller components. The final product is then released as a series of products, each subsequent release including one more component.

- **Evolutionary** Here the prototype is not discarded and serves as the basis for the next iteration of design. In this case, the actual system is seen as evolving from a very limited initial version to its final release.

- **Time** Building prototypes takes time and, if it is a throw-away prototype, it can be seen as precious time taken away from the real design task. So the value of prototyping is only appreciated if it is fast, hence the use of the term *rapid prototyping*.

- **Planning** Most project managers do not have the experience necessary for adequately planning and costing a design process which involves prototyping.

- **Non-functional features** Often the most important features of a system will be non-functional ones, such as safety and reliability, and these are precisely the kinds of features which are sacrificed in developing a prototype.

- **Contracts** The design process is often governed by contractual agreements between customer and designer which are affected by many of these managerial and technical issues. Prototypes and other implementations cannot form the basis for a legal contract, and so an iterative design process will still require documentation which serves as the binding agreement.