INFORMATION SOCIETIES TECHNOLOGY (IST) PROGRAMME

STATUS

"Software Architecture for Usability"

WORKPACKAGE 5. Integrated Development Process with Usability Techniques

DELIVERABLE D.5.2 SPECIFICATION OF THE SOFTWARE PROCESS WITH INTEGRATED USABILITY TECHNIQUES

Version: 1.0

Submission Date: 25/11/2002

Authors: **Xavier Ferré, Natalia Juristo, Ana Moreno**

Partners: UPM

Stage:	**Confidentiality:**
[ ] Draft	[x] Public - for public use
[ ] To be reviewed by WP participants	[ ] IST – for IST programme participants
[ ] Pending of approval by next consortium meeting	[ ] Restricted – for STATUS consortium and PO
[x] Final / Released to CEC	

DOCUMENT CONTROL

Registration of Changes

Date	Version	Author of Changes	Comments
23 / 8 / 02	0.2	Xavier Ferré	Document structure is established
16 / 9 / 02	0.3	Natalia Juristo, Xavier Ferré	Changes after internal UPM review
25 / 9 / 02	0.4	Natalia Juristo, Xavier Ferré	The document is divided into two parts, and delta structure is modified.
31 / 10 / 02	0.5	Natalia Juristo, Xavier Ferré	Part III. Training Strategy added
20 / 11 / 02	0.6	Natalia Juristo, Xavier Ferré	English language revision
22 / 11 / 02	1.0	Xavier Ferré	Final version to send to CEC

List of STATUS Related Documents

Document Name	Version
D.5.1 Selection of the Software Process and the Usability Techniques for Consideration	1.0
Technical Annex	1.0

ACRONYMS AND ABBREVIATIONS

Acronyms and Abbreviations	Meaning
GOMS	Goals, Operations, Methods and Selection Rules
JEM	Joint Essential Modelling
SWEBOK	SoftWare Engineering Body of Knowledge
WP	Work Package

TABLE OF CONTENTS

0 INTRODUCTION

0.1 Purpose

This document presents the final results of Work Package 5: Integrated Development Process with Usability Techniques. It complements the findings of the project on architectural-related issues with other aspects that need to be incorporated into the development process for achieving a sufficient usability level in the software final product.

As specified in the Technical Annex, the architectural focus supports, but cannot guarantee, usability, since not all usability attributes and factors can be promoted by the software architecture. To complement the software architecture-related findings, a software development process with integrated usability techniques is to be produced within the scope of the project.

When the project was conceived, the planned approach was to define a complete development process that the development organization should follow to achieve a good usability level for the software product. After some discussion with project partners, especially industrial partners, a new approach was devised, which is more flexible and can lead to a more extensive use of the results of WP5. The new approach involves structuring the usability techniques to be integrated in packages that can be incorporated into an existing development process. Nevertheless, there are still some requirements that must be met by the development process for this integration to be feasible. The requirements for a user-centred development process were described in section 2 of D.5.1, and they are: user involvement, adequate understanding of user and task requirements and iterative process. It should be noted that the first two conditions are eased by the packages proposed in WP 5. The third, iterative process, however, is a condition that the development style followed by the organisation that intends to use our results must meet. Having established the minimum requirements for a development process to be a candidate for the inclusion of usability aspects in D.5.1, the usability techniques and activities were studied. Usability activities were then surveyed to establish a preliminary set of usability activities to be considered for inclusion in the packages. Likewise, usability techniques were surveyed, and a preliminary set of usability techniques was established. These results were presented in D.5.1.

This document follows upon the work in D.5.1, going from the selection in D.5.1 to the definition of the packages to be incorporated into the development process. First of all, we need to map usability activities to software engineering development activities so that developers can see where these new activities fit into their picture of the development process. For the purpose of this mapping, some of the activities described in D.5.1 had to be reconsidered. These modifications led to some activities being rearranged (for instance, they have moved from being considered as design activities to be considered as analysis activities), and some techniques have been upgraded to activities. Having fitted usability activities into software engineering activities, we have mapped usability techniques to activities.

If the development process in place at the organisation is iterative, albeit elementary, that is, it has four basic milestones: analysis, design, implementation and evaluation, the match between techniques and activities can provide help enough to developers about when to incorporate usability aspects into their development, as the techniques are classified according to precisely these four development times. However, more and more organisations using iterative development have sophisticated processes in the style of the Unified Process [Jacobson, 99], where the iteration does not take place on the basic four major milestones (analysis, design, implementation and evaluation) but on development stages where one type of activities carry more weight than others (for example, there is a bigger workflow for analysis than for the other activities in the elaboration stage). Therefore, we thought it necessary to further refine the packages we propose so that they also fit into these process types. For this purpose, we had to establish the development stages that make up an elaborate iterative process and relate these to the proposed packages.

Having defined the packages that set out the usability aspects to be incorporated into the iterative development process, the objective of WP5 could be considered as achieved and the document concluded. However, we realised that it took a lot of discussion to arrive at the package definition, all of which is somewhat tiresome for average developers, who just want us to provide the solution they need. So, we have composed a second part of the document, consisting of documentation for developers, which contains the information developers need to apply our results. Therefore, part II presents the guidelines for a developer to fit usability techniques into an existing software development process. It compiles the results of WP 5, packaged for use by software developers.

And we have gone one step further. Technology transfer is not a simple matter, and we are aware that it is not enough to provide developers with a document for them to change the way they do things. 'Care about usability' is a change to the philosophy and viewpoint to which developers are accustomed. So, to help with the technology transfer not only of WP5 but also of all the project results, we have designed a draft training strategy that establishes what type of course developers should attend to be trained in the results of the STATUS project, and incorporate the usability aspects considered in the packages (which cover both traditional usability techniques and activities and the results of WP3) into their modus operandi. Additionally, the training plan provides a project output that can be sold together with the deltas (WP5) and the architectural design that supports usability (WP3).

In short, the purpose of this document is to relate and organize the findings presented in D.5.1, so that they are linked to software process variables, such as type of activity and development time and then package them into delta increments that are easily manageable for software developers. An additional objective of this document is to provide a training strategy to ensure that developers are able to use the packages.

0.2 Document Structure

The document has been divided into three parts. Part I contains the theoretical discussion on the decisions taken for the assignment of usability techniques to delta increments. Part II has a practical focus, compiling the results of Part I into a guideline for developers, which includes the necessary information for the application of the results. Part III proposes a training strategy to be followed by a software development organization interested in using WP5 results.

Part I is divided into the following sections:

- Section 1 details the mapping between the usability activities described in deliverable D.5.1 and the general development activities into which we want to integrate usability techniques, so as to link the work in D.5.1 to the modified structure followed in this document.
- Sections 2 to 4 classify the usability techniques analysed in deliverable D.5.1 by development activities (analysis, design and evaluation) and their subactivities.
- Section 5 describes the time constraints that both usability techniques and activities have to meet for them to be useful in the development process.
- Section 6 presents the structure for the delta increments to be incorporated into a software development process.
- Section 7 details the references used in this part.

Part II contains the following sections:

- Section 1 details the conditions to be met by the existing development process in order to allow for the inclusion of usability techniques (as described in D.5.1).
- Section 2 presents the delta increments that package the usability techniques in sets of related usability techniques to be applied close together in terms of development time. There are two

possibilities within this section: delta increments for a light software development process in section 2.1, and delta increments for an elaborate software development process in section 2.2.

- Finally, section 3 is a catalogue of techniques, designed as a guide to their application and a pointer to further information.

The following sections form part III:

- Section 1 outlines the training course, detailing the course contents.
- Section 2 establishes the duration for the course subjects.
- Section 3 describes the resources needed for the course.
- Section 4 lists the references consulted for this part.

Part I. IMPROVING THE SOFTWARE PROCESS WITH USABILITY ASPECTS

I.1 MAPPING USABILITY ACTIVITIES TO SOFTWARE ENGINEERING ACTIVITIES

In D.5.1, we studied the established usability activities in a usability-centred software development process and selected certain activities from the usability literature. We then took the same approach to review and study usability techniques. Now we need to incorporate these activities and techniques into the traditional software engineering development process. We propose to incorporate the activities and techniques by defining some increments that developers can plug into their development process.

The set of activities defined in D.5.1. is based on the terminology used in the usability field, with which most software developers are not familiar. Therefore, we do need to translate the terms to a generally accepted software engineering terminology, so we can tell the developers where to plug in the increments. For this purpose, we have mapped the usability activities from D.5.1 to the development activities we will consider in this document, as shown in Figure I.1.1. The development activities we have considered come from different sources:

- Since usability activities are intertwined with other development activities in analysis, they can be directly mapped to the different kinds of software engineering analysis efforts. We have followed the SWEBOK (Software Engineering Body of Knowledge) [IEEE, 01] for analysis activities. **We have taken the SWEBOK requirements activities** and have selected those that are relevant for our purpose, because there are usability techniques that need to be introduced: Requirements Elicitation, Requirement Analysis, Requirement Specification and Requirements Validation.
- The activities of **Develop Product Concept and Prototyping have now been allocated as analysis techniques**, while they were previously considered as design activities. Prototyping is considered in software engineering as a technique that can be used in Requirements Elicitation [IEEE, 01]. As for Develop the Product Concept, it is a design activity, but the kind of design that is known as innovation design, where the design team is trying to give form to an abstract concept that defines the system that is going to be built. This concept is usually elicited from users or other stakeholders, and it is fundamental for the success of the requirements engineering efforts. Because of its close connection with requirements activities, we have now considered it to be an analysis activity.
- **For design and evaluation we have created some new activities** in order to make room for usability activities. We have followed this approach for design since usability-related design activities are quite independent from general design activities. More specifically, we have created an activity called Interaction Design to pack the usability activities related to this kind of design. Interaction Design has been further decomposed into Detailed Interaction Design and User Interface Design, for the sake of clarity. Regarding evaluation, we have created another new activity, which is Usability Evaluation, since it groups usability techniques that are independent from other general evaluation activities.
- **Some groups of usability techniques have been upgraded to usability activities**, since they represent a kind of usability activity to be performed. We have taken this approach for the three kinds of activities into which Usability Evaluation is decomposed: Expert Evaluation, Usability Testing and Follow-Up Studies of Installed Systems. Therefore, the Usability Evaluation activity has been decomposed into these three big families of usability evaluation activities. However, walkthroughs can be used during requirements validation, so they have been highlighted in Usability Evaluation activities (on the left in Figure I.1.1) to show this link with analysis. Additionally, we identified in D.5.1 that help needs to be designed, but it seemed to be more a technique than an activity at that time. However, we now recommend approaching Help Design as a task to be accomplished during design. The activities that did not exist in the usability activities decomposition in deliverable D.5.1 and which we have identified as proper usability activities at this stage are highlighted in italics in Figure I.1.1.

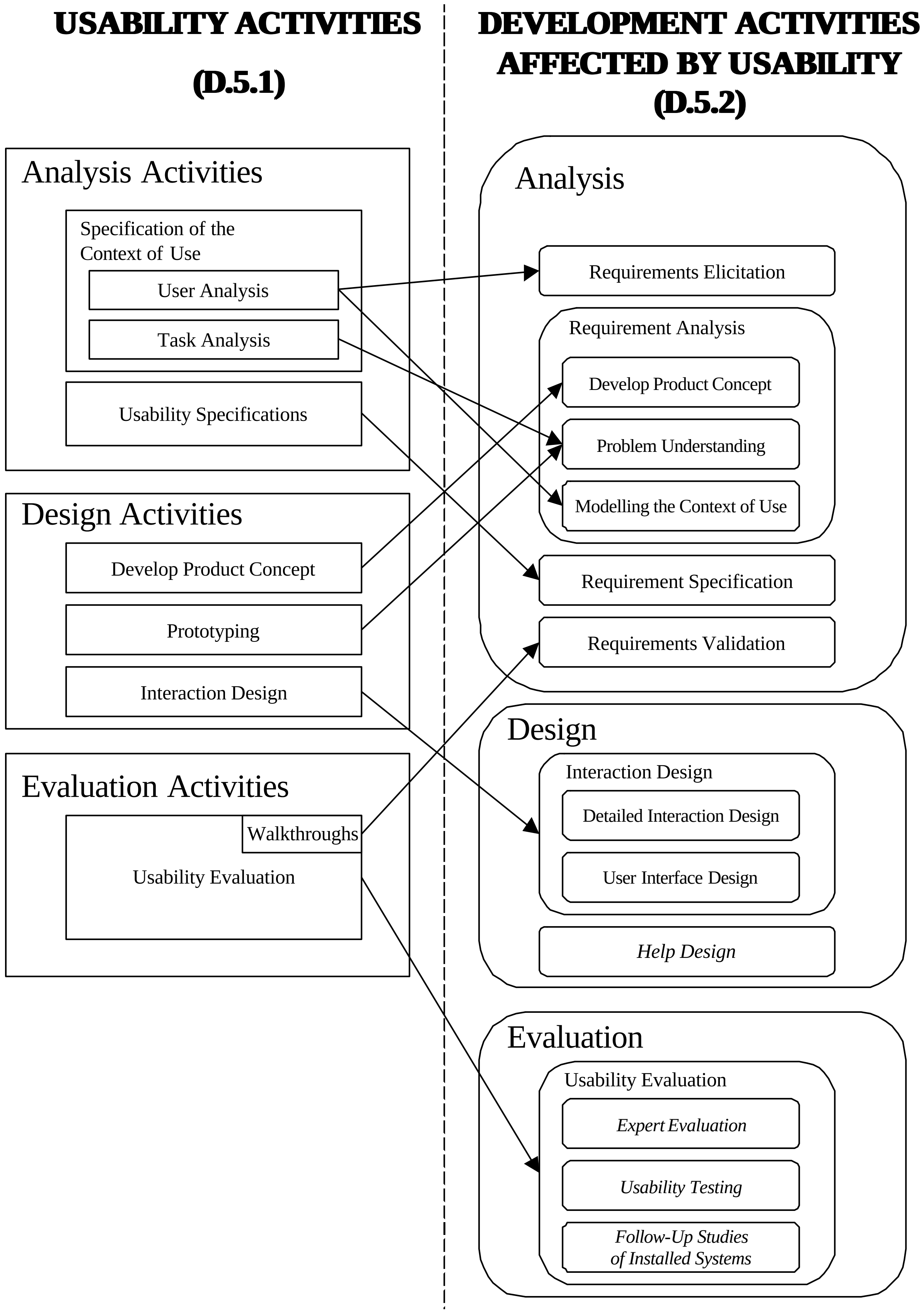

Figure I.1.1 Mapping between Usability Activities and Software Engineering Activities

The other main issue tackled in D.5.1 was usability techniques. The following three sections study their location in relation to the development process: techniques for analysis, design and evaluation.

I.2 TECHNIQUES FOR ANALYSIS

To allocate analysis-related techniques, they will be grouped by the requirements activities established in the SWEBOK [IEEE, 2001]: Requirements Elicitation, Requirement Analysis, Requirement Specification, and Requirement Validation.

I.2.1 Requirements Elicitation

Requirements elicitation is a hard task to perform, and it is crucial for the success of the whole project. The techniques recommended in the usability field for eliciting information about the user may help in the performance of requirements elicitation as a source of inspiration for the requirements engineering team, providing additional sources of information. These techniques are as follows:

- **Ethnographic Observation**: This technique is the most genuine elicitation technique of all the ones considered in this section. It is a method used in anthropology, which is widely employed for usability purposes. Since interface users form a unique culture, ethnographic methods for observing them at the workplace are likely to become increasingly important. As ethnographers, developers need to gain insight into individual behaviour and the organisational context. The information gathered about the users' culture is a source for requirements.
- **Contextual Inquiry**: In contextual inquiry, users and developers participate to identify and understand usability problems within the normal working environment of the user. It is a form of elicitation that is usually performed for evaluation purposes. Nevertheless, it can be also used to find problems with previous versions of the software or with competitor products, so that new requirements that address these issues can be identified.

I.2.2 Requirement Analysis

The requirements obtained as a result of requirements elicitation are classified, and the software boundaries are delimited. We distinguish three kinds of activities in Requirements Analysis: Develop Product Concept, Problem Understanding and Modelling.

I.2.2.1 Develop Product Concept

Before describing the functionalities that the system must provide in detail, there needs to be agreement between all the stakeholders about the kind of system to be developed. The set of techniques described below help the requirements analyst to envision the product and to communicate the concept of the product to the relevant stakeholders to validate the correctness of the chosen approach.

- **Visual Brainstorming**: This is a sketching technique employed for exploring alternative product concepts. After producing initial sketches, the best ideas can be further developed by constructing more detailed screen mock-ups representing the concept, which can be evaluated with the project stakeholders. This can then be followed by developing scenarios, software or video prototypes.
- **Competitive Analysis**: It is a non-usability-specific technique for analysing existing products to get inspiration about the requirements for the system to be built. But analysing competing products from a heuristic usability point of view, or even performing some sort of empirical user tests, can help the requirements engineering team to identify what is most needed and what is not needed at all. Even if an analysed product does not belong to the target domain, it can serve as a good starting-point for establishing similarities with the system that is going to be developed. Widely known commercial products serve as good references for establishing

the product concept, and a competitive analysis of their benefits from a usability point of view can help to focus the discussion and the decision-making process.

- **Scenarios**: For systems that will suffer substantial changes or when a new application is planned, there are usually no reliable data on the range and distribution of task frequencies and sequences. An early and easy way to describe a new system is to write scenarios of usage and then, if possible, to act them out as a form of theatre. This can lead to the identification of requirements that would otherwise remain uncovered and to the definition of the kind of problems the system will help to solve. Scenarios can be used to get information from users and/or domain experts or to approach system usage for the first time. Scenarios should be used with the special flavour with which usability authors describe it, even though it is a technique already used as part of requirements engineering.
- **Post-It Notes**: Because of their versatility and flexibility, post-it notes can serve as a basis for defining the system concept in a collaborative procedure in which the customer and/or user representatives take part. The information gathered can be displayed in a post-it placeholder. All the participants in the meeting can look at the post-it placeholder to find out what has been discovered so far and explore more knowledgeably what additional information is needed. Possible changes can be shown by moving issues around from one category to another in the post-it structure to enhance the discussion. This technique can help in the development of the product concept by providing a tool for group discussion whose use requires no technical knowledge.

I.2.2.2 Problem Understanding

The problem needs to be thoroughly analysed to come up with a definition of what system will be developed to solve the problem. An important part of problem understanding is establishing the system boundaries, and the modelling of the interaction between the system and the environment. Usability is directly affected by the definition of this interaction, in particular, system-user interaction. Therefore, we consider that the usage of usability techniques is especially important for this kind of activity.

For the definition of the system boundaries, and the interaction with the environment, we propose the use of the following techniques:

- **Essential Use Cases**: As conceived, use cases are a user-centred technique, since they describe the system-user interaction from the user point of view. They are used as a complement in Requirements Engineering, usually for object-oriented development, and they are transformed into system-oriented artefacts as development moves forward. However, we recommend adopting the kind of task modelling that is employed in usability as a requirements analysis activity, which is truly user-centred. We want to use the technique keeping the user-centeredness. For this purpose, we propose using the specific distinction, made by Constantine and Lockwood [Constantine, 99], between essential use cases and detailed use cases. Essential use cases are defined at a higher level of abstraction in terms of user intentions and system responsibilities, keeping a technology-free and implementation-independent focus. They can be used to work with use cases at the beginning of the development process, without having to make too many decisions on the details of the user interface. Note that essential refers to the abstract focus used for the use case description and it is applicable to all use cases, it does not refer to a particular set of especially important use cases.
- **Cognitive Task Analysis (GOMS)**: In the design of the interaction between the system and the user, apart from the physical steps that the user will take, such as clicking on a button or entering data, there are some cognitive or mental activities that take place in the user's mind. Cognitive Task Analysis deals with modelling the user's internal representation and processing that occurs for the purpose of designing tasks that can be undertaken more

effectively by humans. With GOMS (Goals, Operations, Methods and Selection Rules), we can take into consideration cognitive issues that are raised by the specific design of the task at hand, combined with the physical actions. Only the two more abstract levels would need to be explored for Problem Understanding, that is, goals and operations. Methods and selection rules can be explored to complement design activities (see section I.3.1.1 below). According to Preece et al. [Preece, 94]: "GOMS has a number of problems, both with the ease of using the method itself and on the results it produces (see Reisner, 1987) [...] the method requires a lot of time, skill and effort to use". Despite these negative remarks, Preece et al. also note that a number of authors find it useful. We propose the usage of a GOMS model because it is the best option for cognitive task analysis. Although it can be difficult to apply, there are no easier alternatives that provide the possibility of applying predictive metrics.

- **Prototypes for Problem Understanding**: One big problem during analysis is the possibility of pursuing the wrong system. System models are alien to a non-technical audience and can lead to communication problems when shown to stakeholders, but prototypes are understandable by customers or users as a draft of the future system. They can help in face of a situation such as "I cannot explain what I want, but I will recognize when I see it". Prototyping is the form of modelling that can be more easily transmitted to the project stakeholders when they include people not familiar with technical models. Even though prototyping is not a usability-specific technique, it is used extensively in Usability Engineering. We will focus on non-functioning prototypes, as they are the techniques with which software developers are less familiar. In the same direction, requirements animation is fairly common in traditional software development, but we recommend using it with a usability focus. The different kinds of non-functioning prototypes plus requirements animation are described below:
  - **Paper Prototypes**: They include paper prototypes, computer drawings prepared with graphics software, and non-functioning mock-ups created using programming tools. This kind of prototypes overcome the problem with customers who think that when they see a functioning prototype most of the system has already been built. With paper prototypes, customers or users are able to envision what is being proposed, but they get the idea that a lot of work still has to be done.
  - **Chauffeured Prototypes**: The user watches while another person, usually a member of the development team, "drives" the system. The system can be just paper prototypes that are given a dynamic dimension with the explanations and changes performed by the "chauffeur" of the session.
  - **Wizard of Oz**: The user interacts with a screen, but instead of a piece of software responding to the user actions, a developer is sitting at another screen answering the user requests. The user is unaware of the fact that the answers are being given by a person instead of a software system. This kind of prototyping session is halfway between paper prototypes and requirements animation, as the user gets the feeling of a working system, even if there is almost no implementation behind it.
  - **Requirements Animation**: Possible requirements are demonstrated in a prototype, which can then be assessed by users. These prototypes are also known as mock-ups, and they involve some limited implementation of system functionality.

I.2.2.3 Modelling the Context of Use

The context of use is an important issue for understanding the problem before making any design-related decision that could be based on wrong premises. The assumption is that usability issues are an important part of the problem, and the techniques described in this section help the developers to model the details of the environment that are important from a usability point of view and the tasks the user needs to perform.

The techniques for modelling the context of use are:

- **Structured User Role Model**: A role model is a list of the user roles to be supported by a system, which describes each role in terms of the needs, interests, expectations, behaviours and responsibilities that characterise and distinguish that user role. User roles and their relationships are represented in a user role map. These models can be difficult to define, especially for the inexperienced analyst and for complex systems. The structured user role model offers a systematic way of capturing as much relevant information as possible about the relationships of users to the planned system. Unlike other approaches to user modelling, which are somewhat vague, we prefer this technique to model the user because it provides the kind of information needed, albeit with a more defined structure that can help non-usability experts to work with user information. This technique is mostly concerned with requirement analysis, but it is also relevant for requirements elicitation, as it can help to identify the right sources for the elicitation activities.
- **Operational Modelling**: The operational model is a collection of various operational and contextual influences that can play a role in usability. These collections are called operational profiles. Operational factors that can affect the system to be developed include: aspects of the physical work environment, features and limitations of operating equipment and interface devices, and general and specific operational risk factors. Operational modelling gathers the kind of information that describes the user environment in a broad sense, so it can be a complement to a traditional conceptual model. This model extends the structured user role model.
- **Use Case Diagram - Detailed Use Cases**: Task modelling is a core usability activity. While essential use cases describe user intentions and what the system may offer in response to user goals, detailed use cases address this interaction in more detail, specifying the data exchange that will take place. Use cases (both essential and detailed) are described in a structured manner, where the narrative is divided into two parts: the user action model and the system response model. Additionally, a Use Case diagram is created to represent the overall usage scheme. Detailed Use Cases can be the basis for prototypes to be built and shown to the user for feedback.
- **Joint Essential Modelling (JEM)**: JEM is a structured, facilitated, collaborative process for involving users along with developers in modelling activities. The main models for which this process can be applied are the structured user role model and the use cases (including the use case diagram and a use case prioritisation). JEM is based on JAD (Joint Application Design), a traditional software engineering technique, modified to adopt a usability focus. Therefore, JEM (Joint Essential Modelling) is not related to a particular model, but to a process for incorporating the user into the activity of modelling the context of use.

I.2.3 Requirement Specification

Requirements specification is concerned with the structure, quality and verifiability of requirements. Usability requirements, in a verifiable form, can be added to the software requirements in the form of usability specifications. **Usability specifications** are quantitative usability goals, which are used as a guide for ascertaining when the usability of a software system is good enough. They can be based on objective or subjective measures. Objective measures are commonly associated with a specific **benchmark task**, while subjective measures are commonly associated with a **user questionnaire**.

The inclusion of usability specifications in the requirements specification introduces usability as yet another aspect of the system that can be established quantitatively and in advance, and detailed in the requirements specification. By means of these usability specifications, developers can focus on the level of usability required, and then iteratively design the system to meet the usability goals.

I.2.4 Requirements Validation

Four sub-areas form the knowledge area of requirements validation: requirements reviews, prototyping, model validation and acceptance tests.

I.2.4.1 Requirements Reviews

The specification of the interaction between the system and the users described in the essential use cases can be reviewed by means of heuristic usability evaluation, which will be described in section I.4.1.

I.2.4.2 Prototyping

The prototyping techniques that can be adopted for usability have already been described above in section I.2.2.2. These techniques can be used not only for understanding but also for validation purposes.

I.2.4.3 Model Validation

As usability practice involves a combination of user participation and iterative development, it includes techniques for validating preliminary models. In particular, there is a technique that can be used to validate the model of interaction between the system and the user: walkthroughs. Walkthroughs involve constructing carefully defined tasks from a system specification or screen mock-up, so all they require is some sort of prototype or definition of the system-user interaction (essential use cases, for example). There are the following two variants of walkthroughs:

- **Cognitive Walkthrough**: This is a manual simulation of the cognitive activities of the user to identify potential usability problems.
- **Pluralistic Walkthrough**: Heuristic evaluation of the usability of the product is performed by representative users, product developers, and usability specialists.

I.2.4.4 Acceptance Tests

The set of usability specifications defined as usability goals for the system to be developed are defined quantitatively, so that they can be checked for compliance by means of usability testing with representative users. Usability testing is described in section I.4.2.

I.3 TECHNIQUES FOR DESIGN

Following the same procedure as in section I.2 for analysis activities, we have looked up the classification of design activities in the SWEBOK [IEEE, 2001]. But the approach taken by the SWEBOK does not fit in with our approach, since it explicitly says that user interface design is a part of software development that will not be dealt with. SWEBOK is centred on the internal part of the system exclusively, and it is hard to introduce usability techniques at this level of design.

However, we do not need a decomposition of design development activities in order to incorporate usability techniques(as we do for analysis), since the usability activity at design time is very well delimited, and we can define a new activity called Interaction Design, which encapsulates the usability techniques. Therefore, we will consider Interaction Design as part of design activities in the development process. Interaction Design needs to be coordinated with the other design activities, concerned with the internal structure of the software system. In particular, interaction design should be made to accommodate the definition of the interaction between the system and the environment that has been obtained in requirements analysis. And the internal structure of the system should be designed to provide a good implementation of this interaction with the environment.

The online help subsystem must be carefully designed, and this design activity is also well delimited, so we will consider a separate activity that deals with this issue: Help Design.

I.3.1 Techniques for Interaction Design

We can make a distinction between the design of the detailed interaction of the system with the environment, and the design of the user interface elements and their behaviour. The former is called Detailed Interaction Design, and the latter User Interface Design. Impact Analysis can be applied to both and, generally, to any design decision that could affect the usability of the final product.

- **Impact Analysis**: A great many decisions must be taken in design, and this can help to roughly evaluate the suitability of different design alternatives that solve identified usability issues. This technique can be augmented with the use of other quality attributes apart from usability, so it can help the design team to make decisions. As it is helpful to make some implicit knowledge explicit for design, it is a good technique for interdisciplinary design teams, especially when there is user participation.

I.3.1.1 Detailed Interaction Design

The top-level decisions on interaction must have been taken in requirements activities. Detailed Interaction Design specifies the concrete interaction that will take place: the exchange of information between the system and the user(s). The result of this activity is complemented with the result of User Interface Design, where the details of the elements of the user interface are established.

- **Detailed Use Cases**: As part of requirements activities a set of Detailed Use Cases are produced. These use cases are refined during this step to include the kind of controls to be used for the interaction. The controls are defined abstractly, because the exact control appearance is defined in User Interface Design.
- **GOMS (Goals, Operations, Methods and Selection Rules)**: Once we have the detailed interaction that is going to take place between the user and the system, we can calculate the cognitive demands placed on the user and apply predictive metrics on the final usability of the product. GOMS is a cognitive task analysis method that has associated predictive metrics, so it can serve this purpose. Goals and operations should have been defined as part of analysis activities, so the model would be refined by adding methods and selection rules. Then predictive metrics can be applied to get an idea of the usability improvements that a specific design decision may provide as compared with other alternatives.

I.3.1.2 User Interface Design

User Interface Design activities define the user interface, the elements that the user will see, and the way they behave when the user interacts with them.

- **Screen Pictures**: This technique involves sketching some initial pictures of the screen, including the application/interaction objects, menus, buttons, and icons. The functions are labelled, and notes can be added about the behaviour of objects where appropriate.
- **Menu-selection and dialog box trees**: This technique is used for the design of menu-based user interfaces. Dialog box trees represent the tree structure of menus offering a comprehensive view of the whole system. It allows for consistency and completeness checking.
- **Context Navigation Maps**: Context navigation maps allow for a more precise specification of the transitions that take place between interaction spaces in the course of a use case. Several of these interaction spaces or interaction contexts can be activated when enacting a use case. Their relationships and how to navigate from one to another are represented in a context navigation map.

I.3.2 Help Design

Help facilities are a subsystem that needs to be designed for the software product. The technique of Help Design by Use Cases provides a structure for the help facilities that is based on the usability premises behind use cases.

- **Help Design by Use Cases**: Each use case becomes an entry in the help file. The use cases reflect user goals and they are expressed in the user's language. It is reasonable to think that a help subsystem built around the user's goals and language will be more effective than a system-oriented one.

I.4 TECHNIQUES FOR EVALUATION

Usability evaluation is the most widely explored subfield in usability. The biggest trend in usability has focused on usability evaluation, so there are several usability techniques for each kind of evaluation, each one suited for projects or products with specific characteristics.

The three kinds of usability evaluation activities are:

- Expert Evaluation
- Usability Testing
- Follow-up Studies of Installed Systems

I.4.1 Expert Evaluation

Usability Engineering proposes Expert Evaluation as an alternative for Usability Testing in some iterative cycles, but never as the only source for usability evaluation. Depending on the formality of the evaluation, there are two main groups of expert evaluation techniques: Heuristic Evaluation, less formal; and Inspections, with a greater degree of formality.

- **Heuristic Evaluation**: Heuristic evaluation is done by looking at a system and trying to come up with an opinion about what is good and bad about its usability. It is better to have several evaluators evaluate the same design independently, as they discover far more errors than a single evaluator. The ideal thing is to have usability specialists perform the heuristic evaluation. The evaluator makes a critique founded both on his or her interaction design experience and on generally accepted usability guidelines.
- **Inspections**: Inspections refer to any of several forms of more formal, systematic processes for locating usability problems than heuristic evaluations. It is an examination of a finished product, a design, or a prototype from the standpoint of its ultimate usability by intended users. Usability inspections employ developers and/or usability specialists, sometimes in conjunction with users, to identify usability defects. When performed by different stakeholders in a collaborative effort, it is called Collaborative Usability Inspection. In this case, the review process is a team effort that includes software developers, end users, application or domain experts and usability specialists, collaborating to perform a thorough and efficient inspection. There are two variants of inspection, which have a specific focus:
    - **Consistency Inspections**: In consistency inspections, the goal is to identify inconsistencies across interaction contexts and their contents. The evaluators check for consistency of terminology, colour, layout, input and output formats, and so on. When the product belongs to a family of products, teams of designers, at least one from each project, meet to inspect the usability of the different products of the family.
    - **Conformance Inspections**: In conformance inspections, the goal is to identify departures from the governing user interface standards or style guidelines.

I.4.2 Usability Testing

Usability testing implies having some sort of functioning system (it can be either the finished product or a prototype), which is presented to a set of representative users who are asked to perform some usual tasks. The main usability testing techniques can be divided in four groups: the variants of thinking aloud, post-test feedback, usability specifications measurement and field usability testing. Finally, there are some techniques associated with usability laboratories that will be described as optional techniques to be employed only when the software development organization has the resources to set up such facilities.

I.4.2.1 Thinking Aloud

Thinking aloud is a technique that can be employed in any usability test. It involves having a test subject use the system while continuously thinking out loud. Its focus is on qualitative data and not on performance measures. The idea is to get the user's impression while using the system to avoid later rationalisations. The aim of this kind of testing is to detect usability problems, along with the real causes. The following techniques are variants of the basic think-aloud protocol:

- **Constructive Interaction**: It involves having two test users use a system together. It is also called Codiscovery Learning. It aims to overcome the problem of shy test participants who do not verbalise easily. It is based on the fact that people are used to verbalising when they are trying to solve a problem in a collaborative effort.
- **Retrospective Testing**: The usability testing session is recorded on videotape and the user is requested to review the recording. User comments while reviewing the tape are sometimes more extensive than comments while performing the task in the test. The reviewer can stop the tape and ask the user questions at any time, without fear of interfering with the test, which has essentially been completed already. This variant can be useful when the usability testing involves some kind of performance measurement that could be distorted by dialogue with the reviewer.
- **Critical Incident Taking**: This variant implies recording both negative incidents (signs of frustration, either with remarks or actions) and positive incidents (satisfaction or closure expressions). Negative incidents help to identify the more important usability problems, while positive incidents help to identify metaphors or details to be used more thoroughly in the user interface because of their success
- **Coaching Method**: The reviewer (or "coach") steers the user in the right direction while using the system. The user can ask the experimenter questions, and the questions may show up usability problems that would remain uncovered otherwise. The experimenter will answer to the best of his or her ability.

I.4.2.2 Post-Test Feedback

As a complementary source of information, we can give out some questionnaires for the test participant to fill in to get additional information on the problems on which the test is focused. This technique is called Post-Test Feedback, and it can also take the form of a debriefing or interview with each test participant, where they are typically thanked for their participation and reassured about their performance.

I.4.2.3 Usability Specifications Measurement

The usability specifications document specifies quantitative usability goals, as defined above in section I.2.3. Usability tests can be performed in a prototype to measure how far the usability of the product is from the levels established in the specifications document. Performance measurement is as follows: the test participant is asked to perform the typical tasks defined in the requirements specification, and both the number of errors and the performance time are measured for each task. An alternative name for this kind of testing is Benchmark Tasks.

The product is considered to be finished from a usability point of view when the performance measures attain the levels specified in the specifications document.

I.4.2.4 Field Usability Testing

Field testing takes the usability tests into the workplace. It is performed at the user organisation, although not necessarily at the user workstation and office. There is a variant called **Direct**

Observation, where the reviewer observes the user working with the product or a prototype, but without interfering with the user's work. The observer must work unobtrusively.

There are three other variants of Field Usability Testing that we consider to be optional:

- **Video Recording**: Video logging is an alternative to direct observation, which is much preferred because it provides a permanent record to which you can return as often as necessary later. Analysing video data can be very time consuming. A ratio of 5:1 is often cited: one hour of videotape could take five hours or even a day or more to analyse. For this reason we include this technique as optional.
- **Verbal Protocol**: This technique is like the above but involving audio recording, and it has similar advantages and disadvantages.

I.4.2.5 Laboratory Usability Testing

Laboratory Usability Testing involves tests conducted in a fixed setting specifically configured for usability testing. The main advantage of this kind of usability testing is that it provides a controlled and consistent environment in which to evaluate software usability. Comparing the results of different tests, different users or different systems is easier and more defensible under these conditions.

The deployment of a full-scale usability laboratory is very expensive, and some organizations may find it to have a low investment return. Therefore, we consider the technique of performing usability tests in a laboratory especially prepared for the purpose (with a one-way mirror, several video cameras, etc.) as optional. These kinds of premises are appropriate when the organization is of substantial size or when the budget for usability investment is high enough.

I.4.3 Follow-up Studies of Installed Systems

Installed systems can provide the most faithful information of all the evaluation-related usability techniques. Follow-up studies apply to given software projects, in which the usability of an existing system needs to be improved and to maintenance efforts in the development cycle. Questionnaires and interviews are a typical form of performing follow-up studies of installed systems.

- **Questionnaires**: They provide the development team with user opinions, but no direct information on usability. They are especially suited for getting the user's subjective satisfaction. Questionnaires can be administered by mail, e-mail or with the software itself. Closed questions usually have some form of rating scale. A questionnaire is sometimes used before and after studies of user performance. These are known as pre- and post-questionnaires.
- **Interviews**: Interviews may be conducted personally or over the phone. There are two variants of this technique:
    - **Structured Interviews**: The interview has a fixed structure, and there is no exploration of individual attitudes.
    - **Flexible Interviews**: They generally have some set topics, but no set sequence, and the interviewer is free to follow the respondents' replies and to find out about personal attitudes.

Questionnaires and interviews are the most essential form of usability evaluation of installed systems, but there are other techniques grouped as follows: focus groups, logging actual use, user feedback and surveys. Each technique applies to projects with specific characteristics.

- **Focus Groups**: In a focus group, some users are brought together to discuss their needs and views after the system has been in use for some time. Each group is run by a moderator who is responsible for maintaining the focus of the group on whatever issues are of interest.

- **Logging Actual Use**: Logging involves having the computer automatically collect statistics about the detailed use of the system. It has two main advantages: it does not require the researcher to be present, and it is unobtrusive. It is usually a way of getting information about the field use of a system after release, but it can be used as a supplementary method in usability tests. If software logging is to be applied, the software architecture should make it easy for system managers to collect data about the patterns of system usage, speed of user performance, rate of errors or frequency of requests for online assistance. There are two variants of this technique:
  - **Time-Stamped Keypresses**: A record of each key pressed is kept, along with the exact time of the event.
  - **Interaction Logging**: The whole interaction is recorded, so it can be reproduced completely in real-time.
- **User Feedback**: Feedback can be collected by giving users access to special electronic mail addresses, network newsgroups, or bulletin boards. Users can send their complaints and requests for change or improvement. There are some specific ways of communication that can be employed for collecting user feedback, as follows:
  - **Online or Telephone Consultants**: Consultants are an excellent source of information about the problems users are having and can suggest improvements and potential extensions.
  - **Online Suggestion Box or Trouble Reporting**: Electronic mail can be employed to allow users to send messages to the maintainers or designers. Such an "online suggestion box" encourages some users to make productive comments, since writing a letter may be seen as requiring too much effort.
  - **Online Bulletin Board or Newsgroup**: Users may have questions on the usage or applicability of a software package, and they cannot address anyone in particular. Bulletin boards and newsgroups can be helpful in this case.
  - **User Newsletters and Conferences**: In systems with a substantial number of users who are geographically dispersed, managers have to work harder to create a sense of community. Printed newsletters have an appealing air of respectability. Conferences allow workers to exchange experiences with colleagues and face-to-face meetings increase the sense of community among users. We will consider this communication channel to be optional, since it is only appropriate for systems with a very big user community.
- **Surveys**: Written user surveys are a familiar, inexpensive and generally acceptable companion for usability tests and expert reviews. They can be useful for identifying unsatisfied needs in existing market products.

I.5 WHEN TO INCORPORATE USABILITY INTO THE DEVELOPMENT PROCESS

The approach for integrating usability techniques into the software process has been to define a set of increments to be embedded into an existing development process, which needs to be iterative for the resulting process to offer a noticeable improvement in the usability of the developed software product. Different times or stages can be defined in an iterative process, where one and the same activity may be more or less important or have a different meaning. For instance, during the early stages of the analysis activity, the discovery role is more predominant than in subsequent stages, where specific requirements are being analysed and an understanding role takes over. Besides assigning techniques to activities, as we have done in the previous section, we feel that there is the need to establish constraints concerning the stages where the techniques can be applied. This relationship between usability techniques and activities and development stages is discussed in this section.

The early efforts in the software development process, where the problem is clearly delimited and the basic information is gathered for the later development in the iterative cycles, have been termed **elaboration.** For usability techniques to be applied in iterative cycles (*i*), they will be classed as techniques that are useful for application at any time in the cycle, which we will call **central moments**, and techniques that are suited for application at the end or **final moments** of an iterative cycle. Finally, we will detail the usability aspects that are mainly useful for **evolution** cycles, when the software product is already in operation but needs to be adapted at the end of the development process. These cycles are transition cycles in the Unified Process [Jacobson, 99].

Figure I.5.1 shows the relationship between these stages in development time.

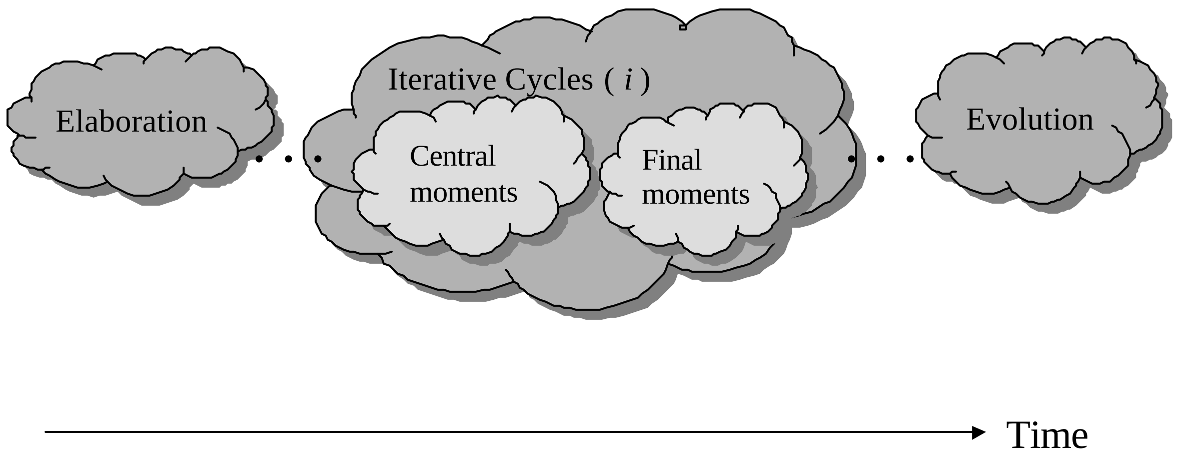

Figure I.5.1 Stages in the Development Process

I.5.1 Time Constraints for Usability Technique Application

I.5.1.1 Techniques for the Elaboration Stage

Before the actual iterative cycles begin, there must be an initial effort where the needs are identified and the general scheme that the system will follow is established. The products of this stage should be quite stable, even though they are open to changes in the iterative development cycles.

The following techniques are clearly to be applied at elaboration time, because they are good for approaching the problem for the first time or handling a solution that is not well defined:

- Ethnographic Observation
- Contextual Inquiry

- Visual Brainstorming
- Scenarios
- Paper Prototypes
- Chauffeured Prototypes

Other techniques can be applied at a later time, but they can help at elaboration time because they are good for coming up with design ideas on the product concept:

- Competitive Analysis
- Post-It Notes

Modelling the user and his or her environment, and the basic dialogue between the system and the user is a prerequisite for any development that aims to care about the user and about the usability of the resulting product. For this reason, the following techniques should be applied at elaboration time, even though they can be applied later as well in order to refine the products:

- Essential Use Cases: It is not necessary to describe all the identified use cases when there are a lot. It suffices to detail the main ones to assure that elaboration is not too time consuming.
- Structured User Role Model
- Operational Modelling
- Cognitive and Pluralistic Walkthrough: Walkthroughs evaluate an interaction dialogue, so as soon as these dialogues are defined in the essential use cases, walkthroughs can be applied as an evaluation technique.

The specifications document should include Usability Specifications, so this technique will be applied at elaboration time if the document is created at this stage (it usually is).

There are other techniques that, even though they could fit in well at elaboration time, require a greater effort than the ones detailed above. So they should be applied at elaboration time only in projects with characteristics that will have a less iterative component and can, therefore, afford a bulkier elaboration phase. These techniques are Cognitive Task Analysis, which requires a detailed description of means for performing an operation, and prototyping techniques that demand some implementation, such as Wizard of Oz Prototypes and Requirements Animation.

I.5.1.2 Techniques for Iterative Cycles (*i*)

There are some techniques that can be applied any time during the iterative cycles, and these are:

- Impact Analysis
- Detailed Use Cases
- GOMS
- Screen Pictures
- Menu-selection and Dialog Box Trees
- Context Navigation Maps
- Help Design by Use Cases

Some techniques are adequate for application at the end of a development cycle, that is, in the final moments:

- Heuristic Evaluation

- Inspections: Consistency, conformance and collaborative usability inspections.
- Thinking Aloud: Constructive interaction, retrospective testing, critical incident taking, and coaching method.
- Performance Measurement
- Questionnaires
- Laboratory Usability Testing

I.5.1.3 Techniques for the Evolution Stage

This moment in the development time groups the activities performed after the system has reached initial operational capability in the customer organization. The usability techniques to be applied at this time are techniques to evaluate the usability of an installed system. They are as follows:

- Direct Observation
- Video Recording
- Audio Protocol
- Questionnaires
- Structured and Flexible Interviews
- Focus Groups
- Logging Actual Use: Time-stamped keypresses and interaction logging.
- User Feedback: Online or telephone consultants**,** online suggestion box or trouble reporting, online bulletin board or newsgroup, user newsletters and conferences.
- Surveys

I.5.2 Time Constraints for Usability Activities

As detailed in the previous section on techniques, we will assign development time constraints to usability activities to define the stage in which each activity is applied.

Table I.5.1 shows the significance of each usability activity at the different stages during development time. We have made a distinction between stages where a usability activity is *essential*, that is, is part of the core activities that define the kind of thing to be done at that time; and *non-essential*, that is, the activity makes some contribution at that stage in development time, but it is not a basic activity to perform. Significance is a qualitative value, expressing that the activity is part of what are considered core activities at the development stage in question (essential), or is an activity with some importance at that stage, but plays a secondary role (non-essential). Please note that non-essential does not necessarily mean optional or not important, it is just that it is not an activity that is usually identified with the stage in question, like, for example, Expert Evaluation at Elaboration, which is very important but is not part of the core activities at that stage. Both significance values, essential and non-essential, set out the presence of each usability activity in the total development process, but there may be cases where some activities are performed in other development stages apart from the ones detailed in the table.

USABILITY ACTIVITY \ STAGE IN DEVELOPMENT			ELABORATION	ITERATIVE CYCLES (*i*)		EVOLUTION CYCLES
				CENTRAL MOMENTS	FINAL MOMENTS	
REQUIREMENTS	Elicitation		*essential*	*non-essential*		
	Analysis	Develop Product Concept	*essential*	*non-essential*		
		Problem Understanding	*essential*	*non-essential*		
		Modelling the Context of Use	*non-essential*	*essential*		
	Specification		*essential*	*non-essential*		
	Validation		*essential*	*non-essential*		
DESIGN	Interaction Design	Detailed Interaction Design		*essential*		
		User Interface Design	*non-essential*	*essential*		
	Help Design			*essential*		
EVALUATION	Usability Evaluation	Expert Evaluation	*non-essential*		*essential*	
		Usability Testing			*essential*	
		Follow-up Studies of Installed Systems				*essential*

Table I.5.1 Significance of each Usability Activity at Development Time Stages

Figure I.5.2 shows another way of looking at the relationship between cycles and activities, it is a distribution of work across the different kind of activities, related to the time in the development process when each effort is performed. Each horizontal line represents a kind of activity, and the height of the red line indicates the amount of work of that kind to be done at that particular development stage. For example, elicitation is mostly performed in Elaboration cycles (with more emphasis on the early stages), while some elicitation activities are performed at the beginning of the central moments within the Iterative Cycles, and a small amount of work may be done in Evolution cycles. The X-axis represents time. Therefore, slopes in different lines denote a certain precedence between the different kinds of activities, like, for example, between the different Requirements activities within Iterative cycles: first, there is some elicitation, followed by some development of the product concept (overlapping with the previous task), and then some problem understanding activities, and so on. Note that the amount of work on each activity represented in Figure I.5.2 is approximate, it should not be taken literally. It represents a specific software development process that we have taken to illustrate general issues regarding development time.

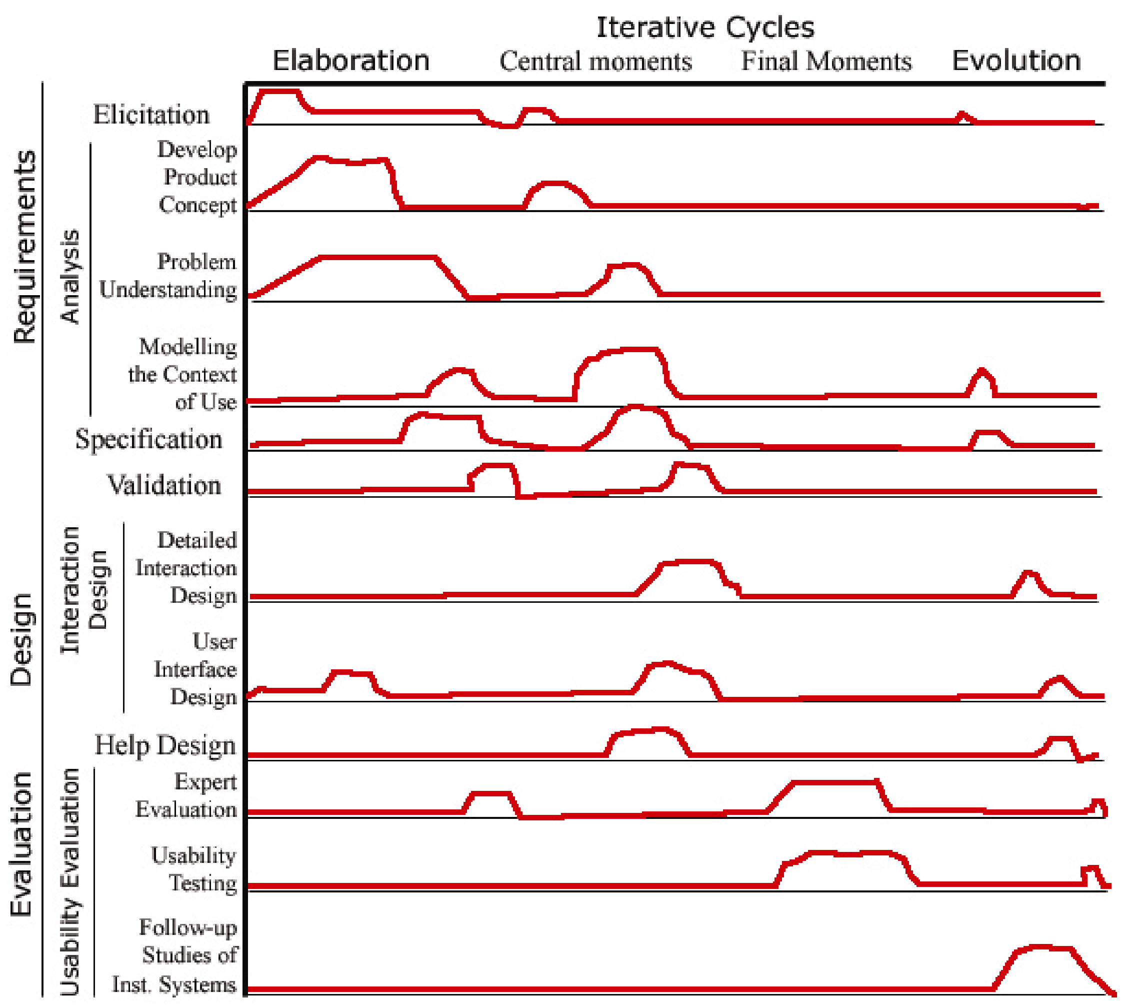

Figure I.5.2 Amount of Work on each Type of Activity at the Different Development Stages

I.5.2.1 Requirements Activities

Requirements activities are evidently performed at the beginning of the development process, because the problem and its subtleties need to be understood before any kind of design or implementation is conceived, to ensure that the solution is built upon the right premises. Therefore, requirements activities are the core activities at elaboration time, but they are also necessary in iterative cycles.

Requirements Elicitation and Develop Product Concept are essential at elaboration time, which is the stage in the development process that is mostly given over to clarifying the problem and to defining the lines along which the project will run. Elicitation will provide the information needed at elaboration time, and the product concept will have to be developed at elaboration time as a guide for subsequent activities in the development process.

Problem Understanding, Requirement Specification and Validation are essential activities to be performed at elaboration time, because this is when the specifications document is to be produced, since it is a core document for later software development tasks. Problem Understanding is a prerequisite for Requirement Specification, since a good understanding of the problem is a prerequisite for writing the right requirement specifications. Validation is performed on specifications, so ideally it should also be done at elaboration time, with the aim of having a validated specifications document by the end of elaboration. Then, other development activities performed after elaboration time can be based on the agreed specifications.

The activity of Modelling the Context of Use is part of the tasks to be performed at elaboration time, but it is not part of the core activities performed there. It is non-essential in the sense that it complements other essential activities that define with more emphasis what is supposed to be done at elaboration time. Modelling is an essential activity in the central moments within the iterative cycles, because the tasks and other details of the context of use must be dealt with in detail within each iterative cycle. As this modelling is performed, other questions may be raised concerning requirements issues, so some Requirements Elicitation, and some activities related to Developing the Product Concept and to Problem Understanding are also performed at central moments within iterative cycles. They complement the modelling mainly performed at this time, so they are non-essential activities at this stage in development.

I.5.2.2 Design Activities

Design is performed as a core activity within the iterative cycles, and it is performed not at the end of each cycle, where most design activities should have already been completed, but at the central moments. For this reason, the three kinds of design activities considered in this work (Detailed Interaction Design, User Interface Design and Help Design) are essential at central moments within iterative cycles.

The user interface is the part of the implementation that the user can understand better, so its design may be undertaken at the early stages of the development in order to get feedback from the user. For this reason, even if it is not part of the activities that define the elaboration stage in development time, User Interface Design is present at elaboration time as a non-essential activity.

I.5.2.3 Evaluation Activities

In order to evaluate a product, this product must have an evaluable form. As the development progresses, more and more tangible products are produced. So it is natural that evaluation activities are essential at stages that are either the final moments within an iterative cycle or final moments in the development time, that is, in the evolution cycles. Some early products can be also evaluated at elaboration time, but evaluation activities are not essential at this time. Some Expert Evaluation may be performed at elaboration time, where it is included as a non-essential activity. Usability Testing is a central activity at the final moments within each iterative cycle, because it provides a way to measure how far the product is from the specified usability levels. It gives the usability level that guides the next iterative cycle activities. Finally, Follow-up Studies of Installed Systems cannot be performed until the system is operational at the customer organization, so they are part of the activities performed in evolution cycles, and they are essential activities there, because they guide any usability improvements that may be developed at this stage in development time.

I.6 DEFINITION OF PROCESS INCREMENTS

We will class the usability activities and techniques to be applied in the development process as increments, called deltas, grouping techniques that are meant to be applied together according to the nature of the activities to which they belong (analysis, design or evaluation) and to the moment in development time when they can prove more effective for improving the usability of the software product.

We have defined eight deltas in order to get a better match with the general stages of an iterative software development process (as described above in section I.5.2):

- **E1**: Early Analysis
- **E2**: Usability Specifications
- **E3**: Early Usability Evaluation
- **E4**: Regular Analysis
- **E5**: Interaction Design
- **E6**: Architectural Design
- **E7**: Regular Usability Evaluation
- **E8**: Usability Evaluation of Installed Systems

Analysis activities, as seen in section I.2, are the activities that allow for a greater subdivision and call for careful integration with software engineering activities. Therefore, we have three deltas for analysis activities (E1, E2 and E4), plus delta E3, which, although formed by evaluation techniques, is applied in analysis activities. Traditional usability design activities are quite uniform and can be integrated in just one delta, E5, while the new usability activities and techniques defined as a result of WP3 will be grouped in delta E6. Usability evaluation activities, apart from the above-mentioned delta E3, have been divided into the activities to be performed during the main iterative cycles (delta E7), and the activities to be performed once we have an operational system working in the customer organization (delta E8). Figure I.6.1 shows how the deltas group similar kinds of activities to be performed close together in development time. Each triangle represents one of the deltas, and they are placed over the distribution of work represented in Figure I.5.2 above. The location along the X-axis represents the moment in development time where the delta should be applied, and the location on the Y-axis represents the kind of activities the delta groups. Note that the size of the deltas is not meaningful, as its only purpose is to cover the activities each increment contains. Delta E6 is currently empty and not located in the process, because we are waiting for WP3 to finish in order to incorporate its results. As the results of WP3 may also provide techniques to be applied in deltas other than E6, the other deltas include a reference to the possible techniques contributed by WP3.

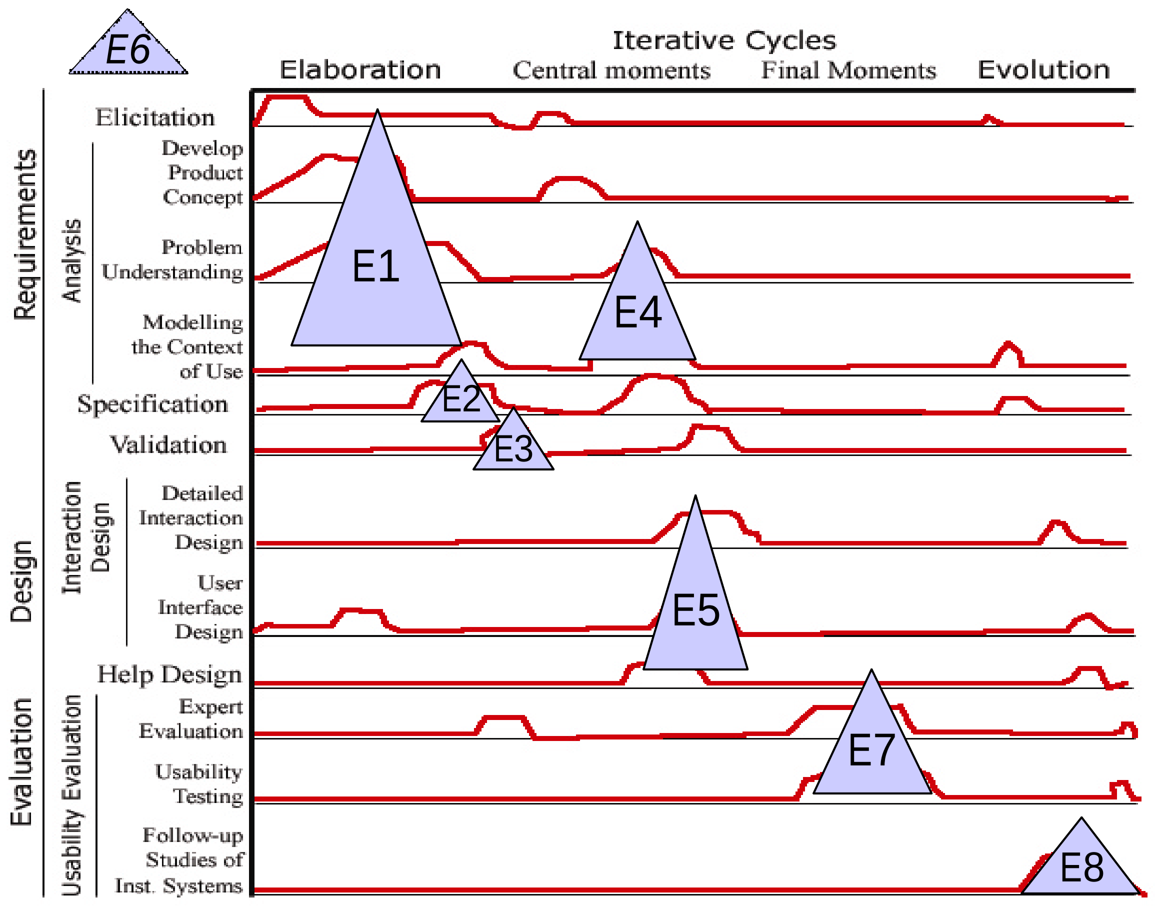

Figure I.6.1 Example of Location of Delta Increments in an Elaborate Software Development Process

So as not to repeat contents, the increments defined are explained in Part II of the document and not here, because they are an integral part of the documentation to be delivered to the developers. However, it should be mentioned here that each delta will be described according to the following structure:

- **Purpose**: The reasons why the delta should be added to an existing development process in order to improve the usability level of the resulting software product.
- **Phase**: Main type of activity: analysis / design / evaluation
- **Stage**: Development process stage where it is applicable.
- **Participants**: Members of the development team and other stakeholders who are meant to participate in the application of the techniques.
- **Activities/Techniques/Products** : List of the usability techniques that the delta groups, along with the documents or models produced by each technique. The techniques are grouped by the activity required to produce each product.

At the request of the industrial partners, we have defined an alternative delta grouping for software development organizations that apply a light iterative software development process. A light software development process means a process that is confined to dividing the development process into iterative cycles that are quite similar to each other. In such processes, the activities are divided into the four major stages of Analysis, Design, Implementation and Testing. There is no further subdivision of the different activities of which each stage is composed (for example, no distinction is made between the different subphases of Analysis). To make the deltas more manageable in this type of organisations, we decided to group the deltas defined above by the three major stages of Analysis, Design and Testing (there are no specific usability techniques for Implementation). As a result, we have the following deltas to add to a light software development process: Analysis (L1), Design (L2), and Evaluation (L3). The correspondence between the deltas defined for an elaborate software development process (E1-E8) and the deltas defined for a light development process (L1-L3) is shown in Figure I.6.2.

The deltas described for the light development process have the same structure as defined above, except for the Stage field, which is not necessary, as it is considered that no distinction is made between the different development cycles in a development process of this type.

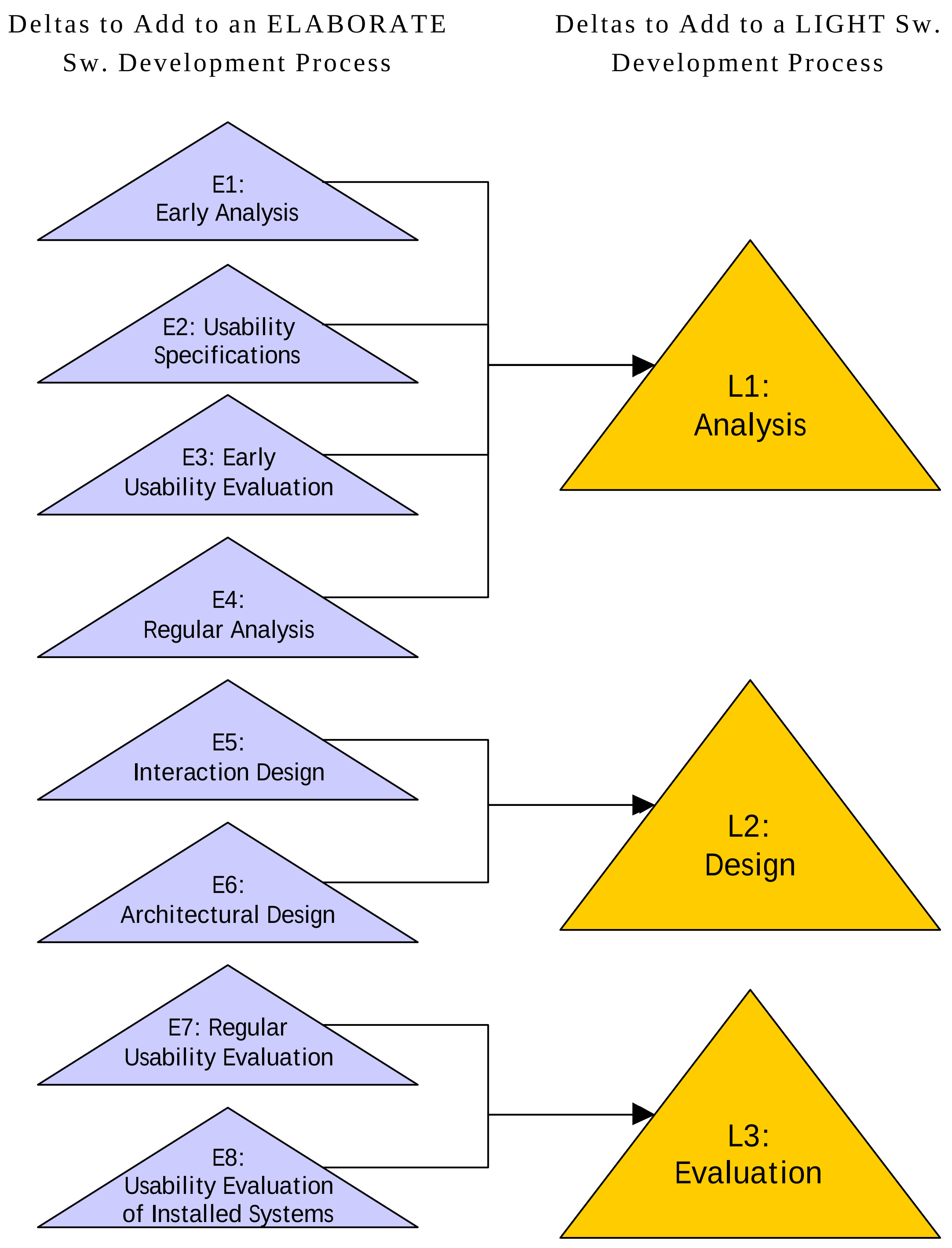
Deltas to Add to an ELABORATE Sw. Development Process
Deltas to Add to a LIGHT Sw. Development Process
E1: Early Analysis
E2: Usability Specifications
E3: Early Usability Evaluation
E4: Regular Analysis
E5: Interaction Design
E6: Architectural Design
E7: Regular Usability Evaluation
E8: Usability Evaluation of Installed Systems
L1: Analysis
L2: Design
L3: Evaluation

Figure I.6.2 Correspondence between deltas for an elaborate and for a light development process

I.7 REFERENCES FOR PART I

[IEEE, 01] IEEE Software Engineering Coordinating Committee. Guide to the Software Engineering Body of Knowledge - Trial Version 1.00. IEEE, May 2001.

[Constantine, 99] L. L. Constantine, L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design*. Addison-Wesley, New York, NY, 1999.

[Jacobson, 99] I. Jacobson, G. Booch, J. Rumbaugh. *The Unified Software Development Process*. Addison Wesley, 1999.

[Preece, 94] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, T. Carey. *Human-Computer Interaction*. Addison Wesley, 1994.

Part II. DOCUMENTATION FOR DEVELOPERS

II.1 CONDITIONS FOR IMPROVING YOUR DEVELOPMENT PROCESS WITH USABILITY ASPECTS

Usability is a difficult attribute to tackle. It is sometimes identified with common sense, and, therefore, every developer thinks that he or she knows enough to be able to produce usable products. However, usability is directly related to the humans who are to operate the software product and it is no easy endeavour to cater for humans. The usability techniques grouped in process increments that are detailed in this document may be of help for achieving the objective of a product with a sufficient usability level, but there are also some important requirements concerning the way software development is approached in the software development organization. An approach where user requirements are frozen at the beginning of development is not suitable for dealing with the complexity of human-computer interaction. The intricacy of the human side in such interaction makes it almost impossible to create a correct design at the first go. Cognitive, sociological, educational, physical and emotional issues may play an important role in any user-system interaction. Interaction design must, therefore, be tested and refined all through the development process in order to obtain a satisfactory result from the point of view of usability.

Therefore, iterative development is a must. Since the usability level of the system cannot be predicted in advance, some kind of usability evaluation is needed at the end of every iterative cycle. The requirement of an iterative process is closely linked to the need to perform quality measures at the end of each cycle. Therefore, if you want to incorporate increments into your development process, you must be developing software with an iterative process, whatever the particular process may be. All software development organizations have their own development processes, and each one probably uses a different terminology. We have established the requirement that the development process must be based on iterative refinement, and this is the issue that establishes the minimum common ground upon which we define a general schema of an iterative process. However, an iterative development process can range from very elementary to very sophisticated. A basic or light iterative process means a process in which all the cycles are symmetric and composed of the substages of Analysis, Design, Implementation and Evaluation, with no further subdivision of these stages. On the other hand, you may be using an elaborate development process, with different cycle types, as are used, for example, in the Unified Process [Jacobson, 99]. Figure II.1.1 shows one view of an elaborate iterative process. Prior to the iterative cycles there is an initial exploration stage, which we have called Elaboration. Afterwards, within the iterative cycles, we make a distinction between the main part of each cycle (Central moments) and the last part of each cycle (Final moments), where certain activities are performed, typically evaluation activities. Finally, when we have a system that could be installed and operated at the customer's site, the cycles are called Evolution.

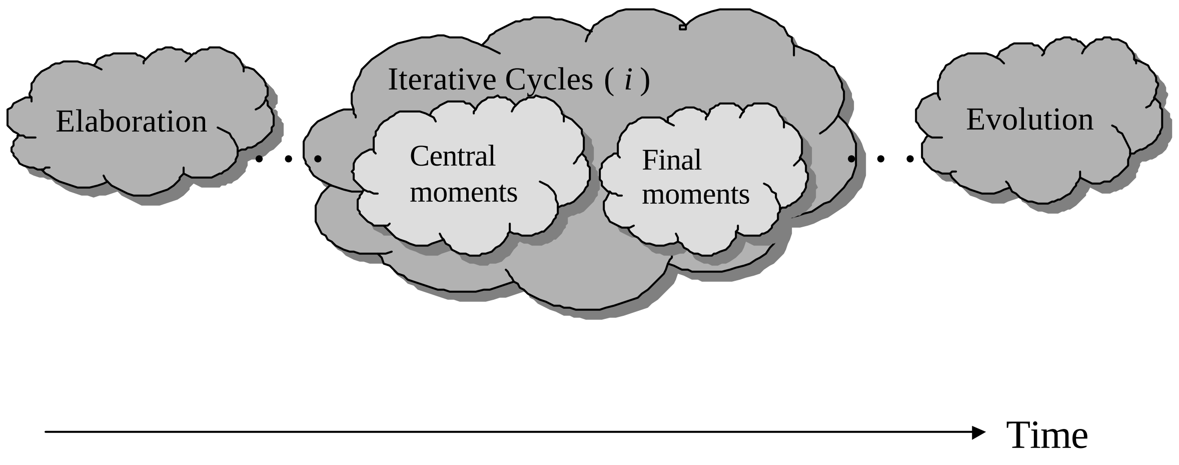

Figure II.1.1 Stages in the Development Process

An explanation of each moment represented in Figure II.1.1 follows:

- **Elaboration**: Before the actual iterative cycles begin, there must be an initial effort where the needs are identified and the general scheme that the system will follow is established. The products of this stage should be quite stable, even if they are open to changes in the iterative development cycles. Elaboration may consist of several cycles if necessary.
- **Iterative cycles - Central moments**: From the beginning of each iterative cycle until the moment where some testable result is produced.
- **Iterative cycles - Final moments**: The last stage within each iterative cycle, where the activities to be applied are mainly given over to the evaluation of the results produced in the cycle and the subsequent rework.
- **Evolution**: The stages in the development process that occur once the product has reached initial operational capability in the customer organization.

In order to incorporate the proposed increments into your elaborate iterative process, you should follow the indications on the development time constraints that are included in each delta. There we use the terminology of Figure II.1.1, therefore, you need to adapt these generic stages to the specific phases defined in your organization's development process.

Depending on the type of development process in place at your organisation, the usability techniques have been packaged in two different versions, one with deltas to be included in light processes (section II.2.1) and another with deltas to be included in elaborate processes (section II.2.2).

II.2 INCREMENTS FOR YOUR DEVELOPMENT PROCESS

A delta groups usability techniques that are meant to be applied together depending on the nature of the activities to which they belong (analysis, design or evaluation) and to the moment in development time as described in the previous section.

The following fields describe each delta:

- **Purpose**: The reasons why the delta should be added to an existing development process in order to improve the usability level of the resulting software product.
- **Phase**: Main type of activity: analysis / design / evaluation
- **Stage**: Development process stage where it is applicable (only for deltas for integration in an elaborate development process): elaboration, iterative cycles (central moments, final moments), and evolution.
- **Participants**: Members of the development team and other stakeholders who are meant to participate in the application of the techniques: customers, users, usability specialists, and developers.
- **Activities/Techniques/Products** : List of the usability techniques that the delta groups, along with the documents or models produced by each technique. The techniques are grouped the activity required to produce each product.

Usability techniques are explained in section II.3. Nevertheless, we advise you to attend a training course, like the one described in Part III of this document, in order to develop the necessary usability skills, adopt the right focus, and train in the usage of the deltas and usability techniques.

We will deal first with the deltas to be added to a light software development process, and then with the set of deltas to be applied to an elaborate software development process.

II.2.1 Do You Have a Light Software Development Process?

The basic condition to be able to apply the deltas with usability techniques is that the development process is iterative. However, an iterative process can take many forms, and we have opted to select two representative process models. The one we deal with in this section is what we have termed light software development process. First, we are going to define what light development process means and then we will describe the three deltas that would have to be added to a development process of this type.

II.2.1.1 What Does Light Development Process Mean?

When a software development organisation uses a basic iterative development process, this is normally confined to dividing the development into iterative cycles that are fairly similar to each other. We call this type of software development processes light development processes, where the activities are divided into the four major stages of Analysis, Design, Implementation and Testing. There is no further subdivision of the different activities of which these stages are composed (for example, no distinction is made between the different stages of Analysis). To fit the usability techniques to this type of cycle, they have been grouped by three major deltas: Analysis (L1), Design (L2) and Evaluation (L3). No usability technique is proposed for implementation and, therefore, there is no delta for the Implementation stage.

Even if the development process of your organisation is more elaborate than the light development process described, you should apply these deltas if your development process is closer to this schema than to an elaborate process, such as the one described in section II.2.2. Within the three deltas described for a light process, the techniques have been grouped according to the type of activity in which they fit, and this information can be used to adapt the use of these techniques to your particular development process.

II.2.1.2 Delta L1. Analysis

PURPOSE	Usability techniques can give the existing tasks of requirements elicitation, analysis, specification and validation the user-centred flavour that ensures that usability is sufficiently catered for in later development activities. Some techniques may help to model the problem from a user perspective, trying to understand the user goals and how he or she will operate the system in order to accomplish these goals.
PHASE	Analysis
PARTICIPANTS	Customer, users (specifically for JEM, but they can participate in the rest of techniques), usability specialists (for Usability Specifications)

ACTIVITIES	**TECHNIQUES**	**PRODUCTS**
ELICITATION	Ethnographic Observation	*-Structured User Role Model* *-Operational Model* *-Use Case Diagram*
	Contextual Inquiry	
REQ. ANALYSIS – MODELLING THE CONTEXT OF USE	Structured User Role Model	*-Structured User Role Model*
	JEM	*-Structured User Role Model* *-Essential Use Cases* *-Use Case Diagram*
	Operational Modelling	*-Operational Model*
REQ. ANALYSIS – DEVELOP PRODUCT CONCEPT	Post-It Notes	*-Product Concept*
	Visual Brainstorming	
	Competitive Analysis	*-Product Concept* *-List of needs and key/differentiating features*
	Scenarios	*-Scenarios*
REQ. ANALYSIS – PROBLEM UNDERSTANDING	Essential Use Cases	*-Essential Use Cases*
	Prototypes (paper and chauffeured)	*-Paper prototype*
REQUIREMENT SPECIFICATION	Usability Specifications	*-Usability Specifications*
REQS. VALIDATION – MODEL VALIDATION	Cognitive Walkthrough	*-Prioritised usability problems*
	Pluralistic Walkthrough	
REQ. ANALYSIS – MODELLING THE CONTEXT OF USE	Detailed Use Cases	*-Use Case Description* *-Use Case Diagram*
REQ. ANALYSIS – PROBLEM UNDERSTANDING	GOMS	*-GOMS model*
	Wizard of Oz Prototypes	*-Prototype*
	Requirements Animation	
	Techniques from WP3	

II.2.1.3 Delta L2. Design

PURPOSE	To design the interaction between the system and the user(s), employing user-centred techniques, and to build an architectural design that considers usability.
PHASE	Design
PARTICIPANTS	Users as part of the design team, developers

ACTIVITIES	**TECHNIQUES**	**PRODUCTS**
DESIGN	Impact Analysis	-*Prioritised redesign decisions*
USER INTERFACE DESIGN	Screen pictures	-*Specification of the graphical user interface elements*
	Menu-selection and Dialog Box Trees	-*Menu tree* -*Dialog-box tree*
	Context Navigation Maps	-*Context navigation map*
HELP DESIGN	Help Design	-*Structure of the help facility*
	Techniques obtained from WP3	

II.2.1.4 Delta L3. Evaluation

PURPOSE	To evaluate the usability of the subsystem being developed at each iterative cycle, applying the typical usability evaluation techniques found in the literature.
PHASE	Evaluation
PARTICIPANTS	Users, developers

ACTIVITIES	**TECHNIQUES**	**PRODUCTS**
EXPERT EVALUATION	Heuristic Evaluation	*- Prioritised usability problems found in interaction design (For consistency and conformance inspections, the problems are related to the issue being evaluated only)*
	Usability Inspections	
USABILITY TESTING	Thinking aloud	*-Usability problems identified in the system, along with the possible causes based on the user's way of reasoning and on user goals*
	Performance Measurement	*-Values for the usability attributes detailed as benchmark tasks in usability specifications*
	Laboratory Usability Testing (optional[1])	*-Values for the usability attributes detailed as benchmark tasks in usability specifications, plus multimedia material and interaction logging data for further analysis*
	Post-Test Feedback / User Questionnaires	*-Values for subjective usability attributes detailed in usability specifications as user questionnaires*
FOLLOW-UP STUDIES - QUESTIONNAIRES, INTERVIEWS AND SURVEYS	Questionnaires	*-The user's subjective opinion captured in the responses*
	Structured and Flexible Interviews	
	Surveys	
FOLLOW-UP STUDIES – FIELD USABILITY TESTING	Direct Observation	*-List of usability problems with an indication of the real conditions when the incident happened*
	Video/audio recording	*-Multimedia material to show to developers so they can better understand user needs and frustrations, plus a more detailed list of usability problems along with user comments and the surrounding conditions that occurred when the incident happened*
FOLLOW-UP STUDIES – FOCUS GROUPS	Focus Groups	*-User feedback on the problems of the software product*
FOLLOW-UP STUDIES - AUTOMATIC LOGGING	Logging Actual Use	*-Pointers to possible usability problems when the actual interaction varies from the expected one according to task and interaction models*
FOLLOW-UP STUDIES – USER FEEDBACK	Online User Feedback Facilities	*-Suggestions for change or improvement*
	Techniques obtained from WP3	

[1] Laboratory Usability Testing only to be applied if the resources needed to set up a usability laboratory are available.

II.2.2 Do You Have an Elaborate Software Development Process?

As in the above section, we will first describe what an elaborate software process means and then detail the deltas that group the usability techniques that a development process of this type should incorporate.

II.2.2.1 What Does Elaborate Development Process Mean?

An elaborate software development process is one that takes into account the possibility of having different kinds of cycles, depending on the problem being solved and on whether development is at the early or advanced stage. These elaborate development processes have a series of common guidelines that we will use to describe where the deltas specifying the usability techniques should be fitted in.

The basic schema of a process of this type was described, in terms of cycles in development time, in section II.1, and, as shown in Figure II.1.1, we have considered the stages of Elaboration, Iterative Cycles (central moments and final moments), and Evolution. As regards the different types of activity carried out in each cycle, Figure II.2.1 shows an example of an application of this type of development process. Each horizontal line represents a kind of activity, and the height of the red line indicates the amount of work of that kind to be done at that particular development stage. For example, elicitation is mostly performed in Elaboration cycles (with more emphasis on the early stages), where some elicitation activities are performed at the beginning of the central moments within the Iterative Cycles, and a small amount of work may be done in Evolution cycles. The X-axis represents time. Therefore, slopes in different lines denote a certain precedence between the different kinds of activities, like, for example, between the different Requirements activities within Iterative cycles: first, there is some elicitation, followed by some development of the product concept (overlapping with the previous task), and then some problem understanding activities, and so on. Note that the amount of work on each activity represented in process Figure II.2.1 is approximate, it should not be taken literally.

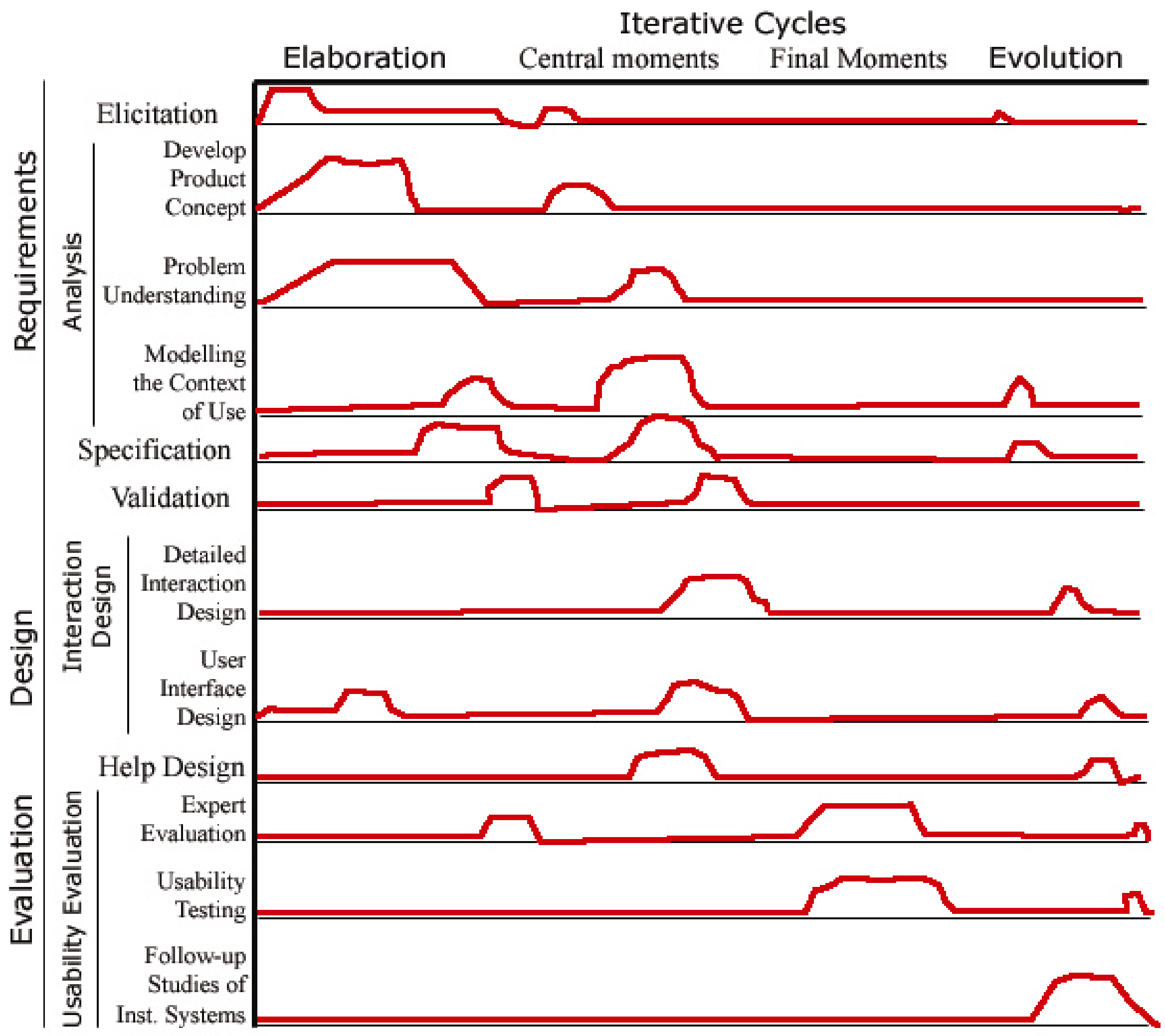

Figure II.2.1 Example of the Different Kinds of Activities Performed at each Development Stage

In this kind of software development process, not all cycles are the same, but they usually resemble the particular process shown in Figure II.2.1, and the general structure of cycles according to the development time is as described in section II.1 above. If you have a process of this kind you should map it to the general terminology we have described, so that the indications given in each delta about the phase and stage help you to position the delta in your development process framework.

The eight deltas of usability techniques to be included in an elaborate software development process are better tuned to an elaborate process than the ones described in the previous section, since they have additional information regarding the stage where they are to be applied. Therefore, they have an additional field, stage, that states the development stage in which they are applicable.

Figure II.2.2 shows how the deltas fit in a software development process like the one described in Figure II.2.1. Each triangle represents one of the deltas. The location along the X-axis represents the development stage where it should be applied, and the location on the Y-axis represents the kind of activities addressed by its usability techniques. Note that the size of deltas is not meaningful. Delta E6 is not located in the process, because it has not yet been defined (it will incorporate the findings of WP3).

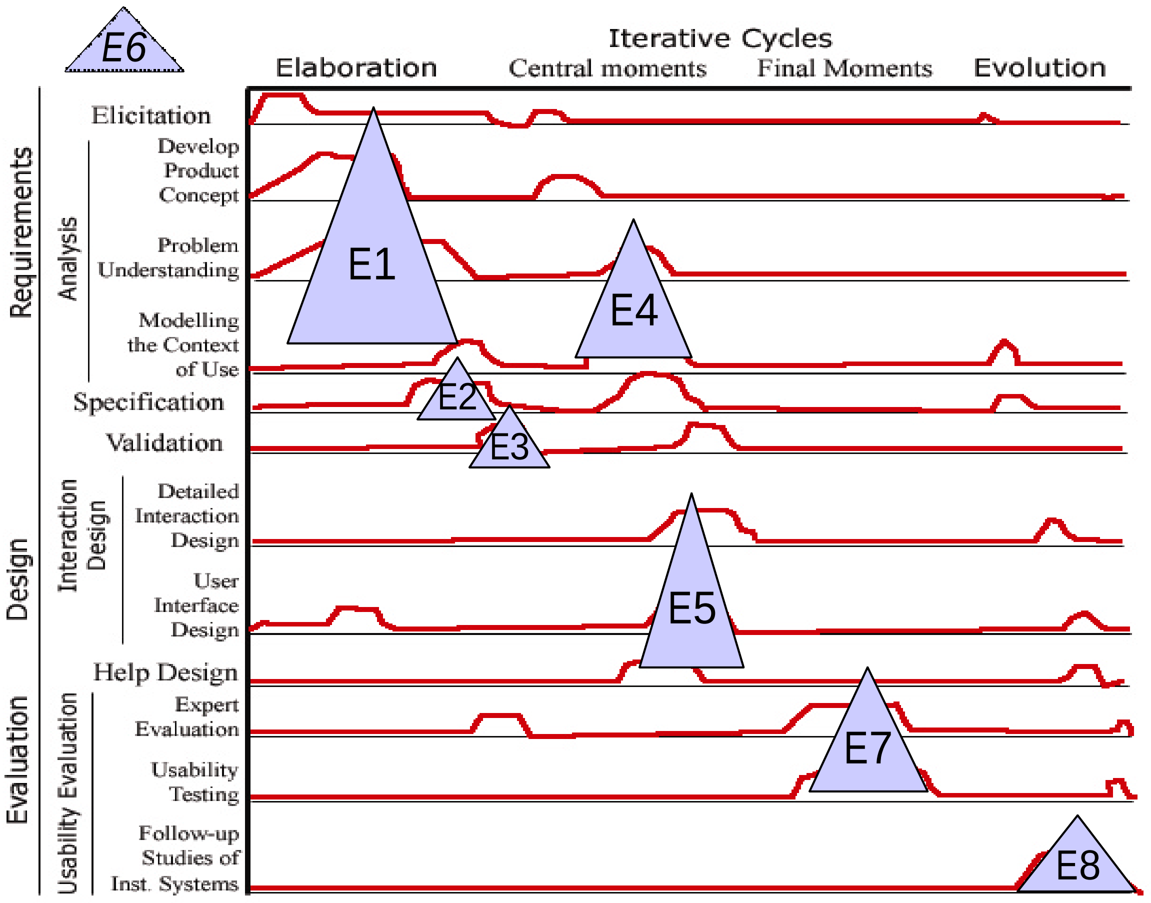

Figure II.2.2 Example of Location of Delta Increments in an Elaborate Software Development Process

II.2.2.2 Delta E1: Early Analysis

PURPOSE	Usability offers several techniques for analysis at the early stages of the project. These activities can give the tasks of requirements elicitation and analysis the user-centred flavour that ensures that usability is sufficiently catered for in later development activities.
PHASE	Analysis
STAGE	Elaboration
PARTICIPANTS	Customer, users, developers

ACTIVITIES	**TECHNIQUES**	**PRODUCTS**
ELICITATION	Ethnographic Observation	*-Structured User Role Model* *-Operational Model* *-Use Case Diagram*
	Contextual Inquiry	
REQ. ANALYSIS - MODELLING THE CONTEXT OF USE	Structured User Role Model	*-Structured User Role Model*
	JEM	*-Structured User Role Model* *-Essential Use Cases* *-Use Case Diagram*
	Operational Modelling	*-Operational Model*
REQ. ANALYSIS – DEVELOP PRODUCT CONCEPT	Post-It Notes	*-Product Concept*
	Visual Brainstorming	
	Competitive Analysis	*-Product Concept* *-List of needs and key/differentiating features*
	Scenarios	*-Scenarios*
REQ. ANALYSIS – PROBLEM UNDERSTANDING	Essential Use Cases	*-Essential Use Cases*
REQ. ANALYSIS – PROBLEM UNDERSTANDING	Prototypes (paper and chauffeured)	*-Paper prototype*
	Techniques from WP3	

II.2.2.3 Delta E2: Usability Specifications

PURPOSE	To define the usability goals that the future system will have to meet.		
PHASE	Analysis		
STAGE	Elaboration		
PARTICIPANTS	Customers, developers, usability specialists		

ACTIVITIES	**TECHNIQUES**	**PRODUCTS**
REQUIREMENT SPECIFICATION	Usability Specifications	-*Usability Specifications*
	Techniques from WP3	

II.2.2.4 Delta E3: Early Usability Evaluation

PURPOSE	Techniques to evaluate the products created at elaboration time from a usability point of view.
PHASE	Evaluation
STAGE	Elaboration
PARTICIPANTS	Representative users (for Pluralistic Walkthroughs), developers

ACTIVITIES	**TECHNIQUES**	**PRODUCTS**
REQS. VALIDATION – MODEL VALIDATION	Cognitive Walkthrough	- *Prioritised usability problems*
	Pluralistic Walkthrough	
	Techniques from WP3	

II.2.2.5 Delta E4: Regular Analysis

PURPOSE	To model the problem from a user perspective, trying to understand the user's goals and how he or she will operate the system in order to accomplish those goals.
PHASE	Analysis
STAGE	Iterative cycles – central moments
PARTICIPANTS	Users as part of the development team, developers

ACTIVITIES	**TECHNIQUES**	**PRODUCTS**
REQ. ANALYSIS – MODELLING THE CONTEXT OF USE	Detailed Use Cases	*-Use case description* *-Use case diagram*
REQ. ANALYSIS – PROBLEM UNDERSTANDING	GOMS	*-GOMS model*
	Wizard of Oz Prototypes	*-Prototype*
	Requirements Animation	
	Techniques from WP3	

II.2.2.6 Delta E5: Interaction Design

PURPOSE	To design the interaction between the system and the user(s), employing user-centred techniques.
PHASE	Design
STAGE	Iterative cycles - central moments
PARTICIPANTS	Users as part of the design team, developers

ACTIVITIES	**TECHNIQUES**	**PRODUCTS**
DESIGN	Impact Analysis	-*Prioritised redesign decisions*
USER INTERFACE DESIGN	Screen pictures	-*Specification of the graphical user interface elements*
	Menu-selection and Dialog Box Trees	-*Menu tree* -*Dialog-box tree*
	Context Navigation Maps	-*Context navigation map*
HELP DESIGN	Help Design	-*Structure of the help facility*
	Techniques from WP3	

II.2.2.7 Delta E6: Architectural Design

This delta will be defined to group the results of WP 3 regarding the architectural design process.

PURPOSE	
PHASE	
STAGE	
PARTICIPANTS	
TECHNIQUES	

ACTIVITIES	**TECHNIQUES**	**PRODUCTS**

II.2.2.8 Delta E7: Regular Usability Evaluation

PURPOSE	To evaluate the usability of the subsystem being developed at each iterative cycle, applying the typical usability evaluation techniques found in the literature.
PHASE	Evaluation
STAGE	Iterative cycles - final moments
PARTICIPANTS	Users, developers

ACTIVITIES	**TECHNIQUES**	**PRODUCTS**
EXPERT EVALUATION	Heuristic Evaluation	*- Prioritised usability problems found in interaction design (For consistency and conformance inspections, the problems are related to the issue being evaluated only)*
	Usability Inspections	
USABILITY TESTING	Thinking aloud	*-Usability problems identified in the system, along with the possible causes based on the user's way of reasoning and on user goals*
	Performance Measurement	*-Values for the usability attributes detailed as benchmark tasks in usability specifications*
	Laboratory Usability Testing (optional[2])	*-Values for the usability attributes detailed as benchmark tasks in usability specifications, plus multimedia material and interaction logging data for further analysis*
	Post-Test Feedback / User Questionnaires	*-Values for subjective usability attributes detailed in usability specifications as user questionnaires*
	Techniques from WP3	

[2] Laboratory Usability Testing only to be applied if the resources needed to establish a usability laboratory are available

II.2.2.9 Delta E8: Usability Evaluation of Installed Systems

PURPOSE	To obtain information about the usage of a system which can be already operated at the user organization, which is relevant from a usability point of view.
PHASE	Evaluation
STAGE	Evolution
PARTICIPANTS	Users that are already using some version of the product, developers

ACTIVITIES	**TECHNIQUES**	**PRODUCTS**
QUESTIONNAIRES, INTERVIEWS AND SURVEYS	Questionnaires	*-The user’s subjective opinion captured in the responses*
	Structured and Flexible Interviews	
	Surveys	
FIELD USABILITY TESTING	Direct Observation	*-List of usability problems with an indication of the real conditions when the incident happened*
	Video/audio recording	*-Multimedia material to show to developers so they can better understand user needs and frustrations, plus a more detailed list of usability problems along with user comments and the environmental conditions that occurred when the incident happened*
FOCUS GROUPS	Focus Groups	*-User feedback on the problems of the software product*
AUTOMATIC LOGGING	Logging Actual Use	*-Pointers to possible usability problems when the actual interaction varies from the expected one according to task and interaction models*
FOLLOW-UP STUDIES – USER FEEDBACK	Online User Feedback Facilities	*-Suggestions for change or improvement*
	Techniques from WP3	

II.3 CATALOGUE OF USABILITY TECHNIQUES

This catalogue includes a description of each usability technique appearing in the deltas. The aim of this annex is to act as a reference manual for developers, so they can directly apply the usability techniques or follow the pointers to further information.

Each technique is described using the following fields:

- Description: A description of the objectives of the techniques, and the main concepts upon which the technique is based.
- How-to: Explanation of how to apply the technique.
- Participants: Stakeholders that can participate in the technique.
- Main reference (to find out more): The main reference the reader can consult if he or she wants to learn more about the issue.

II.3.1 Ethnographic Observation

Description

Ethnography is a traditional method in anthropology for studying a particular tribe or culture. In this context, the ethnographer acts as an uninformed outsider whose job is to understand as much as possible about the “natives” from their own point of view. The ethnographer participates, overtly or covertly, in people’s daily lives for an extended period of time. It gives the developer information about the context where the user performs his or her tasks, which would be very difficult to apprehend otherwise (for example, by means of interviews).

As ethnographers, developers gain insight into individual behaviour and the organizational context in early phases of system development. The difference between developers acting as ethnographers and anthropologists is that, apart from trying to understand the user, developers observe the usage of existing software products for the purpose of changing and improving those products. Additionally, the available time for ethnographic observation in software development is a lot less than the time anthropologists spend immersed in a culture.

How-to

An ethnographic observation comprises four phases: preparation, field study, analysis and reporting.

For study preparation, it is advisable to understand the user organization policies and work culture and to gain access and permission to observe or interview. A set of initial goals and questions should be prepared beforehand as well.

It is very important for the success of the observation to develop rapport with managers and users in the user organization and to make all the observation and/or interviews at the user’s workplace. The developer should be prepared to follow unplanned paths in the observation, since it is the user’s normal actions that should guide the field study. An understanding of the different views that different classes of users may have about the tasks at hand or the goal priority should be identified in the study. Considerable emphasis should be put on interpreting data in relation to the context.

After the information has been collected, it is analysed and interpreted. Information may include video, annotations in notebooks, snapshots and artefacts from the activities being observed. Analysing and interpreting the data can be very time consuming, especially if video is involved. Some tools may be used for this process.

Finally, the conclusions should be reported to the development team and optionally to the users that have been observed. It may be necessary to consider multiple audiences and goals.

Ethnographic observation can increase trustworthiness and credibility, since developers learn about the complexities of an organization firsthand by visits to the workplace. Personal presence allows them to develop working relationships with several end users to discuss ideas, and it can be a starting point for including users as active participants in later activities in the development process.

Participants

Developers and the user organization as a whole.

Main Reference

J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, T. Carey. *Human-Computer Interaction*. Addison Wesley, 1994. pp. 664-668.

II.3.2 Contextual Inquiry

Description

Contextual inquiry is based on performing elicitation at the customer organization, at the users' workplace, observing how they work and discussing how the work gets done with them. The work may be performed either manually, using a competitor product or using a previous version of the software product under development. Contextual inquiry may be used for usability evaluation purposes as well.

In a contextual inquiry interview, the developer sits beside the user and observes how he or she performs the work, interrupting him or her every now and then to ask why a particular action has been taken or what its purpose is. The interview should be performed in an environment that is as close as possible to the usual working environment of the user, because the developer is looking for first-hand knowledge, the kind of understanding about the work structure that the user cannot formulate, unless he or she is performing the work at that very time.

How-to

A contextual inquiry interview is not like a traditional interview, where the interviewer is in full control, deciding what and what not to ask. It is better to follow the master / apprentice model, where the user is the master, and the developer is the apprentice that wants to learn how the work is done. This model of action for the developer aims to make the user focus on his or her work. Nevertheless, the master / apprentice model should not be taken literally, since the developer must play an active role in probing the user every time he or she needs an explanation. The interview should be a combination of watching and probing. A sense of partnership should be formed between developer and user, in the sense that they are both looking to explain the internal logic behind the user's actions, as there can be a lot of tasks that the user does routinely and he or she cannot completely explain. Therefore, the developer must try to make out the work structure and find patterns and distinctions in the way people organize work. Not only the developer gains a better understanding of the user's work, the user himself or herself also acquires increased insight into his or her work by being forced to look at it from an external perspective. Users themselves are sometimes surprised about some of the actions they perform routinely when they look at them from an analytical point of view.

The contextual inquiry interview has the following steps:

1. Conventional interview: The developer asks the user about his or her work, and the kind of tasks he or she is going to perform that day.
2. The transition: The developer states the rules for the rest of the session. The developer watches the user, and interrupts him or her whenever the developer needs an explanation for some action. The user may ask the developer to hold off it is a bad time for being interrupted.
3. The proper contextual interview: The user starts doing his or her usual work tasks, and the developer observes and interprets.
4. The wrap-up: The developer summarizes what he or she has learned about the user's work, trying to focus on the broader context of the organization. This is the last chance for the user to correct and elaborate on the developer's understanding.

The information that is being gathered by the developer must be interpreted together with the user. The user is not a passive subject who answers when asked, the user actively participates in the process of finding the work structure and the motivations behind any of the his or her actions. The knowledge gathered about how the user performs his or her work should be the basis for establishing the requirements for the software product to be developed.

Participants

A developer and a user, working as a team for unveiling the user's work structure.

Main Reference

H. Beyer, K. Holtzblatt. *Contextual Design. Defining Customer-Centered Systems*. Morgan Kaufmann, 1998. Chapters 3 and 4.

II.3.3 Structured User Role Model

Description

A structured role model collects and organizes information about users in roles, guiding the derivation of essential use cases and highlighting operational context facets of user roles that are likely to be of significance for designing an effective interaction.

The information about users is organised as a series of collections called profiles, because they do not consist of a single factor but combine numerous factors and define a range or distribution of characteristics among users within a given role.

How-to

The structured user role model is formed by profiles. The main profiles are as follows:

- **Incumbents**: Common characteristics that users who play a given role share. There are three categories into which the elements in this profile may fall: domain knowledge, system knowledge and other background knowledge.
- **Proficiency**: How usage proficiency is distributed over time and among users in a given role.
- **Interaction**: Patterns of usage associated with a given role. The kind of information in this profile falls in one or more of the following categories: frequency, regularity, continuity, concentration, intensity, complexity, predictability or locus of control.
- **Information**: Nature of the information manipulated by users in a role or exchanged between users and the system. The information in this profile may offer details on the input origins, the flow direction, the information volume and/or the information complexity.
- **Usability criteria**: Relative importance of specific usability attributes with respect to a given role.
- **Functional support**: Specific functions, features, or facilities needed to support users in a given role.

Some roles are highlighted as focal roles, which are the ones that are judged to be the most common or typical or that are deemed particularly important from a business perspective or from the standpoint of risk.

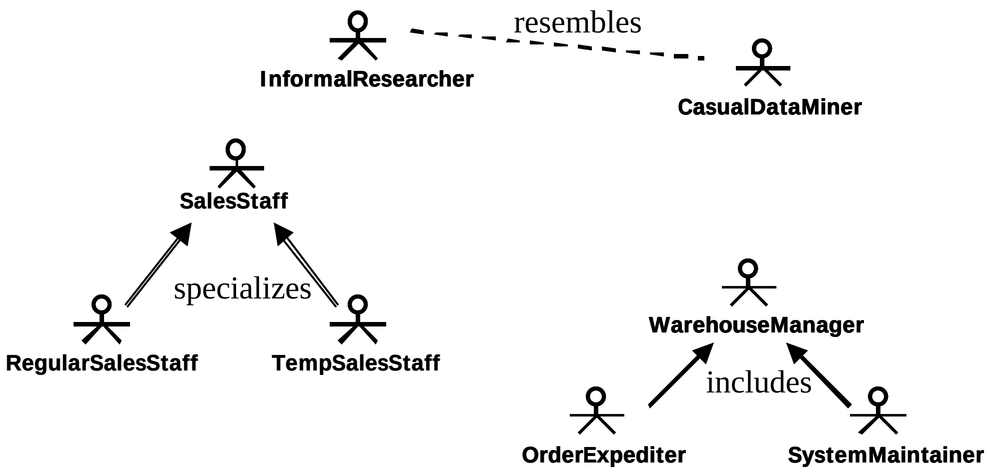

Figure II.3.1 Example or User Role Map

Any elicitation technique can be used to acquire the information about users to complete the profiles in the structured user role model. The more complex the user population for a system under development is, the more complete the profiles for a user role must be. For simple systems just some data on a few of the profiles may be enough to describe each user role, and then the user role model is not said to be structured.

Apart from the profiles that describe the details on each user role in an organised manner, a user role map like the one in Figure II.3.1 also represents the relationships between roles. Figure II.3.1 shows the user roles identified for a statistical analysis package. We say that there is affinity between two roles, which is represented by a dashed line, if we identify some similarity or resemblance between them (*InformalResearcher* and *CasualDataMiner* have an affinity relationship). When a user role is a subtype of another, we say that the former specializes the latter, and this is represented by a double-lined arrow that goes from the more specific role to the more abstract one (*RegularSalesStaff* and *TempSalesStaff* are both subtypes of the general *SalesStaff*). Finally, there is a composition relationship when one role combines the characteristics or features of two or more other roles and is composed of these other roles, which we represent by a single-lined arrow (*WarehouseManager* includes both the *OrderExpediter* and the *SystemMaintainer*).

Participants

Developers act as modellers, and other stakeholders act as sources of information for the model.

Main Reference

L. L. Constantine, L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design*. Addison-Wesley, New York, NY, 1999. Chapter 4. pp. 69-96.

II.3.4 JEM (Joint Essential Modelling)

Description

Joint Essential Modelling, or JEM, is a structured, facilitated, collaborative process for concurrent usage-centred modelling. It is based on JAD (Joint Application Design).

In JEM, users and developers join in a collaborative effort to define the essential models and reach agreement on core requirements. The objective is to reach consensus on the tasks to be supported by the system under development. Although other models play some part, the principal medium of exchange are use cases, whether essential or detailed. The primary deliverables from a JEM process are the structured role model, use cases with detailed narratives and focal use cases. Deliverables that are desirable but optional include essential use case narratives, the use case map, use case prioritisation and a glossary.

JEM is based on carefully delineated roles for participants and highly focused activities for the collaborative sessions.

How-to

The main roles in JEM are the users, which are the most important participants; the lead analyst, who is designated to assure appropriate technique leadership and expertise in modelling; the facilitator, who functions as a neutral process leader; and the scribe, who must track the full process of modelling and decision making on top of noting down the results. Other potential participants include a sponsor, who begins and ends, but does not actually conduct, modelling sessions; and other members of the development team who may contribute with technical knowledge to the modelling decisions.

The JEM process consists of five basic activities that are carried out in a series of one or more meetings: premodelling and consolidation, role modelling, task modelling, model auditing and feature allocation.

The preparation and consolidation process prepares materials and an agenda for the subsequent sessions and also generates a candidate list of user roles used as a guide for participation in later joint modelling. Following the development of role and task models, these models are audited for completeness, correctness, and consistency. To complete the process, use cases are prioritised and allocated to project iterations. These activities are carried out in a series of sessions as follows:

1. Framing session: The purpose of this session is to establish the framework within which the joint modelling sessions will operate. The deliverables of this session are a draft statement of essential purpose for system, a preliminary list of candidate user roles, and a list of participants for subsequent modelling sessions.
2. Modelling sessions: User role and use case models are developed collaboratively during these sessions. The deliverables from this phase include a final statement of essential purpose, a user role model, a user role map, a list of use cases with an identified essential purpose, use case narratives, a use case map (optional), and identification of focal use cases.
3. Review session: First the group reviews the models to ensure that they are complete, correct, and consistent. Second, use cases are sorted to identify which capabilities are to be supported and when.

Participants

Developers, users and other stakeholders.

Main Reference

L. L. Constantine, L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design*. Addison-Wesley, New York, NY, 1999. pp. 499-509.

II.3.5 Operational Modelling

Description

Operational modelling aims to model the environment in which the system to be developed will be used. Of all the aspects concerning the environment, the operational model represents only the ones that are most likely to affect the usage of the system. It is a collection of various operational and contextual influences that can play a role in usability. The information in the operational profiles complements and extends the user profiles in the structured user role model, and it can provide insight into the planned usage of the system that can prove very relevant for interaction design.

The operational profiles are the following ones:

- **Operational risk**: Type and level of risk associated with a given role or for a specific use case or set of use cases.
- **Device constraints**: Limitations or constraining characteristics of the physical equipment.
- **Environment**: Relevant factors of the physical environment.

How-to

Operational modelling involves filling in the operational profiles with the information elicited from the different sources of information. A description of the kind of information that each profile specifies follows.

Operational Risk Profile: Operational risk refers to what is at stake if the user and the system fail to correctly complete tasks. For example, what are the consequences of an input error, a failure to complete a transaction, a system lockup, or a delay in processing. Where operational risk is higher in connection with particular roles or use cases, special attention needs to be paid to mechanisms that assure input accuracy and accurate interpretation of output.

Device Constraints Profile: The device constraints profile identifies equipment characteristics associated with specific roles, use cases, or the system as a whole. There may be limitations on the input side, the output side, or both. These constraints include screen size, resolution, and colour depth; keyboard or keypad size and layout; and special controls such as sliders, toggle switches, rotary knobs, or similar items. It is especially interesting to describe the device constraints in projects where the devices are fixed by economics or the user community. We may also include as device constraints physical barriers or impediments between users and a system, like, for example, the heavy gloves that a factory worker may be wearing, which means that the worker cannot make use of any complicated keyboard shortcuts.

Environment Profile: The environment profile is formed of physical factors, such as the type of user location (office, home, factory, etc.), the level of ambient noise, the lighting conditions, temperature, humidity, or the presence of vibration. The information gathered may reflect any kind of physical condition of the environment that may affect system usage. A key issue here is the level of distraction due to the physical environment. Distractions may be physical (like a noisy fan or repeated phone calls) or mental (like trying to remember to do something that must be temporarily postponed).

Participants

Different stakeholders can provide the information we need to fill in the operational profiles. The users of the system under development and the customer are the main sources of information, but other sources such as trade unions, professional associations or security enforcing organizations may be valuable as well.

Main Reference

L. L. Constantine, L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design*. Addison-Wesley, New York, NY, 1999. pp. 308-313.

II.3.6 Post-It Notes

Description

Post-it notes may be used in a workshop-like meeting where several participants work together to identify, group and discuss different issues. They are particularly suited for identifying and analysing information. The main advantages of this technique are the flexibility allowed by the post-it, which can be placed in a given location and then moved to another category as the discussion goes on, and the visual effect of seeing all the issues at a glance in the post-it placeholder. It is particularly suited for mixed teams of developers and users, since no technical knowledge is required for its use, and it helps to build team consensus.

There are several variations of this technique. When used to classify all the information gathered from contextual inquiry sessions as notes, it is called an affinity diagram. Another variant consists of using post-it notes for deciding which tools and materials are needed in the different interaction contexts of a user interface.

How-to

For elicitation purposes, post-it notes can be used to represent information elicited from users and structure this information. The common structure is raised bottom-up from the pieces of information represented in each note. Categories are not predefined, but created and checked with the other participants as they appear. The notes are placed on a vertical surface that is accessible to all participants, so they can approach it and move whatever note they wish to. As each note is placed, other participants may add similar notes close to it. All participants should be able to contribute, it is no good if someone takes control of positioning and moving the notes, it should be a participative effort. When there is general agreement on the resulting structure, the issues discussed are broadly organized according to the user's logic and needs. Then the results are recorded for further use in the development process.

When this technique is used to model the abstract elements of the user interface, sheets of paper or areas of a whiteboard are set aside for each interaction space. The names for the interaction spaces should be informative of the operations they are for, general names like "main screen" or "work space" may be indicative of a lack of coherence or not enough deliberation. As the model is elaborated, new interaction spaces may have to be created or existing ones may be combined. Different coloured post-it notes can be used to colour-code the elements in the interface: for active controls vs. passive controls, for example. Each post-it note represents a necessary element in an interaction space. Using post-it notes, major reorganizations can be made easily and then be discussed by the development team. This ease encourages experimentation and exploration.

Participants

Developers and users. Other stakeholders that possess relevant information to be elicited may participate as well.

Main Reference

For affinity diagrams: H. Beyer, K. Holtzblatt. *Contextual Design. Defining Customer-Centered Systems*. Morgan Kaufmann, 1998. pp. 153-163.

For Post-It notes for user interface modelling: L. L. Constantine, L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design*. Addison-Wesley, New York, NY, 1999. pp. 127-133.

II.3.7 Visual Brainstorming

Description

With visual brainstorming we try to generate visual ideas of abstract concepts. Visual ideas are a basis for discussion in a brainstorming session.

How-to

Paper prototypes may be used for exploring all kinds of design ideas, and they can help the development team to think about the organizing metaphor for a system. Visual brainstorming involves using exploratory paper prototypes as a means for facilitating communication in brainstorming sessions.

One of the first things you learn in design is to put forward a number of alternatives so that you can then compare them. Having a lot of display space is important for doing this because you can then make design ideas visual. One of the things you can do with visual things is superimpose them, or put them side by side and quite often when you start doing that you like one better than another. Until you have made a comparison, you have no idea why you prefer one to another. The criteria emerge from the comparison. It is not just picking the right idea, but recognizing the right idea in all the mess of different alternatives produced.

Evaluation also comes into brainstorming: when you stop generating ideas you have to start evaluating them. The best exploratory designs produced in the visual brainstorming can then be further developed by constructing cardboard prototypes, which can be evaluated with users.

When trying to develop the product concept, a metaphor may need to be devised for the system to work upon. A metaphor is a way of describing a concept in a more accessible and familiar form. We can shape the system following a metaphor, like, for example, the desktop analogy used for Mac and Windows operating systems. Metaphors can be used to present a coherent image of the whole system (therefore, defining the product concept) or to deal with specific functions or parts of the system. Visual brainstorming can be used to explore any user interface-related design idea, but it is specially well-suited for developing metaphors.

Participants

Developers, users and relevant stakeholders for the definition of the product concept.

Main Reference

J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, T. Carey. *Human-Computer Interaction*. Addison Wesley, 1994. pp. 456-461.

II.3.8 Competitive Analysis

Description

If we view a competing product as a prototype, then we can evaluate it to see if it matches the objectives we have set for our system. In particular, we can analyse existing products heuristically, and we can perform usability tests with users. A competing product is already fully implemented and can, therefore, be tested very easily.

If several competing products are available for analysis, we can perform a comparative analysis of their differing approaches to support the user goals. This will provide ideas for the system we are developing, especially for developing the product concept. It can also provide a list of ad hoc guidelines for approaches to specific issues that seem to work, and things that should be avoided.

Competitive analysis basically involves studying existing products to find out their strengths and weaknesses. These analysed products may be competing products, but they may also be products from different fields that address issues that are similar to the ones the system will have to deal with. Commercial products that are widely known serve as good references for establishing the product concept, and a competitive analysis of their benefits from a usability point of view can help to focus the discussion and the decision-making process.

How-to

When competitive analysis is used for establishing planned levels in usability specifications, then usability tests are performed with the competing products in order to measure the values for the benchmark tasks.

When used for developing the product concept, competitive analysis involves performing a heuristic evaluation of existing software products to identify their main strengths and weaknesses, and to identify approaches that might be valuable for the user. These approaches or design ideas may be discussed to decide whether they are appropriate for the system we are going to develop.

Participants

Developers. Users may point out other products they identify as very usable and/or useful, as candidates for competitive analysis. Showing the approaches of existing products to users may help as well to focus the discussion, as it will be based on tangible products instead of abstractions.

Main Reference

J. Nielsen. *Usability Engineering*. AP Professional, 1993. pp. 78-79.

II.3.9 Scenarios

Description

A scenario is a personalized, fictional story with characters, events, products and environments. They help the development team to explore ideas and the ramifications of design decisions in particular situations. Scenarios are populated with fictional, but possible, characters who want to undertake real work. During design activities later in the development process, developers may refer to the scenario characters with comments such as "John would not understand that arrangement". Scenarios also provide a useful source of hypothetical cases for evaluation.

Scenarios are useful where there is no available data about the range and distribution of user task frequencies and sequences, especially for highly innovative systems. In these less well-defined projects, developers find day-in-the-life scenarios helpful to characterize what happens when users perform typical tasks. Scenarios can represent common or emergency situations, with both novice and expert users, and they are especially suited when multiple users must cooperate or multiple physical devices are used.

How-to

A scenario details an interaction example illustrating the flow of specific user actions needed to get some result, concentrating on what the user will see, what the user must know, and what the user can do. It is an encapsulated description of:

- an individual user
- using a specific set of computer facilities
- to achieve a specific outcome
- under specified circumstances
- over a certain time interval.

An example of scenario for the Eurochange system (a system for currency exchange) follows:

> Path Smith has just arrived at Geneva International Airport en route to a large conference on Human-Computer Interaction. Pat is carrying a laptop and a large, heavy suitcase and needs to get to the conference centre quickly. Looking around for a bank in order to get some local currency, Pat sees the Eurochange machine with its blue flag style logo showing a circle of twelve stars.
> Pat goes up the machine. It seems similar to the automatic teller machine that Pat uses regularly. Pat puts down the suitcase, takes out a credit card and inserts it into the slot. A message is displayed on the screen:
>
> *Enter your PIN*
>
> Pat thinks for a few moments and then types a four-digit number on the numerical pad, listening to the reassuring beep that follows each number pressed. The machine pauses for a few seconds and then displays:
> *Select currency required*
> Pat pauses again. What is the currency in Switzerland? Pat browses the currencies available, sees "Swiss Franc (CHF)" and presses the key. The machine displays the message:
>
> *Exchange rate is 1.47 CHF to 1 EUR*
> *Enter amount required in Swiss Francs in units of [10]*
> *Press <Proceed>*
>
> Pat types 253 and presses <Proceed>. A message is displayed:
>
> *Machine deals in bank notes only*
> *Smallest bank note is [10] CHF*
> *Enter new amount to obtain CHF or press <Cancel>*
>
> Pat enters 260 and presses <Proceed>. There is a whirring noise and a few other indeterminate clunks and clicks. The credit card is returned from the card entry slot and the money deposited in the delivery slot, with a printout of the transaction.

Storyboards (pictorial representations of scenarios, like the ones used by film directors) may provide additional support to the situations described in scenarios.

Scenarios may serve to convey a shared understanding of the product concept and the kind of users and tasks for which it is intended. It may be used as well to show to the customer what could be offered or provided if the system is actually developed.

Participants

Developers create scenarios, but they are validated with the customer and evaluated with users.

Main Reference

J. M. Carroll. "Scenario-Based Design", in *Handbook of Human-Computer Interaction. Second Edition*, edited by M. Helander, T. Landauer and P. Prabhu. North-Holland, 1997. Chapter 17. pp. 383-406.

II.3.10 Use Cases

Description

A use case is a case of use, or one kind of use to which a system can be put. It is:

- Supplied functionality
- An external, “black box” view
- A narrative description
- Interaction between a user (in some user role) and a system
- A use of a system that is complete and meaningful to the user

Each use case describes, in narrative form, an interaction that is complete, well defined, and meaningful to some users. The narrative of the use case is divided into two parts: the user action model, which shows the actions the user takes; and the system response model, which shows what the system does in response.

Depending on the level of abstraction at which the use case is described, there are two forms for use cases: essential and detailed or concrete. Essential use cases describe a generalised, abstract, technology-free and implementation-independent interaction, in the language of the application domain and of users. On the other hand, detailed use cases reflect the actual interaction as it happens between the user and the system, so they include restrictions imposed by internal design decisions and according to a particular user interface design. Detailed use cases can also be called concrete, since they reflect the concrete instantiation of the abstract description in its essential form.

The use case diagram or use case map represents the use cases supported by the whole system, and the interrelationships among them and with the users. The use case map along with the narratives of use cases form the use case model.

How-to

Detailed use cases are the ones employed in object-oriented software development. The usage of these conventional use cases has presented some major problems and limitations from a usability point of view. Conventional use cases are usually employed early in the development process as a starting point for development. Nevertheless, they contain too many built-in assumptions, often hidden or implicit, about the form of the user interface that is yet to be designed. As models, they lean too closely toward implementation and do not stick closely to the problems faced by users. But they are useful for linking an external view of the system with the design of its internal part. Therefore, detailed use cases are useful as a technique for improving the usability of the software product, but not as the first approach to describing the interaction between the users and the system. Below we will describe essential use cases as an appropriate initial approach to interaction modelling, which will serve as basis for designing the best scheme to support interaction (using some of the other usability techniques described in this catalogue).

An essential use case is a structured narrative, expressed in the language of the application domain and of users, comprising a simplified, generalised, abstract, technology-free and implementation-independent description of one task or interaction that is complete, meaningful and well defined from the point of view of users in some role or roles in relation to a system and that embodies the purpose or intentions underlying the interaction.

An essential use case is based on the purpose or intentions of a user rather than on the concrete steps or mechanisms by which that purpose or intention might be carried out.

The first step for the creation of the use case model is to identify the use cases that the system must support. The structured user role model is a starting point for use case identification. For each user role we can ask ourselves what these kinds of users are trying to accomplish, what they need to do in order to fulfil the role, or what capabilities are required to support whatever these users need to accomplish. Once the use cases have been identified, narratives are written for each use case, and the relationships between them are defined.

To write a use case narrative, we must identify the essential purpose or user intent embodied in the interaction. The name of the essential use case should be simple, it should imply purposeful, goal-directed action. Transitive gerunds, verbs of continuing action with a direct object, make good names for essential use cases. Examples of essential use case names are: *findingCustomer*, *verifyingOrder* or *insertingMathSymbol*. If the user purpose is not well expressed or fully implied by the name of the use case, then an explicit purpose clause should be added to the head of the narrative, describing and detailing the purpose or goal from the user perspective.

Figure II.3.2 shows the narrative for the use case *gettingCash* from an ATM (Automatic Teller Machine). The narrative for the essential use case is on the left-hand side of the figure, while the narrative for the detailed case is on the right-hand side.

ESSENTIAL USE CASE

gettingCash

USER INTENTION	SYSTEM RESPONSIBILITY
identify self	
	verify identity
	offer choices
choose	
	dispense cash
take cash	

DETAILED USE CASE

gettingCash

USER INTENTION	SYSTEM RESPONSIBILITY
insert card	
	read magnetic strip
	request PIN
enter PIN	
	verify PIN
	display transaction option menu
press key	
	display account menu
press key	
	prompt for amount
enter amount	
	display amount
press key	
	return card
take card	
	dispense cash
take cash	

Figure II.3.2 Example of Essential Use Case vs. Detailed Use Case Narrative

Participants

Developers to build the use case model. Users may participate in defining the use cases the system will support and the narrative for essential use cases.

Main Reference

L. L. Constantine, L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design*. Addison-Wesley, New York, NY, 1999. Chapter 5. pp. 97-123.

II.3.11 Paper and Chauffeured Prototypes

Description

Paper prototypes are simple drawings of a user interface. They are passive prototypes in the sense that they are not interactive like a real user interface. A variant of paper prototypes is the sketching technique, where initial sketches of the user interface are drawn in order to explore design alternatives for the product concept.

How-to

You do not need to be an extremely good drawer to produce a good paper prototype. Simple, quick pencil sketches often serve quite well, even if they only marginally resemble any actual software user interface. Accuracy and graphical detail are usually less important than the overall structure and visual organization of each interaction context.

Paper prototypes can be presented to users and members of the development team for comment and improvement. At the early stages of development, it may be undesirable to express design ideas by means of working software, since this may take a significant effort to build. Once there is a glossy version of an idea, it is too easy to get carried away thinking that this must be the only or the best solution to the problem.

A paper prototype does not represent animation or interaction, so it must be supplemented with explanations from the development team. Pull-down menus may be simulated by means of Post-It notes or tape, as in the prototype situated on the left-hand side of Figure II.3.3.

The main advantage of paper prototypes is their flexibility, since changes are very easy to make to a drawing, or a new one can be quickly sketched. The main disadvantage is that sketches are sometimes not very close to a real software system, while prototyping is supposed to produce a product that is as similar to the intended system as possible.

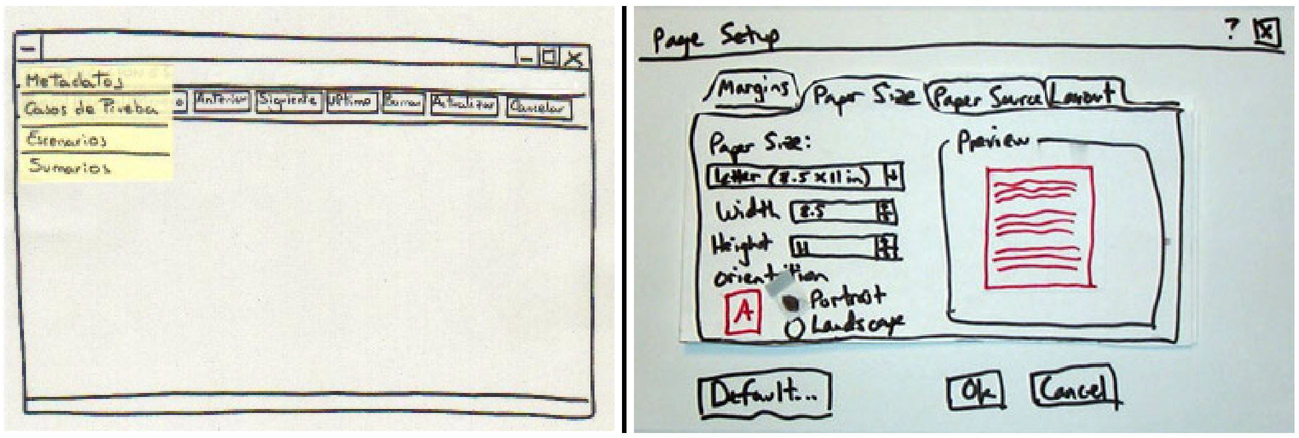

Figure II.3.3 Two examples of Paper Prototypes

Chauffeured prototyping involves the user watching while a member of the development team “drives” the system. It is a way to test whether the interface meets the user needs without the user actually having to carry out low-level actions with the system.

Participants

Developers, users and other stakeholders may participate in evaluation sessions that are based on paper prototypes.

Main Reference

L. L. Constantine, L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design*. Addison-Wesley, New York, NY, 1999. pp. 213-214.

II.3.12 Usability Specifications

Description

Usability specifications are quantitative usability goals, which are used as a guide for finding out when a software system is good enough in usability terms. They must be defined before design begins, and they must be testable to be able to decide whether the software product attains the specified usability level. Usability specifications are based on the basic five usability attributes or their subattributes, and they are related to a particular task (use case). Table II.3.1 shows part of a sample usability specification table.

By establishing usability specifications early in the development process, and monitoring them at each iteration, you can determine whether your system is, indeed, moving towards an improved, more usable, result.

Usability Attribute	Measuring Instrument	Value to be measured	Current Level	Worst Acceptable Level	Planned Target Level	Best Possible Level	Observed Results
Performance in normal use	“Answer Request” task	Length of time taken to successfully perform the task (minutes and seconds)	2’ 53’’	2’ 53’’	1’ 30’’	50’’	
Performance in normal use	“Answer Request” task	Number of errors during task performance	0	0	0	0	
First Impression	Questionnaire	Average score (range –2 to 2)	-	0	1	2	

Table II.3.1 Excerpt from a Sample Usability Specification table

How-to

The first step is to identify the usability attributes or subattributes that we want to cater for. Depending on the kind of product developed, some attributes might be irrelevant, for example, efficiency could be a secondary goal for a walk-and-use kiosk, while learnability would have top priority.

Attributes like satisfaction or first impression may be evaluated by means of subjective measures, normally in the form of questionnaires.

For performance-related attributes, a set of benchmark tasks must be selected and associated with each attribute. A benchmark task is a typical, representative use case a user will perform. Measuring a user’s performance on a benchmark task provides an objective usability metric for the related usability attribute. Benchmark tasks should be as specific as possible, so that there is little variability in their enactment by different users. The benchmark task in the first row in Table II.3.1 could be described as follows: “Suppose you are at the Help Desk counter, and you receive a request. You decide to answer the request, and you look for the answer in your knowledge base ...”.

The value to be measured must be then decided. For a benchmark the main values we can collect are the time to complete a task, or the number of errors during task performance. It is usually sensible to measure both of them, and we would have two rows in the Usability Specification table that are based on the same task.

The last step in defining usability specifications involves establishing the range of levels:

- The current level may refer to the value for the usability attribute in question with the current version of the system or with a competitor we want to challenge with our more usable product. When we are automating a manual procedure, it may refer to the time required to manually perform the task. If the system is very innovative, this field could be left blank.
- The worst acceptable level is the lowest acceptable level of user performance. It means that if the system does not reach this minimum level for any of the attributes in the specifications, the system is unacceptable from a usability point of view. The value for the current version of the product is usually taken as a reference to establish this level, and the level will be higher if the current version is unsatisfactory.
- The best possible level is a realistic upper limit. It should be an attainable level, not a wild dream. A hypothetical expert user should be able to attain this level. You can use developers as users to establish the best possible level, since they are the ones who are better acquainted with the subtleties of the interaction design. Another possibility is to use GOMS to provide theoretical estimates of expert error-free task performance.
- The planned target level is the attainment of unquestioned usability success. It is the most important value, because it is the actual requirement equivalent to traditional requirements. The other values must be filled in beforehand to help to set the planned target level in a sensible range of values. If there is a competitive system with a high usability level, it may serve as a reference for setting the planned target level.

When a system prototype exists, usability testing may be used to establish these levels at reasonable values.

Much expertise in the issue is required to establish good levels for the usability specification table, and this is why a usability specialist might be needed to apply this technique the first times it is used.

The observed results column will be filled in when the specifications are tested by means of usability testing and questionnaires.

Participants

Developers, usability specialists (to establish the range from worst acceptable level to best possible level).

Main Reference

D. Hix, H.R. Hartson. *Developing User Interfaces: Ensuring Usability Through Product and Process*. John Wiley and Sons, 1993. Chapter 8. pp. 221-248.

II.3.13 Cognitive Walkthrough

Description

Cognitive walkthrough is a technique for evaluating user interfaces by analysing the mental processes required of users. Like heuristic evaluation, the results are based on the judgement of the cognitive walkthrough analyst, instead of on results with real users. The difference is that it is focused on specific tasks, instead of on assessing the usability of the system as a whole.

In a cognitive walkthrough, correct sequences of actions are analysed, asking if they will actually be followed by users. The cognitive walkthrough analyst identifies problems by tracing the likely mental processes of a hypothetical user. The analysis considers matters like user background knowledge that influence mental processes but are not part of the user interface. The technique aims to identify likely usability problems in the user interface and to suggest reasons for these problems.

Cognitive walkthroughs were developed for systems that can be learned by exploratory browsing, but they are useful even for systems that require substantial learning.

How-to

The cognitive walkthrough analyst must begin by defining the assumed user background. The user structure role model should provide this information. Then the analyst must choose a representative task and devise a realistic usage of this task. If usage scenarios have been created, they can be a good source for realistic usage of tasks.

The analyst determines one or more correct sequences of actions for the chosen task. A correct sequence of actions is one that developers would be happy to see users use. Often there will be more than one acceptable way of performing a task. It these variations are important a cognitive walkthrough can be done on more than one, but often it will be sensible to choose the most common, or perhaps the most problematic.

The final step in the preparation of the cognitive walkthrough is to work out as fully as possible what the user will see at each step of the sequence or sequences to be examined. This may sometimes force the developers to create a partial design that is detailed enough to indicate the key interface features along the path. Screen sketches and /or dialogue flow (use cases) are usually enough to perform a cognitive walkthrough.

In the walkthrough itself, the analyst works through the sequence of correct actions, considering the state of the interface before and after each action, trying to determine how likely it is that users will follow that path. The kind of questions the analyst must try to answer are detailed in Table II.3.2.

Questions to ask about each correct action
• Will the user be trying to achieve the right effect? • Will the user notice that the correct action is available? • Will the user associate the correct action with the desired effect? • If the correct action is performed, will the user see that progress is being made?

Table II.3.2 Questions the Cognitive Walkthrough Analyst Tries to Answer

For each correct action, the analyst must construct a success or failure story. If all the answers to all the questions in Table II.3.2 are “yes”, including an explanation, then it is a success story. If the answer to one or more of the questions is “no” or “not always”, the analyst has a failure story. The explanation of this answer will tell the development team why the analyst expects that some users will have trouble at this point.

Participants

Developers acting as cognitive walkthrough analysts. If the analysts are not part of the development team that has designed the user interface, members of this team may participate to indicate the expected behaviour of the users with the user interface.

Main Reference

J. M. Carroll. “Cognitive Walkthroughs”, in *Handbook of Human-Computer Interaction. Second Edition*, edited by M. Helander, T. Landauer and P. Prabhu. North-Holland, 1997. Chapter 30. pp. 717-732.

II.3.14 Pluralistic Walkthrough

Description

A pluralistic usability walkthrough is a collaborative process involving users, developers and other stakeholders, where all participants are expected to play the role of users. The participants evaluate the interaction design by trying to perform a given task, and they stop at each step to have a group discussion about its usability. The goal of the technique is coordinated empathies to help developers to put themselves in the shoes of users.

How-to

A pluralistic walkthrough is driven by a task scenario chosen in advance and for which a storyboard, a series of screen sketches or paper prototypes representing the various contexts or a working prototype have been prepared. For each step in the task, all participants independently decide on what action or actions they would take next and note these on their own copies of the storyboard. No discussion takes place until all participants have completed a given step. When the discussion of the step begins end users speak first to prevent the developers dominating the discussion.

This technique is relatively slow, since all participants have to be at the same step at the same time. But this technique has some appealing advantages, cited by participants in this kind of evaluation, such as:

- They feel that their viewpoints have been heard,
- their expertise was valued, and
- their design concerns were remedied to satisfaction.

Participants

Developers, users and any other kind of stakeholders. All of them participate trying to think in terms of the end user.

Main Reference

R. G. Bias. “The Pluralistic Walk-Through: Coordinated Empathies”, in *Usability Inspection Methods*, edited by J. Nielsen and R. L. Mack. Wiley, 1994.

II.3.15 GOMS (Goals, Operators, Methods and Selection Rules)

Description

The GOMS model emerged from Cognitive Psychology theory to model how the human acts when trying to accomplish a goal by performing a task, which is formed by actions. These actions may be either physical (like pressing a button) or mental (cognitive operations such as recalling a name or deciding which option to choose). It aims to be an engineering model for usability, designed to produce quantitative predictions of how well humans will be able to perform tasks with a proposed design.

The GOMS model is based on *goals* (edit document) and subgoals (change a word) that the user formulates; the *operators* available to users, like motor, perceptual or cognitive primitives (click the mouse, look at the menubar); the *methods* users compose out of sequences of these operators to achieve the goals or subgoals (selection is done by moving the cursor to point to the word and double-clicking the mouse); and the *selection rules* needed to decide what to do next if the user has several goals pending or if there are several methods that will accomplish a given goal (the word can be removed by selecting it an issuing a “cut” command or by backspacing over it).

Each operation and selection rule is modelled as taking a certain amount of time, and therefore the developer can calculate the time need to perform several tasks by adding up the time for all the individual steps. The GOMS model is limited to error-free performance by expert users, and it can be used to produce performance estimates for this kind of users.

GOMS is actually a family of models, since there are variants for the notation used to describe the different elements that take part in a GOMS model. Of these, NGOMSL is a “natural” method of expressing the GOMS model.

How-to

NGOMSL is a complicated technique for cognitive task analysis, and it takes time and effort to master it. We will just give a brief indication of how it works, and the reader may consult the main reference below if he or she is interested in applying the technique.

In NGOMSL, learning time and execution time are predicted based on a program-like representation of the procedures that the user must learn and execute to perform tasks with the system. NGOMSL stands for Natural GOMS Language, because the notation used is a natural language structure to represent the user methods and selection rules.

NGOMSL starts after an initial task analysis has been performed, that is, after the user goals have been identified. The methods must be defined for each goal, by asking the question “how do you do it on this system?”. Each method is described as a series of steps. If all the operators in a method are primitive, this is the final level of analysis. However, if some operators are high-level, they must be examined to decide whether a method of analysis is needed. Then we can calculate a time estimate for each goal.

The NGOMSL model below describes how to move an object in the Macintosh Finder tool:

```
Method for goal: move an object.
        Step 1. Accomplish goal: drag object to destination.
        Step 2. Return with goal accomplished.
```

There is a submethod for describing the dragging operation:

```
Method for goal: drag item to destination.
        Step 1. Locate icon for item on screen.
        Step 2. Move cursor to item icon location.
        Step 3. Hold mouse button down.
        Step 4. Locate destination icon on screen.
        Step 5. Move cursor to destination icon.
```

Step 6. Verify that destination icon is reverse-video.
Step7. Release mouse button.
Step8. Return with goal accomplished.

Participants

Developer.

Main Reference

D. Kieras. "A guide to GOMS Model Usability Evaluation using NGOMSL", in *Handbook of Human-Computer Interaction. Second Edition*, edited by M. Helander, T. Landauer and P. Prabhu. North-Holland, 1997. Chapter 31. pp. 733-766.

II.3.16 Requirements Animation and Wizard of Oz Prototypes

Description

As opposed to paper prototyping, requirements animation involves building a working system whose appearance resembles the finished product. These prototypes are called active prototypes. Software prototypes are well known in software development. Therefore, we will not go into a lengthy description of and explanation of how-to do software prototyping and will focus on a less costly group of working prototypes: Wizard of Oz prototyping.

Wizard of Oz prototyping involves having some kind of behind-the-scenes manipulation to produce the responses of a working system, usually by means of a person providing the responses.

How-to

In a typical Wizard of Oz prototyping setting, the user interacts with a screen, but instead of a piece of software responding to the user requests, a developer is sitting at another screen answering the queries and responding to the real user. The user is unaware of the trick, so the perception of using a real working system is not spoiled. This kind of prototyping is widely used to prototype and test out user interface designs of many kinds, but especially for exotic or unusual configurations. For example, the development team may want to try out a telephone-based interface that mixes limited voice recognition with telephone keypad responses. The behaviour of the system is simulated by a person at the other end of the telephone line.

An advantage of this kind of prototyping for the development team is that extra understanding can be achieved through being involved so closely with the users.

Participants

The user using the prototype and the developer acting a facilitator of the evaluation, plus the hidden developer in the case of Wizard of Oz prototyping.

Main Reference

J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, T. Carey. *Human-Computer Interaction*. Addison Wesley, 1994. pp. 538-542.

II.3.17 Impact Analysis

Description

Impact or cost/importance analysis is a technique for deciding between design options, by relating the options to the usability problems that are affected and choosing the ones that address the most important usability problems. It is performed once we have a set of usability problems identified in any kind of usability evaluation activity. For each usability problem, we can propose a design decision, and use impact analysis to prioritise these decisions in order to undertake the redesign effort.

It is a tool for making the trade-offs necessary in any design process, which are based, in this case, on the usability issues that are addressed.

How-to

In an impact analysis, the development team considers the relative importance of the usability problems found, and the cost of the solutions as listed in a table like the one in Table II.3.3.

Usability Problem	Effect on User Performance	Importance	Solution(s)	Cost	Resolution
Too much window manipulation	10 of 35 minutes	High	Fix window placement automatically, but allow user to reposition it	6 hours	
Black arrow on black background	N/A	Low	Reverse arrow to white on black	1 hour	

Table II.3.3 Example of Table for Impact Analysis

Actually, impact analysis begins once all columns except the Resolution column have been completed for all observed problems. Depending on the number of design decisions to be considered, tools may be used for decision making, such as a graphical representation of problem distribution on the importance vs. cost scale. In principle, highly important problems should be tackled first, but the development team must also consider the resources allocated for the design activity and act accordingly. The process of deciding between design improvements that need to be made is not easy, and all this technique does is provide information in a structured manner so that the development team can make an informed decision.

Participants

Developers, and users if they participate in the design effort.

Main Reference

D. Hix, H.R. Hartson. *Developing User Interfaces: Ensuring Usability Through Product and Process*. John Wiley and Sons, 1993. pp. 316-330.

II.3.18 Screen Pictures

Description

Screen pictures are produced in order to define the appearance of the elements that form each screen of the system being developed, like buttons, text fields and scrollable lists. Screen pictures depict the information provided by the menu-selection and dialog-box trees, and by the context navigation map. If a GOMS model has been built, the information about the behaviour of the screen elements must be related to the information present in the GOMS model.

The information conveyed in screen pictures should allow user interface implementers to create the actual user interface. As screen design can be very time consuming, the use of screen design tools (or prototyping tools) can be very helpful, and the screens produced automatically become the model and it is not necessary to create a previous model that is implemented afterwards.

How-to

In order to create screen pictures, you should sketch some preliminary screen pictures, including the interaction /application objects, menus, buttons, and icons. You can label the functions and add notes about the behaviour of objects, where appropriate. Figure II.3.4 shows an example of a screen picture.

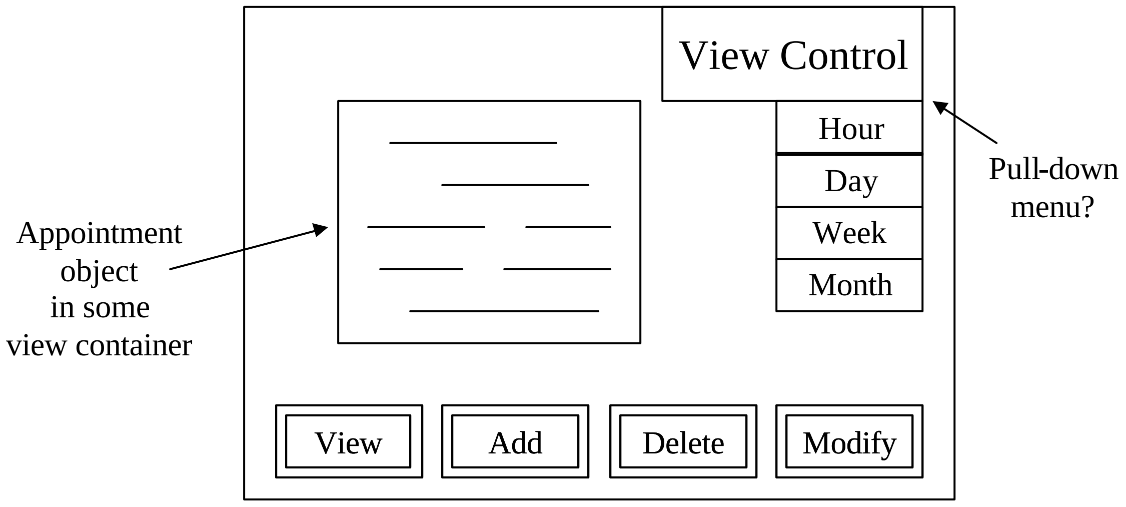

Figure II.3.4 Example of Screen Picture for a Personal Assistant

In the early screen pictures, the elements do not need to be represented with their final appearance, they may be concepts that identify the kind of manipulation needed or the kind of information represented, like an “area selector” or a “word list”. We do not need to commit ourselves to specific widgets in the user interface. If the concept of what role they play in the screen is clear, but the exact widget that is best for this purpose is not, then we can represent the element as a mere role. These concepts can be converted into interface widgets as the design is refined to become the actual screen in the user interface.

Software support for building screen pictures is available in the form of interface-building tools. The use of this kind of tools makes the process of screen design faster, and it may allow a collaborative approach that engages users and other stakeholders in the revision-redesign effort.

The creation of screen pictures is closely related to requirements animation and Wizard of Oz prototyping. Screen pictures may be used for prototyping, and some details on the behaviour of the different elements in the interface may need to be added so they can be implemented. Some interface-building tools support system development in addition to user interface prototyping. In this case, the specification of the user interface is the prototype itself, since it also encompasses the behaviour.

Participants

Developers. The screen pictures may be evaluated with users to allow a refinement of the solution.

Main Reference

D. Hix, H.R. Hartson. *Developing User Interfaces: Ensuring Usability Through Product and Process*. John Wiley and Sons, 1993. pp. 134-144.

II.3.19 Menu-Selection and Dialog Box Trees

Description

In a menu-based system, menu trees represent the structure of menu navigation. Menu trees are powerful as a specification tool since they show users and other stakeholders the complete and detailed coverage of the system. Like any map, a menu tree shows high-level relationships and low-level details.

Similar comments apply to dialog-boxes. Printing out the dialog boxes and showing their relationships by mounting them on a wall is very helpful for gaining an overview of the entire system to check for consistency and completeness.

How-to

When we cannot create an interaction scheme based on direct manipulation (as in the desktop metaphor in the Mac and Windows operating systems), we can use menu selection. If the menu items are written using familiar terminology and are organized in a convenient structure and sequence, users can select an item easily.

When a collection of items grows and becomes difficult to maintain under intellectual control, designers can form categories of similar items, creating a tree structure. Menu trees represent this structure. With large systems, the menu tree may have to be laid out on a large wall or floor, but it is important to be able to see the whole structure in one go to check for consistency, completeness, and lack of ambiguity or redundancy.

It is difficult to group menu items in a tree so that they are comprehensible to users and match the task structure. Problems include overlapping categories, extraneous items, conflicting classifications in the same menu, unfamiliar jargon, and generic terms. The members of the development team may discuss all these issues while they all share a view of the complete tree structure represented in a menu-selection tree.

Websites that are organised in a highly hierarchical structure can be easily represented by means of tree menus. Figure II.3.5 shows the menu tree of a website.

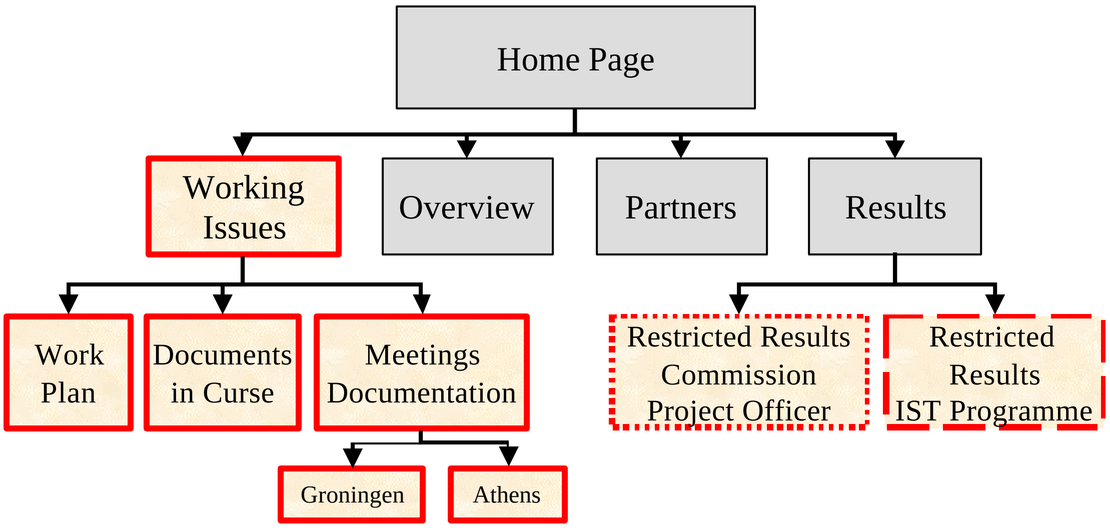

Figure II.3.5Example of a Website Menu Tree

Participants

Developer. Users may participate in the discussion of menu tree alternatives.

Main Reference

B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, 1998. pp. 247-252.

II.3.20 Context Navigation Maps

Description

A context navigation map models the interconnections among the various interaction spaces of the user interface. It gives more dynamic expressiveness than tree menus, by specifying the transition between the different interaction contexts that occurs when a use case is enacted. A context navigation map (or navigation map for short) represents the structure of the user interface by modelling the relationships among interaction contexts.

How-to

The navigation map is formed by boxes and arrows connecting these boxes. A box represents each interaction space. Arrows connecting interaction spaces represent possible transitions between them, such as calling up a dialogue from a command button or switching views through a menu. Arrow labels may indicate menu selection by means of a vertical bar (like, for example, View | Toolbars), the activation of a command button by means of square brackets ([Apply]) or an icon or tool selection by means of angle brackets (<Page Width>). Figure II.3.6 details the notation for navigation maps.

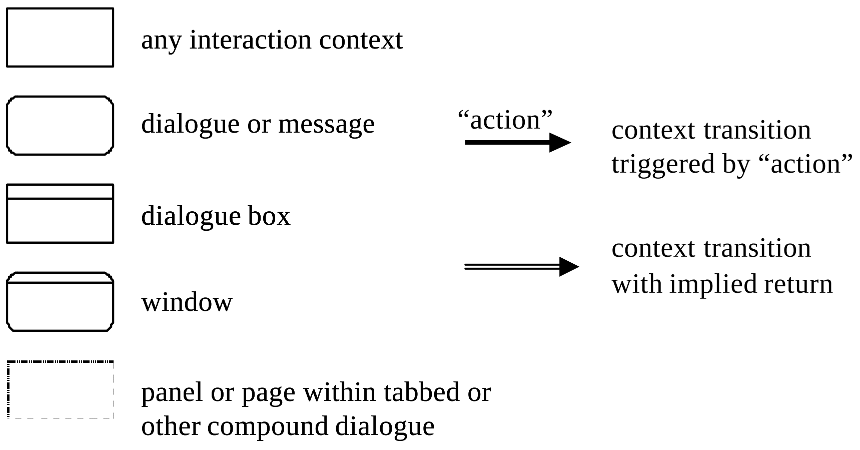
any interaction context
dialogue or message
dialogue box
window
panel or page within tabbed or other compound dialogue
“action”
context transition triggered by “action”
context transition with implied return

Figure II.3.6 Notation for Navigation Maps

The navigation map models the way users can navigate through the various interaction contexts within the user interface in the course of enacting use cases. When a single use case is represented, the navigation map models the behavioural view, and this is the most usual application of navigation maps. An example of a behavioural view is shown in Figure II.3.7. When the map combines all the behavioural views for the various use cases of the system in a single diagram, the result is called an architectural view. For big systems with a lot of interaction spaces the architectural view may get unwieldy, and too many transitions can lead to spaghetti-like diagrams.

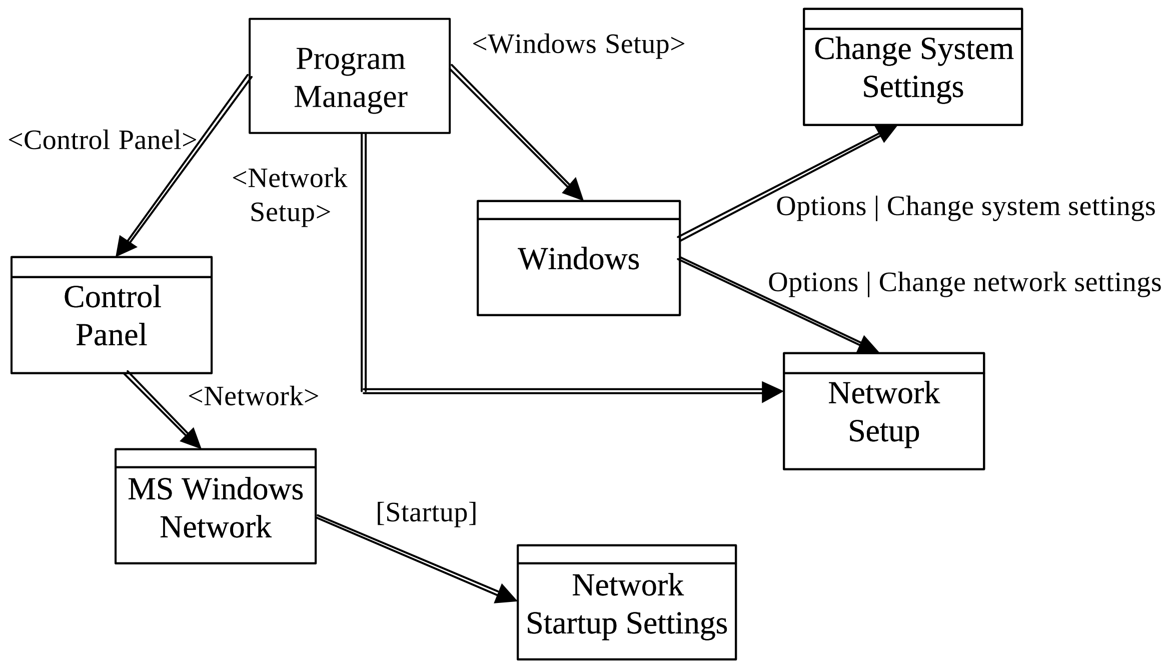

Figure II.3.7 Example of Navigation Map representing the Behavioural View

Participants

Developers.

Main Reference

L. L. Constantine, L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design*. Addison-Wesley, New York, NY, 1999. pp. 135-145.

II.3.21 Help Design

Description

The design of the help subsystem is important for the overall usability of the software product. Good help will not turn an unusable product into a highly usable one, but well-written, well-organised and accessible help can compensate to some extent for limitations in software.

Essential use cases express what the users may want to accomplish with the system, so they are an excellent basis for organising help contents. This kind of structure is useful for providing procedural help, that is, help with how to perform a task.

Additionally, the design of the help subsystem may be undertaken for a set of general help use cases (or help cases), which give a response to common help requests made by users of all kind of systems: *seekingIdentification*, *seekingInstruction*, *seekingClarification*, *seekingElaboration*,*seekingReminder*, *seekingLocation*, and *exploringFeatures*.

How-to

Procedural help is most helpful when it is organised by use cases that are titled and written in the ordinary language of the users and the application domain and are well indexed. Use cases are a natural way of organising and providing access to help because they represent the basic intents of users. Each essential use case is a complete and well-defined task based on something a user might try to accomplish. If the essential use case model has been well constructed, it will reflect how users think about and conduct their work. Each use case then becomes an entry in the help file.

To design support for other kinds of help, we can focus on the common help requests described as help cases (use cases for help seeking). Note that procedural help is expressed by means of the help case *seekingInstruction*. Table II.3.4 details common user questions and the corresponding help case.

User Question	Help case	
What is this?	seekingIdentification	
	indicate object	brief description
How do I...?	seekingInstruction	
	identify task	operational sequence
What should I do?	seekingSuggestion	
	request	hint
What do you mean?	seekingClarification	
	request	different explanation
Tell me more	seekingElaboration	
	request	details or advanced
Remind me about...	seekingReminder	
	identify feature/task	brief synopsis
Where is...?	seekingLocation	
	identify feature	give place, routing
What can I do?	exploringFeatures	
	request	overview, topic map

Table II.3.4 Help Cases for Common User Help Requests

Participants

Developers.

Main Reference

L. L. Constantine, L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design*. Addison-Wesley, New York, NY, 1999. Chapter 11. pp. 231- 264.

II.3.22 Heuristic Evaluation

Description

Heuristic evaluation is performed to identify the usability problems of a system, so that they can be attended to as part of an iterative design process. It involves having a small set of evaluators examine the interaction design and judge its compliance with recognised usability principles (the heuristics).

It can be used as a complement to usability testing with users, since it usually reveals different kinds of usability problems than usability testing.

How-to

Each evaluator inspects the interaction design alone. After the evaluations have been completed, the evaluators may gather to report their findings. This procedure is important in order to ensure independent and unbiased evaluations from each evaluator.

During the evaluation session, the evaluator goes through the interaction scheme (screen sketches and/or use case description) several times and inspects the various dialogue elements and compares them with the list of recognised usability principles. These heuristics are general rules that seem to describe common properties of usable interfaces, like the ones described in Table II.3.5. In addition to the checklist of general heuristics to be considered for all dialogue elements, the evaluator is also allowed to consider any additional usability principles or results that come to mind that may be relevant for any specific dialogue element.

Usability Heuristics
• Use simple and natural dialogue • Speak the users' language • Minimize user memory load • Be consistent • Provide feedback • Provide clearly marked exits • Provide shortcuts • Provide good error messages • Prevent errors

Table II.3.5 Example of List of Usability Heuristics

The number of evaluators to be employed depends on the criticality of system usability, but it is clearly better to combine evaluations by several evaluators than have a single evaluator. Experts recommend using about five evaluators and certainly at least three.

Unlike other evaluation methods, such as walkthroughs, the evaluators decide on their own how they want to proceed with evaluating the interface, instead of following the predefined paths given by use cases.

The output from heuristic evaluation is a list of usability problems in the interaction design, annotated with references to the usability principles that were, in the opinion of the evaluator, violated by the design in each case.

Participants

Developers acting as evaluators.

Main Reference

J. Nielsen. *Usability Engineering*. AP Professional, 1993. pp. 155-162.

II.3.23 Usability Inspections

Description

Inspections have a long history in software development. The goal of all inspections is to find defects. Usability inspections are aimed at identifying usability defects. The object of inspection may be a finished product, a design or a prototype. Usability inspections refer to systematic processes for inspection, as opposed to heuristic evaluation, which is a less formal usability assessment technique.

When different stakeholders perform the inspection in a collaborative effort, it is called collaborative usability inspection. In this case, the review process is a team effort that includes software developers, end users, application or domain experts and usability specialists, collaborating to perform a thorough and efficient inspection.

There are two variants of inspection, which have a specific focus: consistency inspections and conformance inspections.

How-to

In consistency inspections, the goal is to identify inconsistencies across interaction contexts and their contents. The evaluators check for consistency of terminology, colour, layout, input and output formats, and so on. When the product belongs to a family of products, teams of designers, at least one from each project, meet to inspect the usability of the different products of the family.

In conformance inspections, the participants inspect the system interaction for compliance with specified standards or with style guidelines. All participants must be familiar with the applicable standards and/or style guidelines.

Collaborative usability inspections, if well conducted, can be more productive than expert inspections. The focus needs to be kept on the user perspective, in order to identify the usability problems that might arise. Developers need to adopt the mindset of an impatient and intolerant user. The presence of actual users in inspections helps to catalyse taking the user perspective. Two special roles in the inspection team are the lead reviewer, who organizes the inspection meetings and moderates the process; and the inspection recorder, who maintains a complete log of identified defects. Another special role in the team may be the continuity reviewer, who has special responsibility for identifying inconsistencies. Apart from members of the development team, it may be useful to have some developers who have not participated in the development effort, because they bring a fresh perspective into the inspection. Members of the development team are not allowed to defend, explain, excuse or rationalize any aspect of their design or the decisions leading to it. Developers should also avoid making implied promises to the users. The comments and inputs from users should be given special weight in the inspection process, without allowing these to dictate interaction design decisions. The lead reviewer should encourage user participation and protect users from criticism or antagonistic questioning. Users and domain experts should be regarded as authorities, but not as arbiters. Finally, usability experts may also contribute to collaborative usability inspections.

Participants

Developers as evaluators. Users and other stakeholders in collaborative usability inspections.

Main Reference

L. L. Constantine, L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design*. Addison-Wesley, New York, NY, 1999. pp. 397-415.

II.3.24 Thinking Aloud

Description

Thinking aloud is a technique for performing usability tests with users. The evaluator asks participants (users) to talk out loud while working during a usability testing session, indicating what they are trying to do, or why they are having a problem, what they expected to happen that did not, what they wished had happened, and so on. By verbalising their thoughts, test participants enable the developer to understand how they view the system, and this helps to identify the major user misconceptions.

The strength of thinking out loud is on qualitative data and not on performance measures. The idea is to get the user's impression while using the system to avoid later rationalisations. The aim of this kind of testing is to detect the parts of the dialogue that are more problematic from a usability point of view, along with the real causes of the problems.

There are some variants of this technique: constructive interaction, retrospective testing, critical incident taking and coaching method.

How-to

Thinking aloud is can be employed in any usability test. There is no difference in the test preparation with performance measurement usability testing. But, before starting with the test, the evaluator must encourage the test participant to think out loud, maintaining a running monologue about what he or she is doing as it is being done.

The evaluator may find that some participants are not good at thinking aloud while they work. They will not talk much, and the evaluator will have to prod them constantly to find out what they are thinking or trying to do. This has an impact on performance measures, so we do not advise combining thinking aloud with performance measurement.

User comments are sometimes indicators of personal user likes or dislikes, so developers should take care not to change part of the system just because of a comment by a single user. It is the responsibility of the evaluator to interpret the user comments and not just accept them indiscriminately. For example, users using a mouse for the first time will often direct a large proportion of their comments toward aspects of moving the mouse and pointing and clicking. In this case, the evaluator should try to abstract from the mouse problems in the dialogue and focus on other system issues.

The following techniques are variants of the basic think-aloud protocol:

- **Constructive interaction**: It involves having two test users use a system together. It is also called codiscovery learning. It aims to overcome the problem of shy test participants, who do not verbalise easily. This is based on the fact that people are used to verbalising when they are trying to solve a problem in a collaborative effort.
- **Retrospective testing**: The usability testing session is recorded on a videotape and the participant is requested to review the recording. Participant comments while reviewing the tape are sometimes more extensive than comments while performing the task in the test. The evaluator can stop the tape and ask the participant questions at any time, without fear of interfering with the test, which has essentially been completed already. This variant may be useful when the usability testing involves some kind of performance measurement that could be distorted by the dialogue with the reviewer.
- **Critical incident taking**: This variant implies recording both negative incidents (signs of frustration, either with remarks or actions), and positive incidents (satisfaction or closure expressions). Negative incidents help to identify the more important usability problems, while positive incidents help to identify metaphors or details to be used more thoroughly in the user interface because of their success

- **Coaching method**: The evaluator (or “coach”) steers the participant in the right direction while using the system. The participant can ask the evaluator questions, and the questions may show up usability problems that would remain uncovered otherwise. The evaluator will answer to the best of his or her ability.

Participants

Developers as evaluators, and users as test participants.

Main Reference

J. Nielsen. *Usability Engineering*. AP Professional, 1993. pp. 195-200.

II.3.25 Performance Measurement / Laboratory Usability Testing

Description

Performance measurement through usability testing is used for assessing whether usability goals set in usability specifications have been met. It can be used as well for comparisons with competing products.

Performance is measured by having a group of users perform a predefined set of test tasks while collecting time and error data. When the test is performed in a special room prepared for usability testing, it is called laboratory usability testing. A laboratory is usually composed of two rooms separated by a one-way mirror: the evaluation room where participants carry out the tests and the main evaluator gives instructions; and the control room, where additional evaluators and other members of the development team can observe the test, without disturbing the test participant. Usual equipment for a usability laboratory includes a video camera to record the screen, another one for recording the participant, tools for software logging and monitors to show in the control room what is happening in the evaluation room.

The opposite to laboratory testing is field testing, where the system is taken to the user environment instead of taking the participant to the system, and the usability test is performed in the user organization.

How-to

Before performing any test, the first step is to develop the experiment. The participants must be selected, trying to get a representative sample of the total user population. Information in the structured user role model should serve to select an adequate distribution of test participants. The tasks to be employed in the tests must be defined as well, the benchmark tasks that appear in the usability specifications must be tested, but additional representative tasks may be tested as well, not to get performance measurements but to identify usability problems. The evaluator should write down the tasks in the order that the participant will be asked to perform them. This list of tasks may be either given to the participant or read out loud by the evaluator one task at a time. Finally, the evaluator must define the protocol and procedures for the test. This includes the preparation of introductory instructional remarks for participants, which should briefly explain the purpose of the experiment, the system to be tested and what the participant will be expected to do. It is important to make clear to all participants that the purpose of the test session is to evaluate the system, not the participant. An informed consent form should also be prepared for participants to sign, stating that the participant is volunteering for the experiment, that the data may be used if the participant's name or identity is not associated with the data, that the participant understands that the experiment is in no way harmful and that the participant may discontinue the experiment at any time.

It is advisable to perform a few pilot tests with three or four participants to ensure that all parts of the experiment are ready. Pilot testing may show up inadequate wording of the tasks that the participants are being asked to perform, or that some part of the procedure needs to be changed. After pilot testing the test plan is refined in order to proceed with the greater part of the testing effort using an improved test plan.

For the test session, the evaluator will usually be sitting beside the participant, especially when qualitative data needs to be collected. The test participant is asked to perform the tasks defined for the test, and both the number of errors and the performance time are measured for each task. It may be necessary to prompt the participant during the session, primarily during qualitative data collection, to get the desired information. The think-aloud technique and its variants may be applied for this purpose in any usability test.

The data collected during the test sessions must then be analysed. Quantitative data will be formed by performance times, error rates, and also by the user preference expressed in questionnaires. Qualitative data will come from the user comments that the evaluator has taken down or extracted from an audio or video recording of the session. The information gathered in usability tests can tell the development team whether or not the development is going in the right direction (that is, whether we are coming closer to the goals in the usability specifications or not), and it can point out the issues in the interaction dialogue that are a source of usability problems. After an impact analysis, decisions are taken about which usability problems will be tackled first in the next cycle redesign effort.

Participants

Developers as evaluators, users as test participants. It is important to use the wording "participant" instead of "test subject", since it is the software product or prototype, not the user, that is being tested.

Main Reference

J. Rubin. *Handbook of Usability Testing*. John Wiley and Sons, 1994.

II.3.26 Questionnaires and Surveys

Description

Questionnaires are used to determine a user's subjective satisfaction with the system. Measuring user satisfaction provides a subjective (but, nevertheless, quantitative) usability metric for the related usability attribute. Some usability specifications will be related to user satisfaction, and questionnaires are the way to check whether the level specified for this attribute has been reached. Questionnaires are usually administered to usability test participants after the test has taken place, so they can give their opinion about specific parts of the user interface and about the overall system.

When questionnaires are distributed to a lot of users, they are called surveys. While questionnaires issued to usability test participants may contain questions about specific parts that have been used in the test, surveys usually gather opinions on more generic issues. Additional information that is usually collected has to do with individual user characteristics, such as background (age, gender, education), experience with computers, familiarity with specific features (virtual reality, macros, shortcuts), and so on.

How-to

It is advisable to do a pilot study before sending questionnaires to a large number of users in order to ensure that it is well designed. Care must be taken to make the questions unambiguous, and the questionnaire in general should be as simple as possible to increase the chance of respondents completing and returning the questionnaire.

Questions may be open, where the respondent is free to provide his or her own answer, or closed, where the respondent is asked to select an answer from a choice or alternative replies. Closed questions usually have some form of associated rating scale. The most commonly used scale for HCI studies is the semantic differential scale. This scale is based on bipolar adjectives (such as easy-difficult, clear-confusing) at the end points of the scale and respondents rate on a scale between these paired adjectives. This is the scale used in the questions in Table II.3.6.

Once the questionnaires have been given to the selected population, the responses obtained on the different rating scales are converted into numerical values and statistical analysis is performed. The main statistics used in the analysis of surveys data are means and standard deviations.

Table II.3.6 shows sample questions belonging to a questionnaire to be administered to test participants after a usability testing session. The tool to be tested provided facilities for managing problem resolution tasks in a Help Desk. A differential semantic scale with five choices was used for each question, centring the scale around zero. As a mid-scale reading, zero is an appropriately neutral value. Negative scale readings correspond to negative user opinions and positive readings to positive opinions. The final category of questions is focused on overall user reactions.

	unsatisfactory				satisfactory
General satisfaction	-2	-1	0	1	2
	not suitable				suitable
Suitability for problem solving tasks	-2	-1	0	1	2
	worst				better
General comparison with existing system	-2	-1	0	1	2
	too little				enough
Feedback provided to user actions	-2	-1	0	1	2

Overall opinion about the system:	terrible				wonderful
	-2	-1	0	1	2
	frustrating				satisfying
	-2	-1	0	1	2
	dull				stimulating
	-2	-1	0	1	2
	difficult				easy
	-2	-1	0	1	2
	rigid				flexible
	-2	-1	0	1	2

Table II.3.6 Sample questions from a User Preference Questionnaire

Beta-testing is a survey-based form of evaluation. In beta-testing, a working but not completely finished version is supplied to a big pool of customers who are willing to test the product using it to perform their work (or to fulfil their goals). In addition to questions on possible system failures, beta-testers may be asked to answer preference questions after their usage of the system.

Participants

Developers create questionnaires and analyse the data, while users fill in the questionnaires stating their personal opinions.

Main Reference

J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, T. Carey. *Human-Computer Interaction*. Addison Wesley, 1994. pp. 631-638.

II.3.27 Interviews

Description

Interviews involve having an interviewer read questions to a respondent and writing down the responses. For the creation of the questionnaire, refer to section II.3.26 Questionnaires and Surveys.

After usability testing the evaluator may interview the participant to get the user's subjective opinion, instead of letting the participant fill in a written questionnaire. Interviews are more flexible, since the evaluator may ask follow-up questions that not were in the script.

How-to

Interviews need to be planned for them to yield useful results, much in the same way questionnaires must be carefully planned before being administered to users.

There are two main kinds of interviews: structured, where the questions are predetermined and flexible interviews, where the interviewer is free to follow the interviewee's replies and to find out personal attitudes. Flexible interviews are less formal, and they are adequate for requirements elicitation and for gauging users' opinions about a particular idea. No matter how flexible the interview is going to be, a rough plan of the topics to be discussed is still needed.

The interviewer should make the interviewee feel comfortable, establishing interviewer-interviewee rapport. For example, some people feel embarrassed when they criticise a system, particularly when they have to describe their own difficulties in using it.

When the interviewer has a set of questions prepared in case the interviewee digresses or does not say much, it is called a semi-structured interview. A variant for drawing out more information from the interviewee is prompted interviewing, where the interviewer stimulates the interviewee by saying things like "... and can you tell me a bit more about that" or "...and what do you mean by...". Alternatively, prompting may take the form of showing the interviewee an alternative item such as a screen design, in order to promote further discussion or generate new ideas for discussion.

The trade-off to be considered in structured vs. flexible interviewing is that the less structured the interview is, the more scope there is for picking up relevant issues but the harder it is for the interviewer. Flexible interviews on usability issues have been predominantly used to determine the user's understanding of the interaction scheme. An issue to consider is that the interviewer should avoid asking leading questions that beg a particular response.

As for questionnaires and surveys, when preparing an interview with domain experts (who are usually a scarce resource), it is better to do a small pilot study to be able to refine the interview script.

Participants

Developers as interviewers, and users or other stakeholders as interviewees.

Main Reference

J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, T. Carey. *Human-Computer Interaction*. Addison Wesley, 1994. pp. 628-631.

II.3.28 Direct Observation and Video/Audio Recording

Description

Individual users may be directly observed doing specially devised tasks or doing their normal work, with the observer making notes about interesting behaviour or recording their performance in some way, such as timing sequences of actions. This is called direct observation. When the observation takes place in the user organization, it is called field usability evaluation.

Video recording can be either an alternative to direct observation or a backup for what happens in a usability evaluation session. For field usability evaluation, audio recording can be useful as well to record the user comments.

How-to

The evaluator should be prepared to take copious notes as activities proceed during a usability evaluation session. It may be useful to have a second evaluator also observing the session in order to help take notes. Especially for usability testing sessions, the first evaluator may be in charge of conducting the session (giving instructions, prompting the user) and timing tasks where necessary, while the second evaluator may be in charge of just taking notes.

Even if the evaluator (or evaluators) is fast at note taking, the record of the observation will usually be incomplete. Direct observation only allows for one go at data collection, so the evaluator rarely gets a full record of user activity for detailed analysis. The evaluator has to make decisions about what is important to record and has no chance to revise that decision and look at alternative data later on. For these reasons, if a permanent record is needed, video recording equipment may be used to record usability evaluation sessions. Usability laboratories are usually equipped with video cameras and perhaps some video editing equipment as well. The main advantage of videotaping is to capture every detail that occurs during the session. If multiple cameras are available, one can be aimed at the participant’s hands and the screen, and another at a broader view including the participant’s face. Audiotaping may be done when videotaping is not available, as, for example, in field testing. Just having a record of all the user’s comments may prove invaluable for later data analysis.

The main disadvantage of videotaping is the time it takes to edit the taped material. A ratio of 5:1 (analysis time to recording time) is often cited, that is, it usually takes five hours to analyse one hour of videotape. When more than one camera is used, the editing is also very time consuming, since synchronization problems may arise.

Short video clips of users experiencing problems with a given software product can have a big influence on a development team, especially, if the development team is reluctant to make changes to what they consider to be their already perfect design. These same video clips can also be useful for convincing management that there is a usability problem in the first place.

Participants

Developers as observers, and users.

Main Reference

D. Hix, H.R. Hartson. *Developing User Interfaces: Ensuring Usability Through Product and Process*. John Wiley and Sons, 1993. pp. 309-313.

II.3.29 Focus Groups

Description

Focus groups are a somewhat informal technique that can be used to assess user needs and feelings after the system has been in use for some time. Focus groups often bring out spontaneous reactions and ideas from users through the interaction between the participants and have the major advantage of allowing some group dynamics and organizational issues. Focus groups are especially appropriate for limited user communities.

How-to

In a focus group, a group of users are brought together to discuss new concepts and identify issues over a period of about two hours. Each group is run by a moderator who is responsible for maintaining the focus of the group on whatever issues are of interest. From the user perspective, a focus group session should feel free flowing and relatively unstructured, but, in reality, the moderator has to follow a preplanned script for what issues to bring up.

To prepare a focus group, the moderator needs to prepare a list of the issues to be discussed and set goals for the kinds of information that are to be gathered. During the group session the moderator has the difficult job of keeping the discussion on track without inhibiting the free flow of ideas and comments. Also, the moderator needs to ensure that all members of the group get to contribute to the discussion and guard against having the opinions of any single participant dominate unduly. After the session, data analysis can be as simple as having the moderator write a short report summing up the prevailing mood in the group, illustrated with a few colourful quotes.

Focus group discussions may be held after a set of individual user interviews have been conducted. Then, focus-group discussions may be valuable to ascertain the universality of comments. Individual interviews are costly and time consuming, so usually only a small fraction of the user community is involved. On the other hand, group discussions offer more representative results. .

Participants

Developer acting as moderator, and users.

Main Reference

J. Nielsen. *Usability Engineering*. AP Professional, 1993. pp. 214-217.

II.3.30 Logging Actual Use

Description

Logging involves having the computer automatically collect statistics about the detailed use of the system. It is mainly used to collect information about the field use of a system after release, but it can also be used as a supplementary method during usability testing to collect more detailed data. It is unobtrusive, so it does not interfere with the user's normal usage of the system.

When the actual use of the system is logged, this information is particularly useful because it shows how users perform their actual work and because it is relatively easy to automatically collect data from a large number of users working under different circumstances. Typically, an interface log will contain statistics about the frequency with which each user has used each feature in the system, and the frequency with which various events of interest (like, for example, error messages) have occurred.

When undertaking a major redesign for a system that has been in use, it is very helpful to rely on interaction log information to guide the redesign effort.

How-to

For this technique to be applied, the software architecture should make it easy for system managers to collect data about the patterns of system usage, speed of user performance, rate of errors or requests for online assistance.

There are different software logging tools that can be employed for logging actual use, but there are two main categories: time-stamped keypresses and real-time interaction logging. Logging time-stamped keypresses simply provides a record of each key that the user presses along with the exact time of the event. Interaction logging is similar, except that the recording includes real-time information, which means that it can be replayed in real time so the observer can see the interaction between the user and the computer exactly as it happened.

Logging may be well intentioned, but user rights to privacy should be respected. Links to specific user names should not be collected, unless necessary. When logging aggregate performance crosses over to monitoring individual activity, managers must inform users of what is being monitored and how the information will be used.

It is usual to combine video, audio and keypresses or interaction logging. The advantage of using combinations of data capture techniques is that evaluators can relate revealing data about body language and comments with records of the actual human-computer interaction. The main disadvantage of this approach is the cost of setting up this kind of synchronised equipment.

Participants

Developers must provide the mechanisms for data logging in the software product design.

Main Reference

B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, 1998. pp. 146-147.

II.3.31 Online User Feedback Facilities

Description

Once the system is in use, the user community is the best source for information on the usability weaknesses of the system. Feedback from the users can be collected by giving them access to special electronic mail addresses, network newsgroups, or bulletin boards. Users can send their complaints and requests for change or improvement.

How-to

Offering a help line or a communication channel with users can be implemented in different forms. These are the main ways of gathering user-initiated feedback:

- **Online or Telephone Consultants**: They can provide extremely effective and personal assistance to users who are experiencing difficulties. Many users feel reassured if they know that there is a human being whom they can address if problems arise. These consultants are an excellent source of information about problems users are having and can suggest improvements and potential extensions. Some organizations offer a toll-free number for users, while others charge for consultation by the minute.
- **Online Suggestion Box or Trouble Reporting**: Email can be employed to allow users to send messages to the maintainers or designers. Such an "online suggestion box" encourages some users to make productive comments, since writing a letter may be seen as requiring too much effort.
- **Online Bulletin Board or Newsgroup**: Users may have questions about the suitability of a software package for their application, or may be seeking someone who has had experience using an interface feature. They do not have any individual in mind, so email does not serve their needs. Then bulletin boards and newsgroups can be helpful. Electronic bulletin boards or newsgroups allow the posting of open messages and questions. Mailing lists may be used as well for this purpose.

By soliciting user feedback by any of these ways, the development team can gauge user attitudes and elicit useful suggestions. Furthermore, users may have more positive attitudes towards the system if they see that the software development organization genuinely desires comments and suggestions on the piece of software they are using.

Participants

Users, developers to analyse the change proposals, dedicated personnel for the communication channel with the users.

Main Reference

B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, 1998.

Part III. TRAINING COURSE IN USABILITY

III.1 COURSE CONTENTS

The course will be divided into subjects, starting with the introductory topics and basic usability concepts and then going on to deal with the different individual techniques that are part of the deltas that are going to be added to the software development process.

III.1.1 Usability Awareness

The goal of this subject is to provide developers with usability awareness to make them aware of the need for usability and to specify the concept, clearing up common misconceptions of the issue. The aim is to overcome the barrier erected by the perception that usability is a subject unrelated to their field of work as software developers or whose importance is low upon their list of priorities.

The contents to be covered in this subject are as follows:

- Why care about usability?
- Brief introduction to ergonomics
- Usability as a quality attribute
- Real life examples of unusable systems (both software and non-software)
- Common usability misconceptions
- Usability principles / slogans (philosophy)
- The user-centred perspective for software development
- Financial impact analysis (Cost-justifying usability)

III.1.2 Basic Usability Concepts

This subject addresses the main usability concepts, which will give software developers an understanding of the basis on which the techniques covered in the remainder of the course are founded.

The topics covered in this subject are as follows:

- Usability attributes
- Usability and the user interface
- Human perception and cognitive issues
- User and task analysis
- Usability specifications
- Interaction styles
- Interaction design
- Prototyping / Iterative design
- User involvement (participatory design)
- Usability heuristics
- Usability evaluation
- Usability laboratories

III.1.3 Analysing the User and User Context

This subject deals with the usability techniques related to problem analysis and the development of the first conception of what the system under development will be like. It will include practical exercises and the study of existing systems. The techniques covered correspond with delta L1 for light software development processes and deltas E1, E2, E3 and E4 for elaborate software development processes.

The topics that will be covered are as follows:

- Elicitation Techniques:
  - Ethnographic Observation
  - Contextual Inquiry
- Develop Product Concept:
  - Post-It Notes
  - Visual Brainstorming
  - Competitive Analysis
  - Scenarios
- Prototyping:
  - Paper Prototypes
  - Chauffeured Prototypes
  - Wizard of Oz Prototypes
  - Requirements Animation
- Problem Understanding:
  - Essential Use Cases
- Modelling the Context of Use:
  - Structured User Role Model
  - Operational Modelling
  - JEM
- Early Usability Evaluation:
  - Cognitive Walkthrough
  - Pluralistic Walkthrough
- Usability Specifications

III.1.4 Interaction Design

This subject will deal with interaction design, including the design of the user interface but also focusing on the general interaction scheme with which the system will work. The techniques that will be covered correspond to part of delta L2 or delta E5.

The topics covered in this subject are as follows:

- Impact Analysis
- Detailed Interaction Design:

  - Detailed Use Cases
  - GOMS
- User Interface Design:
  - Screen Pictures
  - Menu-selection and Dialog Box Trees
  - Context Navigation Maps
- Help Design:
  - Help Design by Use Cases

III.1.5 Architecting Software for Usability

This subject will cover the techniques developed in WP3 for developing an architectural design that takes into account the usability of the final system. The techniques dealt with in this subject correspond to part of delta L2 or delta E6.

(The topics to be covered in this subject will be specified here when the results of WP3 are available).

III.1.6 Usability Evaluation

In this subject, developers will learn how to evaluate usability, becoming acquainted with the available techniques and their application conditions. The techniques dealt with in this subject correspond to delta L3 or to deltas E7 and E8.

The topics covered by this subject are as follows:

- Expert Evaluation:
  - Heuristic Evaluation
  - Inspections: Consistency, conformance and collaborative usability inspections
- Usability Testing:
  - Thinking Aloud: Constructive interaction, retrospective testing, critical incident taking, and coaching method
  - Performance Measurement
  - Post-Test Feedback / User Questionnaires
  - Laboratory Usability Testing
- Follow-up Studies of Installed Systems:
  - Direct Observation
  - Video Recording
  - Audio Protocol
  - Questionnaires
  - Structured and Flexible Interviews
  - Focus Groups
  - Logging Actual Use: Time-stamped keypresses and interaction logging

III.2 COURSE DURATION

The total course duration is 60 hours, plus the time for the subject *Architecting software for usability*. The time dedicated to each subject is detailed in Table III.2.1.

Subject		Duration (hours)
	Usability Awareness	4
	Basic Concepts on Usability	5
Analysis Techniques	Elicitation Techniques	4
	Develop Product Concept	3
	Prototyping	4
	Essential Use Cases	3
	Modelling the Context of Use	4
	Walkthroughs	3
	Usability Specifications	3
Interaction Design	Impact Analysis	2
	Detailed Interaction Design	5
	User Interface Design	5
	Help Design	3
Architecting Software for Usability		-
Usability Evaluation	Expert Evaluation	3
	Usability Testing	4
	Follow-up Studies of Installed Systems	5

Table III.2.1 Duration of Course Subjects

III.3 COURSE RESOURCES

The following resources will be needed to teach the course:

- Classrooms with computers for each or every two students.
- Audiovisual equipment for showing recordings of usability testing sessions.
- Software for building prototypes and support software for applying some techniques (for example, diagram modelling and software logging).

III.4 REFERENCES FOR PART III

[ACM, 02]	ACM SIGCHI. *Curricula for Human-Computer Interaction*, 2002 http://www1.acm.org/sigs/sigchi/cdg/
[Ferré, 01]	X. Ferré, N. Juristo, H. Windl, L. Constantine. “Usability Basics for Software Developers”. *IEEE Software*, vol.18, no.1, pp. 22-29. January/February 2001.
[Myers, 99]	Brad A. Myers. *Overview of HCI Design and Implementation. Notes for HCI Talk by Brad A. Myers*, 1999 http://www.cs.cmu.edu/~bam/uicourse/special/
[Nielsen, 93]	J. Nielsen. *Usability Engineering*. AP Professional, 1993.
[Preece, 94]	J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, T. Carey. *Human-Computer Interaction*. Addison Wesley, 1994.
[UMD, 02]	University of Maryland. *CMSC 434/828S Intro to HCI - Syllabus*. http://www.cs.umd.edu/~bederson/classes/cmsc434/syllabus.htm