Ensuring quality of the user interface with the GUIDe process model

Sari A. Laakso, University of Helsinki, Dept. of Computer Science
Karri-Pekka Laakso, Interacta Design Oy

Abstract

In many projects, the problems with the user interface do not become evident until the system is already in use and the users are trying to get their work done. At that point, it may be very expensive to make changes. The problems may be caused by an insufficient or even non-existent interaction design process, as a result of which the user interface emerges as a by-product of the implementation. Another source of problems may be customer requirements that change during the project and become evident only gradually as the software is being implemented and the product's functionality is concretized into the user interface.

In our GUIDe process model (Goals - User Interface Design - Implementation), the user interface is designed in the beginning of the project as a concrete requirement for the system architecture or database design. In addition to the user interface, the design process systematically produces functional and data requirements.

In GUIDe, an explicit user interface design phase ensures that it will be possible to test the user interface and the system's applicability to its intended use at such an early stage that any changes indicated by the test results, even major ones, will still be easy to make. Making the necessary changes to the 'screenshot' sequences of the user interface specification requires less work and is much cheaper than using a functioning piece of software as an iteration tool for customer requirements and user interface design. The user interface specification is an illustrative miniature model of the system, which makes it also an effective communications tool between the developer and the customer.

Keywords: GUIDe, user interface design, interaction design, utility, usability, goal-based use cases, customer requirements, requirement specifications, process model.

ACM CCS: H.5.2, D.2.1

Contents

1 Introduction

Typically, the problems with a new user interface do not become evident until the software has been taken into use, and its users are trying to reach their actual goals with it for the first time. In the worst case, some essential features may be missing completely. In addition to missing features and data, the user typically faces usability problems that are caused by inefficient or confusing interaction with the system and that lead to waste of time, errors and need for extra training. Some of the interaction shortcomings are so bad that they have to be corrected at a huge cost before the system can be taken into use.

In many cases, problems are caused by not designing the user interface at any stage, and by not testing its appropriateness until the system is implemented. The functionality of the user interface may be developed according to lists of features approved by the customer, without comparing the lists with the end users' goals or the enterprise's business processes - which have possibly not been found out at all. The user interface solutions for the features listed in the specifications may be a by-product of code-writing. In some projects, a small amount of resources are set aside for user interface design, but the analysis of user workflows or goals, which is needed as a basis for the design, has not been done at all.

Instead of testing the system and its user interface in real use only after the deployment, the design of the whole system can be grounded on the use situations and users' goals from the very beginning. In our GUIDe process model (Goals – User Interface Design – Implementation) the users' workflows and goals are determined at the beginning of the project, and workflows are streamlined as needed at this stage. After discovering the goals, the user interface designer derives the user interface solutions from the goals, and draws 'screenshots' of interaction, i.e. detailed image sequences of the functional logic of the user interface. The user interface images are an exact miniature model of the system to be implemented at the following stage, and they show the functionality available in the system.

Thanks to the miniature model, the user interface, i.e. the features, the ways to use the features, and the data contents and presentation, can be tested before starting to implement the system, making it easy to fix the problems. Furthermore, the customer is given the possibility to comment on concrete solutions from the beginning of the project. From the descriptions of workflows and goals, the customer sees what the system must be able to do, and the series of screenshots give an exact picture of how the users' workflows will be carried out with the system. In the later phases of the project, the tested user interface specification can function as input for e.g. concept analysis, database design and system architecture design.

In Chapter 2, we will consider some of the problems in process models currently in use, especially from the viewpoint of user interface quality. In Chapter 3, the phases and methods of the GUIDe process model will be proposed as a solution. In Chapter 4, the GUIDe methods will be adapted as a part of other process models, while evaluating their advantages and the challenges they pose.

2 Deficiencies in user interface design in process models

In many process models currently in use, the user interface is developed as a by-product of other activities that typically focus on implementation. In this chapter, we will look at two kinds of process models and their problems from the viewpoint of user interface design. Chapter 2.1 considers the principles for the waterfall model, and chapter 2.2 models based on an incremental development.

2.1 The waterfall model

The strictly segmented waterfall model (Figure 2.1) typically does not contain a separate design phase for the user interface, but the requirements for the functionality and usability are included in the requirements specification. When the requirements have been set, the documentation focuses on the system architecture design and other details of implementation, and the user interface is built on top of the functionality and data contents. What the finished system will look like and how it will behave for the end user will not become evident until the latter stages of the project, when the implementation is being finished.

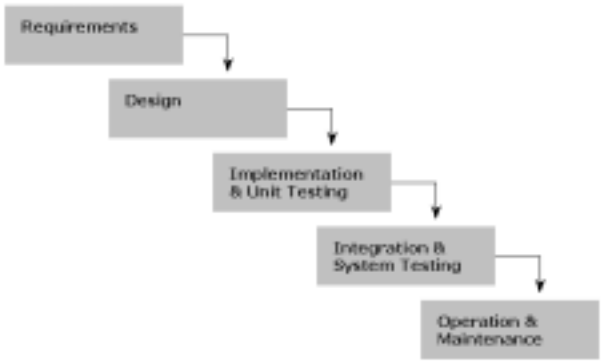


Figure 2.1. The phases of the waterfall model.

Since the user interface, including the functional logic of the program, will not become evident until the final stages of the project, it will not be possible to ensure that the system is appropriate to the users' goals and workflows until the system is implemented. In many cases, the user interface has evolved on top of the implementation, showing concepts of the implementation model to the user. The users are expected to learn the functional logic of the implementation solutions, though they are irrelevant to their work tasks, and to perform complicated and often error-prone actions that do not support their goals. In addition, it may not be possible to complete some of the tasks without the help of external aids, such as writing memos or copying and processing data in a suitable spread sheet program outside the system.

In a finished system, there are often shortcomings and problems that may be difficult to correct at this late stage; the deployment of the system is delayed, unnecessary time is spent on the problems and the errors, and the corrections are expensive. According to one estimate, the cost for changes rises by an exponent of ten for each step in the waterfall model [Boehm81]. Thus, user

interface problems should be foreseen and should be attended to at the very start of the project.

2.2 Incremental software development

Process models that are based on incremental development, such as the evolutionary model (Figure 2.2) and the Extreme Programming (XP) model [Beck00] (Figure 2.3), have tried to solve the problem of late corrections by implementing the software in stages, piece by piece. At the beginning, the project group designs and codes only a small part of the features, which are then tested. Then more features are designed, coded and tested, until the whole system is ready. The client cannot give a complete set of requirements in the beginning of the project, but the requirements gradually become evident as the system is implemented and the client begins to see what the system will be like and how it will work: "I can't tell you what I want, but I'll know when I see it" [Boehm88, p. 63].

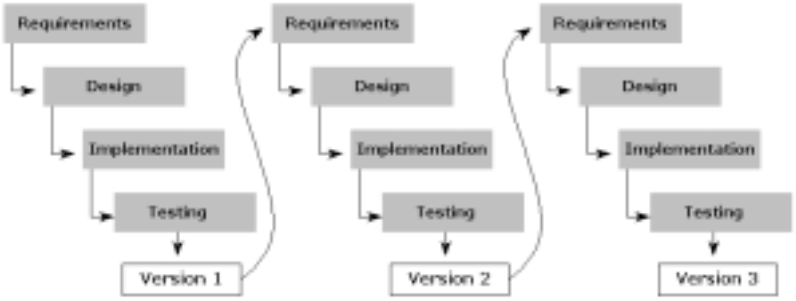


Figure 2.2. Phases of evolutionary development.

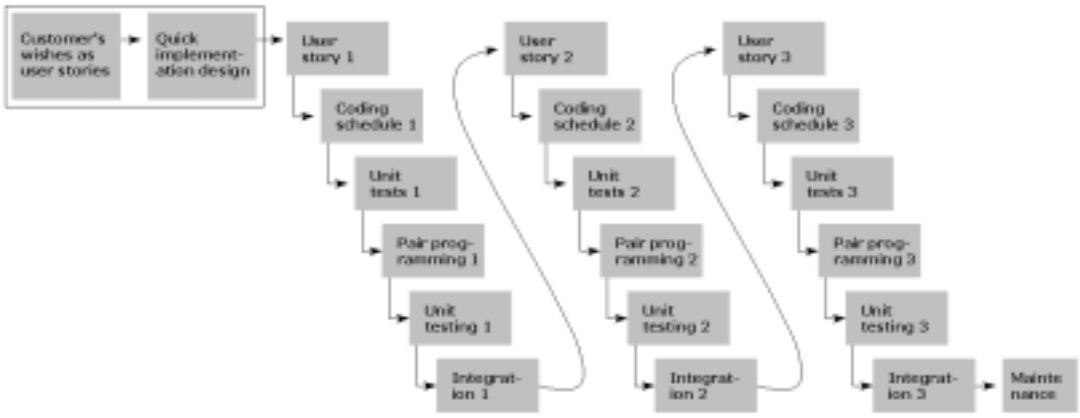


Figure 2.3. Phases of Extreme Programming.

From the point of view of developing a good user interface, the models based on incremental development are even more problematic than the waterfall model, in some sense. With the incremental models, the worst case is that the user interface is developed as a by-product of the implementation, without any proper user interface design (e.g. XP), and even in the best case, the user interface is designed one piece at a time at the beginning of each evolutionary cycle (the evolutionary model).

We cannot design a good user interface one piece at a time, because we have to be able to perform all the main tasks with the same user interface from

beginning to end. If features A and B, which are needed to perform task 1, are planned at different stages, using these features after each other may not be straightforward. The transition from one feature to another may demand a lot of navigation from the user, or the data produced by feature A is not visible at the same time as the data from feature B, though the user may need to compare the two.

Let us assume that the user needs the data produced by features A and B at the same time in task 1, and in task 2, he needs the data produced by the features A and C at the same time (Figure 2.4). For this, a good user interface design would show a visualization of the data produced by features A and B at the same time for task 1 (on the left), and the data produced by features C and A for task 2 (on the right). If the user interface is not designed to support whole tasks (goals), but instead piece by piece, the user interface will most probably show separate areas (e.g. windows, as in Figure 2.4 on the right) with the data produced by feature A on the first, feature B on the second, and feature C on the third. It may be difficult to move between the windows, and the user may not be able to see the data from features A and B at the same time. Instead, the data should be visualized and organized to support the task at hand (Figure 2.4 on the left).

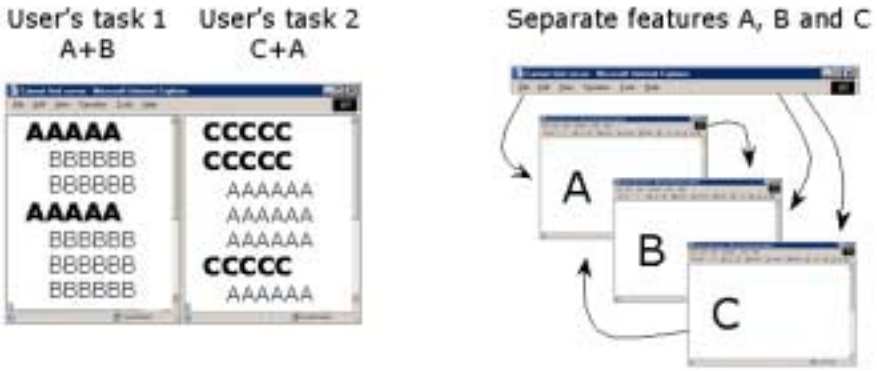


Figure 2.4. Dependencies between tasks and features.

Another problem with developing a user interface piecemeal is that when a new functional piece is added to the system, a corresponding piece must be added to the user interface, typically changing the existing user interface. If we are developing a good user interface solution that is appropriate to the users' workflows, instead of just creating disjoint functional areas, adding a new function will affect not just the existing user interface, but also the code – and not just the user interface code, but often also the implementation solutions below it.

Changing user interface solutions can, at worst, change the performance requirements or underlying system architecture. Figure 2.5 shows an example of a change made to the user interface in one evolutional cycle that greatly affects the performance requirements. In the first stage, only a few basic features have been implemented in this imaginary web-banking system, and links to them have been placed in the navigation bar on the left. With the help of these links, the user may look at his transactions, one at a time. In the second stage, the possibility to follow a stock portfolio has been added to the system, which means that it would be sensible if the navigation bar on the left would show the latest changes in the user's shares. At the same time, it is decided that the user's balances will be shown in the navigation bar. In this

way, the user may reach many goals by just opening the main view of the system and looking at the overview in the navigation bar.

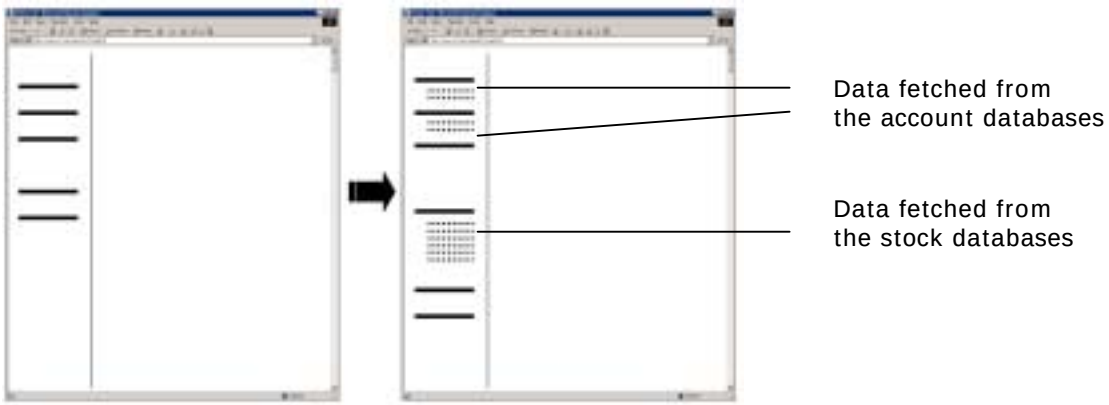


Figure 2.5. Development of a user interface between evolutionary cycles.

Let us consider the banking system in Figure 2.5 and assume that during the first evolutionary cycle, we have only implemented the functions that fetch the balance information from the bank's database for the account that the user clicks on the navigation bar on the left (the first screen in Figure 2.5). In the next evolutionary stage, the stock portfolio features are added and the user interface of the navigation bar is updated so that the latest changes in the shares are shown on the bar (the second screen in Figure 2.5). In the old user interface, a database query is aimed at one bank database at a time, and it is initiated only when the user chooses a link from the navigation bar. The new user interface demands that all the data in the navigation bar overview is fetched from all the account and stock databases when the user logs into the system, which means that multiple queries are needed. In spite of that, the system should load quickly both after logging in and during use.

Not all situations like the one above call for redesigning the main architecture, but in many situations, it would be a considerable advantage to look at the final user interface design before planning the implementation strategies, because good user interface solutions bring many new requirements to the implementation design. Requirements such as preserving the system state from one session to another, supporting incomplete input, autosave or undo features, may cause huge changes in the architecture solutions or database structure if they are added only afterwards.

3 The GUIDe process model

The problem with the waterfall model is that the actual functionality of the system that is developed becomes evident to the client and the user too late, making it difficult and expensive to make changes. The starting point for process models based on incremental development, again, is the idea that requirements cannot be established completely from the start. The system is developed piecemeal to avoid at least some of the extra work of making changes. However, this usually leads to a bad user interface that is developed piece by piece according to the customer's opinions, and a lot of unnecessary code is written during the process.

In developing the GUIDe model, our starting point is the assumption that the requirements can be established exactly enough at the beginning of the project, as long as appropriate methods are available and requirements can be described in an appropriate format. We have developed methods for GUIDe that enable us to establish exact enough requirements at the start of a project, and, additionally, solutions for the question of who will establish the requirements, in what form they are to be communicated with the customer and users, and how to test their accuracy more reliably than by asking the customer and users for their opinions.

Chapter 3.1 will give an overview of the GUIDe phases. The contents of the phases will be described in more detail in Chapters 3.2-3.5.

3.1 The phases of GUIDe

GUIDe is a process model that is founded on goal-based use cases and goal-derived design. GUIDe adds a new component into the requirements specification, i.e. goal-based use cases that are described from the users' point of view, and bridges the gap between textual use case descriptions and the user interface solution.

In GUIDe, the user interface is designed at the beginning of the project as a specification for the implementation and database design, for example. So it will be possible to test the interaction and the system's suitability to its intended use at such an early stage that any changes indicated by the tests, even large ones, will still be quick and easy to implement. The user interface specification can also be seen as a concrete requirement for the design of the implementation.

The main phases of GUIDe are shown in Figure 3.1. At the specification stage, the user interface designers analyze the users' workflows and their goal-based use cases, primarily through field studies. After that they create a user interface specification by deriving it from the users' main goals and workflows, and test the user interface with suitable evaluation methods, typically with walkthroughs and usage simulations.

When the user interface specification is ready and tested, we move on to design the implementation. At the implementation stage, instead of using the waterfall model, we can progress according to the prototyping model or via an evolutionary cycle, like XP, depending on their suitability to the case in hand.

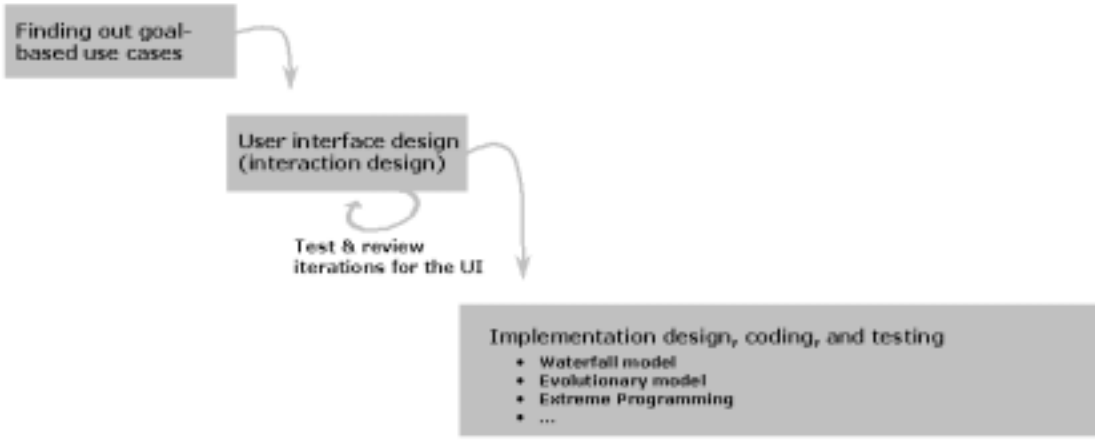


Figure 3.1. The main phases of GUIDe.

3.2 Finding out goal-based use cases

At the beginning of a project, the user interface designers will find out the users' actual workflows and goals, typically through field studies. At this stage, they do not make lists of the system's features, but instead, the functionality that will be needed will become evident at a later stage of the user interface design process.

At the field study stage, the designers will typically conduct contextual user interviews and user observations, which will reveal the existing workflows, tasks, and underlying goals. Goal-based use cases are derived from these workflows, because with workflow scenarios it is difficult to distance oneself from the current procedures and to design workflows and user interfaces that allow the users to achieve their goals more efficiently.

The designer should look for goals that are sufficiently (but not too much) above the system. The criterion is to find those goals that are just above the features of the system at hand. Figure 3.2 shows an example of a goal-based use case that has been created for user interface design of a university library. We must not stay on the level of search facilities or reservations of the books, but find a concrete goal above the system.

Use case: Card's book on information visualization in the university libraries

Goal of the researcher: Hannu Toivonen knows that in Card's book on *Information Visualization*, there are good examples of glyphs for his lecture of Research databases course.

Status data:

Times and dates
- Today it is Monday 1st September, 9.30 am.
- The lecture will be on Tuesday 9th Semptember, 10-12 am.

The book to be needed
- Hannu has skimmed the book before, but he does not have the book in his room.
- Hannu remembers that the book is called 'Information Visualization' or something like that, and one of the authors is Card.
- Full reference data: Card Stuart, MacKinlay Jock, Shneiderman Ben, Readings in Information Visualization: Using Vision to Think.

Availability of the book
- In the local library, 3 pieces, all borrowed.

- In the library of Faculty of Medicine, there are 3 pieces: 2 borrowed, 1 available. Hannu has not visited the library before and does not know the location of it.
- In the library of Faculty of Behavioural Sciences and the library of Psychology, there are 2 pieces, all of them borrowed.

Figure 3.2. An example of a goal-based use case (abbreviated).

When creating goal-based use cases, it may first seem that defining so concrete use cases is futile, since there appears to be dozens or even hundreds of different use cases. In reality, however, the use cases can be categorized into an amazingly small but representative set that covers the application area surprisingly well. To design a reservation system for movie tickets or a journey planner for the public transportation in Helsinki, a total of 3 to 6 use cases from different categories seem enough for designing the user interface and functionality of the system. However, at the beginning of the project we need to find out a much larger number of goals, because after a closer inspection many of them will turn out to be from the same category, i.e. they are just variations of the same use case.

3.3 Goal-derived user interface design

After finding out the users' workflows and writing the use case descriptions, the workflows and enterprise's business processes are streamlined. For example, the workflow of the previous use case (Figure 3.2) can be streamlined. Today, the user, Hannu, can achieve his goal by making a reservation for the book, and then picking it up in the library of Faculty of Medicine. However, by changing the current workflow, we can allow him to get the book with the university's internal mail service, and make more straightforward design. In the streamlined workflow, we do not need a feature for making reservations, but instead, we should design an 'order via internal mail' feature and show estimated time of delivery. The costs of the new solution can even be lower than those of the current workflow. In some cases, the result of streamlining the processes is that some parts of the system become superfluous, and sometimes even a whole system turns out to be unnecessary and it does not need to be unimplemented.

The design process of the user interface begins with selecting the first goal to be supported. The designer selects the goal (a high-level use case) and creates the functionality and the user interface solution needed to reach it. However, he should avoid adding any features or data that are not specifically needed to perform this use case. Then the designer gradually integrates support for new use cases into the user interface. As the design process advances, the designer will continuously simulate the user interface by performing one use case at a time. In this way, we can verify that the more recent use cases have not made it more difficult to perform those that were used earlier in the design.

The system can not end up with features that are not needed to perform the use cases, because during the design for each use case, we only create functionality and user interface that support that one goal. If the design team or customer comes up with a potentially useful feature during the design, the user interface designers must examine it: is there a goal-based use case that has not been attended to and for which this feature is needed? If no such use case is found, the feature is documented to be useless until somebody is able to provide a use situation where the feature would bring additional value to the user. On the other hand, if a previously unattended use case is found, it is included in the requirements specification and in the design. This typically

results in other changes as well, in addition to including the suggested new feature.

The user interface design process of the GUIDe is very similar to XP implementation iterations [Beck00], but XP's user stories have been substituted with goal-based use cases that are defined by user interface designers. User interface design is done in pairs (cf. XP pair programming) in such a way that the features required for the use case are always added on top of the previous solution and any necessary changes to the previous solution are made at the same time. A user interface solution is built incrementally, but during the process it requires large corrections at times (refactoring). The use cases are written beforehand (like XP's unit tests) and all the tests are run after adding the new functionality for each use case. The difference between the two processes, XP's software development process and GUIDe's user interface design process, is that in Guide's user interface design, the development cycles are even shorter than in XP, and experts are responsible for the use cases, not the customer, though in the end the customer approves the goal-based use cases and their prioritization.

When the first version of the user interface is designed entirely, the designer will draw up a user interface specification that shows, with the help of detailed image sequences, how the users progress towards their goals step by step, i.e. how they complete their workflows. The paper prototypes created during the design process, as well as the final specification can be tested with suitable review methods (chapter 3.4).

3.4 Review of a complete version

After designing a complete version of the user interface according to the incremental process described in Chapter 3.3, the design should be evaluated on the level of system testing, before moving on to designing the implementation. The main goal of the review should be to make sure that all the needed features and data for completing the workflows have been included in the user interface (*utility*), and the interaction is *efficient*. It is only when the functionality and efficiency aspects have been secured that it makes sense to start fixing any problems related to learnability and intuitiveness, because learnability is a lot easier to add on top of valid functionality and efficient interaction, than vice versa.

A cost-effective way to find out problems with functionality and efficiency is to conduct walkthroughs with users [Bias91] that can be arranged already during design, when the first draft of the whole user interface has been completed. At a walkthrough meeting, a couple of users are typically present with 1-2 user interface specialists, one of whom shows screenshots of the system and steers the walkthrough session in appropriate directions for gathering feedback. During the walkthrough, shortcomings and misconceptions of the user's tasks and workflows will be found out. In addition, the walkthrough typically reveals some exceptions of the tasks that have not been found during field studies. Learnability problems are not so important in this phase.

Another good review method for finding out efficiency problems is an expert review based on simulation of workflows. By usage simulations, the designers can remove unnecessary functional steps and stages of mental processing from the user interface even before running any user testing. Because the review process does not include real users, the shortcomings of the functionality and the data will not be found based on domain knowledge. Instead, a lot of open

questions will turn up, and these questions are the key to missing tasks, data, and functionality. Afterwards, the designers ask the users explanations for open questions that have been found during the simulation.

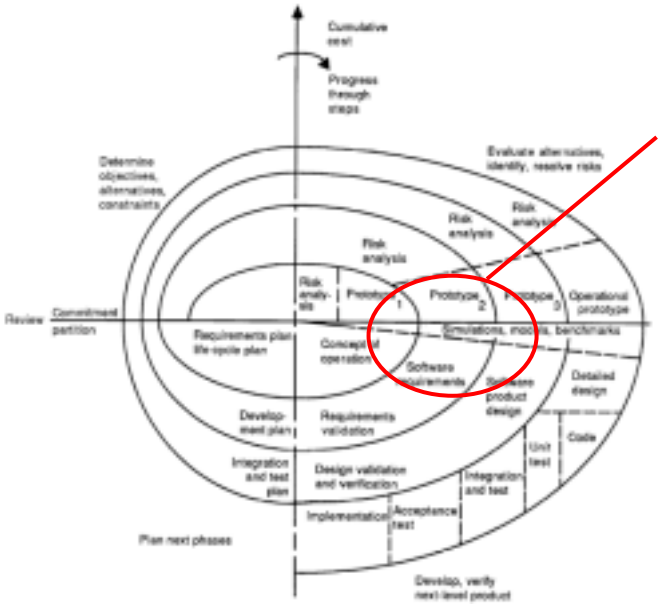
It is often difficult to predict the learnability and intuitiveness of the user interface without the help of the users. Small aspects such as the locations of objects, visual details and word choices can change the user's interpretation in an unexpected way. *Usability tests* [Nielsen93, Chapter 6], where one user at a time completes test tasks set by the designer, are good for bringing forth any learnability problems.

After each evaluation cycle, the problems that were discovered are fixed and the user interface specification is updated. The user interface solutions and system functionality still only exists as images that can be quickly changed according to the test results. However, the goal is not to iterate for the sake of iterating. Instead, we aim to minimal iteration, i.e. try to make the user interface solution so good that it does not need a great deal of correcting at the testing stage.

3.5 Designing, coding and testing the implementation

After the use case specifications, user interface design, and reviews, all completed at the beginning of the project, we move on to the implementation phase. This includes at least implementation design, coding and testing, and in many cases things such as concept analysis, object modeling and database design, as well. The implementation phase can follow the waterfall model if the implementers have enough experience with creating similar systems, or it may follow the evolutionary model or the XP model of cycles that add new functions in stages.

What all the alternatives have in common is that the user interface specification is given as input for the specifications at the implementation stage. The goal of this process for creating the user interface specification is to lower the risk of producing a wrong kind of user interface as much as possible. This is one of the greatest risks in designing an interactive system [Boehm88]. Figure 3.3 shows the Boehm spiral model with the most important GUIDe methods added. Adding the use case analysis and user interface design to the beginning of the project, similarly to the GUIDe model, will lessen user interface risks and take the project in the direction of the waterfall model. The latter part of the project may be completed according to prototyping of implementation techniques or according to the waterfall model, for example, depending on the remaining risks.



Field studies for use cases and user interface design in GUIDe

Figure 3.3. The spiral model of Boehm [Boehm88].

Even though an exact design of the user interface has been specified at the beginning of the project, during the implementation design and code-writing some compromises often have to be made to the user interface. Some user interface solutions that are good from the user's point of view may turn out to be too laborious to implement with the tools available in the project, for example, so it makes sense to lessen their usability a little in order to save on expenses. When a project following the GUIDe model comes upon a need for compromise, the implementation designer or the programmer describes the problem to the user interface designers and delegates the design of a new compromise solution to them. In this way, we can ensure that the programmer does not have to start designing the user interface, a job that they usually have no experience with. If the user interface designers design the implementation compromises, too, they should not end up containing any new big usability problems.

Especially in those cases where the general public will use the system, we recommend a final usability test with a few test users. At the end of the project, some small features affecting learnability can still be corrected (such as moving related components closer to each other, selecting intuitive labels). They can be corrected quickly but can affect the user's interpretation.

4 Applying the GUIDe model to a project

The main improvement that the GUIDe model offers is the user interface design phase, which is scheduled at the beginning of the project and leads to an exact user interface specification. Chapter 4.1 will consider examples of how this method is added to the waterfall and XP models. Chapter 4.2 will evaluate the advantages of GUIDe and the challenges it brings to the whole project.

4.1 The phases of GUIDe as a part of the waterfall and XP models

If we schedule the user interface design and the analysis of users' goals as their own phases at the beginning of the project, we can solve the problem of finding out the need for changes at too late a stage in the waterfall model (Figure 4.1). The descriptions of users' goals and the user interface images are the part of the system that the customer and the users will understand, and iteration with the help of screen images is very inexpensive compared to iteration with the help of code. The greatest and most expensive risks shift from the requirements specification and user interface design into the implementation solutions.

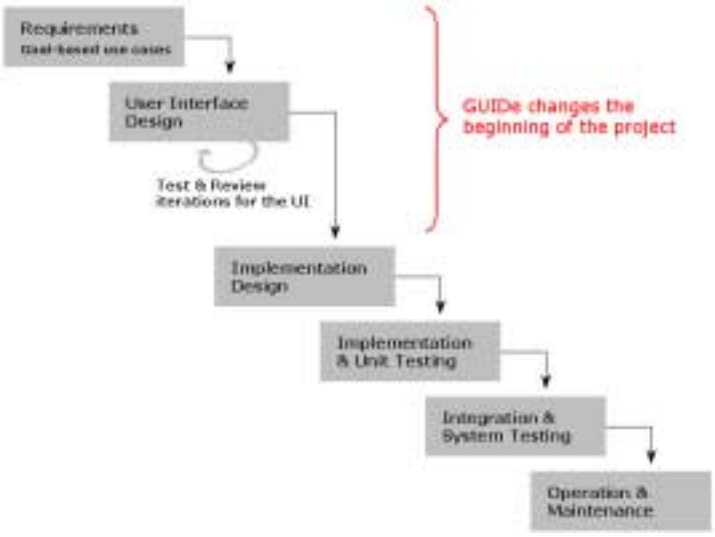


Figure 4.1. GUIDe as a part of waterfall model.

GUIDe can also be adapted to models based on incremental software development. In Figure 4.2, the GUIDe phases have been added to the XP model.

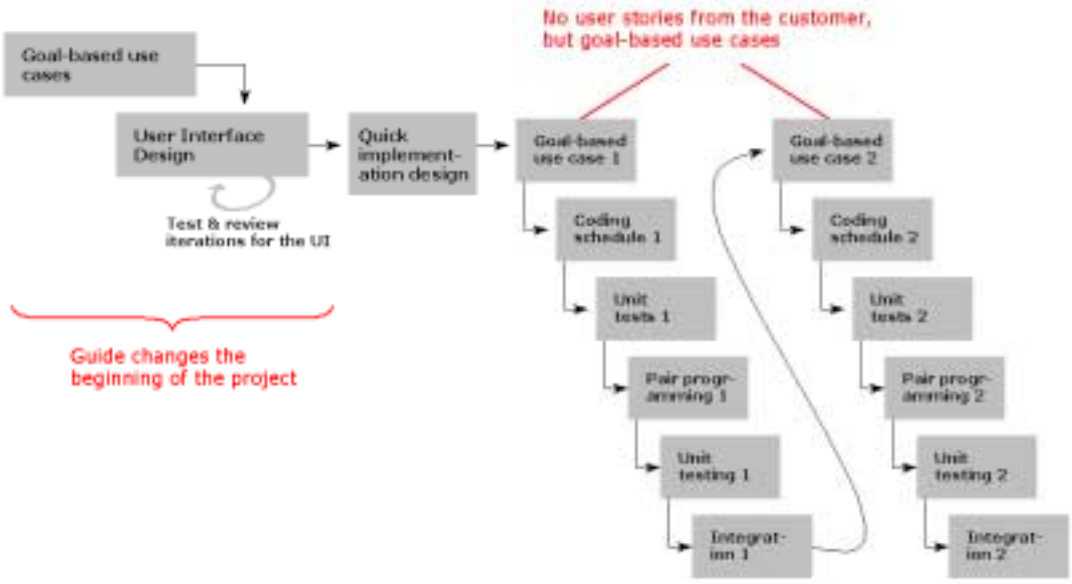


Figure 4.2. GUIDe as a part of XP-model.

As we can see in Figure 4.2, the use case analyses and user interface design in GUIDe have been placed at the beginning of the project. An additional difference to the original XP model is that the responsibility for defining the requirements is not left to the customer, who would create requirements by writing user stories. Instead, the user interface designers will analyze the end users' goals, workflows, business processes and the customer's needs with the appropriate methods. Of course, the customer should also be involved in analyzing workflows and prioritizing use cases, especially if representatives of the customer will be end users of the system, but the whole responsibility for finding out workflows should not be left to the customer. Typically, the customer's representatives does not have the know-how or the resources for making such analyses.

When the project moves on to testing and implementation cycles according to the XP model, the system is implemented one use case or partial use case at a time. The main rule is that functionalities for use cases with the highest priority are implemented first so that it will be possible to perform useful goals or parts of them with the system at an early stage.

GUIDe differs from XP in that GUIDe tries to reduce the likelihood of missing requirements to the bare minimum with the help of the user interface design, whereas the philosophy of XP is that this problem cannot be avoided, i.e. a sufficient set of requirements can never be defined before the implementation starts. Since, according to XP, the problem cannot be solved, its effects are minimized by adapting the whole project for ever-changing requirements. If the procedures of GUIDe are used before the short design stage of XP (Planning Game), we can minimize the risk of changes in requirements during the project and at the same time enhance the quality of the user interface. The implementation can, however, be in many cases most cost-efficient if it is completed according to XP.

4.2 Advantages and requirements of GUIDe

One of the greatest advantages with GUIDe is that it gives a systematic method to find out the functional and data requirements of a system. In addition, the interaction is designed at such an early stage that the user interface can be tested before coding, and costs for fixing the design are low. The use case simulations and usability tests performed when evaluating the user interface are considerably more reliable methods than e.g. the opinions of the customer.

The users and clients are very well able to evaluate the goal-based use cases that are set as the starting point for user interface design, because they describe the users' day-to-day work, on which the users are the best experts. Workflows can be streamlined before starting to design the user interface, and that may have a much larger impact on the end result than any user interface or implementation solutions. In some cases, unnecessary phases and even whole computer systems can be removed from the workflow.

Thanks to the exact user interface specification, the communication with the customer and the end users becomes more efficient, as the image sequences will show them what the new system will look like and how it will work. It is easier for the customer's representatives to review a concrete user interface specification than long lists of written requirements, where it is difficult to evaluate and ensure that they are extensive enough.

A precondition for using GUIDe is that the project team knows how to perform goal-based use case analyses and how to design user interfaces derived from goals. In addition, the team should know at least some method to evaluate user interface solutions. If the team does not know enough about user interface design, scheduling the user interface design phase at the beginning of the project will not necessarily eliminate user interface risks sufficiently. However, development projects for interactive systems are well-advised to use the GUIDe approach of specifying the user interface at an early stage, because the user interface solutions created as a by-product of programming would hardly be better than a user interface specification created at the start of the project, which can be tested in some way before the implementation starts.

One of the disadvantages of GUIDe is that the user interface design enters the critical path of the whole project. This problem can be alleviated by performing implementation experiments and creating implementation prototypes on those parts of the system that are known to be critical or that are not well understood, in parallel with the user interface design.

With GUIDe, there is also a risk that some important but rarely occurring use cases are overlooked at the beginning of the project, so that the user interface is not designed to support them at all. If that happens, the problems will not appear until the system has been taken into use and the users are trying to perform real use cases with the program. On the other hand, the same risk exists to a much greater extent for all projects that rely on requirement specifications.

In projects based on evolutionary cycles, the problem of overlooked use cases would be discovered before implementation only if one of the project parties happened to come to think of the missing requirement in the midst of the development process, or if it became evident in one of the versions implemented after a development cycle. With GUIDe, we try to find any deficiency by demonstrating a miniature model of the whole system to the

customer, and by testing the user interface specification on the users at the beginning of the project. In this way, the majority of possible deficiencies should be caught before implementation. If the project team does not know enough about how to find out use cases, how to design the user interface or how to evaluate it, user interface risks can first be minimized with the help of the GUIDe methods, and then an implementation iteration according to the XP model, based on a user interface specification, can be used. In this way, GUIDe can be used to minimize the risk of missing requirements at the beginning of the project and to improve the quality of the user interface, and with the help of the XP implementation iterations, any deficiencies left can be caught at an early stage.

It is simple to leave the responsibility for the completeness of the requirements and the evaluation of user interface solutions to the customer, as in XP, but the disadvantage is that it does not guarantee a good user interface or appropriate functionality of the system. The customer typically does not know how to evaluate a user interface nor does the customer master any design methods, so he is not competent to say which user interface solution is good and which is not. Neither can the customer know what requirements should be discovered for user interface design, nor does he typically know how to make use case analyses or have time for them. However, the customer and especially the end users are the best experts on the use of the system and its workflows and goals. GUIDe utilizes their expertise in these fields but does not try to hand over the responsibilities for tasks that they do not know well enough.

5 Conclusion

In many process models currently in use, like the waterfall model or the evolutionary model, user interface design has been barely acknowledged or it is non-existent. As a consequence, interactive systems that have been designed according to these models have inefficient and obscure user interface solutions, as well as deficiencies in the functionality and data contents that are needed to perform the tasks the system was created for. These systems have not been designed to support the user's workflow; instead, their suitability to performing the users' goals is not tested until the system is deployed, in the worst cases.

The starting point for the development of some of the process models, like XP, has been the assumption that it will not be possible to establish the requirements exactly enough at the beginning of the project, because the customer will not be able to say what he wants. Thus, the system is developed piecemeal and the customer is asked for feedback and suggestions during the development process. This method does not decrease the amount of problems with the user interface either, because programmers seldom know how to design a good user interface and the customers do not know how to evaluate it, so any amendments are usually based on the customer's opinions. In addition, the iteration demanded by the customer in these models is usually made with the help of a working program or a prototype, leading to unnecessary code-writing.

Instead of accepting the problems caused by deficient requirement specifications and the changes demanded by the client, the GUIDe model tries to solve these problems. When defining the GUIDe model, we have found out that the appropriateness of the system behavior that is visible to the user (i.e. the user interface) is the highest risk in the development process of an interactive system. Therefore, we have developed a systematic method to eliminate that risk.

A user interface design phase is scheduled at the beginning of a project. During this phase, required functionality and interaction are derived from goal-based use cases, which are discovered through field studies. With the help of the resulting user interface specification, we can test already before designing implementation solutions whether the specified system will support the users' goals:

- does the user interface contain the functionality and tha data that are needed in the real use situations (utility), and
- are the user interface designs straightforward, efficient, and learnable (usability).

References

Beck00	Beck K., Extreme programming explained: embrace change. Addison-Wesley, Reading, MA, 2000.
Bias91	Bias R., Walkthroughs: Efficient Collaborative Testing. IEEE Software, Vol. 8, No. 5, 1991, p. 94-95. [pdf]
Boehm81	Boehm B. W., Software Engineering Economics. Prentice Hall, 1981.
Boehm88	Boehm B. W., A Spiral Model of Software Development and Enhancement. IEEE Computer, Vol. 21, No. 5, May 1988, p. 61-72. [pdf]
Nielsen93	Nielsen J., Usability Engineering. Academic Press, San Diego, CA, 1993.

Updated October 3, 2004 / Sari A. Laakso, email: salaakso@cs.helsinki.fi