Planning a project for real-time systems for energy production planning using a Usage-Centered Design
Project Planning for

Real-Time Systems for Energy Production Planning

Using Usage-Centered Design

Book: Software for Use
Lockwood, Mayhew, Beyer & Holtzblatt
Table of content

INTRODUCTION ................................................................................................................................................. 4
USABILITY TERMS .............................................................................................................................................. 5
PROJECT PLAN ................................................................................................................................................ 6
COLLABORATIVE REQUIREMENTS DIALOGUE .............................................................................................. 6
  GATHERING REQUIREMENTS ......................................................................................................................... 6
  OUR PARTICIPANT ROLES USING THE JEM PROCESS: ............................................................................. 7
TASK MODELING ............................................................................................................................................... 8
  DEFINING USER ROLE MODELS AND ROLE MAPS - USER ROLE MODELS ........................................... 8
  BUILDING THE STRUCTURE ............................................................................................................................ 9
  USER ROLE MAPS ......................................................................................................................................... 9
  DEFINING ESSENTIAL USE CASES AND USE CASE MAP ........................................................................ 10
INTERFACE CONTENT MODELING .................................................................................................................. 11
IMPLEMENTATION MODELLING ...................................................................................................................... 12
USABILITY INSPECTION .................................................................................................................................... 13
  INSPECTION ROLES ....................................................................................................................................... 13
TESTING ............................................................................................................................................................... 14
INNOVATIVE INTERFACES ............................................................................................................................. 15
ONGOING PROCESSES DURING THE WHOLE PROJECT .............................................................................. 16
  HELP SYSTEM AND DOCUMENTATION ........................................................................................................ 16
  FITTING THE OPERATIONAL CONTEXT ........................................................................................................ 16
CONCENTRIC CONSTRUCTION ....................................................................................................................... 17
ARCHITECTURAL ITERATION ......................................................................................................................... 17
TIMETABLE ......................................................................................................................................................... 18
ANALYSIS ........................................................................................................................................................... 19
APPENDIX A ...................................................................................................................................................... 20
Introduction
The authors define certain activities, which are often overlapping, to perform while designing a project.

Figure 1. Tasks during system project

The shapes in the figure above represent the activities of usage-centered design. The boundaries separating the activities aren’t always delineated; activities are in other words often overlapping. The time flows from top to bottom but it shouldn’t be confused with the ever so bad ”waterfall” software development cycle. The process begins with a trio of activities (the blue and red shapes). These are aimed at establishing the basic nature of the requirements of the system. The upper blue shape - **Collaborative Requirements Dialog** - is a negotiation between developers and their users to establish requirements and needs of the system to be designed. The lower blue shape, also known as the heart of the trio, is the **Task Modeling**. This is considered as the very core of usage-centered design process. The purpose is to develop a clear and complete picture of the work to be supported through role and task models. Task modeling interacts with another design and development that is outside the immediate scope of usage-centered design, namely **Domain Modeling** (red shape), which is a representation of all the interrelated concepts and constructs in the application domain. Domain modeling establishes the vocabulary of the system and its operations. Task modeling overlaps with **Interface Content Modeling**. Experienced designers may develop both task models and content models in concurrent modeling activities. This action overlaps with **Implementation modeling** which is the process of detailed design and prototyping. **Usability inspection** appears twice in the process model, once before Concentric Constructions and Architectural Iteration (two yellow shapes), and once after. Because the final goal of a system development is a working and usable system, the design and refinement activities must eventually lead to actual construction and testing of working system. Concentric Constructions and Architectural Iteration constitute the implementation phase of usage-centered development. Concentric Constructions is a process for
developing working systems in layers. Architectural Iteration is a method for maintaining sound internal software architecture as successive layers are added to a system.

Running parallel with all the modeling and design activities are two complex and specialized activities: **Operational Contextualization** and **Standards and Style Definition**. The purpose of Operational contextualization is to adapt the design to the actual operational conditions and environment in which the system will be deployed. Task organization need to be the primary factor shaping the interface of the architecture, with operating context as a secondary influence.

Users are involved in many of the above mentioned steps, namely during the Collaborative Requirements Dialog, Task Modeling, Domain Modeling, Usability inspection and Standards and Style Definition.

**Usability terms**

*According to the authors’ opinion it would be misleading to call usage-centered design a methodology, it is rather a collection of coordinated activities that work together that contribute to software usability.* The activities in usage-centered design are a concurrent engineering process where both independent and separable activities are carried out in a parallel whenever found practical and suitable.

(1) **Utility** means that a system does something worthwhile, something that is in itself of sufficient value to justify the investment in equipment and programming. To make useful software we have to choose appropriate problems to solve. (2) **Capability** is another factor that is required for creating usable systems. A system must be capable of doing what it’s supposed to do. Capability and Utility are indeed two pivotal ingredients, but without usability they tend to be useless. (3) **Usability** is the most important subject according to the authors. It is a far more complicated term than “utility” and “capability” because it is about systems in interaction with people, a point where matters become muddier and messier. The five facets of usability are:

- Learnability
- Rememberability
- Efficiency in use
- Reliability in use
- User satisfaction

Highly usable systems are easy for people to learn and help people to work efficiently while making fewer mistakes and boosting their productivity. A highly usable system make people feel good about using them, like a good tennis racket, or an intuitive ticket booking system, or a finely written book about usage-centered design, just to name a few.

To design dramatically more usable tools, it is not users who must be understood, but usage. Usage-centered design focuses on the work that users are trying to accomplish and on what the software will need to supply via the user interface to help them accomplish it.

Further definitions about usability are provided in Appendix A.
**Project Plan**

Our client is an energy producer who uses real-time systems for energy production planning.

Our client is relying on correct and updated information (that of course also should be easy to interpret correctly). They want to optimize their plans both for the next day and for the future to maximize their revenue. Simply, they want to produce more energy when they could sell it expensive and produce less when they have to sell it cheap, but they also must fulfill the directive they got. Since the selling prices for energy constantly changes (like an ordinary stock market) and the producing costs for the energy varies, this is a complex task to be successful with. Especially as there is an aspect of time involved, the decisions must not be too time-consuming (though not as fast as at an ordinary stock market).

To achieve this they have to be aware of their own situation and about how high their instant production is, as well as the future conditions including both coming days and in a longer perspective. They also have to know about the surroundings, what the demands of the market would be and how much other producers will produce. Once a day they also must send information about their producing plans for the next day.

We assumed the numbers of employees using this particular system are few and all of them are expert users and perform similar tasks.

The rest of this report is about the activities we plan to perform.

**Collaborative Requirements Dialogue**

According to authors, far better results are obtained when users are intensively involved in setting requirements and in other carefully targeted activities, such as collaborative usability inspections, but are not heavily involved as participants in the actual designing of the user interface.

The most effective use of users involves them at a small number of very specific points in the development process toward very specific ends. It aims to make efficient use of their time and of the developers’ time. The usage-centered activity model is user-involved rather than user-centered.

**Gathering Requirements**

Besides observation of users at work as well as interviews and discussions, we will use Joint Essential Modeling (JEM) to define the requirements.

Here is a quote from the book that explains the meaning of JEM:

“Joint Essential Modeling, or JEM, is a structured process for collaborating with users to develop usage-centered requirements specifications through concurrent modeling. In concurrent modeling, the various necessary models are developed more or less simultaneously, moving back and forth among role, task, content, navigation, and implementation models.”

The primary deliverables from using the JEM process in our project are:
- User role model
- Use cases, with narratives in concrete form
- Focal use case

Optional deliverables that we will include:
- Essential use case narratives
- Use case map
- Use case prioritization
Jointly developing the user role and use case models is the basic objective of JEM.

Our Participant Roles using the JEM process:

1. Sponsor: A sponsor, who may be a customer, a user supervisor, or a project manager, is desirable. A participating sponsor begins and ends modeling sessions but does not actually conduct them.
2. Users: Once candidate user roles have been identified, all nominated user roles should, if possible, be presented among users.
3. Process Roles:
   - Facilitator: The facilitator functions as a neutral process leader.
   - Scribe: The scribe will track the full process of modeling and decision-making. The scribe must have a background in software development as well as knowledge of the conventions of the joint modeling process and familiarity with the models being developed.
   - Lead analyst: A lead analyst will be designated to assure appropriate technical leadership and expertise in usage-centered modeling. The lead analyst serves as a consultant and resource to the group on technical matters concerning modeling and usage-centered design.
4. Other potential participants: If possible, other useful contributors like quality assurance staff or testers, user interface designers or other usability professionals, trainers, and documentation specialists can be included.

Activity Overview
1. Pre-modeling and consolidation
2. Role modeling
3. Task modeling
4. Model auditing
5. Feature allocation

These activities are carried out in a series of sessions, which we refer to as follows:

1. Framing session
   The framing session collects and consolidates available information, both in preparation for modeling and in order to generate a list of candidate user roles.
   The deliverables from this process are:
   - Draft statement of essential purpose for system
   - Preliminary list of candidate user roles
   - List of participants for subsequent modeling sessions

2. Modeling session
   To develop a structured user role model, two separate sessions will be needed.
   The deliverables for the role-modeling phase include:
   - Final statement of essential purpose
   - Structures user role model
   - User role map
   In this session, the draft statement of essential purpose will be presented to the group for discussion, review, and agreement. Also, the list of candidate user roles from the framing session will be reviewed and revised.
   All use cases will be identified in task modeling phase. The deliverables from this phase include:
- List of use cases with essential purpose indentified
- Use case narratives (in concrete and essential form)
- Use case map
- Identification of focal use cases

3. Review session
   - Auditing
     The goal of auditing is to ensure completeness and correctness while also simplifying the model.
   - Allocation
     The primary focus of allocation is to decide what features and capability will be immediately implemented in the system and which might be deferred to later releases or even permanently omitted.

Task Modeling

Defining User Role Models and Role Maps - User Role Models
All that we have learned from our users needs to be confirmed with them to make sure that we are on the right track. We can then implement what we’ve learned into the software we are to build. In other words, a user role is an abstraction of needs, interests, behaviors and other characteristics of a potential user. To be able to define role models we would have to use Structured Role Model method. This method would help us to understand the relationship between the users and the planned systems operational context - the environment and context in which the software will be used.

We will use the following profiles to define User Roles in our project:

1. **Incumbents**: aspects of the actual users who will play in a given role.
   The important information falls into three categories:
   - **Domain knowledge** - Application domain that the system supports.
   - **System knowledge** – How much users in this role can be expected to know about the system itself, how it operates, and how to use it.
   - **Other background** – Any potentially relevant information about training, education, experience, intelligence, or sophistication of users.

2. **Proficiency**: how usage proficiency is distributed over time and among users in a given role. The proficiency profile refers to the level of skill (as distinguished from the knowledge) in operation of the system by users in a given role. It can range form novice to intermediate to expert levels of usage.

3. **Interaction**: patterns of usage associated with a given role.
   The role interaction profile includes sundry information about the typical or expected patterns of usage of users in this role, including such things as the frequency and periodicity of interaction:
   - **Frequency**: How often will the user take on this role?
   - **Regularity**: Is there a regular period or is usage more or less sporadic?
   - **Continuity**: Is interaction within this role essentially continuous or is it more intermittent?
   - **Concentration**: Is usage concentrated into bursts or batches or is it more distributed?
   - **Intensity**: What is the rate of interaction?
   - **Complexity**: How complex are interactions within this role?
- **Predictability**: Are interactions within this role more or less predefined and predictable or are they highly variable?
- **Locus of control**: Is the interaction driven by the process or by the user?

4. **Information**: nature of the information manipulated by users in a role or exchanged between users and the system. Where information originates and how it flows between user and system has important implications for user interface design. This includes:
   - **Input origins**: Where does the input from the user in this role originate? What is its ultimate or actual source?
   - **Flow direction**: Does information flow predominantly from or to the user?
   - **Information volume**: How much information is available and of interest to the user?
   - **Information complexity**: How complex is the information available and of interest to the user?

5. **Usability criteria**: relative importance of specific usability objectives with respect to a given role.
   Each of the aspects of a user role embodied in the other profiles can affect the relative importance of various design objectives or usability criteria profile. In addition to core usability criteria, other criteria often come into play. These include accuracy, clarity of presentation, comprehensibility, and flexibility of operation. For example, large amount of information and complex interaction may suggest that ease of navigation needs particular attention in the user interface design.

6. **Functional support**: specific functions, features, or facilities needed to support users in a given role.

7. **Operational risk**: type and level of risk associated with the user’s interaction with the system in a given role.
   *Operational risk profile*: an assessment of the potential costs or consequences of error or failure in the use of a system.

**Building the structure**

We construct a user role model by asking questions – questions of ourselves, of our users, or of the available informants:

1. Who would or could use the system?
2. What is the general class or group to which they belong?
3. What distinguished how they would or could use the system?
4. What characterizes their relationship to the software?
5. What do they typically need from the software?
6. How do they behave in relation to the software, and how do they expect the software to behave?

It is easier to start with a particular user in mind and then generalize toward the abstract role. Our approach to constructing a basic user role model is to begin by brainstorming a list of candidate roles with a minimum of discussion or details. Then, sorting through it, arranging similar or related roles into groups, considering ways to simplify and generalize the model, looking for opportunities to eliminate overlap or duplication.

Next, we should fill in the details, describing the characteristic needs, interests, expectations, and behaviors of each role.

Then, we will define Focal Roles. Focal roles are those few user roles judged to be the most common or typical or that are deemed particularly important from some other perspective.

**User Role Maps**

A broader understanding of the system is often useful. By creating a User Role Map you can capture the big picture, how all roles fit together and how they are interrelated within the system.
Within a user role map, roles can be interrelated in several different ways: by **affinity** (recognized similarity or resemblance, represented by a dashed line between the roles), by **classification** (subtype of another, represented by a double-lined arrow), or by **composition** (combine the characteristics or features of two or more, represented by a simple arrow).”

When creating the map we will start with identifying the affinity relationships and derive the other more specific relationships among user roles where these are identifiable.

**Defining Essential Use Cases and Use Case Map**

From the previous modeling session we will get a user role map and focal user roles as inputs to this modeling session. The main goal for us in this task modeling session is to understand the work involved in a particular user role. We prefer to use the concept of essential use cases, and not the traditional use case, which is divided into two parts: the user action model (shows what the user does) and the system response model (shows what the system does in response). The problem with using the traditional use cases is that they contain built-in assumptions about the system and the user interface that have not been created yet.

Instead of focusing on the actions we want to focus on the intentions of the user. The essential use case is based “on the purpose or intentions of a user rather than on the concrete steps or mechanisms by which a purpose or intention might be carried out”. This model consists of three components:

1. Statement of the overall user purpose or intention
2. User intention model
3. System response model

The model name comes from that it only includes essential steps that are of intrinsic interest to the user. Therefore it is shorter than the conventional use case.

Back to the process of building use cases, the first step of the process is that our group has a brainstorming session where we examine the user roles given from the earlier session. During this examination, candidates for use cases will emerge. First we will only write down what distinguishes the use cases from one another using the information, and after that we review and refine it. There will probably be a number of possible solutions; so we will put some effort and time to making it as simple and well reasoned as we could. Then we, starting with the focal ones, write the use cases in our list as structured narratives (containing the user-system interaction explained above). Using these, we try to reduce them into an essential form to avoid built-in assumptions about the system, through shifting focus from actions to intentions. We must always try to focus on the essential use cases and users intentions. One way to do this is to, throughout the brainstorming process, ask ourselves questions like:

- What are the users in this role trying to accomplish?
- To fulfill this role, what do users need to be able to do?
- What capabilities are required?

These questions will help us identify the different essential roles that users may have.

After the use case-session, where we have defined our use case models, we will have a workshop together with the appropriate amount of users in which we check with the users whether we have got a properly understanding. Hopefully it is easy for the users to understand them, because it is their work roles that we have tried to identify, and therefore it would be possible for them to verify them. It is important that we inform the users that the goal of the modeling is to represent the
purpose of interactions and not the mechanisms, because sometimes the users might find the essential use case oversimplified.

After identifying the essential use cases and having them reviewed and approved by the users, we will construct a use case map.

Use cases can be interrelated in a use case map made by separating out distinct interactions and showing their relations. The relationships that we will be looking for when examining the use cases are:

- Specialization: some use cases are specialized versions of others.
- Extension: Use case can extend another use case if it represents inserted or alternative patterns of interaction within the course of the use case being extended.
- Composition: Use case can be decomposed into component parts or sub-cases representing subordinate or included patterns of interaction.
- Affinity: represents relationships between use cases where the exact nature of that relationship is unclear, or ambiguous.
- Focal use cases: The focus of attention around which the user interface or some portion of the user interface will be organized.

Outputs of these sessions are a list of use cases with narratives, essential use cases and a use case map. These will be needed for the next step of the modeling process, called Interface Content Modeling.

**Interface Content Modeling**

The inputs to this modeling process are the use case map and the narratives from the previous session.

When we are satisfied with our use case map and narratives we will start designing the interface through abstract prototyping. This means we want our designers to get a better picture of the user’s work, before our programmers start coding visual prototypes. Abstract prototyping includes identifying what content is needed, in forms of tools, information and things like that, and how this should be organized to support workflow and efficient use. Doing in this way, we get a good chance to consider our plan for the implementation step later on, and it is easy for us to change bad solutions before they are coded.

The interaction content defines what tools and materials the user would need to fulfill a use case. When creating the interaction context model we use the relationships between the use cases and the narratives about them, which we identified when creating the use case map. If a use case is unrelated to other use cases, it will have its own interaction context, but often use cases contain overlapping procedures or are related in some other way and then the best solution could be to support them with a common interaction context. There could also be a reason for splitting a use case in two or more interaction contexts, if the use case is very long or complicated.

To perform this, we will use post-it notes since they are rather uncomplicated to use and makes it easy to experiment with different solutions without making new sketches. It also helps us avoid stereotypes and prevents us from locking up with a specific solution. Furthermore, the users will not interpret post-it notes as the actual interface and therefore it will help them to keep focus on what content is needed and not care that much about how it would look like in the end. We will start with defining what information is needed for the actual use case, and after doing that we conclude what
tools is needed. This information is represented with notes in different size and colors for different contents, like tools or data, and put on a big sheet that represents a single interaction space. To represent how the using of the components would look like we group them in a meaningful way that reflects this. In addition, the used terminology should be based on the glossary that was a result of the domain modeling activities, and therefore will be consistent with the vocabulary used in the use case narrative.

After this session we will write a content list for each interaction space, which will contain the list of all tools in that interaction space. This list will make it simpler constructing requirements models.

The context navigation map illustrates the user’s navigation among different interaction contexts when acting a use case, and can show any problems that may be in how tasks are distributed over the interaction contexts. The map should reflect the complexity of the system in a user’s perspective, including how often and in what direction the user has to change attention between different contexts. When creating this map we frequently check the consistency with the other models we made for the role modeling and task modeling.

Something that really is worthwhile to do is to consider whether to have a larger number of small and simple interaction contexts, or to decrease the number of different contexts but then will have a growth in complexity. Since the operators in our system are rather experienced we would design the system with fewer but more complex interaction spaces to facilitate a faster usage.

At our client’s office the operator has several screens to pay attention to at some point in his or her work. To switch back and forth between these screens could reduce the efficiency of work and increase the risk or making errors, when the operator has to switch contexts. To decrease the load on the user’s short-term memory, we will synchronize interaction contexts when appropriate. This will give the users multiple views of the information, and at the same time, the decision about in which contexts he or she wants to interact with the system.

From this session, we will get a content model, documented from the big sheet/s supplemented with a content list, which will facilitate the work with constructing requirements models. We will also get a context navigation map that reflects the navigation through the system when enacting the use cases.

**Implementation Modeling**

Since it could be difficult to start programming directly from the interaction contexts identified in the previous step, we use implementation modeling to visualize the design of the implemented interface. We do this as simple sketches which we add comments to, and it will be performed by the same persons that made the essential model since they have best understood the user requirements.

The implementation model, a visual design that more closely resembles the final implemented interface, is a chance for us as designers to work out the finest details of the user interface design without having to cut code. The implementation model initiates, after we’ve sat down and discussed, what the interface should look like and what it should be able to do to be as usable as possible. We start this process by designing prototypes. There are different types of prototypes that we could use. A passive prototype just sits there and does nothing, like a drawing, a sketch on a piece of paper or like a wax figure at Madame Tassaud; it looks real but it doesn’t act like the real thing. It can also be a nonworking mock-up, like a cell phone-dummy. Passive prototypes alone are not enough to express the implementation model. An active prototype on the other hand is an implementation model that actually functions to some degree or another. These can take a variety of forms, such as software mock-ups, simulations and limited implementations. In our case, using a passive prototype is more appropriate than an active prototype since the systems is quite extensive.
**High-fidelity prototype** resembles the user interface it depicts, while **low-fidelity prototypes** only vaguely look like the real thing. **High fidelity prototyping seems like the adequate prototype for our design.** A **Horizontal prototype** is a thin slice off the top, meaning most or all of the user interface has little or no functionality, it only exhibits minimal activity, which for us is preferred ahead of **Vertical prototypes** that on the other hand is a narrow slice through the system, representing the more or less complete design for one segment of the user interface.

The first major question we must ask ourselves in order to create a usable system is: What is the user looking at most of the time when interacting with the system? This is known as the **Focal interaction.** What form will be used to realize the focal interaction contexts?

**One of the most important objectives according to us is to reduce context switching as much as possible.** Every action takes the user to a new dialogue box and, from there, to a sub-dialogue. This is not recommended. The user should therefore have a broad overview of all/most of the information without having to open a window that puts itself “over” the previous one. Another vital goal should be to reduce window management overhead/size. This management is all that time the user spends moving and resizing windows, dragging, tilting, minimizing etc. This takes up to about 10% of the whole time spent working. Our solution to this is not to banish resizable windows but to make more intelligent use of available features and mechanisms.

We do not only want to group together those components that are thought of together or are used together, but we also want to make tools and materials that are similar or related appear to be similar or related on the user interface. In general, displaying more data of a given type is better than displaying less. Users can usually find what they are looking for much faster by scanning a list or grid visually than by performing a search or by scrolling or paging through data. The human eye and brain form a remarkably efficient search engine. This is yet another reason why we would try to minimize the number of screens used in current system, without the compromise of making the user having to switch interaction context too often. An experienced user (especially) has the ability to find what he’s looking for within the interface even though there is a huge amount of information in front of him on a single screen.

**Usability Inspection**

In our project we will use Collaborative Usability Inspection method for assessment and identification of defects. The following quote explains the concept:

“**A Collaborative Usability Inspections** [Constantine, 1994a] is a systematic examination of a finished product, a design, or a prototype from the standpoint of its ultimate usability by intended end users. The review process is a team effort that includes software developers, end users, applications or domain experts, and usability specialists collaborating to perform a thorough and efficient inspection. This approach has features in common with both heuristic evaluation and pluralistic usability walk-throughs, and it borrows techniques from expert evaluations, but it was designed to be a faster and more efficient process for developers to learn.”

“A Collaborative Usability Inspections can be conducted at any point in the product development life cycle.”

**Inspection Roles**

The group carrying out Collaborative Usability Inspection (inspection team) consists of software developers, end users or end-user representatives, and usability specialists. The roles we will need to assign within our inspection team are: Lead Reviewer, Inspection Recorder, Continuity Reviewer, Developers, User and domain experts and Usability specialists. The lead reviewer
enforces the rules and agenda for the inspection and is also responsible for preparation to assure that the inspection can proceed efficiently and for follow-up activities to assure that appropriate action is taken on the results of the inspection. The inspection recorder is responsible for keeping a complete record of all defects identified, noting their location in the system, summarizing the problem, classifying it into useful functional categories, and assigning an estimate of the severity of the defect. Because the inspection recorder needs to understand the technical details of the discussion, he or she must be a professional developer. Also, prepared forms for recording defects can greatly simplify and speed up the process. It is the responsibility of continuity reviewer to watch for inconsistencies among different parts of the user interface and to monitor departures from the established standards, guidelines, or requirements. Developers are not permitted to explain, defend, excuse, or rationalize any aspect of their design or the decisions leading to it. For the best results, the inspection team should include at least one genuine end user. If no genuine end user is available, it is especially important that a domain expert participate in the inspections. The usability specialists functions as a source of expertise but not as the final arbiters of interface design decisions.

A possible inspection process that would be suited for our project would consist of:

1. **Preparation:** It is the responsibility of the Lead Reviewer to see that all advanced preparation is complete before convening the inspection. Participants are not expected to review materials in advance; it is better to approach from a fresh perspective.

2. **Interactive inspection:** The system is actually used (or usage is simulated) to carry out representative tasks. The inspection team walks through these scenarios to develop a feel for how the system would work in normal use. In enacting inspection scenarios, the user or users should take the lead in suggesting actions or in operating the system or prototype. Also, a collaborative inspection should not be allowed to become a user review with a lot of developers sitting around watching.

3. **Static inspection:** In the second pass, the static inspection, the inspection team “visits” or reviews all interaction contexts, independent of the order in which these might be encountered in ordinary use. It is not possible by enacting realistic usage scenarios to inspect all paths through a user interface. Static inspection should consider icons, buttons, labeling, layout, messages, controls, interactions, and so forth, in detail.

4. **Finalization and follow-up:** Follow-up activities include:
   1. Estimating the difficulty and cost of corrections for defects
   2. Sorting defects by severity, functional category, and estimated cost
   3. Reviewing user interface architecture for possible refinement or reorganization
   4. Grouping defects for assignment to particular personnel or scheduling for particular releases or versions
   5. Completing redesign of affected portions of the interface
   6. Implementing changes

**Testing**

There are numerous different approaches of testing. Two variants we will consider are: laboratory testing and field testing. Laboratory testing involves tests conducted fixed setting specifically configured for usability testing. There are certain areas such as Test room, Discussion room, Control room, Office and storage and Observation platform. Visual/audible controls are also available. Although, however comfortable and "home pleasant", the usability lab is still an artificial setting in which subjects will act differently than they do at work. Field testing takes the usability tests into the workplace. In our case we would use field testing to investigate how users use the current system seeing as there are so few users.
One of the most important things for us to remember while performing usability testing, wherever it may be conducted, is for us not to influence the subject. We must try not to help users with conscious or unconscious hints, such as glancing at the mouse or nodding.

We would encourage the users to “think out loud” during our usability test. The purpose of this is to gain access to what users are thinking about and what their intentions are as they try using the system. The drawback of this approach is that many users tend to “work” differently when asked to think out loud. An approach we could consider using to this problem is the deferred-reflection strategy. The recall of earlier thought while viewing videotapes was facilitated when a person’s own face was visible on the screen. So, the tester is recorded with one ore more cameras. Afterwards the interface designers look for signs of puzzlement, hesitation and so on. Selected segments are then shown to the tested person who then can answer questions like “what were you thinking when doing this and that?” Recall of the subjects “inner dialogue” is excellent using this technique.

The usability test should give us an idea of what we should implement into our interface. Using these steps could give us a good perspective of what we should get prepared for. In conclusion we say that the importance of testing is vast. The interface suffers from not being tested by the scrutiny of their users before handing it over.

**Innovative interfaces**

Here we will look at how practical and creative concerns can be combined to innovative user interfaces and how to create a usable interface.

![Figure 2. Creative expansion and critical constraints in the innovation process](image)

An innovation process works in an iterative manner with times of creativity and times of constrain, which include analysis. After a “creative session” (green lines) the developers thinking and reasoning is brought back down to earth (dotted lines) by considering practical limitations and testing the ideas against the realities of the real world. This alternation of creativity and constraints continue until they run out of time or are satisfied with the results. The innovation is finished off by refining and polishing the details (blue line).

Seeing as the user had 13 different screens with information to deal with, a lot of aspects are to be considered when designing the new interface. He or she shouldn’t have to focus on as many different screens; the chance of him missing vital information is huge. We should, in an iterative manner, create a new, better, more suited interface, preferably with as few screens as only one or two. By minimizing the number of the screens, we also minimize the risk of the user missing important information.
We must also discuss the importance of Active Overloading. We start with passive visual components and then load them with active behavior. The technique is referred to as overloading because a control ends up with more than one meaning and function. By doing this it makes controls more versatile (new behaviors) without taking up more screen real estate (in other words the buttons/controls have multiple functions without consuming unnecessary space on the interface). By adding more than one feature per button, we could probably create an interface that works on fewer screens but with the same amount of information than the recent 13 screens.

Ongoing processes during the whole project

Help System and Documentation
Good help for software can save a lot of time and effort and it requires special skills to write good documentation and help files. Therefore we will put a lot of effort in creating good help and helpful messages for the users. The help and documentation development will be concurrent activities in parallel with our software design and programming.

One example on how software design development can be used to create help and documentation in parallel is the way use cases can be used to organize help in a procedural way. If we have well designed use cases, they should reflect what the user experiences and therefore work well for organizing the help files as entries.

We should also have a continuous discussion about what kind of help we want to provide to the users because documentation and help serve different purposes. Documentation is not suited for help, it should record or document. Printed documentation has numerous advantages; it can be read everywhere, batteries never wear out, it doesn’t cover the screen.

We also have to discuss the many alternatives to display and design error messages, both with users and developers. The error messages have to be clear and informative and not irritate or accuse the user of doing things wrong. We will also try to use graphics to represent the exact information that the message is trying to say.

Because of the complex type of the work our client performs, where errors can be devastating and lead to loss of money, we want to create a system that helps to reduce and prevent errors, for example formatted fill-in-fields and flexible parsing of inputs can be used. The help systems should be properly field-tested along with early-release versions.

Fitting the Operational Context
To create truly usable software we have to have the operational context in mind all the time. Those aspects of the operational context that are most likely to affect user interface design decisions constitute the operational context model, or the operational model. Factors that can affect user interface and details design, which we should pay attention to throughout the process, are:

- Characteristics of users and the user roles
- Aspects of the physical work environment
- Features and limitations of operating equipment and interface devices
- General and specific operational risk factors

The operational model will serve as a repository for potentially significant information about the operational context. A structural role model, serves not only as an input for task modeling but also as a repository for holding operational context information associated with user roles. It can help to
prioritize our usability objectives and focus our resources on those aspects of user interface design that will have the greatest impact on producing efficient, effective, error-free use of the system.

**Concentric construction**

Essential use cases can serve many functions, for example when organizing the interface and the internal architecture, and also when organizing the implementation process itself. Using essential use cases makes it rather simple to stage the implementation process, starting with a central core of basic capability and expanding outward. Use cases become the suitable unit of product delivery because each use case represents one useful piece of work, one meaningful task to some users. This kind of staging can be used for planning successive releases or as insurance against project overrun or premature cutoff of resources. Concentric construction from core capability outward ensures that, whenever development is stopped, the last working version is likely to be usable.

Concentric construction helps us as developers to target different user populations and markets more successively. Use case analysis lets us avoid serious misallocation of features and resources.

**Architectural iteration**

Every developer’s worst-case scenario is to end up having an architectural collapse. At the outset of all software projects, the developers must make some basic assumptions and establish an overall framework within which to construct the system. In the beginning, the basic internal architecture may be sound and well suited to the problem at hand, but after a while, with each successive round of iterative refinement as the application and requirements grow, those early assumptions will be less well suited, if at all applicable. The farther away from the first basic assumptions you get, it will become more and more difficult to shoehorn in new features. Eventually, the software reaches a point of instability where almost any attempt to revise or even correct one part of the software brings the whole thing crashing down causing an architectural collapse. This is something that we could avoid by refining the architecture repeatedly along with code it is based upon. In architectural iteration, the basic software architecture is reviewed and revised by us on each round to ensure its continued viability.
We need a group of 6-8 people working on the project including developers, usability experts and project leaders. Although we need 34 weeks with 6-8 people, it can be accomplished in less time according to the parallel characteristic of these activities. Thus we will consider 22 weeks to perform the whole project.
**Analysis**

All and all we think that the authors have quite a good approach, but there are some things we would exclude and other aspects that we would add to make the approach more according to us.

According to the authors we should limit the number of iterations, otherwise we have to do these iterations forever as “users will always find something wrong or something more they want or something they do want”. We do not agree with the authors about this way of looking at users. When you are designing an interface, of course you sometimes must make the decision that you’re done with the selection of what solution you have, but that decision should rather be based on considering different solution suggestions and then have found the best one, even though it won’t be perfect.

Sometimes we get the feeling that the authors mainly use an iterative manner for information processing, structuring use case maps, content navigation maps etc. inside of the activities, like task modeling and implementation modeling. We would prefer the whole process to be more iterative instead of iterations mainly inside of each activity. Our understanding of the book is that once we have terminated an activity and created the artifacts belonging to it, we won’t return to that activity again.

We have excluded some of the testing methods that the authors either have created themselves or recommended. One of these is for example the Gedanken experiment, which we find way to simple to make use of for such a complex project that we are conducting. Another method that the authors themselves have created includes the metrics they present as equations. There are a number of equations the authors have come up with to calculate the metrics, with which the developer could presumably get information of the usability of a system. We considered this type of method quite difficult to use and inadaptable because of people’s objectivity and the way different people put values (numbers) on the experienced usability.

Another thing that made out eyebrows curl is what they had to say about the use of metaphors in interfaces:

“Command buttons that look like doorknobs or some other form of metaphor are likely to confuse and slow down the users.”

Metaphors are indeed usable according to other sources, so we don’t fully agree with what the authors say about this.

Finally, they are mostly focused on designing a system based on the task that is going to be performed and they do not consider the social aspect of workgroups and the way people usually work in reality.
Appendix A

Definitions
Usability definitions
There are five key elements in usage-centered design. They are as follows:

1. **Pragmatic design guidelines.**
   These guidelines help designers make reasonable decisions that lead to systems of high usability. The guidelines include both usability rules and design principles and contribute to the goals presented under "Usability". By following pragmatic design guidelines the designers of user interfaces can be pointed toward superior solutions, they get a broad overall view and know which paths to take and in which direction to go. In contrast, the design principles in usage-centered design provide narrower guidance on more specific issues in software usability.

There are certain rules that point out the general directions toward usability, in other words, the design principles gives the designer a map that shows route to take for designing highly usable interfaces. However, the rules should not be taken as a guarantee of good result, they are simply mere heuristics that point the way to better designs. The five rules are:

1. **Access**
   The system should be usable, without help or instruction, by a user who has knowledge and experience in the application domain, but no prior experience with the system. In other words, the interface should be accessible enough even for first-time users of the system.

2. **Efficacy**
   The system should not interfere with (or impede) efficient use by a skilled user who has substantial experience with the system.

3. **Progression**
   The system should facilitate continuous advancement in knowledge, skill, and facility and accommodate progressive change in usage as the user gains experience with the system. So instead of leaving users on their own to master advanced features, good designs help users become power users.

4. **Support**
   The system should support the real work that users are trying to accomplish by making it easier, simpler, faster, or more fun or by making new things possible. The Support Rule is the very heart of the usage-centered design. It is based on the notion that all software systems are tools, and good tools support work.

5. **Context**
   The system should be suited to the real conditions and actual environment of the operational context within which it will be deployed and used. It should suit the work task it has been given.

There are also **six principles** that are vital to user interface design. These principles are:

1. **Structure**
   Organize the user interface purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users, putting related things together
and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. Metaphors can be useful for structuring user interfaces and making them more accessible to users, especially first-time users. But, strained metaphors can make a user interface harder to understand, and simplistic simulations can make them harder to use.

2. **Simplicity**
Make simple, common tasks simple to do, communicating clearly and simply in the user’s own language and providing good shortcuts that are meaningfully related to longer procedures.
You cannot make everything simple. Common tasks and tasks that are simple from the user’s perspective should be made short and simple within the user interface.

3. **Visibility**
Keep all needed options and materials for a given task visible without distracting the user with extraneous or redundant information. The goal is to go beyond WYSIWIG (What you see is what you get) to WYSIWYN (What you see is what you need). WYSIWYN interfaces make visible those things that the user must have to complete the task at hand. Good designs do not overwhelm users with too many alternatives or confuse them with unneeded information.

4. **Feedback**
Keep users informed of actions or interpretations, changes of states or condition, and errors or expectations that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.

5. **Tolerance**
Make the system flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing while also preventing errors whenever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions reasonably; the system shouldn’t do anything stupid when confronted with unexpected input or actions.

6. **Reuse**
Reuse internal and external components and behaviors, maintaining consistency with purpose rather than merely arbitrary consistency, thus reducing the need for users to rethink and remember. Not only do inconsistent user interfaces make for less usable software, but they also require more programming.

2. **Model-driven design process;**
The authors argue "through models, developers can better understand usage and readily represent their understanding in ways that assist communication with the users and guide the work of programmers. // Abstract models encourage innovations; by leaving open more options they invite us to fill in the blanks in imaginative ways. // Essential models are more robust than concrete models because they are more likely to remain valid in the face of changing requirements and technology."

The presented models are:
- Role model - relationship between users and system
- Task model - structure of tasks that user will need to be accomplished
- Contend model - the tools and materials to be supplied by the users
To complete the specification of the user interface design (and to bridge the gap between the models above), two additional models are added:

- Operational model - the operational context in which the system is deployed and used
- Implementation model - the visual design of the user interface and description of its operation

3. Organized development activities
One works after some form of procedure or organized process, even though some developers prefer to work in a spontaneous manner. Usage-centered design is a streamlined, organized process that can be scaled to suit projects of varying size and scope. Its activities can be flexibly rearranged to suit various objectives or constraints, and many of the activities can be carried out concurrently for greater efficiency and faster delivery of software.

4. Iterative (incremental) improvement
It’s impossible to get a system perfect the first time, which is why an iterative approach is needed. Successive refinements are based on inspections and tests. Actual implementation can be completed in a series of iterations, starting with essential core of most needed features and facilities and expanding outward.

5. Quality measures
An innovative suit of software metrics, supported by usage-centered design process, can augment usability inspections, reviews and testing by giving early indications of the relative quality of designs.