

# Usability Engineering

Usability is not a single, one-dimensional property of a user interface.

Usability has multiple components and is traditionally associated with these five *attributes*:

- Learnability
- Efficiency
- Memorability
- Errors
- Satisfaction

**Learnability: The system should be easy to learn so that the user can rapidly start getting some work done with the system.**

Most fundamental usability attribute, since most systems need to be easy to learn, and since the first experience most people have with a system is that of learning to use it.

To measure the initial ease of learning one simply picks some users who have not used the system before and measures the time it takes them to reach a specific level of proficiency in using it.

The most common way to express the specified level of proficiency is simply to state that the users have to be able to complete a certain task successfully. One can specify a certain minimum time to consider them as having "learned" the system.

Because users have a tendency to jump right in and start using the system, one should not just measure how long it takes users to achieve complete mastery of a system but also how long it takes to achieve a sufficient level of proficiency to do useful work.

**Efficiency of Use: The system should be efficient to use, so that once the user has learned the system, a high level of productivity is possible.**

Expert user's steady-state level of performance at the time when the learning curve flattens out.

A typical way to measure efficiency of use is to decide on some definition of expertise, to get a representative sample of users with that expertise, and to measure the time it takes these users to perform some typical test tasks.

**Memorability: The system should be easy to remember, so that the casual user is able to return to the system after some period of not having used it, without having to learn everything all over again.**

Having an interface that is easy to remember is important for casual users and users who for some reason have temporarily stopped using the program.

Improvements in learnability often also make an interface easy to remember, but in principle, the usability of returning to a system is different from that of facing it for the first time.

One way to measure memorability is to perform a standard user test with casual users who have been away from the system for a specified amount of time, and measure the time they need to perform some typical test task.

It is also possible to conduct a memory test with users after they finish a test session with the system and ask them to explain the effect of various commands or to name the command (draw the icon) that does a certain thing.

**Few and Noncatastrophic Errors: The system should have a low error rate, so that users make few errors during the use of the system, and so that if they do make errors they can easily recover from them. Further, catastrophic errors must not occur.**

Users should make as few errors as possible when using a computer system. The system's error rate is measured by counting the number of errors made by users while performing some specified task.

Some errors are corrected immediately and have no other effect than to slow down the user. Such errors need not be counted separately, as their effect is included in the efficiency of use.

Other errors are more catastrophic in nature, either because they are not discovered by the user, leading to a faulty work product, or because they destroy the user's work, making them difficult to recover from.

**Subjective Satisfaction: The system should be pleasant to use, so that users are subjectively satisfied when using it; they like it.**

Can be specially important for systems in which their entertainment value is more important than the speed with which things get done.

Subjective satisfaction might be measured by simply asking the users for their subjective opinion.

Questionnaires should be subjected to pilot testing to make sure that the questions are interpreted properly by the users.  
study.

# HCI Principles

For computers to be widely accepted and used effectively they need to be well designed. This is not to say that all systems have to be designed to accommodate everyone, but that computers should be designed for the needs and capabilities of the people for whom they are intended.

Ultimately, users should not even have to think about the intricacies of how to use a computer.

Key principles to ensure good HCI (Donald Norman): **visibility and affordance**. Controls need to be visible, with good mapping with their effects, and their design should also suggest their functionality.

Primary design principles of direct manipulation interfaces (Norman): **affordances, constraints, mappings, and feedback**.

**Affordances: What sort of manipulation can be done to a particular object.**

Ex: Doors, afford opening, whereas a chair affords support.

Affordances play a large part in the design of objects but what is important is perceived affordance – what a person thinks can be done with the object.

For example, does the design of the door suggests that it should be pushed open or pulled?

Unfortunately, aesthetics sometimes conflict with good affordance and the appearance of the object takes precedence over its use.

**Constraints:** Whereas affordances suggest the scope of an object in terms of what it can do and how we can interact with it or move it, constraints limit the number of possibilities of what can be done with it.

There could be **physical, semantic, cultural, and logical** constraints.

Ex: vert/horiz scrollbar.

**Mappings:** How the spatial and conceptual relations between different parts of a system relate to their controls and their outcomes.

Mappings are good if they appear natural and intuitive to the users.

Bad mappings exist when the relations are inconsistent or incompatible – like pushing a car indicator upwards when wanting to turn left.

Ex: Stoves, electric seats.

**Feedback:** The sending back to the user information about what action has actually been done and what result has been accomplished.

# The Usability Engineering Lifecycle

UE is not a one-shot affair where the user interface is fixed up before release of the product.

UE is a set of activities that ideally take place throughout the lifecycle of the product, with significant activities happening at the early stages before the user interface has even been designed.

Usability cannot be seen in isolation from the broader corporate product development context where one-shot projects are fairly rare.

“Human factors involvement with a particular product may ultimately have its greatest impact on future product releases”

UE can still be successful even if it does not include every possible refinement at all of the stages.

The lifecycle model emphasizes that one should not rush straight into design.

The least expensive way for usability activities to influence a product is to do as much as possible before design is started, since it will then not be necessary to change the design to comply with the usability recommendations.

## Know the User

Two factors with the largest impact on usability:

- **Individual user characteristics**
- **Variability in tasks**

Users include installers, maintainers, system administrators, and other support staff in addition to the people who sit at the keyboard.

Even though “know the user” is the most basic of all usability guidelines, it is often difficult for developers to get **access** to users.

### Individual User Characteristics

- work experience, educational level, age, previous computer experience
- reading and language skills
- amount of time users will have available for learning
- work environment and social context

## Task Analysis

The users' overall goals should be studied as well as how they currently approach the task, what their information needs are, and how they deal with exceptional circumstances or emergencies.

The users' model of the task should also be identified, since it can be used as a source for metaphors.

Observe **effective** users and user **strategies** and “**workarounds**” as hints of what a new system could support.

Identify the **weaknesses** of the current situation: points where users fail to achieve goals, spend excessive time, or are made uncomfortable (these present opportunities for improvements).

A typical outcome of a task analysis is:

1. A list of all the things users want to accomplish with the system (**goals**).
2. All the information they will need to achieve them (**preconditions**).
3. The steps that need to be performed and the interdependencies between these steps
4. All the various outcomes and reports that need to be produced
5. The criteria used to determine the quality and acceptability of these results
6. The communication needs of the users as they exchange information with others while performing the task or preparing to do so.

Observation, interviews, concrete examples of their work . . .

Typical interviewer questions:

- Why . . . ?
- How . . . ?
- Why do you not . . . ?
- Do errors ever occur when . . . ?
- How do you discover and correct them ?

Users should also be asked to describe:

- Exceptions
- Remarkable instances of success and failures
- Problems
- What they liked best and least
- What changes they would like
- What ideas they have for improvements
- What currently annoys them

## **Functional Analysis**

A new computer system should not be designed simply to propagate sub-optimal ways if doing things.

One should not analyze just the way users currently do the task, but also the underlying functional reason for the task:

**What is it that really needs to be done, and what are merely surface procedures which can, and perhaps should, be changed.**

Examples:

- Face to face interaction in CSCW
- Paging in printed manuals

Of course, there is a limit to how drastically one can incorporate changes, to the functional analysis should be coordinated with a task analysis.

## The Evolution of the User

Users will not stay the same. Using the system changes the users, and as they change they will use the system in new ways.

It is impossible to forecast these changes completely as users will always discover new uses for compute systems after some period of use, but a flexible design will stand a better chance of supporting these new uses.

A typical change is that users become experts after some time and want interaction shortcuts (accelerators).

It is important not to design just for the way users will use the system in the first short period after its release.

## Competitive Analysis

Prototyping is an important part of the usability process, and existing, perhaps competing, products are often the best prototypes we can get of our own product.

An existing product is fully implemented and can be tested and analyzed very easily.

If several competing products are available, one can perform a **comparative analysis** of their differing approaches to the various user interface design issues.

Sometimes, competitive analysis will involve the study of non-computer interfaces.

Competitive analysis does not imply stealing other people's designs. One would hope to do better based on their strengths and weaknesses.

## Goal Setting

Usability comprises several components that can sometime conflict. Normally, not all usability aspects can be given equal weight in a give project, so you will have to make your **priorities** clear on the basis of your analysis of the users and their tasks.

Before starting the design of a new user interface, it is important to discuss the usability metrics of interest to the project and to specify the goals of the user interface in terms of measured usability.

For each usability attribute of interest, different levels of performance can be specified a part of the goal-setting process.

At least specify the **min** level acceptable. A more detailed specification can also include the planned level one is aiming for and the current level of performance.

Additionally, it can help to list the current value of the usability attribute as measured for existing or competing interfaces, and also the theoretically best possible value.

Usability goals are reasonably easy to set for new versions of existing systems or for systems that have a clearly defined competitor on the market.

The minimum acceptable usability would normally be equal to the current level, and the target usability could be derived as an improvement that was sufficiently large to induce users to change systems.

For completely new systems without any competition, usability goals are much harder to set.

## Parallel Design

Several designers work out preliminary designs.

The goal is to explore different design alternatives before one settles on a single approach that can then be developed in further detail and subjected to more detailed usability activities.

It is important to have the designers (teams) work independently, since the goal is to generate as much diversity as possible.

Usually, it is possible to generate new combined designs after having compared the set of initial designs, taking advantage of the best ideas from each design.

If several fundamentally different designs are available, it is preferable to pursue each of the main designs a little further in order to arrive at a small number of prototypes that can be subjected to usability evaluation before the final approach is chosen.

It is especially important to employ parallel design for **novel** systems where little guidance is available for what interface approaches work the best.

Parallel design is a very cheap way of exploring the design space, exactly because most of the ideas will not be need to be implemented, the way they might be if some of them were not tried until later as part of iterative design.

The main financial benefit is its parallel nature, which allows several design approaches to be explored at the same time, thus reducing the development schedule.

## Participatory Design

Even if we follow the advice from “**Know the User**” one still cannot answer all issues that come up during the design.

Instead of guessing, designers should have access to a pool of representative users after the start of the design phase.

Users often raise questions that the development team has not even dreamed of asking. Especially true with respect to potential mismatches between the users’ actual task and the developers’ model of the task.

Users should be involved in the design process through regular meetings between designers and users.

**Users are not designers . . .** However, they are very good at reacting to concrete designs they do not like or that will not work in practice.

Participatory design should not just consist of asking users what they want, since they often do not know what they want or what they need, or even what the possibilities are.

For larger development projects, thought should be given to periodically refreshing the pool of users who participate in the project since they risk becoming less representative of the average user . . .

. . . furthermore it is dangerous to rely too much on information from a small set of users that never changes

. . . on the other hand, there are trade-offs involved in changing user representatives.

## **Coordinating the Total Interface**

Consistency is one of the most important usability characteristics. It should apply across the different media which forms the total user interface (application screen, documentation, help systems, tutorials, training classes).

Consistency is not just measured at a single point in time but should also apply over successive releases of a product and across entire product families.

To achieve consistency of the total interface it is necessary to have some centralized authority for each development project to coordinate the various aspects of the interface.

Interface standards, Shared culture, Existing products, Development tools and libraries help achieve consistency.

## **Guidelines and Heuristic Evaluation**

Guidelines list well-known principles of user interface design which should be followed in the development project.

Different types of guidelines:

- General: provide feedback
- Category-specific: (GUIs) ensure the main objects of interest are visible on the screen and that their most important attributes are shown.
- Product specific: (graphical file syst) have each file and subdirectory represented by an icon and use different icon shapes to represent different classes of objects.

The difference between standards and guidelines is that a standard specifies how the interface should appear to the user, and a set of guidelines provides advice about the usability characteristics of the interface.

Hopefully a given standard will follow most of the traditional usability guidelines so that the interfaces based on it will also be as usable as possible.

Example:

**Guideline:** users should always be able to back out from any undesirable state.

**Standard:** an *undo* command should always be available and it should be shown as an icon at the top right of the screen.

**Standard:** return to the previous system state whenever the user hits the *escape* key.

## Prototyping

Early usability evaluation can be based on prototypes of the final systems that can be developed much faster and cheaper, and which can thus be changed many times until a better understanding of the user interface design has been achieved.

The entire idea of prototyping is to save on the time and cost to develop something that can be tested with real users.

**Vertical prototypes:** reduce the number of features. A narrow system that does include in depth functionality, but only for a few selected features.

It can only test a limited part of the full system, but it will be tested in depth under realistic circumstances with real user tasks.

**Horizontal prototypes:** reduce the level of functionality. A surface layer that includes the entire user interface to a full-featured system but with no underlying functionality.

A simulation of the interface where no real work can be performed. Makes it possible to test the entire UI. They can often be implemented fast and they can be used to assess how well the entire interface “hangs together” and feels as a whole.

In addition to reducing the proportion of the system that is implemented, prototypes can be produced faster by:

- Placing less emphasis on the efficiency of the implementation.
- Accepting less reliable or poorer quality code.
- Using simplified algorithms that cannot handle all the special cases.
- Using a human expert operating behind the scenes to take over certain computer operations that would be too difficult to program. *Wizard of Oz* technique.
- Using a different computer system than the eventual target platform.
- Using low-fidelity media.
- Using fake data and other content.
- Using paper mock-ups instead of a running computer system.
- Relying on a completely imaginary prototype where the experimenter describes a possible interface to the user orally, posing a series of “what if (the interface did this of that ...)” .

Prototypes may sometimes be used for a special form of participatory design called **interactive prototyping**, where the prototype is developed and modified on the fly as a tester comments on its weak spots.

Paper mock-ups are often used for interactive prototyping sessions which allow the users to modify the designs (PICTIVE).

A prototype is a form of design specification, and the final implementation of a UI is often performed with the prototype as a major way of communicating the design to the developers.

One needs to be aware that not every aspect of the prototype should be replicated in the final system, and the designers should inform developers about which aspects of the prototype are intentional and which are arbitrary.

## Scenarios

The ultimate minimalist prototype. They describe a single interaction session without any flexibility for the user. A scenario is an encapsulated description of

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

Scenarios have two main uses:

- During the design of a UI as a way of expressing and understanding the way users eventually will interact with the future system.
- During early evaluation of a UI design to get user feedback without the expense of constructing a running prototype.

Example: A scenario for the use of an ATM:

1. The user approaches the ATM and inserts the card. No matter the side, the machine reads it correctly.
2. The ATM asks the user to enter the PIN, and the user does it using the keypad.
3. The ATM presents a menu of four options, “W-\$100”, “W-Other amounts”, “Deposit”, and “Other transactions”.
4. The user presses the button for “Withdraw \$100”, and the ATM pays out that amount, deducting it from the user’s account. If the user has more than one account, it is deducted from the one with the highest balance.
5. The ATM returns the card.

The scenario raises some questions for the design of the user interface to the ATM

## **Interface Evaluation**

In general, scenario descriptions are good tools in the early stages of design because they can be generated and edited before the UI has been fully designed.

Scenarios describing possible uses of envisioned future systems are also helpful for early participatory design exercises, since users will find it easier to relate to the task-oriented nature of the scenarios than to the abstract, and often function-oriented nature of systems specifications.

Scenarios can also be used for user testing if they are developed with slightly more detail than a pure narrative.

Mock-up drawings and video tapes of users interacting with a simulated system are tools used to build more elaborated scenarios.

### **Meta-Methods**

To ensure the successful application of the UE methods, it is important to supplement each method with the following meta-methods (methods that apply to methods):

- Write down an explicit plan for what to do when using the method.
- Subject this plan to an independent review by a person who can critique it from a fresh perspective.
- Perform a pilot activity. Then revise your plan to fix the difficulties that invariably will be found during the pilot.

## **Prioritizing Usability Activities**

It is not always possible to perform all the recommended usability activities in any give project.

J. Nielsen's approach to budget or time constraints stress:

- Visit to user site
- Prototyping through scenarios
- Simplified thinking aloud
- Heuristic evaluation

According to a survey conducted by the Nielsen of 13 UE specialists on the usability impact of several methods (1-no impact to 5-absolutely essential):

- Task analysis of the user's current task (4.7)
- Iterative design (4.7)
- Empirical tests with real users (4.5)
- Participatory design (4.4)
- Visit to customer location before start of design (4.3)
- Field study to find out how system is actually used after installation (4.3)

## Be Prepared

In emergency cases (help with projects gone astray or especially pressed for time), better results may be expected if the usability specialists have prepared for such eventualities in advance.

The following precautions can be taken during any less hectic periods that may be available between urgent projects:

- Get a good UI prototyping tool and acquire proficiency in using it.
- Learn appropriate techniques for usability inspection and heuristic evaluation, and get familiar with the relevant interface standards and guidelines.
- Build up an understanding of the types of users, tasks, applications, and computer platforms that are typical for your organization.
- Set up procedures that will allow you to recruit test users easily when they are needed.
- Find and train a usability champion in each project group that does not have its own full-time usability specialist.
- Read more usability books and articles and attend conferences.