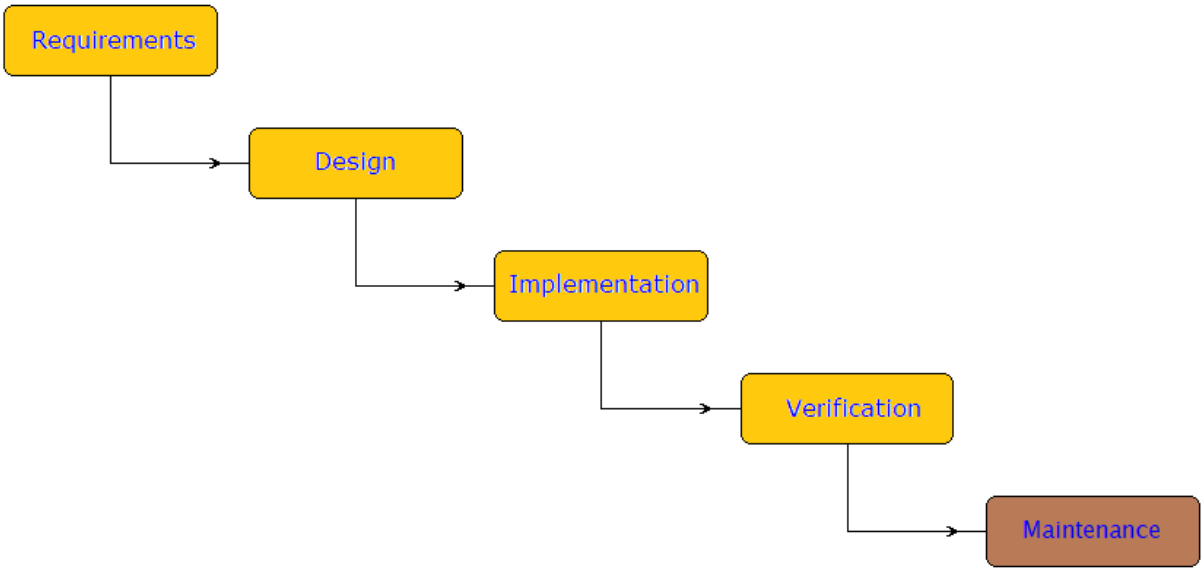
The Usability Design Process

Most software design models are based on the *waterfall model*.

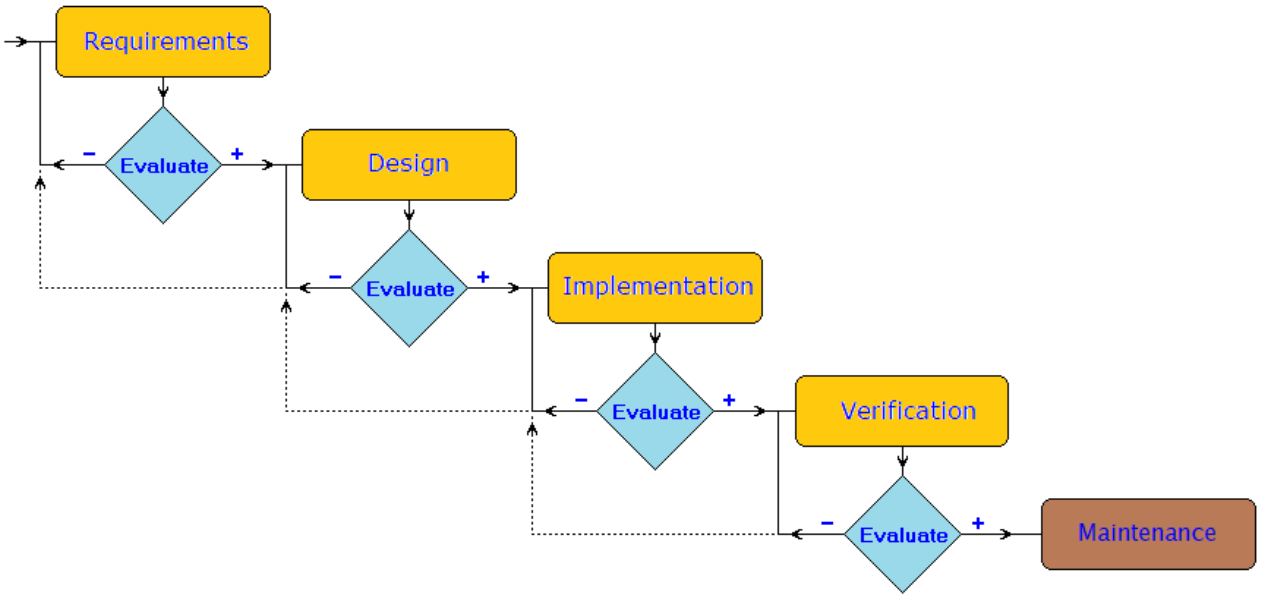


The basic waterfall approach is not very flexible.

It can be made more flexible by adding an evaluation phase at each stage.

Done properly, this delivers good results.

However, adding evaluation at every stage makes this is an expensive and time-consuming approach.



Note that:

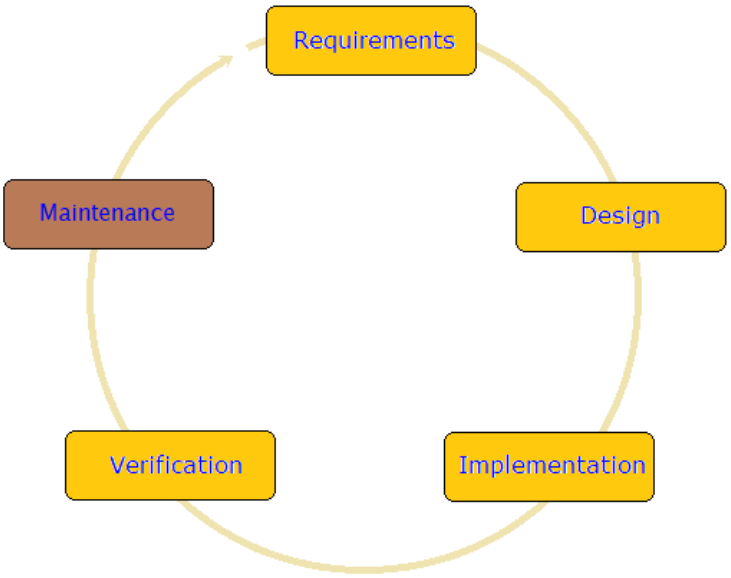
- the results are evaluated at each stage
- there is provision to return from any stage in the cycle to any earlier stage if problems are discovered.

As design and development processes became more user-focussed, speed and flexibility became more important than precision and predictability.

Many companies abandoned waterfall approaches in favour of *agile* development.

Agile development focusses on rapid iteration.

Products are developed and re-developed in 2-4 week cycles known as *sprints*.

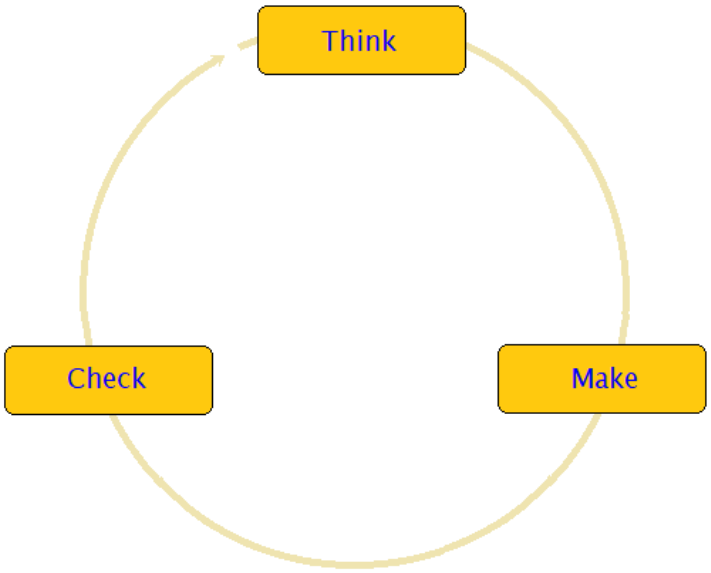


However, this fast turnaround leaves little time for user-testing, etc.

Companies adopting agile approaches often neglected usability.

This led many companies to develop *Lean UX* (Lean User Experience) approaches.

Like agile development, Lean UX is based on an iterative cycle, but with just three steps: *Think*, *Make* and *Check*.



Lean UX addressed some of the issues with Agile development, but often led to waste and repetition of work.

Most modern approaches are *Dual Track*:

- The *Discovery Track* involves identifying users, defining requirements, building prototypes, etc.
    - This typically runs on a long cycle (four weeks or more)
- The *Delivery Track* involves building deliverable products.
    - This typically runs on a short cycle (around two weeks)

Most of these approaches share several key stages:

- Requirements gathering
    - find out who the users are, what they want to do with the system, etc.
- Develop a conceptual design
- Implement the design
    - taking into account, cognitive and perceptual factors, guidelines, etc.
- Evaluate and revise as necessary
    - using heuristics and/or metrics, cognitive modelling, controlled studies, etc.

Approaches which focus on UX/Usability are often collectively termed *user-centred design*.

No one method guarantees good results.

Therefore, it is useful to think in terms of having a 'toolkit' of approaches: different combinations of approaches can be used as appropriate for particular projects.

Requirements Gathering

The first stage in the design of an interface is to identify the requirements.

This involves consideration of a number of questions, such as:

1. What area of expertise will the application be developed for?
2. Who are the users?
3. What do the users want to do with the application?
4. Where and how will the application be used?

1. What is the area of expertise?

In order to use most applications it is necessary to have a certain level of knowledge about the relevant *domain*.

For example, someone using a graphics application may need to know about:

- layers and transparency
- vector and bitmap formats
- colour spaces
- image file formats and compression, etc.

The task of identifying domain knowledge is known as *domain analysis*.

Most software applications involve some element of *domain knowledge*.

However, it is often so ingrained that it is not recognised as domain knowledge.

- experts may be so familiar with their field that they regard some domain knowledge as *general knowledge*
- if they cannot distinguish between general knowledge and domain knowledge, we cannot rely on them to identify domain knowledge.

Therefore, domain analysis should involve talking to both:

- experts in the relevant field(s)
- end-users (or potential end-users).

One useful approach is to conduct a pen-and-paper 'walk-through' of a transaction or operation with representatives of both groups.

This can identify many issues concerning domain knowledge.

2. Who are the users?

It is imperative to know who the system is being designed for.

Therefore the designer should start by identifying the target users.

One approach is to draw-up a *profile* which includes factors such as:

- Educational background
- Computing skills/experience
- Culture
- Attitude
- Age
- Physical abilities and disabilities

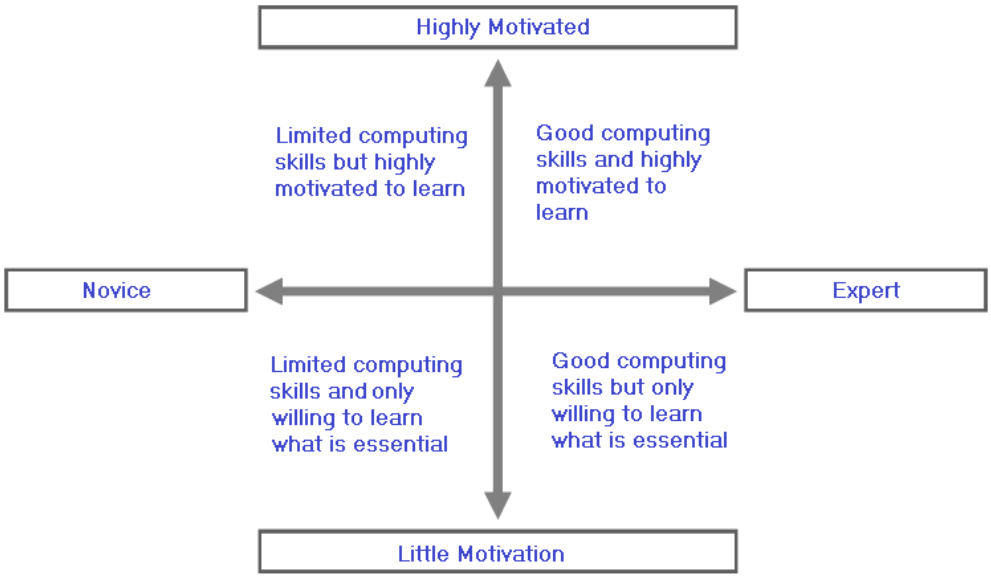
However, it's very difficult to design for a large, loosely-defined group.

Studies suggest such that applications designed in this way often fail to meet the needs of *any* of their users.

A better approach is to *segment* potential users into a number of smaller groups.

The users might be segmented according to two criteria, e.g.:

- Computing experience (novice/expert)
- Motivation (unmotivated / highly motivated)



This allows the user population to be divided into four groups. The needs of each group can then be separately assessed and addressed.

If desired, each group can be represented by a profile of an imaginary user. These profiles are called *personas*.

A persona:

- should cover all the factors listed above, but should also include other details, such as likes and dislikes, habits, etc..
- can be a composite, combining characteristics from a number of real people, but should be consistent and realistic.
- should read as the description of a real person.

Personas might be developed for users such as:

- A young person who uses IT regularly for entertainment and education purposes.
- A middle-aged person who uses IT regularly at work.
- A middle-aged person who does not use computers at work but has a home-computer and uses it for email, web-surfing and in connection with their hobby.
- A retired person who has little knowledge of computers, and who uses them only when absolutely necessary.

In many cases, personas are developed to reflect each of the groups identified through segmentation.

In other cases, personas can be developed without the need for segmentation.

The use of personas is common practice in many industries and has proved very successful.

It has been found that a product developed to meet the specific needs of a particular group often meets the needs of a much wider group.

When dividing the users into groups, it may also be necessary to distinguish between:

- *primary users*
    - People who use the system/interface directly
- *secondary users*.
    - People who specify a task that is performed by a primary user.

For example, in the case of an archive:

- the *primary users* might be archivists, who perform searches of archival databases.
- the *secondary users* might be researchers, who enlist the help of archivists to find historical records, etc.

The needs of the two groups may be different.

Both sets of needs should be identified and catered for.

Another example is a mobile app to help children with Cystic Fibrosis manage their condition.

- Clinicians access and manage the recorded data using a web interface tailored to their needs.
- The children and their parents use the mobile app.

The interaction needs of both the children and their parents had to be taken into account when designing the mobile app.

3. What do the users want to do?

In identifying needs we must distinguish between:

- Needs identified by professional designers/developers.
    - These are often referred to as *normative needs*
- The needs of the end-user. These can be difficult to determine. It often helps to think in terms of:
    - *Expressed needs* - what end-users SAY they want.
    - *Felt needs* - what end-users ACTUALLY want (or would like) from the system.

There may be considerable discrepancies between normative and end-user needs, and possibly conflicts.

In many cases, users lack the technical vocabulary or understanding to express felt needs, but feel them nonetheless.

Unmet felt needs are often expressed as a general dissatisfaction with a system.

Some end-users needs may be excessive and impractical.

However, this does not mean they can be ignored.

Such needs often reflect a deeper dissatisfaction with the system which cannot be overcome but can be ameliorated through reasonable compromise.

The traditional methods for determining user needs are:

- direct observation (where possible)
- questionnaires
- interviews.

Direct Observation

One way to identify the requirements of likely users is to study existing users of similar systems.

Ideally, you should observe people who are using the system for their own ends, unprompted by you.

For example, if the task is to develop a better ATM interface, you could (with the bank's permission) use video to monitor people using existing ATMs.

You could then note any problems they encounter.

It's much more difficult to monitor users of applications that are carried-out mainly on personal or mobile devices, perhaps in the privacy of the home.

An alternative is to ask a group of people to carry out certain relevant operations in the laboratory, and then observe them.

However, this gives no guarantee that they are carrying out the same operations, in the same way, as they would if un-prompted.

Where an application is based on - or is similar to - existing systems, it may be possible to use *artefacts* to identify non-normative needs.

An artefact is an object or aid used in the performance of a task. Examples include:

- Notes stuck to the computer detailing (e.g.) keyboard short-cuts
- Reference manuals kept close at hand (perhaps with some pages well-thumbed), or photocopied sheets from reference manuals pinned in a prominent position.
- Manuals created by the users themselves.

Artefacts such as these can provide clues as to how users work with a system and what problems they encounter.

Questionnaires

Where direct observation is not possible, a useful approach is to ask people:

- if they have used a similar system or carried out the relevant operation(s) before
- if so, what their experiences were.

The questions might cover:

- How much experience they have with relevant systems?
- What kinds of tasks, queries, etc., they have carried out using this type of system?
- Did they encounter particular problems?
- If they have tried several competing systems, did they find one easier to use than another, and if so, in what way?

The questioning can be carried out through questionnaires or interviews (discussed below).

Considerable thought must be given to the design of questionnaires.

- The designer/developer inevitably starts from his/her own perspective, and thus is most likely to identify normative rather than end-user needs.
- Being aware of the problem helps, but isn't a complete solution.
- Checking a questionnaire for evidence of a normative bias is a good start, but is unlikely to identify topics that have been excluded altogether through the operation of this bias.
- Having the questionnaire reviewed by others may help.

Interviews

Where direct observation and questionnaires are either unsuitable or inadequate, an alternative is to conduct *walk-throughs*.

- Conducted during face-to-face interviews
- Potential users are asked to describe how they would carry out the task step-by-step.
- Pen and paper only, no computer system or other hardware.
- This allows the subject to describe the process as he/she sees it rather than as it is usually carried-out.

The exercise can be useful in illustrating how the end-user views the task or process, and hence in identifying the appropriate ordering of stages, etc..

It may also identify steps that are likely to be forgotten by end-users or incompletely understood.

Walk-throughs can be conducted either in their own right or as a preliminary step to designing a questionnaire or preparing a list of questions for a more formal interview.

Interviews versus questionnaires:

- Interviews are usually less structured than questionnaires.
    - This can be overcome to some extent by having the interviewer use a fixed list of questions.
    - However, this probably removes the main advantage of an interview, which is the opportunity to ask questions spontaneously as they arise.
- Questionnaires provide a more formal, structured setting than interviews, ensuring consistency between respondents.
    - Allowing respondents to fill-in questionnaires in their own time removes the risk of bias from an interviewer, ensuring that respondents are free to provide their own perspective.

4. How will the application be used?

The environment in which an application will be used may have an impact on its usability.

For example, consider an ATM:

- It will be used out-of-doors, so it must be usable in all weathers.
- The screen must be viewable under all lighting conditions.
- Users might be wearing outdoor clothing that impedes movement and makes them 'clumsier' than they would otherwise be.
- Users may only be able to devote part of their attention to the task because:
    - they are surrounded by other people and feel pressured or concerned about their privacy.
    - They are simultaneously trying to control small children.
    - etc.

Applications that run on PCs or mobile devices generally pose fewer problems since the user typically has more control over where and how the device is used.

However, there may still be issues with distraction, sharing of information (if more than one person needs to view the task data to discuss it), etc..

Issues to be considered include:

- Physical Aspects
    - e.g., lighting, noise levels, space/layout, etc.
- Safety Aspects
    - e.g., distractions (for safety-critical systems)
- Social Aspects
    - e.g., individual/group working, task-sharing/dependency, etc.
- Organisational Aspects
    - e.g., IT Policy, user support, etc.

Requirements Gathering - Summary

What is the domain?	Use *domain analysis* to identify *domain knowledge*
Who are the users?	Age, education, IT knowledge, etc. Difficult to design for a broad range of users, so *segment* user-population into a few groups represented by *personas*. May need to cater for both *primary* and *secondary* users
What do the users want to do?	Needs: *normative* & *end-user* (*felt*/*expressed*). Use *direct observation*, *questionnaires*, *interviews*, and *artefacts* as appropriate.
Where/how will it be used?	*Physical*, *safety*, *social* and *organisational* issues may affect usage and hence design.

At the end of the requirements-gathering process we should have:

- Considered the domain, and the associated domain knowledge.
- Identified users who might wish to use such a system, i.e.:
    - segmented target-users according to appropriate criteria (computing-experience, motivation, etc.), and /or...
    - drawn up a representative set of personas
- Identified the tasks our users wish to carry out
- Considered the conditions/environment in which the interface (and system) will be used.

Using this information we can draw-up a *Requirements Document*.