Usability Heuristics

1. Simple and Natural Dialogue

- User Interfaces should be simplified as much as possible

- Every additional feature or item of info is:
  - One more thing to learn
  - One more thing to possibly misunderstand
  - One more thing to search through

- Interfaces should mask user's task

- Present exactly the info the user needs – *and no more* - at exactly the time and place where it is needed

- Information that will be used together should be displayed close together

Graphic Design

- Things are seen as belonging together if:
  - they are close together
  - are enclosed by lines or boxes
  - move or change together
  - look alike with respect to shape, color, size or typography

- Menus can use dividing line or color coding to split options into related groups

- Dialog boxes can group related features and enclose them in boxes or separate them by lines or white space

- Graphic design can also help users prioritize their attention by making the most important dialogue elements stand out

Use of Color

- Don't over-do it
- Color-coding should be limited to no more than 5 to 7 different colors
- Light grays and muted pastel colors are often better as background colors than screaming colors are
- Make sure the UI can be used without colors
- Try to use colors only to categorize, differentiate, and highlight, not to give info, especially quantitative info.

Less Is More

- Based on a proper task analysis, it is often possible to identify info that is truly important to users and which will enable them to perform almost all of their tasks.
- Extraneous info not only risks confusing the novice user, but also slows down the expert user.
- The "*less is more*" rule does not only apply to the info contents but also to the choice of features and interaction mechanisms for a program.
- Alternatives can be provided if users can easily recognize the conditions under which each interaction technique is optimal so that they can consistently choose them without additional planning.

2. Speak the Users' Language

- The terminology should be based on the users' language and not on system oriented terms.
- As far as possible, dialogues should be in the users' native language.
- One should take care not to use words in non-standard meanings.
- When the user population has its own specialized terminology for its domain, the interface had better use those specialized terms rather than more commonly used, but less precise, everyday language.
- View the interaction from the users perspective.
- The system should not force naming conventions or restrictions on objects named by the user.
- Let users vote on the names, based on a shortlist of possibilities.
- It should be possible for the users to define aliases.

Mappings and Metaphors

- Aim at good mapping between the computer display of info and the users' conceptual model of the info.
- To discover such mappings you can use techniques such as task analysis, order recall, card sorting, etc.
- Metaphors are a possible way to achieve a mapping between the computer system and some reference system known to the users in the real world.
- Metaphors may mislead users, or may limit the understanding to those aspects that can be inferred from the metaphor.
- One should take care not to inadvertently imply more than one intended.
- Metaphors also present potential problems with respect to internationalization.

3. Minimize User Memory Load

- Computers remember things very precisely, they should take over the burden of memory from the user as much as possible.
- The computer should display dialogue elements to the user and allow them to choose or edit them.
- When users are asked to provide input, the system should describe the required format or provide an example or default value.
- Provide users info of the range of legal inputs and the unit of measurement.
- The system should be based on a small number of pervasive rules that apply throughout the UI.
- The use of generic commands is one way to let a few rules govern a complex system.

4. Consistency

- If users know that the same command or action will always have the same effect, they will feel more confident in using the system, and they will be encouraged to try out exploratory learning strategies.
- The same info should be presented in the same location on all screens and dialog boxes and it should be formatted the same way to facilitate recognition.
- Consistency is not just a question of screen design, but includes considerations of the task and functionality structure of the system.

5. Feedback

- The system should continuously inform the user about what it is doing and how it is interpreting the user's input.

- Feedback should not wait until an error situation has occurred: the system should also provide *positive* feedback, and should provide *partial* feedback as info becomes available.

- Feedback should not be expressed in abstract and general terms but should restate and rephrase the user's input to indicate what is being done with it.

- Different types of feedback may need different degrees of persistence in the interface.

Response Time

- Feedback becomes especially important in case the system has long response times for certain operations.

- Normally, response time should be as fast as possible, but it is also possible for the computer to react so fast that the user cannot keep up with the feedback.

- In cases where the computer cannot provide fairly immediate response, continuos feedback should be provided to the user in the form of a percent-done indicator or running progress feedback in terms of the absolute amount of work done.

System Failure: Informative feedback should also be given in such cases. No feedback is almost the worst possible feedback since it leaves users to guess what is wrong.

6. Clearly Marked Exits

- Users do not like to feel trapped by the computer. To increase their feeling of being in control of the dialogue, the system should offer the user an easy way out of as many situations as possible.

- In many cases, exits can be provided in the form of an ***undo*** facility that reverts to the previous system state.

- A basic principle of UI design should be that users will make errors no matter what else is done to improve the interface, and one should therefore make it as easy as possible to recover from these errors.

- The exit and undo mechanisms should be made visible and should not depend on the user's ability to remember some special code or obscure combination of keys.

7. Shortcuts

Even though it should be possible to operate a UI knowing just a few general rules, it should also be possible for the experienced user to perform frequently used operations faster.

Typical ***accelerators*** include:

- abbreviations

- having function keys or command keys that package an entire command in a single keypress

- double-clicking on an object to perform the most common operation on it

- having buttons available to access important functions directly from within those parts of the dialogue where they may be most frequently needed.

- Users should be allowed to jump directly to the desired location in large information spaces.

- Popular locations may be given easy to remember names that have to be typed by the user.

- Users may be allowed to give their own names to those locations they find especially important (build a list of bookmarks). Based on heuristic 3, the user should have easy access to the list of bookmarks.

- Users should be able to reuse their interaction history.

- System provided default values are faster to recognize and accept.

8. Good Error Messages

Basic rules for error messages:

a) They should be phrased in clear language and avoid obscure codes.

b) They should be precise rather than vague or general.

c) They should constructively help the user solve the problem

d) They should be polite and should not intimidate the user or put the blame explicitly on the user.

In addition to good error messages, systems should also provide good error recovery.

Instead of putting all useful info in all messages, it is possible to use shorter messages (faster to read) as long as the user is given easy access to a more elaborate message.

9. Prevent Errors

Even better than having good error messages is to avoid the error situations in the first place.

There are many situations that are known to be error-prone and systems can often be designed to avoid putting the user in such situations.

> **Ex**: every time users have to spell something, there is a risk of spelling errors. So selecting rather than typing is an easy way to redesign a system to eliminate this category of errors.

User errors can be identified as candidates for elimination through redesign either because of their ***frequency*** or their serious ***consequences***.

Serious consequence errors can also be reduced in frequency by asking users to ***reconfirm*** that they "really mean this" before going ahead with the dangerous actions.

One should not use confirmation dialogues so often, though, that the user's answer becomes automatic.

Avoid Modes

Modes are a frequent source of user error and frustration and should be avoided if possible.

However, they are almost impossible to avoid totally in complex interfaces. Thus, one can at least prevent many mode errors by explicitly recognizing the modes in the interface design.

By showing states clearly and distinctly, a designer follows the principle of providing feedback, and thus make it less likely that the user will mistake the current mode.

In addition to having clear status indicators showing the current mode, the interface should also exhibit clear differences between user actions in different modes to minimize the risk of confusing individual interface elements.

Similarly, system feedback should be sufficiently varied to provide additional differentiation between modes.

Mode confusion can also be prevented by the use of so called ***spring-loaded modes*** where the users are only in the mode as long as they actively hold down a button or perform some other action that will automatically take tem out of the mode as soon as they let go.

Even without the added usability problem of modes, one should in general avoid having too similar commands. Ex: case sensitive.

10. Help and Documentation

Except for ***walk-up-and-use*** systems, most user interfaces warrant a manual and possibly a help system. Also, regular users of a system may want documentation to enable them to acquire higher levels of expertise.

"It is all explained in the manual" should never be the systems designer's excuse when users complain that an interface is too difficult.

The fundamental truth about documentation is that most users ***do not*** read manuals. A corollary to this is that if users ***do*** want to read the manual, then they are probably in some kind of panic and will need immediate help. This observation indicates the need for good, task-oriented search and lookup tools for manuals and online documentation.

The main principle to remember about online help and documentation is that these facilities add an extra set of features to a system, thus complication the interface just by virtues of existing.

Online help has the advantage over documentation that it can be context-sensitive.

One important aspect of either is the quality of the writing, especially including the structuring of the info and the readability of the text.

A mayor in-depth study of online help concluded that “the quality of help texts is far more important than the mechanisms by which they are accessed”.

A second corollary to the finding is that when users do want to read the manual, they often will not be able to find it. This can be overcome by using online documentation.

Heuristic Evaluation

Heuristic evaluation is done by looking at an interface and trying to come up with an opinion about what is good and bad about the interface.

HE involves having a small set of evaluators examine the interface and judge its compliance with recognized usability principles (heuristics).

A single evaluator will miss many usability problems. Thus, it is recommended the use of about five, and certainly at least three.

The HE Process

HE is performed by having each individual evaluator inspect the interface alone.

Only after all evaluations have been completed, are the evaluators allowed to communicate and have their findings aggregated. This is important in order to ensure independent and unbiased evaluation from each evaluator.
During the evaluation, the evaluator goes through the interface several times and inspects the various dialogue elements and compares them with the heuristics. Of course, the evaluator is also allowed to consider any additional usability principle or results that come to mind that may be relevant.

In principle, each evaluator decides how to proceed. However, its recommended to go through the interface at least twice:

- ***First pass***: intended to get a feel for the flow of the interaction and the general scope of the system.
- ***Second pass***: allows to focus on specific interface elements while knowing how they fit the larger whole.

It is possible to perform HE of user interfaces that exist on paper only. This makes the method suitable for use early in the usability lifecycle.

Based on the type of application, the evaluators will be able to used the system, or require assistance to use the interface. Typical ***usage scenarios*** are a common approach for those cases.

The output from using the HE method is a list of usability problems, annotated with references to those principles that were violated by the design.

HE does not provide a systematic way to generate fixes to the usability problems or a way to asses the probable quality of any redesigns.

However, since it aims at explaining each observed usability problem with reference to established principles, it will often be fairly easy to generate a revised design according to the guidelines provided by the principle that was violated.

One possibility for extending the HE method to provide some advice is to conduct a debriefing session after the last evaluation session.

HE is explicitly intended as a ***"discount usability engineering"*** method.

Although HE can be performed by people with little or no usability expertise, it is preferable to use quality specialists as the evaluators, and optimal performance requires double specialists.

Usability Heuristics

- ***Visibility of system status:*** the system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

- ***Match between system and the real world:*** the system should speak the users' language, with words, phrases, and concepts familiar to the user, rather than system oriented terms. Follow real-world conventions, making info appear in a natural and logical order.

- ***User control and freedom:*** users often choose system functions by mistake and will need a clearly marked "exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

- ***Consistency and standards:*** users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

- ***Error prevention:*** even better than good error messages is a careful design which prevents a problem from occurring in the first place.

- ***Recognition rather than recall:*** make objects, actions, and options visible. The user should not have to remember info from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

- ***Flexibility and efficiency of use:*** accelerators –unseen by the novice user– may often speed up the interaction for the expert user to such an extent that the system can cater both inexperienced and experienced user. Allow users to tailor frequent actions.

- ***Aesthetics and minimalist design:*** dialogues should not contain info which is irrelevant or rarely needed. Every extra unit of info in a dialogue competes with the relevant units of info and diminishes their relative visibility.

- ***Help users recognize, diagnose, and recover from errors:*** error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

- ***Help and documentation:*** even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such info should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.