Usability

Paul Jackson

School of Informatics
University of Edinburgh

Interaction design

- Primarily slides here are about user interface design
- Much also applies to interface design in general

What's the problem?

Much software is found to be hard to use by the intended end users.

Many problems are described as "user errors" (or even "luser errors").

Most user errors are actually interface design failures.

Why does this happen?

Human limitations

Humans have *very* limited short-term memory: 5–7 items.

Humans make mistakes, especially under stress.

Humans have widely varying capabilities, both physical and mental.

Humans have widely varying personal preferences.

Humans brains organise their perceived world differently.

Principles of UI design

There are some fairly basic principles which, when followed, will reduce the chances of really bad design.

- User familiarity
- Consistency
- Minimal surprise
- Recoverability
- User guidance
- User diversity

We'll consider each of these in a little more detail.

For more detail see Sommerville online chapter on *Interaction Design*.

User familiarity

The interface should ‘look familiar’ to the users – it should use concepts and entities from the existing experience of the users. For example:

1. An air traffic control system should present users with (representations of) aircraft, height, speed etc.; not anything to do with the implementation.
2. Familiarity is the guiding philosophy behind the desktop metaphor on PCs. This application also shows the dangers: a file on a PC is not a file in a filing cabinet, it’s just conceptually similar.

Consistency

Similar operations should be represented in similar ways (e.g., Cancel, drop down icon).

Across the application? Across all applications?

Windows and Macintosh strive for great consistency - see iOS. Traditional Unix doesn't (but some distros more than others do e.g., Ubuntu).

What does 'similar' mean? Are all 'delete' operations similar?

Minimal surprise

Avoid situations where the user will be surprised by the behaviour of the system (‘why on earth did it do that?’).

Surprise is often caused by modal applications: same key may have different effects in different modes.

Recoverability

Allow the user to recover (easily) from errors.

Once upon a time, there were editors without an ‘undo’ command!

Get confirmation of irreversible actions (e.g. deletion). (But users learn to confirm automatically...)

Checkpointing/autosaving is a valuable technique.

User guidance

Covers several things. Context-sensitive help and tooltips, for example.

Provide meaningful error messages. *Segmentation fault* or *java.lang.NullPointerException* is not useful.

Make sure error messages are written from the *user's* point of view, not the system's. This is harder than it sounds - e.g., error may be flagged a long way from the code that mirrors the user experience

User diversity

Remember that users may be colour-blind, blind (therefore using a text-to-speech device), deaf (not hearing your alerts), not fluent in English (confused by your sentences), casual users, power users, . . .

Wherever possible, provide choice of, and in, interfaces.

Legal obligations enshrined in Equality Act 2010 (superseding Disability Discrimination Act 1995), applying in particular to websites.

W3C has Accessibility Guidelines - Priority 1 probably required, Priority 2 EU recommended. There have been court cases in the UK and the US.

Example

Suppose an application tries to save a file to disk. The save fails because the user's disk quota has been exceeded. The application does not indicate to the user that there has been any problem.

Which, if any, of the usability guidelines (User familiarity, Consistency, Minimal surprise, Recoverability, User guidance, User diversity) does the application violate?

1. None, the luser should have checked that there was enough space.
2. None, it's the operating system's job to tell the user if there isn't enough space
3. ?

Web Accessiblity Example

Some of W3C Priority 1 Web Content Accessibility Guidelines

- Provide a text equivalent for every non-text element
- Ensure that all information conveyed with color is also available without color
- Organize documents so they may be read without style sheets
- Ensure that equivalents for dynamic content are updated when the dynamic content changes
- Provide redundant text links for each active region of a server-side image map
- For data tables, identify row and column headers
- Ensure that pages are usable when scripts, applets, or other programmatic objects are turned off or not supported

Task analysis

Know what tasks users want to do with the system *and in what order and combination.*

E.g. suppose your application asks the user a question. It's annoying if to answer it, the user typically needs to look up information that's found elsewhere in the application.

User stories and use cases can be useful here: they give you a starting set of plausible usage scenarios.

Caveat: don't "play safe" by providing access to everything from every page: this can be confusing. Can be indicative of poor understanding of user tasks.

User interaction

How do users interact with the system? Choice depends on task *and on user preferences*. For example [Shneiderman]:

- Direct manipulation, such as drag-and-drop.
- Menu selection, perhaps on a directly selected object.
- Form fill-in, as typically used for data entry.
- Command language, as used by traditional systems.
- Natural language, usually as a front end to a command language but now e.g., Siri.

Exercise: give one advantage and one disadvantage for each of these styles.

Information presentation

How should information be *presented* to the user?

N.B. This need not and should not depend on how information is *represented* internally!

Perhaps as text, graphs, tables, pie charts, colour coded maps, . . .

E.g. continuously varying information is best presented in an analogue representation, not as numbers.

Data visualization means presenting (usually large) amounts of data in an abstracted visual representation, possibly with virtual reality navigation. (E.g. network congestion.)

Presentation should depend on the audience, e.g., graphs vs. graphics

See work by Edward Tufte.

Colour

deserves a lecture in itself. A few of guidelines [first 5 from Shneiderman]:

- Use few colours, no more than four or five per context, no more than seven in total.
- Colour changes should signal something significant.
- Colour code to help users, not for the sake of it.
- Colour code consistently. (E.g. if red means error in most of the application, don't use red for a normal stop button.)
- Be careful about colour pairings. Don't do this.
- In general, vary colours along only one of the three dimensions (*hue*, *saturation*, *brightness*) to make a distinction.
- Know the output technology. Primary green is often unreadable on screen, but would be readable in print.
- Remember that around 10% of men are red–green colour-blind!

Interface design and evaluation

Interfaces should be designed iteratively, regardless of the model for code design. *Only real end users* are good judges of the interface.

Design prototype interface; *evaluate* with users; *analyse* results. Repeat if necessary.

Then repeat whole process with actual interface.

In expensive or critical software, involve professionals from appropriate fields (e.g. ethnography).

Evaluation is hard. To do it properly – get professionals. Otherwise, read HCI books!

Reading

Required: Article on UK law about website accessibility

Suggested: Most popular articles at
`https://www.nngroup.com/articles/`

Suggested: The W3C Accessibility guidelines

Suggested: Sommerville, 10th Ed, Ch 29 on Interaction design (online).