

Usability

Nigel Goddard

School of Informatics
University of Edinburgh

Engineering is...

... managing the tension
between the constraints of the
mathematical universe and the
flexibility of human intelligence

Perdita Stevens

Of course not all software has users

If you're developing software with no human interface (e.g., a module deep within a system) then usability, strictly speaking, is not an issue.

However, developers of things that interact with your software are human, and in fact many of the same concerns arise.

UI design \longleftrightarrow API design \longleftrightarrow design

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 chapter 16.

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.
3. In a biospecimen inventory control system, the interface should represent things like freezers, freezer boxes, samples, and aliquots.

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.

Exercise: when was the last time you were surprised by a computer? How could the application have avoided surprising you? Next time you are surprised, consider how that could have been avoided.

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

Exercise: what was the last incomprehensible error message you encountered? What would have been better? Next time you get such a message, consider how it could be improved.

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 user 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 Accessibility Example

Some of W3C Priority 1:

- ▶ 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.

Instrument your UI!

Your best estimate of what users want to do is what users do.

- ▶ Log what they do
- ▶ Analyse the data
- ▶ Improve the experience

Easy with web UIs. Google Analytics does lots for you!

Unifying principle: mind the user's model

Humans instinctively spot patterns and build mental models of their world.

Users will build a mental model of how your software behaves.

When it violates their model, they will be confused.

So a good maxim is: try to keep the user's model and reality in sync. (NB you have some control over each.)

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!

Suggested exercises

1. Next time you use a (non-trivial) piece of software, analyse its interface carefully. How much better could you do by applying the principles we've discussed?
2. Consider the design of an API, e.g. the interface to a class which should be reusable. What makes it good/bad?

But do users have responsibilities?

Ideally, yes. E.g.

- ▶ use due care and attention
- ▶ RTFM
- ▶ report bugs sensibly

You can't make sure your users do these things, but you can make sure you do them when you're the user. Let's focus on the last.

Software entomology

Bugs and new features have something in common: both are requirements from users. How does one keep track of them?

Many projects use a *bug tracking system* for both bug reports and new feature requests. E.g. Bugzilla, Gnats, RT (used by our support). These provide extensive support for receiving, tracking, notifying, monitoring etc.

But much depends on the quality of the user input ...

Reporting: Trac

Ticket	Summary	Component	Version	Type	Owner	Status	Created
#1914	Units often refuse to mine metal and stone	UI & Simulation		defect		reopened	2013-04-14
#2000	OOS in a multiplayer game against Aegis	Core engine		defect		reopened	2013-06-24
#2158	Out of sync when played in multiplayer mode	Core engine		defect		new	2013-09-25
#2180	Aegis AI errors on Skirmish maps	UI & Simulation		defect	wraitii	new	2013-10-04
#2240	Rejoin Lockout	Core engine		defect		new	2013-10-26
#1819	[Git] [Step 1] Update autobuild utility for git work-flow	Build & Packages		enhancement	Philip	new	2013-01-12
#2116	[PATCH] Allow resetting splash screen	UI & Simulation		enhancement	Itms	new	2013-09-08
#1814	[Git] [Step 2] Create update.bat to sync windows binaries	Build &		task	Philip	new	2013-

OOS in a multiplayer game against Aegis

Opened **5 months** ago
Last modified **2 months** ago

Reported by: **scythewirler**

Owned by:

Priority: **Release Blocker**

Milestone: **Alpha 15**

Component: **Core engine**

Keywords:

Cc:

Description

I got another OOS error while playing on Oasis 9 against Aegis with a friend.

It seems there was a discrepancy concerning whether one of the AI's units was walking or idle between the client and the host.

This may be a duplicate of [#1881](#), but here are the logs just the case.

What can users do?

It's next year, you're doing your Operating Systems practical, you've got through parts 1 and 2, and you've started part 3. You've done a bit of work earlier in the day, and think you're in good shape for the deadline tomorrow. In the evening, you start up the system ... and it doesn't work.

You know that if you report the problem to the course forum there's a good chance that the lecturer will reply that evening.

What do you say in your post?

What do you think?

The single most important thing to do is:

1. keep the bug report concise - no more than 30 lines of 80 characters
2. include tediously detailed information about what exactly you did that didn't work
3. make an intelligent attempt at diagnosing the problem
4. include full information about what operating system etc. you're using
5. They are all important

How to deal with bugs as a user

and make programmers love you.

When something goes wrong, **STOP!** Don't do anything until you've engaged your brain.

Compose your bug report. This should:

- ▶ enable the developer to reproduce the bug. That means they need to know *exactly* what you did, what your system is, etc.
- ▶ failing that, tell them exactly what went wrong. Describe everything: what you typed/clicked, what messages appeared. Don't edit according to what you think relevant – you're probably wrong in what you think.
- ▶ If you can, try to diagnose the problem – but keep *your* diagnosis completely separate from the report of what happened.

Effective bug reporting

For more on this topic, see the following

Required Reading: How to Report Bugs Effectively, at
<http://www.chiark.greenend.org.uk/~sgtatham/bugs.html>

For you, you will often *be* the developer, and you have to deal with bugs in other people's software. Some of this may be open source. You may have to ask for help on stackoverflow or mailing lists. If so ...

Strongly Suggested Reading:
<http://www.catb.org/~esr/faqs/smart-questions.html>

Reading

Required: Article on UK law about website accessibility

Required: How to Report Bugs Effectively

Suggested: <http://www.useit.com/> especially the Alertboxes.

Suggested: The Smart Questions page

Suggested: The W3C Accessibility guidelines