The user's role

Perdita Stevens

School of Informatics
University of Edinburgh

Why are we starting here?

Partly for convenience – some of this lecture used to be at the end of the course, but it didn't have to be, and I want to use it in the assignment...!

But also because *most* software engineering issues – and, I claim, most of what makes it different from other engineering – essentially concerns humans (users, stakeholders, developers, maintainers...).

Their (your!) needs and capabilities matter.

Engineering is...

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

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.

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.

[This is largely taken from 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, an ATC system should present users with (representations of) aircraft, height, speed etc.; not anything to do with the implementation.

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.

Across the application? Across all applications?

Windows and Macintosh strive for great consistency. Traditional Unix doesn't.

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?

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, so this is of limited use.)

Checkpointing is a valuable techique. (One of the main reasons for using configuration management tools!)

User guidance

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

Provide meaningful error messages. *Segmentation fault* is not useful.

Make sure error messages are written from the *user's* point of view, not the system's.

Exercise: what was the last incomprehensible error message you encountered? What would have been better?

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.

Be aware of the Disability Discrimination Act

MCQ

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. How many 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. 1
4. 2
5. 3
6. 4
7. 5
8. 6

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.

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.)

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 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.

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 newsgroup, there's a good chance that the lecturer will reply that evening.*

What do you say in your post?

* the lecturer is Julian Bradfield, there really is :-)

Software entomology

Bugs and new features have somewhat 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 ...

MCQ

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

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.

Reading

Required: Page on UK law about website accessibility from RNIB (see webpage : v long URL!)

Required: How to Report Bugs Effectively

Suggested: `http://www.useit.com/` especially the Alertboxes.

Suggested: The Smart Questions page

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 Usenet or mailing lists. If so ...

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