User Interfaces

Lecture 23

User Interface Design

Hamza Bennani
hamza@hamzabennani.com

October 2, 2018

User Interfaces

- A common boundary or interconnection between a machine and a human being
- Its purpose is to allow a user to control the machine effectively
- Available technology has changed greatly over time

User Interface Design

- It is surprisingly easy to design bad user interfaces
  - e.g., start coding first, worry about the interface later
- How do you design a good user interface?
  - “Know thy users, for they are not you!”
- You must try to understand:
  - User’s goals in using the software you design
    - Specific tasks users undertake in pursuit of those goals
  - Their vocabulary in describing their actions
  - Their skill with similar software
  - Their attitudes toward similar software

How to understand potential users:

1. Direct Observation:
   - Interviews and on-site visits: find out user goals, common tasks, where the software might be used, etc.
2. Case Studies:
   - Interaction with representative users, e.g. after a prototype has been developed, or a redesign is being planned
3. Surveys:
   - Carefully designed surveys can allow you to gather statistics on how your software might be used
4. Personas:
   - Imaginary users and how they might interact with your software. Often quite detailed

User Research

Your users: occasional, intermediate, or expert?

- Occasional user: not motivated to learn interface
  - Kiosks in tourist centres or museums; online purchases
- Intermediate user: some motivation to learn interface
  - Microsoft Office; email clients; web browsers
- Expert user: uses your application a lot
  - Xcode; Adobe Illustrator; system administration tools
- Classification has nothing to do with a user's general skill on computers
  - Just because a user is a highly motivated expert doesn't mean they will tolerate a bad user interface

UI Design Patterns
Solutions to common UI problems

- Even if you don't do any user research at all, your applications should incorporate many of these patterns
- Safe exploration
- Instant gratification
- Satisficing
- Changes in midstream
- Deferred choices
- Incremental construction
- Habituation
- Spatial memory
- Prospective memory
- Streamlined repetition
- Support keyboard-only operation

Safe Exploration

"Let me explore without getting lost or getting into trouble."

- Undo/redo for document changes
- Back button, escape for navigation
- Easily unsubscribe from services on a webpage
- "Trash can" in the operating system

Instant Gratification

"I want to accomplish something now, not later."

- Predict the first thing most users want to do:
  - Google: enter a search term
  - Photoshop: open a canvas
  - Xcode: start a new project or open an old one
- Delay prompting users for information when an application first starts
  - No registrations, instructions, advertisements, etc.

Statisficing

"This is good enough. I don't want to spend more time learning to do it better."

- Satisficing (Satisfying + Sufficing)
  - People will sometimes accept good enough if best means more time or effort
  - Make labels short and easy to read
  - Organise layouts to communicate meaning (use dividers, boxes, whitespace, etc.)
  - Make navigation easy (use back buttons, escape, etc.)
  - Use simple interfaces to avoid visual overload

Changes in Midstream

"I changed my mind about what I was doing"

- Don't lock users into interactions they have to finish, except when absolutely necessary
  - e.g., installation wizards; modal dialogs; etc.
- Support reentrance (stop & continue later)
  - e.g., non-modal dialog boxes; draft emails; skip, defer, and other "I'll finish it later" buttons

Deferred Choices

"I don't want to answer that now; just let me finish!"

- Allow the user to answer only the bare minimum number of questions at any given time
- Mark fields that are required versus optional
- Hide optional fields
- Provide good defaults wherever possible
- Make it possible and straightforward to return to deferred choices later

Incremental Construction

"Let me change this. That doesn't look right; let me change it again. That's better."

- Creative activities require incremental improvements and constant testing e.g.: art, programming, writing
- Provide constant feedback e.g.: Does it compile? How does it work so far?
- Make sure program runs quickly e.g.: pre-compiled headers for quick building

Habituation

"That gesture works everywhere else; why doesn't it work here, too?"

- Consistency across applications:
- Provide standard key-actions, e.g., ⌘S for save, ⌘X, ⌘C, ⌘V for cut, copy, paste, etc.
- Consistency within an application:
- Avoid changing key-actions across different modes of operation
- Avoid confirmation dialogs
- If the user is habituated to clicking OK, then they will click OK without reading the dialog anyway

Spatial Memory

"I swear that button was here a minute ago. Where did it go?"

- Keep buttons and tools in the same relative locations within your application
- Don't resize windows, panels, etc. unless requested by user
- Re-open the application in the same state that it was when the user last quit macOS does this for you automatically
- Be careful when you hide details, or reorganise menu items since that can be disorienting

Prospective Memory

"I'm putting this here to remind myself to deal with it later."

- Examples include:
  - Calendars, reminders, alerts, etc.
  - Browser bookmarks
  - Email flags in inbox
  - Notes in a document (e.g., red highlighted text or track changes in Word, or a macro such as \todo in LaTeX)
- Provide flexibility to allow users to implement their own reminder system

Streamlined Repetition

"I have to do this how many times?"

- Automate repetitive tasks, examples include
  - Find and Replace dialogs
  - Photoshop has a "record actions" feature that records user's operations and can then play them back on different images
  - Unix style scripting allows extreme automation
  - AppleScript in macOS allows users to control an application

Support Keyboard-Only Presentation

"Please don't make me use the mouse."

- Keyboard alternatives to app. functionality
  - Keyboard shortcuts for menus, e.g., ⌘S for save
  - Selection from lists using arrows, shift-arrows, etc.
  - Tab between text fields; push buttons with return; toggle switches with space; dialogue box defaults
- Lots of this is standard with a Cocoa app.

Interface Idioms

- Interface idioms are UI styles that have become accepted for certain tasks
  - Word-processors/text editors, drawing/painting programs, games, IDEs, etc.
- Think about how your application fits in with existing idioms
  - If you adopt accepted styles your UI will be familiar and easy to use

Content Organisation Patterns

How you organise your application data and actions will help you determine how to organise your windows

- Two-Panel Selector, used in Mail, Finder
- Canvas + Palette, used in Illustrator, Photoshop
- One-Window Drill-down, often used on smartphones
- Alternative Views, used in Word, PowerPoint, Finder
- Wizards, for software installation, printer setup, internet purchases
- Extras on Demand, to hide complexity when not needed
- Multi-Level Help, different ways to access help

Two-Panel Selector

- The two-panel selector is a commonly used method to list objects and categories
- Used to show data and context of that data

One-Window Drill Down

- One-window drill-down shows one window of information at a time
  - Used in web browsers
  - Commonly used on handheld devices such as phones
- Can be combined with navigational aids such as back button or bread-crumbs

Canvas + Palette

Canvas + palette is almost always used for a drawing application

Alternative Views

- Alternative views can be combined with other patterns to allow additional flexibility in viewing information
- Commonly used in word processors, web browsers, etc., to provide different views of a document

Wizards

- Wizards provide restricted paths through potentially complicated, generally uninteresting processes
- Software installation, printer set-up, purchases on the internet, etc.

Extras on Demand

- Extras on demand: aim to hide complexity when it is not needed
- Use the 80/20 rule: show UI elements for 80% of the use cases, then use “extras on demand" for remaining 20% of the use cases

Multi-Level Help

Have help available in many different forms throughout your interface