Users and Usability Principles

CS 4640
Programming Languages for Web Applications

[Jakob Nielsen and Hoa Loranger, “Prioritizing Web Usability”,
Ben Shneiderman, Nicholas Diakopoulos, Steven Jacobs, Catherine Plaisant, Maxine Cohen, Niklas Elmqvist, “Designing the User Interface: Strategies for Effective Human-Computer Interaction”]
What Will You Learn About Usability?

- How to break down the **essential characteristics** of usable software from an **analytical** viewpoint

- **Engineering principles** for designing and building software interface that are
  - Fast to **learn**
  - **Speedy** to use
  - Avoid user **errors**
  - Increase **retention**
  - Improve user’s **subject satisfaction**

- How to **recognize** and **articulate** the difference between “this program sucks” and “I can improve this program by changing X, Y, and Z”

- **Life-long habits** for engineering usable products .. Usability is not just for software
Usability Principles of the Day

• **Understand** the users
• Design for the **user**
• Match the users’ **mental model**
• Follow the **7 ± 2** rule
• **Have a point, make** your point
• Prevent **errors**
• Reduce **excise tasks**
• Nine golden **rules** of UI design
• Shneiderman’s **five criteria**: Learn, Speed, Errors, Skills, SS
Understand the Users

It is important to know **who** the user is

- Work experience
- Computer experience
- Age
- Education
- Reading skills
- Language skills
- Work environment
- Task frequency
- … many more possibilities

Do users look at web apps the way they **are**?

Or do users look at web apps the way they **think**?
Design of UIs

- **Inside-out design**
  - Develop a system
  - Then add the interface

- **Outside-in design**
  - Design the interface
  - Then build the system to support it

When design decisions are made, either the developer must conform to the users, or the user must conform to the developer.

Web sites sink or swim based on the usability.
Mental Models

Information in the head
- what we memorize
- Knowledge for using a UI

Information in the world
- what we see

Declarative knowledge ("of")
- Facts and rules
- Easy to write down and teach
- Usually requires memorization

Procedural knowledge ("how")
- To accomplish a task
- Hard to teach and learn
- Taught by demonstration and learned through practice
- Requires deeper understanding

Imprecision

Syntactic constraints

Mental models
(users’ perception of reality)
Example: Driving a Car

• When we push the gas pedal, the car goes faster
  - Mental: pushing makes it go faster
  - Implementation: more gas to the engine, more pressure, pistons go faster, tires go faster …

• When we turn the wheel, the car turns
  - Mental: turning the wheel turns the tires
  - Implementation: turning the wheel turns something else (with help of a motor for power steering), which causes something else to turn, which puts the tires into a different angle
UIs and Mental Models

Telephones: I want to call mom, not 1-434-xxx-xxxx

Compile: I want to run my program, not compile, run

File manager: dragging a file from window to window is
• Move on the same disk
• Copy from USB thumb drive to disk

Calendars: paper calendars require paging,
online calendars can scroll
UIs and Mental Models

Predicting storms

• Does the prediction “30% chance of rain” mean
  • It will rain in 3 of 10 locations in the area
  • Less rain than if 50% chance, but more than if 10%
  • On 3 days out of 10 with these “weather conditions,” it has rained in the past

Dice

• If I roll a 6 five times in a row, what is the probability that my next roll will be a 6?

Interfaces should conform to the user’s mental model
Fundamental Software Design Principle
The 7 ± 2 Rule

- Human's short term memory can only hold about seven things at a time (plus or minus 2)

When we get more than about 7 items, we get confused
Have a Point, Make Your Point!

You have less than two minutes to convince first time visitors to stay on your web site

Every page must justify **WHY** the user should stay

---

**< OBVIOUS >**  
**< OBVIOUSLY CLICKABLE >**

---

**REQUIRES THOUGHT >**

---

**Report**

---

**Hmm. [Milliseconds of thought]**
I guess that’s the link. **Click**

---

**Report**

---

**Hmm. Does that do anything?**

---

[Figures from Steve Krug, “Don’t Make Me Think.”]
Preventing Errors

• People often make mistakes

• Faster computers can increase errors

• Prevention strategies:
  • Flow: Users make fewer mistakes when the flow through the UI make sense
  • Education: Better error messages can reduce errors

• The software can prevent the user from making dangerous choices

• Software seatbelts: If the dangerous choice must be available, allow it with a hesitation (“are you sure?”)
“Stuff” happens

• If an error is possible, someone will make it

• Good UI designers must assume all possible mistakes will happen

  • Design to minimize the chances of mistakes
  • Design to minimize the consequences of mistakes
  • Design to maximize recovery from mistakes

• Do not assume users are perfect
Helping Users Choose Action

• **Visibility**
  - The user can see the *state* of the system and how to use it

• **Good conceptual model**
  - The system works the way the users *expect*

• **Good mappings**
  - Users can see *relationships* between actions and results, controls and effects, and state and appearance

• **Feedback**
  - The system tells the user *what happened* at every step

> When something seems easy to use, it was probably hard to design
Reduce Excise Tasks

Don’t make me think

also means

Don’t make me do work that’s not related to my goal
Excise Tasks

- **Overhead** relates to solving problems:
  1. **Revenue Tasks**: Sub-tasks that work to solve the problem directly
     - Studying
     - Doing homework
     - Listening to lectures
     - Taking tests
  2. **Excise Tasks**: Sub-tasks that must be done but that are not really part of the problem
     - Driving to school
     - Parking!
     - Doing homework that does not reinforce concepts

- Excise tasks satisfy the needs of the **tools** or **process**, not the **users**
Example: GUI Excise

• Competent command lines users see a lot of excise in GUI – primarily the navigation
  • Using the mouse
  • Having to go through multiple screens
  • Generally – GUIs require more navigation

• Example: Changing background in all class slides
  • PPT: More than 30 minutes; load each file separately, 1 or 2 minutes to change each file
  • VIM: Less than five minutes; one process, repeat searching and commands (assuming text files)

• Convert 20 files to PDF
  • Word: about an hour, print dialog for each file
  • Latex: 3 minutes with a simple shell script, 10 by hand
Example: Command Line Excise

• Users must learn all the syntax – a significant tax!
  • Equivalent to learning programming languages
  • CLs are primarily preferred by programmers

• Command line users will often make extensive use of shortcuts and customization in GUIs
Techniques to Avoid Excise

• Put the *mouse focus* in the first input box

• Don’t *interrupt flow* unless necessary

• Try not to show *error* messages

• Don’t ask users to “**correct**” what they don’t understand

• Don’t *separate* input from output

• Don’t require *passwords* for everything
  • Authentication is almost always excise!

• Don’t make users *remember* where files are
  • MUST let users define file organization
  • MS Word does not, eclipse does not
Techniques to Avoid Excise (cont.)

• Don’t make users move or resize windows

• Don’t make users remember or reenter personal settings

• Don’t make users enter unnecessary data
  • Telephone number as a DB key – use the name or invent a number!

• Don’t make users confirm actions – unless undo is impossible
  • https://www.youtube.com/watch?v=3Sk7cOqB9Dk

• Avoid or correct errors
Memory – Auto-Customization

• Remember what the user did the last time

• Avoid unnecessary questions

• Imagine a boyfriend (or girlfriend) that asked you every time whether you wanted cream with your coffee!

• Dialog boxes ask questions, buttons offer choices
Nine Golden Rules of UI design

1. Build a UI that is consistent
2. Design usable and discoverable shortcuts
3. Provide appropriate feedback
4. Yield closure
5. Provide appropriate error handling
6. Allow users to undo all actions
   • Use hesitation for the operation that cannot be “undone”
7. Put the user in charge
8. Reduce the load
9. Design for the user
Shneiderman’s 5 Criteria for Measuring Usability

1. **Time to learn**: The time it takes to learn some basic level of skills

2. **Speed of UI performance**: Number of UI “interactions” it takes to accomplish tasks

3. **Avoiding user errors**: How often users make mistakes

4. **Retention of skills**: How well users remember how to use the UI after not using for a time

5. **Subjective satisfaction**: The lack of annoying features
1. Time to Learn

- **How long** it takes to learn to use an interface
- With complicated UIs, learning happens in “plateaus”
- **Well designed** interfaces make
  - The first plateau *easy* to get to
  - Subsequent plateaus *clearly* available

<table>
<thead>
<tr>
<th>Plateau 1</th>
<th>Plateau 2</th>
<th>Plateau 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>initial set of commands</strong></td>
<td><strong>additional commands</strong></td>
<td><strong>additional commands</strong></td>
</tr>
<tr>
<td>Ability to complete at least one simple task</td>
<td>More tasks, more choices, or more speed</td>
<td>More tasks, more choices, or more speed</td>
</tr>
</tbody>
</table>
2. Speed of UI Performance

• This is about **navigating** through the interface, **not** how fast the software or network runs

• *Interaction points* are places where the users interact with the software (e.g., buttons, text boxes, or commands)

• Speed of UI performance is roughly the **number of interactions** needed to accomplish a task

• Good UI designers need to reduce the number of *keyboard-to-mouse* switches

---

**Speed of user interface – NOT software, hardware, or network**
3. Rate of User Errors

• Users will always make **mistakes**

• UIs **can encourage or discourage** mistakes
  - Consistency, instructions, navigation, …

• Consider:
  - Entering letter **grades** in a dropdown instead of **radio buttons**
4. Retention of Skills

• “Once you learn to ride a bicycle, you never forget”

• Some interfaces are easy to remember, some are hard

• If they flow logically (that is, match the user’s mental model or expectations), they are very easy to remember

• If an interface is very easy to learn, then the retention is not important – users can just learn again

• Retention is typically more important with UIs that are hard to learn
5. Subjective Satisfaction

• Subjective satisfaction is defined to be how much the users “like” the UI

• How **comfortable** the users are with the software

• This depends on the user (thus the word “subjective”)

• Think of it in reverse: Users are **unhappy** when there is something annoying in the interface
  
  • Blinking
  
  • **Ugly colors**
  
  • Spelling errors in **massages**

• Most important in **competitive** software systems
  
  • Like … everything on the Web!
Tradeoffs Among Criteria

• We always have **tradeoffs** among the criteria

• Most people today equate “**user friendly**” with “**time to learn**” – this is a **narrow** view

• Making a UI **easier** to learn often **slows** it down!
  • Example: Many **GUIs** are easy to learn, but slow
  • Many **command languages** are fast, but hard to learn

• To be an effective UI designer, we must consider each criterion carefully and **prioritize before designing**

• Decide what is **acceptable** for each of the five criteria
Summary

• Good UIs take **time to design**

• Designing a good UI requires **thinking like the user** instead of an engineer
  • Engineers often think they are users

• **Different users** want **different things**
  
  Engineers love features
  They want to do everything the technology allows!

  All an interface designer has to do is
  • **Be polite**
  • **Be considerate**
  • **Be clear**