Today

- Recap OO design patterns
- User Interface (UI)
  - Usability
  - UI design
  - UI widget toolkits
  - Prototyping

Design pattern

- Addresses a recurring, common design problem.
- Provides a generalizable solution.
- Provides a common terminology.

1. Structural
   - Composite
   - Decorator
   - ...
2. Behavioral
   - Observer
   - Strategy
   - ...
3. Creational
   - Factory (method)
   - ...

Some uses of design patterns in UIs

- **Identify these patterns**: Composite, Decorator, Strategy, Observer
- **Examples from Java Swing**:
  1. ButtonModel and JButton
  2. JScrollPane
  3. JPanel, JScrollPane, JButton, JTextField, ...
  4. Composite.setLayout and FlowLayout, GridLayout, ...
Some uses of design patterns in UIs

- **Identify these patterns:** Composite, Decorator, Strategy, Observer
- **Examples from Java Swing:**
  1. ButtonModel and JButton: Observer
  2. JScrollPane: Decorator
  3. JPanel, JScrollPane, JButton, JTextField, …: Composite

Usability and software design

- **Usability:** the effectiveness of users achieving tasks
- Usability and good UI design are closely related
- A bad UI can have serious results...

Florida, 2018 (Broward County)

Florida 2018 Undervote
The Share of Voters Who Cast Ballots for Both Senate and Governor

Achieving usability

- **User testing and field studies**
  - having users try the product and gathering data
- **Evaluations and reviews by UI experts** (often from Human-Computer Interaction or HCI as well as Human Factors)
- **Prototyping**
  - Paper prototyping
  - Mock up tool prototyping
  - Code prototyping
- **Good UI design focuses on the user**
  - not on the developer, not on the system environment
How do we avoid bad UI?

• Learn from bad UI designs (i.e., past mistakes)
• Apply good UI design principles (or rules)
• Build UI prototypes

How do I know whether my UI is good or bad?

• What are the ways in which UI quality can be quantified?
• What are some examples of software you use that have an especially good/bad UI?
• What do you think makes them good/bad?

Schneiderman's 8 Golden Rules

1. Strive for consistency.
2. Give shortcuts to the user.
3. Offer informative feedback.
4. Make each interaction with the user yield a result.
5. Offer simple error handling.
6. Permit easy undo of actions.
7. Let the user be in control.
8. Reduce short-term memory load on the user.

UI design: Sample 1 and Sample 2

(from Designing the User Interface, by Ben Schneiderman of UMD, noted HCI and UI design expert)
UI design: Sample 1 and Sample 2

UI design: Sample 3 and Sample 4

UI design: Sample 2

Key issue with Sample 2

- Does not reduce short-term memory load on the user:
  - Large number of tabs
  - Large number of other components
  - Complicated layout
  - Many colors
UI design: Key issues with Sample 3

- Does not strive for consistency:
  - Not using standard icons

- Does not reduce short-term memory load on the user:
  - Many colors

UI design: Sample 5 and Sample 6

UI design: Key issues with Sample 6

- Does not offer informative feedback

- Does not offer simple error handling
UI widget toolkits

- What are common components of such widget toolkits?
- What commonly used desktop applications provide widget toolkits?
- For UI design, how to select among widgets (or components)?

<table>
<thead>
<tr>
<th>Common components of UI widget toolkits</th>
</tr>
</thead>
<tbody>
<tr>
<td>• A frame or window</td>
</tr>
<tr>
<td>• A label</td>
</tr>
<tr>
<td>• A button</td>
</tr>
<tr>
<td>• A check box</td>
</tr>
<tr>
<td>• A radio button</td>
</tr>
<tr>
<td>• A text field</td>
</tr>
<tr>
<td>• A list</td>
</tr>
<tr>
<td>• A combo box</td>
</tr>
<tr>
<td>• A menu</td>
</tr>
<tr>
<td>• A dialog box</td>
</tr>
<tr>
<td>• Other…</td>
</tr>
</tbody>
</table>

Commonly used desktop applications that provide widget toolkits

- Programming language environment (e.g., Java Swing, Eclipse SWT)
- Web browser (e.g., CSS/HTML/javascript)
- Operating system (e.g., GTK+ for Fedora, Aqua for Mac)

Other applications?
UI design – Supporting user actions

- Use buttons for single independent actions that are relevant to the current screen.
  - Try to use button text with verb phrases such as “save” or “save”, not generic: “OK”, “Yes”, “No”
  - Use mnemonics or Accelerators (Ctrl-S)
- Use toolbars for common actions.
- Use menus for infrequent actions that may be applicable to many or all screens.
  - Users hate menus! Try not to rely too much on menus. Provide another way to access the same functionality (toolbar, hotkey, etc.)

UI design – Selecting among alternatives

- Use check boxes for independent on/off switches
- Use radio buttons for related choices, when only one choice can be activated at a time

UI design – Eliciting other user input

- Use text fields (usually with a label) when the user may type in anything they want
- Use lists when there are many fixed choices (too many for radio buttons); all choices visible on screen at once
- Use combo boxes when there are many fixed choices; don’t take up screen real estate by showing them all at once
- Use a slider or spinner for a numeric value

UI design – Supporting multiple screens

- Use a tabbed pane when there are many screens that the user may want to switch between at any moment
- Use dialog boxes or option panes to present temporary screens or options
An example UI design

- Did the designer choose the right components? Assume there are 20 collections and 3 ways to search

Feedback on the example UI design

- For eliciting user input, consider a combo box instead of a list
- For selecting among YES/NO alternatives, consider a check box instead of a radio button
- The button labels are unclear — In particular, “Default” could be “Reset to Default”

What's the point of prototyping?

- Should I do it? — If so, when should I?
- Should I make my prototype on paper or digitally (either mock up or code)?

Prototyping

- **prototyping**: Creating a scaled-down or incomplete version of a system to demonstrate or test its aspects.

- Reasons to do prototyping:
  - aids UI design
  - provides basis for testing
  - team-building
  - allows interaction with user to ensure satisfaction
Some prototyping methods

1. Paper prototyping: a paper version of a UI
2. Mock up tool prototyping: a digital version of a UI
3. Code prototyping:
   • Written by hand
   • Automatically generated by UI builder

Creating a “paper” prototype

• gather materials
  – Paper, pencils/pens, scissors, …
  – Whiteboard, markers, …

• identify the screens in your UI
  – consider use cases, inputs and outputs to user

• think about how to get from one screen to next
  – this will help choose between tabs, dialogs, etc.

Example paper prototype screen
Some advantages/disadvantages of paper prototyping

- Can be done by non-technical people
- More conducive to working in teams
- Much faster to create than mock up or code and can change faster
- More visual bandwidth (can see more at once)
- Feels less permanent or final
- Doesn’t have a real-world look & feel

Some prototyping methods

1. Paper prototyping: a paper version of a UI
2. Mock up tool prototyping: a digital version of a UI
3. Code prototyping:
   - Written by hand
   - Automatically generated

Mock up tool prototyping

- Create a UI design by dragging/dropping components from a UI widget toolkit (e.g., Infragistics Indigo Studio)

- Uses of a mock up tool:
  - A static image of the UI design
  - An animation of the UI design

Demonstration of Indigo Studio mock up for Page Setup

https://www.infragistics.com/
Some advantages/disadvantages of mock up tool prototyping

- May be able to done by non-technical and/or technical people
- More conducive to working in teams
- Generally faster to create than code and can change faster than code
- Has a real-world look & feel

Some prototyping methods

1. Paper prototyping: a paper version of a UI
2. Mock up tool prototyping: a digital version of a UI
3. Code prototyping:
   - Written by hand
   - Automatically generated

Code prototyping

1. Written by hand
   - implement a quick version of your code
2. Automatically generated by UI builders (e.g., Visual Studio)
   - draw a UI visually by dragging/dropping UI components on screen to automatically generate the code for those components

Some advantages/disadvantages of code prototyping

- Generally needs to be done by technical people
- Not as conducive to working in teams
- Usually more effort than paper or mock ups
- Has real-world look & feel
- Generated code is very difficult to understand
Current course assignments

- Participation 2 (due today) and 3 (will be posted on Moodle)

- Homework 1: Code review, architecture, & design (is posted on the webpage)

- Final project selection (is posted on the webpage)

Homework 1

- Three in a Row game
  - Implemented in Java

- Topics:
  - Code review
  - Architecture patterns
  - Design patterns

- Due: September 24, 2020 9 AM EDT

Final project selection

- Form team of 4 or 5 students

- Select one of the following 4 topics:
  1. MSR 2020 mining challenge
  2. Replication study
  3. Model inference for inferring processes
  4. EleNa: Elevation-based navigation

- Due: Thursday September 24, 2020 9 AM EST

Final project: Selected topic

1. Read some background materials (e.g., papers, user manuals, code)
2. Start to develop
3. Create and give a mid-point presentation
4. Continue to develop
5. Create and give a final presentation
6. Put together final “documentation”
Final project:
MSR 2020 mining challenge objectives
- Read 8-10 papers
- Select one or more research questions
- Propose an approach to investigate the research question(s)
- Develop experiments to evaluate the proposed approach by applying to the provided dataset
- Study the experimental results


Final project:
Replication study objectives
- Read 4-5 papers
- Select one of the debugging/testing tools (e.g., SimFix)
- Learn about the benchmark (i.e. input data) for the selected tool
- Replicate the experiments to evaluate the selected tool by applying to the same data
- Extend the experiments to further evaluate the selected tool on additional data

Final project:
Model inference objectives
- Read 4-5 papers
- Select a model inference tool or tools (e.g., Synoptic)
- Select a reasonable way to generate traces
- Develop experiments to evaluate the selected tool(s) by applying to the generated traces
- Study the experimental results
  - Develop an automated approach to diff two FSAs

https://github.com/ModelInference/synoptic

Final project:
Elevation-based Navigation (EleNa)
- **Goal:** Develop a software system that determines, given a start and an end location, a route that maximizes or minimizes elevation gain, while limiting the total distance between the two locations to x% of the shortest path
- **Components:**
  - Data model that represents the geodata
  - A component that populates the data model, querying, e.g.,
    https://www.openstreetmap.org
  - The actual routing algorithm that performs the multi-objective optimization
  - Another component that outputs or renders the computed route
**Final project: EleNa objectives**

- Read any necessary technical documents
- Design two or more components
- Implement the designed component(s)
- Build a test plan for the implemented component(s) and carry out that test plan
- Demo at the final presentation
- Submit your documentation and version control repository