Today

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

### Design pattern

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

<table>
<thead>
<tr>
<th>1. Structural</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Composite</td>
</tr>
<tr>
<td>- Decorator</td>
</tr>
<tr>
<td>- ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2. Behavioral</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Observer</td>
</tr>
<tr>
<td>- Strategy</td>
</tr>
<tr>
<td>- ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3. Creational</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Factory (method)</td>
</tr>
<tr>
<td>- ...</td>
</tr>
</tbody>
</table>

### Some uses of design patterns in user interfaces

- **Identify these patterns:** Composite, Decorator, Strategy, Observer

- **Examples from Java Swing:**
  1. JButton and ButtonModel
  2. JScrollPane
  3. JPanel, JScrollPane, JButton, JTextField, ...
  4. Composite.setLayout and FlowLayout, GridLayout, ...
Some uses of design patterns in user interfaces

• **Identify these patterns:** Composite, Decorator, Strategy, Observable

• **Examples from Java Swing:**
  1. JButton and ButtonModel: Observer
  2. JScrollPane: Decorator
  3. JPanel, JScrollPane, JButton, JTextField, ...: Composite

User Interfaces (UIs)

How do we avoid bad UI?

• Learn from past mistakes

• Build prototypes
Big questions

- 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?
- What's the point of prototyping? Should I do it?
  - When should I?
  - Should I make my prototype on paper or digitally?

Usability and software design

- **usability**: the effectiveness of users achieving tasks
  - Human-Computer Interaction (HCI).
  - Usability and good UI design are closely related.

Achieving usability

- User testing and field studies
  - having users use the product and gathering data
- Evaluations and reviews by UI experts
- Prototyping
  - Paper prototyping
  - Code prototyping

- Good UI design focuses on the user
  - not on the developer, not on the system environment

Florida, 2018 (Broward County)
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.

(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 1 and Sample 2

UI design: 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: Sample 3 and Sample 4

UI design: Sample 3 and Sample 4

UI design: Sample 3 and Sample 4

UI design: Sample 3 and Sample 4

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

- **good**

- **bad**

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)?

Common components of UI widget toolkits

- A frame or window
- A label
- A button
- A check box
- A radio button
- A text field
- A list
- A combo box
- A menu
- A dialog box
- Other…
**Commonly used desktop applications that provide widget toolkits**

- Programming language environment (e.g., Java Swing, Eclipse SWT)

Other applications?

**Common 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)

---

**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 “Cancel”, 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

- Good UI dialog? 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”
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

Why do paper prototypes?

• much faster to create than code
• can change faster than code
• more visual bandwidth (can see more at once)
• more conducive to working in teams
• can be done by non-technical people
• feels less permanent or final
Where does paper prototyping fit?

When in the software lifecycle is it most useful to do (paper) prototyping?

- Requirements are the what and design is the how. Which is paper prototyping?

- Prototyping
  - helps uncover requirements and upcoming design issues
  - during or after requirements but before design
  - shows us what is in the UI, but also shows us details of how the user can achieve goals in the UI

Paper prototyping usability session

- user gets tasks to perform on a paper prototype
- observed by people and/or recorded
- a developer can "play computer"

Creating a paper prototype

- gather materials
  - paper, pencils/pens
  - tape, scissors
  - highlighters, transparencies

- 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.

Representing interactive widgets

- buttons / check boxes: tape
- tabs, dialog boxes: index cards
- text fields: removable tape
- combo boxes: put the choices on a separate piece of paper that pops up when they click
- selections: a highlighted piece of tape or transparency
- disabled widgets: make a gray version that can sit on top of the normal enabled version

- computer beeps: say "beep"
1. Select a group of 4 or 5
2. Draw a rough paper prototype for a music player application (e.g., WinAmp or iTunes).
   • Assume that the application lets you store, organize, and play songs and music videos.
   • Draw the main player UI and whatever widgets are required to do a search for a song or video.
   • After the prototypes are done, we’ll try walking through multiple UIs together.

Things to think about:
• How many clicks are needed? What controls to use?
• Could your parents figure it out without guidance?

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 of a music player application

https://www.infragistics.com/

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

Current course assignments

• Homework 1: Code review
• Final project selection
Homework 1: Code review

- Three in a Row game
- Implemented in Java

Topics:
- Code review
- Architecture & design patterns
- Implementation
- Testing

Due: Thursday February 27, 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: Tuesday March 3, 2020 9 AM EST

https://people.cs.umass.edu/~hconboy/class/2020spring/CS520/FinalProject.pdf

1. MSR 2020 mining challenge:
   Software heritage graph dataset

   Antoine Peltre, Diomidis Spinellis, Stefano Zacchiroli.
   The Software Heritage Graph Dataset: Public software development under one roof.
   preprint, bibtex

   Contents:
   - Dataset
   - Relational schema
   - Setup on a PostgreSQL instance
   - Setup on Amazon Athena
   - Setup on Azure DataBricks


2. Replication study

- Automatic generation of oracles for exceptional behaviors from Javadoc comments
- EvoSuite: Automated test generation
- Automated program repair tools
  - SOSRepair: Expressive Semantic Search for Real-World Program Repair?
  - SimFix: Automated Program Repair
Automated program repair (APR) tool: Architecture

Test suite, including passing tests and failing tests

Patched program (now passes ALL tests)

Automated program repair (APR) tool:
Example buggy program

1 int triangle (int a, int b, int c) {
2     if (a <= 0 || b <= 0 || c <= 0)
3         return INVALID;
4     if (a == b && b == c)
5         return EQUILATERAL;
6     if (a == b || b != c) // bug!
7         return ISOSCELES;
8         return SCALENE;
9 }

Example from:
https://cacm.acm.org/magazines/2019/12/241055-automated-program-repair/fulltext

Automated program repair (APR) tool:
Example test suite

<table>
<thead>
<tr>
<th>Test Id</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>Expected output</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>INVALID</td>
<td>Pass</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>EQUILATERAL</td>
<td>Pass</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>ISOSCELES</td>
<td>Pass</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>ISOSCELES</td>
<td>Fail</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>ISOSCELES</td>
<td>Fail</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>SCALENE</td>
<td>Fail</td>
</tr>
</tbody>
</table>

Example from:
https://cacm.acm.org/magazines/2019/12/241055-automated-program-repair/fulltext
Automated program repair (APR) tool:
Example patched program

```c
int triangle(int a, int b, int c) {
    if (a == -1 || b == -1 || c == -1)
        return INVALID;
    if (a == 1 && b == 1 && c == 1)
        return EQUILATERAL;
    if (a == 2 && b == 2 && c == 3)
        return ISOSCELES;
    return SCALENE;
}
```

Automated program repair (APR) tool:
Example human bug fixed program

```c
int triangle(int a, int b, int c) {
    if (a <= 0 || b <= 0 || c <= 0)
        return INVALID;
    if (a == b && b == c)
        return EQUILATERAL;
    if (a == b || b == c) // bug fix!
        return ISOSCELES;
    return SCALENE;
}
```

3. Model inference for processes:
Architecture

Log represented as a sequence of method calls

Model inference tool

Generated model represented as a finite state automation (FSA)


Model inference for processes:
Example log

```log
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
10:15:15 155.103 08:01:20:10 GET HTTP/1.1 check-out.php
```

Model inference for processes: Example generated model

Model inference for processes: Learning objectives

- Select a model inference tool or tools (e.g., Synoptic, InvariMint)
- 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

4. EleNa: Elevation-based navigation

- **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., OpenStreetMap
  - The actual routing algorithm that performs the multi-objective optimization
  - Another component that outputs or renders the computed route