Software Design II
Goals

- Design (don't implement) the details
- Understand UI decisions
- Understand coding decisions
- Understand algorithms/data structure decisions
- Practice
Recap last week

- Idea/Problem
- Use cases
- Workflow
- Components
- Functional Requirements
- Non-functional requirements
What's next?

Where do we begin?
Ideas

- Server interface
- 3rd-party interactions
- Database structure
- Internal client components
- Protocol
- Internal data structures
- UI
Exercise 1: UI

Design the UI for the web interface of a network intrusion detection/prevention system (on paper)
Exercise 1: UI

Spend 5 minutes with your partner: see how he or she navigates the interface, then switch.
Software GUI...
Which one is your?
Exercise 1: UI

Lessons Learned?
Why UI First?

- Communicate to customer!
- It's the “most important thing”
  - Well, sometimes....
- Feedback
- Important things to consider?
Important UI Considerations

- Who is your user?
- Response & feedback
- Instruction/Training
- Where to begin
- What is/are the main screen(s)?
  - “Dashboards”
- Security?
- Simplicity vs. Power
  - Cognitive Load
Cognitive Load/Overhead

What is it?
Cognitive Load

Too much cognitive load.

That's why I can't do the dishes. Too much cognitive load.

It's a load of something, anyway.
Cognitive Load/Overhead

Cognitive Overhead — “how many logical connections or jumps your brain has to make in order to understand or contextualize the thing you’re looking at.” - David Demaree
Cognitive Load/Overhead

http://techcrunch.com/2013/04/20/cognitive-overhead/
Why it's important

“Minimizing cognitive overhead is imperative when designing for the mass market. Why? Because most people haven’t developed the pattern matching machinery in their brains to quickly convert what they see in your product (app design, messaging, what they heard from friends, etc.) into meaning and purpose.”
Cognitive Simplicity

- Complex
  - iCloud
  - QR Codes
- Simple
  - Shazaam
  - Nintendo Wii
- More examples?
How to keep it simple

- Make people work more
- Real-time feedback
- Make it run slower
  - “The Labor Illusion: How Operational Transparency Increases Perceived Value”
- Test on drunk people
  - Or old people and young people
- Ask how they think it works
UI Design

Where to begin?
Where to begin

- What is the FIRST interaction with the user?
- What have you identified as the main use cases?
- What interface is needed for these use cases?
First Interaction

1. Customer says: "Customers can't figure out our user interface."
   Dilbert responds: "They should read the manual."
2. Dilbert says: "Our manual is more confusing than our user interface."
3. Support says: "They can use our online support database."
   Dilbert says: "That's more confusing than our manual."
4. Dilbert says: "We have no money to fix any of that."
5. Dilbert says: "In situations like this, I like to go to my special place."
6. Dilbert says: "Someday I hope to have a special place that's big enough for my entire body."
7. Dilbert says: "Problem solved."
Do you need another button/screen?

Why did you add this button to the user interface?

You told me to.

Why would I tell you that?

You always suggest random changes to create the illusion of adding value.

Well, remove that button. It's only on your copy.

© Scott Adams, Inc./Dist. by UFS, Inc.
“Wireframes”

- Illustrate the interface, without functionality
- Two types
  - Static
  - Dynamic
- Lots of tools, free and not.
Static vs. Dynamic

When should you use static wireframes?

When should you use dynamic wireframes?

Which should we use?
Static vs. Dynamic

What should YOU use for YOUR project?
Non-UI Projects

- What WILL you show the customer?
- What will you show during your demonstration?
- How will you show success?
- What is your interface?
- How will customers interact with your interface?
Group Exercise

- Design “first screen”
- Draw UI and associated screens for main use cases
  - In what ways can the user interact with it?
  - What happens on such interactions?
- Pick a team member to come to the board to draw the concept and describe it
"We're out of crayons..."
Group Exercise

• Where did you start?
• With which user did you start?
  – End-user? Advertiser? Restaurant managers?
Pseudocode

Why write it?
## Why Write It?

<table>
<thead>
<tr>
<th>Observed Activity</th>
<th>Pseudocode</th>
<th>Programming Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drawing Diagrams</td>
<td>Mental operations → perceptual operations</td>
<td>Lots of mental operations</td>
</tr>
<tr>
<td>Multiple Representations</td>
<td>Program structure visibility; don't need new representations</td>
<td>Hard to visualize structure</td>
</tr>
<tr>
<td>Communicating with colleagues</td>
<td>Show decisions &amp; motivations</td>
<td>Doesn't convey thinking about solution</td>
</tr>
<tr>
<td>Problem Exploration</td>
<td>Low overhead/Easy to make changes</td>
<td>High cost</td>
</tr>
</tbody>
</table>
## Why Write It?

<table>
<thead>
<tr>
<th>Observed Activity</th>
<th>Pseudocode</th>
<th>Programming Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Printing &amp; Annotating</td>
<td>Highlight &amp; mark arbitrarily</td>
<td>Can comment, still must tidy-up code</td>
</tr>
<tr>
<td>Indicating commitment</td>
<td>Bring attention to important areas</td>
<td>Code all looks the same</td>
</tr>
<tr>
<td>Making structure visible</td>
<td>Evolution of solution is visible</td>
<td>Previous versions not as apparent</td>
</tr>
</tbody>
</table>
# Why Write It?

<table>
<thead>
<tr>
<th>Observed Activity</th>
<th>Pseudocode</th>
<th>Programming Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>To-Do lists</td>
<td>Add notes in any format</td>
<td>Generally limited to comments</td>
</tr>
<tr>
<td>External memory aid</td>
<td>Portable and accessible</td>
<td>Fixed</td>
</tr>
<tr>
<td>Order of development</td>
<td>Any order</td>
<td>Semi-linear, not arbitrary</td>
</tr>
</tbody>
</table>

Published in: Human-Computer Interaction, 1994, Volume 9, pp. 225-246
Pseudocode Considerations

- Who is it for?
- Why are you writing it?
- How detailed?
- Avoid syntactic details – not programming language specific
  - Or is it?
Pseudocode

- Start with interfaces
- Write pseudocode for interface methods
- Identify new, private/internal methods
- Identify algorithms
- Identify external interactions/dependencies
- Write pseudocode for those
- Iterate....
Characteristics of a good API

What say you?
Characteristics of a Good API

- Easy to learn
- Easy to use, *even without documentation*
- Hard to misuse
- Easy to read and maintain code that uses it
- Sufficiently powerful to meet requirements
- Easy to extend
- Appropriate to audience
Process of API Design

• Gather requirements (not proposed solutions!)

• Use your API
  – Start BEFORE you've implemented it or even specified it completely
  – SPI even more important – write *three* plugins

• Maintain realistic expectations
  – Expect to make mistakes
API Principles

- Do one thing; do it well
  - Confusion over what to name it?
- Small as possible and no smaller
  - When in doubt, leave it out (hard to remove)
- Loose-coupling to implementation
- Minimize accessibility/Information hiding
- Names matter & Documentation matters
- Coexist peacefully with platform
- Consider performance
API & Performance

- Component.getSize() returns Dimension
- Dimension is mutable....
- Each getSize() allocates a Dimension
- Many, many, many, many object allocations
Class Design

- Minimize mutability
  - Thread-safe, simple, performance
- Subclass ONLY where it makes sense
  - is-a vs. has-a
- Design and document for Inheritance
  - Or prohibit it
Method Design

- Don't make client do anything the module could do
- Principle of Least Astonishment
- Report errors as soon as possible after they occur (compile time?)
- Provide programmatic access to all available data in string form (getStackTrace vs. printStackTrace)
- Overload with care
Method Design

- Use appropriate parameter and return types
  - Interface types vs. classes
  - Use most specific type available
  - Don’t use floating point unless it really, really makes sense
- Consistent parameter ordering (e.g. dst/src)
- Avoid long parameter lists
- Avoid return values that demand exceptional processing (null vs. empty collection)
Exception Design

- Indicate exceptional conditions – not for control flow!
- Favor unchecked exceptions
- For checked exceptions, provided accessors
Source

How to Design a Good API and Why It Matters by Joshua Bloch, Principal Software Engineer, Google

http://lcsc05.cs.tamu.edu/slides/keynote.pdf
Data Structures

- Analyze parameters to methods
- Analyze interfaces
- What data will be transferred
- How to manage the data?
- What data is persistent and what is dynamic?
- Where is it stored?
- What is external?
Protocols

- How did the data get there?
- In what format was the data?
- How is it serialized?
- What are your latency and throughput requirements?
- What standards already exist?
- How are they secured?
Pseudocode Exercise

- Functions, algorithms, and data structures
  - Server
  - Client
- Two teams for each component
Working together

- UI Design
- Pseudocode
- Algorithms
- Data structures
UML

What is it?
And here’s where the user begins his transaction...
That’s a terrible design! Give me that pen.
Here’s where the Boss kicks your butt because your design stinks!

Oh yeah? Well, here’s me hitting you with a hammer because your code is filled with bugs!
And this is me getting my revenge...
I think it was a mistake to send them to that UML seminar.

http://hackles.org
Pseudocode vs. UML

- UML accounted for 53% of variability in programming performance
  - Unique contribution of pseudocode is 1%
  - UML as sole predictor explains 65%
- Source: “Relationships between Logic Depiction, UML Diagramming, and Programming” @ 3rd annual conference of Computing and Information Technology Research and Education
UML: I don't use it...

But I have friends who do...