CS 520
Theory and Practice of Software Engineering
Fall 2022

Manual reviews

September 15, 2022
Today

- Recap: Architecture and design goals
- Manual reviews
- User interfaces
Recap: Software architecture and design

Goals
- Lower complexity: separation of concerns, well defined interfaces
- Simplify communication
- Allow effort estimation and progress monitoring
Recap: Software architecture patterns

- **Pipe and filter**
  
  - `grep CS520 grades.csv | cut -f 1 -d ',' | sort | uniq -c`

- **N-tier / Client-Server**

- **MVC (Model-View-Controller)**

Example 1:
Which architecture pattern is being applied?

Example 2:
Which architecture pattern is being applied?
Manual reviews: Overview

**Goal:** Find errors in the software development process ideally as early as possible

**Approach:** A systematic examination of the software artifacts (e.g., requirements, design documents, code, test plan, external documentation) by one or more people to search for errors
Manual reviews: Goals (for this course)

From the developers’ perspective:
- Understandability
- Modularity
- Extensibility
- Testability
- Debuggability

From the users’ perspective:
- Correctness
- Understandability
- Usability
Setup and goals

- 2-, 3-, or 4-person teams
- Examples
  - **First phase**
    - For each of 4 examples, decide whether it represents good or bad practice.
    - **Goal:** discuss and reach consensus on good or bad practice.
  - **Second phase** (known solutions)
    - For each example, try to understand why it is good or bad practice.
    - **Goal:** come up with one or more explanations or a counter argument.
Example 1: good or bad?

src/Model-exp.java
src/Model-old.java
src/Model.java
src/View.java
src/GraphicalView.java
src/GraphicalView-opt.java
src/TextualView.java
Example 2: good or bad?

```java
/**
 * Sets the width of this rectangle to the given positive number.
 * @param width The new width
 * @throws IllegalArgumentException when the width is a negative number
 */
public void setWidth(int width) { … }
```
Example 3: good or bad?

```java
public enum PaymentType {DEBIT, CREDIT}

public void doTransaction(double amount, PaymentType payType) {
    switch (payType) {
        case DEBIT:
            // process debit card
            break;
        case CREDIT:
            // process credit card
            break;
        default:
            throw new IllegalArgumentException("Unexpected payment type");
    }
}
```
Example 4: good or bad?

```java
public void addStudent(Student student, String course) {
    if (course.equals("CS520")) {
        cs520Students.add(student);
    } else {
        allStudents.add(student);
    }
}
```
Manual reviews: Proposed improvements

- Naming conventions
- Decomposition into packages, classes, and methods
- Type safety for methods (e.g., enum types instead of ints/strings)
- Pre- and post-conditions for methods (e.g., defensive programming techniques, run-time assertions)
Manual reviews: Proposed improvements (cont.)

- Version control system (e.g., git)
- Documentation (e.g., internal comments, javadoc comments, README files, build files)
- Test suite (e.g., junit)
- Configurable logging
A Closer Look at 12 Powerful Code Review Tools

In this section, we review the most popular static code review tools.

- Review Board
- Crucible
- GitHub
- Phabricator
- Collaborator
- CodeScene
- Visual Expert
- Gerrit
- Rhodecode
- Veracode
- Reviewable
- Peer Review for Trac

https://kinsta.com/blog/code-review-tools/
User interfaces: Usability and UI design

• **Usability**: the effectiveness of users achieving tasks

• Usability and good UI design are closely related

• A bad UI can have serious results...
Example: Florida, 2018 (Broward County)

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

- 3.7% of Senate Ballots Blank
- 1% or Less of Senate Ballots Blank
- More Votes for Senate than Governor
User interfaces: How do we avoid bad UI?

• Learn from bad UI designs (i.e. past mistakes)

• Apply good UI design principles (or rules)
  − Hold UI design reviews by human factors and UI experts

• Build UI prototypes
User interfaces:
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...
User interfaces:
Common UI widget toolkits

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

Other applications?
User interfaces:
Common UI 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)
User interfaces:
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)
Round 1: Good or bad UI design?
UI design:
Sample 1 and Sample 2
UI design:
Sample 3 and Sample 4
UI design:
Sample 5 and Sample 6
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
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
Example:
Indigo Studio mock up for Page Setup

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

• Could be 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
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 conductive to working in teams
• Usually more effort than paper or mock ups
• Has real-world look & feel
• Generated code is very difficult to understand