CS 520
Theory and Practice of Software Engineering
Spring 2022

Collaborative development

March 10, 2022
Recap: Test driven development process

1. Add new test case(s) for a new feature and run all test cases
   - The new test case(s) should fail
2. Implement that new feature and run all test cases
   - All tests should pass
3. Refactor the implementation as needed to improve its quality and run all test cases
   - All tests should still pass
4. Repeat

https://en.wikipedia.org/wiki/Test-driven_development
Recap: Some common refactoring patterns

- move class/field/method
- rename class/field/method
- extract class/field/method
- encapsulate field/method
- ...

E.g., https://refactoring.com/catalog/
Recap: Automated support for those patterns
Today

- Agile development
- Scrum
- Pair programming
- Collaborative development exercise
Agile development

• Fast paced
• Frequent releases
• Developer centered
  – Do we need managers?
Scrum

• A very popular flavor of Agile to rapidly iterate in Sprints
  – Each Sprint develops then releases the product

• Three pillars:
  – Transparency
  – Inspection
  – Adaptation

• Used by large tech companies such as Facebook, Google, Microsoft

https://www.scrum.org
Three roles

• Product owner
  – represents the customer specifying the goal

• Development team
  – Performs Sprints
  – delivers software product that satisfies that goal

• Scrum master
  – buffer between team and outside world
  – prevents distractions, barriers
Many aspects of Scrum

• Sprints
• Stand-up meetings
  – What did I do yesterday?
  – What will I do today?
  – Do I see any impediment from our goal?
• Reviews
Pair programming

• Requirements specification, designing, implementing, testing, etc.

• Pair-work facilitates
  – transparency
  – no single point of failure
  – decision making
  – focus
  – creativity
Collaborative development exercise

• Further develop a Figure editor available here: https://github.com/LASER-UMASS/cs520-Spring2020.git

• Form pairs that will collaboratively work on:
  – specification
  – design
  – Implementation
  – testing
Figure editor

This figure is not yet complete.
Figure editor (v1): MVC architecture

Separates data representation (Model), visualization (View), and client interaction (Controller)
Figure editor (v1): Model API

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>getCaption()</td>
</tr>
<tr>
<td>javax.swing.ImageIcon</td>
<td>getImage()</td>
</tr>
<tr>
<td>boolean</td>
<td>isComplete()</td>
</tr>
<tr>
<td></td>
<td>Returns true if this figure is complete, meaning its Image is non-null and its caption is non-null and non-empty, and false otherwise.</td>
</tr>
<tr>
<td>void</td>
<td>setCaption(java.lang.String newCaption)</td>
</tr>
<tr>
<td></td>
<td>Sets the caption to the given non-null and non-empty String.</td>
</tr>
<tr>
<td>void</td>
<td>setImage(javax.swing.ImageIcon newImage)</td>
</tr>
<tr>
<td></td>
<td>Sets the image to the given non-null ImageIcon.</td>
</tr>
</tbody>
</table>
Model (v1): FSA specification

• Complete the behavioral specification (written as an FSA)

NOTE) The violation state is implied (not shown).
Model (v1): Implementation

- Use the Model API and FSA for the implementation

```java
setImage

public void setImage(javax.swing.ImageIcon newImage)
Sets the image to the given non-null ImageIcon.
Parameters:
newImage - The ImageIcon must be non-null
Throws:
java.lang.IllegalArgumentException - if the ImageIcon is null
```
public class FigureModel
{
}
Figure editor (v1): Testing

- Write test cases for the Model
- All test cases should pass
Figure editor (v2): MVC architecture

Separates data representation (Model), visualization (View), and client interaction (Controller)
Figure editor (v1): Observer pattern

Observable
{abstract}

# observers:Set<Observer>

+ register(o:Observer)
+ unregister(o:Observer)
+ stateChanged()

MyObservable

- state:State

+ getState():State
+ setState(state:State)

<<interface>>

Observer

+ update()

MyObserver

+ update()
Figure editor (v1): Observer pattern

**Observable**

{abstract}

- # observers: Set<Observer>
- + register(o:Observer)
- + unregister(o:Observer)
- + stateChanged()

**MyObservableModel**

- state: State
- + getState(): State
- + setState(state: State)

**Observer**

<<interface>>

- + update()

**MyObserver**

- + update()

**Model**

**View**
Figure editor (v2): Implementation

Java provides the following two classes:

- [https://docs.oracle.com/en/java/javase/15/docs/api/java.desktop/java/beans/PropertyChangeSupport.html](https://docs.oracle.com/en/java/javase/15/docs/api/java.desktop/java/beans/PropertyChangeSupport.html)
- [https://docs.oracle.com/en/java/javase/15/docs/api/java.desktop/java/beans/PropertyChangeListener.html](https://docs.oracle.com/en/java/javase/15/docs/api/java.desktop/java/beans/PropertyChangeListener.html)

How could the Observer design pattern be implemented using these classes?
Figure editor (v2): Testing

- Perform regression testing with the existing test cases for the Model
- Add new test cases for the Observer design pattern
Topics covered

• Documentation, e.g.,
  – README, javadoc, internal comments
• Specification, e.g.,
  – Natural language, FSAs
• Architecture & design, e.g.,
  – Patterns (MVC, Observer)
  – Class diagrams
• Implementation
  – Pair programming
  – Java, Swing
• Testing
  – JUnit
Week 7 Participation Questionnaire

On Moodle, submit your collaborative exercise:

• Implementation
• Test suite