Objectives

- Picasso!

1

Review: Typical Trajectory of Projects

Hopefully, you're here!

I get it! I am writing code
and redesigning as necessary

I am confident enough to write a little code

I am starting to get it.
I have the mental model for the code base

This code is too complex!
I can't understand this/do this project!

Understanding/confidence

Time committed to project
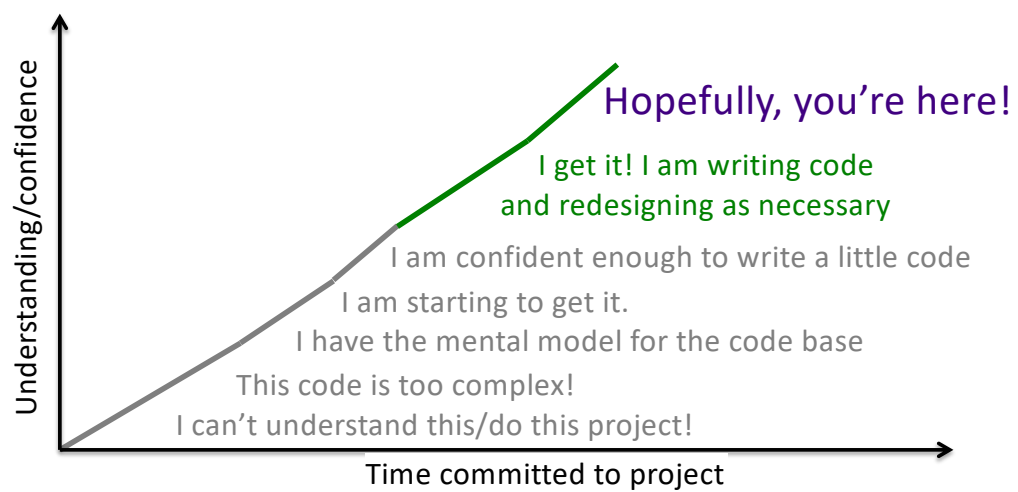
2

Review

- What is a design pattern?
  - ➢ What design patterns have we discussed?
    - What problems do they solve?
  - ➢ What design patterns are used in the Picasso project?
    - (This could vary by team)
- Why do we need to convert the input to postfix?
- What is our git workflow?
- What is a merge conflict? How do you resolve it?

3

Review: Design Pattern

> General reusable solution to a commonly occurring problem in software design

- Not a finished design that can be transformed directly into code
- Description or *template* for how to solve a problem that can be used in many different situations
  - ➢ "Experience reuse", rather than code reuse

4

Merge Conflict

- Occurs when competing changes to the same lines in a file
  - ➢ Git doesn't know how to resolve the merge
- Resolving: manually edit the conflicted file to what you want to keep in the merge
  - ➢ Stage change, commit and explain your fix
  - ➢ Push branch

5

PICASSO

6

Start Thinking About

- Reporting errors to users
  - ➤ Currently: in the printed output but users aren't going to see that
  - ➤ Helpful errors → translated for users
- Opening a file that contains an expression
- Handling new operations
  - ➤ Order of operations
  - ➤ Assignment statement
- Functions with multiple arguments, image names
- Extensions

7

Project Goals

- Everyone contributes significantly to the project
  - ➤ Has at least one part where they can say "I made this!"
- Everyone understands the code and its design
  - ➤ All of it. Well, 90% of it, at least at a high level
- Everyone feels valued as a team member

8

Contributing to the Team

- Always some concern that your grade is based on lines of code written
  - Number of lines of code is not a good indicator of work or quality of code

9

Tip: Comparing Binary Operators

- Likely need to implement the equals method in various classes (e.g., Addition, Subtraction, …)
- Stop after you've written two
- Compare the methods
  - Is there a code smell? Refactor!

10

Tip: Error Handling

- Don't do too much translation too soon
- Can mask *your programming* errors (that aren't user error errors)

11

Final Implementation: Documentation

- You leave, I'm still here, trying to use [grade] your code
- Documentation
  - ➢Extensions aren't always obvious
  - ➢State in README
- Javadocs: Purpose of Java classes
  - ➢Update comments
  - ➢Auto-generated daily
  - ➢Can be seen on the project web site

12

Final Implementation: Tagging

- While given code had compiler errors because of using test-driven development, there should be no compilation errors in final, tagged version

13

Secondary Goals

- You're going to figure out that your final design isn't perfect—maybe not even good!
  - ➢ Fix more critical and/or smaller things
    - Refactoring!
  - ➢ Note larger things
    - analysis/post-mortem due at end of finals week

> Good judgment comes from experience.
> How do you get experience?
> **Bad judgment** works every time.

14

Final Project: Project Analysis - Individual

- Understand teammates' design/code/parts
  - ➤ *At least* at a high level
- Contents: Description, Planning, Status, Code Analysis, Collaboration, Future Work
  - ➤ Complete specification online

15

Project Planning

- Review project specifications
- Make sure you know what tasks are left
  - ➤ Intermediate deadline provides some direction, but there are a variety of other tasks that can be implemented.
- Be agile!

16

Looking Ahead

- Wednesday: Course Retrospective
- Friday: Intermediate Deadline, Demo
- Finals Week
  - ➤ Thursday: Final Implementation Deadline
  - ➤ Friday - noon: Final Analysis

17