Objectives

- Prototypes
- Singleton Design Pattern

1

Review

- What is the Picasso project?
  - What are its components?
- What can we do to help our team succeed?
- What is the spiral model of development?

2

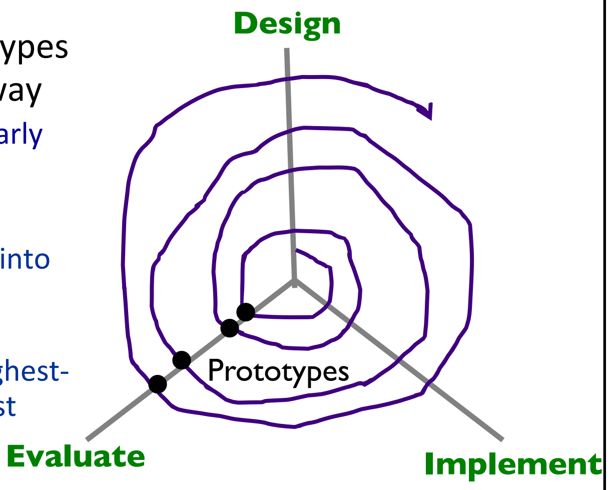
Review: Teams Work Best When They are **Interdependent**

- In code terms, we want *loose coupling*
  - Depend on each other but don't depend on their details
- Consider
  - Are you allowing your team to truly be interdependent?
  - Who might be you be ignoring?
  - Who might be allowing themselves to feel inadequate?
  - How do you show appreciation for each other and yourself?

3

Review: Spiral Model

- Idea: smaller prototypes to test/fix/throw away
  - Finding problems early costs less
- In general…
  - Break functionality into smaller pieces
  - Implement most depended-on or highest-priority features first



[Boehm 86]

Radial dimension: cost

4

PROTOTYPES

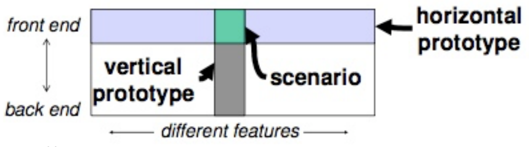
Prototypes Overview

- Demonstrate one part/purpose
  - Focus on one thing, not everything else
- Purpose/Dimensions
  - Functionality
  - Interaction
  - Implementation

Prototypes: Fidelity

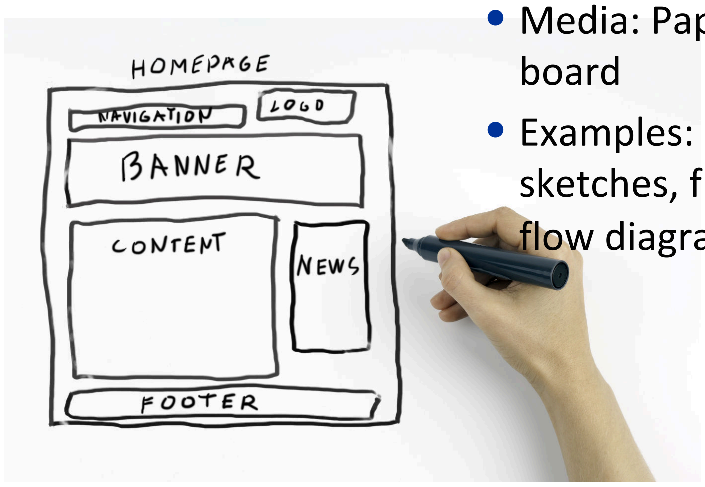
- ***Fidelity***: how similar to finished product
- Low: omits details
- High: closer to finished project
- Multi-dimensional
  - Breadth: % of features covered
    - Low-breadth: Only enough features for certain tasks
  - Depth: degree of functionality
    - Low-depth: Limited choices, canned responses, no error handling

From Nielsen, *Usability Engineering*



7

Low Fidelity Prototypes



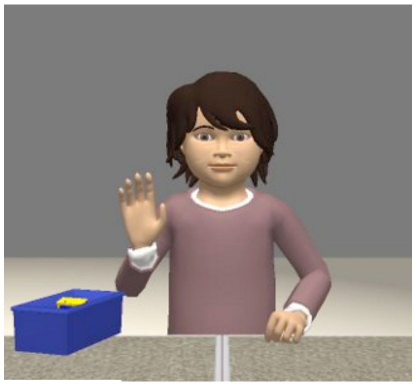
- Media: Paper, White board
- Examples: storyboard, sketches, flipbook, flow diagram

8

High Fidelity Prototypes

- Media: HTML (non-interactive), PowerPoint, Video
- Examples: Mockups, Wizard of Oz

Virtual Peer for Autistic Children



http://articulab.hcii.cs.cmu.edu/

9

What Kind of Prototype is Picasso?

- Both for given code and for preliminary implementation
- High fidelity with respect to the GUI
- Depth
  - From GUI → Backend → GUI
  - But limited implementation of GUI features and Picasso language

10

SINGLETON DESIGN PATTERN

11

Problem: Too Many Objects!

- Sometimes, we only want one object to ***ever*** be created for a class
- Often because there is some state that needs to be coordinated across the application

12

Solution: Singleton Design Pattern

- Make the constructor private
- Make a public method for accessing the one and only instance

Solution: Singleton Design Pattern

- Make the constructor private
- Make a public method for accessing the one and only instance (a static variable)

```
public class SemanticAnalyzer implements SemanticAnalyzerInterface {

    private static SemanticAnalyzer ourInstance;

    public static SemanticAnalyzer getInstance() {
        if (ourInstance == null) {
                ourInstance = new SemanticAnalyzer();
        }
        return ourInstance;
    }

    private SemanticAnalyzer() {
        …
    }

    public ExpressionTreeNode generateExpressionTree(Stack<Token> tokens)
```

Access to object

Private constructor

When Does Picasso Use the Singleton Design Pattern?

- Specialized analyzers need to refer to *the* SemanticAnalyzer to parse its functions/operations

```
return new Floor(
        SemanticAnalyzer.getInstance().
                generateExpressionTree(tokens) );
```

- Need to call methods on that one-and-only object

15

Is Picasso's Use of the Singleton Design Pattern the Best Design?

- Is this the best design? <shrug/>
- Alternative 1, could pass in the SemanticAnalyzer as another parameter:

```
public ExpressionTreeNode
   generateExpressionTree(Stack<Token> tokens,
        SemanticAnalyzer semAnalyzer);
```

- Alterative 2: could make SemanticAnalyzer's methods be static
  - Requires making state static

None of these changes are required; just explaining alternatives

16

Picasso Code: ReferenceForExpressionEvaluations

This implementation (from the “old” version of the code) is different than what we will have.
But, it is a helpful reference.

```
…
PLUS {
    public RGBColor evaluate(RGBColor left, RGBColor right) {
        double red = left.getRed() + right.getRed();
        double green = left.getGreen() + right.getGreen();
        double blue = left.getBlue() + right.getBlue();
        return new RGBColor(red, green, blue);
    }
},
…
```

What are `left` and `right` referring to?

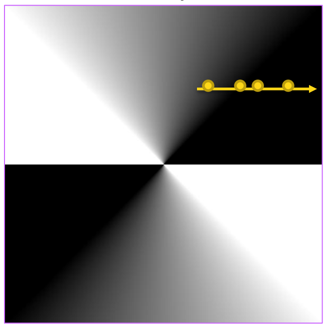
17

x/y is not the same as y/x

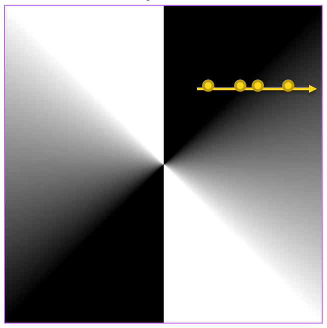
(placement of points is not exact in illustration)

Consider points, holding y steady at -.5

x/y



y/x



Y	X	.3	.45	.55	.7
Y=-.5		-.6	-.9	-1.1	-1.4
Color:		Mid-gray	Dark gray	Black	Black

Y	X	.3	.45	.55	.7
Y=-.5		-1.67	-1.11	-.91	-.71
Color:		Black	Black	Dark gray	Mid dark gray

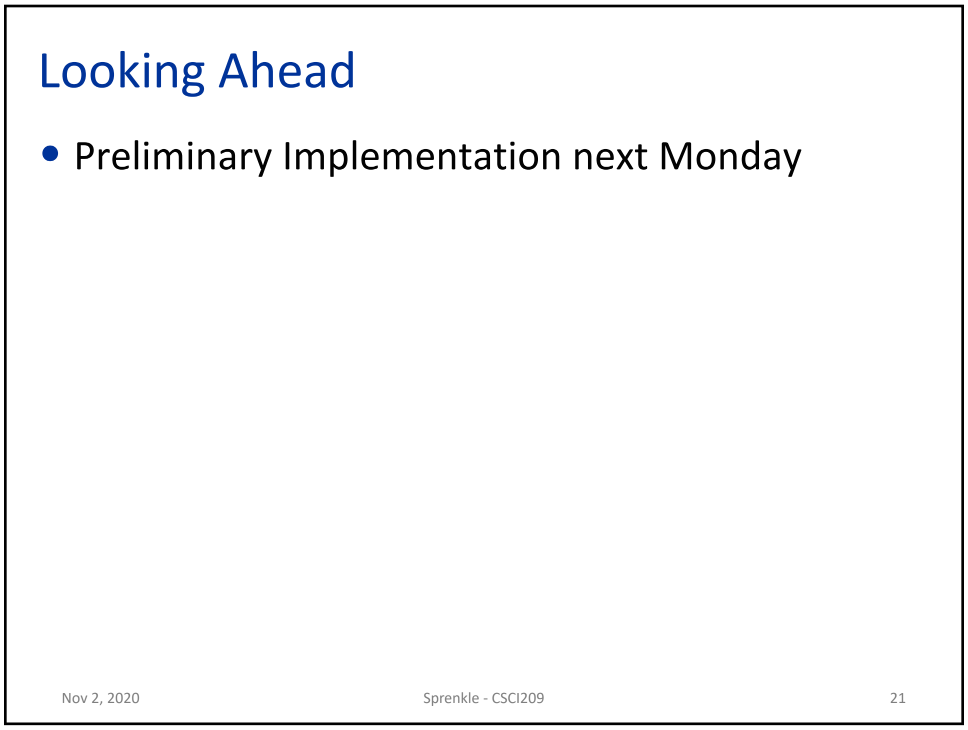
18

Team Collaboration/Planning

- An hour of thinking/design will save hours of coding

Preliminary Implementation

- Goals
  - Get your team working together
  - Find kinks in design
    - Rework now instead of later
- Tag your version
- Can keep working after that
  - Return to the tagged version for Monday’s demo



Looking Ahead

- Preliminary Implementation next Monday

21