Objectives

- Compiler Optimizations
- Python vs Java
- Software Development
  - Agile

Review

Get out the compiler optimization handouts

1. What is garbage collection?
   - How does it work in Java?
   - What are the tradeoffs in garbage collection?
2. What does the compiler do?
   - How is compiling different from interpreting?
3. Compare Java and Python
   - First, focus on their characteristics (just the facts, not tradeoffs)
   - Then, think about pros and cons, preferences

Compiled vs Interpreted Languages

In pure forms

Compiled

- – Spends a lot of time analyzing and processing the program
- Resulting executable is some form of machine- specific binary code
- Computer hardware interprets (executes) resulting code
- ✓ Program execution is fast
  - Efficient machine/byte code generation
  - Performance gains

Interpreted

- ✓ Relatively little time spent analyzing and processing the program
- Resulting code is some sort of intermediate code
- Another program interprets resulting code
- – Program execution is relatively slow
- ✓ Faster development/prototyping

Compiler Optimization Examples*

- What is the optimization?
  - ➢ How is the resulting code more efficient?
- For each optimization approach, generally,
  - ➢ should you make these optimizations yourself?
  - ➢ Or, is it something that only the compiler should do?
  - ➢ Key question: what is likely to change?

*Not literally what the code optimizations look like

- Optimizations are in byte code
- CSCI210 may help illuminate why these decrease runtime

Compiler Optimization: Example 1

Original:

```
for(int i = 0; i < 10; i++ ) {
        int j = 10;
        System.out.println(i + ",  " + j);
}
```

Optimization 1

```
int j = 10;
for(int i = 0; i < 10; i++ ) {
        System.out.println(i + ",  " + j);
}
```

Optimization 2

```
for(int i = 0; i < 10; i++ ) {
        System.out.println(i + ",  " + 10);
}
```

Compiler Optimization: Example 2

Original:

```
for( int i = 0; i < 10; i++ ) {
        if( i == 0 ) {
                System.out.println("Do this");
        }
        else {
                System.out.println("Do that");
        }
}
```

Optimization 1

```
System.out.println("Do this");

for( int i = 1; i < 10; i++ ) {
        System.out.println("Do that");
}
```

Optimization 2

```
System.out.println("Do this");
System.out.println("Do that");
System.out.println("Do that");
System.out.println("Do that");
…
```

Compiler Optimization: Example 3

Original:

```
public void f(int i) {
    a[0] = i + 0;
    a[1] = i * 0;
    a[2] = i - i;
    a[3] = 1 + i + 1;
}
```

Optimization 1

```
public void f(int i) {
  a[0] = i;
  a[1] = 0;
  a[2] = 0;
  a[3] = i + 2;
}
```

Compiler Optimization: Example 4

Original:

```
int add(int x, int y) {
    return x + y;
}

int sub(int x, int y) {
    return add(x, -y);
}
```

add method stays the same

Optimization 1

```
int sub(int x, int y) {
    return x + -y;
}
```

Optimization 2

```
int sub(int x, int y) {
    return x - y;
}
```

Compiler Optimization: Example 5

```
class Parent {
        void final f() {
                System.out.println("f");
        }
}
```

```
for( Parent p : parentArray ) {
        p.f(); // check p's actual type at runtime
                // and execute its method f
}
```

Optimization:

```
for( Parent p : parentArray ) {
        System.out.println("f");
}
```

Compiler Tradeoffs

- Upfront costs
  - Searching for optimizations
  - Make optimizations
    - Typically not Big-O efficiency improvements (unless program is written really inefficiently)
  - Iterative process: make optimizations and then look for more optimizations
- Improved runtime
  - Expect executed many more times than compiled

Different Perspectives on the Program

To the Compiler

- This is my one shot to validate the program and optimize it!

To You/Developer

- The long view: I am compiling the program now, but I could change the program later.
    - It should be easy to update the program; otherwise, I could introduce bugs.

LANGUAGE COMPARISON

Language Comparison

Java

Python

1) Focus on their characteristics (just the facts, not tradeoffs)
2) Pros and cons, preferences

Language Comparison

Java

- Entirely Object-oriented*
  - Procedural
  - Functional - newer
- Statically, strongly typed
- Compiled

Python

- Object-oriented
  - Also procedural and functional programming
- Dynamically, strongly typed
- Interpreted

Pros and cons of using each?

Rest of the Semester

- Shift from learning Java, specifically, to learning how to develop software (abstractly) with Java as our implementation/example
- Why Java?
  - Popular language
  - Many frameworks and tools for Java
  - Java’s structure allows for strict adherence to design techniques
- Just a start on Java
  - You’ll need to continue learning more Java on your own

SOFTWARE DEVELOPMENT

Programming is not Software Engineering

"It's Programming if 'clever' is a compliment.
It's Software Engineering if 'clever' is an accusation."
-- Titus Winters, Google Software Engineer

- This course is software development…
  - We're moving from programming towards software engineering
  - One metric: how long you think before you code

https://twitter.com/tituswinters/status/1143595692113481728

Extra credit

No Silver Bullet: Essence and Accidents of Software Engineering

“Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these, one seeks bullets of silver that can magically lay them to rest.

“The familiar software project, at least as seen by the nontechnical manager, has something of this character; it is usually innocent and straightforward, but is capable of becoming a monster of missed schedules, blown budgets, and flawed products. So we hear desperate cries for a silver bullet—something to make software costs drop as rapidly as computer hardware costs do.

“But, as we look to the horizon of a decade hence, we see no silver bullet. **There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity**. In this article, I shall try to show why, by examining both the nature of the software problem and the properties of the bullets proposed.”

by Frederick P. Brooks, Jr., 1986

Software Engineering

- Software Engineering is a relatively new field
  - Still learning best practices

Takeaway: We will employ lots of techniques that help make software development process more efficient without sacrificing software quality

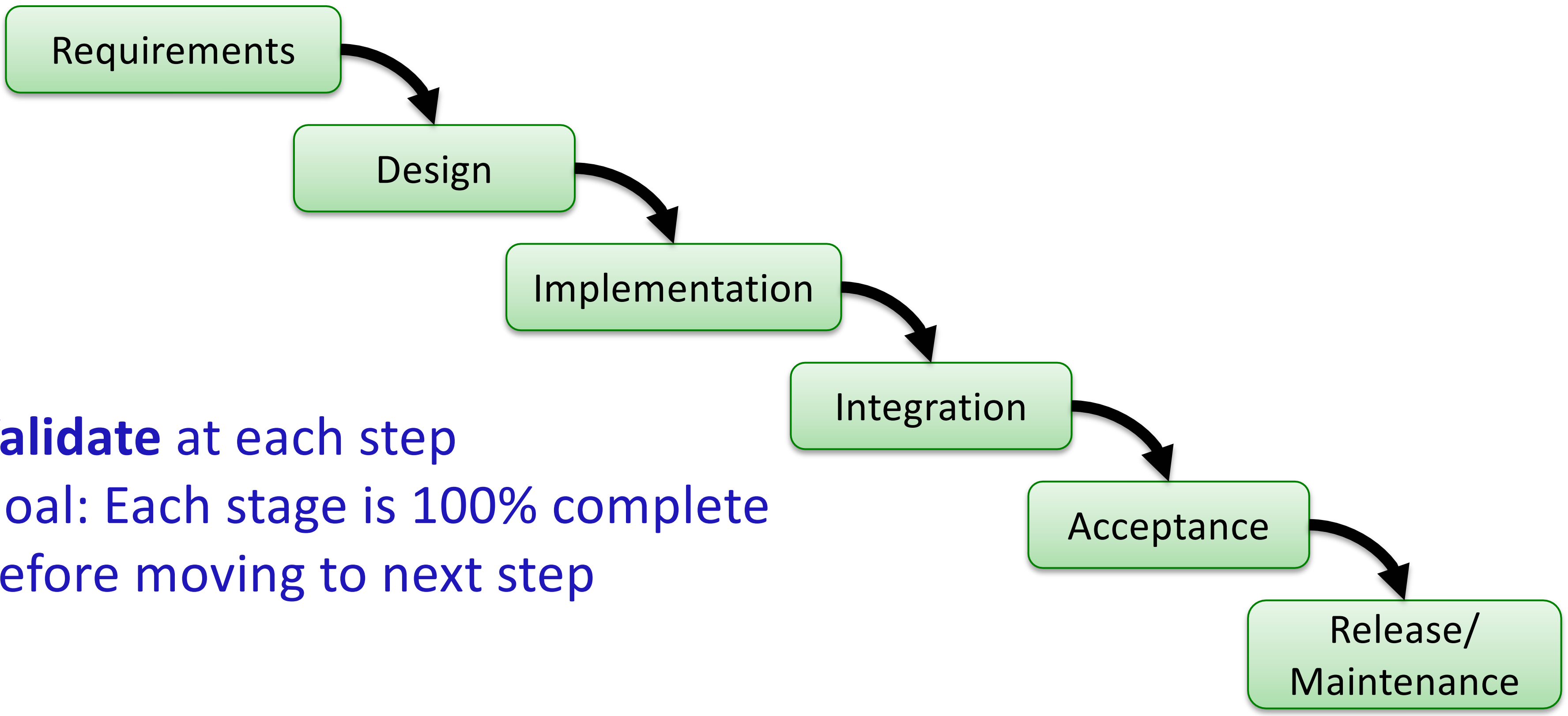
How to Implement an Effective Solution

1. Understand the problem
2. Understand external constraints
3. Design an effective solution to the problem
4. While designing the solution, design some ***tests*** to verify that the problem is solved (and remains solved)
5. Code the effective solution to the problem
6. Teach other team members about your solution to the problem

How to Implement an Effective Solution

1. Understand the problem (interact with ***people***)
2. Understand external constraints (interact with ***people***)
3. Design an effective solution to the problem
4. While designing the solution, design some ***tests*** to verify that the problem is solved (and remains solved)
5. Code the effective solution to the problem
6. Teach other team members about your solution to the problem (interact with ***people***)

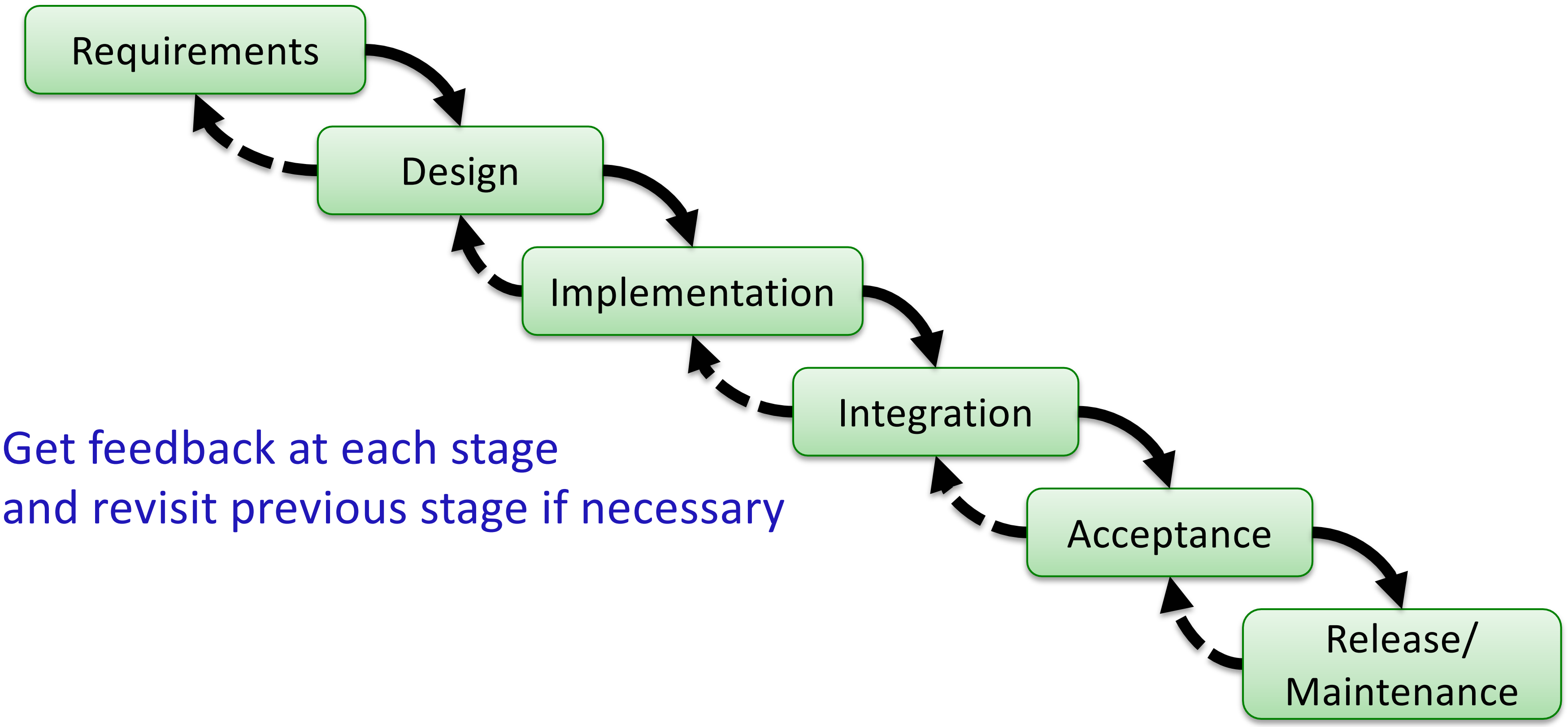
Traditional Software Engineering Process: Waterfall Model



Validate at each step

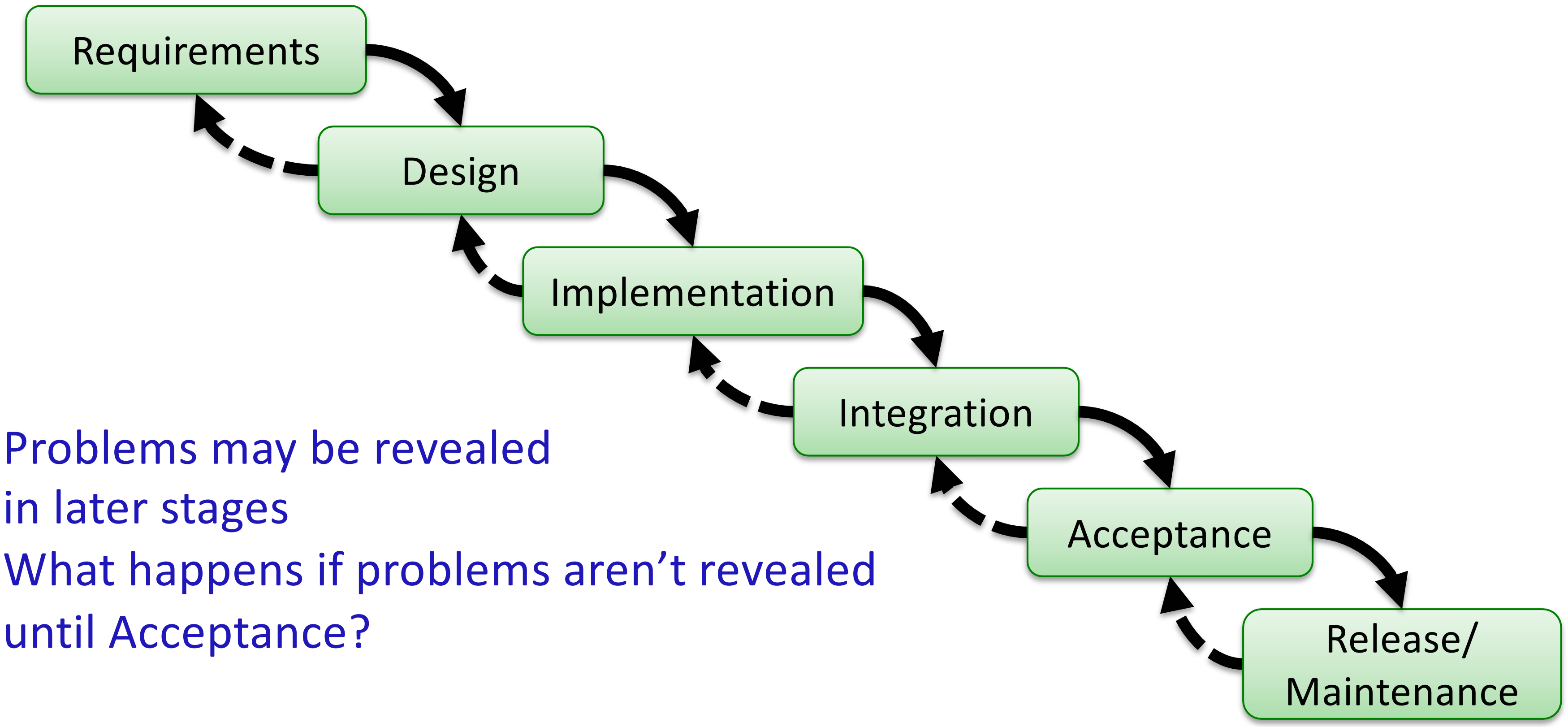
Goal: Each stage is 100% complete before moving to next step

Feedback in Waterfall Model



- Get feedback at each stage
  and revisit previous stage if necessary

Feedback in Waterfall Model

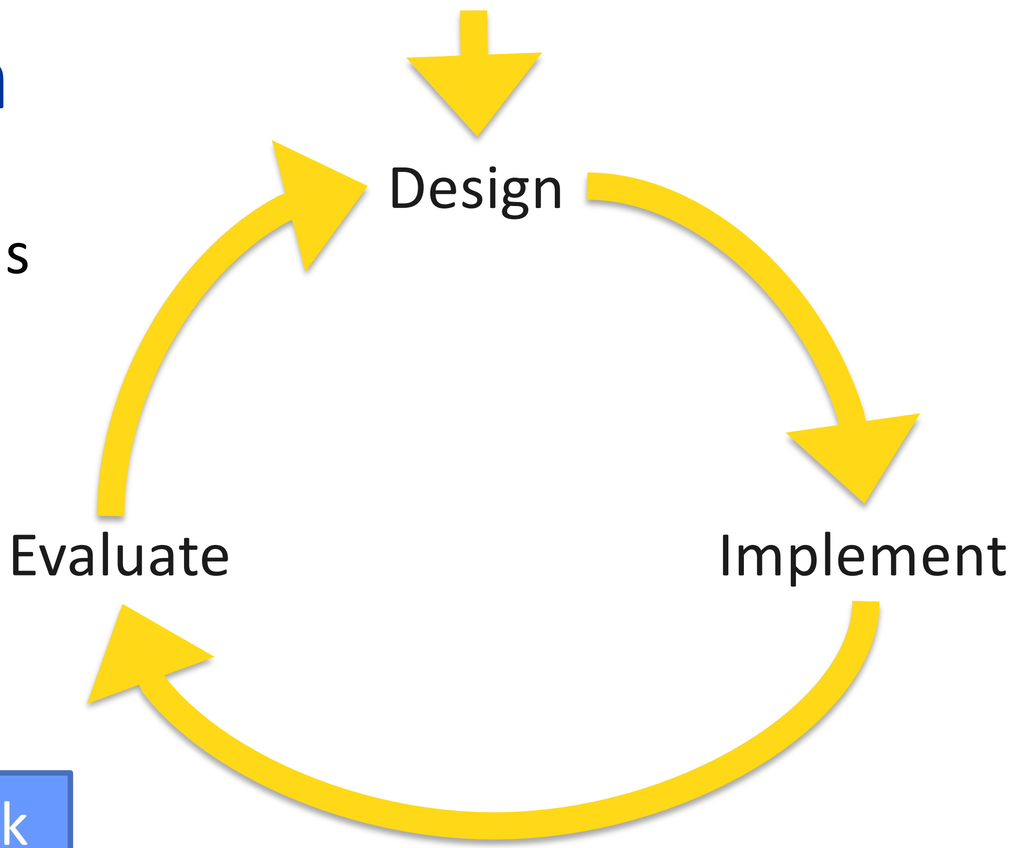


- Problems may be revealed in later stages
- What happens if problems aren't revealed until Acceptance?

Iterative Design

Various implementations

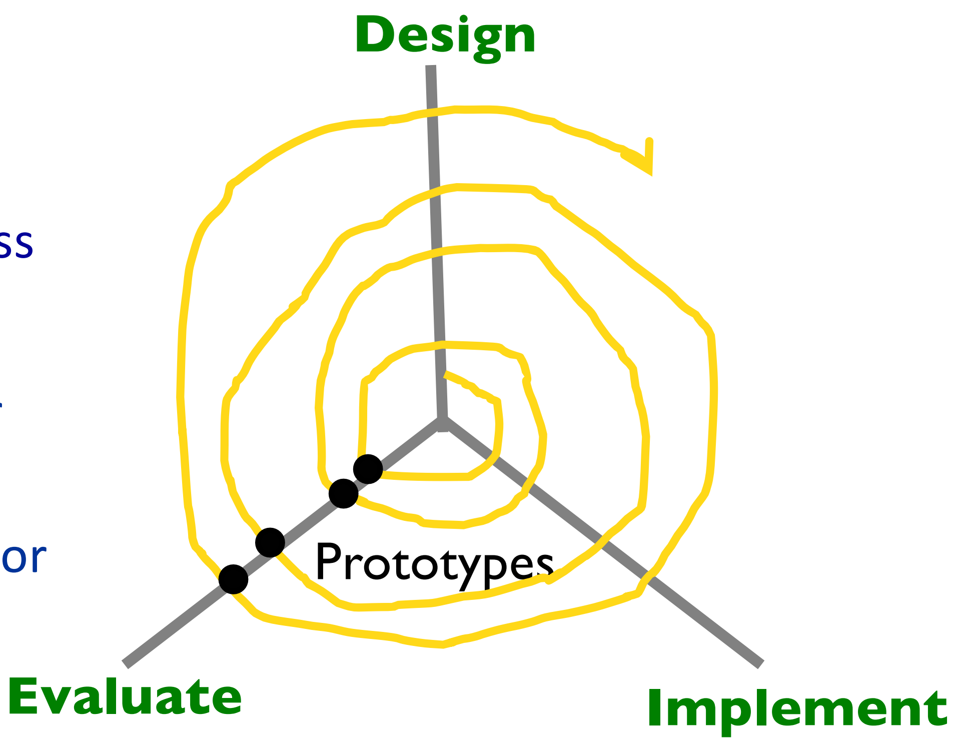
Get feedback/requirements from users/clients



Goals: Frequent feedback
→Identify problems early
→Higher quality product

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



Radial dimension: cost

[Boehm 86]

Looking Ahead

- No office hours today
- Assignment 5 due next Friday