# Objectives

- Java wrap-up
  - Compilation benefits
  - Comparing with Python
- Software Development



# Review

- What decisions do you make when choosing how to use Java streams?
  - E.g., choosing which streams to create for your I/O?
- What are the tradeoffs to Java's design decisions on streams?

# COMPARATORS



## Alternative Sorting

- What if object is `Comparable` but does not sort the way you want?
  - Special case
    - Don't want to change class
    - Don't have access to class
  - Example: want to sort strings so capital and lowercase letters are the same
- Use `Comparator` interface

# Comparator<T> Interface

- Declares two methods:
  - `int compare(T o1, T o2)`
    - Compare two objects and return a value as if we called `o1.compareTo(o2)`
  - `boolean equals(Object other)` ← Have default from `Object`
    - Check if this `Comparator` equals `other`
- Overloaded versions of `sort` in `Arrays` and `Collections`
  - `Arrays: void sort(Object[] array, Comparator c)`
  - `Collections: void sort(List list, Comparator c)`

`EmployeeNameComparator.java`



## COMPILATION

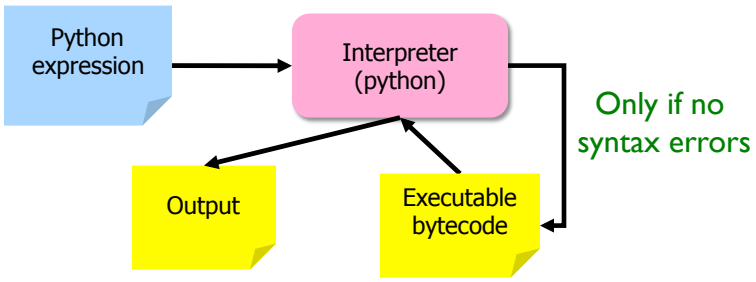
## Review

- How is compiling different from interpreting?
  - What does the compiler do?



## Python Interpreter

1. Validates Python programming language expression(s)
   - Enforces Python syntax rules
   - Reports syntax errors
2. Executes expression(s)



# Java Compiler


Source code Executable code

- Lexical analysis, parsing, semantic analysis, *code generation*, and *code optimization*
- Code optimization: dead code eliminator, inline expansion, constant propagation, …



# Compiling

- Translates high-level programming language to machine code or byte code
  - Java: .java → .class == bytecode
- Compiler optimization techniques
  - Generate *efficient* bytecode/machine code
  - Examples: get rid of unused local variables, transform loops, inline method calls
  - In Java: static typing for additional gains
- Can execute generated code multiple times
  - Performance gain
  - Interpreted → have to re-verify the code each time executed

## Compiler Optimization Examples

- What is the optimization?
  - How does it make the code more efficient?
- Why is it okay for the compiler to write code this way but not for you?



## Compiler Optimization Examples

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

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

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

# Compiler Optimization Examples

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

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

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

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



# Compiler Optimization Examples

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

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

## Compiler Optimization Examples

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

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

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

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



## Compiler Tradeoffs

- Upfront costs
  - Searching for optimizations
  - Make optimizations
    - Typically not Big-Oh efficiency improvements (unless programmer is really bad)
- Improved runtime

# LANGUAGE COMPARISON



## Language Comparison

**Java** **Python**

## Language Comparison

**Java**

- Entirely Object-oriented*
- Statically, strongly typed
- Compiled

**Python**

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

Pros and cons of using each?



## Summary: Compiled vs Interpreted Languages

**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

## No Silver Bullet: Essence and Accidents of Software Engineering *by Frederick P. Brooks, Jr., 1986*

"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.

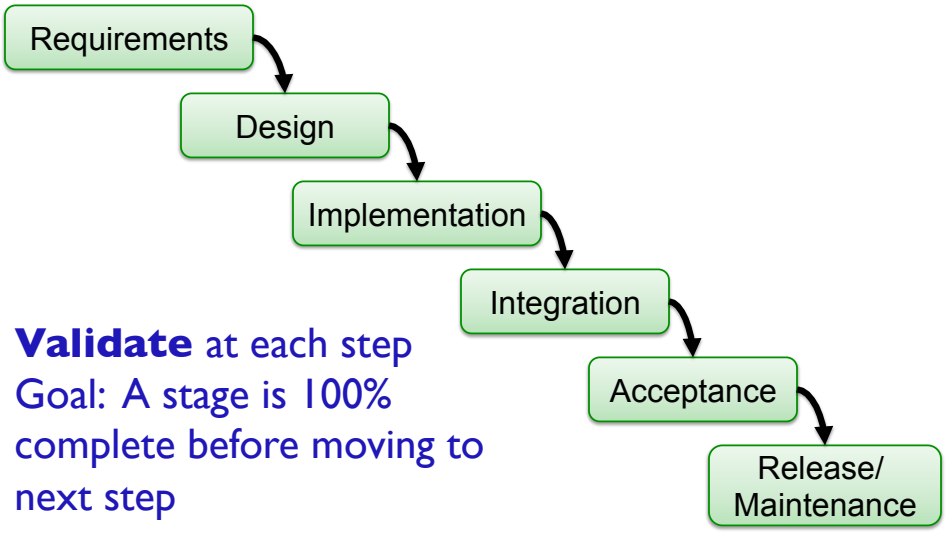
"But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity. In this article, I shall try to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



## Who is Fred Brooks?

- UNC Professor
- Turing Award winner
- "The most important single decision I ever made was to change the IBM 360 series from a 6-bit byte to an 8-bit byte, thereby enabling the use of lowercase letters. That change propagated everywhere."

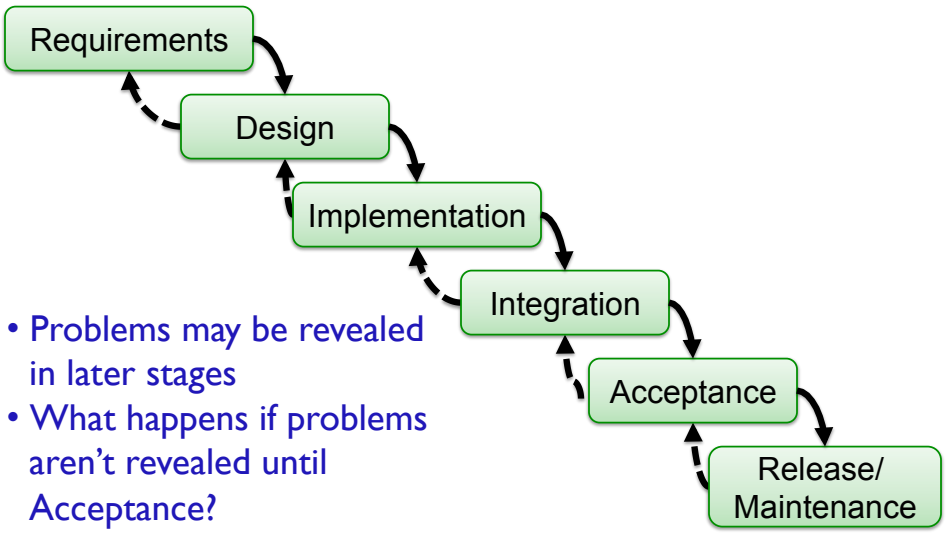
# Traditional Software Engineering Process: Waterfall Model




**Validate** at each step
Goal: A stage is 100% complete before moving to next step

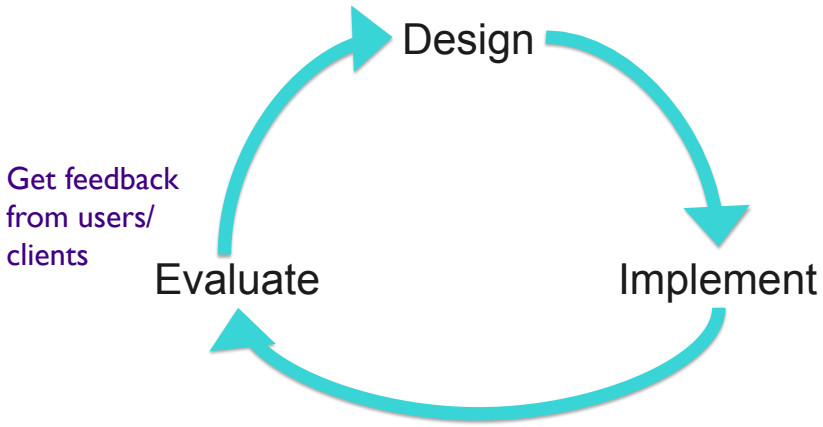


# Feedback in Waterfall Model




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

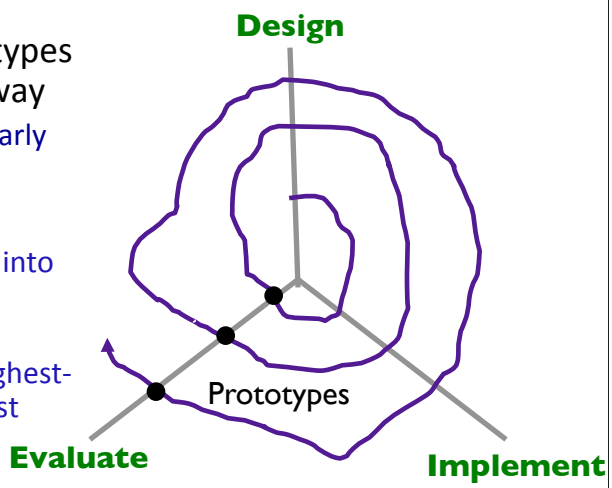
# Iterative Design






# 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

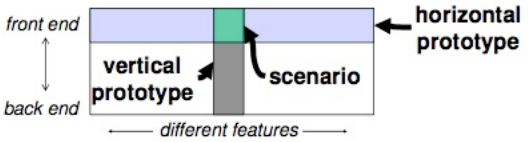



[Boehm 86]

Radial dimension: cost

# Prototypes

- Purpose/Dimensions
  - Functionality
  - Interaction
  - Implementation
- Fidelity:
  - Low: omits details
  - High: closer to finished project
  - Multi-dimensional
    - Breadth: % of features covered
      - Only enough features for certain tasks
    - Depth: degree of functionality
      - Limited choices, canned responses, no error handling

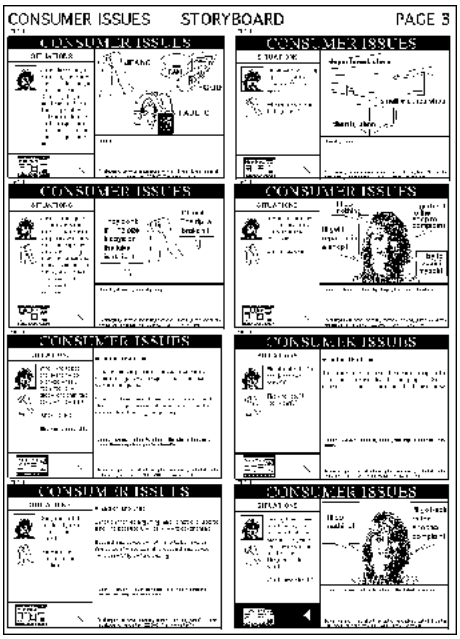



From Nielsen, *Usability Engineering*



# Low Fidelity Prototypes

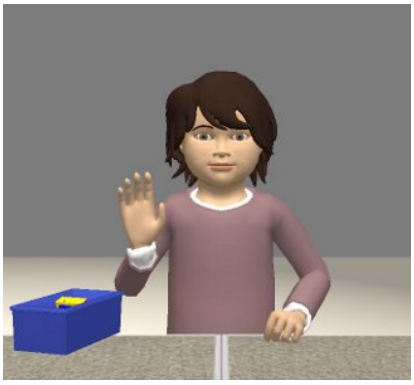
- Media: Paper
- Examples: storyboard, sketches, flipbook, flow diagram



## High Fidelity Prototypes

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

Virtual Peer for Autistic Children



http://www.articulab.justinecassell.com/projects/samautism/index.html

## How to Implement an Effective Solution

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

## Spiral Model Steps

- Design a {method, class, package}
- Implement the {method, class, package}
- Test the {method, class, package}
- Fix the {method, class, package}
- Deploy the {method, class, package}
- Get feedback
  - Probably will require modifications to design
  - May even need to rollback a previous version
- Repeat



## Agile Development Framework: Scrum

- **The Scrum framework in 30 seconds**
  - Product owner creates prioritized wish list, *a product backlog*
  - Team works in a *sprint*, usually 2-4 weeks
    - During planning, team picks a subset of wish list, *a sprint backlog*, and decides how to implement those pieces
    - Daily Scrum: team meets daily to assess its progress
      - ScrumMaster keeps the team focused on its goal
    - At end of sprint, work should be potentially shippable:
      - ready to hand to a customer, put on a store shelf, or show to a stakeholder
    - The sprint ends with a sprint review and retrospective
  - Repeat sprint

https://www.scrumalliance.org/why-scrum

## Agile Development Framework: Tools

- Trello
- Kanban



## Looking Ahead

- Assign 7 due Wednesday