

SWE 261P: Software Testing and Debugging

Professor James A. Jones
Department of Informatics
<http://www.ics.uci.edu/~jajones>

Administrative Stuff

- Lectures: Thursdays

4:00PM–6:50PM

- Course Exam: Wednesday, Mar. 11

Office Hours: Immediately following class, or otherwise by appointment

Schedule

- Schedule is posted on the website
- Will likely change throughout the quarter.

Schedule (may change throughout the quarter)

Date	Content	Assignment Due
Week 1, Thurs. Jan. 7	Course Introduction, Testing Fundamentals	Assigned: <ul style="list-style-type: none">• Participation: Preliminary Course Survey. Due Friday (Jan. 9) night at midnight• Before class next week (Jan. 14):<ul style="list-style-type: none">◦ Read/skim these two articles:<ul style="list-style-type: none">▪ https://www.satisfice.com/exploratory-testing▪ https://www.vogella.com/tutorials/JUnit/article.html◦ Install Eclipse, IntelliJ, or VS Code IDE (your choice of which, and you may wish to try each one)
Week 2, Thurs. Jan. 14	Testing Principles, JUnit, Exploratory Testing, Acceptance	Assigned: <ul style="list-style-type: none">• Participation: Sign up for your project groups and software systems. Groups can be of size 1, 2 or 3 students. Due Thurs., Jan 23 by 4pm
Week 3, Thurs. Jan. 21	Functional Testing, Partitioning, Equivalence Partitioning, Boundary Testing, Combinatorial Testing	
Week 4, Thurs. Jan. 28	Parameterized Testing in JUnit, Functional Models, Finite State Machines, State Analysis, Control Flow Analysis, Structural Testing, Instrumentation, Code Coverage	
Week 5, Thurs. Feb. 4	Test Driven Development (TDD), Behavior Driven Development (BDD), Continuous Integration, GUI Testing	
Week 6, Thurs. Feb. 11	Mocking, Testable Design	
Week 7, Thurs. Feb. 18	Code Review, Static Analysis Tools, Mutation Testing, Debugging	
Week 8, Thurs. Feb. 25	Project Report Interviews	
Week 9, Thurs. Mar. 4		
Week 10, Thurs. Mar. 11		

Administrative Stuff

- Website: Canvas
- Class Files (Slides, Handouts, Assignments): Canvas
- Submission: Canvas
- Email Mailing List: Canvas

Materials

- No text book required
- However, a good supplemental (and optional) resource is “Software Testing and Analysis”, Pezzè and Young
- Laptop
- Paper and pen/pencil (or tablet) for in-class exercises
- Slides will be posted before or after class
- I'll also link a number of free online resources

Acknowledgements

Course materials being used from

- Rosenblum's course on Validation and Verification
- Pezzè and Young's testing and analysis slides
- Amman and Offutt's software testing materials
- Harrold's program analysis course
- Richardson's software testing and analysis
- Mauricio Aniche and Arie van Deursen's automated software testing course

Attendance

- You are responsible for material that is distributed during class.
- If you miss class, please arrange beforehand with a friend to get notes/handouts for you.
- If you know you are going to miss something important, contact the instructor (me) and/or the TA beforehand. It's much easier to accommodate planned absences
- You are responsible for all material taught in this course.

Announcements

- All announcements will be sent out through Canvas

Me

James A. Jones — “Jim”

- Professor of Information and Computer Science
- Department of Informatics
- Office: ISEB 2450
- Email: jajones@uci.edu

Teaching Assistant

- Md Rakib Hossain (mdrh@uci.edu)
 - Office Hour: TBA

Topics We Will Cover

(Tentative)

- Testing Fundamentals
- Combinatorial Testing
- Structural Testing
- Regression Testing
- Testing Driven Development
- Testing and Profiling
- Testing Processes
- Debugging
- Continuous Integration
- BDD
- Static Analysis
- Dynamic Analysis

Course Structure

- In-class:
 - Lectures
 - Group discussions
 - Practicums (i.e., in-class practical exercises)
 - Course exam
- Outside class:
 - Homework assignments will comprise the overall Course Project that will require using the techniques taught in class

Grading

- Class participation (10%)
- Project (40%)
- Project Presentation (20%)
- Exam (30%)

Project

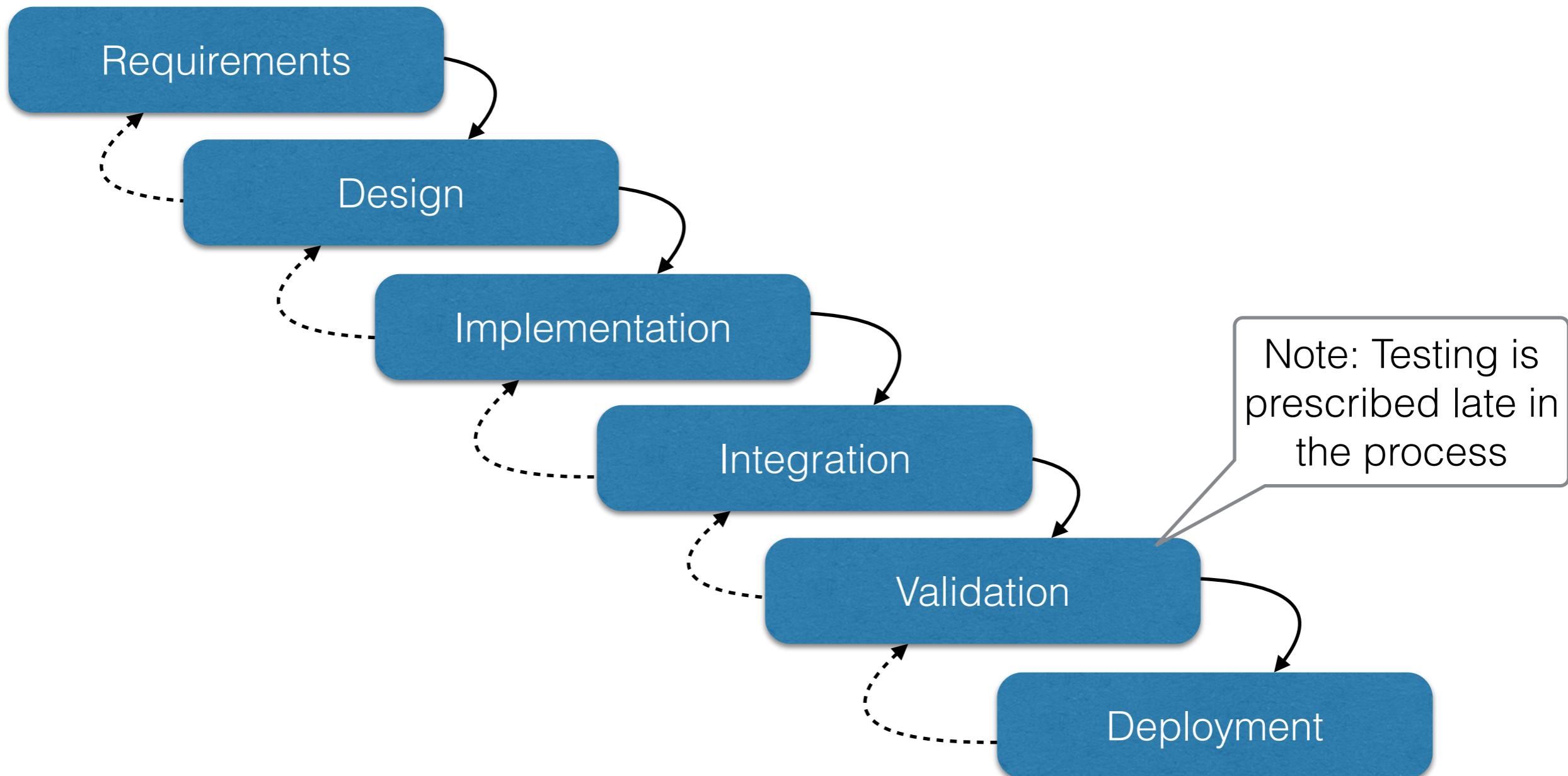
- Choose a **Java** software system
 - At least 15K LOC
 - At least 100 classes
 - Not developed by you: was developed by others and is used by thousands of people. Publicly available, open source.
 - Must be unique in class: each group has their own program — no two groups may share the same system.
 - Must be recognizable, well-used, popular project
 - Look for projects that already have test cases
 - Look for projects that have some sort of user interface (GUI, command line, etc.) and is not only a library.

Exam

- Last day of class (week 10)
- The exam will cover the topics that we cover throughout the whole course (i.e., comprehensive)
- The slides will be the best study guide for the exam
- In-class exercises will also be important, so attendance is important

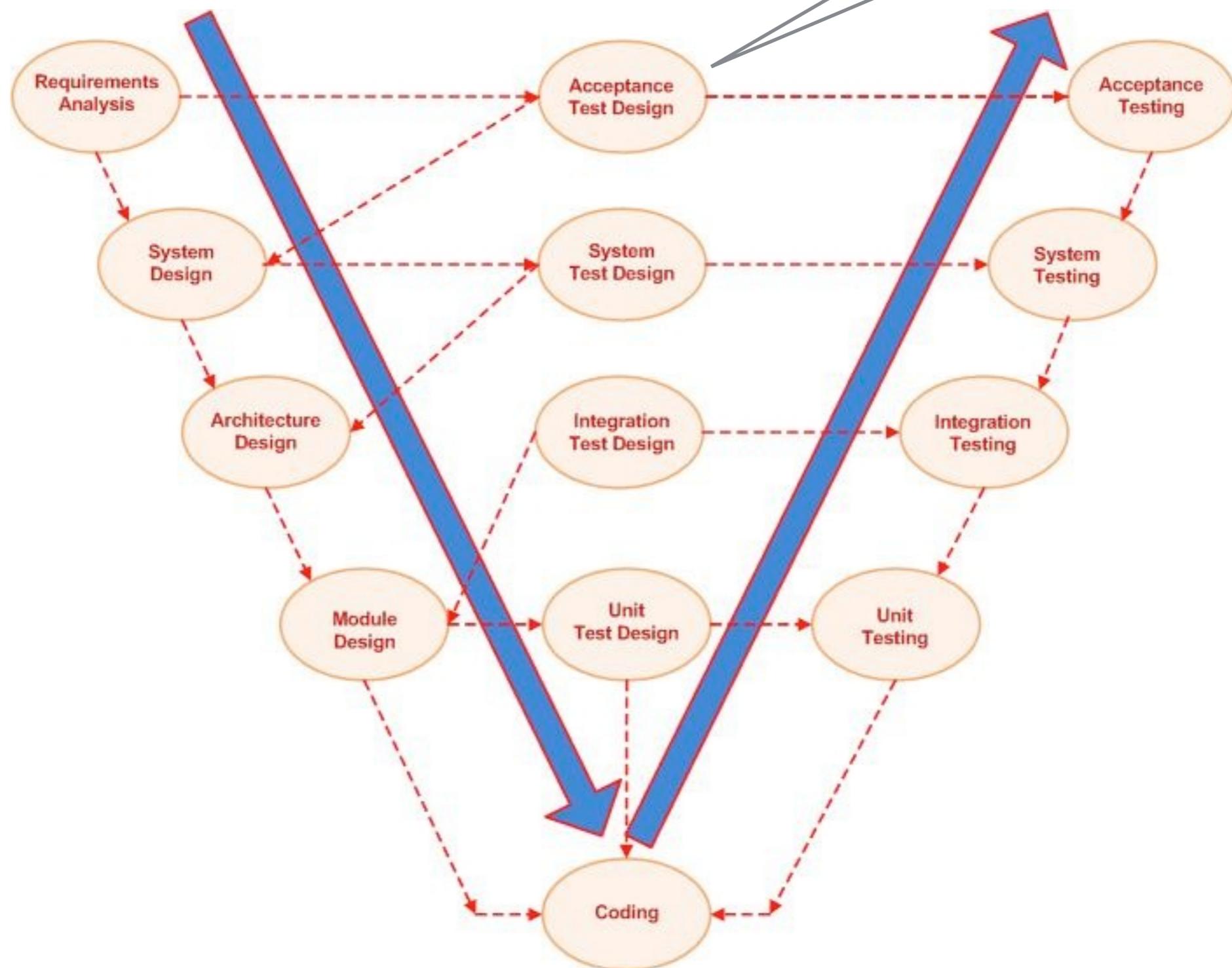
Testing Fundamentals

Waterfall Model (Royce, 1970)

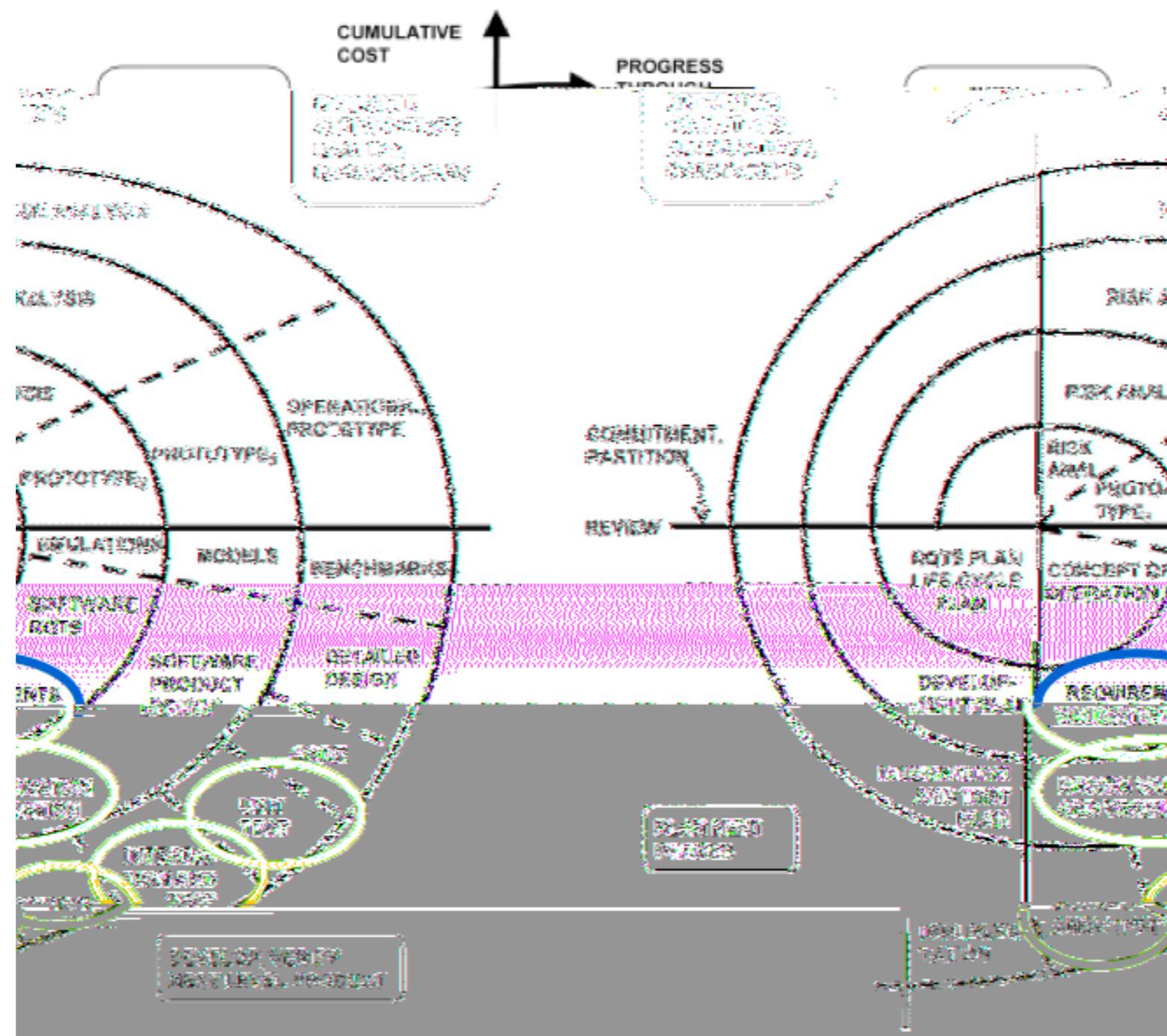


V-model

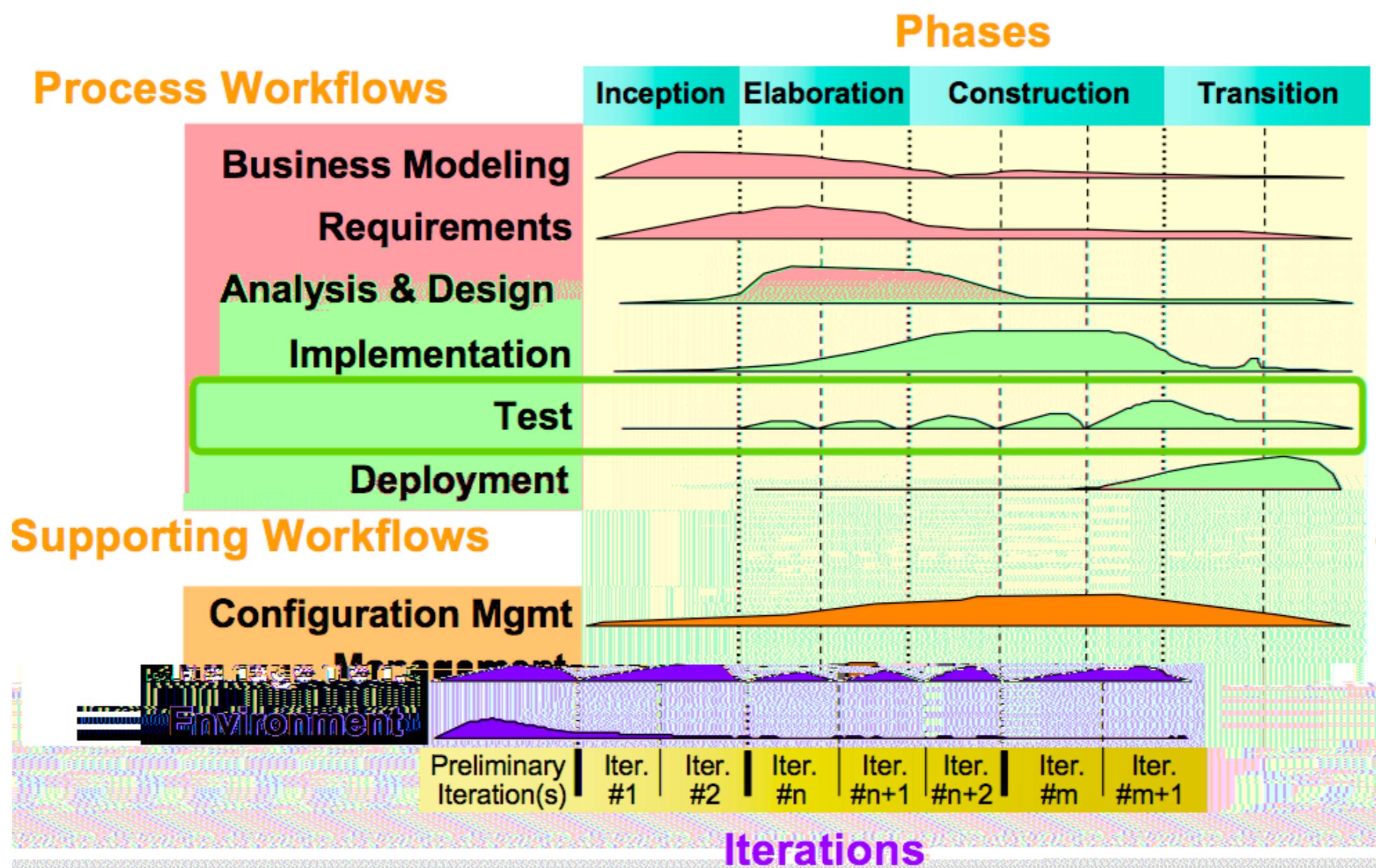
Note: Test design is incorporated earlier in the process



Spiral Model (Boehm, 1988)



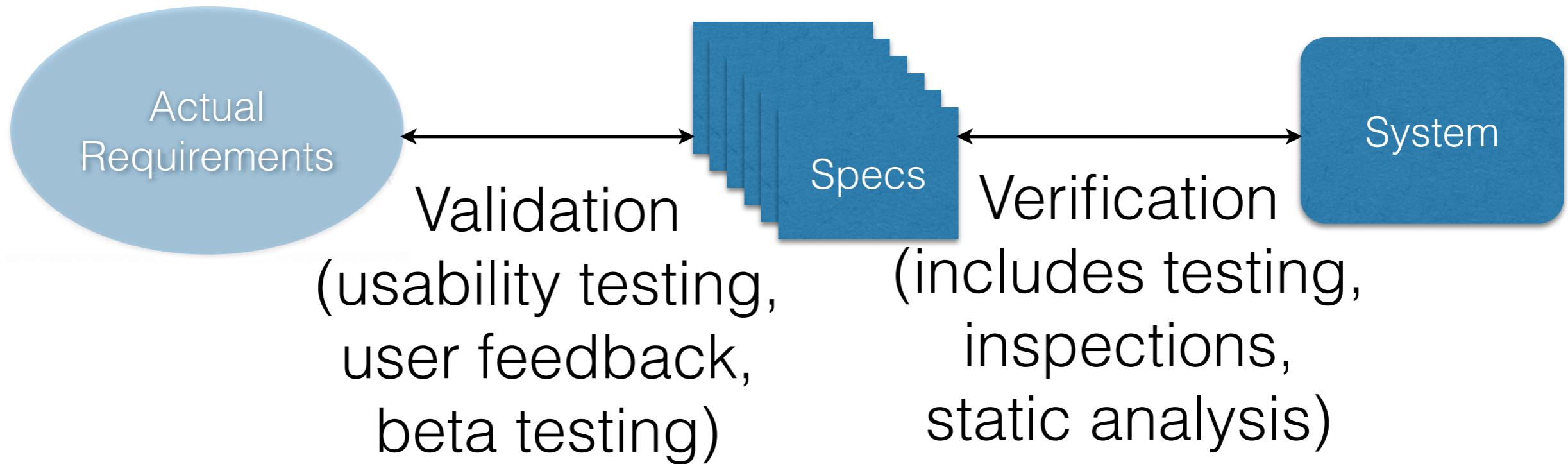
Unified Software Development Process (Jacobson, 1999)



Lesson: Quality Assurance activities actually start very early or at the very beginning of a project (in one form or another), and as the field of software engineering matures, there is greater recognition that QA is more important earlier and throughout the development process.

Fundamentals

- Validation and Verification (“V&V”)
- Verification: “Are we building the machine right?”
- Validation: “Are we building the right machine?”

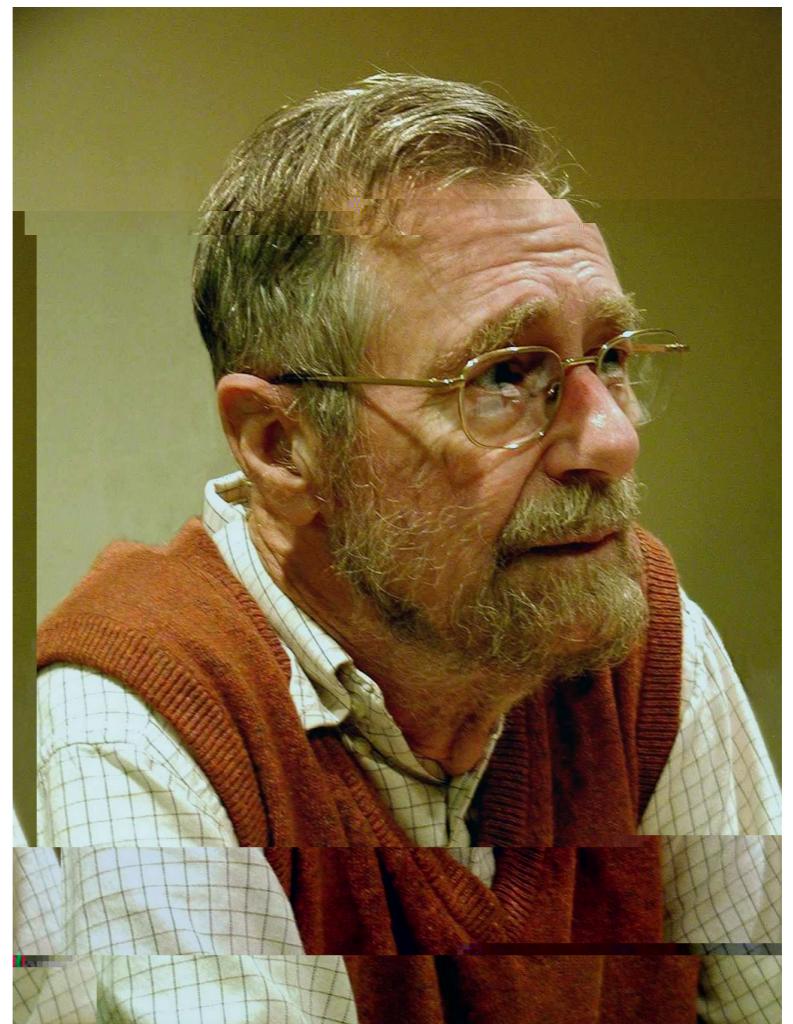


Terminology

- **Test Case**: A defined set of input values (and possibly a defined starting state) that cause a program to take some defined action, with an expected output
- **Test Suite**: A collection of test cases
- **Test Oracle**: A mechanism for determining whether the actual behavior of a test case execution matches the expected behavior

A Famous (or Infamous) Quote

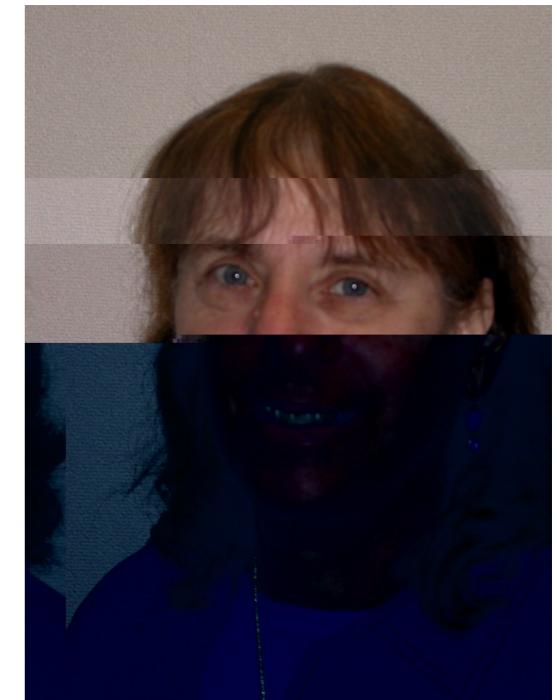
- An oft-repeated disparagement of testing that ignored the many problems of his favored alternative (formal proofs of correctness)
- But, essence of this quote is true



- Essence of this quote is true
 - Testing allows only a sampling of an enormously large program input space
 - The difficulty for the tester is to construct effective samples
 - Reveal faults
 - Reveal performance bottlenecks
 - Demonstrate reliability, usability, ...

Testing Oracles and Non-testable Programs

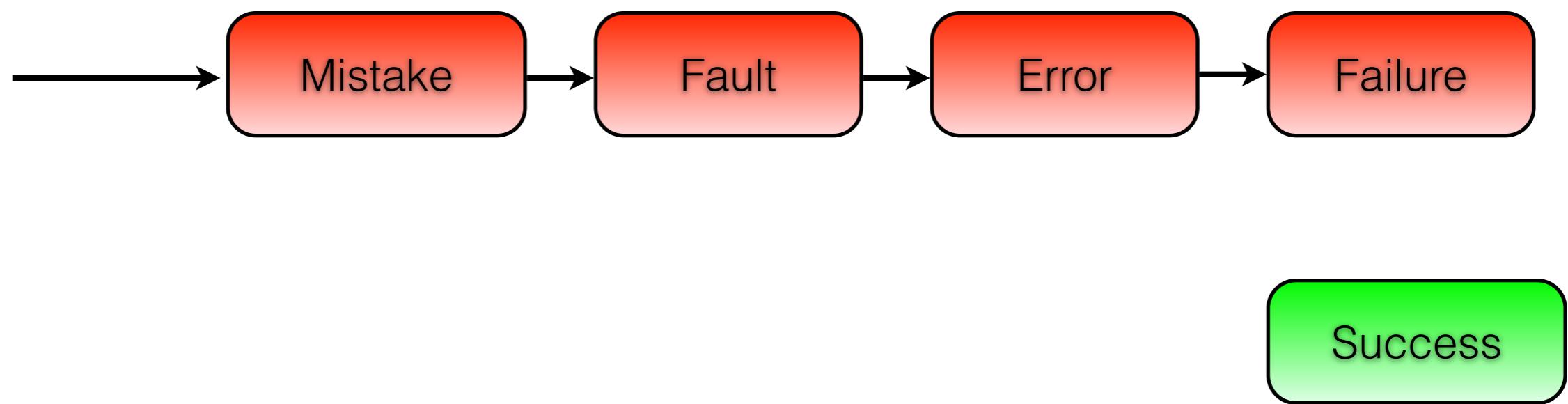
- Weyuker observed that many programs are “non-testable,” in the sense that it is nearly impossible to construct an effective oracle for them
- Problems so complex that software is needed to solve them. How do we check their answers?



Terminology

- **Mistake**: A human action that produces an incorrect result
- **Fault**: (same as “bug”) An anomaly in the source code of a program that may lead to an error
- **Error**: The runtime effect of executing a fault, which causes a deviation from correct behavior and may result in a failure
- **Failure**: The manifestation of an error

IEEE Standard Glossary of Software Engineering
Terminology. IEEE Std. 610.12-1990



Fault vs. Error vs. Failure Example

```
void rotateLeft(int* array, int  
size) {  
    int i;  
    for (i=0; i<size; i++) {  
        array[i] = array[i+1];  
    }  
}
```

Specification: C program taking an array of integers and rotating the values one position to the left, with wraparound

Test case 1

array input



size

0

array output



No error, no failure

Fault vs. Error vs. Failure Example

```
void rotateLeft(int* array, int  
size) {  
    int i;  
    for (i=0; i<size; i++) {  
        array[i] = array[i+1];  
    }  
}
```

Specification: C program taking an array of integers and rotating the values one position to the left, with wraparound

Test case 2

array input	size	array output
 0	2	 0

Error, but no Failure

- The loop accesses memory outside of the array
- But the output array is coincidentally correct

Fault vs. Error vs. Failure Example

```
void rotateLeft(int* array, int  
size) {  
    int i;  
    for (i=0; i<size; i++) {  
        array[i] = array[i+1];  
    }  
}
```

Specification: C program taking an array of integers and rotating the values one position to the left, with wraparound

Test case 3

array input	size	array output
0 1 66	2	1 66 66

Failure!

- The output array is incorrect

Fault vs. Error vs. Failure

```
void rotateLeft(int* array, int  
size) {  
    int i;  
    for (i=0; i<size; i++) {  
        array[i] = array[i+1];  
    }  
}
```

But, what exactly is “the fault”?

- Depends greatly on one’s point of view
 - There are potentially many faults here
 - The loop indexes array outside its bounds
 - The loop never moves array[0] to another position
 - The loop never saves array[0] for later wraparound
 - There are also many possible fixes
 - The fix actually determines what “the fault” was!

Terminology

- ***Test Effectiveness***: The extent to which testing reveals faults or achieves other objectives
- ***Test Plan***: A document describing the scope, approach, resources, and schedule of intended testing activity
- ***Testing versus Debugging***: Testing reveals faults, while debugging is used to remove faults

Assignments

By Friday night (Jan. 9):

Complete the survey in Canvas by Friday night (Jan. 9). You can find it in the Quizzes section. It is not “graded” — that is, there are no right or wrong answers. It is anonymous — I can see who took it and who did not, but I cannot see who said what. It is simply for me to get a sense for at what level I should teach this course. I’ll count participation in the survey toward your participation in the course.

I will present the aggregated results of the survey in class next week.

Before class next week (Jan. 14):

- Read/skim these two articles (they are linked on Canvas):
 - <https://www.satisfice.com/exploratory-testing>
 - <https://www.vogella.com/tutorials/JUnit/article.html>
- Install Eclipse, IntelliJ, or VS Code IDE (any one is fine)