Objectives

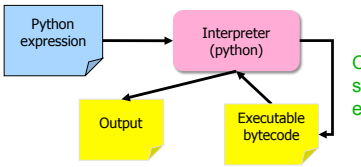
- Wrap-up Language Comparison
- Software Development

Review

- Why do we need `Comparators`?
- What is the benefit of using jar files?
- How do we create a jar file? Extract the contents of a jar file?
- What are the 3 preconnected streams?
  - How do we access them in Java?

Python Interpreter

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



Java Compiler

- 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: .class → bytecode
- Compiler optimization techniques
  - Generate *efficient* bytecode/machine code
  - Examples: get rid of unused local variables, transform loops
  - 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

What can we do in Python that we can't do in Java?

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

Language Comparison

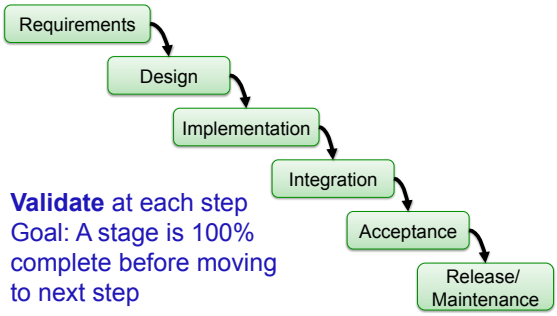
Java
- Object-oriented
- Statically typed
- Compiled

Python
- Object-oriented
- Dynamically typed
- Interpreted

Pros and cons of using each?

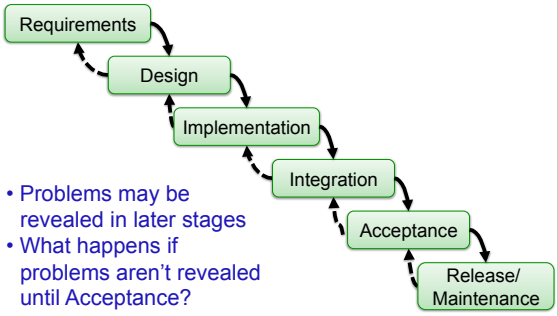
SOFTWARE LIFE CYCLE

Traditional Software Engineering Process: Waterfall Model



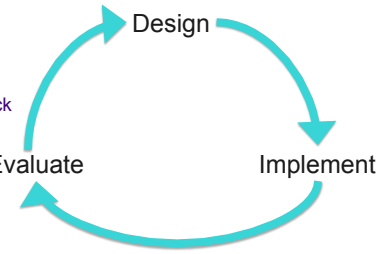
Validate at each step
Goal: A stage is 100% complete before moving to next step

Feedback in Waterfall Model



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

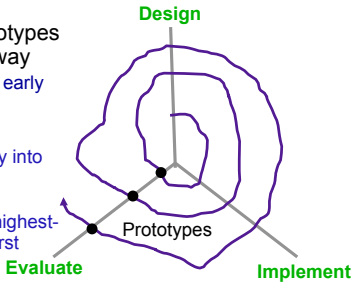
Iterative Design



Get feedback from users/ clients

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

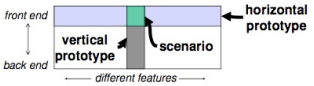


Radial dimension: cost

[Boehm 86]

Prototypes

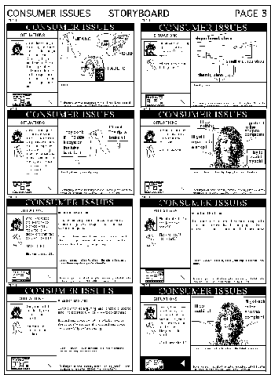
- Purpose/Dimensions
  - Functionality
  - Interaction
  - Implementation
- Fidelity:
  - Low: omits details
  - High: closer to finished project
  - Multi-dimensional
    - Breadth: % of features covered
      - Only enough features for certain tasks
    - Depth: degree of functionality
      - Limited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



From Nielsen, *Usability Engineering*

Low Fidelity

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



High Fidelity

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

Virtual Peer for Autistic Children

http://articulab.northwestern.edu/projects/samautism/

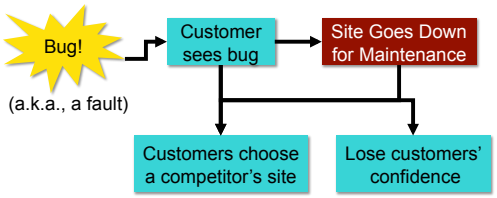
Spiral Model Steps

- 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
- Repeat

SOFTWARE TESTING PROCESS

Why Test Programs?

- Consider an online bookstore



Microsoft Windows Vista Testing

- Beyond their internal testing …
  - 5 million people beta tested
  - 60+ years of performance testing
  - 1 Billion+ Office 2007 sessions
- Still, users found correctness, stability, robustness, and security bugs

Type 1 Bugs: Compile-Time

- Syntax errors
  - Missing semicolon, parentheses
- Compiler notifies of error
- Cheap, easy to fix

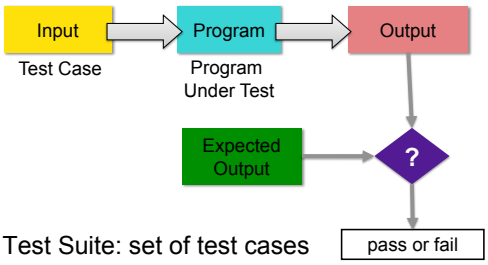
Type 2 Bugs: Run-Time

- Usually logic errors
- Expensive to locate, fix

Aside: Objections to "Bug" Terminology

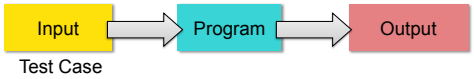
- "Bug"
  - Sounds like it's just an annoyance
    - Can simply swat away
  - Minimizes potential problems
  - Hides programmer's responsibility
- Alternative terms
  - **Defect**
  - **Fault**

Software Testing Process



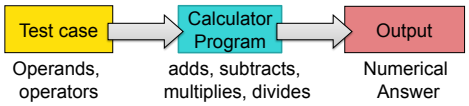
- Test Suite: set of test cases

Software Testing Process



- Tester plays devil's advocate
  - ***Hopes*** to reveal problems in the program using "good" test cases
  - Better tester finds than a customer!
- How is **testing** different from **debugging**?

How Would You Test a Calculator Program?



- What test cases/input?

Example Test Cases for Calculator Program

- Basic Functionality
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Order of operations
- Invalid Input
  - Letters, not-operation characters (&,$, …)
- "Tricky" Cases
  - Divide by 0
  - Negative Numbers
  - Long sequences of operands, operators
  - VERY large, VERY small numbers