Testing

Review session Monday, 1-2:00PM here
- bring any questions

Questions
- Why is software like physics?
- Why is it not like physics?

Questions
- Why, according to McConnell is software engineering so complex?
Questions

- Why is software engineering like gardening?

Questions

- What are advantages of developing software in small steps?

Questions

- What are the 12 practices and write a sentence or two about each

There are 3 kinds of people

- those who can count
- and those who cannot
What are the 13 practices?

- Whole Team
- Metaphor
- Planning Game
- Simple design
- Small releases
- Customer tests
- Pair programming
- Test-driven development
- Design improvement
- Collective code ownership
- Continuous integration
- Sustainable pace
- Coding standards

First Test, then code

- Write a test case using JUnit
- Case fails
  - Add just enough to make it pass
- Add new cases
  - Add just enough ...

How long does this go on?

- Dijkstra – "Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence."

Dijkstra – "Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence."
Problems with exhaustive testing

- To repeat, not possible
- Simple cases are easy
- Exhaustive tests are complex and time consuming
- Writing complicated tests is a buggy exercise

Consider `maxOfThreeInts`

```java
public int maxOfThreeInts(int n1, int n2, int n3) {
    return n2;
}
```

Exhaustive testing

```java
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        for (int k = 0; k < 10; k++) {
            result = maxOfThreeInts(i, j, k);
            check to make sure really is
        }
    }
}
```

Representative testing

- Demonstrate the primary, expected behavior of segment of code
  - `maxOfThreeInts(9,2,7) == 9`
- representative test – pick some trip set of numbers and check this
  - might use `maxOfThreeInts(9,2,9)`
  - `maxOfThreeInts(9,9,9)`
Test Suites

- Must do this from the beginning
  - really do need to test first, then code
- test suite, set of tests, is another product
- grows in breadth as you continue
- gives you freedom to refactor code
- must be automated - junit

Testing-first benefits

- Forces simplicity
  - you try to find representative cases
  - find the smallest such set
  - once done, now you concentrate on code
- Goal is now, pass the test
  - use least amount of code
  - may be ugly
  - we can change later (because of all the tests to guide us)

Testing first clarifies the task at hand

- Writing to a spec is hard and open-ended
- Writing to a test is self contained
- If you don't understand what to test, you don't understand what to code

Testing-first frees you from on-the-fly editing

- Common scenario:
  - you write a big chunk of code
  - aha moment
  - do I recode or just keep it?
- With large test suite
  - aha moment can be delayed until tests pass
  - then refactor and pass the tests
Test Suites as documentation

- Encapsulates developer's intent when writing code
- Electronic diary of development of system
- Can be reviewed by outsider to see status
- Serves as guide to future changes desired

Using testing with CVS

- You check out code and run junit – code will run
- You select task
- You write tests
- You write code to meet tests
- You get green bar
- You commit changes to project

Test Suite Maintenance

- As software grows, test suite needs maintenance
  - You change code, return values, etc.
  - Old tests fail
  - You may need to change the tests, return values, etc.
  - Treat suite like other software, refactor!

Test Suite Maintenance

- You may recognize bugs in your system
  - Introduce a test as documentation of the bug
  - This helps to alert team to error
  - Prevents missing problem in the future.
What about design?

- We don't design first, then code
- We design and build in tandem