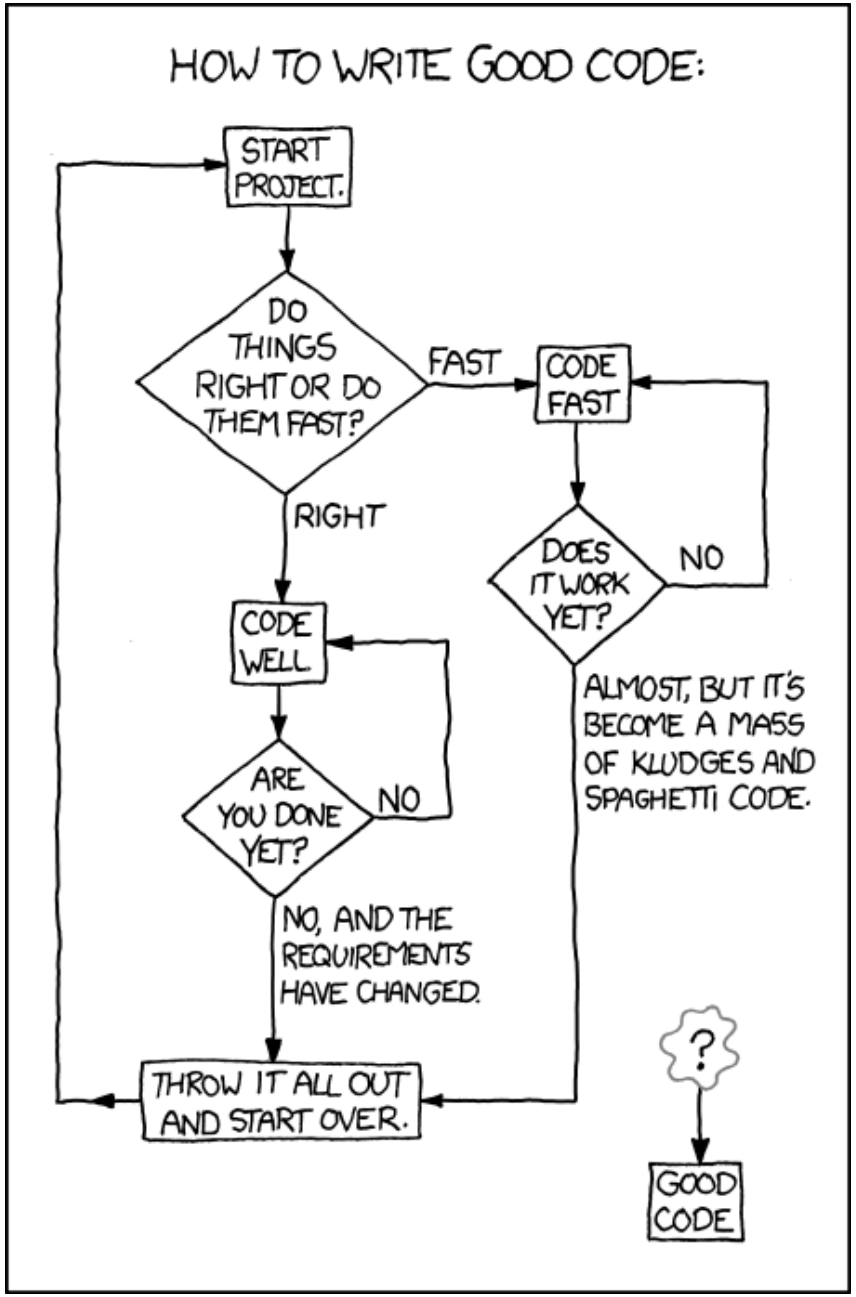
Testing I

Andrew Hu



Administrivia

- Attendance: http://tinyurl.com/uwigem/18sp/attendance/
- Spec meetings
  - Adjust spec based on feedback
  - Go implement!
    - Have until next week, since the deadline was extended

Aside: When will we work on a real project?

- Start low-risk
- Experience we gain from practice/“toy projects”
  - Code collaboration
  - Teamwork
  - Software

We will start working as a team after Pacman and the EE lab has been completed ☺

Agenda

- How to get good code
- Unit Testing
- Project Meeting

How to avoid bugs?

Use a language with types (like Java)

Test your code

Ask other people to look at your code

These 3 things prevent >90% of all bugs

How to write a simple test?

Call a method with one set of input

```
result = o.mapOnto(input);
if(result != expected){
  print("mapOnto() failed with "+result);
}
```

What happens when a test fails?

- You go look inside the code, but your method calls several other methods
- What is causing the problem?
  - Your code?
  - The methods you are calling?

```
void mapOnto(int[] input){
  for(int i : input){
    input += i;
  }
  this.updateFields();
}
```

Unit test

- Fix this by making sure that every single method has its own test
- Unit tests build on top of each other
- When every method called inside another method is tested, we know where the problem is

Unit Tests

```
void mapOnto(int[] input){
  for(int i : input){
    input += i;
  }
  this.updateFields();
}
```

```
void tests(){
  testUpdateFields();
  //^^This passes
  testMapOnto();
  //^^This fails
  /*Which part of mapOnto
  is causing a problem?*/
}
```

How to test an entire system?

- Testing a video game?
- Testing a utility, like Word?

Write internal tests that chain together multiple methods?

Have a person try to use the software?

Integration test

- Multiple method calls one after another
- Then verify that the state is as expected

System Testing

- Have a human use the system
- Write down if anything weird happens
- Often called “play testing” in video game development

Remember Pre & Postconditions?

- Author assumes the precondition is true when the method is called
- User assumes the postcondition is true after the method is called
- Some things often show up as both pre and postconditions

Invariant

- If something is true before and after every method call it is invariant
- E.g.
  - The array field is never null
  - The sum field always represents the sum of the elements in the array
- Since these things relate to the representation of your data, it is called the representation invariants

Check Rep

- The representation invariant should *never* be false
  - We can just call a method to check that the invariant is true, at the end of every method
- This is the “check rep”

Toy Example

"Sum Set" Example

- We want to do two operations: add to the set, and get the sum
- Example implementation:

```
class SumSet {
  private List<Integer> nums;
  public void add(int i) {
    nums.add(i);
  }
  public int sum();
}
```

SumSet Sum Method

- How to efficiently return the sum? (the simple answer please ☺)

```
class SumSet {
  private List<Integer> nums;

  public void add(int i) {
    nums.add(i);
  }
  public int sum() {
    // What goes here?
  }
}
```

SumSet Sum Method

```
class SumSet {
  private List<Integer> nums;

  public void add(int i) {
    nums.add(i);
  }
  public int sum() {
    int result = 0;
    for (int i : nums) {
      result += i;
    }
    return result;
  }
}
```

SumSet Sum Method

```
SumSet s = new Sumset();

// Add some elements
for(int i=0;i<10;i++){
  s.add(i);
}

// Print the sum
print(s.sum());

// Print the same sum
print(s.sum());
```

```
class SumSet {
  private List<Integer> nums;

  public void add(int i) {
    nums.add(i);
  }
  public int sum() {
    int result = 0;
    for (int i : nums) {
      result += i;
    }
    return result;
  }
}
```

“Memo-ization”

- Use “memo-ization” to return the sum this efficiently
  - After traversing the list, save the sum into a variable (`currSum`)
  - As long as nothing is added, our sum is still added
  - When we add another int, recalculate the sum

SumSet Sum Method

```
SumSet s = new SumSet();

// Add some elements
for(int i=0;i<10;i++){
  s.add(i);
}

// Print the sum: O(n)
print(s.sum());

// Print the sum: O(1)
print(s.sum());
```

```
class SumSet {
  private List<Integer> nums;
  private int currSum;

  public void add(int i) {
    nums.add(i);
    calcSum();
  }
  public void calcSum() {
    currSum = 0;
    for (int i : nums) {
      currSum += i;
    }
  }
  public int sum() {
    return currSum;
  }
}
```

SumSet Representation Invariants

- Two things are always true about the SumSet
- The list (`nums`) is never null
- `currSum` is always equal to the sum of all the elements

SumSet Check Rep

- Have a method called `checkRep()` that checks that these two things are true after every method call

Coverage testing

- How do we know that we've actually tested all of our code?
- Not just all the methods, but all the possible different inputs
- Coverage testing tells you how many times each line was run after a series of tests