TESTING

Prof. Chris Jermaine
cmj4@cs.rice.edu

What is “Testing?”

- Before we define it, let’s motivate it... why test?

What is “Testing?”

- Before we define it, let’s motivate it... why test?
  - — To be sure we have code without “bugs”
  - — That is, to be sure that in a production environment, our software will never encounter a use case for which it produces incorrect behavior

What is “Testing?”

- Before we define it, let’s motivate it... why test?
    - — To be sure we have code without “bugs”
    - — That is, to be sure that in a production environment, our software will never encounter a use case for which it produces incorrect behavior
    - — Note: if we could formally verify code, no testing! But doesn’t work in practice
- Is testing important?

What is “Testing?”

- Before we define it, let’s motivate it... why test?
  - To be sure we have code without “bugs”
  - That is, to be sure that in a production environment, our software will never encounter a use case for which it produces incorrect behavior
  - Note: if we could formally verify code, no testing! But doesn’t work in practice
- Is testing important?
  - Always, but sometimes more than others
  - Software to manipulate control surfaces on B787?
    Testing arguably more important than any other part of engineering process!
  - Next Windows release?
    Testing might take a back seat to core software development
  - Importance of correctness dictates how much effort is put into testing!

What is “Testing?”

- To inexperienced/poor engineers, or when getting code out is key
  - — “Testing is showing that my code works on a few key inputs”

What is “Testing?”

- To inexperienced/poor engineers, or when getting code out is key
  - “Testing is showing that my code works on a few key inputs”
- I’d argue that in a perfect world
  - “Testing is the process of developing a set of software use cases, such that if a code works on those use cases, the software will not fail in the real world”

What is “Testing?”

- To inexperienced/poor engineers, or when getting code out is key
  - “Testing is showing that my code works on a few key inputs”
- I’d argue that in a perfect world
  - “Testing is the process of developing a set of software use cases, such that if a code works on those use cases, the software will not fail in the real world”
- Unfortunately, this is almost never possible
- Best way to ensure quality testing, make it *adversarial*
  - Test engineers should *want* to make production developers look bad
  - They should strive to find flaws in code

Testing in the Real World

- No industry standard
  - — Every shop tends to have its own way of doing things
- But are two major approaches
  - — The “waterfall model”

    Requirements then Design then Implementation then Verification then Maintain
  - — “Agile development”

    Write tests first, after design, then code... keep writing more tests as code base grows...

    constantly run tests over code (nightly, automatically)

At What Granularity Should One Test?

- At many levels of granularity!

At What Granularity Should One Test?

- At many levels of granularity!
- “Unit testing”
    - — Test each software component to make sure it implements its interface precisely

At What Granularity Should One Test?

- At many levels of granularity!
- “Unit testing”
  - — Test each software component to make sure it implements its interface precisely
- “Integration testing”
  - — As you build software components that fit together
  - — Test them to make sure they work correctly together

At What Granularity Should One Test?

- At many levels of granularity!
- “Unit testing”
    - — Test each software component to make sure it implements its interface precisely
- “Integration testing”
    - — As you build software components that fit together
    - — Test them to make sure they work correctly together
- “System testing”
    - — When you have whole system, make sure its outputs match its inputs

How Should the Tests Be Developed?

- Two main approaches
- “White Box Testing”
  - — Aware of internals of component being tested
  - — Try to take all paths in the code
  - — Pay special attention to corner cases in control flow statements
  - — Try to “mess up” internal data structures
- “Black Box Testing”
  - — Only have specification, don’t have understanding of inside
  - — Tests are written to try to find cases where code does not meet spec
  - — Pay special attention to corner cases in spec

    Correct bahavior right after iniatialization, after all data have been removed, when data are added in strange order, etc.

Black Box Testing Example

```
public int factorial (int n) {...}
```

- What tests make sense here?

Black Box Testing Example

```
public int factorial (int n) {...}
```

- What tests make sense here?
  - — zero, one, two
  - — several tests of arbitraty larger numbers (14, 23, 31)
  - — bad input: negative input number
  - — bad input: very large input number (1000)

Black Box Testing Example

```
public int isPrime (int n) {...}
```

- What tests make sense here?

Black Box Testing Example

```
public int isPrime (int n) {...}
```

- What tests make sense here?
  - — One (corner case)
  - — Bad input: zero, negative number
  - — Exhaustive list of small primes: 2, 3, 5, 7, 11, 13, up to 97
  - — Set of randomly selected larger primes: 859433, 1257787, 1398269, 2976221...
  - — Exhastive list of small non-primes: 4, 6, 8, 9, 10, up to 100
  - — Set of randomly selected larger non-primes
  - — Set of randomly selected numbers with two prime factors
  - — Set of randomly selected squares of primes

White Box Testing Example

```
public boolean palindrome(String s) {
  int low = 0, high = s.length() - 1;
  while (high > low) {
    if (s.charAt(low) != s.charAt(high))
      return false;
    low++; high--;
  }
  return true;
}
```

- What tests make sense here?

White Box Testing Example

```
public boolean palindrome(String s) {
  int low = 0, high = s.length() - 1;
  while (high > low) {
    if (s.charAt(low) != s.charAt(high))
      return false;
    low++; high--;
  }
  return true;
}
```

- What tests make sense here?
  - — null s, or while loop gets skipped (s is empty)
  - — "return false" never executed (two strings the same)
  - — "return false" hit immediately (two strings differ on first char)
  - — "return false" hit in middle of string (two strings differ in middle)
  - — "return true" is executed (two strings the same)
  - — even/odd s length ('cause of low++, high--)

Questions?