CSE 3902: Software Testing and Quality Assurance

Justin Holewinski

The Ohio State University

Software Testing

Types of Testing

- *Unit Tests*: Tests written for specific sections of code, such as individual functions
  - Loading a texture
  - Detecting a key press
  - Checking for collision between hard-coded objects
  - Testing all possible collision types
- *End-to-End Tests*: Tests written against the whole application, usually for testing the application as a whole
  - Launching application and playing through a level
- *Smoke Tests*: Tests written to verify very basic functionality
  - Testing that the application starts
- *Positive Tests*: Tests ensuring code functions as expected
  - Expected to pass
- *Negative Tests*: Tests ensuring errors are handled gracefully
  - Expected to fail *gracefully*

Code Coverage

Code Coverage: Percentage of your code and/or logic that is covered by at least one test

Can be applied to many different units:

- Methods: how many functions are covered by at least one test?
- Statements: how many statements are covered by at least one test?
- Branches: have both sides of conditionals been tested?
- Paths: have all control-flow paths through a region of code been exercised?
- Too many more to list

Code Coverage

```
public void DoSomething(int a, int b)
{
    if (a < 0)
    {
        ActionOne();
    }
    else
    {
        ActionTwo();
    }

    if (b > 0)
    {
        ActionThree();
    }
    else
    {
        ActionFour();
    }
}
```

Test Cases:

- `DoSomething(-1, 1)`
- `DoSomething(1, -1)`

What is our *branch* coverage?

What is our *path* coverage?

Best Practices

- Aim for 100% code coverage, even if you “know” the code works
    - Even simple tests are useful to give yourself confidence a later refactor does not break your product
- Try to break the code
    - Can you glitch the game to cheat?
    - Use negative tests to enforce graceful handling of error conditions
    - Be creative!
- Use requirements document to drive creation of tests, not the code
- Add tests as pre-merge check for Pull Requests
    - Prevent source control from accepting broken changes in main development branch
- Write tests early and often

Test-Driven Development

Traditional model:

- Design
- Write code to implement design
- Write tests to verify code meets design

Test-driven development (TDD) flips this around:

- Design
- Write tests to verify design based on requirements
- Write code with the goal of passing tests

Effectively changes coding goals from "implement design" to "pass tests"

Requires very rigorous test authoring!

Quality Assurance

Quality Assurance (QA) goes beyond software testing

Examples:

- Is the application easy to install?
- Is the application easy to start?
- Is there adequate documentation on the application?
- Does the application run on older hardware?
- Does the application have accessibility features?

Games have some unique categories:

- How is the game balance? Too easy? Too hard?
- Is the game able to maintain a consistent frame rate?

Consider how to incorporate playtesting into your sprint timeline