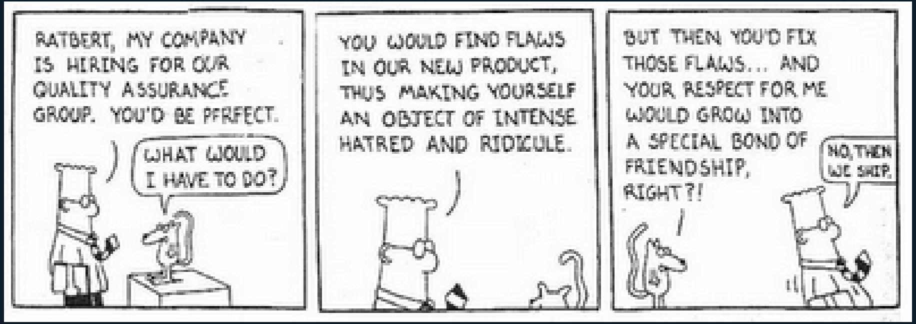
# Unit Testing

A methodical way to test your code to make sure it works and continues to work as it should


RATBERT, MY COMPANY IS HIRING FOR OUR QUALITY ASSURANCE GROUP. YOU'D BE PERFECT.
WHAT WOULD I HAVE TO DO?
YOU WOULD FIND FLAWS IN OUR NEW PRODUCT, THUS MAKING YOURSELF AN OBJECT OF INTENSE HATRED AND RIDICULE.
BUT THEN YOU'D FIX THOSE FLAWS... AND YOUR RESPECT FOR ME WOULD GROW INTO A SPECIAL BOND OF FRIENDSHIP, RIGHT?!
NO, THEN WE SHIP.

## Code Error Revision

Remember that there are four types of error we encounter when writing code:

- Compiler errors
  - syntax
- Linker
  - missing declarations
- Run time crashes
  - asserts and exceptions
- Logical
  - the code works but doesn't do what you hoped it would

# Anatomy of bugs

- When you write code syntax errors are usually the first errors you encounter
- Linker errors come next when multiple source files are linked and external references cannot be resolved
- Crash bugs occur when the code is tested and are usually caused by memory leaks, indexing past the end of arrays, failing to check that API calls return appropriate values etc.
- Finally as you become more proficient in the language, and begin to implement more complex algorithms, logical errors become the major problem

## Testing

- When you write code do you:
  a) Write a huge chunk of code then fix all the compiler, linker and run time errors in one hit?
  b) Write a few lines of code then fix the compiler, linker errors but fix run time errors when you have more code
  c) Try and organize your task list so you can compile, link and run small chunks of code as you go along?

## It doesn’t really matter how you work

- What is important is that you:
  - Test your new code thoroughly
  - Continue to test old code when you add new
  - Test consistently and often
- Unit testing is a method which automates some of the testing process

## What is unit testing?

- The core idea is to write code to test code
- More specifically
  - Break the code down into units which have clearly defined functionality
  - Devise a series of tests which will test whether the unit works correctly
  - Write code to automate those tests
  - Add those tests to the build process

## Simple example

The following code snippet show a function and a simple unit test for it:

```
float add(float a, float b)
{
    return a + b;
}
```

```
float void testAdd()
{
  assert(add(0, 0)==0);
  assert(add(1, 0) == 1);
  assert(add(0, 1) == 1);
  assert(add(1, 1) == 2);
  assert(add(-1, 0) == -1);
  // Add all the tests that are needed to
  // test the functionality of the code
}
```

# How it works

- When our application is set up, we call all our test functions
- When the code is run any test which fails triggers an assert
- The test code will only trigger in debug (why?) so there is no overhead in release mode
- Assert error messages are tricky to interpret but if we are running from the debugger we can go straight to the code
- We can write more sophisticated reporting mechanisms if we require them

# Unit testing has a clear goal

“The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy.”

http://en.wikipedia.org/wiki/Unit_testing

We’ll now list the benefits

## Find problems early

- As soon as the tests are run any problems in new or old code will be found and easily traced.
- In Test Driven Development (TDD) Unit tests are often created before the code is run
  - When you run your code for the first time the unit test is already in place to check you have implemented correctly
  - Test Driven Development (TDD) works very well with Extreme Programming and Scrum paradigms.

## Facilitates change

- Change can be frightening, but unit testing helps to streamline the change process by ensuring that code always functions the same
  - Refactoring of code is an important step in code maintenance but it can lead to subtle bugs
  - Developers often have to change from one platform to another, 32 to 64 bit,  or windows to *linux* for example
  - External libraries may be changed

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

## Simplifies Integration

- Integration testing involves:
  - Breaking code down into small units
  - Devising test for those small units(e.g. our add function)
  - Assembling the small units into larger units (e.g. a vector maths library using the add function)
  - Then producing unit tests which test the larger unit
- This makes it easier to track down our bugs

## Documentation

- Unit tests form a kind of living documentation
  - If a programmer wants to know what a function does they can study the unit tests and gain a basic understanding of how the function should behave
  - If they need to change the function (change from 32 to 64 bit int for example). They don’t need to understand complex documentation to be sure the function still behaves as required

## Design

- Designers can specify unit tests as part of the design process.

- Programmers can implement the unit tests as the first step to implementing the design

# Integration with an automated build process

- Programmer submits a change to the code repository
- A special application which runs in the background, called a *build machine*, detects the change
  - It rebuilds the code base
- As part of the build process the unit tests are automatically run on the new build
  - The build machine alerts the team if a bug is found
  - Any bugs are fixed immediately before moving on

# Designing tests: black or white box?

- Unit tests are great but they are only as good as the tests which are employed.
- Black Box Tests: Written by someone who doesn’t know how the code will work
  - Test the result of all the interfaces
- White Box Tests: Written by the programmer who wrote the code to be tested
  - Test edge cases such as number precision
- Often unit tests get added as development progresses and unforseen bugs are found

## Summary

- Unit tests automate the debugging of code
- They encourage a systematic approach to programming
- They catch bugs early in the process and make bugs easier to fix
- They can be built into the build process which reduces the load on QA
- They take a little bit of time to set up but always save time in the long run
  - You are testing your code anyway right? So why no automate the process!

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

# References

- How it unit testing works:
  - http://gamesfromwithin.com/stepping-through-the-looking-glass-test-driven-game-development-part-1
  - http://en.wikipedia.org/wiki/Unit_testing
- Some problems to be aware of
  - http://www.gamasutra.com/blogs/PiotrTrochim/20150112/233974/Unit_tests_vs_functional_tests__round_101.php
  - http://www.gamasutra.com/view/feature/130682/automated_tests_and_continuous_.php
- Actual instructions on how to integrate into the jenkins build machine
  - https://wiki.jenkins-ci.org/display/JENKINS/Unit+Test#UnitTest-Overview