- Unit testing – establish the presence of bugs
- Debugging – locate and remove the cause of bugs

White vs Black box testing

- White box testing
  - Transparent/glass/clear box testing
  - You know how the code is supposed to work
  - Your tests test for how it is supposed to work
- Black box testing
  - Tester doesn’t know how the code works
  - All that is know is that is the output for an input
  - This is for release and user acceptance testing

Test driven development TDD

- You take the specifications and start writing tests for a class
- Then every time you run your code it auto checks
- Its Black Box testing
  - Did the right answer come out for you put it?
- It can be done at different levels
  - Unit is most common, but it can be used for integration etc.

Advantages :

- Integrates Specification and Coding and Testing
  - It’s as much specification as it is testing
  - It’s also part of documentation
- Makes you think about how code is used, before it’s built
- Plan before you write
- Code while thinking of the bigger picture rather than the current function
- When you make a change – it checks that you haven’t broke anything

The default fall-back is to embedded testing inside the code. It’s bad as our file gets too big. So we choose to use separate programs for testing.

Recommended approach

- Write the test first
- Write an empty class (with stub methods)
- Check that the tests fail
- Write code for the first test (just enough code to pass the test)
- Only move on when the tests for that code pass

Good test

- Line Coverage (has every line of code been tested)
- Function Coverage (has every function been tested)
- Condition Coverage (has each evaluation point been tested)
- Path coverage (has every possible route trough the code been tested)
- Entry/Exit coverage (has every call/return option been tested)
- Should never have to edit them
- Should cover all possibilities with multiple asserts