Software Testing

Introduction

Introduction

- Software is one of the most complex things built by humans.
- To get it right you need to consider:
  - will the software produce the correct results?
  - will the software work with edge cases like no data?
  - will the software have the required efficiency?
  - will the software fail under high loads?
  - will the software do something sensible when it receives bad data?
  - has the software been integrated with the rest of the system so that there are no interactions which cause errors?
  - if there is an unanticipated error, can the software recover and return itself to a stable state?

Software Testing

- Software testing is the process of
  - testing your software to see if it works and
  - meets all of its performance
  - and load requirements.

What Does Testing Prove?

- Testing proves that the tests you ran work correctly.
- It does not prove that your software is correct.
- Your software could pass all tests with flying colours yet still have undiscovered bugs in it.
- The amount of bugs discovered is proportional to the amount of time spent debugging.

Debugging

- Testing to determine if there are problems is only part of the process.
- After you determine that the software is incorrect, you need to find and fix the problem.
- This can be a highly complex and involved process that can take a large amount of time.

The Debugging Process

- The process of debugging can be broken down into smaller steps:
  - locate the source of the bug,
  - determine how to fix the bug,
  - implement the solution.

Testing Versus Debugging

- Testing and debugging are not the same thing.
- Testing runs the software and determines that the results are correct.
- Debugging is the process of trying to locate and fix a problem in code that you know has a bug in it.

The Need for Testing

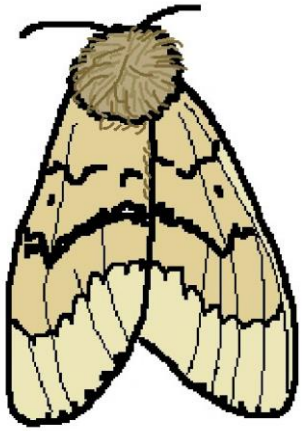
- Untested software usually fails at some point.
- Only testing can give us confidence that the software might be usable.
- Testing does not prove the software is correct.

Famous Software Bugs

- The history of software is littered with some of its greatest failures.
- By looking at these, we will see how even professionals can fail

The First Bug

- The first bug was discovered some 70 years ago by Grace Hopper, who was the developer of the COBOL language.
- She was using the Harvard mark II computer and discovered that a calculation was not being performed correctly and the problem was tracked down to a moth which was stuck between relays inside the computer.
- As a result of this, all problems associated with software have been described as bugs.



Therac-25

- Therac-25 was a machine to treat cancer with radiation
- It was built by Atomic Energy of Canada Ltd.
- The original version used hardware stops to prevent errors.
- The revised version used software to prevent errors.
- There was a bug in parallel programming that caused several deaths.
- At first, they thought it was an electrical problem as software cannot fail.
- The programmer did not understand how to do parallel programming.

The Ariane 5 Disaster

- In June of 1996 the very first Ariane 5 rocket was launched.
- 37 seconds after launch, the rocket rotated 90 degrees, in the wrong direction.
- less than two seconds later started to break up due to aerodynamic forces.
- This triggered the Rockets self-destruction mechanism causing it to explode in midair.
- This disaster cost approximately $370 million.
- A 64 bit floating point value was used to track a guidance variable.
- at some point in the calculation the value was assigned to a 16 bit integer.
- This worked for a few minutes until the value got too large.
- It then sent incorrect values to the guidance system.

Types of Testing

- Unit testing
- Black box
- White box
- Integration
- Functional
- End-to-end
- Load
- Security
- Acceptance
- Regression

Unit testing

- It breaks the code into small units, usually functions, and tests them to determine if they produce the correct results.
- The idea is to test small units of code and make sure that they work perfectly before they are combined with other pieces of code into much larger programs.
- It is easier to find a bug in a small piece of code than it is to find it in a larger program.
- If we know that the individual units that we use in a program are free from bugs, we do not have to look at the low-level code to check for bugs but rather look at the way it has been integrated to find the bugs.
- Unit tests normally test functions by passing them a known piece of data and checking to see that the result is what is expected.
- In most cases, this is done by automated tools that run a series of tests and report the results.
- Later in the course, we will be looking at tools to aid in the running of unit tests.

Black box testing

- Black box testing is a way of designing unit tests.
- It treats every functional unit as a black box which means that we have no idea what is inside the black box but we know that if we put a certain value into the box then a certain value should come out of the box.
- Black box testing relies entirely on knowing that a certain input should produce a certain output.
- This is normally the first approach to designing unit tests.

White box testing

- White box testing is almost the opposite of black box testing.
- Whereas with black box testing, we know nothing about the internal structure of the code, with white box testing we take advantage of the internal structure of a code to design tests that test every path through the code.
- If we simply rely on black box testing there, is no guarantee that we would have tested every single path through the code.
- White box testing produces additional tests which test every path through the code. This results in much higher confidence that the code works correctly.

Integration testing

- While unit testing tests the low-level functionality, we also need to test to make sure that the individual units were combined correctly and that they work as a whole.
- This is the job of integration testing which does not look at the individual functions but looks at groups of functions to ensure that they work together correctly.

Functional Testing

- Functional testing is similar to integration testing but focuses on the business requirements of the application.
- Verifies that the output is correct without checking the internal state of the system.
- An integration test might check that you accessed the database, updated some information, and committed your change. The functional test for the same thing might simply check to see that the correct value had been stored in the database without checking to see exactly how it was done.

End to End Testing

- End to end tests replicate the user behavior and make sure that all the tasks the user would perform work as expected.
- It is similar to acceptance testing because testers will replicate end-user behavior, like making a transaction through the website.
- However, Acceptance Testing is typically executed by business users, while end-to-end testing is performed by a technical testing team or quality assurance (QA).

Load Testing

- Load testing checks that the software functions correctly under high loads.
- Example: For a web application it might test to see if it can handle 100 simultaneous connections. A large data application might want to ensure that the database can handle 1000 simultaneous queries and give the required response time.
- Load testing is often combined with stress testing which tries to take it beyond the normal operating requirements to see when it will actually break.
- If stress testing reveals that your software will not break until well beyond your expected usage of it, then you are assured your software is in good shape.

Security Testing

- Security testing determines if the software meets the security requirements.
- Example:
  - It might involve checking that every web page can only be reached after signing on to the website.
  - It might involve checking the level of user access to make sure that users can only access the information they are allowed to.

Acceptance Testing

- Acceptance testing verifies that the application meets all the business requirements.
- The entire finished application is tested, usually by running examples of how the user would actually use the system.
- If all of the acceptance tests are met, then it indicates that the customer requirements have been met.

Regression Testing

- Regression testing means we perform our existing tests after every change to the code.
- The goal of regression testing is to make sure that any changes we made to the code do not break the existing code.
- It is used in many software development methodologies where, after a change is made to the code, the regression tests are run to ensure that it is bug-free before being checked into the repository.
- The goal of regression testing is to make sure that the code checked into the shared repository is free of bugs.