UMT MOOC
Massive Open Online Course

MINISTRY OF HIGHER EDUCATION

UNIVERSITI MALAYSIA TERENGGANU

UMT MOOC
Massive Open Online Course

SOFTWARE ENGINEERING

UNIVERSITI MALAYSIA TERENGGANU

MINISTRY OF HIGHER EDUCATION

UNIVERSITI MALAYSIA TERENGGANU

UMT MOOC
Massive Open Online Course

SOFTWARE TESTING

Prepared by:

Fatimah Ghazali

Fakulti Informatik dan Komputeran

Universiti Sultan Zainal Abidin

Email: fatimah@unisza.edu.my

Objectives

The objective of this chapter is to introduce software testing and software testing processes.

You will understand the stages of testing from testing, during development to acceptance testing by system customers;

You will have been introduced to techniques that help you choose test cases that are geared to discovering program defects;

You will understand test-first development, where you design tests before writing code and run these tests automatically;

Know the important differences between component, system, and release testing and be aware of user testing processes and techniques

Contents

- Development testing
- Test-driven development
- Release testing
- User testing

Introduction

- Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use.
- When you test software, you execute a program using artificial data. You check the results of the test run for errors, anomalies, or information about the program's non-functional attributes.

Introduction

Testing process has 2 distinct goals:

- To demonstrate to developer and the customer that the software meets its requirements. For custom software, this means that there should be at least one test for every requirement in the requirement document. For generic software products, it means that there should be tests for all of the system features, plus combinations of these features, that will be incorporated in the product release.
- To discover situation in which the behaviour of the software is incorrect, undesirable, or does not conform to its specification. These are a consequence of software defects. Defect testing is concerned with rooting out undesirable system behaviour such as system crashes, unwanted interactions with other systems, incorrect computation, and data corruption.

Introduction

- The first goal leads to validation testing, where you expect the system to perform correctly using a given set of test cases that reflect the system's expected use.
- The second goal leads to defect testing, where the cases are designed to expose defects. The test cases in defect testing can be deliberately obscure and need not reflect how system is normally used. Of course, there is no definite boundary between these two approaches to testing.
- During validation testing, you will find defects in the system, during defect testing, some of the tests will show that the program meets its requirements.
- Diagram 8.1 explain the differences between validation testing and defect testing

Introduction

- Testing is part of a broader of software verification and validation (V&V).
- Verification and validation are not the same thing, although they are often confused.
- Barry Boehm, a pioneer of software engineering, succinctly expressed the difference between them (Boehm,1979):
  - Validation: are we building the right product?
  - Verification: are we building the product right?

Introduction

- V&V processes are concerned with checking that software being developed meets its specification and delivers the functionality expected by people paying for the software.
- These checking processes start as soon as requirements become available and continue through all stages of the development process.
- The aim of verification is to check that the software meets its stated functional and non-functional requirements.
- Validation, however, is a more general process. The aim of validation is to ensure that the software meets the specification demonstrating that the software does what the customer expects it to do.

Introduction

- Validation is essential because requirements specification do not always reflect the real wishes or needs of system customers and users.
- The ultimate goal of V&V processes is to established confidence that the software system is ‘fit to purpose’. This mean that the system must be good enough for its intended use. The level of required confidence depends on the system’s purpose, the expectation of the system users, and the current marketing environment for the system:
  1. Software purpose
  2. User expectation
  3. Marketing environment

Introduction

1. Software purpose: The more critical the software, the more important that it is reliable. For example, the level of confidence required for software used to control a safety-critical system is much higher than that required for a prototype that has been developed to demonstrate new product ideas.
2. User expectation: Because of their experiences with buggy, unreliable software, many users have low expectation of software quality. They are not surprised when their software fails. When a new system is installed, users may tolerate failures because the benefits of use outweigh the costs of failure recovery. In these situations, you may not need to devote as much time to testing the software. However, as software matures, users expect it to become more reliable to more thorough testing of later versions may be required.
3. Marketing environment: When a system is marketed, the sellers of the system must take into account competing products, the price that customers are willing to pay for a system, and the required schedule for delivering that system. In a competitive environment, a software company may decide to release a program before it has been fully tasted and debugged because they want to be the first into the market. If a software product is very cheap, users may be willing to tolerate a lower level of reliability.

Inspection and Testing

As well as software testing, the V&V process may involve system inspections and reviews.

Inspections and reviews analyse and check the system requirements, design models, the program source code and even proposed system tests.

These are so-called ‘static’ V&V techniques in which you don’t need to execute the software to verify.

Figure 8.2 shows that software inspections and testing support V&V at different stages in the software process

The arrow indicate the stages in the process where the techniques may be used

Inspection mostly focus on the source code of a system, but any readable representation of the software, such as its requirement or a design model, can be inspected.

When you inspect a system, you use knowledge of the system, its application domain, and the programming or modelling language to discover errors

there are three (3) advantages of software inspection over testing

During testing, errors can mask (hide) other errors. When an error leads to unexpected outputs, you can never be sure if later output anomalies are due to a new error or are side effects of the original error. Because inspection is a static process, you don't have to be concerned with interactions between errors. Consequently, a single inspection session can discover many errors in a system.

Incomplete versions of a system can be inspected without additional costs. If a program is incomplete, then you need to develop specialized test harnesses to test the parts that are available. This obviously adds to the system development costs.

As well as searching for program defects, an inspection can also consider broader quality attributes of a program, such as compliance with standards, portability, and maintainability. You can look for inefficiencies, inappropriate algorithms and poor programming style that could make the system difficult to maintain and update.

- Inspection cannot replace software testing.
- Inspection are not good for discovering defects that arise because of unexpected interactions between different parts of program, timing problems, or problems with system performance.

Model of The Software Testing Process

- Figure 8.3 is an abstract model of the ‘traditional’ testing process, as used in plan-driven development.
- Test cases are specifications of the inputs to the test and the expected output from the system (the test results), plus a statement of what is being tested.
- Test data are the inputs that have been devised to test a system. Test data can sometimes be generated automatically, but automatic test case generation is impossible, as people who understand what the system is supposed to do must be involved to specify the expected test results.
- Test execution can be automated. The expected results are automatically compared with the predicted results so there is no need for a person to look for errors and anomalies in the test run.

Three Stages of Testing

MINISTRY OF HIGHER EDUCATION

Universiti Malaysia Terengganu

UMT MOOC Massive Open Online Course

Development testing, where the system is tested during development to discover bugs and defects. System designers and programmers are likely to be involved in the testing process.

Release testing, where a separate testing team test a complete version of the system before it is released to users. The aim of release testing is to check that the system meets the requirements of system stakeholders.

User testing, where users or potential users of a system test the system in their own environment. For software products, the ‘user’ may be an internal marketing group who decide if the software can be marketed, released and sold. Acceptance testing is one type of user testing where the customer formally tests a system to decide if it should be accepted from the system supplier of if further development is required.

- In practice, testing process usually involves a mixture of manual and automated testing.
- In manual testing, a tester runs the program with some test data and compares the results to their expectations.
- They note and report discrepancies to the program developers.
- In automated testing, the tests are encoded in a program that is run each time the system under development is to be tested.
- This is usually faster than manual testing, especially when it involves regression testing – re-running previous tests to check that changes to the program have not introduces new bugs.
- However, testing can never be completely automated as automated tests can only check that a program does what it is supposed to do.
- It is practically impossible to use automated testing to test systems that depend on how things look (e.g., a graphical user interface) or to test that a program does not have unwanted side effects.

Subtopic: Development Testing

- Development testing includes all testing activities that are carried out by the team developing the system.
- The tester of the software is usually the programmer who developed that software, although this is not always the case.
- Some development processes use programmer/tester pairs (Cusamano and Selby,1998) where each programmer has an associated tester who develops test and assists with the testing process.
- For critical systems, a more formal process may be used, with a separate testing group within the development team.
- They are responsible for developing tests and maintaining detailed records of test results.

During Development, Testing May be Carried Out at Three Levels of Granularity

1. Unit testing, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
2. Component testing, where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.
3. System testing, where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.

Development testing is primarily a defect testing process, where the aim of testing is to discover bugs in the software.

It is therefore usually interleaved with debugging – the process of locating problems with the code and changing the program to fix these problems.

1. Unit Testing

- Unit testing is the process of testing program components, such as methods or object classes.
- Individual functions or methods are the simplest type of component.
- Your test should be call to these routines with different input parameters.
- You can use the approaches to test case design to design the function or method tests.
- When you are testing object classes, you should design your tests to provide coverage of all the features of the object. This mean that you should:
  - Test all operations associated with the object;
  - Set and check the value of all attributes associated with the object;
  - Put the object into all possible states. This means that you should simulate all events that cause a state change.

- Whenever possible, you should automate unit testing.
- In unit testing, you make use of a test automation framework (such as Junit) to write and run your program tests.
- Unit testing framework provide generic test classes that you extend to create specific test cases.
- They can then run all of the tests that you have implemented and reported, often through some GUI, on the success or failure of the tests.
- An entire test suite can often be run in a few seconds so it is possible to execute all the test every time you make change to the program.

An automated test has 3 parts:

- A setup part, where you initialize the system with the test case, namely the inputs and expected outputs.
- A call part, where you call the object or method to be tested.
- An assertion part where you compare the result of the call with the expected result. If the assertion evaluates to true, the test has been successful; if false, then it has failed.

- Sometimes the object that you are testing has dependencies on other objects that may not have been written or which slow down the testing process if they are used.
- For example, if your object calls a database, this may involve a slow setup process before it can be used.
- In these cases, you may decide to use mock objects.
- Mock objects are with the same interface as the external objects being used that simulate in functionality.
- Therefore, a mock object simulating a database may have only a few data items that are organized in an array.
- They can be accessed quickly, without the overheads of calling a database and accessing disks.
- For example, if you system is intended to take action at certain times of day, your mock object can simply return those times, irrespective of the actual clock time.

Choosing Unit Test Cases

- Testing is expensive and time consuming, so it is important that you choose effective unit test case. Effectiveness, in this case, mean two things:
    1. The test cases should show that, when used as expected, the component that you are testing does what it is supposed to do.
    2. If there are defects in the component, these should be revealed by the test cases.
- Therefore, you should write two kinds of test cases.
- The first of these should reflect normal operation of a program and should show that the component works.
- For example, if you are testing a component that creates and initializes a new patient record, then your test case should show that the record exists in a database and that its fields have been set as specified.
- The second of test case should be based on testing experience of where common problems arise. It should use abnormal inputs to check that these are properly processes and do not crash the component.

2 Possible Strategies That can be Effective in Helping You Choose Test Cases:

MINISTRY OF HIGHER EDUCATION

Universiti Malaysia Terengganu

UMT MOOC Massive Open Online Course

Partition testing, where you identify groups of inputs that have common characteristics and should be processed in the same way. You should tests from within each of these groups.

Guideline-based testing, where you use testing guidelines to choose test cases. These guidelines reflect previous experience of the kinds of errors that programmers often make developing components.

1. Partition Testing

- Equivalence partitioning testing is effective approach to testing because it helps account for errors that programmers often make when processing inputs at the edges of partitions.

2. Guideline-based Testing

- Choose inputs that force the system to generate all error messages;
- Design inputs that cause input buffers to overflow;
- Repeat the same input or series of inputs numerous times;
- Force invalid outputs to be generated;
- Force computation results to be too large or too small

2. Component Testing

- Software component composite components that are made up of several interacting objects.
- Components may be:
  - Individual functions or methods within an object;
  - Object classes with several attributes and methods;
  - Composite components with defined interfaces used to access their functionality.
- Testing composite components should therefore focus on showing that the component interface behave according to its specification.
- There are different types of interface between program component and consequently, different types of interface error that can occur:
  1. Parameter interface
  2. Shared memory interface
  3. Procedural interface
  4. Message passing interfaces

Interface Error

Interface error are one of the most common forms of error in complex systems. These error fall into 3 classes

- Interface misuse
- Interface misunderstanding
- Timing errors

Interface Testing General Guidelines

1. Examine the code to be tested and explicitly list each call toa an external component.
2. Where pointers are passed across an interface, always test the interface with null pointer parameters.
3. Where a component is called through a procedural interface, design tests that deliberately cause the component to fail.
4. Use stress testing in message passing systems.
5. Where several components interact through shared memory, design tests that vary the order in which these components are activated.

3. System Testing

- System testing during development involves integrating components to create a version of the system and then testing the integrated system.
- System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interface.

Test-Driven Development (TDD)

- TDD is an approach to program development in which you interleave testing and code development.
- The steps in te TDD process are as follow:

1. Start by identifying the increment of functionality that is required.
2. Write a test for this functionality and implement this as an automated test.
3. Then run the test, along with all other tests that have been implemented.
4. Then implement the functionality and re-run the test.
5. Once all tests run successfully, move on to implementing the next chunk of functionality.

Benefit of TDD

- Better problem understanding
- Code coverage
- Regression testing
- Simplified debugging
- System documentation

Release Testing

- Release testing is the process of testing a particular release of a system that is intended for use outside of the development team.
- Goal : to convince the supplier of the system that it is good enough for use.
- RT is a black-box testing process where tests are derived from the system specification.
- Three type of RT:

1. Requirement-based testing
2. Scenario testing
3. Performance testing

User Testing

- User testing (UT) is a stage in the testing process in which uses provide input and advice on system testing.
- UT is essential because it is influences from the user's working environment have a major effect on the reliability, performance, usability and robustness of a system.
- Three type of UT:

1. Alpha testing, where users of the software work with the development team to test the software at the developer's site.
2. Beta testing, where a release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.
3. Acceptance testing, where customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment.