Software Testing

Testing at the end of the development has two risks:

Projects tend to **run over-time and over-budget**. Testing is often rushed, omitted, or ignored thanks to pressures

It is **more work to fix** problems discovered late in a process than to do things right from the start

When should testing be done?

"Test early, Test often"

To ensure quality and save time and money, accessibility evaluations should start **right at the beginning of product design**

What can be caused by software bugs?

- Ugly presentation
- Not showing output
- Unable to do operation
- Lost of time and money
- Privacy leaks, lost accounts
- Injuries, Life-threatening dangers and disasters

Why so many bugs in software?

- Requirements are often vague
- Software can be really complex

The Objectives of Testing

1- **It does work**: Show that the system works

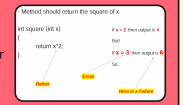
2- **It does not work**: Try to make a unit fail

3- **Reduce the risk of failures**

4- **Reduce the cost of testing**

Basic Definitions:

- Failure
  - A **failure** is occur when external behavior of a system does not conform to the system specification
  - Observable incorrect behavior
- Fault
  - A **fault** is the adjudged cause of an error
  - Incorrect code
  - **Fault** = **Bug** = **Defect**
- Error
  - An **error** is a state of the system
  - An **error** state could lead to a **failure** in the absence of any corrective action by the system
  - A cause to **fault**



What is Software Testing?

A software process based on well-defined software quality control and testing standards, testing methods, strategy, test criteria, and tools

Very important **software development phase**

Software testing is a process used to identify the correctness, completeness, and quality of a developed software

Important Considerations

- Detect failures by **choosing test inputs carefully**.
- **Determine the faults** leading to failures.
- **Repair the faults** leading to failures.
- **Re-test** the module/system.

Software Testing Terminology

- **Test case**
  - A test case consists of inputs, steps/actions, and expected results
- **Test suite**
  - A group of test cases
- **Test plan**
  - A document that specifies how a system will be tested, including criteria for success
- **Testing process**
  - The testing process involves developing test plans, designing test cases, running the test cases, and evaluating the results

Who does Software Testing?

- **Test manager**
  - Manage and control a software test project
  - Supervise test engineers
  - Define and specify a test plan
- **Software Test Engineers and Testers**
  - Define Test Cases
  - Write Test Specifications
  - Run Tests
- **Independent Test Group**
- **Development Engineers**
  - Only perform unit and integration tests
- **Quality Assurance Group and Engineers**
  - Perform system testing
  - Define software testing standards and quality control process

What Exactly Does a Software Tester Do?

The goal of a software tester is to **find bugs**, find them **as early as possible**, and **fix** them

What Makes a Good Software Tester?

Good understanding of the development process and its products

Ability to anticipate

Traits Software testers must have:

- They are explorers
- They are troubleshooters
- They are relentless
- They are creative

Software Testing Principles

Principle #1

Complete testing is impossible

Principle #2

Software testing is not simple activity

Why?

Quality testing requires understanding to the system completely, also the very tight schedule and lack of test tools

Principle #3

Testing is risk-based

Principle #4

Testing must be planned

Principle #5

Testing requires independence

Principle #6

Quality software testing depends on:

- Good understanding
- Cost-effective testing methodology
- Good engineers with creativity

Designed by: Soft Conclusion

YouTube link: Soft Conclusion - YouTube

Miro link: Software Testing & QA midterm MindMap

Levels of Dynamic Testing:

Unit Testing
Performed at individual unit of software, performed by developers (programmers)

Integration Testing
Performed by combining all the individual units, performed by developers or testers

System Testing
Performed to ensure whether the system performs meet the requirements

Acceptance Testing
Performed to verify if system met the business requirements and if ready to deployment

Dynamic Testing (Validation)

Dynamic Testing is the process of testing and analyzing the dynamic behavior of the code

Done during **Validation** process

Two categories of Dynamic Testing:
Functional Testing
Non-Functional Testing

Software Testing Methods

Static Testing (Verification)

Static Testing is a process of testing the software manually, without executing the software

This testing is also called as **Non-execution technique** or **verification testing**

No Need for computer in Testing

It has two parts:

Review

To find and eliminate errors or ambiguities in documents such as requirements, or design, etc.

Types of Reviews:
- Walkthrough
- Inspection

Static Analysis

The code written by developers is analyzed (by tools⚒) for structural defects that may lead to errors

Types of defects found by **static analysis**:
- A variable with an undefined value
- Programming standards violations
- Security vulnerabilities

Techniques for Dynamic Testing:

📦 Black Box Testing

- **Functional Testing** techniques
- Based entirely on the **requirements** and **specs**
- Focus on **inputs** and **output**
- No bothering with **internal code**
- Can be applied to all **dynamic testing levels**
- Tester only aware of what the software is supposed to do, not how it does it

Who does **Black Box testing**?
- Testers
- Business Analyst
- Clients

Why not done by developers?
- To test the software from end-user perspective

When it is done?
- Once the functionality is developed

Techniques involved in **Black Box testing**:
- Equivalence Class
- Boundary Value Analysis
- Domain Tests
- etc.

✎ What is the difference between Requirements and Specifications?

Specifications are requirements with more details, for example: login is the requirement, but login with unique user id and password is the specification

📦 White Box Testing

Testers require knowledge of how the software is implemented, and how it works

Also known as:
- **Structural Testing**
- **Clear Box Testing**
- **Glass Box Testing**

Primarily examines **source code** with a focus on **control flow** and **data flow**

Techniques involved in **Whit Box testing**:
- Statement Coverage: aim at exercise all programming statements
- Branch Coverage: Running series of tests to ensure all branches are tested
- Path Coverage: Testing all possible paths, which mean each statement and branch is covered

Verification & Validation (V & V) Process

V & V takes place at each phase of software development life cycle

V & V has 2 principals:

- The discovery of defects in a system
- The assessment of whether the system is useful and useable in operational situation

verification is concerned with whether the system is well-engineered, error-free, and so on, while **validation** is concerned with checking that the system meet the customer's needs

Software is not necessarily 100 % free of defects

Verification & Validation (V & V)

Verification

Verification is an activity to confirm that something conforms to its specifications, standards, regulations, etc.

In other words, **verification** ensures that "**you built it right**"

Are we building the product right?

Verification plays an important role in quality control

Verification Tools

⚒ Walkthroughs (Reviews)

A **walkthroughs** is an activity in which people **other than the author** walk through every sentence of the software information

⚒ Inspections

It ensure that all required components are **ready for the next stage**

⚒ Audits

Used as a QA tool, to ensure project execution, to the organization's defined software development processes

Usually duration of a project **audit** is 1-2 hours

Validation

Validation the process to evaluate a system (or component of the system) **during or at the end of the development**, to determine whether it satisfies specified requirements

Are we building the right product?

V & V Goal

To develop a software that **fits the purpose** it was developed for

Designed by: Soft Conclusion
YouTube link: Soft Conclusion - YouTube
Miro link: Software Testing & QA midterm MindMap

A **Test Case** contains:
- A sequence of steps describing actions to be performed
- Test data to be used
- An expected response for each action performed

A **Test Suite** is a **collection of test cases**

Test engineers can **focus on different aspects of a system**, one at a time, while evaluating its quality

Test engineers can **prioritize their tasks** based on test categories

Planning the **system testing** phase based on test categorization lets a **test engineer** to **obtain a well-balanced test suite**

- Creating a test plan
- Designing a test suite
- Preparing test environments
- Executing the tests
- Monitoring the process of test execution

System Testing Activities

System Test Advantages

3. System Testing

System testing is a critical phase in a software development process, due the need to meet a tight schedule close to delivery date

System testing objective:
To establish whether an implementation conforms to the requirements specified by the customers

1. Unit Testing

Testing program **units** in isolation (in a stand-alone manner)

Unit Test is performed by using the **White Box** method

To verify that each **distinct execution** (distinct path in a unit) of a program unit produces the expected result

Example → JUnit (testing framework for Java)

What is a unit?

A **unit** is the smallest testable part of software

A program **unit** is a piece of code, such as Functions, Procedures, Methods, or Class

Two reasons for testing a unit in a stand-alone manner:

- Errors found during unit testing easily fixed
- Removes dependencies of the unit

How programmers needs to test a unit?

- Execute every line of code
- Execute every predicate in the unit, and evaluate them to true or false separately
- Observe that the unit performs its intended

Testing Levels

4. Acceptance Testing

The objective of **acceptance testing** is to **measure the quality** of the product, rather than searching for the defects

Acceptance testing is performed by the customers based on their expectations from the product

Types of Acceptance Test:

- User acceptance testing (UAT)
- Business acceptance testing (BAT)

2. Integration Testing

Any type of software testing that seeks to verify the **interfaces** between components

To verify the functional, performance, and reliability between the modules that are integrated

Approaches to perform system integration test

📈 **Incremental Approach**
Integration testing is conducted in an **incremental manner** as a series of **test cycles**, in each **test cycle**, a few more modules are integrated to generate a larger build.

⬇️ **Top-down Approach**
In a **hierarchical system**, there top-level module which is decomposed into a few second-level modules or further to third-level modules, and so on. A terminal module is one that is no more decomposed.

⬆️ **Bottom-up Approach**
In a **hierarchical system**, integration begins with the lowest level modules (lowest level modules does not invoke another module)

🌯 **Sandwich Approach**
Using a mix of the **top-down** and **bottom-up** approaches.

💥 **Big bang Approach**
All the modules are individually tested, then **put all modules together** to construct the entire system, then the whole system. For large systems, this approach is not recommended.

The objective of **system integration** is to build a **working version** by putting the modules together in an **incremental** manner

Who perform integration testing?

Integration testing is performed by:

- **Software Developers**
- **Integration Test Engineers**
  - Need to know the details of the software

System architects should be involved in the integration testing because they have the bigger picture of the system

Integration Test Advantages

- Defects are detected early
- Easier to fix defects
- Scheduling of defect fixes is flexible
- Early feedback on health and acceptability of modules and system

Types of Interfaces

1. Procedure Call Interface: A procedure in one module calls a procedure in another module

2. Shared Memory Interface: A block of memory is shared between two modules

3. Message Passing Interface: One module initialize the field of a data structure, and send message to another module, like:
- Client–server or web based systems

Designed by: Soft Conclusion
YouTube link: Soft Conclusion - YouTube
Miro link: Software Testing & QA midterm MindMap

Path Testing
- Execute all possible **control flow paths** through the program
- 100% path coverage

Statement Testing
- Execute all statements in a program at least once under some test
- 100% statement coverage

Branch Testing
- Execute enough tests to assure that every branch alternative has been exercised at least once
- 100% branch coverage

Two basic kinds of statements in a program unit:

- **Assignment statements**
  - Assignment symbol "= "
- **Conditional statements**
  - if(), for() loop, and while() loop

Sometimes, a small number of **conditional statements** can lead to a complex **control flow**

Statement coverage: Executing individual program statements and observing the outcome

For a program unit with **few paths**, executing all paths is desirable and achievable

For a program unit with **many paths**, executing every distinct path is NOT PRACTICAL

What paths do I select for testing?
- The path that visits most of the program

Criteria

Path Selection Criteria

Flow Control Testing

- **Control flow testing** is a kind of structural testing
- Performed by **programmers** to test their code
- It uses the program's **control flow** as a model
- It is a technique that comes under **white box testing**
- Based on selecting set of test **paths** through the program

Program Path

The execution of **sequence of instructions**, from **entry point** to **exit point** of a **program unit**

A program may contain a large number of paths, or even an infinite number of paths

Assumptions

Advantages of selecting paths

- **Specifications** are correct
- Data is defined and accessed properly
- There are no **bugs** except bugs that affect **control flow**

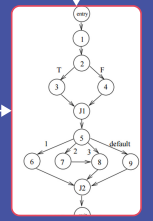
1. All program constructs are exercised (execute) at least once

2. We do not generate test inputs which execute the same path repeatedly

3. We know the program features that have been tested and those not tested

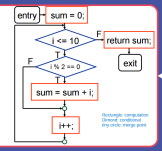
```
Program Trivial
1.  read(n);
2.  if (n<0) then
3.      write("negative");
    else
4.      write("positive");
    endif;
5.  switch (n)
    case 1:
6.      write("one");
        break;
    case 2:
7.      write("two");
    case 3:
8.      write("three");
        break;
    default:
9.      write("other");
    endswitch;
end Trivial
```

Conversion Example




```
int evensum(int i)
{
  int sum = 0;

  while (i <= 10) {
    if (n%2== 0)
      sum = sum + i;
    i++;
  }
  return sum;
}
```



Control Flow Graph (CFG)

A **Control Flow Graph** is a **graphical representation** of all paths that might be traversed through a program during its execution

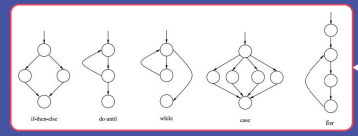
Consists of a set of nodes N and a set of edges E
G = (N, E)

Each node represents a set of program statements

Types of nodes

Decision node
Contains a conditional statement that creates 2 or more control branches (e.g. if or switch statements)

Merge node
Usually does not contain any statement, the point where multiple control branches merge



Designed by: Soft Conclusion
YouTube link: Soft Conclusion - YouTube
Miro link: Software Testing & QA midterm MindMap