your excellence partner

Quality Partners™

Effective Software Testing (EST)

Quality Partners™

www.SQ-Partners.com

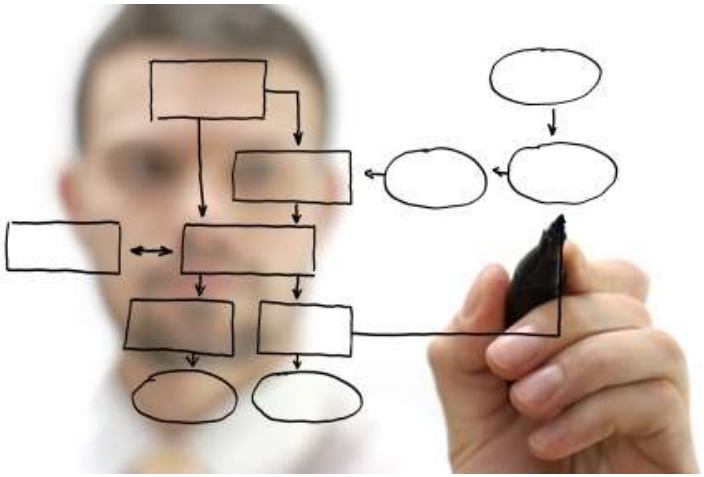
Course contents

- 1. Introduction to Software Engineering
- 2. Introduction to Quality Control Engineering
- 3. Quality Control Lifecycle During the SDLC
- 4. Test Planning and Management
- 5. How to Verify the Software Requirements
- 6. Effective Test Design Techniques
- 7. How to Create Effective Test Cases
- 8. Non-functional Testing Techniques
- 9. Test Reporting and Monitoring
- 10. Test Time Estimation and Defects Seeding

Part 1

Introduction to Software Engineering

What is Software Engineering?



Software engineering is the application of a systematic and quantifiable approach to the design, development, and maintenance of software.

What is the SDLC Model ?

"Software development life cycle" model describe phases of the software cycle and the order in which those phases are executed.

There are many models, and many companies adopt their own, but all have similar patterns or references.

What are the common SDLC Models ?

1. Big Bang
2. Code and Fix
3. Waterfall
4. Iterative
5. V-model

Note: There are many other models, and many companies adopt their own, but most models are inherited from the above listed models.

1. Big-Bang Model :

- The Approach

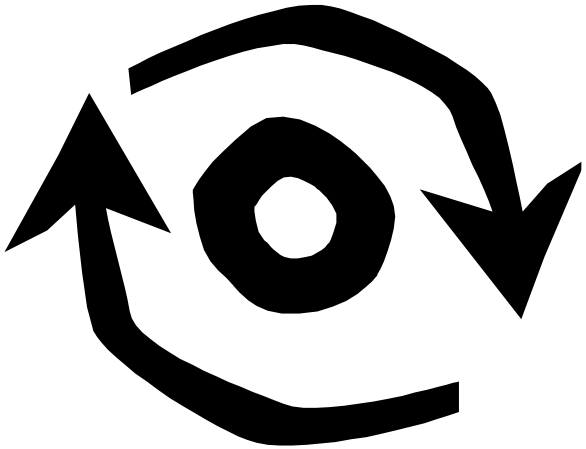
  People+Money+Energy=Perfect Software or Nothing!

- Advantages

  Simple, needs less planning, less formal development process

- Disadvantages

  No formal testing activities, hard to manage and monitor, risky



2. Code and Fix Model :

- The Approach

  Informal specification → loop of code and fix till enough-is-enough decision →Release

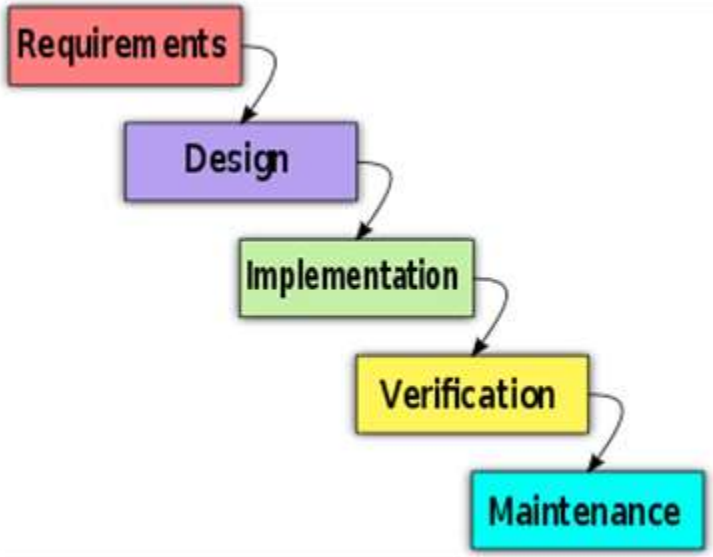
- Advantages

  Little planning and documentation work

- Disadvantages

  Hard to manage the testing and control the quality

3. Waterfall Model :



- The Approach

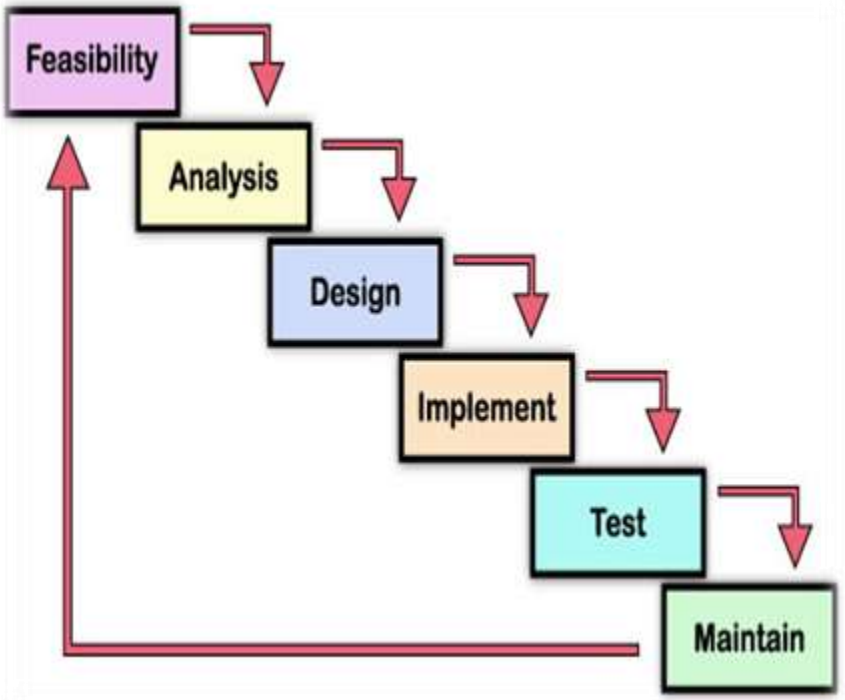
  Discrete phases, a review after each phase will decide to move next or stay at this phase till it's ready.

- Advantages

  Easy to test (clear requirements), easy to manage and monitor

- Disadvantages

  High bugs fixing and changes cost, the customer has minimal involvement during the project



4. Iterative Model :

- ## The Approach

Dividing the project into builds, each build can be executed by waterfall model.

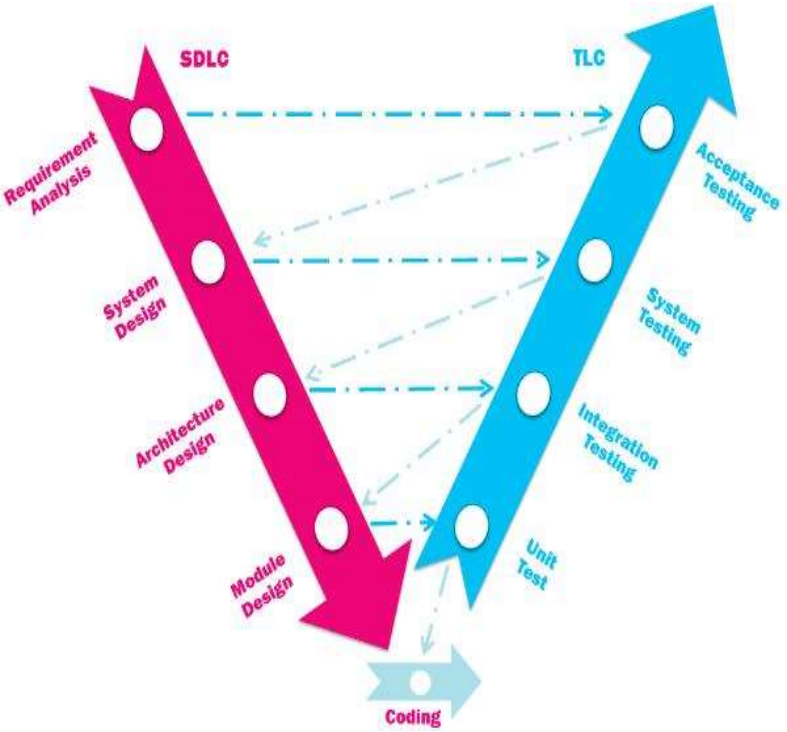
- ## Advantages

Easy to test, customer is tightly Involved, change cost is low.

- ## Disadvantages

Time consuming.

5. V- Model :



- ## The Approach

For every single phase in the development cycle there is a directly associated testing phase. It is an extension of the waterfall.

- ## Advantages

Testing is exist in every phase.

Defects are detected early

- ## Disadvantages

Time consuming.

Part 2

Introduction to Quality Control Engineering

What is Software Quality Control (QC)?

QC is the set of procedures used by organizations to ensure that a software product will meet its quality goals at the best value to the customer, and to continually improve the organization's ability to produce software products in the future.

Software quality control refers to specified functional requirements as well as non-functional requirements.

These specified procedures and outlined requirements leads to the idea of Verification and Validation and software testing.

What is Software Testing?

A quality control activity aimed at evaluating a software item against the given system requirements.

This includes, but is not limited to, the process of executing a program or application with the intent of finding software bugs.

Testing is a QC activity.

What is the Verification and Validation (V&V)?

Verification: Confirming that something (software) meets its specification.

Validation: Confirming that it meets the user's requirements.

The two major V&V activities are reviews, and testing.

Testing is part of the V&V activities.

What is the Software ‘Bug’ ?

Things the software does that while it is not supposed to do, or

something the software doesn't while it is supposed to.

Why Does Software Have Bugs ?

- Miscommunication or no communication.
- Programming errors.
- Changing requirements.
- Time pressures.
- Ego.
- Poorly documented code.
- Software development tools.

What should the Tester do?

1. Find the bugs
2. Find them early
3. Make sure that they have been fixed

What should not the Tester do?

1. Fix the bugs!
2. Test with good faith!
3. Test with no requirements!
4. Not to report the obvious bugs!
5. Mock at others because of their bugs!

Main Qualifications for a Good Software Tester:

- Be familiar with the SDLC being used.
- Have excellent attention to details.
- Have excellent communication skills (oral and written).
- Have strong personality and the ability to resist pressures and say 'no' to other managers when quality is insufficient.
- Be able to run meetings and keep them focused.
- Patient, Mature and diplomatic.
- Able to prioritize things and take critical decisions.

Part 3

Quality Control Lifecycle Through the SDLC

Testing Lifecycle main activities:

1. Test Planning.
2. Test Analysis and Design.
3. Test Cases preparation.
4. Test Execution.
5. Test Reporting and Monitoring.
6. Bugs Reporting and Regression Testing.
7. Releasing Reporting.
8. User Acceptance Test Execution.

What are the Testing Lifecycle main activities?

Project Phase	Team Deliverables	QC Deliverables
Initiation	○ Charter ○ Scope	○ Scope Verification
Planning	○ Project Plans	○ QC Plan ○ UAT Plan
Analysis	○ SRS ○ Use Cases	○ SRS Verification ○ Test Design ○ Test Cases
Design	○ Logic Design ○ DB Design (ERD) ○ GUI Design (Mockups)	○ Design Verification
Development and Testing	○ Builds	○ Functional and Non-Functional Testing ○ Bugs Reports ○ Final Release Report (FRR)
Delivery	○ Release	○ User Acceptance Test

Part 4

Test Planning and Management

What are the main components of the good Test Plan?

1. Introduction
    - 1.1 Objectives
    - 1.2 Scope (In scope and out of scope)
    - 1.3 Roles & Responsibilities
    - 1.4 Risks and Dependencies
    - 1.5 Deliverables

2. Resources and Environment Requirements
    - 2.1 Staffing and Training needs
    - 2.2 Test Environment
    - 2.3 Test Tools Requirements
    - 2.4 Test Data

What are the main components of the good test plan (cont'd)?

3. Schedule

4. Test Monitoring and Reporting

5. Pass/Fail criteria

Part 5

How to Verify the Software Requirements

What is the Software Requirements Specification Document?

A software requirements specification (SRS) is a comprehensive description of the intended purpose and environment for software under development. The SRS document fully describes what the software will do and how it will be expected to perform.

The Characteristics of a good Requirements:

1.Complete: The information covers all aspects of this user function.

2.Consistent: The information for this user function is internally consistent. There are no conflicting statements.

3.Correct: The item is free from error.

4.Testable: Can the requirement be easily mapped to specific test cases?

The Characteristics of a good Requirements : (Cont'd)

5. **Specific** :The item is exact and not vague; there is a single interpretation; the meaning of each item is understood; the specification is easy to read. Use well defined terms and enough information. Terms such as "High", "Low", "Quick", "Fast", "Big" are not acceptable.

6. **Relevant**: The item is related to the problem and its solution.

7. **Expressed** in the User's Language: All requirements shall be expressed using no design and in the users terminology.

The Characteristics of a good Requirements : (Cont'd)

8. **Understandable**: Difficult language should not be used. Be clear. If formal notations are used, make sure they are easy to understand. Figures and tables are helpful, but must be explained.

9. **Traceable**: The item can be traced to its origin in the problem environment. Also, you must be able to find who originated a requirement, who wrote it, what other requirement does this one impact. This information is important to make the requirements usable and maintainable (changeable).

The Characteristics of a good Requirements : (Cont'd)

10. **Feasible**: The item can be implemented with the available techniques, tools, resources, and personnel, and within the specified cost and schedule constraints .

11. **Free of unwarranted design details**: The requirements specifications are a statement of the requirements that must be satisfied by the problem solution, and they are not obscured by proposed solution to the problem.

12. **Prioritized**: The priority of this item relative to other requirements has been recorded.

Part 6

Effective Test Design Techniques

What is Test Design document?

Documentation specifying the details of the test approach for a software feature or combination of software features and identifying the associated tests.

Test Design Techniques:

1. Equivalence Partitioning

Equivalence partitioning is a test design method for deriving test cases. In this method, classes of input conditions called equivalence classes are identified such that each member of the class causes the same kind of processing and output to occur. In this method, the tester identifies various equivalence classes for partitioning. A class is a set of input conditions that are is likely to be handled the same way by the system. If the system were to handle one case in the class erroneously, it would handle all cases erroneously.

1. Equivalence Partitioning (Cont'd):

Why we should use Equivalence Partitioning when designing test cases:

1. To reduce the number of test cases (by avoiding the redundant cases), hence reducing the testing time and cost.

2. To assure better test coverage, by testing all input classes, hence improve the test efficiency.

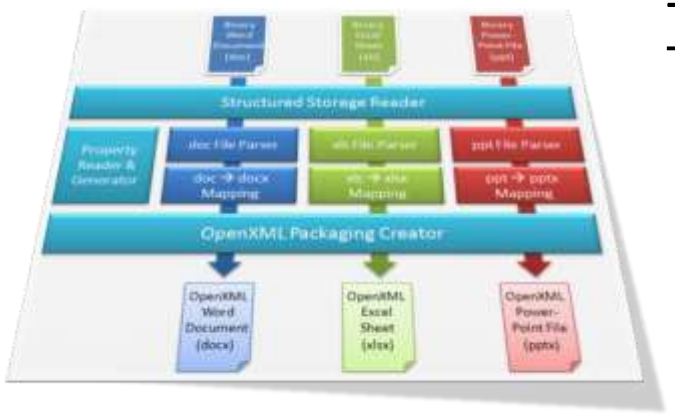
Test Design Techniques:

2. Boundary Values Analysis (BVA)

BVA is a methodology for designing test cases that concentrates software testing effort on cases near the limits of input ranges (the boundaries).

Note: BVA should be performed after identifying the Equivalence Partitions of the software.

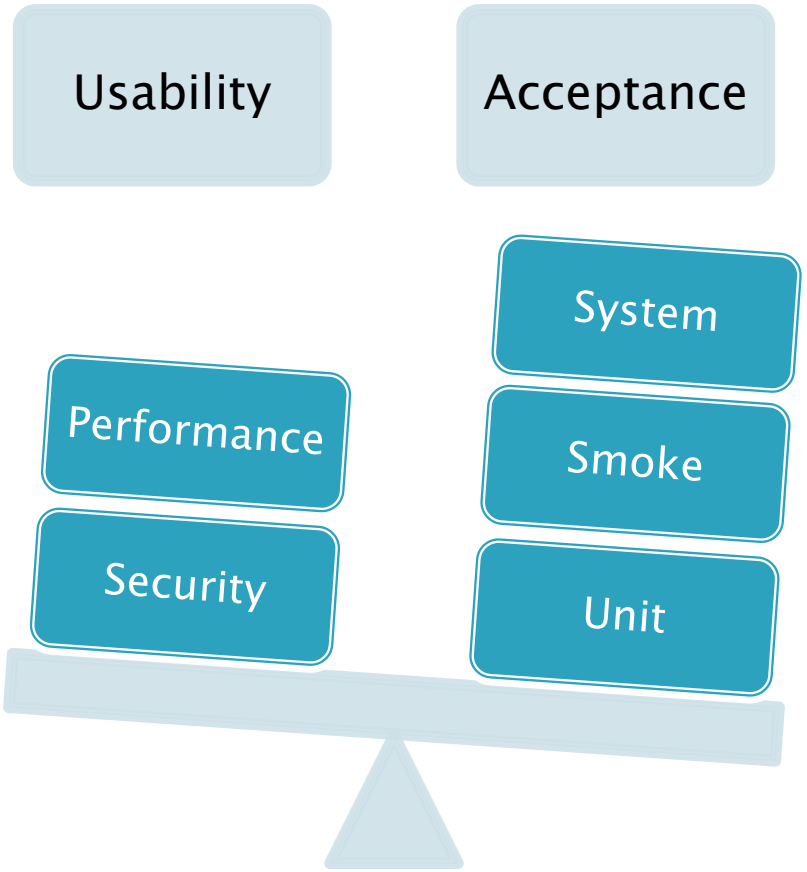
BVA generates test cases that highlight errors better than equivalence partitioning. The trick is to concentrate software testing efforts at the extreme ends of the equivalence classes. At those points when input values change from valid to invalid errors are most likely to occur.



What are the main components of the Good Test Design document?

1. Scope
    - 1.1 Features to be tested
    - 1.2 Testing types to be applied
2. Test Cases Structure
3. Test Levels
4. Test Approach (Design Matrix)
5. Pass/Fail criteria

Testing Types Vs Testing Levels:



Part 7

How to Create Effective Test Cases

What are the characteristics of a good Test Case?

- Testability; easy to be executed
- Use active case, do this, do that, system displays this, does that
- Simple, conversational language
- Exact, consistent names of fields, not generic
- Don't explain Windows basics
- 10-15 steps or less. In the case of integration test case's try to provide references instead of explaining the functionality again and again
- Always update your test cases along with new build

Part 8

Non-Functional Testing Techniques

Non-Functional Testing Types:

- Performance: the response time, utilization, and throughput behavior of the system.
  - Volume : Testing a software application for a certain data volume.
  - Load : Creating demand on a system or device and measuring its response.
  - Stress: Determine the stability of a given system or entity.
- Compatibility : Measures how well the system operates on different operating environment.
- Recovery : Testing a system's ability to recover from a hardware or software failure.

Non-Functional Testing Types (Cont'd):

- Availability: Is it available when and where I need to use it?
- Efficiency: How few system resources does it consume?
- Flexibility: How easy is it to add new features?
- Maintainability: How easy it is to correct defects or make changes?
- Portability: Can it be made to work on other platforms?
- Security: The system's ability to resist unauthorized attempts at usage or behavior modification, while still providing service to authorized users.
- Accessibility testing: measure how well the system interface serve users with disabilities.

Non-Functional Testing Types (Cont'd):

- Reliability: How long does it run before causing a failure?
- Reusability: How easily can we use components in other systems?
- Usability: How easy is it for people to learn or to use?
- Installability: How easy is it to correctly install the product.

Part 9

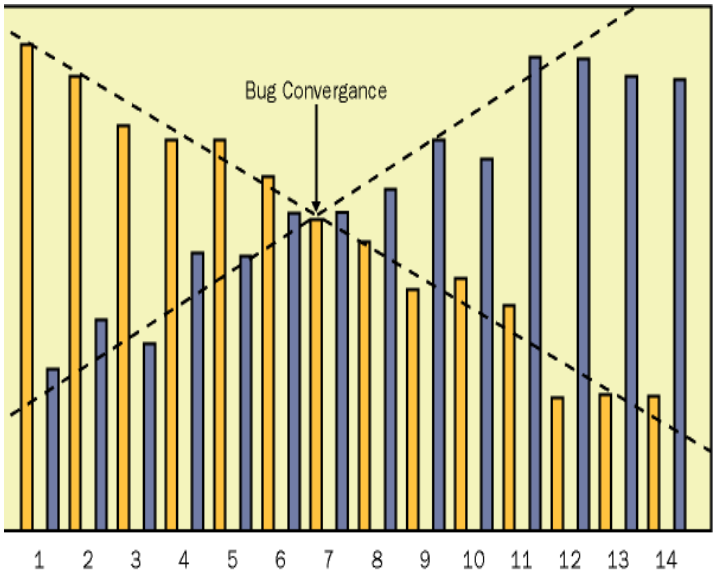
Test Reporting and Monitoring

How to write effective Bug report

- Report the pattern not an example.
- Don’t describe what's wrong, only!
- Be direct to the point (don’t tell a story!).
- Report the bug, but not how to solve it.
- Don’t mix between priority and severity.
- Don’t use: CAPS, red letters, red circles, ‘!’ , ‘?’.
- Don’t use your personal judgment.
- Write with the fixer language, not with yours.
- Minimize the options.
- Don’t use the word ‘Sometimes’.

Part 10

Test Time Estimation



Bug Convergence

- Bug convergence is the point at which the team makes visible progress against the active bug count. At bug convergence, the rate of bugs resolved exceeds the rate of bugs found; thus the actual number of active bugs decreases.
- Because the bug count will still go up and down, even after it starts its overall decline, bug convergence usually manifests itself as a trend rather than a fixed point in time.

Defects Seeding

- Defect seeding is a practice in which defects are intentionally inserted into a program by one group for detection by another group. The ratio of the number of seeded defects detected to the total number of defects seeded provides a rough idea of the total number of unseeded defects that have been detected.

$$IndigenousDefects_{Total} =$$

$$(SeededDefects_{Planted} / SeededDefects_{Found}) * IndigenousDefects_{Found}$$

Author Contacts

your excellence partner
Quality Partners™

Belal Raslan

Email: braslan@sq-partners.com