UNIT V-SOFTWARE ENGINEERING-18BIT41C

UNIT V: Verification and Validation Techniques: Quality Assurance – Static Analysis – Symbolic Execution – Unit Testing and Debugging – System Testing – Formal Verification.
Software Maintenance: Enhancing Maintainability During Development – Managerial Aspects of Software Maintenance – Configuration Management – Source-Code Metrics – Other Maintenance Tools and Techniques.

Software Quality Assurance (SQA) is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedures as well as standards suitable for the project and implemented correctly.

Software Quality Assurance is a process which works parallel to development of a software. It focuses on improving the process of development of software so that problems can be prevented before they become a major issue. Software Quality Assurance is a kind of an Umbrella activity that is applied throughout the software process.

Software Quality Assurance have:

- A quality management approach
- Formal technical reviews
- Multi testing strategy
- Effective software engineering technology
- Measurement and reporting mechanism

Major Software Quality Assurance Activities:

- **SQA Management Plan:**
  Make a plan how you will carry out the sqa through out the project. Think which set of software engineering activities are the best for project.check level of sqa team skills.
- **Set The Check Points:**
  SQA team should set checkpoints. Evaluate the performance of the project on the basis of collected data on different check points.
- **Multi testing Strategy:**
  Do not depend on single testing approach. When you have lot of testing approaches available use them.
- **Measure Change Impact:**
  The changes for making the correction of an error sometimes re introduces more errors keep the measure of impact of change on project. Reset the new change to change check the compatibility of this fix with whole project

- **Manage Good Relations:**
  In the working environment managing the good relation with other teams involved in the project development is mandatory. Bad relation of SQA team with programmers team will impact directly and badly on project. Don't play politics.

Benefits of Software Quality Assurance (SQA):

- ✓ SQA produce high quality software.
- ✓ High quality application saves time and cost.
- ✓ SQA is beneficial for better reliability.
- ✓ SQA is beneficial in the condition of no maintenance for long time.
- ✓ High quality commercial software increase market share of company.
- ✓ Improving the process of creating software.
- ✓ Improves the quality of the software.

Static Analysis

Static analysis involves a set of methods used to analyze software source code or object code determine how the software functions and establish criteria to check its correctness. Static analysis studies the source code without executing it and reveals a wide variety of information such as the structure of the model used, data and control flow, syntax accuracy, and more.

There are several types of static analysis methods-

- **Control Analysis :-**
  This software focuses on examining the controls used in calling structure, control flow analysis and state transition analysis. The calling structure is related to the model by identifying the calling and call structure. The calling structure can be a process, subroutine, function, or method. Control flow analysis checks the sequence of control transfers. Furthermore, it inefficient constructions in the model. A graph of the model is created in which the conditional branches and model junctions are represented by nodes.

- **Data Analysis :-**
  Ensures proper operation is applied to data objects such as data structures and linked lists. In addition, this method also ensures that the defined data is used properly. Data analysis involves two methods, namely, data dependency and data-flow analysis. Data dependency is necessary to assess the accuracy of synchronization across multiple processors. Data flow analysis checks the definition and context of variables.
- **Fault/Failure Analysis :-**
  It analyzes faults (incorrectly component) and failure (incorrect behavior of model component) in the model. This method uses the input-output transformation description to identify the conditions that are cause for the failure. To determine the failures in certain conditions the model design specification is checked.
- **Interface Analysis :-**
  This software verifies and verifies interactive and distribution simulations to check the code. There are two basic techniques for interface analysis and user interface analysis examines sub model interfaces and determines the accuracy of interface structure. User interface analysis examines the user interface model and for the precautionary steps taken to prevent errors during the user's interaction with the model. This method also focuses on how accurately the interface is integrated into the overall model and simulation.

Symbolic Execution

Symbolic execution is a software testing technique that is useful to aid the generation of test data and in proving the program quality.

Steps to use Symbolic Execution:

- The execution requires a selection of paths that are exercised by a set of data values. A program, which is executed using actual data, results in the output of a series of values.
- In symbolic execution, the data is replaced by symbolic values with set of expressions, one expression per output variable.
- The common approach for symbolic execution is to perform an analysis of the program, resulting in the creation of a flow graph.
- The flowgraph identifies the decision points and the assignments associated with each flow. By traversing the flow graph from an entry point, a list of assignment statements and branch predicates is produced.

Issues with Symbolic Execution:

- Symbolic execution cannot proceed if the number of iterations in the loop is known.
- The second issue is the invocation of any out-of-line code or module calls.
- Symbolic execution cannot be used with arrays.
- The symbolic execution cannot identify of infeasible paths.

Symbolic Execution Application:

- Path domain checking
- Test Data generation
- Partition analysis
- Symbolic debugging

UNIT TESTING AND DEBUGGING

What is Unit Testing?

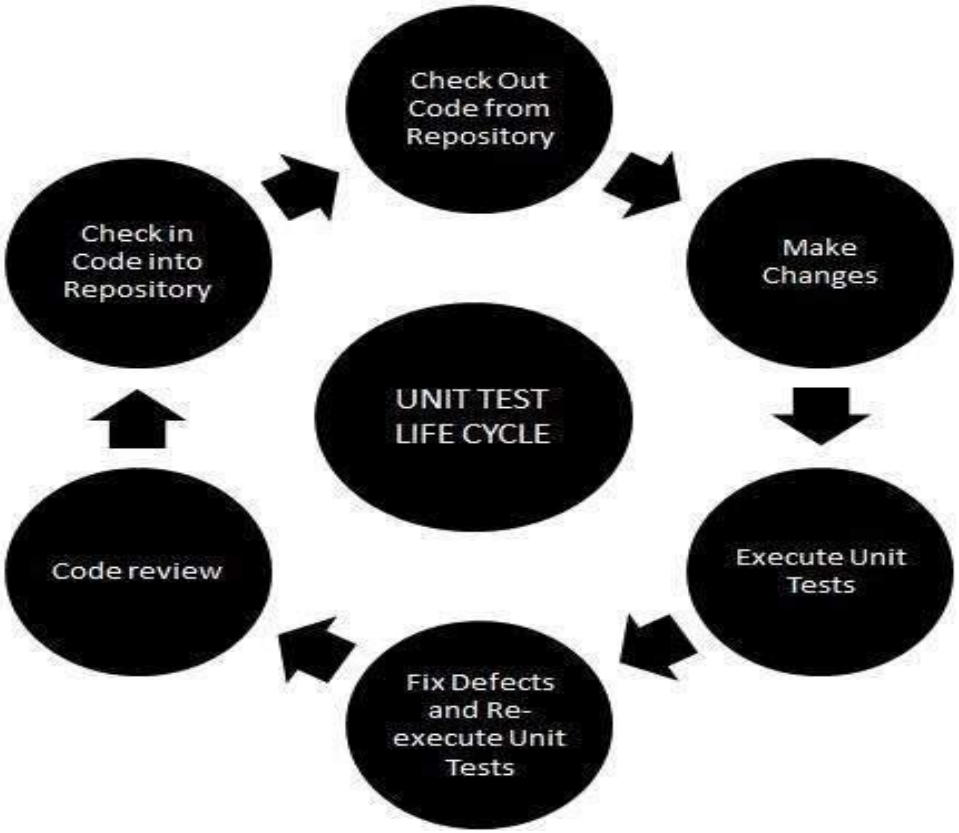
Unit testing, a testing technique using which individual modules are tested to determine if there are any issues by the developer himself. It is concerned with functional correctness of the standalone modules.

The main aim is to isolate each unit of the system to identify, analyze and fix the defects.

Unit Testing - Advantages:

- Reduces Defects in the Newly developed features or reduces bugs when changing the existing functionality.
- Reduces Cost of Testing as defects are captured in very early phase.
- Improves design and allows better refactoring of code.
- Unit Tests, when integrated with build gives the quality of the build as well.

Unit Testing LifeCyle:



Unit Testing Techniques:

Black Box Testing - Using which the user interface, input and output are tested.

White Box Testing - used to test each one of those functions behaviour is tested.

Gray Box Testing - Used to execute tests, risks and assessment methods.

Debugging

debugging is the process of fixing a bug in the software. In other words, it refers to identifying, analyzing and removing errors. This activity begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software. It is considered to be an extremely complex and tedious task because errors need to be resolved at all stages of debugging.

Debugging Process: Steps involved in debugging are:

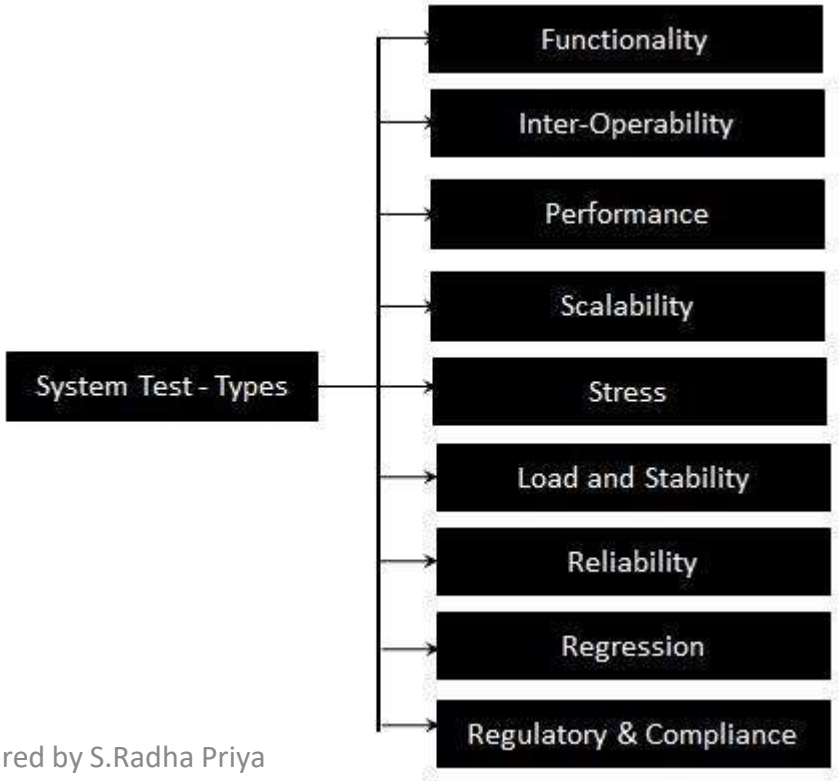
- Problem identification and report preparation.
- Assigning the report to software engineer to the defect to verify that it is genuine.
- Defect Analysis using modeling, documentations, finding and testing candidate flaws, etc.
- Defect Resolution by making required changes to the system.
- Validation of corrections.

Debugging Strategies:

- Study the system for the larger duration in order to understand the system. It helps debugger to construct different representations of systems to be debugging depends on the need. Study of the system is also done actively to find recent changes made to the software.
- Backwards analysis of the problem which involves tracing the program backward from the location of failure message in order to identify the region of faulty code. A detailed study of the region is conducting to find the cause of defects.
- Forward analysis of the program involves tracing the program forwards using breakpoints or print statements at different points in the program and studying the results. The region where the wrong outputs are obtained is the region that needs to be focused to find the defect.
- Using the past experience of the software debug the software with similar problems in nature. The success of this approach depends on the expertise of the debugger.

SYSTEM TESTING

- System Testing (ST) is a black box testing technique performed to evaluate the complete system the system's compliance against specified requirements. In System testing, the functionalities of the system are tested from an end-to-end perspective. System Testing is usually carried out by a team that is independent of the development team in order to measure the quality of the system unbiased. It includes both functional and Non-Functional testing.



FORMAL VERIFICATION

- Formal verification is the process of proving or disproving properties using formal methods (i.e., mathematically precise, algorithmic methods). A formal proof of a property provides a guarantee that no simulation of the system considered will violate the property. This eliminates the need for writing additional test cases to check the property.

- Properties for formal verification are represented in a mathematically precise, machine-readable language for which there are formal semantics.

- Assertions are the preferred language for writing such properties.

- Adding assertions will help verify the design feature sets and also obtain coverage for assessing IP quality.

- Formal Verification can be applied at block level to eliminate functional bugs early in the design cycle and match block specifications.

- It is difficult to assess the need for additional patterns for exhaustive verification if coverage goals are not integrated as part of formal verification.

Advantages of Formal Verification

- Reduce vector set for either dynamic or random simulations. Formal verification is vectorless and checks assumptions made by designers
- Smaller vector set for corner cases are easier to detect at block level and bugs can be fixed prior to SoC integration
- Automatic generation of testbench
- Bugs found at Post layout due to incorrect timing constraints are detected early. Example: Multi cycle paths and False Paths
- Validates constraints from designers
  - Synthesis constraints are input to Formal Verification
  - Helps in any invalid timing constraints for IP by the designers
  - Helps in reducing post layout debug time when IP is integrated in SoC

SOFTWARE MAINTENANCE

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

- **Market Conditions** - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
- **Client Requirements** - Over the time, customer may ask for new features or functions in the software.
- **Host Modifications** - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

Types of maintenance

Following are some types of maintenance based on their characteristics:

- **Corrective Maintenance** - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.
- **Adaptive Maintenance** - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.
- **Perfective Maintenance** - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.
- **Preventive Maintenance** - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Real-world factors affecting Maintenance Cost

- The standard age of any software is considered up to 10 to 15 years.
- Older softwares, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced softwares on modern hardware.
- As technology advances, it becomes costly to maintain old software.
- Most maintenance engineers are newbie and use trial and error method to rectify problem.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
- Changes are often left undocumented which may cause more conflicts in future.

What is Software Configuration Management?

Software Configuration Management(SCM) is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle. The primary goal is to increase productivity with minimal mistakes. SCM is part of cross-disciplinary field of configuration management and it can accurately determine who made which revision.

Tasks in SCM process

- Configuration Identification
- Baselines
- Change Control
- Configuration Status Accounting
- Configuration Audits and Reviews

Configuration Identification:

- Configuration identification is a method of determining the scope of the software system. With the help of this step, you can manage or control something even if you don't know what it is. It is a description that contains the CSCI type (Computer Software Configuration Item), a project identifier and version information.

Baseline:

- A baseline is a formally accepted version of a software configuration item. It is designated and fixed at a specific time while conducting the SCM process. It can only be changed through formal change control procedures.

Activities during this process:

- Facilitate construction of various versions of an application
- Defining and determining mechanisms for managing various versions of these work products
- The functional baseline corresponds to the reviewed system requirements
- Widely used baselines include functional, developmental, and product baselines
- In simple words, baseline means ready for release.

Change Control:

- Change control is a procedural method which ensures quality and consistency when changes are made in the configuration object. In this step, the change request is submitted to software configuration manager.
- Activities during this process:
- Control ad-hoc change to build stable software development environment. Changes are committed to the repository
- The request will be checked based on the technical merit, possible side effects and overall impact on other configuration objects.
- It manages changes and making configuration items available during the software lifecycle

Configuration Status Accounting:

- Configuration status accounting tracks each release during the SCM process. This stage involves tracking what each version has and the changes that lead to this version.

Activities during this process:

- Keeps a record of all the changes made to the previous baseline to reach a new baseline
- Identify all items to define the software configuration
- Monitor status of change requests
- Complete listing of all changes since the last baseline
- Allows tracking of progress to next baseline
- Allows to check previous releases/versions to be extracted for testing

Configuration Audits and Reviews:

- Software Configuration audits verify that all the software product satisfies the baseline needs. It ensures that what is built is what is delivered.
- **Activities during this process:**
- Configuration auditing is conducted by auditors by checking that defined processes are being followed and ensuring that the SCM goals are satisfied.
- To verify compliance with configuration control standards. auditing and reporting the changes made
- SCM audits also ensure that traceability is maintained during the process.
- Ensures that changes made to a baseline comply with the configuration status reports
- Validation of completeness and consistency