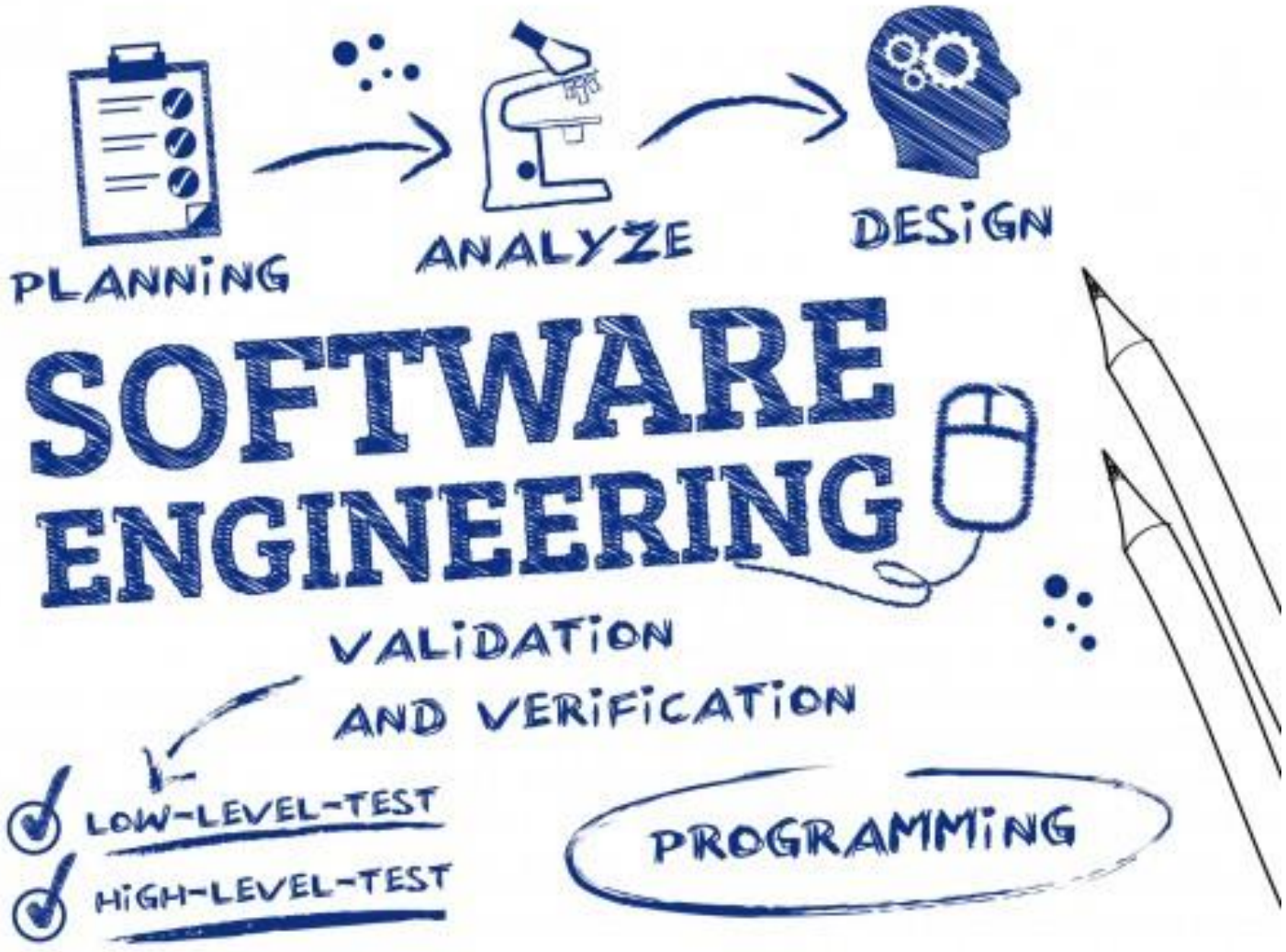


Prof. Tirup Parmar

S.Y. B.Sc(IT) – Sem4

Software Engineering Notes - 2017

Mumbai University

Course Code: USIT404

Syllabus

Unit	Details
4	**Verification and Validation:** Planning Verification and Validation, Software Inspections, Automated Static Analysis, Verification and Formal Methods. **Software Testing:** System Testing, Component Testing, Test Case Design, Test Automation. **Software Measurement**: Size-Oriented Metrics, Function-Oriented Metrics, Extended Function Point Metrics **Software Cost Estimation:** Software Productivity, Estimation Techniques, Algorithmic Cost Modelling, Project Duration and Staffing
5	**Process Improvement:** Process and product quality, Process Classification, Process Measurement, Process Analysis and Modeling, Process Change, The CMMI Process Improvement Framework. **Service Oriented Software Engineering:** Services as reusable components, Service Engineering, Software Development with Services. **Software reuse:** The reuse landscape, Application frameworks, Software product lines, COTS product reuse. **Distributed software engineering**: Distributed systems issues, Client–server computing, Architectural patterns for distributed systems, Software as a service

UNIT IV

Verification and Validation

Verification	Validation
Are we building the system right?	Are we building the right system?
Verification is the process of evaluating products of a development phase to find out whether they meet the specified requirements.	**Validation** is the process of evaluating software at the end of the development process to determine whether software meets the customer expectations and requirements.
The objective of Verification is to make sure that the product being develop is as per the requirements and design specifications.	The objective of Validation is to make sure that the product actually meet up the user's requirements, and check whether the specifications were correct in the first place.
Following activities are involved in **Verification**: Reviews, Meetings and Inspections.	Following activities are involved in **Validation**: Testing like black box testing, white box testing, gray box testing etc.
Verification is carried out by QA team to check whether implementation software is as per specification document or not.	**Validation** is carried out by testing team.
Execution of code is not comes under **Verification**.	Execution of code is comes under **Validation**.
Verification process explains whether the outputs are according to inputs or not.	**Validation** process describes whether the software is accepted by the user or not.
Verification is carried out before the Validation.	**Validation** activity is carried out just after the Verification.
Following items are evaluated during **Verification**: Plans, Requirement Specifications, Design Specifications, Code, Test Cases etc,	Following item is evaluated during **Validation**: Actual product or Software under test.

Cost of errors caught in **Verification**is less than errors found in Validation.	Cost of errors caught in **Validation** is more than errors found in Verification.
It is basically manually checking the of documents and files like requirement specifications etc.	It is basically checking of developed program based on the requirement specifications documents & files.

Objectives

- To introduce software verification and validation and to discuss the distinction between them
- To describe the program inspection process and its role in V & V
- To explain static analysis as a verification technique
- To describe the Cleanroom software development process

Verification vs validation

- Verification:
  "Are we building the product right".
- The software should conform to its specification.
- Validation:
  "Are we building the right product".
- The software should do what the user really requires.

The V & V process

- Is a whole life-cycle process - V & V must be applied at each stage in the software process.
- Has two principal objectives
  - The discovery of defects in a system;
  - The assessment of whether or not the system is useful and useable in an operational situation.

V& V goals

- Verification and validation should establish confidence that the software is fit for purpose.
- This does NOT mean completely free of defects.
- Rather, it must be good enough for its intended use and the type of use will determine the degree of confidence that is needed.

V & V confidence

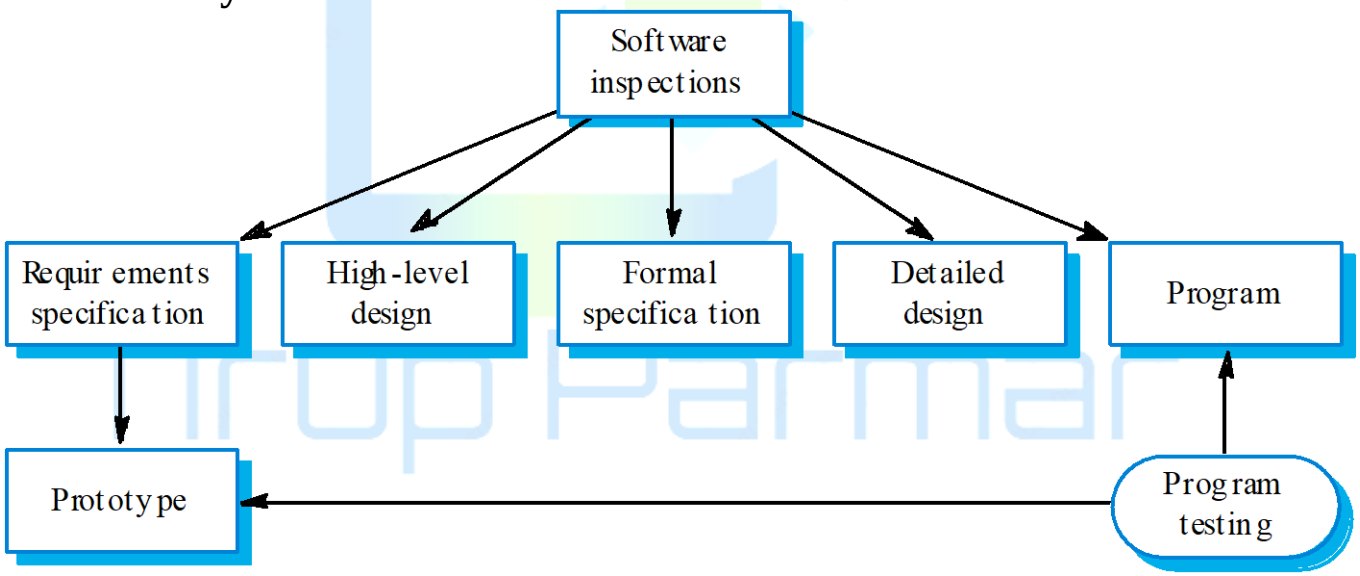
- Depends on system's purpose, user expectations and marketing environment

- Software function
  - The level of confidence depends on how critical the software is to an organisation.
- User expectations
  - Users may have low expectations of certain kinds of software.
- Marketing environment
  - Getting a product to market early may be more important than finding defects in the program.

Static and dynamic verification

- Software inspections. Concerned with analysis of the static system representation to discover problems *(*static verification)
  - May be supplement by tool-based document and code analysis
- Software testing. Concerned with exercising and observing product behaviour (dynamic verification)
  - The system is executed with test data and its operational behaviour is observed

Static and dynamic V&V



Program testing

- Can reveal the presence of errors NOT their absence.
- The only validation technique for non-functional requirements as the software has to be executed to see how it behaves.
- Should be used in conjunction with static verification to provide full V&V coverage.

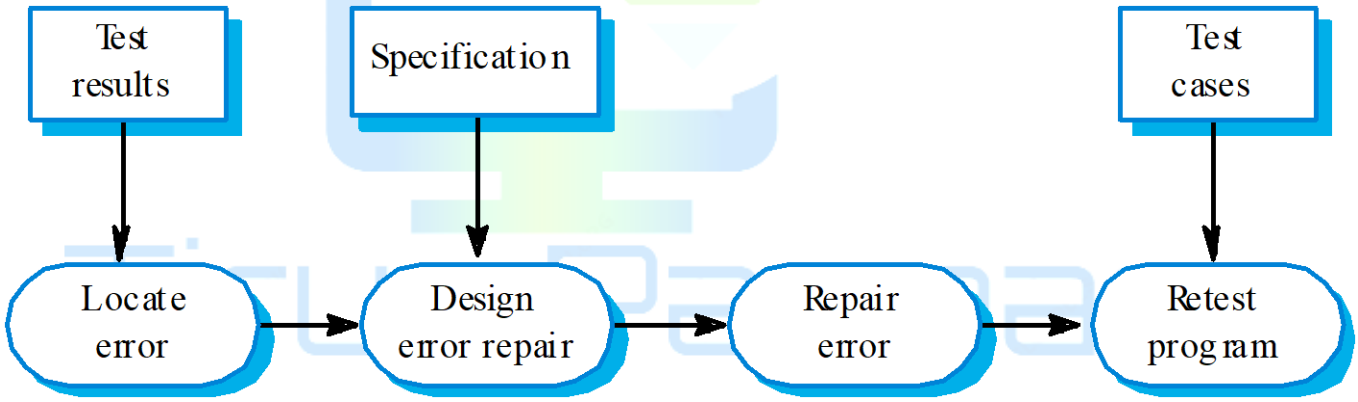
Types of testing

- **Defect testing**
  - Tests designed to discover system defects.
  - A successful defect test is one which reveals the presence of defects in a system.
- **Validation testing**
  - Intended to show that the software meets its requirements.
  - A successful test is one that shows that a requirements has been properly implemented.

Testing and debugging

- Defect testing and debugging are distinct processes.
- Verification and validation is concerned with establishing the existence of defects in a program.
- Debugging is concerned with locating and repairing these errors.
- Debugging involves formulating a hypothesis about program behaviour then testing these hypotheses to find the system error.

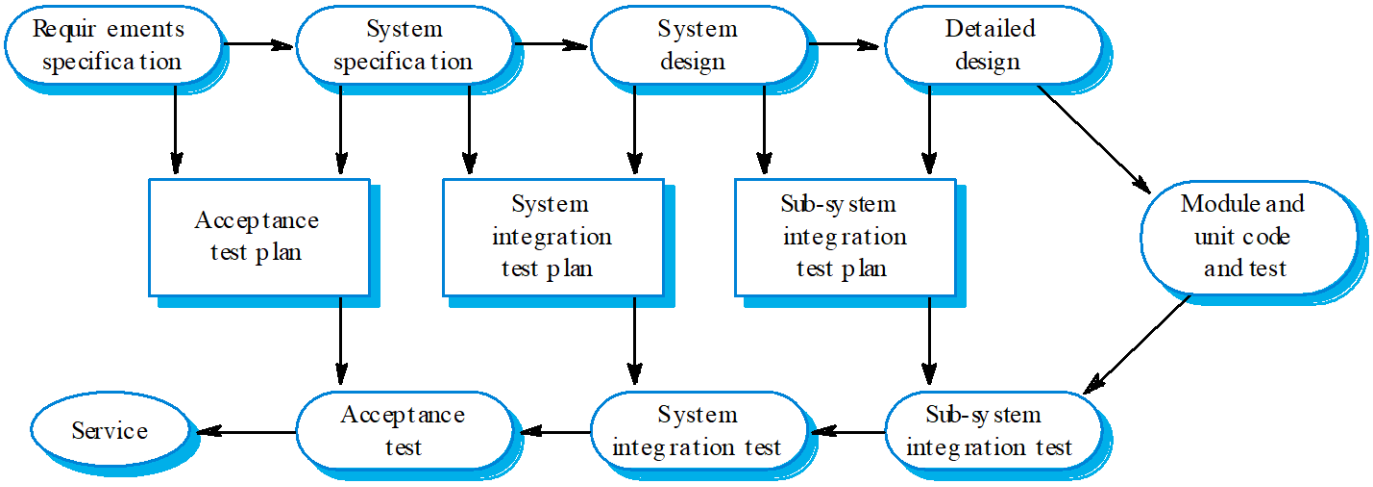
The debugging process



V & V planning

- Careful planning is required to get the most out of testing and inspection processes.
- Planning should start early in the development process.
- The plan should identify the balance between static verification and testing.
- Test planning is about defining standards for the testing process rather than describing product tests.

The V-model of development



The structure of a software test plan

- The testing process.
- Requirements traceability.
- Tested items.
- Testing schedule.
- Test recording procedures.
- Hardware and software requirements.
- Constraints.

The software test plan

The testing process
A description of the major phases of the testing process. These might be as described earlier in this chapter.

Requirements traceability
Users are most interested in the system meeting its requirements and testing should be planned so that all requirements are individually tested.

Tested items
The products of the software process that are to be tested should be specified.

Testing schedule
An overall testing schedule and resource allocation for this schedule. This, obviously, is linked to the more general project development schedule.

Test recording procedures
It is not enough simply to run tests. The results of the tests must be systematically recorded. It must be possible to audit the testing process to check that it been carried out correctly.

Hardware and software requirements
This section should set out software tools required and estimated hardware utilisation.

Constraints
Constraints affecting the testing process such as staff shortages should be anticipated in this section.

Software inspections

- These involve people examining the source representation with the aim of discovering anomalies and defects.
- Inspections do not require execution of a system so may be used before implementation.
- They may be applied to any representation of the system (requirements, design,configuration data, test data, etc.).
- They have been shown to be an effective technique for discovering program errors.

Inspection success

- Many different defects may be discovered in a single inspection. In testing, one defect, may mask another so several executions are required.
- Reuses domain and programming knowledge so reviewers are likely to have seen the types of error that commonly arise.

Inspections and testing

- Inspections and testing are complementary and not opposing verification techniques.
- Both should be used during the V & V process.
- Inspections can check conformance with a specification but not conformance with the customer's real requirements.
- Inspections cannot check non-functional characteristics such as performance, usability, etc.

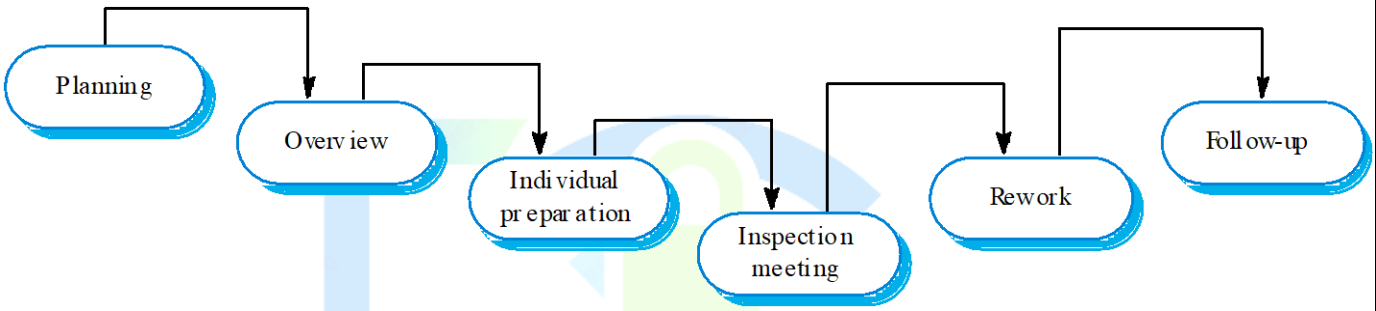
Program inspections

- Formalised approach to document reviews
- Intended explicitly for defect detection (not correction).
- Defects may be logical errors, anomalies in the code that might indicate an erroneous condition (e.g. an uninitialised variable) or non-compliance with standards.

Inspection pre-conditions

- A precise specification must be available.
- Team members must be familiar with the organisation standards.
- Syntactically correct code or other system representations must be available.
- An error checklist should be prepared.
- Management must accept that inspection will increase costs early in the software process.
- Management should not use inspections for staff appraisal i.e. finding out who makes mistakes.

The inspection process



Inspection procedure

- System overview presented to inspection team.
- Code and associated documents are distributed to inspection team in advance.
- Inspection takes place and discovered errors are noted.
- Modifications are made to repair discovered errors.
- Re-inspection may or may not be required.

Inspection roles

Author or owner	The programmer or designer responsible for producing the program or document. Responsible for fixing defects discovered during the inspection process.
Inspector	Finds errors, omissions and inconsistencies in programs and documents. May also identify broader issues that are outside the scope of the inspection team.
Reader	Presents the code or document at an inspection meeting.
Scribe	Records the results of the inspection meeting.

Chairman or moderator	Manages the process and facilitates the inspection. Reports process results to the Chief moderator.
Chief moderator	Responsible for inspection process improvements, checklist updating, standards development etc.

Inspection checklists

- Checklist of common errors should be used to drive the inspection.
- Error checklists are programming language dependent and reflect the characteristic errors that are likely to arise in the language.
- In general, the 'weaker' the type checking, the larger the checklist.
- Examples: Initialisation, Constant naming, loop termination, array bounds, etc.

Inspection checks 1

Data faults	Are all program variables initialised before their values are used? Have all constants been named? Should the upper bound of arrays be equal to the size of the array or Size -1? If character strings are used, is a de limiter explicitly assigned? Is there any possibility of buffer overflow?
Control faults	For each conditional statement, is the condition correct? Is each loop certain to terminate? Are compound statements correctly bracketed? In case statements, are all possible cases accounted for? If a break is required after each case in case statements, has it been included?
Input/output faults	Are all input variables used? Are all output variables assigned a value before they are output? Can unexpected inputs cause corruption?

Inspection checks 2

Interface faults	Do all function and method calls have the correct number of parameters? Do formal and actual parameter types match? Are the parameters in the right order? If components access shared memory, do they have the same model of the shared memory structure?
Storage management faults	If a linked structure is modified, have all links been correctly reassigned? If dynamic storage is used, has space been allocated correctly? Is space explicitly de-allocated after it is no longer required?
Exception management faults	Have all possible error conditions been taken into account?

Inspection rate

- 500 statements/hour during overview.
- 125 source statement/hour during individual preparation.
- 90-125 statements/hour can be inspected.
- Inspection is therefore an expensive process.
- Inspecting 500 lines costs about 40 man/hours effort - about £2800 at UK rates.

Automated static analysis

- Static analysers are software tools for source text processing.
- They parse the program text and try to discover potentially erroneous conditions and bring these to the attention of the V & V team.
- They are very effective as an aid to inspections - they are a supplement to but not a replacement for inspections.

Static analysis checks

Fault class	**Static analysis check**
Data faults	Variables used before initialisation Variables declared but never used Variables assigned twice but never used between assignments Possible array bound violations Undeclared variables

Control faults	Unreachable code Unconditional branches into loops
Input/output faults	Variables output twice with no intervening assignment
Interface faults	Parameter type mismatches Parameter number mismatches Non-usage of the results of functions Uncalled functions and procedures
Storage management faults	Unassigned pointers Pointer arithmetic

Stages of static analysis

- Control flow analysis. Checks for loops with multiple exit or entry points, finds unreachable code, etc.
- Data use analysis. Detects uninitialized variables, variables written twice without an intervening assignment, variables which are declared but never used, etc.
- Interface analysis. Checks the consistency of routine and procedure declarations and their use.
- Information flow analysis. Identifies the dependencies of output variables. Does not detect anomalies itself but highlights information for code inspection or review
- Path analysis. Identifies paths through the program and sets out the statements executed in that path. Again, potentially useful in the review process
- Both these stages generate vast amounts of information. They must be used with care.

LINT static analysis

```
138% more lint_ex.c
#include <stdio.h>
printarray (Anarray)
 int Anarray;
{  printf("%d",Anarray);  }

main ()
{
 int Anarray[5]; int i; char c;
 printarray (Anarray, i, c);
 printarray (Anarray) ;
}

139% cc lint_ex.c
140% lint lint_ex.c
```

```
lint_ex.c(10): warning: c may be used before set
lint_ex.c(10): warning: i may be used before set
printarray: variable # of args. lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) :: lint_ex.c(11)
printf returns value which is always ignored
```

Use of static analysis

- Particularly valuable when a language such as C is used which has weak typing and hence many errors are undetected by the compiler,
- Less cost-effective for languages like Java that have strong type checking and can therefore detect many errors during compilation.

Verification and formal methods

- Formal methods can be used when a mathematical specification of the system is produced.
- They are the ultimate static verification technique.
- They involve detailed mathematical analysis of the specification and may develop formal arguments that a program conforms to its mathematical specification.

Arguments for formal methods

- Producing a mathematical specification requires a detailed analysis of the requirements and this is likely to uncover errors.
- They can detect implementation errors before testing when the program is analyzed alongside the specification.

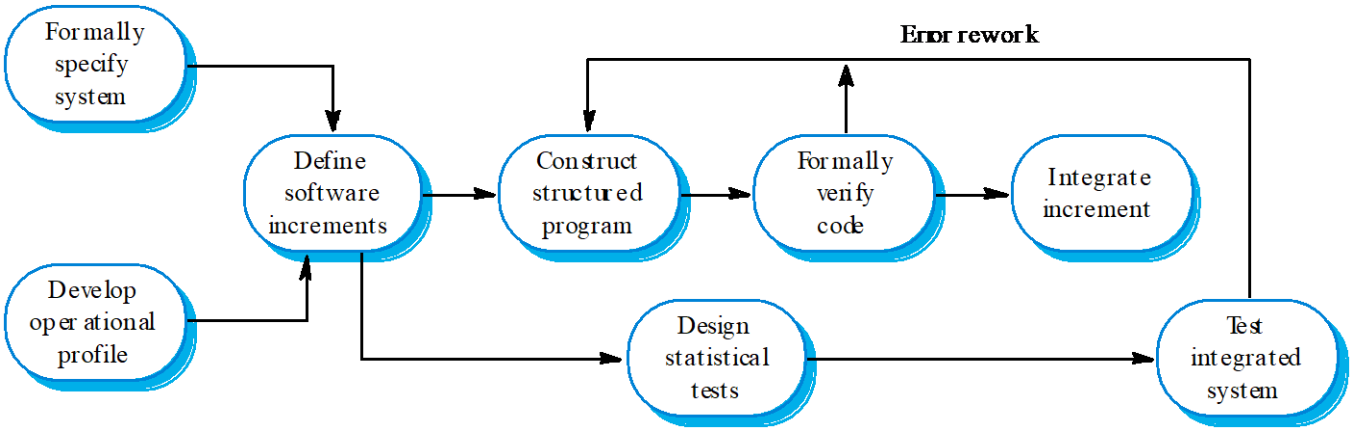
Arguments against formal methods

- Require specialised notations that cannot be understood by domain experts.
- Very expensive to develop a specification and even more expensive to show that a program meets that specification.
- It may be possible to reach the same level of confidence in a program more cheaply using other V & V techniques.

Cleanroom software development

- The name is derived from the 'Cleanroom' process in semiconductor fabrication. The philosophy is defect avoidance rather than defect removal.
- This software development process is based on:
  - Incremental development;
  - Formal specification;
  - Static verification using correctness arguments;
  - Statistical testing to determine program reliability.

The Cleanroom process



Cleanroom process characteristics

- Formal specification using a state transition model.
- Incremental development where the customer prioritises increments.
- Structured programming - limited control and abstraction constructs are used in the program.
- Static verification using rigorous inspections.
- Statistical testing of the system.

Formal specification and inspections

- The state based model is a system specification and the inspection process checks the program against this model.
- The programming approach is defined so that the correspondence between the model and the system is clear.
- Mathematical arguments (not proofs) are used to increase confidence in the inspection process.

Cleanroom process teams

- **Specification team.** Responsible for developing and maintaining the system specification.
- **Development team.** Responsible for developing and verifying the software. The software is NOT executed or even compiled during this process.
- **Certification team.** Responsible for developing a set of statistical tests to exercise the software after development. Reliability growth models used to determine when reliability is acceptable.

Cleanroom process evaluation

- The results of using the Cleanroom process have been very impressive with few discovered faults in delivered systems.
- Independent assessment shows that the process is no more expensive than other approaches.
- There were fewer errors than in a 'traditional' development process.
- However, the process is not widely used. It is not clear how this approach can be transferred to an environment with less skilled or less motivated software engineers.

Key points

- Verification and validation are not the same thing. Verification shows conformance with specification; validation shows that the program meets the customer's needs.
- Test plans should be drawn up to guide the testing process.
- Static verification techniques involve examination and analysis of the program for error detection.
- Program inspections are very effective in discovering errors.
- Program code in inspections is systematically checked by a small team to locate software faults.
- Static analysis tools can discover program anomalies which may be an indication of faults in the code.
- The Cleanroom development process depends on incremental development, static verification and statistical testing.

Software Testing

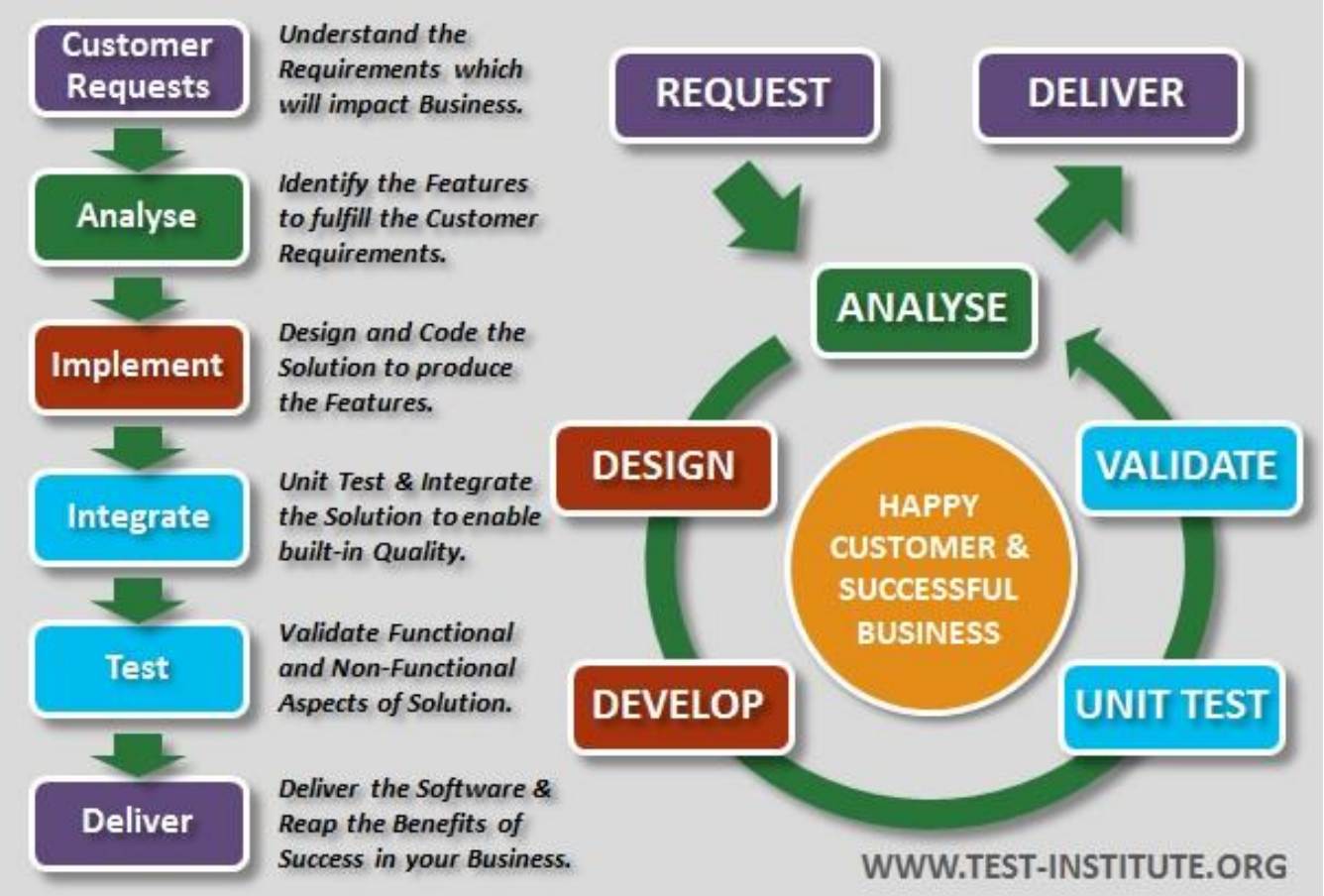


Fig. Software Testing Methodology in Software Engineering

- Software testing is nothing but an art of investigating software to ensure that its quality under test is in line with the requirement of the client.
- Software testing is carried out in a systematic manner with the intent of finding defects in a system. It is required for evaluating the system.
- As the technology is advancing we see that everything is getting digitized. You can access your bank online, you can shop from the comfort of your home, and the options are endless.
- Have you ever wondered what would happen if these systems turn out to be defective?One small defect can cause a lot of financial loss.
- It is for this reason that software testing is now emerging as a very powerful field in IT.
- Although like other products software never suffers from any kind of wear or tear or corrosion but yes, design errors can definitely make your life difficult if they go undetected.
- Regular testing ensures that the software is developed as per the requirement of the client.
- However, if the software is shipped with bugs embedded in it, you never know when they can create a problem and then it will be very difficult to rectify defect because scanning hundreds and thousands of lines of code and fixing a bug is not an easy task.

- You never know that while fixing one bug you may introduce another bug unknowingly in the system.
- Software testing is now a very significant and integral part of software development. Ideally, it is best to introduce software testing in every phase of software development life cycle. Actually a majority of software development time is now spent on testing.

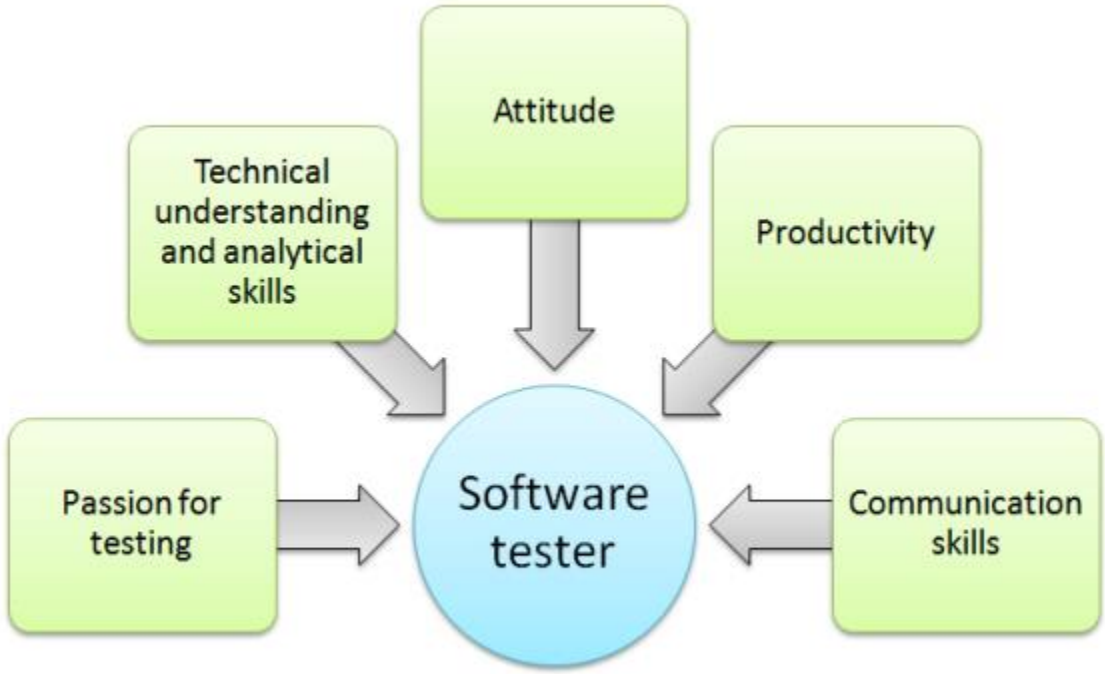
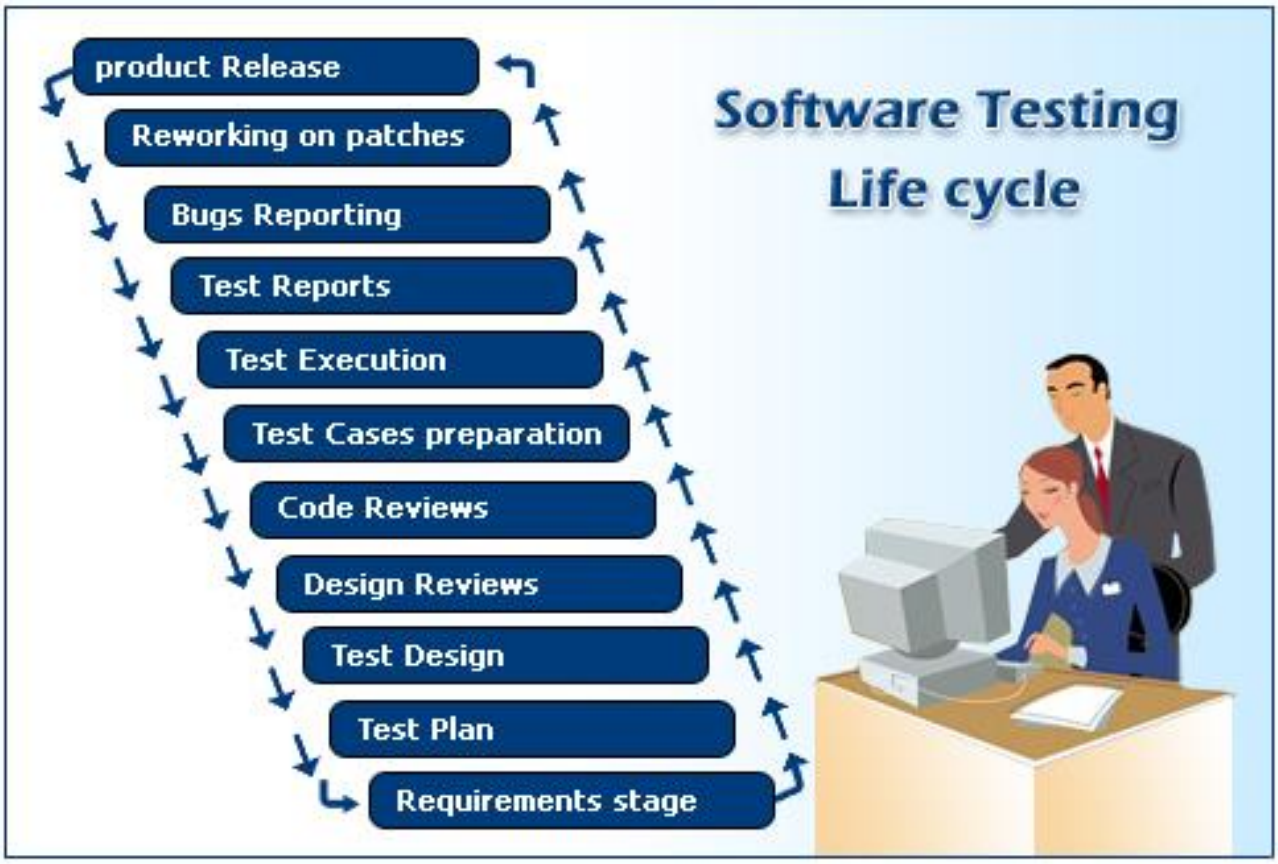


Fig. Introduction To Software Testing

So, to summarize we can say that:

1. Software testing is required to check the reliability of the software
2. Software testing ensures that the system is free from any bug that can cause any kind of failure
3. Software testing ensures that the product is in line with the requirement of the client
4. It is required to make sure that the final product is user friendly
5. At the end software is developed by a team of human developers all having different viewpoints and approach. Even the smartest person has the tendency to make an error. It is not possible to create software with zero defects without incorporating software testing in the development cycle.
6. No matter how well the software design looks on paper, once the development starts and you start testing the product you will definitely find lots of defects in the design.



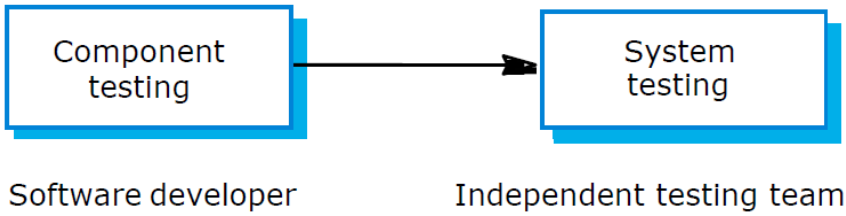
Objectives

- To discuss the distinctions between validation testing and defect testing
- To describe the principles of system and component testing
- To describe strategies for generating system test cases
- To understand the essential characteristics of tool used for test automation

The testing process

- **Component testing**
  - Testing of individual program components;
  - Usually the responsibility of the component developer (except sometimes for critical systems);
  - Tests are derived from the developer's experience.
- **System testing**
  - Testing of groups of components integrated to create a system or sub-system;
  - The responsibility of an independent testing team;
  - Tests are based on a system specification.

Testing phases



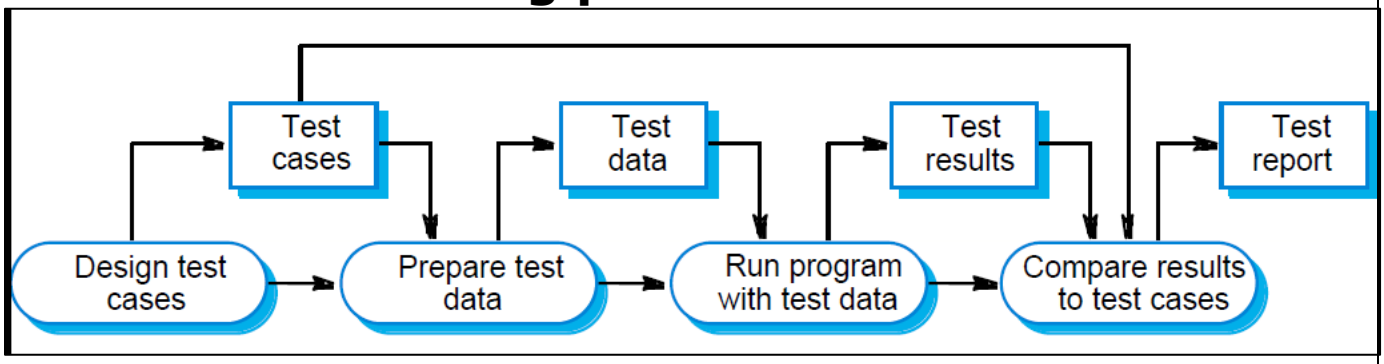
Defect testing

- The goal of defect testing is to discover defects in programs.
- A *successful*defect test is a test which causes a program to behave in an anomalous way.
  - This is the opposite of the view taken by validation testing!
- Tests show the presence not the absence of defects.

Testing process goals

- **Defect testing**
  - To discover faults or defects in the software where its behaviouris incorrect or not in conformance with its specification;
  - A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.
- **Validation testing**
  - To demonstrate to the developer and the system customer that the software meets its requirements;
  - A successful test shows that the system operates as intended.
- We start with defect testing and perform validation testing later in the process.

The software testing process



Note: For many(if not most) people, the term "test case" includes the test data as well.

Testing policies

- Only exhaustive testing could show a program is free from defects.
  - However, exhaustive testing is impossible.

- Testing policies define the approach to be used in selecting system tests. Examples:
  - All functions accessed through menus should be tested;
  - Combinations of functions accessed through the same menu should be tested;
  - Where user input is required, all functions must be tested both with correct input and with incorrect input.