# UNIT-5

**Testing strategies:** A strategic approach to software testing, Test strategies for conventional software, Test strategies for object-oriented software, Validation testing, System testing, Art of debugging.

## ➢ A strategic approach to software testing

Software testing is the process of evaluating a software application to identify if it meets specified requirements and to identify any defects.

Software Testing is a type of investigation to find out if there is any default or error present in the software so that the errors can be reduced or removed to increase the quality of the software and to check whether it full fills the specifies requirements or not.

- In the literature of software engineering various testing strategies to implement the testing are defined.
- All the strategies give a testing template.

**Following are the characteristic that process the testing templates:**

- The developer should conduct the successful technical reviews to perform the testing successful.
- Testing starts with the component level and work from outside toward the integration of the whole computer based system.
- Different testing techniques are suitable at different point in time.
- Testing is organized by the developer of the software and by an independent test group.
- Debugging and testing are different activities, then also the debugging should be accommodated in any strategy of testing.

- Difference between Verification and Validation

| Verification | Validation |
|---|---|
| Verification is the process to find whether the software meets the specified requirements for particular phase. | The validation process is checked whether the software meets requirements and expectation of the customer. |
| It estimates an intermediate product. | It estimates the final product. |
| The objectives of verification is to check whether software is constructed according to requirement and design specification. | The objectives of the validation is to check whether the specifications are correct and satisfy the business need. |

| | |
|---|---|
| It describes whether the outputs are as per the inputs or not. | It explains whether they are accepted by the user or not. |
| Verification is done before the validation. | It is done after the verification. |
| Plans, requirement, specification, code are evaluated during the verifications. | Actual product or software is tested under validation. |
| It manually checks the files and document. | It is a computer software or developed program based checking of files and document. |

## A software testing strategy for conventional software architecture

A strategy of software testing is shown in the context of spiral.

**Following figure shows the testing strategy:**

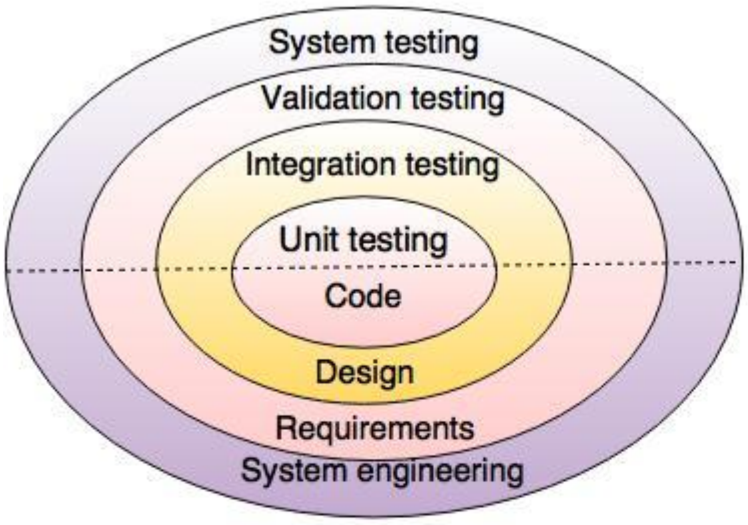


**Fig. - Testing Strategy**

**Unit testing**
Unit testing starts at the centre and each unit is implemented in source code.

**Integration testing**
An integration testing focuses on the construction and design of the software.

**Validation testing**
Check all the requirements like functional, behavioral and performance requirement are validate against the construction software.

**System testing**
System testing confirms all system elements and performance are tested entirely.

**Testing strategy for procedural point of view**

As per the procedural point of view the testing includes following steps.

1) Unit testing
2) Integration testing
3) High-order tests
4) Validation testing

**These steps are shown in following figure:**

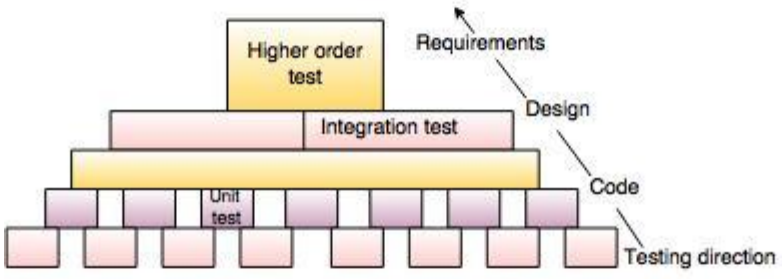
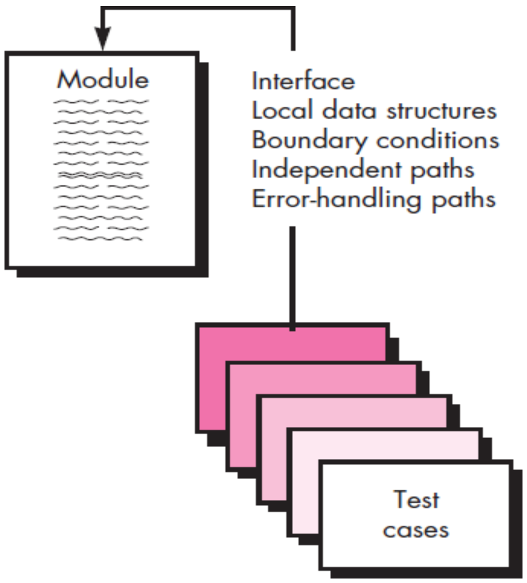


**Fig.- Steps of software testing**

## ➢ TEST STRATEGIES FOR CONVENTIONAL SOFTWARE

**Unit Testing**

The unit test focuses on the internal processing logic and data structures within the boundaries of a component. This type of testingcan be conducted in parallel for multiple components.



**Unit-test considerations:-**

1. The module interface is tested to ensure proper informationflows (into and out).
2. Local data structures are examined to ensure temporary datastore during execution.
3. All independent paths are exercised to ensure that all statements in a module have been executed at least once.

4. Boundary conditions are tested to ensure that the module operates properly at boundaries. Software often fails at its boundaries.
5. All error-handling paths are tested.

If data do not enter and exit properly, all other tests are controversial. Among the potential errors that should be tested whenerror handling is evaluated are:

(1) Error description is unintelligible,

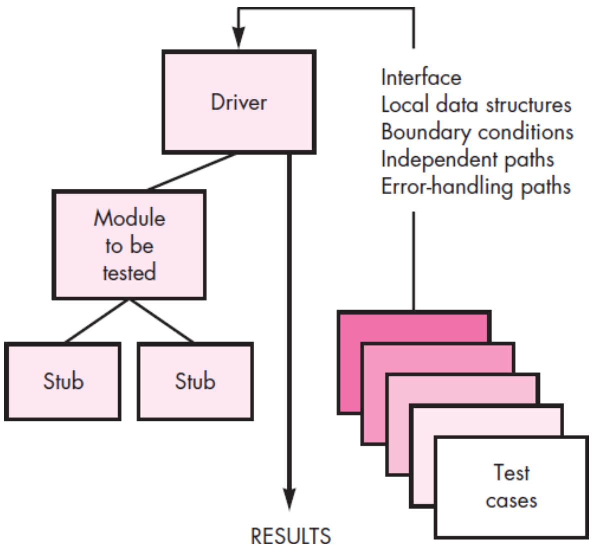
(2) Error noted does not correspond to error encountered,

(3) Error condition causes system intervention prior to error handling,

(4) exception-condition processing is incorrect,

(5) Error description does not provide enough information to assist in the location of the cause of the error.

**Unit-test procedures:-** The design of unit tests can occur before coding begins or after source code has been generate. "*Because a component is not a stand-alone program, driver and/or stub software must often be developed for each unit test*".



**Driver** is nothing more than a "main program" that accepts test casedata, passes such data to the component (to be tested), and prints relevant results.

**Stubs** serve to replace modules that are subordinate (invoked by) the component to be tested. A stub may do minimal data manipulation, prints verification of entry, and returns control to the module undergoing testing.

Drivers and stubs represent testing "overhead." That is, both are software that must be written (formal design is not commonly applied) but that is not delivered with the final software product.

## Integration Testing

Data can be lost across an interface; one component can have an inadvertent, adverse effect on another; sub functions, whencombined, may not produce the desired major function. The objective of Integration testing is to take unit-tested components and build a program structure that has been dictated by design. The program is constructed and tested in small increments,where errors are easier to isolate and correct. A number of different incremental integration strategies are:-

a) **Top-down integration testing** is an incremental approach to construction of the software architecture. Modules are integrated by moving downward through the control hierarchy. Modules subordinate to the main control moduleare incorporated into the structure in either a depth-first or breadth-first manner. The integration process is performed in a series of five steps:

1. The main control module is used as a test driver and stubs are substituted for all components directly subordinateto the main control module.
2. Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced oneat a time with actual components.
3. Tests are conducted as each component is integrated.
4. On completion of each set of tests, another stub is replaced with the real component.
5. Regression testing may be conducted to ensure that new errors have not been introduced.

The top-down integration strategy verifies major control or decision points early in the test process. Stubs replace low-level modules at the beginning of top-down testing. Therefore, no significant data can flow upward in the program structure. Asa tester, you are left with three choices:

(1) Delay many tests until stubs are replaced with actual modules,

(2) Develop stubs that perform limited functions that simulate the actual module, or

(3) Integrate the software from the bottom of the hierarchy upward.

b) **Bottom-up integration**- Begins construction and testing with components at the lowest levels in the programstructure. Because components are integrated from the bottom up, the functionality provided by components subordinate to a given level is always available and the **need for stubs is eliminated**. A bottom-up integration strategy may be implemented with the following steps:

1. Low-level components are combined into clusters (sometimes called builds) that perform a specific software sub

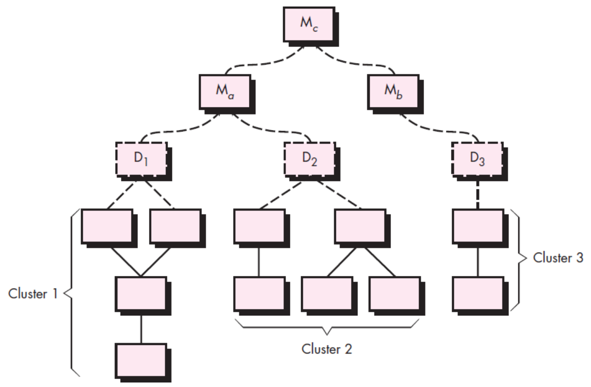
function.

2. A driver (a control program for testing) is written to coordinate test case input and output.

3. The cluster is tested.

4. Drivers are removed and clusters are combined moving upward in the program structure.

Integration follows the following pattern—D are drivers and M are modules. Drivers will be removed prior to integration of modules.



**Regression testing:-** Each time a new module is added as part of integration testing, the software changes. New data flow paths are established, new I/O may occur, and new control logic is invoked**. These changes may cause problems with functions that previously worked flawlessly.** Regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.

Regression testing may be conducted manually or using automated capture/playback tools. Capture/playback tools enable the software engineer to capture test cases and results for subsequent playback and comparison. The regression test suite contains three different classes of test cases:

• A representative sample of tests that will exercise all software functions.

• Additional tests that focus on software functions that are likely to be affected by the change.

• Tests that focus on the software components that have been changed.

**As integration testing proceeds, the number of regression tests can grow**

**Smoke testing:- It** is an integration testing approach that is commonly used when product software is developed. It is designed as a pacing mechanism for time-critical projects, allowing the software team to assess the project on a frequent basis. In essence, the smoke-testing approach encompasses the following activities:

1. Software components that have been translated into code are integrated into a build. A build includes all data files,libraries, reusable modules, and engineered components that are required to implement one or more product functions.

2. A series of tests is designed to expose errors that will keep the build from properly performing its function. The intentshould be to uncover "showstopper" errors that have the highest likelihood of throwing the software project behind schedule.

3. The build is integrated with other builds, and the entire product is smoke tested daily. The integration approach may betop down or bottom up.

Smoke testing provides a number of benefits when it is applied on complex, time critical software projects:

• ***Integration risk is minimized.*** Because smoke tests are conducted daily, incompatibilities and other show-stopper errors are uncovered early,

• ***The quality of the end product is improved.*** Smoke testing is likely to uncover functional errors as well as architectural and component-level design errors.

• ***Error diagnosis and correction are simplified.*** Errors uncovered during smoke testing are likely to be associated with "new software increments"—that is, the software that has just been added to the build(s) is a probable cause of a newly discovered error.

• ***Progress is easier to assess.*** With each passing day, more of the software has been integrated and more has been demonstrated to work. This improves team morale and gives managers a good indication that progress is being made.

**Strategic options:-** The major disadvantage of the top-down approach is the need for stubs and the attendant testing difficulties that can be associated with them. The major disadvantage of bottom-up integration is that "the program as an entity does not exist until the last module is added".

**Selection of an integration strategy depends upon software characteristics and, sometimes, project schedule.** Ingeneral, a combined approach or sandwich testing may be the best compromise.

As integration testing is conducted, the tester should identify critical modules. A critical module has one or more of thefollowing characteristics:

(1) Addresses several software requirements,

(2) Has a high level of control,

(3) Is complex or error prone?

(4) Has definite performance requirements.

Critical modules should be tested as early as is possible. In addition, regression tests should focus on critical module function.

**Integration test work products:-** It is documented in a Test Specification. This work product incorporates a test plan and a test procedure and becomes part of the software configuration. Program builds (groups of modules) are created to correspond to each phase. The following criteria and corresponding tests are applied for all test phases:

1. **Interface integrity.** Internal and external interfaces are tested as each module (or cluster) is incorporated into thestructure.
2. **Functional validity.** Tests designed to uncover functional errors are conducted.
3. **Information content.** Tests designed to uncover errors associated with local or global data structures are conducted.
4. **Performance.** Tests designed to verify performance bounds established during software design are conducted.

A history of actual test results, problems, or peculiarities is recorded in a Test Report that can be appended to the Test Specification.

## ➢ Software Engineering-Object Oriented Testing Strategies

The classical strategy for testing computer software begins with “testing in the small” and works outward toward “testing in the large.

In conventional applications, unit testing focuses on the smallest compilable program unit—the subprogram (e.g., module, subroutine, procedure, component). Once each of these units has been tested individually, it is integrated into a program structure while a series of regression tests are run to uncover errors due to interfacing between the modules and side effects caused by the addition of new units.

**Unit Testing in the OO Context**

When object-oriented software is considered, the concept of the unit changes. Encapsulation drives the definition of classes and objects. This means that each class and each instance of a class (object) packages attributes (data) and the operations (also known as methods or services) that manipulate these data. Rather than testing an individual module, the smallest testable unit is the encapsulated class or object. Because a class can contain a

number of different operations and a particular operation may exist as part of a number of different classes, the meaning of unit testing changes dramatically.
We can no longer test a single operation in isolation (the conventional view of unit testing) but rather as part of a class. To illustrate, consider a class hierarchy in which an operation X is defined for the superclass and is inherited by a number of subclasses. Each subclass uses operation X, but it is applied within the context of the private attributes and operations that have been defined for the subclass. Because the context in which operation X is used varies in subtle ways, it is necessary to test operation X in the context of each of the subclasses. This means that testing operation X in a vacuum (the traditional unit testing approach) is ineffective in the object-oriented context.

Class testing for OO software is the equivalent of unit testing for conventional software. Unlike unit testing of conventional software, which tends to focus on the algorithmic detail of a module and the data that flow across the module interface, class testing for OO software is driven by the operations encapsulated by the class and the state behavior of the class.

**Integration Testing in the OO Context**

Because object-oriented software does not have a hierarchical control structure, conventional top-down and bottom-up integration strategies have little meaning. In addition, integrating operations one at a time into a class (the conventional incremental integration approach) is often impossible because of the "direct and indirect interactions of the components that make up the class".

There are two different strategies for integration testing of OO systems . The first, thread-based testing, integrates the set of classes required to respond to one input or event for the system. Each thread is integrated and tested individually. Regression testing is applied to ensure that no side effects occur. The second integration approach, use-based testing, begins the construction of the system by testing those classes (called independent classes) that use very few (if any) of server classes. After the independent classes are tested, the next layer of classes, called dependent classes, that use the independent classes are tested. This sequence of testing layers of dependent lasses continues until the entire system is constructed. Unlike conventional integration, the use of drivers and stubs as replacement operations is to be avoided, when possible.

Cluster testing is one step in the integration testing of OO software. Here, a cluster of collaborating classes (determined by examining the CRC and object-relationship model) is exercised by designing test cases that attempt to uncover errors in the collaborations.

**Validation Testing in an OO Context**

At the validation or system level, the details of class connections disappear. Like conventional validation, the validation of OO software focuses on user-visible actions and user-recognizable output from the system. To assist in the derivation of validation tests, the tester should draw upon the use-cases that are part of the analysis model. The use-case provides a scenario that has a high likelihood of uncovered errors in user interaction requirements.

Conventional black-box testing methods can be used to drive validations tests. In addition, test cases may be derived from the object-behavior model and from event flow diagram created as part of OOA.

## ➢ <u>Validation Testing</u>

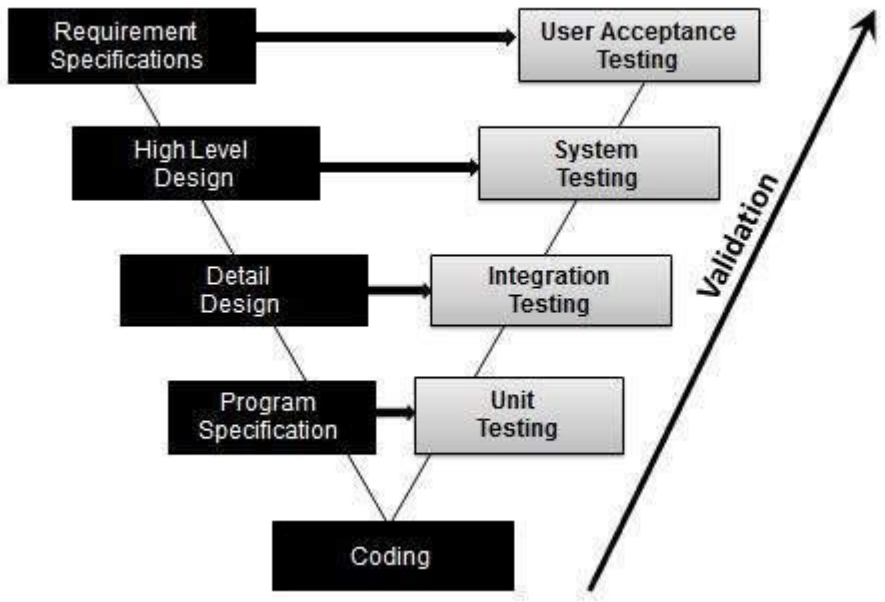
The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

It answers to the question, Are we building the right product?

Validation Testing - Workflow:

Validation testing can be best demonstrated using V-Model. The Software/product under test is evaluated during this type of testing.



Through Validation testing requirements are validated againsts/wconstructed. These are high-order tests where validation criteria must be evaluated to assure that s/w meets all functional, behavioural and performance requirements. It succeeds when the software functionsin a manner that can be reasonably expected by the customer.

1. Validation Test Criteria
2. Configuration Review
3. Alpha and Beta Testing

The validation criteria described in SRS form the basis for this testing. Here, Alpha and Betatesting is performed. Alpha testing is performed at the developers site by end users in a natural setting and with a controlled environment. Beta testing is conducted at end-user sites. It is a "live" application and environment is not controlled.
End-user records all problems and reports to developer. Developer then makes modifications and releases the product.

## ➢ <u>System Testing</u>

In system testing, s/w and other system elements are tested as a whole. This is the last high-order testing step which falls in the context of computer system engineering. Software is combined with other system elements like H/W, People, Database and the overall functioning is checked by conducting a series of tests. These tests fully exercise the computer based system. The types of tests are:

1. Recovery testing: Systems must recover from faults and resume processing within a prespecified time.

It forces the system to fail in a variety of ways and verifies that recovery is properly performed. Here the Mean Time To Repair (MTTR) is evaluated to see if it is within acceptable limits.

2. Security Testing: This verifies that protection mechanisms built into a system will protect it from improper penetrations. Tester plays the role of hacker. In reality given enough resources and time it is possible to ultimately penetrate any system. The role of system designer is to make penetration cost more than the value of the information that will be obtained.
3. Stress testing: It executes a system in a manner that demands resources in abnormal quantity, frequency or volume and tests the robustness of the system.

4. Performance Testing: This is designed to test the run-time performance of s/w within the context of an integrated system. They require both h/w and s/w instrumentation.

## ➢ The Art of Debugging

Debugging occurs as a consequence of successful testing. It is an action that results intheremoval of errors.
It is very much an art.

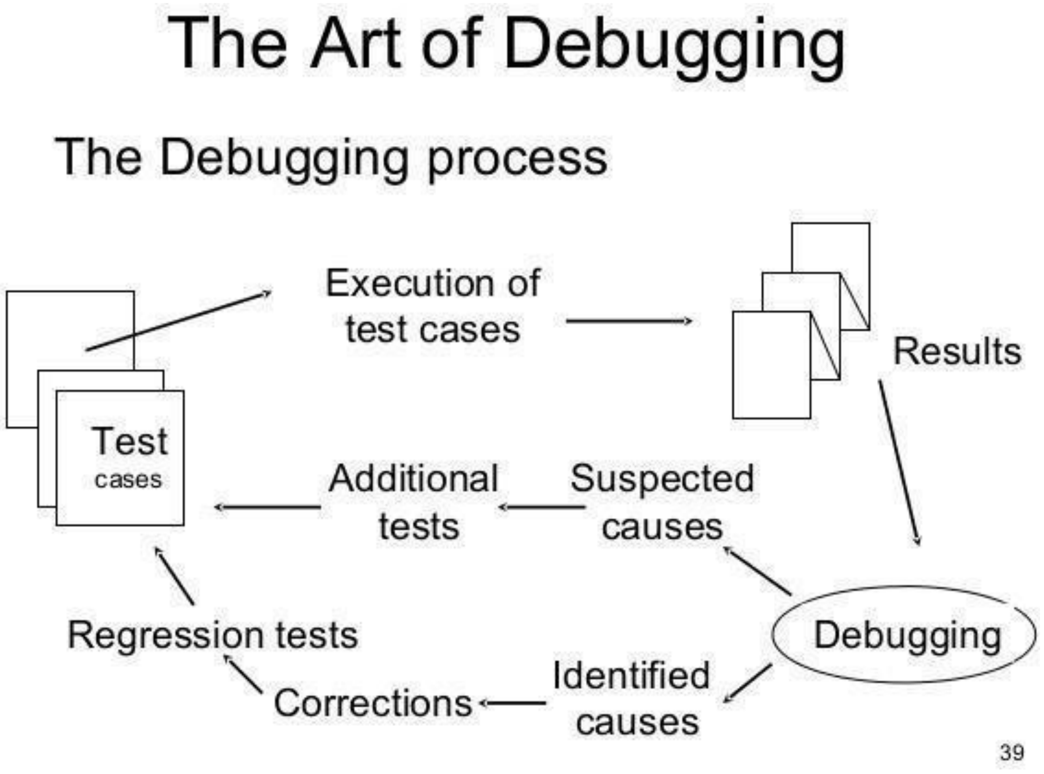


Fig: Debugging process

- Debugging has two outcomes:

1) cause will be found and corrected

2) cause will not be

**found Characteristics of bugs:**

i)Symptom and cause can be in different location

ii) Symptoms may be caused by human error or timing problems Debugging is an innate human trait. Some are good at it and some arenot.

**Debugging Strategies:**

The objective of debugging is to find and correct the cause of a software error which is realized by a combination of systematic evaluation, intuition and luck. Three strategies areproposed:

1)Brute Force Method.

2)Back Tracking

3)Cause Elimination

4) Automated Debugging

**Brute Force**: Most common and least efficient method for isolating the cause of a s/w error.

This is applied when all else fails. Memory dumps are taken, run-time traces are invoked and programis loaded with output statements. Tries to find the cause from the load of information Leads to waste of time and effort.

**Back tracking**: Common debugging approach. Useful for small programs beginning at the system where the symptom has been uncovered, the source code is traced backward until the site of the cause is found. More no. of lines implies no. of paths are un manageable.

**Cause Elimination**: Based on the concept of Binary partitioning. Data related to error occurrence are organized to isolate potential causes. A “cause hypothesis” is devised and data is used to prove or disprove it. A list of all possible causes is developed and tests are conducted to eliminate each

**Automated Debugging**: This supplements the above approaches with debugging tools that provide semi-automated support like debugging compilers, dynamic debugging aids, test case generators, mapping tools etc.