JGi JAIN DEEMED-TO-BE UNIVERSITY | SCHOOL OF COMPUTER SCIENCE AND IT

SOFTWARE ENGINEERING

UNIT V

SOFTWARE TESTING

Prof. Rahul Pawar,
Assistant Professor, School of CS & IT,
MCA Department,
Knowledge Campus

SOFTWARE QUALITY

- **Pressman's definition of "Software Quality“:**

Software Quality is *conformance to*:

- explicitly stated *functional and performance requirements,*
- explicitly documented *development standards*,
- *implicit characteristics* that are expected of all professionally developed software.

- **IEEE Definition of "Software Quality“**

1. **The degree to which a system**, component, or process **meets specified requirements**.

2. The **degree to which a system**, component, or process meets customer or **user needs or expectations**.

Problems with Software Quality

- **Software specifications** are usually ***incomplete and often inconsistent***
- There is ***tension*** **between**:
  - customer quality requirements (efficiency, reliability, etc.)
  - developer quality requirements (maintainability, reusability, etc.)
- **Some quality requirements are *hard to specify*** in an **unambiguous** way
  - **Directly measurable qualities** (e.g., errors/KLOC),
  - **Indirectly measurable qualities** (e.g., usability).

Quality Attributes

Quality attributes apply both to the product and the process.

- ***product***: delivered to the customer
- ***process:*** produces the software product
- ***resources:*** (both the product and the process require resources)
  - Underlying assumption: a quality process leads to a quality product (cf. metaphor of manufacturing lines)

Quality attributes can be external or internal

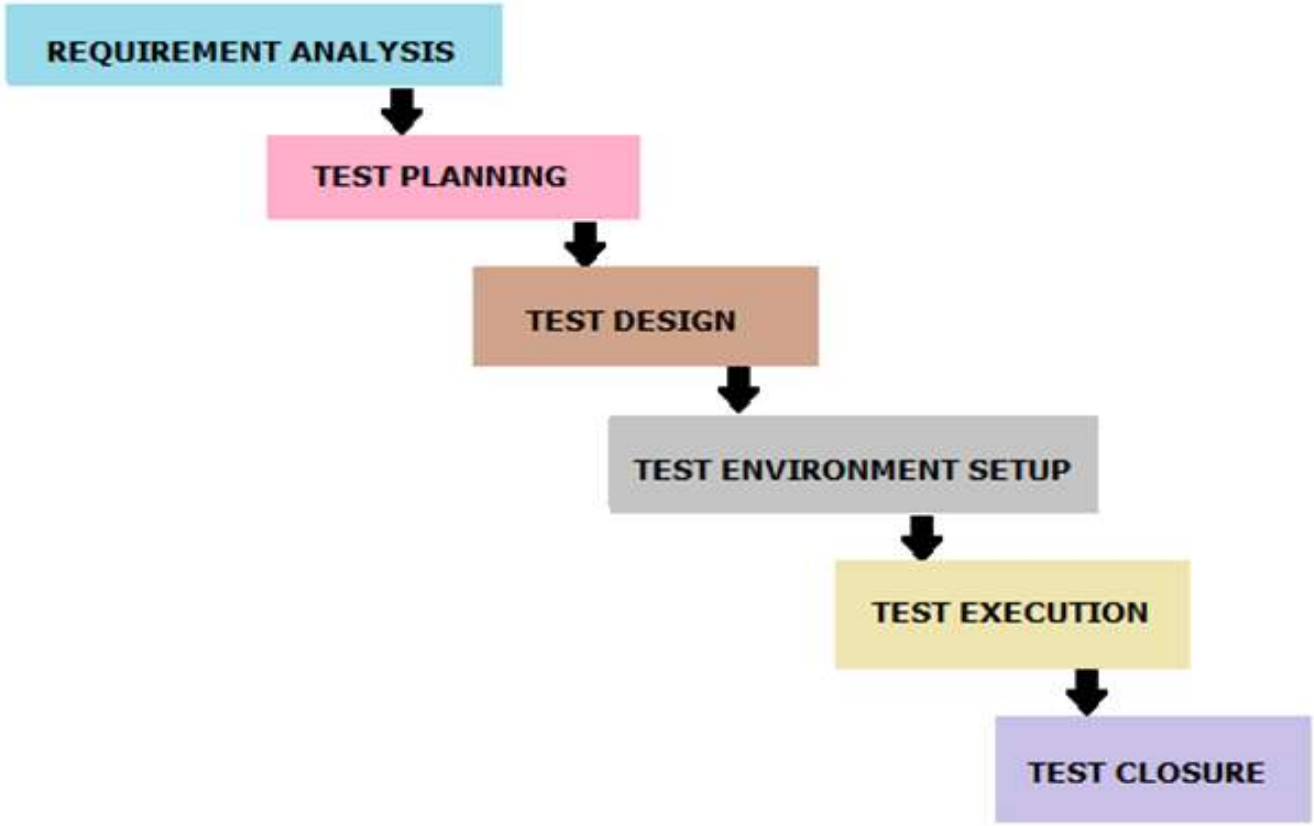
- ***External:*** Derived from the relationship between the environment and the system (or the process). (To derive, the system or process must run)
  - e.g. Reliability, Robustness

- ***Internal:*** Derived immediately from the product or process description (To derive, it is sufficient to have the description)
  - Underlying assumption: internal quality leads to external quality
  - e.g. Efficiency

Quality Factors	Definitions
Correctness	The extent to which a program satisfies its specifications and fulfills the user's mission objectives.
Reliability	The extent to which a program can be expected to perform its intended function with required precision.
Efficiency	The amount of computing resources and code required by a program to perform a function.
Integrity	The extent to which access to software or data by unauthorized persons can be controlled.
Usability	The effort required to learn, operate, prepare input, and interpret output of a program.
Maintainability	The effort required to locate and fix a defect in an operational program.
Testability	The effort required to test a program to ensure that it performs its intended functions.
Flexibility	The effort required to modify an operational program.
Portability	The effort required to transfer a program from one hardware and/ or software environment to another.
Reusability	The extent to which parts of a software system can be reused in other applications.
Interoperability	The effort required to couple one system with another.

SOFTWARE TESTING LIFE CYCLE

- **STLC** identifies what test activities to carry out and when to accomplish those test activities. Even though testing differs between Organizations, there is a testing life cycle.



Requirement Analysis:

- Entry criteria for this phase is BRS (Business Requirement Specification) document.
- During this phase, test team studies and analyzes the requirements from a testing perspective. This phase helps to identify whether the requirements are testable or not.
- If any requirement is not testable, test team can communicate with various stakeholders (Client, Business Analyst, Technical Leads, System Architects etc) during this phase so that the mitigation strategy can be planned.
- **Entry Criteria:** BRS (Business Requirement Specification) **Deliverables:** List of all testable requirements, Automation feasibility report (if applicable)

Test Planning

- Test planning is the first step of the testing process. In this phase typically Test Manager/Test Lead involves determining the effort and cost estimates for the entire project.
- Preparation of Test Plan will be done based on the requirement analysis.
- Activities like resource planning, determining roles and responsibilities, tool selection (if automation), training requirement etc., carried out in this phase.
- The deliverables of this phase are **Test Plan & Effort estimation documents.**

Test Design

- Test team starts with test cases development activity here in this phase.
- Test team prepares **test cases, test scripts (if automation) and test data.**
- Once the test cases are ready then these test cases are reviewed by **peer members or team lead**. Also, test team prepares the **Requirement Traceability Matrix (RTM).**
- RTM traces the requirements to the test cases that are needed to verify whether **the requirements are fulfilled**.
- The deliverables of this phase are **Test Cases, Test Scripts, Test Data, Requirements Traceability Matrix**

Test Environment Setup

- This phase can be started in parallel with Test design phase. Test environment setup is done based on the hardware and software requirement list.
- Some cases test team may not be involved in this phase. Development team or customer provides the test environment.
- Meanwhile, test team should prepare the smoke test cases to check the **readiness of the given test environment.**
- **Entry Criteria:** Test Plan, Smoke Test cases, Test Data **Deliverables: Test Environment. Smoke Test Results**.

Test Execution

- Test team starts executing the test cases based on the planned test cases.
- If a test case result is Pass/Fail then the same should be updated in the test cases.
- Defect report should be prepared for failed test cases and should be reported to the Development Team through bug tracking tool (eg., Quality Center) for fixing the defects.
- **Retesting will be performed** once the defect was fixed.
- **Entry Criteria:** Test Plan document, Test cases, Test data, Test Environment.
  Deliverables: Test case execution report, Defect report, RTM.

Test Closure

- The final stage where we prepare Test Closure Report, Test Metrics.
- Testing team will be called out for a meeting to evaluate cycle completion criteria based on **Test coverage, Quality, Time, Cost, Software, Business objectives**.
- Test team analyses the **test artifacts** (such as Test cases, Defect reports etc.,) to identify strategies that have to be implemented in future, which will help to remove process bottlenecks in the upcoming projects.
- Test metrics and Test closure report will be prepared based on the above criteria.
- **Entry Criteria:** Test Case Execution report (make sure there are no high severity defects opened), Defect report **Deliverables: Test Closure report, Test metrics**

Achieving Software Quality

- Broad activities that help a software team achieve high quality software:
  - **Quality Assurance** (QA)
    - establishes the infrastructure that supports solid software engineering methods, rational project management, and quality control actions
  - **Quality control** (QC)
    - action that helps ensure each work products meets its quality goals
    - e.g., review design models to ensure that they are complete and consistent

- **Software engineering method**
  - understand the problem to be solved, create a design that conforms to the problems and exhibit characteristics that lead to software that are reliable, efficient, usable, etc.
- **Project management technique**
  - use estimation to verify that delivery dates are achievable, schedule dependencies are understood and conduct risk planning so that problem do not breed chaos.

Testing: Strategic Approach to software Testing

- Testing is the **process of executing a program** with the intention **of finding errors.**
- **Strategic Approach**:
  - To perform effective testing, you should **conduct effective technical reviews**. By doing this, **many errors will be eliminated before testing commences**.
  - Testing **begins at the component level** and works "outward" **toward the integration** of the entire computer based system.
  - **Different testing techniques are appropriate** for different software engineering approaches and **at different points in time.**
  - Testing is conducted **by the developer of the software** and (**for large projects) an independent test group**.
  - **Testing and debugging** are **different activities**, but debugging must be accommodated in any testing strategy.

V & V

- **Verification refers to the set of tasks** that ensure that software correctly implements a specific function.
- **Validation refers** to a different set of tasks that **ensure that the software** that has been built is **traceable to customer requirements**.
- Boehm [Boe81] states this another way:
- Verification: "Are we building the product right?"
- Validation: "Are we building the right product?"

Who Tests the Software?

developer

Understands the system

but, will test "gently"

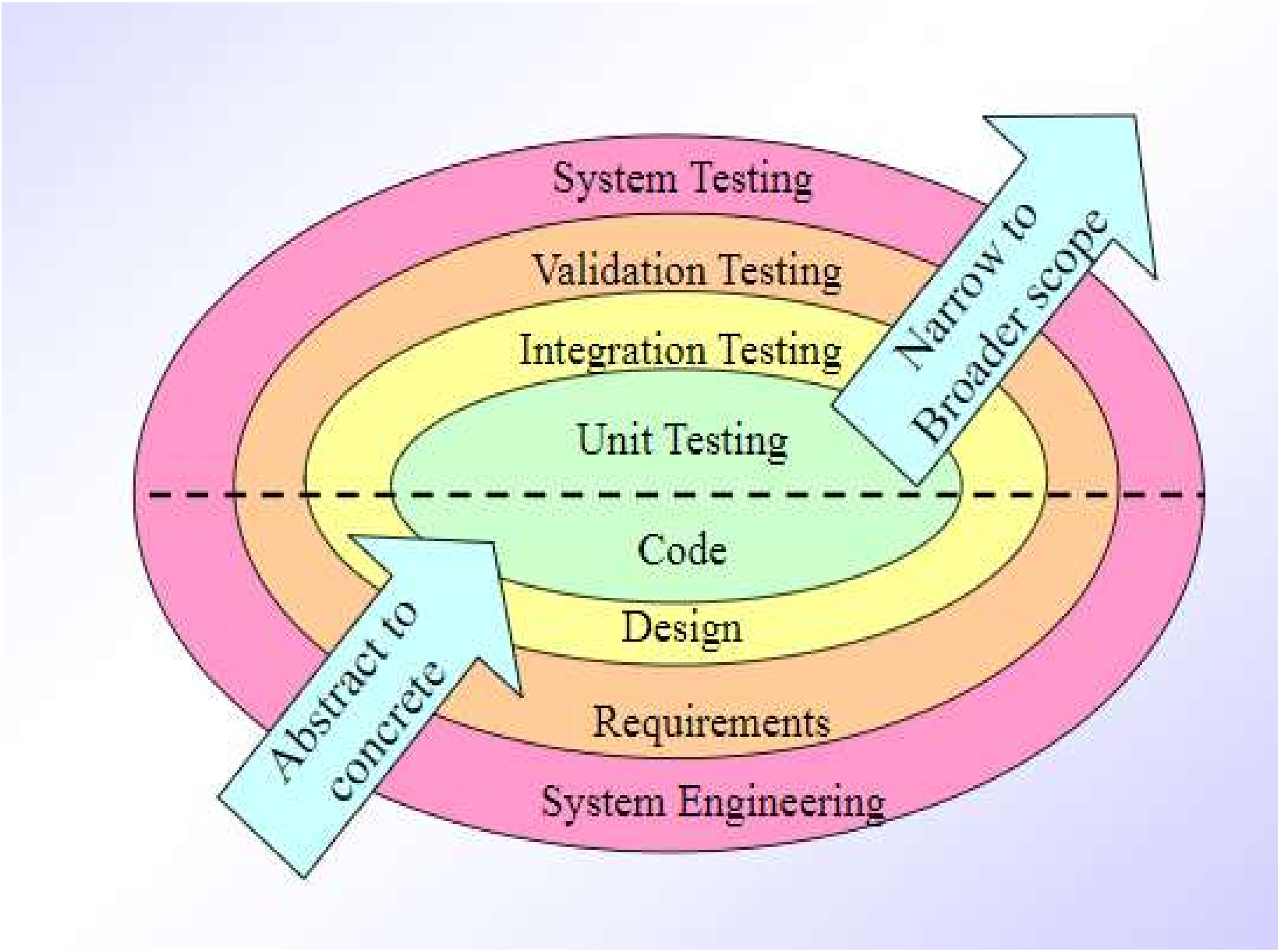
and, is driven by "delivery"

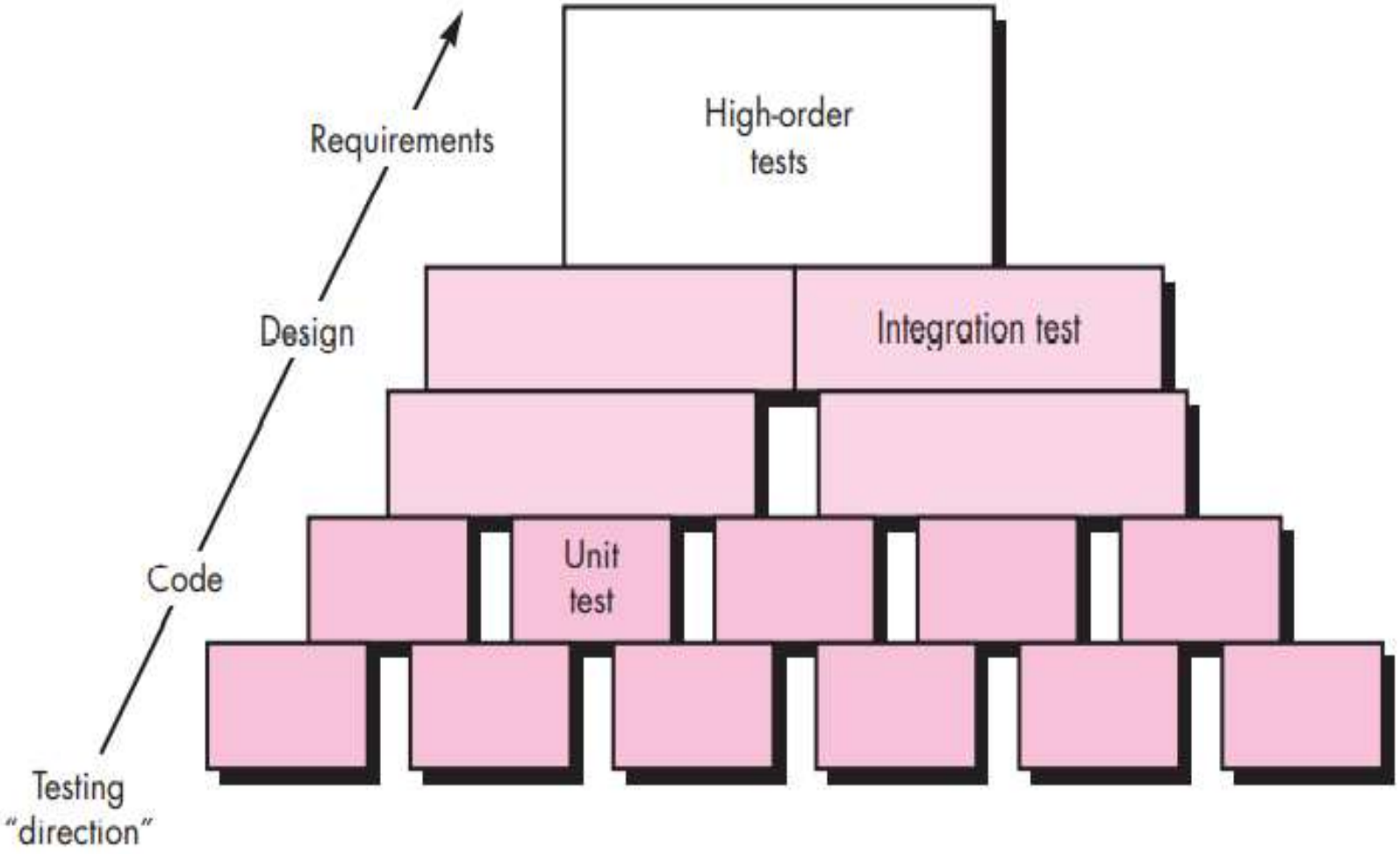
independent tester

Must learn about the system,

but, will attempt to break it

and, is driven by quality


System Testing
Validation Testing
Integration Testing
Unit Testing
Code
Design
Requirements
System Engineering
Narrow to Broader scope
Abstract to concrete


Requirements
Design
Code
Testing
"direction"
High-order
tests
Integration test
Unit
test

Strategic Testing Issues

- Testing is a very important phase in software development life cycle. But the testing may not be very effective if proper strategy is not used. For the **implementation of successful software testing strategy**, the **following issues** must be taken care of: -
- Before the start of the testing process, all the requirements must be specified in a quantifiable manner.
- **Testing objectives must be clarified** and stated explicitly.
- A proper testing plan must be developed.
- Build "robust" software that **is designed to test itself**.
- Use effective formal technical reviews as a filter prior to testing. For this reason, **reviews can reduce the amount of testing effort** that is required to produce high-quality software.

- Conduct formal technical reviews to assess the test strategy and the cases themselves.
- FTRs can uncover inconsistencies, omissions, and outright errors in the testing approach. This saves time and also improves product quality.
- Develop a continuous improvement approach for the testing process.
- The test strategy should be measured. The **metrics collected during testing** should be used as part of a **statistical process control** approach for software testing.

Strategies for Conventional Software

- **Following are the four strategies for conventional software:**

  1) Unit testing

  2) Integration testing

  3) Regression testing

  4) Smoke testing

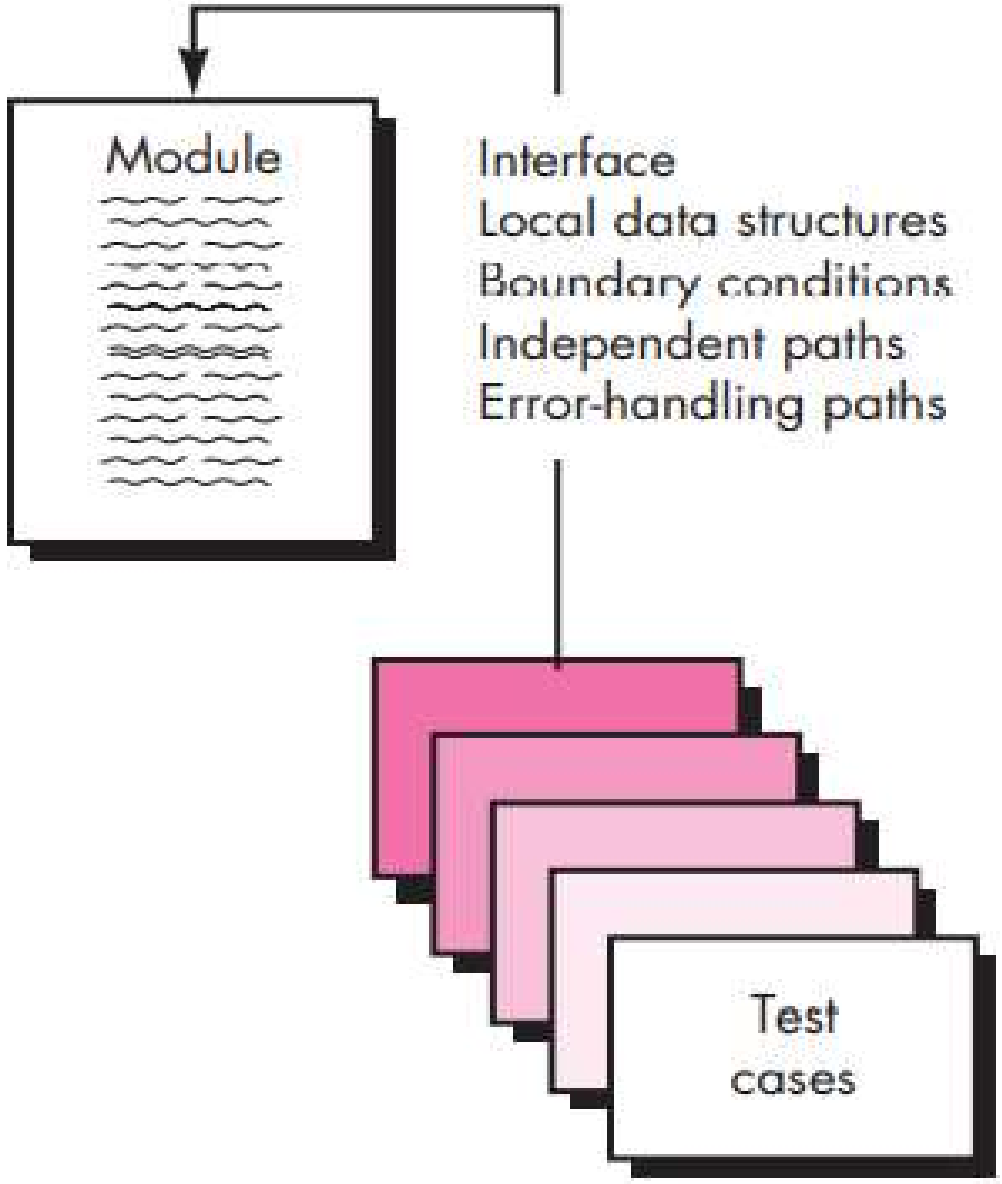
Unit testing

Testing focuses on individual units.

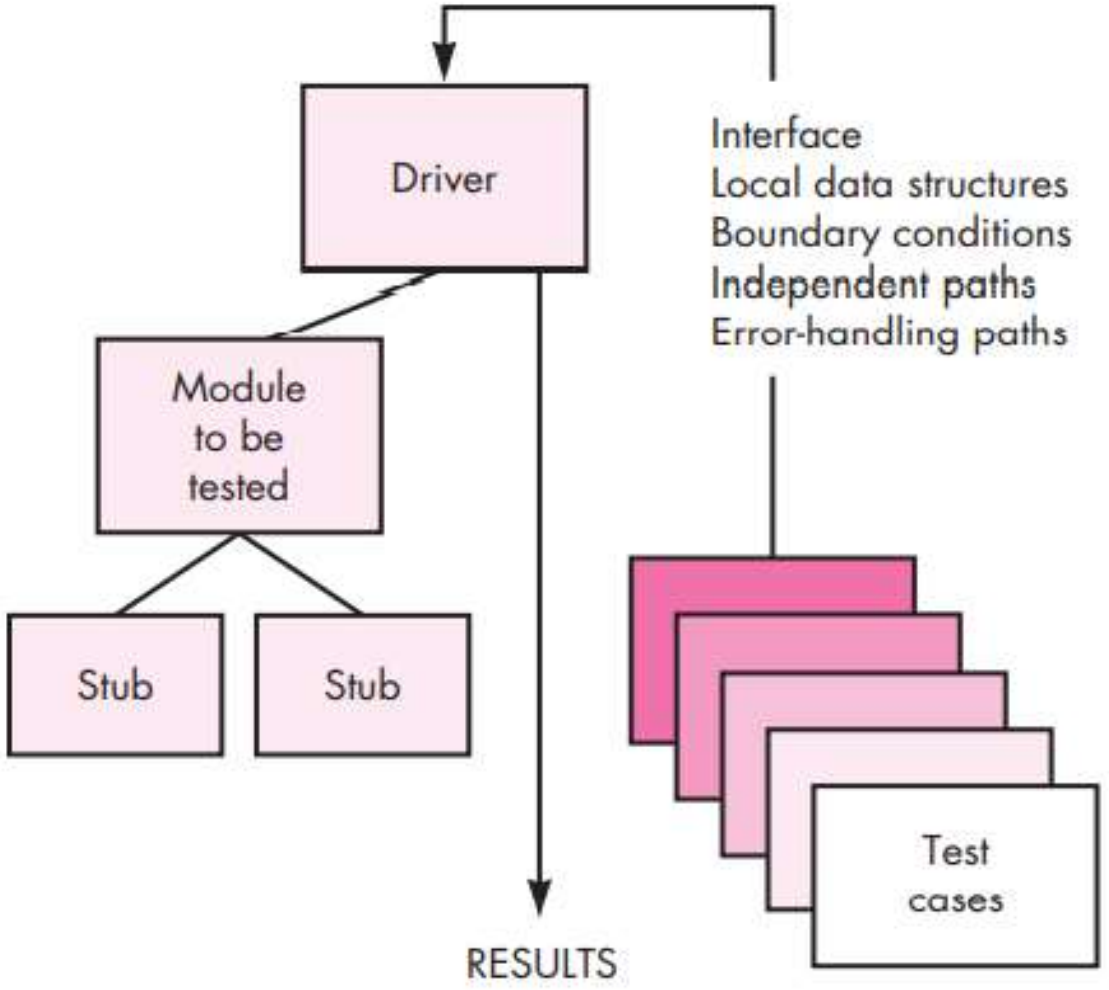
Things to consider as you write test cases for each component:

- ***Interface to the module*** - does information flow in and out properly?
- ***Local data structures*** **-** do local structures maintain data correctly during the algorithms execution?
- ***Boundary conditions*** - all boundary conditions should be tested. Look at loops, First and last record processed, limits of arrays… This is where you will find most of your errors.
- ***Independent paths-*** Try to execute each statement at least once. Look at all the paths through the module.
- ***Error Handling paths-*** Trigger all possible errors and see that they are handled properly,
  - messages are helpful and correct
  - ***exception***-condition processing is correct and
  - error messages identify the location of the true error.

Unit testing



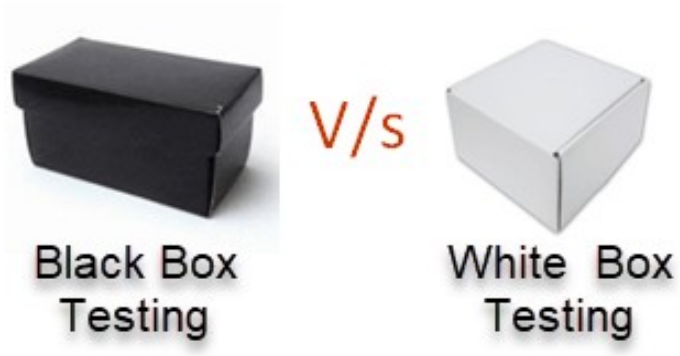
Unit testing - Environment



- Things to consider as you write test cases for each component:
- **Stubs and Drivers:**
  - Additional SW used to help test components.
- **Driver**
  - a main program that passes data to the component and displays or prints the results.
- **Stub**
  - a "dummy" sub-component or sub-program used to replace components/programs that are subordinate to the unit being tested.
  - The stub may do minimal data manipulation, print or display something when it is called and then return control to the unit being tested.

Integration testing

- **Testing focuses on the design and the construction of the SW architecture.**
- This is when we fit the units together.
- **Integration testing includes** both
  - **verification and**
  - **program construction**.
- **Black-box testing techniques** are mostly used.
- Some **white-box testing may be used** for major control paths.



Difference between Blackbox and White Box Testing

Parameter	Black Box testing	White Box testing
Definition	It is a testing approach which is used to test the software without the knowledge of the internal structure of program or application.	It is a testing approach in which internal structure is known to the tester.
Alias	It also knowns as data-driven, box testing, data-, and functional testing.	It is also called structural testing, clear box testing, code-based testing, or glass box testing.
Base of Testing	Testing is based on external expectations; internal behavior of the application is unknown.	Internal working is known, and the tester can test accordingly.
Usage	This type of testing is ideal for higher levels of testing like System Testing, Acceptance testing.	Testing is best suited for a lower level of testing like Unit Testing, Integration testing.
Programming knowledge	Programming knowledge is not needed to perform Black Box testing.	Programming knowledge is required to perform White Box testing.
Implementation knowledge	Implementation knowledge is not requiring doing Black Box testing.	Complete understanding needs to implement WhiteBox testing.

Integration Testing

- Bottom - up testing (test harness).
- Top - down testing (stubs).
- Bing Bang Approach
- Sandwich/Hybrid Approach

- ***Big Bang*** is an approach to Integration Testing where all or most of the units are combined together and tested at one go.

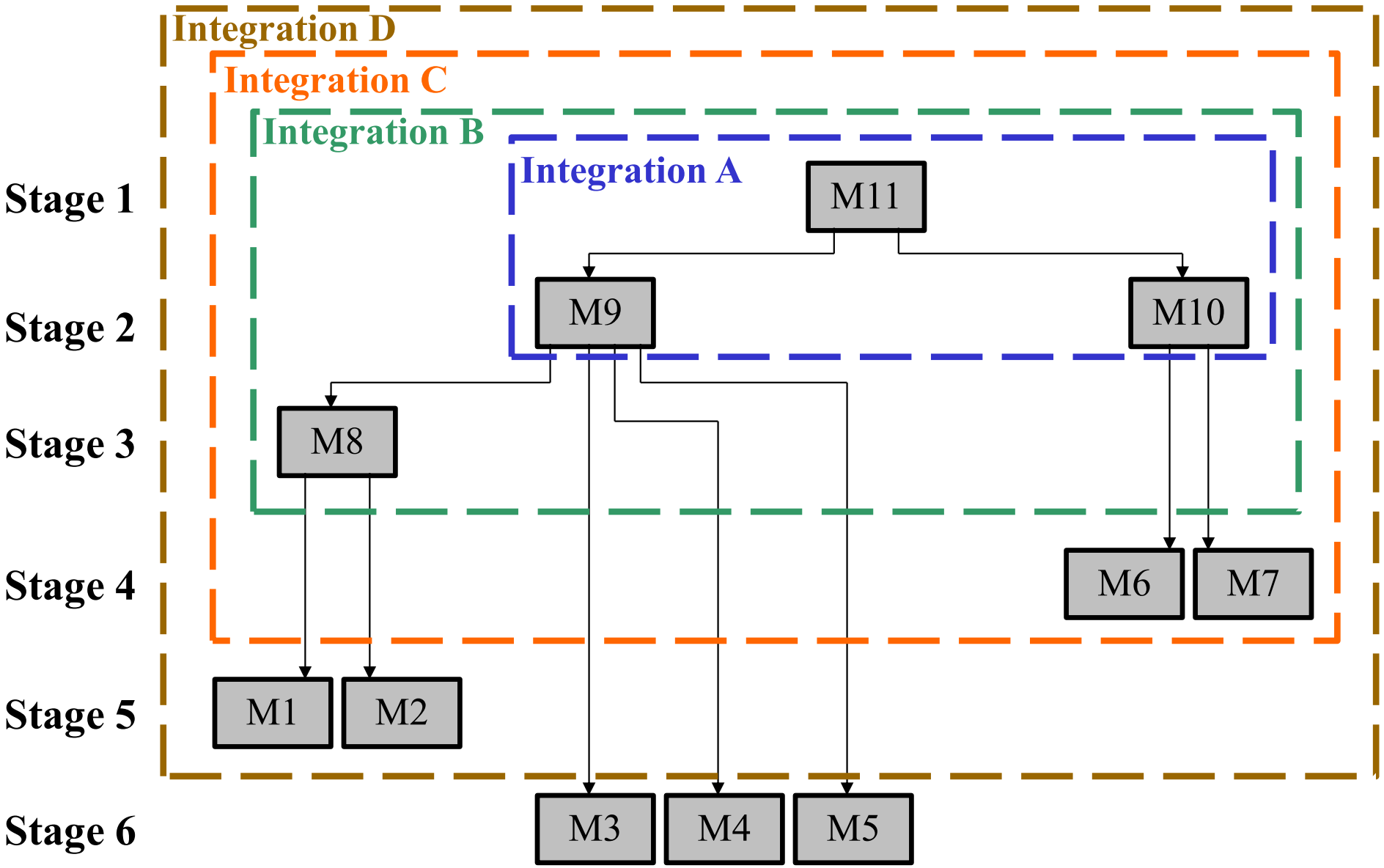
  This approach is taken when the testing team receives the entire software in a bundle.

- ***Sandwich/Hybrid*** is an approach to Integration Testing which is a combination of Top Down and Bottom Up approaches.

Top-Down Integration Testing

- **Main program used as a test driver and stubs are substitutes** for components directly subordinate to it.
- Subordinate stubs are replaced one at a time with real components (following the **depth-first or breadth-first approach).**
- Tests are conducted as each component is integrated.
- On completion of each set of tests and other stub is replaced with a real component.
- **Regression testing may be used** to **ensure that new errors not introduced.**

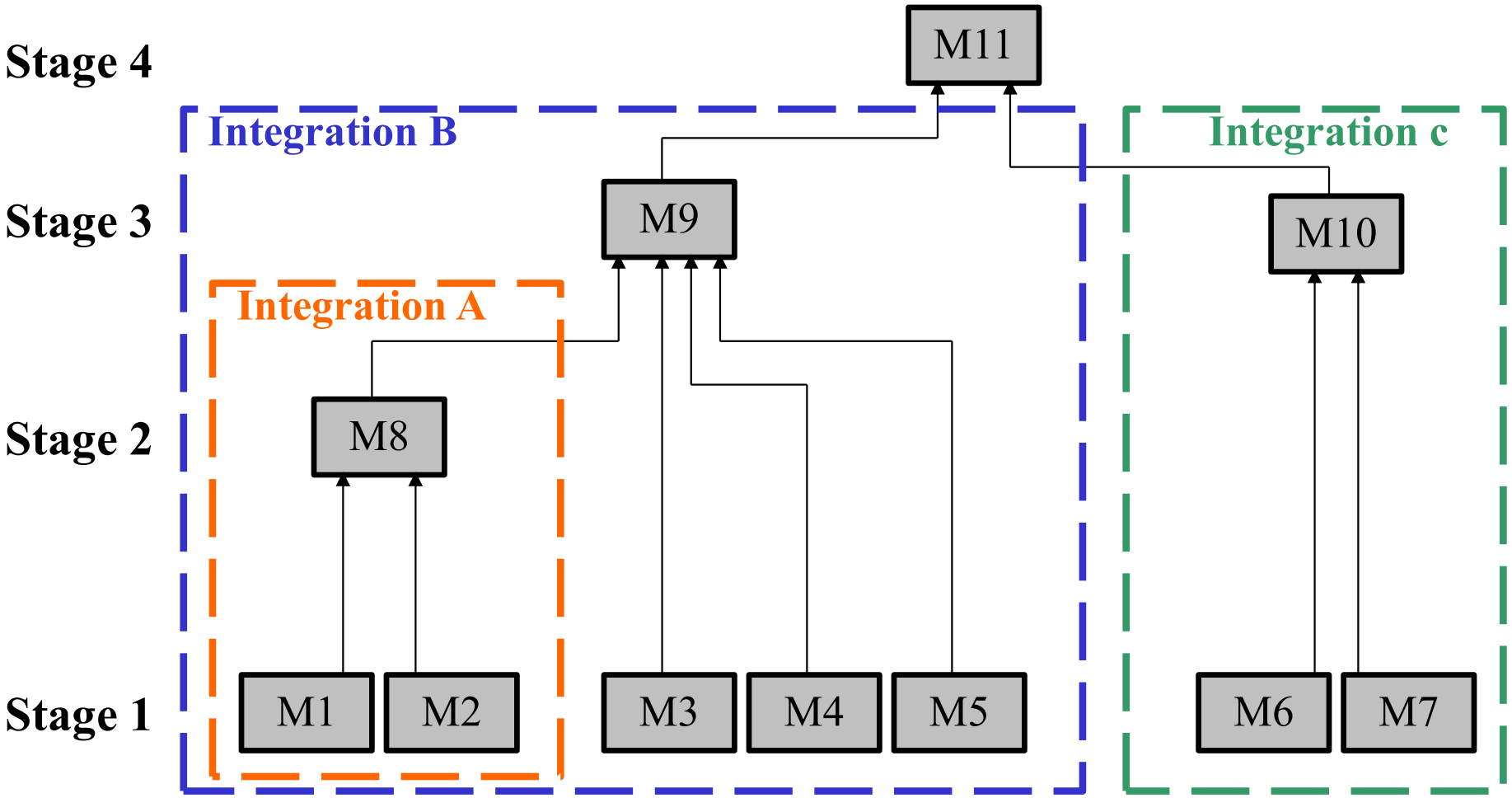
Top-down testing



Bottom-Up Integration Testing

- Low level components are combined in clusters that perform a specific software function.
- A driver (control program) is written **to coordinate test case input and output**.
- The **cluster is tested**.
- **Drivers are removed and clusters are combined** moving upward in the program structure.

Bottom-up testing



Regression Testing

- RT is done to verify that a code change in the software does not impact the existing functionality of the product.
- makes sure that the product works fine as previously with the newly added functionality or any change in the existing feature or once the bug fix is done.
- **Previously executed test cases are re-executed** in order to verify the impact of change.
- Regression means retesting the unchanged parts of the application.

- **Example,** Consider a product X, in which one of the functionality is to trigger confirmation, acceptance, and dispatched emails when Confirm, Accept and Dispatch buttons are clicked.
- Some issue occurs in the confirmation email and in order to fix the same, some code changes are done. In this case, not only the Confirmation emails need to be tested but Acceptance and Dispatched emails also needs to be tested to ensure that the change in the code has not affected them.

Smoke Testing

- A common approach for creating "daily builds" for product software
- Smoke testing steps:
  - 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.
  - A series of tests is designed to expose errors that will keep the build from properly performing its function.
    - The intent should be to uncover "show stopper" errors that have the highest likelihood of throwing the software project behind schedule.
  - The build is integrated with other builds and the entire product (in its current form) is smoke tested daily.
    - The integration approach may be top down or bottom up.

Smoke Testing

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
  - show-stopper errors are uncovered early
  - **reducing** the likelihood of **serious schedule impact** when errors are uncovered
- **The quality of the end product is improved**
  - to uncover functional errors as well as architectural and component-level design errors
  - errors are corrected early, better product quality will result.
- **Error diagnosis and correction are simplified**
  - errors uncovered to be associated with "new software increments"
- **Progress is easier to assess** - improves team morale and gives managers a good indication that progress is being made

Testing Strategies for OO Software

- **Unit Testing in the OO Context**

An encapsulated class is usually the focus of unit testing. However, operations (methods) within the class are the smallest testable units.

Class testing for OO software - tends to focus on
-the **algorithmic detail** of a module
-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.

Testing Strategies for OO Software

- **Integration Testing in the OO Context**

two different strategies for integration testing of OO systems

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

Use-Based testing

- begins the construction of the system by testing those classes that use very few (if any) server classes

- After the independent classes are tested, the next layer of classes, called dependent classes, that use the independent classes are tested

Testing Strategies for OO Software

- **Integration Testing in the OO Context**

Cluster testing is one step in the integration testing of OO software

- a cluster of collaborating classes is exercised by **designing test cases** that **attempt to uncover errors** in the collaborations

Testing Strategies for Web Apps

The following steps summarize the approach:

1. The **content model** for the WebApp is reviewed to uncover errors
2. The **interface model** is reviewed to ensure that all use cases can be accommodated
3. The **design model** for the WebApp is reviewed to uncover navigation errors
4. The **user interface** is tested to uncover errors in presentation and/or navigation mechanics
5. Each **functional component** is unit tested.

Testing Strategies for Web Apps

6. Navigation throughout the architecture is tested.
7. The WebApp is implemented in a variety of different environmental configurations and is tested for compatibility with each configuration
8. Security tests are conducted in an attempt to exploit vulnerabilities
9. Performance tests are conducted
10. WebApp is tested by a **controlled and monitored population** of end users

Validation Testing

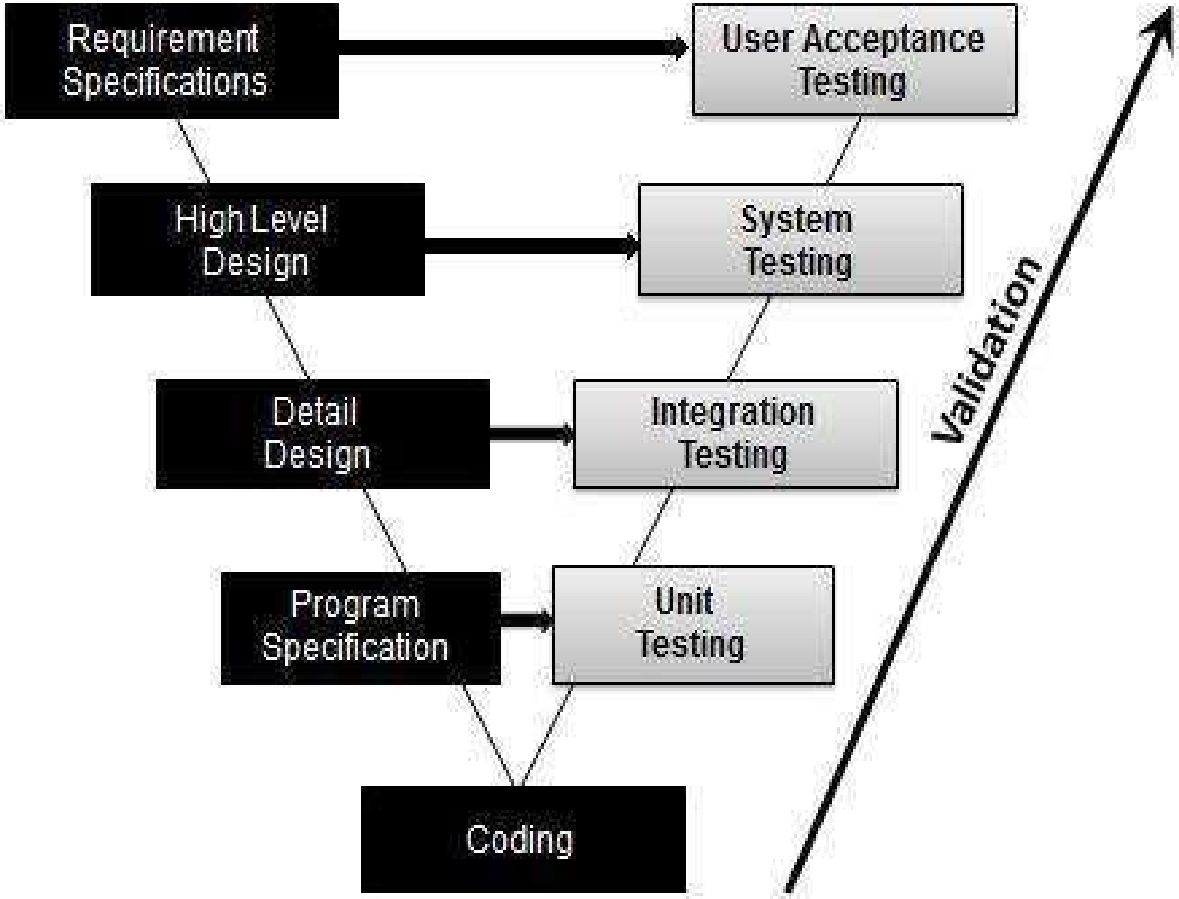
- Ensure that each function or performance characteristic conforms to its specification.
- Deviations (deficiencies) must be negotiated with the customer to establish a means for resolving the errors.
- Configuration review or audit is used to ensure that all elements of the software configuration have been properly developed, cataloged, and documented to allow its support during its maintenance phase.

Validation Testing

- 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.
- Testing ensures that all functional, behavioral and performance requirements of the system were satisfied.
- These were detailed in the SW requirements specification.
- Only Black-box testing techniques are used.
- The goal is to prove conformity with the requirements.
- For each test case, one of two possibilities conditions will exist:

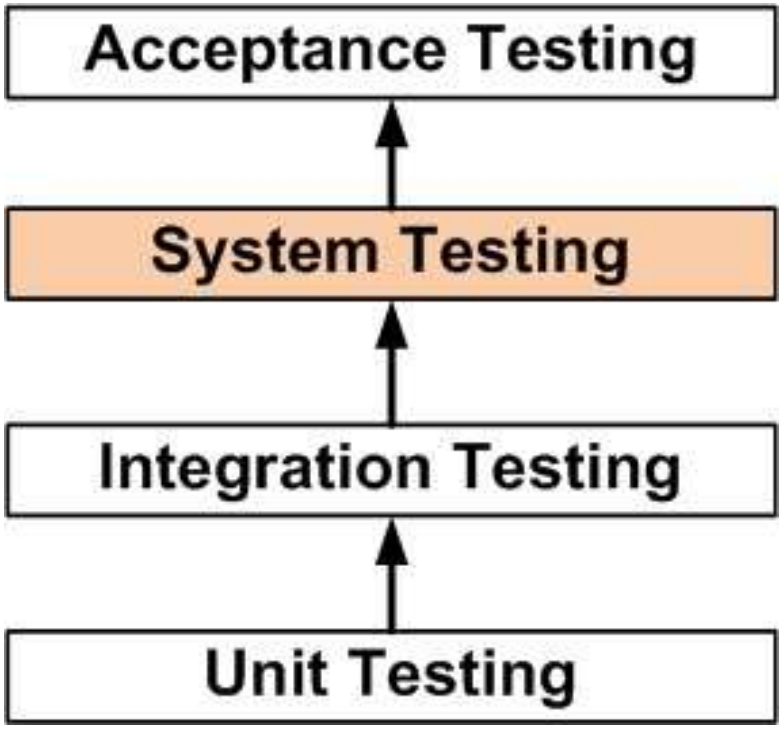
1. The test for the function or performance criteria will conform to the specification and is accepted.
2. A deviation will exist and a deficiency is uncovered.

- This could be an error in the SW or a deviation from the specification
- SW works but it does not do what was expected.

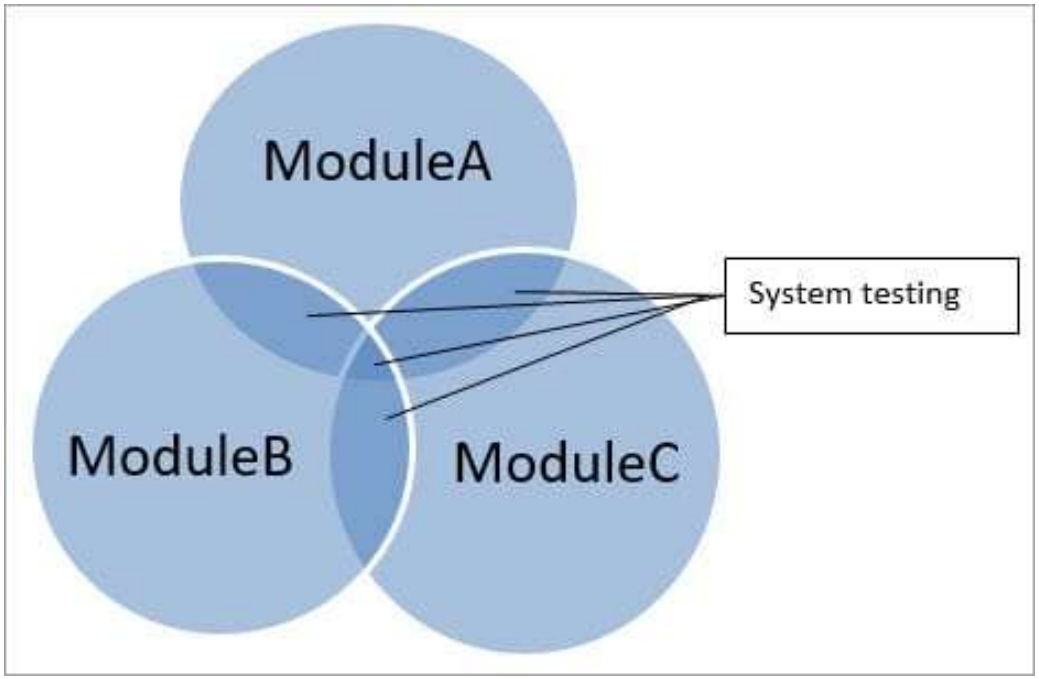


SYSTEM TESTING

- System testing **means testing the system as a whole**.
- All the modules/components are integrated in order to verify if the system works as expected or not.



- If an application has three modules A, B, and C, then testing done by combining the modules A & B or module B & C or module A& C is known as Integration testing.
- Integrating all the three modules and testing it as a complete system is termed as System testing.



- A **series of tests** to **verify that all system elements** have been properly integrated.
- **Recovery Testing**:
  - Forces software to fail in a variety of ways and **verifies** that recovery is properly performed.
- **Security Testing:**
  - is a process that is performed with the intention of revealing flaws in **security** mechanisms and finding the vulnerabilities or weaknesses of software applications.
- **Stress Testing:**
  - Executes the system in a manner that **demands resources in abnormal quantity, frequency** or volume.
- **Performance Testing:**
  - To test the run time performance of a system within the context of an integrated system.

Acceptance Testing

- Making sure the software works correctly for intended user in his or her normal work environment.
- **Alpha test**
  - version of the complete **software is tested by customer** under the supervision of the developer **at the developer's site**
- Beta test
  - version of the complete software is tested by customer at his or **her own site** without the developer being present

Example

- *A car manufacturer does not produce the car as a whole car. Each component of the car is manufactured separately, like seats, steering, mirror, break, cable, engine, car frame, wheels etc.*
- *After manufacturing each item, it is tested independently whether it is working the way it is supposed to work and that is called* ***Unit testing.***
- *Now, when each part is assembled with another part, that assembled combination is checked if assembling has not produced any side effect to the functionality of each component and whether both components are working together as expected and that is called* ***integration testing****.*
- *The whole car needs to be checked for different aspects as per the requirements defined like if car can be driven smoothly, breaks, gears and other functionality working properly, car does not show any sign of tiredness after being driven for 2500 miles continuously, color of car is generally accepted and liked, car can be driven on any kind of roads like smooth and rough, sloppy and straight etc and this whole effort of testing is called* ***System testing*** *and it has nothing to do with integration testing.*

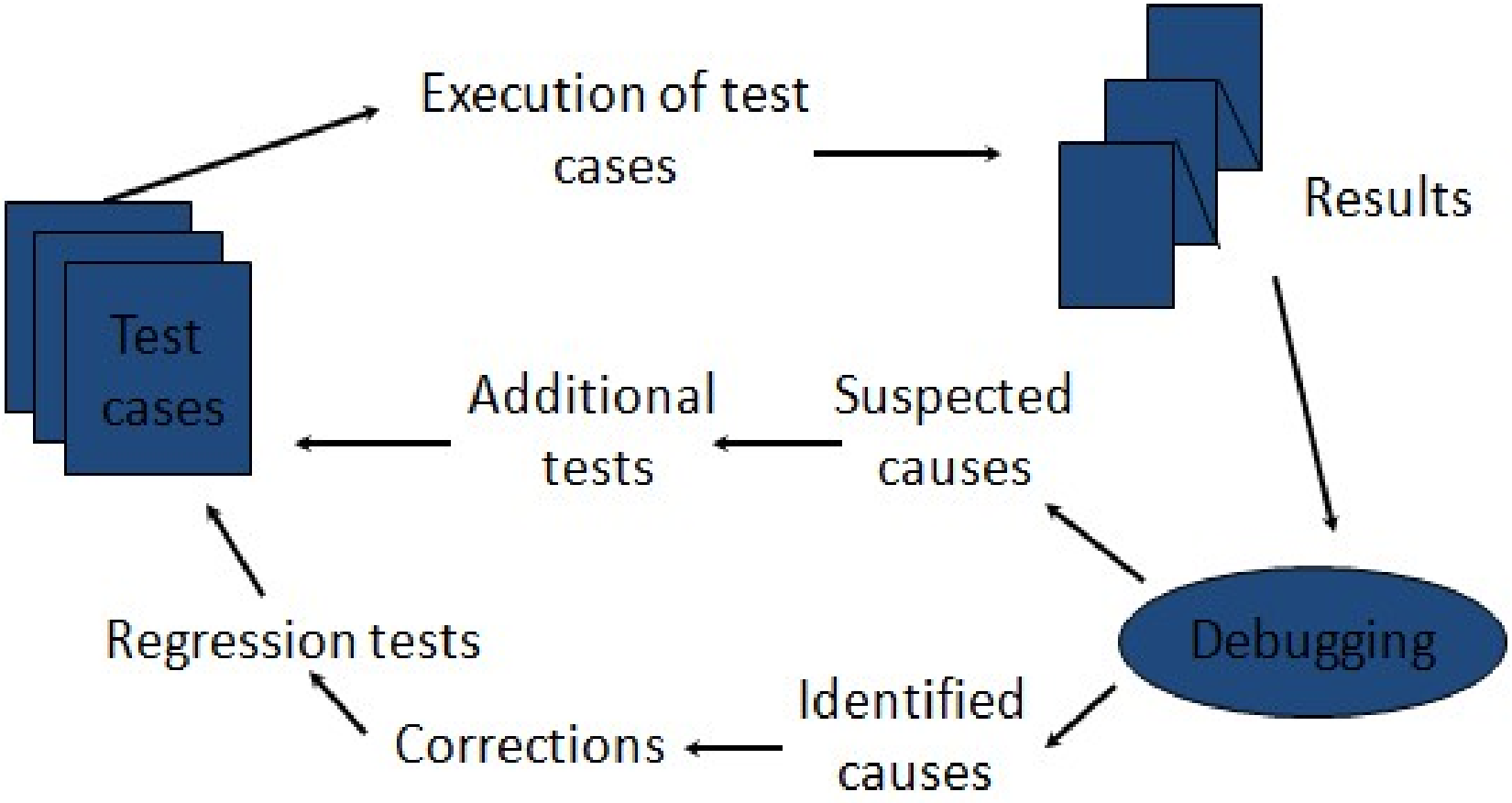
ART OF DEBUGGING

- It is a **consequence of successful testing** - when a test case uncovers an error, it is the debugging process that results in the removal of the error.
- Debugging is an ART. The external manifestation of the error and the cause of the error normally do not share an obvious relationships.

What is a BUG?

- **A fault in a program**, which causes the program to perform in an unintended or unanticipated manner.
- A program that contains a large number of bugs, and/or bugs that seriously interfere with its functionality, is said to be **buggy.**
- Reports detailing bugs in a program are commonly known as **bug reports**, fault reports, problem reports, trouble reports, defect reports etc.

Debugging Process



Debugging Strategies

- As debugging is a **difficult and time-consuming** task, it is essential to develop a proper debugging strategy.

  - **Brute force**
  - **Backtracking strategy**
  - **Cause Elimination**

- **Brute force :** most commonly used but least efficient method. Many developers rely on brute-force tactics when debugging. That involves using the debugger to step across the code from start to finish until you notice something odd.
- In this method, **a printout of all registers and relevant memory locations is obtained and studied**. All dumps should be well documented and retained for possible use on subsequent problems.

Back Tracking Method: It is a quite popular approach of debugging which is used effectively in case of small applications.

- The process **starts from the site where a particular symptom** gets detected, from there on **backward tracing is done across the entire source code** till we are able to lay our hands on the site being the cause.
- Unfortunately, as the **number of source lines increases**, the **number of potential backward paths may become** unmanageably large.

Cause Elimination: It is manifested by induction or deduction and introduces the concept of binary partitioning.

- This approach is also called **induction and deduction**..
- A "**cause hypothesis**" is devised and the data are used to prove or disprove the hypothesis.
- Alternatively, a list of **all possible causes is developed** and **tests are conducted to eliminated each**. If initial **tests indicate that a particular cause hypothesis** shows **promise**, the **data are refined** in an attempt to isolate the bug.

Bebugging is the process of adding known **defects to the application intentionally** for the **purpose of monitoring the rate of detection and removal**.

This process is also known as defect seeding or Fault injection or defect feeding.