Unit - IV Basics of Software Testing

📌 1. Software Testing

✅ **Definition:** Software Testing is the process of **evaluating and verifying** that a software application or system meets the specified requirements. It helps to identify **bugs, errors, or missing requirements** in the software.

🧪 Types of Testing (Brief):

- **Manual Testing**: Done by testers manually.
- **Automation Testing**: Done using tools like Selenium, JUnit.
- **White Box Testing**: Tests internal logic (by developers).
- **Black Box Testing**: Tests functionality without knowing the internal code.
- **Unit, Integration, System, Acceptance Testing** – based on development phases.

✅ Benefits:

- Detects and prevents software defects.
- Ensures reliability and performance.
- Enhances software quality.
- Reduces maintenance cost.
- Ensures customer satisfaction.

❌ Drawbacks:

- Can be time-consuming.
- Might not find all bugs.
- Requires skilled testers.
- Testing tools may be expensive.

📘 **Example:**

Testing a mobile banking app to ensure:

- Login works correctly
- Funds transfer functions properly
- Invalid inputs are handled gracefully

📝 **Key Points:**

- Testing can be done **manually or using tools**.
- It is an essential phase of the Software **Development Life Cycle (SDLC)**.
- The goal is to find defects **before the software reaches the user**.

📌 2. Objectives of Software Testing

🎯 **Main Objectives:**

1. **To detect defects** in the software.
2. **To ensure the software meets requirements**.
3. **To validate and verify** the product's functionality.
4. **To ensure quality** and **reliability**.
5. **To improve performance** and user experience.
6. **To reduce the cost** of failure in production.

✅ **Benefits:**

- Builds **customer confidence**.
- Avoids **future system failures**.
- Ensures **security** in sensitive applications.
- Helps in **decision-making** about release readiness.

📘 **Example:**

Before launching an e-commerce website, testing ensures:

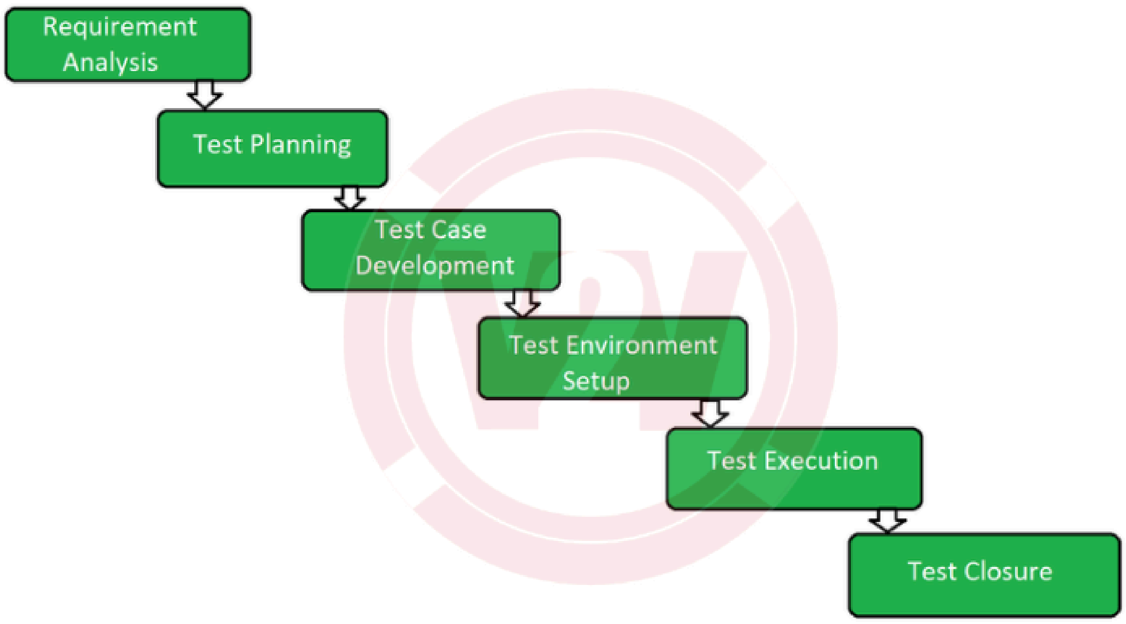
- Products can be added to the cart
- Payments are processed securely
- Orders are stored correctly

📝 **Key Points:**

- Testing is not only to find errors but also to **prove software works correctly**.
- The objective changes with **type of testing (e.g., functional, performance, security)**.

📌 3. Software Testing Life Cycle (STLC)

✅ **Definition:** STLC is a **sequence of activities** carried out during the testing process to ensure software quality goals are met. It defines **what to do and when to do it** in testing.



🔄 Phases of STLC:

Phase	Description
1. Requirement Analysis	Understand what needs to be tested by analyzing documents (SRS, BRS).
2. Test Planning	Plan testing strategy, resources, schedule, and risk management.
3. Test Case Design/Development	Write test cases and test scripts, define expected results.
4. Test Environment Setup	Set up hardware/software to execute tests.
5. Test Execution	Run test cases and report any defects.
6. Test Closure	Evaluate test results, prepare closure report, archive data.

✅ Benefits of STLC:

- Brings **structure and discipline** to the testing process.
- Helps in **early detection of defects**.
- Makes **resource and time estimation easier**.
- Improves **test coverage and quality**.

❌ Drawbacks:

- Can be **lengthy** for small projects.
- Requires **proper coordination** among teams.
- May **delay release** if not planned well.

📘 **Example:**

Let’s say you are testing a food delivery app:

- In **Requirement Analysis**, understand the order, cart, and payment system.
- In **Test Case Design**, write cases like “Add item to cart” and “Apply promo code”.
- In **Execution**, run test cases on both Android and iOS.

📝 **Key Points:**

- STLC is **different from SDLC** (STLC is testing-focused).
- Each STLC phase has **entry and exit criteria**.
- Proper documentation is maintained in every phase.

📘 Failure, Fault, Error, Defect, Bug – Explained in Proper Order

---

✅ 1. Error (Mistake)

🔹 **Definition:** An **error** is a human action that produces an incorrect result. It occurs during any phase of software development – such as requirement gathering, design, coding, etc.

🔹 **Who Causes It?**

Usually caused by developers, analysts, designers, or testers.

🔹 **Example:** A developer writes total = price * qty - discount instead of total = price * qty + tax.

🔹 **Real-World Analogy:** A chef accidentally adds **salt instead of sugar** while making a cake. That’s a **human mistake (error)**.

✅ 2. Fault (Bug or Flaw in Code)

🔹 **Definition:** A **fault** is the result of an error in the code. It is the actual flaw or defect in the program logic.

🔹 **When does it happen?**

After an error is written into the code and compiled.

🔹 **Example:** Wrong discount logic in code causes incorrect bill calculations.

🔹 **Real-World Analogy:** The wrong ingredient (salt instead of sugar) is **already mixed into the cake batter**. The cake **now has a fault**.

✅ 3. Defect (Detected Fault)

🔹 **Definition:** A **defect** is a fault or bug that is **found during testing**. When the actual result deviates from the expected result, it's logged as a defect.

🔹 **Who finds it?**

Testers during the Software Testing Life Cycle (STLC).

🔹 **Example:** A tester notices that a "Buy 1 Get 1" offer is not applied properly and logs a **defect**.

🔹 **Real-World Analogy:** The customer tastes the cake and complains: "It tastes salty!" — The **defect is identified**.

✅ 4. Bug (Informal term for Defect)

🔹 **Definition:** A **bug** is an informal name for a **defect or fault** in the software that causes it to behave unexpectedly.

🔹 **Who reports it?**

Generally reported by testers or users. Used in bug tracking tools like Jira, Bugzilla.

🔹 **Example:** Bug reported in a bug tracker: *"App crashes when clicking the 'Submit' button without entering data."*

🔹 **Real-World Analogy:** When a restaurant manager writes down the issue: *"Cake too salty – kitchen bug!"*

✅ 5. Failure (Execution of a Fault)

🔹 **Definition:** A **failure** is the **actual manifestation of a defect during execution**. It occurs when the software does not perform as intended in the real environment.

🔹 **Who experiences it?**

End users, clients, testers — anyone using the software.

🔹 **Example:** Customer orders online, but **receives an empty cart despite selecting items** → application **failed** during execution.

🔹 **Real-World Analogy:** The salty cake is served to a customer, who refuses to eat it. That's the **failure**.

📘 Test Case, When to Start and Stop Testing

✅ 1. Test Case

🔹 **Definition:** A **test case** is a **set of conditions or steps** designed to verify a specific feature or functionality of a software application. It includes the **test input**, **execution conditions**, **expected results**, and **actual output**.

🔹 **Key Components of a Test Case:**

Field	**Description**
Test Case ID	Unique identifier
Test Description	What is being tested
Preconditions	Setup or state before testing
Test Steps	Step-by-step actions to execute
Test Data	Inputs used in the test
Expected Result	What should happen

Field	Description
Actual Result	What actually happened
Status	Pass / Fail
Remarks	Additional notes

✅ Benefits of Test Cases:

- Ensure **complete test coverage**
- Helps in **repeatability** and **regression testing**
- Useful for **manual and automated testing**
- Acts as a **guide for new testers**
- Helps in identifying **defects systematically**

❌ Drawbacks:

- Time-consuming to write and maintain
- Requires clear understanding of requirements
- May become **outdated** if application changes frequently

📘 Example Test Case:

Field	Example
Test Case ID	TC_001
Test Description	Verify login with valid credentials
Preconditions	User is on login screen
Test Steps	1. Enter username 2. Enter password 3. Click login

Field	Example
Test Data	Username: abhay, Password: 12345
Expected Result	User is redirected to dashboard
Actual Result	User is redirected to dashboard
Status	Pass

✅ 2. When to Start Testing?

🟢 **Testing should start as early as possible in the Software Development Life Cycle (SDLC).**

This is known as the **"Shift Left"** approach.

🕒 Ideal Points to Start Testing:

Phase	Action
Requirement Phase	Review and verify requirements
Design Phase	Static testing, design validation
After Development Starts	Unit testing begins
Post Code Completion	Integration, system, and acceptance testing

---

✅ Benefits of Early Testing:

- Detect defects early, when they are **cheaper to fix**
- Reduces overall development cost and rework
- Improves test planning and preparation time

📘 Example:

In a project where testing began **after coding**, 40 defects were found in production.
But in a similar project where testing began **from the requirement stage**, only 10 defects leaked.

✅ 3. When to Stop Testing?

❓ **Stopping criteria (also called exit criteria) vary based on project and testing goals.**

You cannot test everything, so you stop when **certain conditions are met**.

🛑 Common Conditions to Stop Testing:

Criteria	**Description**
Test Coverage	All planned test cases have been executed
Bug Rate	Defect rate falls below an acceptable level
Deadlines	Testing time or budget is exhausted
Risk Evaluation	Remaining risks are acceptable
Business Goals	All critical business flows are tested and stable
Approval	Product owner or QA manager gives sign-off

❌ Risks if You Stop Testing Too Early:

- Uncovered defects may leak into production
- Reduced software reliability and user trust
- Potential revenue loss or reputation damage

📝 Key Points to Remember:

- **Test Case** = What to test, how to test, and what to expect
- Start testing from the **requirement phase** to catch early defects
- Stop testing when **risk is acceptable and goals are met**
- Use proper **entry and exit criteria** in the Software Testing Life Cycle (STLC)

📘 Quality Assurance (QA), Quality Control (QC), Verification & Validation, Six Sigma, and CMMI

✅ 1. Quality Assurance (QA)

🔹 **Definition:** Quality Assurance is a **process-oriented** activity that focuses on **preventing defects** in the software development process by improving the development and testing processes.

🔹 **Main Goal:**

- To **ensure the process** used to manage and create deliverables is effective and followed correctly.

🧩 **Characteristics:**

- Focuses on **process improvement**
- Involves activities like **process definition, training, audits**
- It is **proactive**

Elements of QA:

1. Standards: Ensure that standards are adopted and followed.

2. Reviews and Audits: Audits are reviews performed by SQA personnel to ensure that quality guidelines are followed.
3. Testing: Ensure that testing is properly planned and conducted.
4. Error/Defect Collection and Analysis: Collects and analyses error and defect data to better understand how errors are introduced and can be eliminated.
5. Changes Management: Ensures that adequate change management practices have been instituted.
6. Education: Takes lead in software process improvement and educational programs.

7. Vendor Management: Suggests specific quality practices vendor should follow and incorporate quality mandates in vendor contracts.

8. Security Management: Ensures use of appropriate process and technology to achieve desired security level.

9. Safety: Responsible for assessing impact of software failure and initiating steps to reduce risk.

10. Risk Management: Ensures risk management activities are properly conducted and that contingency plans have been established.

✅ **Benefits:**

- Reduces risk of defect generation
- Enhances **development lifecycle quality**
- Builds **customer trust**

📘 **Example:** Creating and enforcing coding standards, test plans, and regular process audits.

✅ **2. Quality Control (QC)**

🔹 **Definition:** Quality Control is a **product-oriented** activity that focuses on **identifying and fixing defects** in the final product.

🔹 **Main Goal:**

- To **identify defects** in the finished product before it reaches the customer.

🧩 **Characteristics:**

- Focuses on **defect detection**
- Involves **testing, reviews, inspections**
- It is **reactive**

Quality control activities include:
1. Defining and classifying the severity of defects
2. Inspecting documents

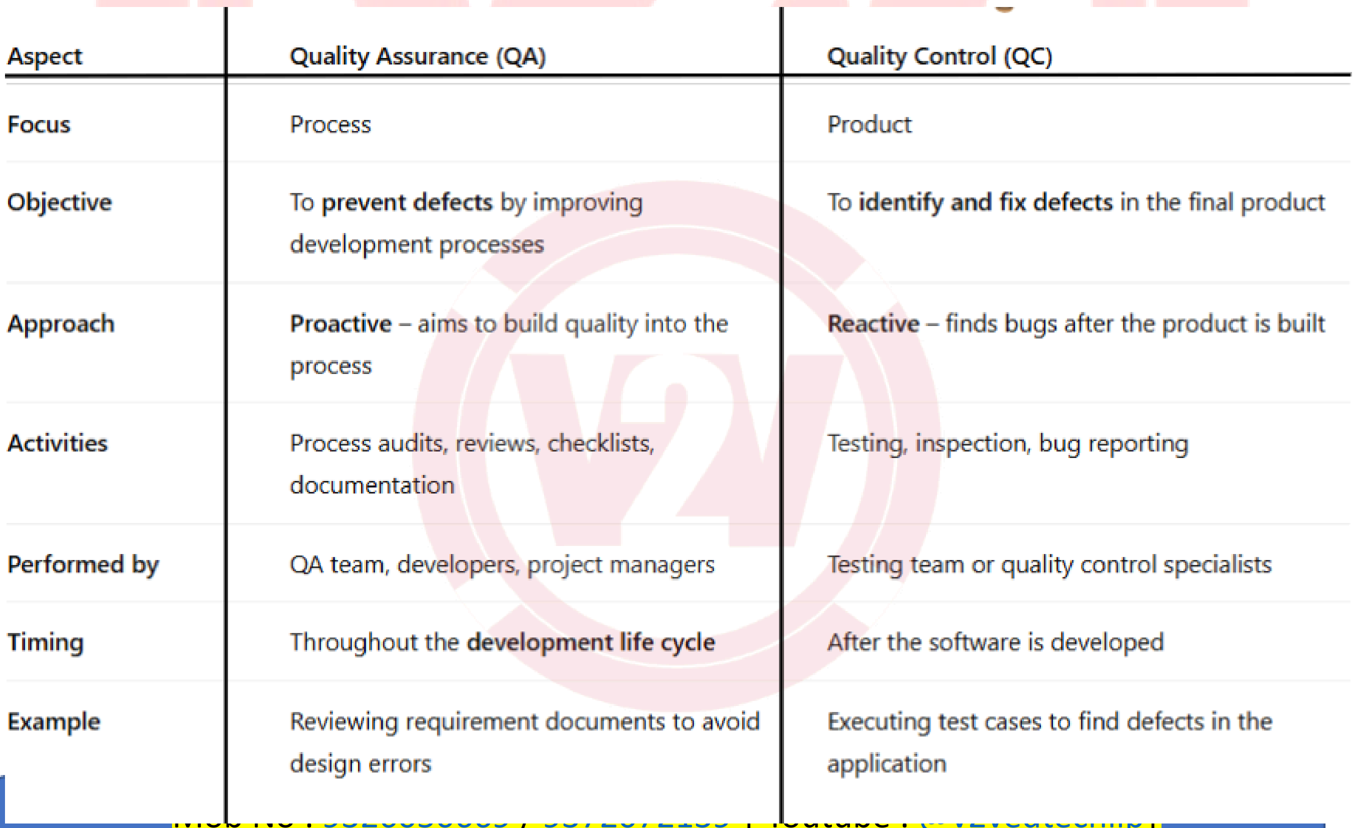
3. Inspecting code (either manually or via automatic static code analysis).
4. Testing executable software. For example: module, unit, integration, system and acceptance testing
5. Recording of defects
6. Setting quality control limits outside of which corrective action must be taken
7. Tracking corrective action on defects
8. Defect data analysis.

✅ **Benefits:**

- Helps identify actual product flaws
- Ensures **software meets functional requirements**

📘 **Example:** Manual testing of the login module to verify that it accepts valid credentials.

🆚 Difference Between QA and QC



Aspect	Quality Assurance (QA)	Quality Control (QC)
Focus	Process	Product
Objective	To **prevent defects** by improving development processes	To **identify and fix defects** in the final product
Approach	**Proactive** – aims to build quality into the process	**Reactive** – finds bugs after the product is built
Activities	Process audits, reviews, checklists, documentation	Testing, inspection, bug reporting
Performed by	QA team, developers, project managers	Testing team or quality control specialists
Timing	Throughout the **development life cycle**	After the software is developed
Example	Reviewing requirement documents to avoid design errors	Executing test cases to find defects in the application

✅ 3. Verification and Validation (V&V)

🔹 **Verification (Are we building the product right?)**

- Ensures the product is developed **according to requirements and design documents**.
- Involves: **Reviews, walkthroughs, inspections**
- Focus is on **process and documents**

🔹 **Validation (Are we building the right product?)**

- Ensures the product **meets user needs and expectations**.
- Involves: **Functional, system, and user acceptance testing**
- Focus is on **actual software output**

🆚 Difference Between Verification and Validation

Feature	**Verification**	**Validation**
Focus	Process compliance	Meeting user needs
Performed when	During development	After development
Method	Reviews, audits, walkthroughs	Testing actual product
Purpose	Ensure correct implementation	Ensure product correctness

✅ 4. Quality Evaluation Standards

📌 A. Six Sigma

🔹 **Definition:** Six Sigma is a **data-driven methodology** that aims to improve quality by **reducing defects** to a rate of **3.4 defects per million opportunities** (DPMO).

Six sigma is regarded as the most widely used strategy or technique for evaluating software quality.
Six sigma is used to eliminate error or faults or bugs in the software by evaluating the performance of the software process.
The objective of six sigma is to reduce variance and improve the processes in an organization. Six sigma minimizes the cost of poor quality.

It monitors day-to-day activities of an organization, in order to make optimal use of resources.

Below six standards of six sigma are described:

1. Good business strategy that balances cost, quality, features and constraints as well as matches business priorities.
2. Decisions made are tied to the bottom line of the organization.
3. Exercise care to use correct measurements in each situation.
4. Consider measuring output for a longer period.
5. Limited scope with adequate importance and reasonable time.
6. Clear quantitative measures to define success.

Concept of DMAIC and DMDAV:

1. DMAIC Method:

· The letters in the abbreviation DMAIC stands for "Define, Measure, Analyze, Improve, Control", the steps in ordered process.

· DMAIC checks whether a process is performing correctly. DMAIC improves the system by improving the process that is not able to meet user requirements.

· DMAIC is used when a product or process is in existence and is not meeting customer specification.

2. DMADV:

· The letters in the abbreviation DMADV stand for "Define, Measure, Analyze, Design, Verify," the steps in the ordered process.

· DMADV is a process defined by Motorola as part of their Six Sigma management philosophy.

· DMADV is applied to new processes to make sure that they achieve Six Sigma quality. Six Sigma sets extremely ambitious goals to minimize the occurrence of flaws in products and services.

When to use DMADV?

1. A product or process is not in existence at an organization and one needs to be developed.

2. The existing product or process exists and has been optimized (using either DMAIC or not) and still does not meet the level of customer specification or Six Sigma level.

· Steps of DMDAV are given below:

D: Define: State the problem, specify the customer set, identify the goals, and outline the target process.

M: Measure : Decide what parameters need to be quantified, work out the best way to measure them, collect the necessary data, and carry out the measurements by experiment.

A: Analyze: Identify performance goals and determine how process inputs are likely to affect process outputs.

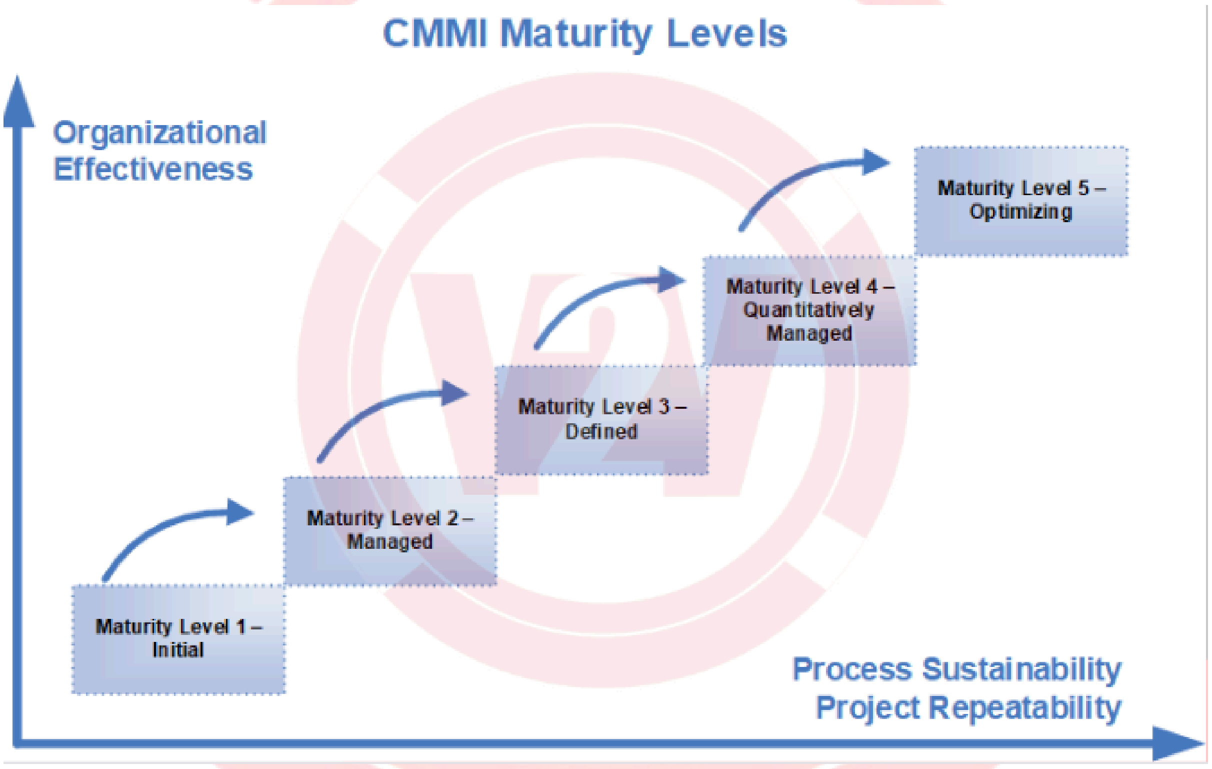
D: Design : Work out details, optimize the methods, run simulations if necessary, and plan for design verification.

V: Verify : Check the design to be sure it was set up according to plan, conduct trials of the processes to make sure that they work, and begin production or sales.

📘 **Example:** A software company applies Six Sigma to reduce the number of bugs in releases from 150 per version to 10.

📌 B. CMMI (Capability Maturity Model Integration)

🔹 **Definition:** CMMI is a **process-level improvement training and appraisal program** that guides software organizations in improving their processes.


CMMI Maturity Levels
Organizational Effectiveness
Maturity Level 1 – Initial
Maturity Level 2 – Managed
Maturity Level 3 – Defined
Maturity Level 4 – Quantitatively Managed
Maturity Level 5 – Optimizing
Process Sustainability
Project Repeatability

🔹 **CMMI Levels:**

Level	Description
1	**Initial** – Unpredictable process

Level	Description
2	**Managed** – Basic project management processes exist
3	**Defined** – Processes are documented and standardized
4	**Quantitatively Managed** – Processes are measured and controlled
5	**Optimizing** – Focus on continuous improvement

✅ **Benefits:**

- Structured and measurable process improvement
- Better risk management
- Consistent quality in projects

2. Continuous CMMI Model:

· Continuous maturity models do not classify an organization according to discrete levels. Rather they are finer-grained models that consider individuals or groups of practices.

· The maturity assessment is not, therefore, a single value but a set of values showing the organization's maturity for each process.

CMMI Levels

· The six-point scales assign a level to a process as follows:

1. Level 0: Not performed: One or more specific goals associated with the process area is not satisfied.

2. Level 1: Performed: The goals associated with the process area are satisfied and for all processes the scope of the work to be performed.

3. Level 2: Managed: At this level, the goals associated with the process areas are met and organizational policies are in place that define when each process should be used.

4. Level 3: Defined: This level focuses on organizational standardization and deployment of processes.

5. Level 4: Quantitatively Managed: At this level, there is an organizational responsibility to use statistical and other qualitative methods to control sub-processes.

6. Level 5: Optimizing: At this highest level, the organization must use the process and product measurements to drive process improvement.

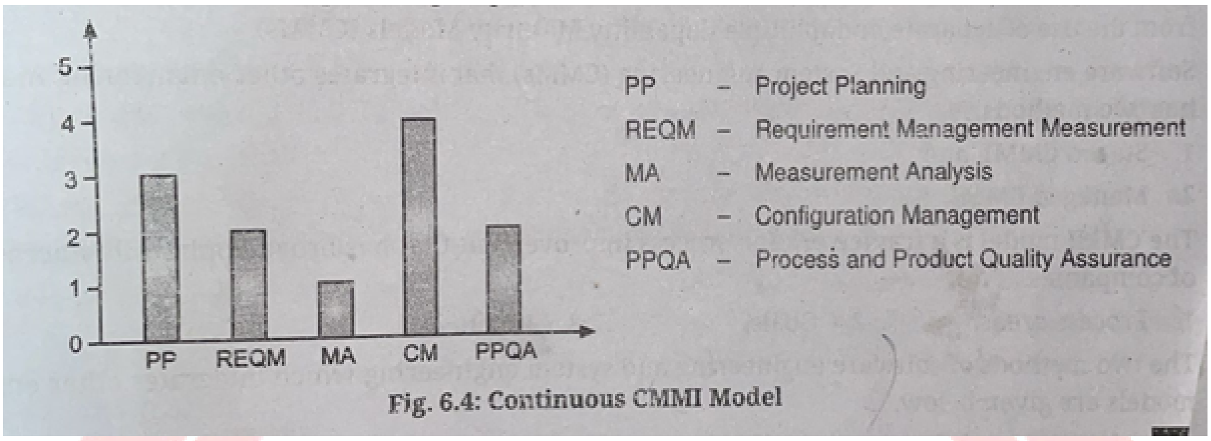


Fig. 6.4: Continuous CMMI Model

📘 **Example:** A company at **CMMI Level 3** has all project processes well documented and uses defined standards for development.

📘 Static and Dynamic Testing

✅ 1. Static Testing

🔹 **Definition:** Static Testing is a **type of software testing** that is **performed without executing the code**. It involves **reviewing documents, code, or design** to detect errors early in the development life cycle.

🔎 What is Reviewed?

- Requirement specifications
- Design documents
- Source code
- Test plans and cases

🔧 Types of Static Testing:

Type	Description
Reviews	Informal or formal checking of documents by peers
Walkthroughs	Author explains code/documents to peers for feedback
Inspections	Formal process with predefined roles to identify issues
Static Code Analysis	Automated tools used to find vulnerabilities or poor coding practices (e.g., SonarQube, Checkstyle)

✅ Benefits of Static Testing:

- Catches errors **early before execution**
- Saves time and cost of fixing bugs later
- Improves documentation and design quality
- Reduces the number of **runtime defects**

❌ Drawbacks:

- Cannot find **runtime or logical errors**
- Needs **skilled reviewers**
- Might not be suitable for all types of projects

📘 Example:

- Reviewing a login module's code and identifying a missing null check before running the software.
- Detecting poor variable naming or incorrect formula in code via code inspection.

✅ 2. Dynamic Testing

🔹 **Definition:** Dynamic Testing is the **process of testing the software by executing the code**. It is used to check the **functional behavior, performance, and output** of the application.

🔎 When is it performed?

- After the development phase, during **test execution**

🔧 Types of Dynamic Testing:

Type	Description
Unit Testing	Tests individual modules (done by developers)
Integration Testing	Checks interaction between components
System Testing	End-to-end testing of the entire system
Acceptance Testing	Validates system against business requirements

---

✅ Benefits of Dynamic Testing:

- Validates the **actual behavior** of the system
- Finds **runtime issues**, such as memory leaks, crashes, and functional errors
- Ensures **performance and usability** meet expectations

❌ Drawbacks:

- Can be **time-consuming**
- Costly if bugs are found late
- Needs proper **test data and environment**

📘 Example:

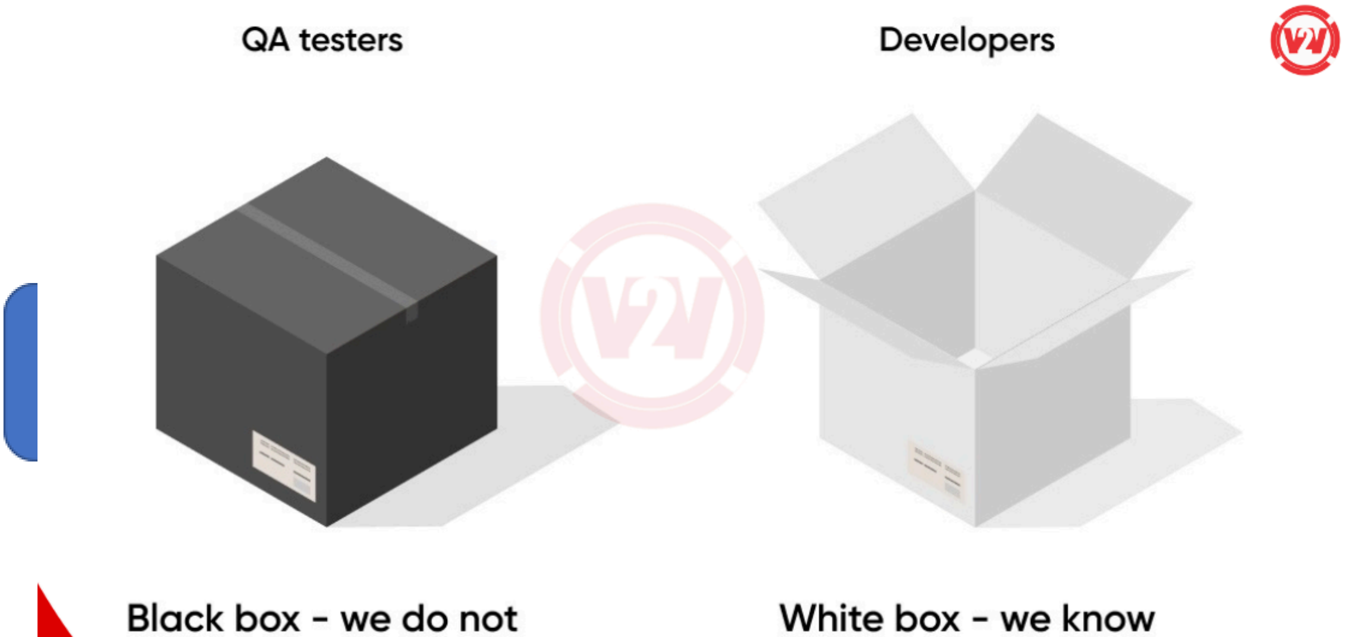
- Running the login module and entering valid/invalid inputs to verify the response.
- Testing an e-commerce app to ensure adding to cart, checkout, and payment work correctly.

🆚 Difference Between Static and Dynamic Testing

Feature	**Static Testing**	**Dynamic Testing**
Execution	**No code execution**	**Requires code execution**
Performed in	Early stages of development	After development (during/after coding)
Focus	Preventing defects	Detecting defects
Cost	**Lower** (early detection)	**Higher** (late defect discovery)
Tools Used	Code analyzers, linters, review tools	Test frameworks (JUnit, Selenium)
Example	Code review, requirement review	Unit test, UI test, performance test

📘 The Box Approaches in Software Testing

White Box Testing **vs** Black Box Testing



- In **Black Box Testing**, testers (often QA professionals) have **no knowledge of the internal code**. They focus only on **inputs and expected outputs**—like trying to guess what's inside a sealed box by shaking it.
- In **White Box Testing**, developers or technically skilled testers have **full visibility of the internal structure**. They test the **logic, code flow, and inner workings**, like examining the contents of an open box.

✅ 1. Black Box Testing

🔹 **Definition:** Black Box Testing is a **testing approach where the tester does not know the internal structure or code** of the application. The focus is entirely on the **inputs and expected outputs**.

🧠 **Think of it as testing from the outside.**
You treat the software like a "black box" — you don't care how it works internally.

🔧 Types of Black Box Testing:

- **Functional Testing** – verifies what the software does
- **Non-Functional Testing** – includes performance, usability, security
- **Regression Testing** – checks old functionalities after changes

---

✅ Advantages:

- Testers do **not need programming knowledge**
- Tests are based on **requirements**, not code
- Effective for **system-level and acceptance testing**
- Helps uncover **missing functions or usability issues**

❌ Disadvantages:

- Limited **test coverage** of internal paths
- Cannot find **hidden or logical errors in code**
- Test case creation may be difficult without clear specifications

---

📘 **Example:**

Testing the login form:

- Input: Username and Password
- Output: Login success or error message
  Tester doesn't check **how** authentication happens internally.

✅ **2. White Box Testing**

🔹 **Definition:** White Box Testing (also called **Glass Box or Clear Box Testing**) is a **testing technique that considers the internal logic, code structure, and flow** of the program.

🧠 **Think of it as testing from the inside.**
The tester knows how the software is built and tests each logic path.

🔧 **Types of White Box Testing:**

- **Unit Testing** – individual functions or methods
- **Integration Testing** – testing interaction between units
- **Code Coverage Testing** – condition, branch, and path coverage

---

✅ **Advantages:**

- Helps in **optimizing code and improving quality**
- Ensures **maximum code coverage**

- Detects **hidden errors**, logical flaws, dead code
- Useful for **security and loop testing**

❌ **Disadvantages:**

- Requires **knowledge of programming**
- Time-consuming to test all code paths
- Not suitable for **large-scale system testing** alone

📘 **Example:**

Testing a login function by:

- Checking code conditions (if, else)
- Validating hash logic for password matching
- Ensuring error messages are handled properly

🆚 **Black Box vs White Box Testing – Comparison Table**

Feature	**Black Box Testing**	**White Box Testing**
Knowledge of Code	Not required	Required
Focus	Functionality & output	Internal logic & code paths
Tester Role	End-user or independent tester	Developer or technically skilled tester
Testing Basis	Requirements and specs	Code structure and logic
Detects	Missing functionality, interface issues	Logical errors, hidden bugs, code coverage

Feature	Black Box Testing	White Box Testing
Tools Used	Selenium, QTP, LoadRunner	JUnit, NUnit, SonarQube, Code coverage tools
Test Level	System, Acceptance	Unit, Integration
Time Required	Less (in general)	More (detailed code-level testing)

🧪 Levels of Testing

Software testing is performed in **multiple levels** to ensure quality at every stage of development. The major levels are:

---

1 Unit Testing

🔹 **Definition**: Unit testing involves testing **individual components or functions** of the software (like methods, classes, or modules).

🔹 **Objective**: To verify that each unit of the software works as expected in isolation.

🔹 **Who performs it?**
Typically done by **developers**.

🔹 **Advantages**:

- Fast and automated.
- Easier to debug.
- Helps catch bugs early in development.

🔹 **Disadvantages**:

- Doesn't test integration.

- Requires understanding of the internal code.

- **Example**: Testing a calculateTotal() function in an online shopping app.

---

2 Integration Testing

- **Definition**: Tests how **different modules or units** work together when combined.
- **Objective**: To ensure data flow and control between modules is working correctly.
- **Who performs it?**

Developers or testers.

- **Approaches**:
  - Top-down
  - Bottom-up
  - Big Bang
  - Sandwich

- **Advantages**:
  - Catches interface-level bugs.
  - Ensures modules interact correctly.

- **Disadvantages**:
  - Complex setup.
  - Hard to isolate defects.
- **Example**: Testing login module integration with user database and session manager.

---

3 System Testing

- **Definition**: Testing the **entire software system as a whole** to ensure it meets the specified requirements.

🔹 **Objective**: To validate the complete functionality and behavior of the system.

🔹 **Who performs it?**
Independent testing team (QA).

🔹 **Types**:

- Functional Testing
- Non-functional Testing (e.g., performance, security)

🔹 **Advantages**:

- Tests the entire application in real environment.
- Ensures business requirements are fulfilled.

🔹 **Disadvantages**:

- Time-consuming.
- Cannot find internal code issues.

🔹 **Example**: Testing an entire e-commerce platform including login, cart, payment, and logout.

---

4 Acceptance Testing

🔹 **Definition**: Final level of testing to determine whether the software is **ready for release** and meets **customer expectations**.

🔹 **Objective**: To get client or end-user approval.

🔹 **Who performs it?**
Clients, stakeholders, or end-users.

- **Types**:
  - Alpha Testing (done internally)
  - Beta Testing (done by external users)

- **Advantages**:
  - Validates software in real use-case.
  - Builds client confidence.

- **Disadvantages**:
  - Feedback might be subjective.
  - Not all edge cases may be tested.

- **Example**: Client testing a new CRM tool before deploying it in their company.

🧠 **Summary Table**

Level	Tested By	Scope	Purpose
Unit Testing	Developers	Individual functions/modules	Check correctness of code logic
Integration Testing	Developers/Testers	Interactions between modules	Check data flow & interactions
System Testing	Testers (QA)	Whole system	Ensure system meets requirements
Acceptance Testing	Clients/Users	Business scenarios	Validate customer satisfaction