**Quiz 2**

**Chapter 3**

### 🔹 1. **Requirements Analysis**

This is the first step where we talk to the customer or users to understand **what they need** from the software.

**Example**: If we are making an online food delivery app, we ask:

* What features do you want? (e.g., order food, track delivery, make payments)
* Who will use it? (customers, delivery riders, restaurants)

### 🔹 2. **Designing / Modeling**

Now that we know what the software should do, we **plan how it will work**. We design how the app will look and how the different parts will connect.

**Example**: In the food app:

* What buttons will be on the home screen?
* How will the user search for food?
* What happens when someone clicks "Order"?

### 🔹 3. **Coding / Development**

This is where **programmers write the code** to build the actual software using programming languages like Java, Python, etc.

**Example**: Developers create features like:

* Login system
* Menu display
* Payment system

### 🔹 4. **Testing**

Before launching, we **test the software** to find and fix any bugs or errors.

**Example**:

* Does the app crash when placing an order?
* Are payments working properly?

### 🔹 5. **Implementation / Integration**

Now, we **install the software and connect it with other systems** if needed. It's made ready for real users.

**Example**: The food app goes live in app stores and is connected to restaurant systems and delivery services.

### 🔹 6. **Operation / Maintenance**

After launch, we **keep the app running smoothly**. We fix any new bugs and add updates based on user feedback.

**Example**:

* Add a new feature like "Live Chat"
* Fix bugs like app freezing

### 🔹 7. **Documentation**

We create **user manuals and technical guides** to help users and future developers understand the software.

**Example**:

* A guide for users on how to order food
* A document for developers explaining how the login system works

### 🌟 Summary Table:

| **Phase** | **What Happens?** | **Example (Food App)** |
| --- | --- | --- |
| Requirements Analysis | Understand what is needed | Ask what features users want |
| Designing / Modeling | Plan the structure and look | Draw app screens and plan features |
| Coding / Development | Write the actual code | Build login, order, and payment systems |
| Testing | Find and fix bugs | Check if order and payment works |
| Implementation / Integration | Launch and connect the app | Release on app stores, link to systems |
| Operation / Maintenance | Keep the app working well | Fix bugs, add new features |
| Documentation | Write guides and instructions | Create user manual and developer guide |

## 🌟 Capability Maturity Model (CMM)

| **Level** | **Name** | **Simple Meaning** | **Example** |
| --- | --- | --- | --- |
| **1** | Initial | No plan, everyone works in their own way | A developer writes code without any rules |
| **2** | Repeatable | Some steps are followed and repeated | Team uses the same plan for every project |
| **3** | Defined | One standard process for all teams | All teams follow the same software rules |
| **4** | Managed | Progress is tracked with data | They count bugs and improve based on numbers |
| **5** | Optimizing | Always improving the process | New ideas and tools are added to get better |

## ✅ **Test Process Improvement (TPI)** – Easy Version

### 🔍 **What is a Test Process?**

A **test process** is the way we check software for bugs or problems.  
It includes activities like:

* Setting test goals
* Planning how to test
* Choosing test types (manual, automated)
* Hiring testers
* Writing test cases
* Giving test tasks to testers
* Running the tests
* Reporting bugs
* Using test tools

## 🔧 **How to Improve the Test Process (TPI Steps):**

| **Step** | **What to Do** | **Simple Example** |
| --- | --- | --- |
| **1** | **Find what needs improvement** | "Testing takes too long." |
| **2** | **Check the current situation** (baseline) | "Right now, testing takes 5 days." |
| **3** | **Decide the desired state and how to get there** | "We want to finish testing in 3 days using automation." |
| **4** | **Make the changes** | "Use better tools and train the testers." |

## 🎯 Goal of TPI:

* **Make testing faster**
* **Find more bugs**
* **Use better tools**
* **Save time and money**

## ✅ **What is TMM (Testing Maturity Model)?**

The **Testing Maturity Model (TMM)** is a framework that helps an organization **check how strong its software testing process is** and **improve it step by step**.

It is similar to the **Capability Maturity Model (CMM)**, but **TMM focuses only on testing** activities. As a company moves through the TMM levels, its testing becomes more organized, effective, and high quality.

### ✅ **TMM (Testing Maturity Model) – Levels Table**

* **TMM** was developed by **Ilene Burnstein** to help organizations **evaluate and improve testing processes**.
* It has **5 levels of maturity**, from unstructured (Level 1) to optimized and data-driven (Level 5).
* Each level includes:
  + **Maturity goals**
  + **Supporting goals**
  + **Activities, tasks, and responsibilities (ATRs)** — from the view of managers, developers, testers, and customers.
* **Main goal**: Help teams **grow step by step** in testing until it becomes a **well-planned, measured, and optimized** part of development.

| **Level** | **Name** | **What It Means** | **Example** |
| --- | --- | --- | --- |
| **1** | **Initial** | No testing goals. Testing starts after coding. Random and not tracked. | Developers test without a plan, just to see if it works. |
| **2** | **Phase Definition** | Set test goals, start planning, define risks and strategies. | Team creates a test plan and prepares basic test cases. |
| **3** | **Integration** | Build a test team, give training, and include testing in all stages. | Testers join early, testing is part of the whole development process. |
| **4** | **Management & Measurement** | Manage and review testing. Measure software quality using data. | Team tracks bugs, reviews work, and checks quality. |
| **5** | **Optimization / Defect Prevention** | Use data to stop bugs early. Improve the process continuously. | Use defect data, automate tests, and apply quality control techniques. |

## ✅ Summary Table

| **Model** | **Focus Area** | **Purpose** | **Levels?** | **Example Use** |
| --- | --- | --- | --- | --- |
| **CMM** | Full software process | Improve software development overall | Yes | Improving how a company builds software |
| **TPI** | Only testing | Improve test process area-by-area | No fixed levels | Focused help on weak test areas |
| **TMM** | Only testing | Evaluate and improve test process step-by-step | Yes | Make testing more mature and standard |

**1. Defect Prevention**

**What it means:**  
These activities try to stop mistakes (errors) before they happen in the software. The goal is to make sure developers don’t create bugs in the first place.

**Example:**

* Giving users drop-down menus or buttons instead of making them type things. This reduces the chance of wrong input.
* Setting default values in forms so users don’t leave fields empty or enter wrong data.

**Simple idea:**  
**"Let’s avoid the problem before it even happens."**

**2. Defect Reduction**

**What it means:**  
These activities find and fix bugs after they’ve been created but before the software is released to users.

**Example:**

* Testing the software to find bugs (like unit testing or system testing).
* Code reviews where one developer checks another developer’s work.

**Simple idea:**  
**"Find and fix the problem before users see it."**

**3. Defect Containment**

**What it means:**  
These activities make sure that if there is a bug, it doesn’t crash the whole system or cause serious problems. It focuses on handling problems safely.

**Example:**

* Auto-save feature: Even if the program crashes, your work is saved.
* Backup systems: If one part fails, another can take over.

**Simple idea:**  
**"If a problem happens, make sure it doesn’t cause big damage."**

**1. Defect Prevention**

**Goal:** Stop mistakes from happening in the first place.

**How it's done:**

* **Remove the source of the error:**  
  Teach developers and testers properly to avoid misunderstandings (like giving training to reduce human errors).
* **Block errors directly:**  
  Use rules or checks in the system to prevent wrong actions.  
  **Example:** You can’t divide by zero in a form because the system blocks it.
* **Improve the development process:**  
  Use tools and methods (like formal design or coding standards) that help people write correct software from the start.

**Simple Examples:**

* Default values in forms
* Dropdown menus instead of text input
* Training the team to avoid known mistakes
* Prevent user from typing invalid data

**2. Defect Reduction**

**Goal:** Find and fix bugs that have already been added to the software.

**How it's done:**

* **Code Reviews/Inspections:**  
  Developers or teams read and check each other’s work to find problems early.
* **Testing:**  
  Run the program and check if everything works as expected. If something breaks, find and fix the issue.
* **Analysis Techniques:**  
  Use methods like:
  + **Boundary value testing:** Checking edge cases (e.g. minimum or maximum input)
  + **Control flow/data flow testing:** Checking how data moves through the program
  + **Simulation/prototyping:** Try out a smaller version of the system to test features

**Simple Examples:**

* Review code with a teammate
* Test login with empty username or special characters
* Check if a loop works for 0, 1, and 100 items

**3. Defect Containment**

**Goal:** Control the effect of bugs so they don’t cause big failures or user problems.

**Why it’s needed:**  
Even after prevention and reduction, some bugs still exist. This method handles them safely.

**How it's done:**

* **Fault tolerance:**  
  System keeps working even if part of it fails.  
  **Example:** If one function crashes, the rest still runs.
* **Recovery:**  
  System can undo or redo actions if something goes wrong.  
  **Example:** Undo button or auto-save.
* **N-Version Programming (NVP):**  
  Using multiple versions of a program to reduce the chance of all failing at once.
* **Safety Assurance:**  
  Identify dangerous situations and make sure they don’t cause accidents.  
  **Example:** In autopilot, the system corrects pilot mistakes automatically.

**Simple Examples:**

* Auto-save in Word
* Try-catch blocks in code for error handling
* Backup system if one part fails
* Warnings or safety checks before deleting important data

**@@@&**

**1. Mega-Process of Software**

This is the **big picture** of how software is made from start to end.

| **Stage** | **Meaning (Easy Words)** |
| --- | --- |
| **Initiation** | Deciding to start the project. Understand what the client or users want. |
| **Development** | Building the software (design, coding, testing). |
| **Maintenance** | Fixing bugs, updating features, or improving performance after the software is used. |
| **Termination** | Ending the project when it's no longer needed. |

**2. Development Components**

These are the **steps inside the development phase**:

| **Step** | **Easy Meaning** |
| --- | --- |
| **Requirement** | Collecting what the users want (e.g. “I want a login page”). |
| **Specification** | Writing down exactly what the system should do. |
| **Design** | Planning how the software will look and work (e.g. screen layout, data flow). |
| **Coding** | Writing the actual program. |
| **Testing** | Checking if the program works properly. |
| **Release** | Giving the final product to the users. |

**3. Process Variations**

These are **different ways** to develop software.

| **Process** | **Description (Easy Words)** |
| --- | --- |
| **Waterfall** | Step-by-step process (finish one step completely before moving to the next). |
| **Iterative** | Build a small part, test it, then add more in cycles (QA is done in each part). |
| **Spiral** | Mix of design, build, test, and risk checking in loops. Focus on **risk and QA**. |
| **Agile (e.g. XP)** | Fast, flexible, team-based. Uses **test-first** (write tests before code), and **pair programming** (2 people code together). |
| **Maintenance Process** | Focus on fixing bugs, updating features, or making the system better **after release**. |

**@@@@@**

**QA in Waterfall Software Process**

| **Stage** | **QA Focus** | **What It Means (Simple Words)** |
| --- | --- | --- |
| **Requirement & Specification** | **Defect Prevention** | Make sure everything is clearly understood and written down. Avoid confusion or mistakes later. |
| **Design** | **Defect Prevention** | Plan the system properly to avoid problems in coding. Catch issues before they become big. |
| **Coding** | **Defect Removal** | Write clean code and review it. Use tools and peer reviews to find and fix bugs early. |
| **Testing** | **Defect Containment** | Test the software to find problems. Use error handling to stop one bug from crashing the system. |
| **Release & Support** | **Defect Containment** | Monitor real-world use. Fix issues fast. Make sure bugs don’t hurt the users or system. |

**Summary in Easy Words:**

* **Early stages** (Requirement, Design): Try to **prevent** problems.
* **Middle stage** (Coding): **Remove** problems while building.
* **Later stages** (Testing, Release): **Handle** problems smartly if they happen.