Unit - V Test and Defect Management

📘 Topic: Test Planning – Preparing a Test Plan

✅ **Definition: Test Planning** is the process of defining the scope, approach, resources, and schedule of intended testing activities.
It results in the creation of a **Test Plan** document, which acts as a blueprint for the testing process.

📝 What is a Test Plan?

A **Test Plan** is a formal document that outlines:

- What to test
- How to test
- When to test
- Who will test

It ensures clarity, resource allocation, timeline definition, and proper control over testing.

🔧 Key Components of a Test Plan:

Section	Description
Test Plan ID	Unique identifier for the plan
Introduction	Overview and objective of testing
Scope	What features will and won't be tested
Test Strategy	Testing approach: manual, automation, tools used
Test Environment	Hardware, software, network setup needed
Test Items	Modules or parts of the system to be tested
Test Deliverables	Reports, test cases, defect logs to be produced
Schedule	Timeframe for test activities
Roles and Responsibilities	Who does what in testing
Entry and Exit Criteria	Conditions to start/stop testing
Risks and Mitigation	Possible issues and how to handle them

🛠️ Steps to Prepare a Test Plan:

1. Understand the project requirements and goals
2. Identify scope and objectives of testing
3. Select testing approach (Black Box, White Box, etc.)
4. Identify required resources (team, tools, time)
5. Define entry and exit criteria
6. Estimate test effort and create a schedule

7. Assign responsibilities
8. Document and review the test plan
9. Get approval from stakeholders

⭐ **Benefits of Test Planning:**

- Provides a **clear roadmap** for testing
- Ensures **better resource management**
- Helps in identifying **risks early**
- Improves **test coverage and quality**
- Facilitates **communication among team members**

⚠️ **Drawbacks (if not done properly):**

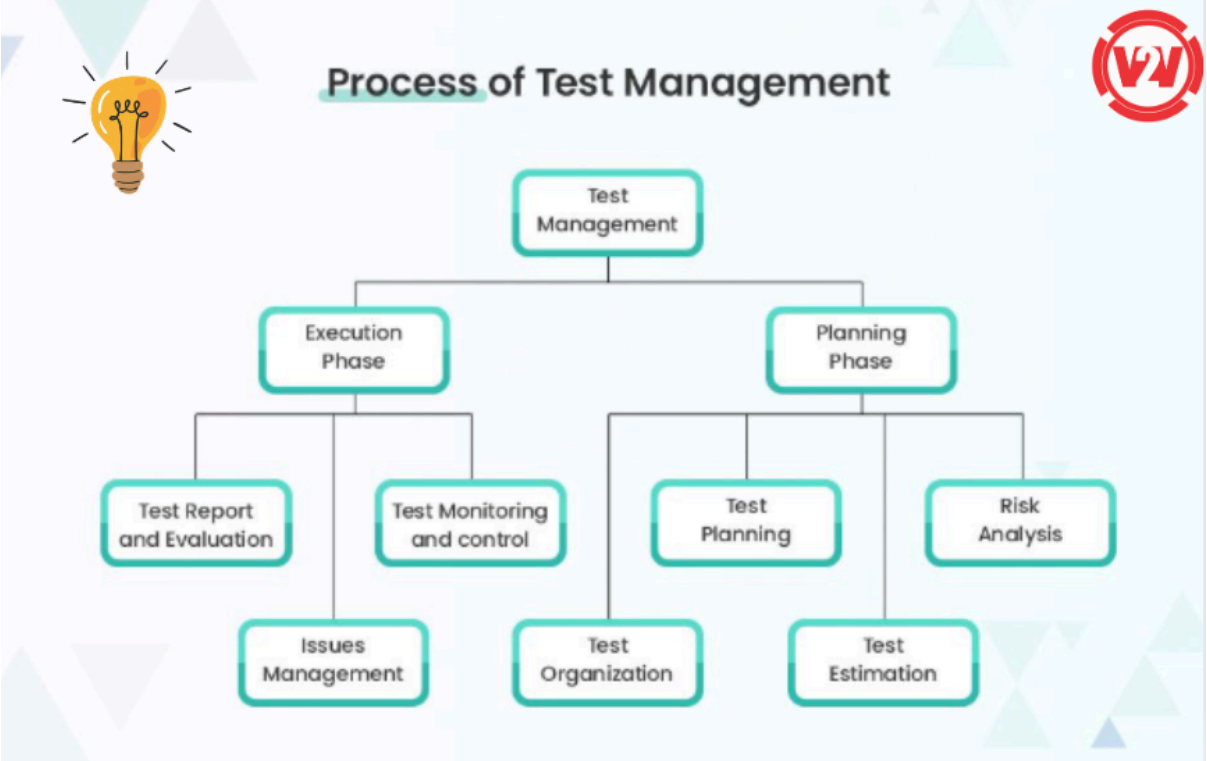
- Can become **outdated** in fast-changing projects
- May lead to **incomplete testing** if scope is missed
- **Rigid plans** may affect agility

🧩 **Test Management: Test Infrastructure Management**

✅ **Definition:**

Test Infrastructure Management refers to the setup, maintenance, and management of the hardware, software, tools, environments, networks, and data required to perform effective software testing.

It ensures that the **testing environment replicates the production environment** as closely as possible for accurate results.



🛠️ Key Elements of Test Infrastructure:

Component	Description
Test Environment	Physical or virtual systems where testing is executed (OS, servers, networks)
Test Tools	Tools used for automation, defect tracking, performance testing, etc.
Test Data	Dummy or mock data prepared to execute test cases
Version Control	Tools like Git to manage code and configuration versions
Build Management	CI/CD tools (e.g., Jenkins) to manage test builds
Network Configuration	Mimicking production network environments (firewalls, bandwidth, etc.)

⚙️ **Responsibilities of Test Infrastructure Management:**

1. **Set up test environments** according to system requirements
2. **Install and configure tools** needed for testing (like Selenium, JIRA, TestNG)
3. **Provide secure and stable environments** for testers and developers
4. **Manage test data** preparation, refresh, and masking
5. **Monitor performance** of infrastructure to avoid bottlenecks
6. **Maintain test environments** in sync with production environments
7. **Control access** and permissions for users in different roles

---

🌟 **Benefits:**

- Enables **consistent, reliable, and repeatable testing**
- Reduces **environment-related failures**
- Helps in early **defect identification**
- Improves **efficiency of automation** and continuous testing
- Ensures **scalability** and flexibility for different testing types (unit, load, integration)

⚠️ **Challenges / Drawbacks:**

- **High cost** of maintaining multiple environments
- **Time-consuming setup** if automation isn't used
- Risk of **configuration mismatches** between environments
- **Dependency issues** between hardware, software, and networks

🧾 **Test Reporting: Executing Test Cases & Preparing Test Summary Report**

✅ 1. Executing Test Cases

🔍 Definition:

Test Case Execution is the process of running a set of predefined test cases on the software application to verify whether it meets the specified requirements.

📋 Steps in Test Case Execution:

1. **Select test cases** from the test plan.
2. **Set up the test environment** (hardware, software, data).
3. **Execute test cases manually or via automation tools.**
4. **Log results**: Pass, Fail, Blocked, or Skipped.
5. **Record actual results** and compare them with expected results.
6. **Raise defects/bugs** for any failed test cases.

📌 Best Practices:

- Maintain traceability between requirements and test cases.
- Log test execution results clearly.
- Update test cases if changes occur in requirements.
- Automate where feasible for regression tests.

🧪 Example:

Test Case ID	Description	Expected Result	Actual Result	Status
TC_01	Login with valid input	Dashboard opens	Dashboard opens	Pass

Test Case ID	Description	Expected Result	Actual Result	Status
TC_02	Login with invalid pwd	Error message shown	No message shown	Fail

---

📊 2. Preparing Test Summary Report

📄 **Definition:** A **Test Summary Report** (TSR) is a high-level document that summarizes the overall testing activities, results, defect statistics, and evaluation of the testing process.

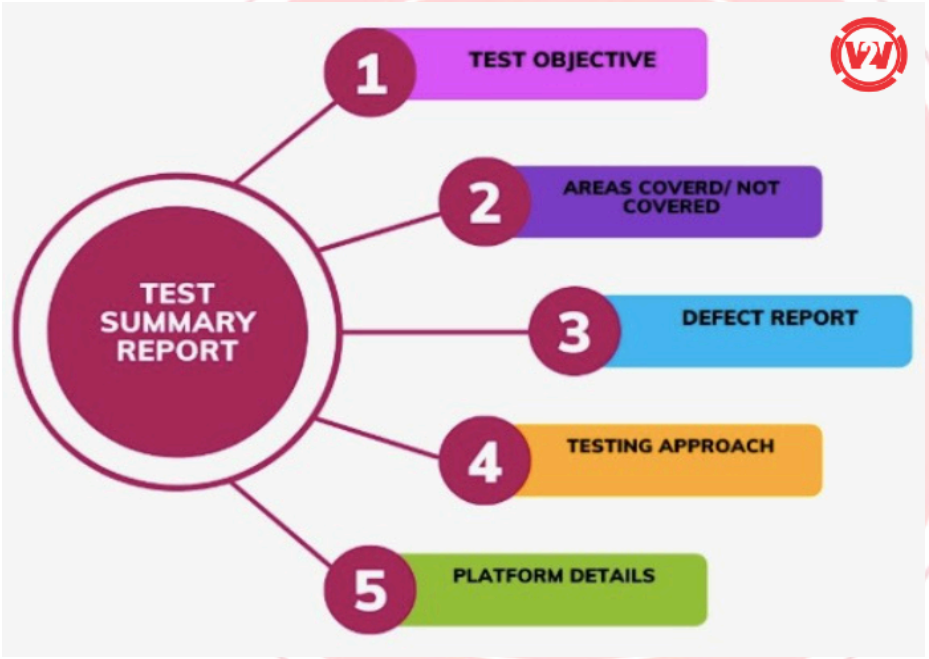
📚 Contents of a Test Summary Report:

Section	Description
Report Identifier	Project Name, Date, Report Version
Objective	Purpose and scope of testing
Test Items	Modules/features tested
Environment	Hardware, OS, test tools used
Test Results	No. of test cases passed/failed/skipped
Defects	Summary of raised and resolved defects
Risks	Outstanding risks or concerns
Recommendations	Next steps (e.g., ready for release?)

📈 Example Summary (Table):

Metric	Value
Total Test Cases Executed	100
Passed	85

Metric	Value
Failed	10
Blocked/Not Run	5
Total Defects Raised	12
Critical Defects Resolved	10
Pending Critical Defects	2



✅ **Benefits of Test Reporting:**

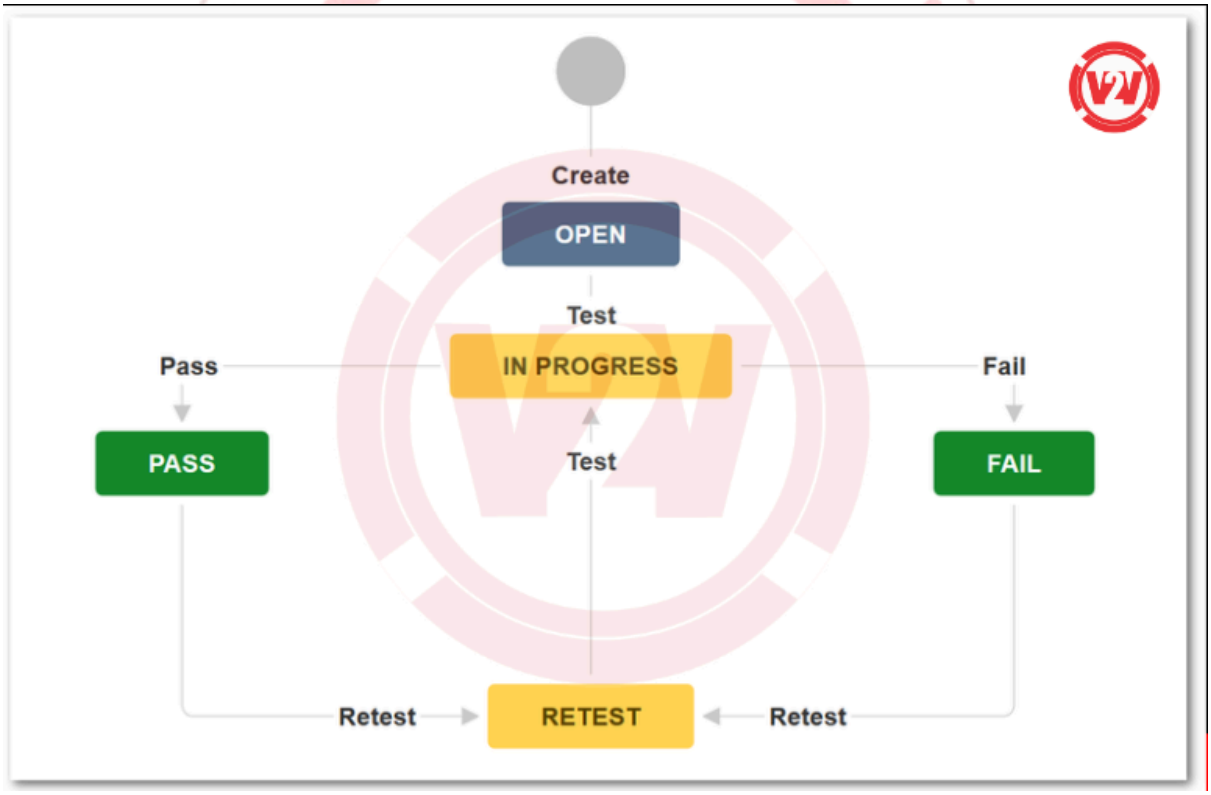
- Gives stakeholders a **clear view of testing progress**
- Helps in **release decisions**
- Ensures **accountability** and traceability
- Documents **quality metrics** for future audits

⚠️ **Drawbacks / Challenges:**

- Time-consuming to prepare without proper tooling
- May miss key insights if not formatted correctly
- Requires detailed **data logging during execution**

🛠️ **Tools for Test Execution & Reporting:**

- **TestRail, HP ALM, JIRA + Zephyr, qTest**
- **Automation tools**: Selenium, TestNG (with reports)



Diagramatical representation of Test Case Execution

🐞 Definition and Types of Defect, Defect Life Cycle, Defect Template

---

🗒️ 1. Definition of a Defect

A **defect** is a **flaw, bug, or error** in a software application that causes it to behave in an unintended or unexpected way. A defect occurs when the actual result deviates from the expected result.

🧠 **In simple words:** When a tester finds an issue in the software that doesn't meet the requirements, it's called a defect.

🧩 2. Types of Defects

🛠️ Type	📋 Description
Functional Defect	Application doesn't behave as per functional requirements

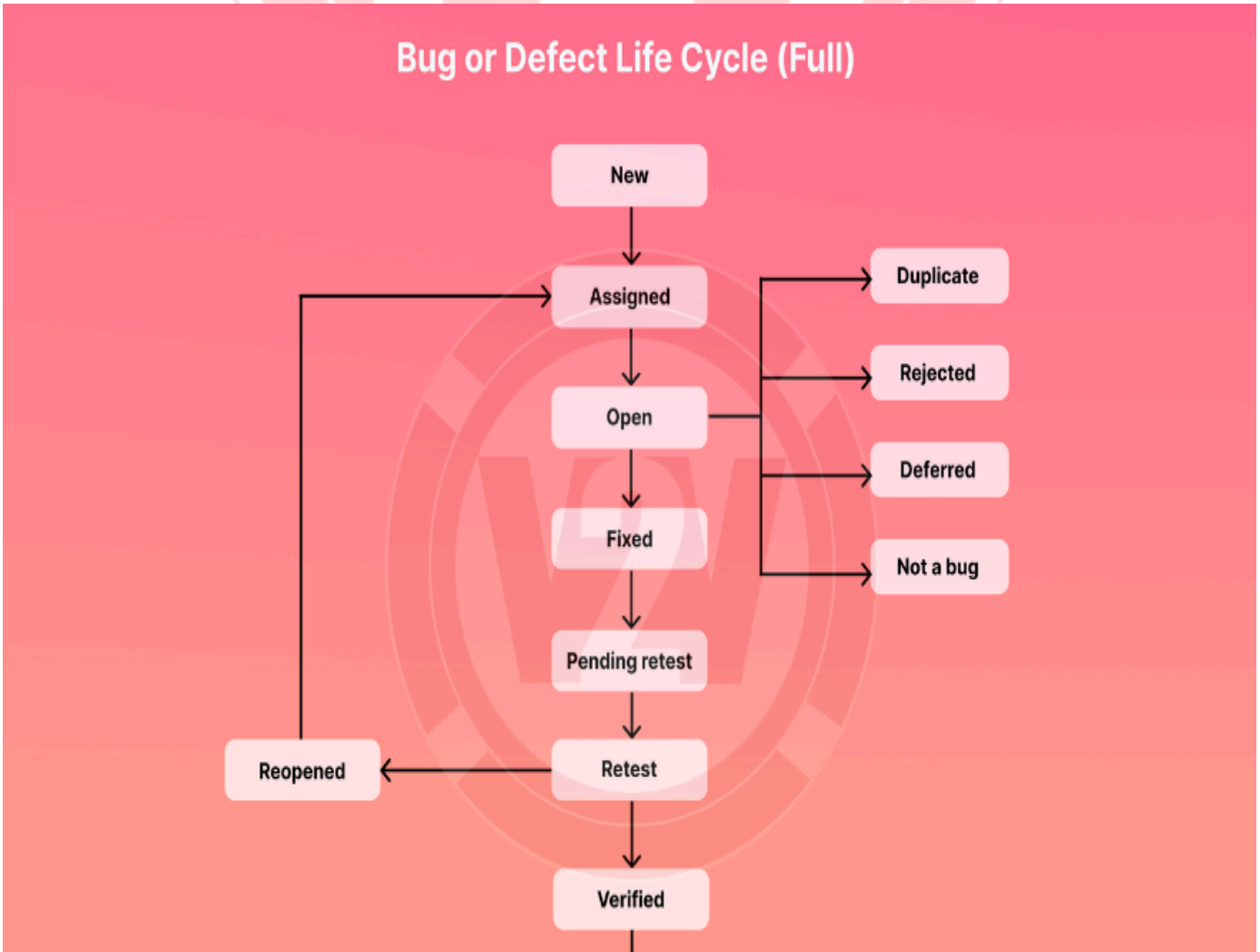
🛠️ Type	📋 Description
Performance Defect	Software fails to perform within acceptable limits (e.g., slow response)
Security Defect	Application fails to protect data or is vulnerable to unauthorized access
Compatibility Defect	Application doesn't work properly across different devices, OS, browsers
Usability Defect	Application is difficult or confusing to use

---

🔄 3. Defect Life Cycle (Bug Life Cycle)

The **Defect Life Cycle** is a series of stages that a defect goes through from its detection to its closure.

🔁 Stages of the Defect Life Cycle:



🔁 **Stages of the Defect Life Cycle:**

1. **New** – Defect is logged for the first time.
2. **Assigned** – Assigned to a developer for analysis/fix.
3. **Open** – Developer has started working on it.
4. **Fixed** – Defect is resolved by the developer.
5. **Retest** – Tester retests the defect.
6. **Verified** – Tester confirms the defect is fixed.
7. **Closed** – Final stage; defect is no longer reproducible.
8. **Reopened** – If the issue reappears after being marked fixed.
9. **Deferred** – Fix is postponed for future release.
10. **Rejected/Not a Bug** – Developer considers it not a valid defect.

📄 **4. Defect Template (Bug Report Template)**

A **Defect Template** is a standardized format used to report and track bugs.

📋 **Common Fields in a Defect Report:**

🏷️ Field	📝 Description
Defect ID	Unique identifier for the defect
Title/Summary	Short title summarizing the defect
Description	Detailed explanation of the defect
Steps to Reproduce	Exact steps to replicate the issue

🏷️ Field	📝 Description
Expected Result	What should happen
Actual Result	What actually happens
Severity	How serious is the defect (Critical, Major, Minor)
Priority	How urgently it needs to be fixed (High, Medium, Low)
Environment	OS, browser, device details, version numbers
Attachments	Screenshots, logs, videos if any
Reported By	Name of the tester who logged the defect
Status	Current stage in defect life cycle
Assigned To	Developer responsible for fixing it
Date Reported/Updated	Timestamp of logging and updates

🧪 **Example Entry:**

Field	Value
Defect ID	BUG_101
Title	Login fails with valid credentials
Description	The user cannot log in even with correct ID/pwd
Steps to Reproduce	1. Go to login page 2. Enter valid details 3. Click Login
Expected Result	User should be redirected to dashboard

Field	Value
Actual Result	Error message shown: "Invalid credentials"
Severity	High
Priority	Critical
Status	New
Assigned To	dev_john

✅ **Benefits of Defect Management:**

- Ensures **quality control** during software development.
- Helps **prioritize** critical issues.
- Improves **team communication** between testers and developers.
- Enables **tracking and accountability** for each defect.

🔍 **Comparison of Manual Testing and Automation Testing**

✅ **Definition:**

Type	Description
Manual Testing	The process of manually executing test cases without using any automation tools to identify bugs or issues in software.
Automation Testing	The process of executing test cases automatically using testing tools and scripts to compare actual outcomes with expected results.

📊 **Comparison Table:**

Aspect	Manual Testing	Automation Testing
Execution	Done by human testers manually	Performed using scripts and tools
Accuracy	Prone to human error	High accuracy and consistency
Speed	Slower as it requires time and effort	Faster execution, especially for repeated tests
Cost	Low initial cost but high in long run	High initial cost but cost-effective over time
Reusability	Test cases must be re-executed each time	Test scripts can be reused multiple times
Best suited for	Exploratory, usability, and ad-hoc testing	Regression, load, and performance testing
Maintenance	No maintenance needed for test scripts	Requires regular script updates with software changes
Human Observation	Can assess look and feel, UI responsiveness, etc.	Limited in visual and user experience assessments
Tools Required	None (maybe Excel for test cases)	Requires tools (e.g., Selenium, JUnit, QTP, TestNG)
Skill Set	Basic domain knowledge and test case understanding	Requires knowledge of programming, tools, and scripts

---

🌟 **Benefits of Manual Testing:**

- Good for **UI testing and exploratory testing**
- Helps uncover **unexpected issues**
- Useful in **early development stages**

- Allows **human insight** and quick feedback

🌟 **Benefits of Automation Testing:**

- Saves time on **regression and repeated testing**
- Enables **parallel and batch testing**
- Improves **test coverage and reliability**
- Easily integrated into **CI/CD pipelines**

⚠️ **Drawbacks of Manual Testing:**

- **Time-consuming** and **less reliable** for repetitive tests
- Not suitable for **performance or load testing**
- Difficult to scale for **large projects**

⚠️ **Drawbacks of Automation Testing:**

- **Expensive and time-consuming** initial setup
- Requires skilled testers with **programming knowledge**
- **Not effective** for ad-hoc or exploratory testing

📐 **Metrics and Measurement**

Metrics are essential tools for making data-driven decisions.
By applying product and process metrics, organizations can:

- **Track progress**
- **Ensure quality**
- **Improve performance**

- **Control costs**

✅ **Definition: Software Metrics** are standard measures used to quantify the attributes of software processes, products, and projects.
They help evaluate quality, productivity, efficiency, and performance.

🎯 **Purpose of Metrics:**

- Monitor software development process
- Identify risks and inefficiencies
- Improve product quality
- Estimate project cost, effort, and duration
- Ensure timely delivery

📊 **Types of Metrics**

There are 3 main categories:

Type	Description
Product Metrics	Measure characteristics of software products like size, complexity, design, performance, and quality.
Process Metrics	Measure the efficiency and effectiveness of the development process (e.g., time, effort, defect rates).
Project Metrics	Measure project-specific attributes such as cost, effort, schedule, productivity, and staffing.

🧱 Product Metrics

Product metrics are **quantitative measurements** related to the **software product** itself, such as its size, complexity, design, or performance.
These help evaluate the **quality and functionality** of the software.

✨ Examples:

Metric	Description
Lines of Code (LOC)	Total number of lines written (used for effort and size estimation).
Cyclomatic Complexity	Measures decision-making complexity in code.
Function Points (FP)	Measures software size based on features and functionality.
Code Coverage	Percentage of code executed during testing.
Defect Density	Number of defects per unit size (e.g., per KLOC).

🧠 **Purpose:**

- Assess product quality
- Determine maintainability
- Predict fault-proneness
- Identify areas needing refactoring

⚙️ Process Metrics

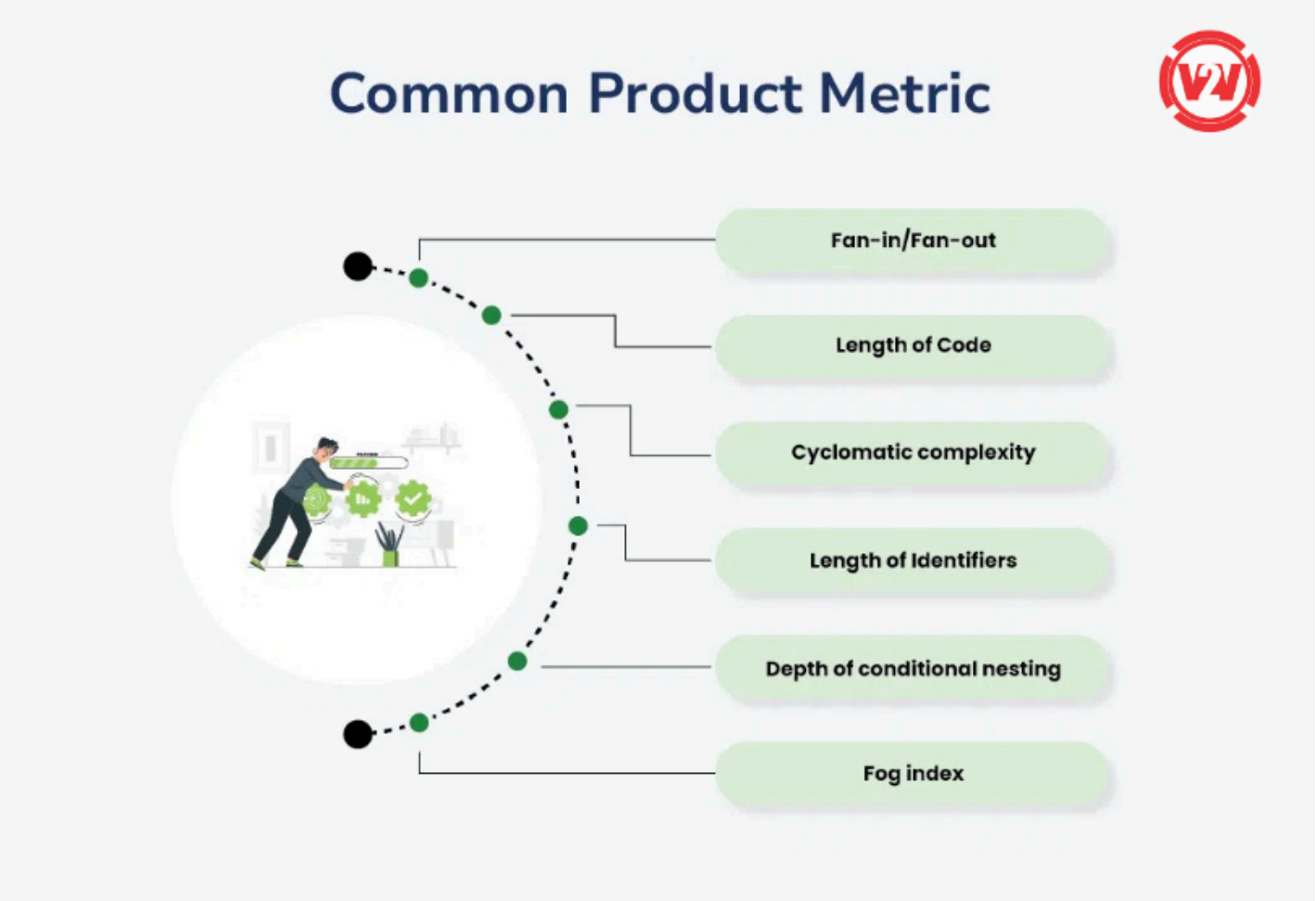
Process metrics are **quantitative measures** related to the **software development process**. They are used to assess the **efficiency, effectiveness, and improvement areas** of the processes followed during software development and maintenance.

✨ **Examples:**

Metric	Description
Defect Removal Efficiency (DRE)	% of defects removed before release.
Mean Time to Failure (MTTF)	Average time between system failures.
Review Efficiency	Number of defects detected during reviews.
Test Effectiveness	Number of defects detected during testing vs total defects.

🧠 **Purpose:**

- Improve development and testing processes
- Reduce cost and effort
- Enhance team productivity
- Optimize timelines and quality



A Common Product Metric Includes

📌 **Comparison Table:**

Aspect	Product Metrics	Process Metrics
Definition	Measure the characteristics of software products	Measure the characteristics of software development processes
Focus	On the software **itself** (code, design, documentation)	On the **processes** used to develop and maintain software
Objective	Ensure software quality, maintainability, and performance	Improve development efficiency and reduce process flaws
Examples	- Lines of Code (LOC) - Cyclomatic Complexity - Function Points - Code Coverage	- Defect Density - Mean Time to Failure (MTTF) - Defect Removal Efficiency (DRE) - Review Efficiency
Usage Stage	During and after product development	During project planning, development, and maintenance
Beneficiaries	Developers, Testers, Maintenance Team	Project Managers, Quality Assurance Teams
Measurement Units	Size, time, complexity, functionality	Time, effort, defect rates, efficiency
Key Benefit	Helps assess and improve product quality	Helps optimize the software process and productivity