innovate achieve lead

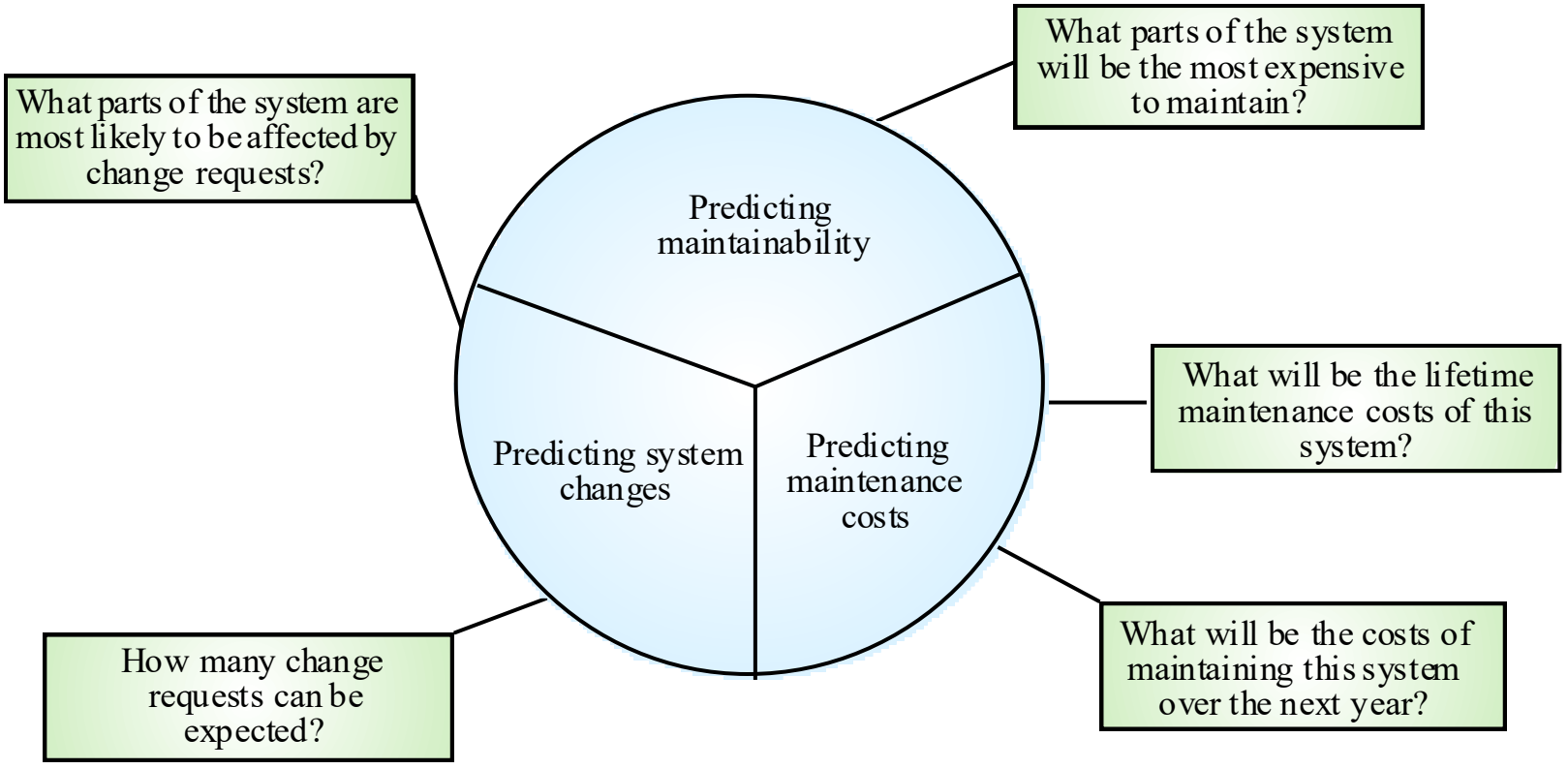
BITS Pilani
Pilani Campus

SE ZG501
Software Quality Assurance and Testing
Module 7

Maintenance Prediction

- Identifies **parts of the system** that may cause problems and be costly to maintain.
- The **ease of accepting changes** depends on how maintainable the system is.
- Making changes can make the system harder to maintain over time.
- **More changes lead to higher maintenance costs**.
- The cost of changes depends on how easy the system is to maintain.

Maintenance Prediction



Maintenance Complexity Metrics

- Predictions of maintainability can be made by assessing component complexities.
- Most maintenance efforts only affect a small number of system components.
- Maintenance complexity depends on
  - complexity of control structures
  - complexity of data structures
  - module size

Maintenance Process Metrics

- Maintainability measurements
  - Number of requests for corrective maintenance
  - Average time required for impact analysis
  - Average time to implement a change request
  - Number of outstanding change requests
- If any if these increases it may signal a decline in maintainability

Maintenance Tools

- Text editors (better than punch cards).
- File comparison tools.
- Compilers and linkage editors.
- Debugging tools.
- Cross reference generators.
- Complexity calculators.
- Control Libraries.
- Full life cycle CASE tools.

Configuration Management

- Software changes are inevitable.
- One goal of software engineering is to improve how easy it is to change software.
- Configuration management is all about change control.
- Every software engineer has to be concerned with how changes made to work products are tracked and propagated throughout a project.
- To ensure quality is maintained the change process must be audited.

Software Configuration Items

- Computer programs
  - source
  - executable
- Documentation
  - technical
  - user
- Data
  - contained within the program
  - external data (e.g. files and databases)

Baselines

- A baseline is an approved version of a document or code.
- It shows that something is finished and ready to move forward.
- It marks an important stage in the project.
- After setting a baseline, changes need approval before being made.

Sources of Change

New market conditions dictate changes to product requirements or business rules

New customer needs demand modification of data, functionality, or services

Business reorganization causes changes in project priorities or SE team structure

Budgetary or scheduling constraints require system to be redefined

Change Requests

Requests can come from users, customers, or management

Change requests should be carefully analyzed as part of the maintenance process before they are implemented.

Some changes requests must be implemented urgently due to their nature

- Fault repair
- System environment changes
- Urgently required business changes

Change Prediction

- Predicting the number of changes requires understanding the relationships between a system and its environment
- Tightly coupled systems require changes whenever the
- environment changes
- Factors influencing the system/environment relationship
  - number and complexity of system interfaces
  - number and volatility of system requirements
  - business processes where the system is uses

Configuration Management Tasks

- Identification
  - Tracking changes to multiple SCI versions
- Version control
  - Controlling changes before and after customer release
- Change control
  - Authority to approve and prioritize changes
- Configuration auditing
  - Ensure changes are made properly
- Reporting
  - Tell others about changes made

Version Control Terms

1. Entity

- A complete group of related items that are at the same version.
- Think of it as one full package or unit.

2. Variant

- A different version of the same entity made for a specific purpose.
- It works alongside other variants but with small changes..

3. New Version

- Created when big changes are made to the entity.
- Shows major improvements or added features.

Change Control Process - 1

Change request is submitted and evaluated to assess its technical merit and impact on the other configuration objects and budget.

Change report containing the results of the evaluation is generated.

Change control authority (CCA) makes the final decision on the status and priority of the change based on the change report.

Change Control Process - 2

Engineering change order (ECO) is generated for each change approved.

- ECO describes the change, lists the constraints, and criteria for review and audit

Object to be changed is **checked-out** of the project database subject to access control parameters for the object.

Modified object is subjected to appropriate SQA and testing procedures.

Change Control Process - 3

Modified object is **checked-in** to the project database and version control mechanisms are used to create the next version of the software.

Synchronization control is used to ensure that parallel changes made by different people don't overwrite one another.

Configuration Management Team

Analysts.

Programmers.

Program Librarian.

Change Control Board

Customer representatives.

Some members of the Configuration management team.

Programmer's View - 1

Problem is discovered.

Problem is reported to configuration control board.

The board discusses the problem

- is the problem a failure?
- is it an enhancement?
- who should pay for it?

Assign the problem a priority or severity level, and assign staff to fix it.

Programmer's View - 2

Programmer or analyst

- Locates the source of the problem
- Determines what is needed to fix it

Programmer works with the librarian to control the installation of the changes in the operational system and the documentation.

Programmer files a change report documenting all changes made.

Change Control Issues

Synchronization (when?)
Identification (who?)
Naming (what?)
Authentication (done correctly?)
Authorization (who O.K.'d it?)
Routing (who's informed?)
Cancellation (who can stop it?)
Delegation (responsibility issue)
Valuation (priority issue)

Software Configuration Audit - 1

- Was the change made exactly as described in the Engineering Change Order (ECO)?
- Was a Formal Technical Review (FTR) conducted to ensure the change is technically correct?
- Did the team follow the proper software development process and apply all required software engineering standards?

Software Configuration Audit - 2

- Do the updated files or components clearly show the change that was made?
- Were the standard rules for recording and reporting the change (SCM standards) followed?
- Were all related documents and software items (SCI = Software Configuration Items) correctly updated?

Brain Break Challenge: Who's the Real Fixer?"

Question:

A team is consistently facing issues where updates made by different developers overwrite each other's changes. This has resulted in version mismatches and lost work.

Based on the configuration management concepts, which specific task or process should be reviewed and improved to address this issue?

A. Baseline approval
B. Synchronization control
C. Change request evaluation
D. Configuration item identification

Configuration Status Report

What happened?

Who did it?

When did it happen?

What else will be affected by the change?

Test Process Improvement

- TPI (Test Process Improvement) is a structured and systematic approach used to enhance the quality and effectiveness of software testing within an organization.
- TPI focuses on **identifying weaknesses and areas for improvement** in the **testing process, tools, and resources**, with the ultimate goal of improving the overall testing capability and software quality.
- Improvement of the test process from various **standpoints.**

Standpoints

- **Identifying defects**: Testing is about finding and fixing defects, such as bugs, missing requirements, and incorrect functionality.
- **Focusing on high-risk areas**: It's not possible to test every combination of scenarios in a software application, so testers should focus on the **most critical areas**.
- **Testing early**: It's more cost-effective to identify and fix defects early in the development process.
- **Verifying requirements**: Testing ensures that the product meets the functional, performance, design, and implementation requirements.
- **Providing information**: Testing can help determine if a product is ready for market.

- **Increasing reliability**: Testing can help identify and fix bugs, making the software more dependable.
- **Improving performance**: Testing can highlight areas for improvement and performance enhancements.
- **Ensuring user-focus**: Testing can help align the software with evolving user needs and feedback.
- **Enabling compatibility**: Testing can ensure that the software performs well across different platforms, devices, and environments.

- An improved process is not only more capable of finding better bugs but also testing the application in ways that uplift its quality and standards.

When to Perform Test Process Improvement?

- Significant rise in bugs
- Increase in complexity (test management)
- Increase in involved resources
- Increase in time of testing
- Increase in testing costs
- Newer methods have arrived
- not retrospected for a long time

Benefits of Test Process Improvement

- Testing improvements
- Good quality software
- Align testing with other phases
- Enhanced Value of Testing
- Reduced Downtime and Minimized Mistakes
- Adherence to Industry Standards
- Sustainable Cost Savings
- Accelerated Project Schedules
- Effective Evaluation of Improvement Efforts

Implement Test Process Improvement

- Test process improvement is a process of analysis and acting upon our observation.
- Implementation process can be divided into four parts.
  - **Diagnose the test improvement**
  - **Initiate the process/planning phase**
  - **Acting on the plan**
  - **Verify, Report, and Learn**

Diagnose the test improvement

Diagnose the situation :

- Determining why we need to perform test improvement
- Report specific problems in the current test process.
- Once we report our findings, we need to dig a little deeper into those areas to explore specific findings that can point us to the exact problems.
- For example, if our concern is cost management, then, we need to document what the costs were earlier, how it has grown over time, graphs depicting a clear view of the growth, what has impacted this high cost in the recent past, any significant change in the resource, etc. Consider this document as the sole evidence of moving further with test process improvement.

Initiate the process/planning phase

- Plan the steps you'll take to improve a process.
- Write down (document) what you want to achieve, the steps to follow, and how much of the process you'll cover.
- Find any possible problems (risks) that could slow down or stop the process, and guess how much delay each risk might cause.
- Set a deadline—when you expect the process improvement to be finished and when to check the results.

Acting on the plan

- With all the blueprints in our hand, we start the actual method of test process improvement.
- This should **follow the guidelines and expectations** described during the planning phase and **focus only on defects pointed out during the diagnosis phase**.
- The **senior testers** are required to monitor and guide this process through their expertise.
- They should also ensure that **delivery dates are not impacted and the process remains on schedule**.

Verify, Report, and Learn

At last, we verify the actions we performed with the results we got and match them with the expectations we set during the planning phase.

Next, we report all our results (including metrics) on the report for approval and sign-off from senior testers and higher management.

Lastly, we retrospect on the test improvement process from phases 1 to 4 and document our learnings throughout the process.

Measure Testing Process Improvement Impact

Define Clear Objectives: Clearly define the objectives of the testing process improvement.

Establish Baseline Metrics: Before implementing improvements, establish baseline metrics to understand the current state of the testing process.

Implement Changes Gradually: Introduce process improvements gradually rather than all at once. This allows for a more controlled assessment of the impact of each change on the overall testing process.

Monitor Key Performance Indicators (KPIs): Identify key performance indicators that directly align with the objectives of the testing process improvement.

Compare Metrics Before and After Implementation Compare the baseline metrics with data collected after the implementation of testing process improvements.

Calculate Return on Investment (ROI): Evaluate the return on investment by comparing the benefits gained from testing process improvements against the costs incurred in implementing those changes.

Utilize Surveys and Feedback: Administer surveys or conduct feedback sessions with the testing team to gauge their perceptions of the impact of process improvements.

Software Testing Process Improvement Models

- Similar to software development models (such as Agile and Waterfall), we rely on many test process improvement models to follow a set standard and improve our testing process.
- These models are often divided into "maturities"
- which means a certain stage that progresses toward better arrangement, organization, and completion.

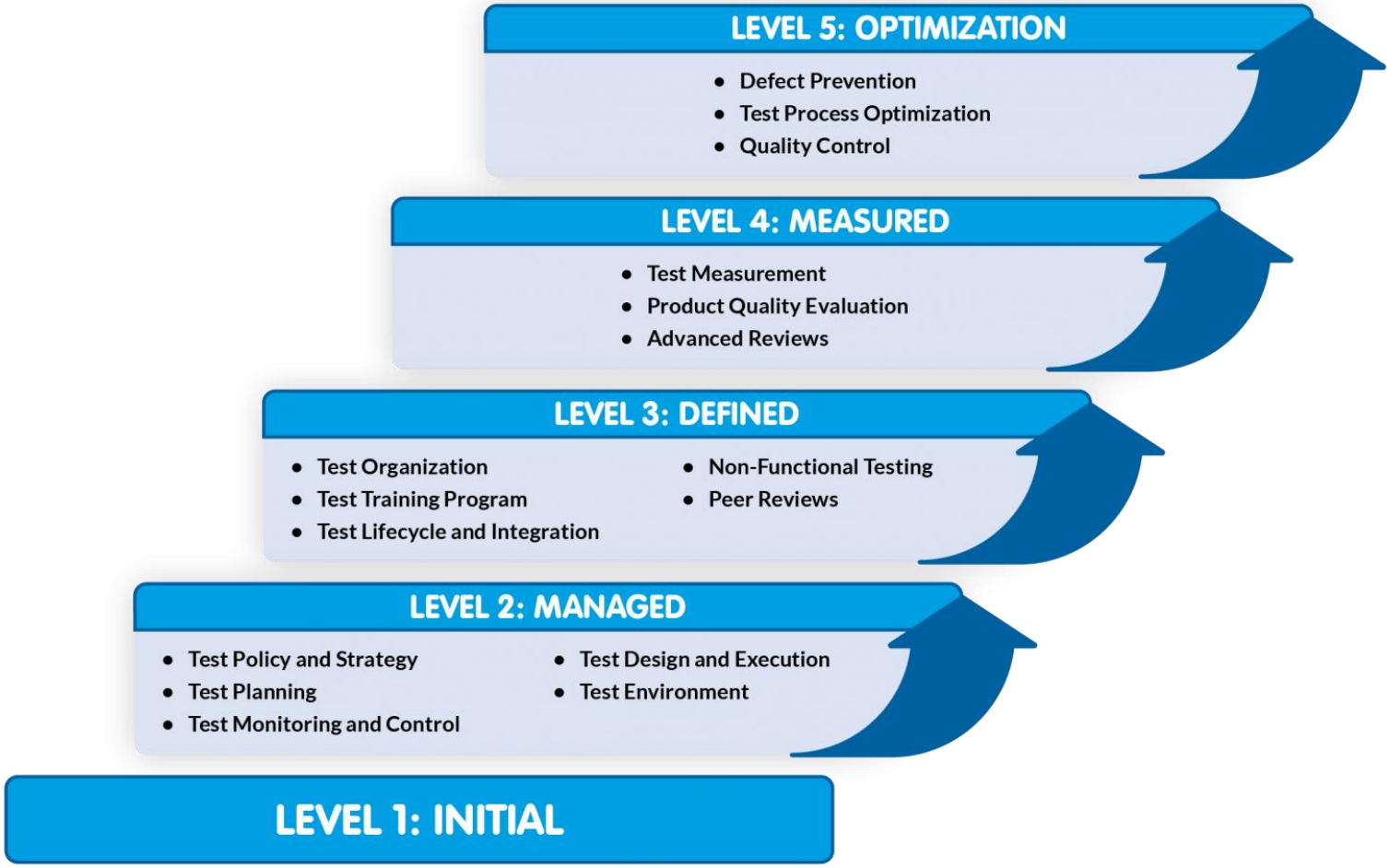
Testing Maturity Model integration (TMMi)

- TMMi is a "*maturity model*" which means it focuses on the maturity levels of each process to effectively implement the improvement in the testing lifecycle.
- Through TMMi, we can determine the maturity level of the testing process in an organization and improve it to achieve higher maturity.
- It was introduced in 2005 as a response to the ineffectiveness of the **Capability Maturity Model (CMM)** in the testing domain.
- TMMi tries to involve the organization structure and all the testers to progress together in the improvement process.

The benefits of using TMMi models are:

- Covers test-related activities extensively.
- Covers the best practices from the existing models.
- Covers the requirements for current trends and needs.
- Covers all aspects of test quality as described by industry experts.
- Provides a common standard for use.
- Can be applied to all phases of the software development lifecycle.

TMMi is divided into five maturity levels.



Maturity Level 1 – Initial

Maturity level 1 is not defined by the TMMi model because it presumes that all organizations start at this basic level. So, whatever you currently do and whatever process you follow, you can assume it to be of maturity level 1. This includes debugging and chaotic (unorganized) processes.

Maturity Level 2 – Managed

If the organization follows even a basic test approach towards testing and manages the same approach, it comes to maturity level 2. At this level, the organization should have a foundational structure in place that includes **test planning, test policy, monitoring, and setting up of a test environment**. This, however, **depends deeply on the project as a lot of things change when a new project is introduced**.

Maturity Level 3 – Defined

Maturity level 3 standardizes the testing process across the organization and expects **each project to follow the same process**. With this level, testing is now integrated very early into the development making it part of the development cycle.

Teams are required to be trained in testing and each member has a specific job for testing i.e. teams are more organized than before.

Maturity level 3 also expects that non-functional testing be planned and executed accordingly with reviews.

Maturity Level 4 – Measured

When the outcomes and parameter values are applied to all the projects to ensure a bug-free application, we reach maturity level 4. The review practices introduced in the maturity level 3 are now more advanced and thorough in nature.

Maturity Level 5 – Optimization

At maturity level 5, the organization has a series of methods set up for optimization of the processes followed in testing up to maturity level 4. Continuous optimization leads to a bug-free application.