Software Quality

Ashima Tyagi
Assistant Professor
School of Computer Science & Engineering

Outline

- Software Quality
- Software Quality Attributes
- McCall quality model
- Software Quality Assurance
- SQA Plans
- Software Quality Frameworks
- SEI-CMM Model

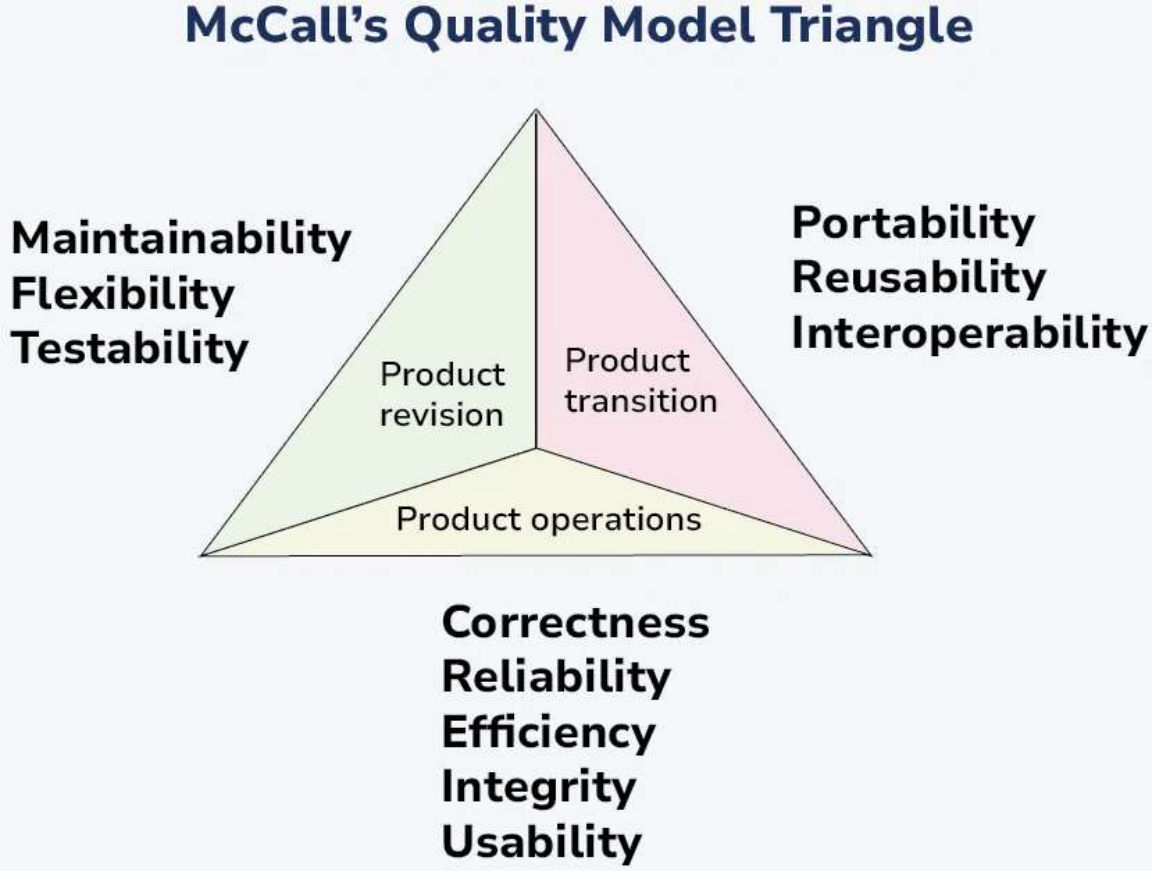
Software Quality

- Software quality refers to the degree to which a software product meets established requirements and user expectations.
- It encompasses various aspects, including functionality, reliability, usability, efficiency, maintainability, and portability.
- There is a misconception that if the software application is bug-free then the quality of the software is high. However, a bug-free product is just one of the Software Quality Attributes.

McCall quality model

McCall's Software Quality Model is incorporated with many attributes, termed software factors, which influence software.

- This model classifies all software requirements into 11 software quality factors. The 11 factors are organized into three product quality factors: Product Operation, Product Revision, and Product Transition.



Product Operation

Product Operation includes five software quality factors, which are related to the requirements that directly affect the operation of the software such as operational performance, convenience, ease of usage, and correctness. These factors help in providing a better user experience.

- Correctness
- Efficiency
- Integrity
- Reliability
- Usability

1. **Correctness:** Correctness refers to the ability to behave or function as per software requirement specifications. This may include navigations, calculations, form submissions, etc. Consider an example of sign-up. Your application should navigate to the terms and conditions page after signing up as per the requirement specification. However, it is landing on the home page without showing the terms and conditions page.
2. **Efficiency:** The software should use system resources efficiently, such as memory and processing power, to achieve its objectives. For example, if you open one of the video editor applications on your desktop, as soon as you open your system freezes, and all the other open application start to behave in an unintended way. This is a bad application design and shows the poor efficiency of the software.

3. **Integrity:** The extent to which the software can control an unauthorized person from accessing the data or software.

4. **Reliability:** Reliability is the ability of software applications to behave as expected and function under the maximum possible load. Let's take an example of an e-commerce website. The website went down as one of the server nodes crashed. In an ideal situation, if the reliability attribute is implemented then the system should automatically failover to the other server node.

5. **Usability:** The software should be user-friendly, intuitive, and easy to use for its intended users. let's consider an e-commerce page – user has purchased an item and wants to return the item. Good usability makes the return option available on the orders page. In some cases, the return option may appear on to contact us page – in this situation user easily gets confused and faces difficulty to find the return option.

Product Revision

Product Revision includes three software quality factors, which are required for testing and maintenance of the software. They provide ease of maintenance, flexibility, and testing efforts to support the software to be functional according to the needs and requirements of the user in the future.

- Maintainability
- Flexibility
- Testability

1. **Maintainability:** The software should be easy to maintain, update, and modify as needed to meet changing requirements or fix issues.

3. **Flexibility:** Flexibility refers to how quickly your application can adapt to future and current technology demands. For example, You are using a third-party library for styling your application. Due to some reasons, the third-party library declares the end of development. Now the question arises of how quickly your application can switch to another library. If it takes longer, then it might cost your business.

3. **Testability:** Testability is how easily QA members can test the software and log a defect and how easy it is to automate the software applications.

Product Transition

Product Transition includes three software quality factors, that allow the software to adapt to the change of environments in the new platform or technology from the previous.

- Portability
- Re-usability
- Interoperability

1. **Portability:** The software should be able to run on different platforms or environments without major modifications. An example of a portability issue – you have designed a web application that works perfectly fine on Android devices but when it is ported to iPhone devices (iOS), it fails to render.
2. **Reusability:** It is the degree to which software components can be reused in another application or the same application. Reusable software components reduce the development cost and effort. For example, your organization is developing two different applications where both the application needs sign-in and sign-up forms. If you don't have a reusable components strategy then you need to develop the same thing two times.
3. **Interoperability:** Interoperability refers to the ability to communicate or exchange data between different systems. Which can be operating systems, databases, or protocols. The interoperability problem arises due to the legacy code base, poorly architectured application, and poor code quality. For example, Your application needs to communicate with the payment gateway but you are facing challenges to integrate with the payment gateway due to various standards mismatch. If you had taken all the security, data, and standard API design approach this wouldn't have happened.

Software Quality Assurance (SQA)

Software Quality Assurance (SQA) is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedures as well as standards are suitable for the project and implemented correctly.

Software quality assurance focuses on:

- software's portability
- software's usability
- software's reusability
- software's correctness
- software's maintainability
- software's error control

Elements Of Software Quality Assurance:

1. **Standards:** The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents. The job of SQA is to ensure that standards that have been adopted are followed, and all work products conform to them.
2. **Reviews and audits:** Technical reviews are a quality control activity performed by software engineers for software engineers. Their intent is to uncover errors. Audits are a type of review performed by SQA personnel (people employed in an organization) with the intent of ensuring that quality guidelines are being followed for software engineering work.
3. **Testing:** Software testing is a quality control function that has one primary goal—to find errors. The job of SQA is to ensure that testing is properly planned and efficiently conducted for primary goal of software.
4. **Error/defect collection and analysis:** SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited to eliminating them.
5. **Change management:** SQA ensures that adequate change management practices have been instituted.
6. **Education:** Every software organization wants to improve its software engineering practices. A key contributor to improvement is education of software engineers, their managers, and other stakeholders. The SQA organization takes the lead in software process improvement which is key proponent and sponsor of educational programs.
7. **Security management**: SQA ensures that appropriate process and technology are used to achieve software security.
8. **Safety:** SQA may be responsible for assessing the impact of software failure and for initiating those steps required to reduce risk.
9. **Risk management**: The SQA organization ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.

What is verification and validation in quality assurance?

Validation is the process of checking whether the specification captures the customer's requirements, while verification is the process of checking that the software meets specifications.

SQA Plans

A Quality Assurance Plan (QAP) is a document or set of documents that outlines the systematic processes, procedures, and standards for ensuring the quality of a product or service.

Importance of Software Quality Assurance Plan

1. Quality Standards and Guidelines
2. Risk management
3. Standardization and Consistency
4. Customer Satisfaction
5. Resource optimization
6. Early Issue Detection

Steps to Develop Software Quality Assurance Plan:

1. Define Project Objectives and Scope
2. Identify Quality Standards and Criteria
3. Identify Stakeholders
4. Define Roles and Responsibilities
5. Conduct a Risk Assessment
6. Establish Quality Assurance Activities
7. Develop Testing and Inspection Procedures
8. Document Processes and Procedures
9. Establish Documentation and Reporting Mechanisms
10. Allocate Resources and Training
11. Define Change Control Processes
12. Review and Approval
13. Monitoring and Continuous Improvement
14. Communication and Training: Communicate the Quality Assurance Plan to all relevant stakeholders and provide training as necessary. Ensure that everyone involved understands their roles and responsibilities in maintaining and improving quality.

Software Quality Framework

A software quality framework is a structured approach or methodology used to ensure that software products meet defined quality standards. It provides a systematic way to manage and improve the quality of software throughout its lifecycle. A software quality framework typically includes the following components:

1. **Quality Planning:** Defining quality objectives, standards, and metrics for the software project.
2. **Quality Assurance:** Processes and activities to ensure that quality standards are being followed throughout the software development lifecycle. This may include reviews, audits, and process improvements.
3. **Quality Control:** Processes and activities to monitor and verify that the software meets the defined quality standards. This may include testing, inspections, and defect tracking.
4. **Quality Improvement:** Continuous improvement activities to enhance the quality of the software and the development process. This may include root cause analysis, process optimization, and lessons learned.
5. **Quality Metrics:** Measurement and analysis of quality-related data to track progress, identify trends, and make informed decisions about quality improvement efforts.
6. **Quality Management:** Overall management of quality within the organization, including leadership, commitment, and support for quality initiatives.

Some common software quality frameworks include **the ISO 9000 series, the Capability Maturity Model (CMM), and the Six Sigma approach.**

ISO Model

The ISO (International Organization for Standardization) has provided internationally recognized models for evaluating software quality.

The most commonly referenced are:

- ISO/IEC 9126 (older, foundational model)
- ISO/IEC 25010 (updated and extended version)

ISO/IEC 9126 Quality Model

It classifies software quality into **6 major characteristics**, each with sub-characteristics:

1. Functionality – Does the software meet its purpose?
2. Reliability – Can it perform under stated conditions?
3. Usability – Is it easy to use?
4. Efficiency – Is it resource-efficient?
5. Maintainability – How easy is it to fix or improve?
6. Portability – Can it run in different environments?

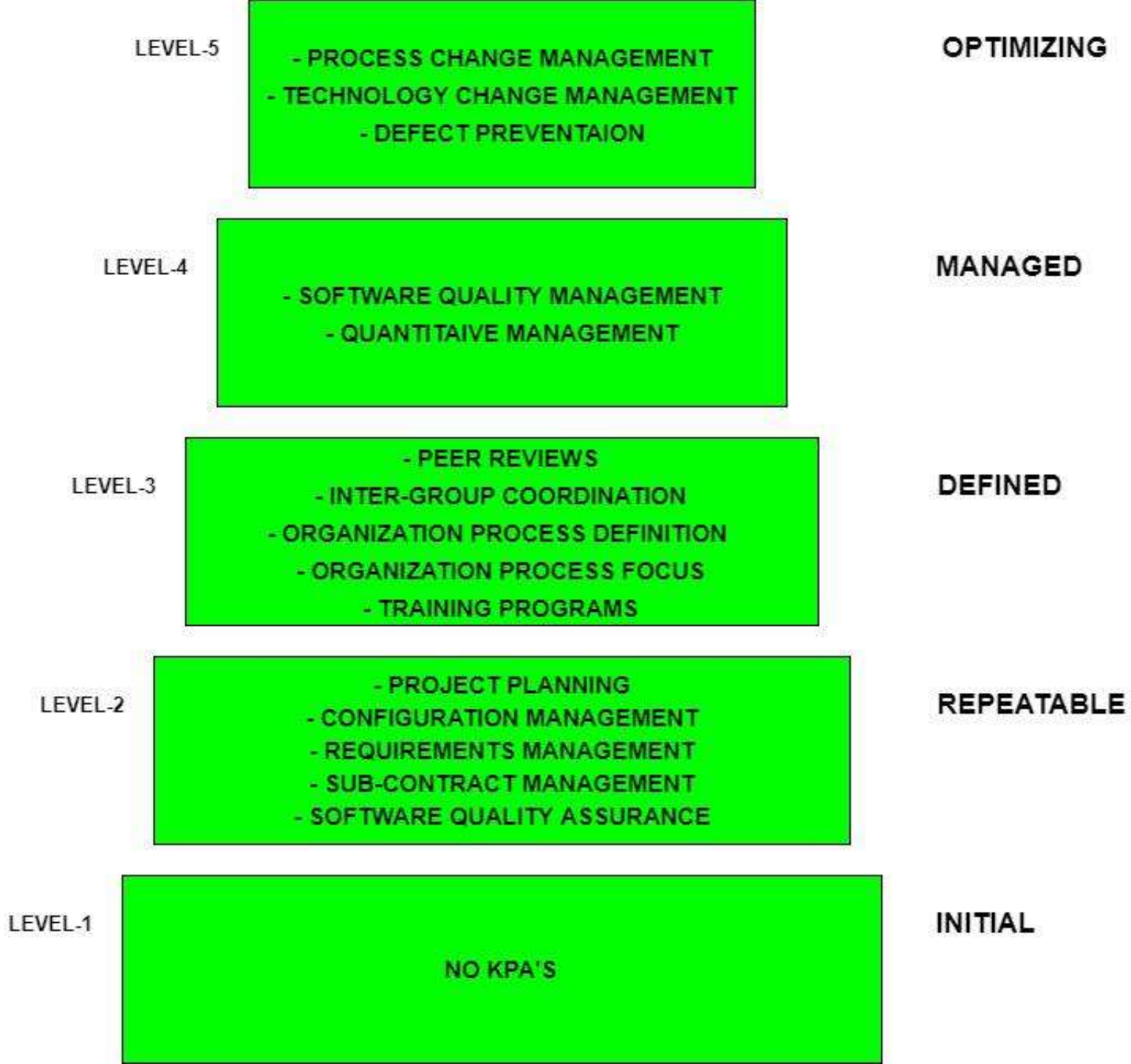
🔄 ISO/IEC 25010 (Updated Version)

Introduces 8 quality characteristics:

ISO 25010 Characteristic	Description
1. Functional Suitability	Accuracy and completeness of functions
2. Performance Efficiency	Response time, resource usage
3. Compatibility	Co-existence and interoperability
4. Usability	Ease of learning and using the system
5. Reliability	Stability and fault tolerance
6. Security	Confidentiality, integrity, authentication
7. Maintainability	Ease of analysis, change, testing
8. Portability	Ease of transfer between environments

Capability Maturity Model (CMM)

- The Capability Maturity Model (CMM) is a tool used to improve and refine software development processes.
- It provides a structured way for organizations to assess their current practices and identify areas for improvement.
- CMM consists of five maturity levels: initial, repeatable, defined, managed, and optimizing.
- By following the CMM, organizations can systematically improve their software development processes, leading to higher-quality products and more efficient project management.
- Each level of maturity shows a process capability level.
- All the levels except level 1 are further described by Key Process Areas (KPA).


LEVEL-5
- PROCESS CHANGE MANAGEMENT
- TECHNOLOGY CHANGE MANAGEMENT
- DEFECT PREVENTAION
OPTIMIZING
LEVEL-4
- SOFTWARE QUALITY MANAGEMENT
- QUANTITAIVE MANAGEMENT
MANAGED
LEVEL-3
- PEER REVIEWS
- INTER-GROUP COORDINATION
- ORGANIZATION PROCESS DEFINITION
- ORGANIZATION PROCESS FOCUS
- TRAINING PROGRAMS
DEFINED
LEVEL-2
- PROJECT PLANNING
- CONFIGURATION MANAGEMENT
- REQUIREMENTS MANAGEMENT
- SUB-CONTRACT MANAGEMENT
- SOFTWARE QUALITY ASSURANCE
REPEATABLE
LEVEL-1
NO KPA'S
INITIAL

Key Process Areas (KPA)

Each of these KPA (Key Process Areas) defines the basic requirements that should be met by a software process to satisfy the KPA and achieve that level of maturity.

Levels of Capability Maturity Model (CMM) are as following below.

1. Level One : Initial – Work is performed informally.

A software development organization at this level is characterized by AD HOC activities (organization is not planned in advance.).

2. Level Two : Repeatable – Work is planned and tracked.

This level of software development organization has a basic and consistent project management processes to TRACK COST, SCHEDULE, AND FUNCTIONALITY. The process is in place to repeat the earlier successes on projects with similar applications.

3. Level Three : Defined – Work is well defined.

At this level the software process for both management and engineering activities are DEFINED AND DOCUMENTED.

4. Level Four : Managed – Work is quantitatively controlled.

Software Quality management – Management can effectively control the software development effort using precise measurements. At this level, organization set a quantitative quality goal for both software process and software maintenance.

Quantitative Process Management – At this maturity level, The performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable.

5. Level Five : Optimizing – Work is Based Upon Continuous Improvement.

The key characteristic of this level is focusing on CONTINUOUSLY IMPROVING PROCESS performance.

Key features are:

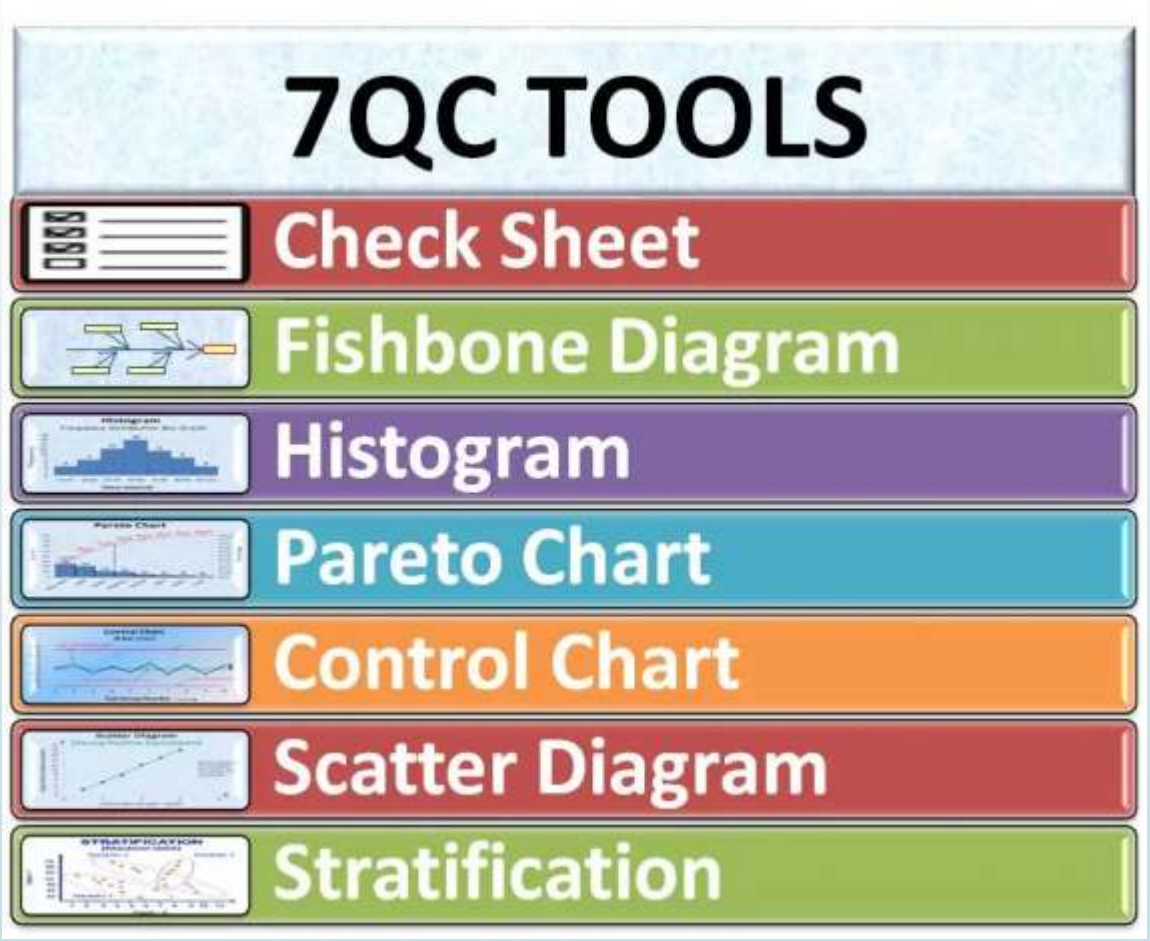
Process change management

Technology change management

Defect prevention

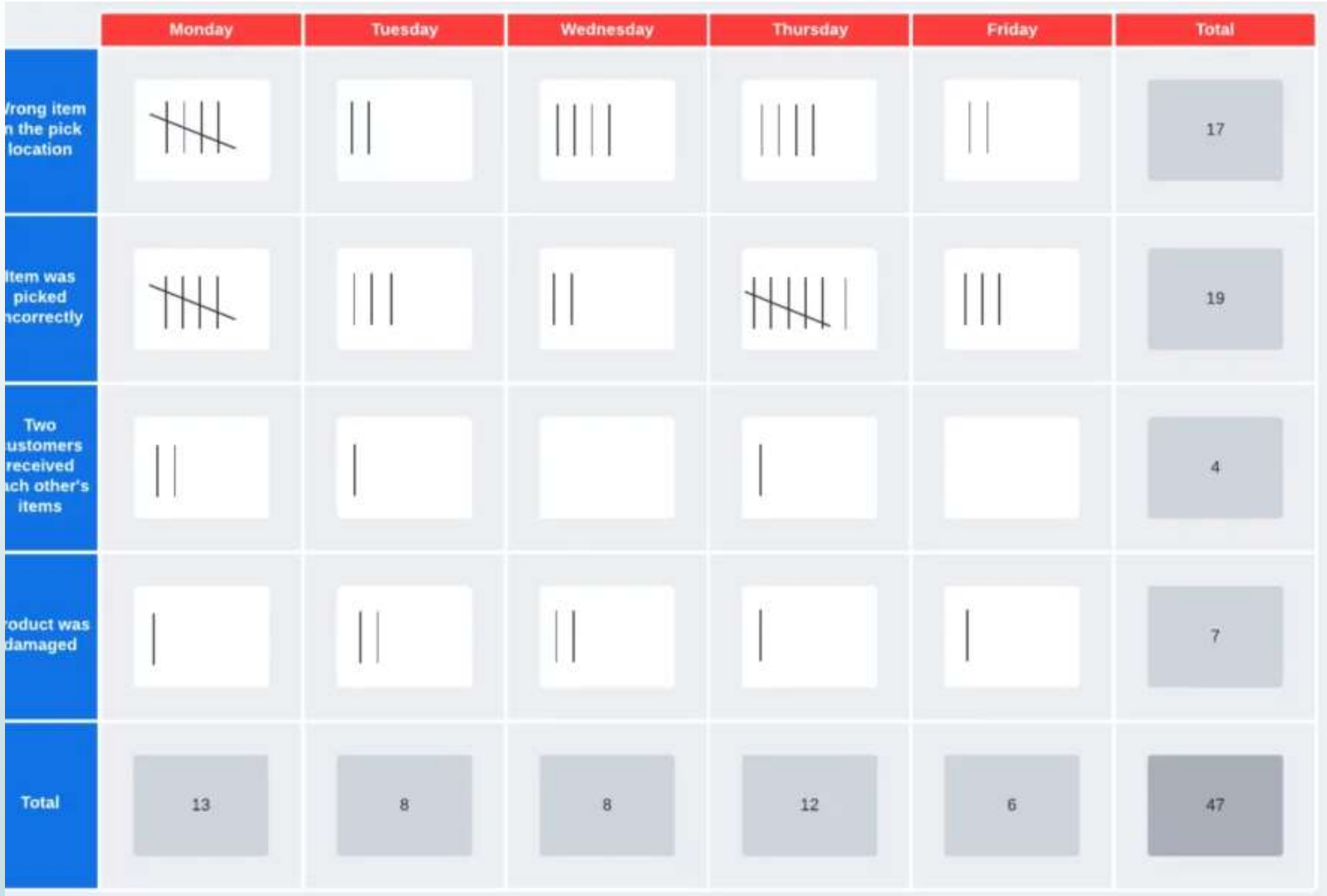
Tools and Techniques for Quality Control

- Quality Control in software engineering ensures that the product meets the required quality standards.
- It focuses on detecting defects in the product.
- Below are the most commonly used tools and techniques for QC, often referred to as part of **the Seven Basic Quality Tools** (from manufacturing, also adapted in software):



1. Checksheet

The **check sheet** is used for collecting, recording, and analyzing the data. Data collection is an important activity in the problem-solving process as it provides a basis for further action. Data may be numerical, observations and opinions, etc.

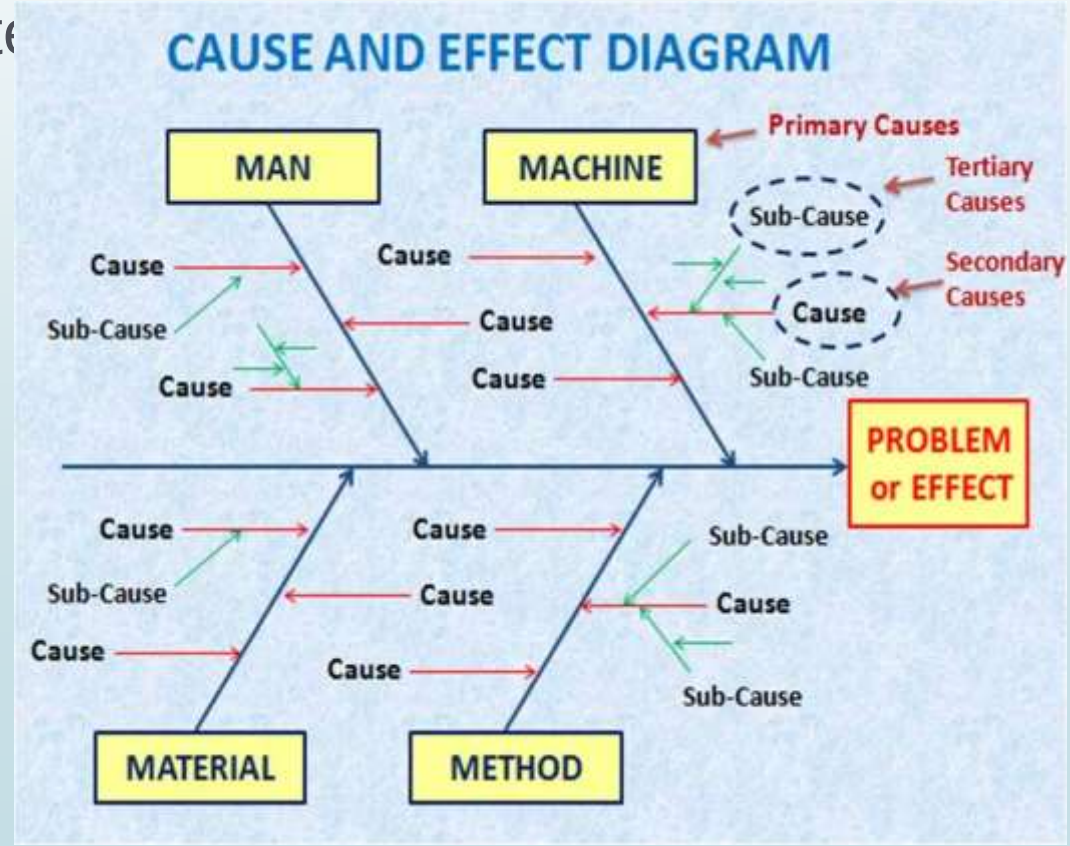


	Monday	Tuesday	Wednesday	Thursday	Friday	Total
rong item n the pick location	𝍸	𝍸	𝍸	𝍸	𝍸	17
tem was picked ncorrectly	𝍸	𝍸	𝍸	𝍸	𝍸	19
Two ustomers received ch other's items	𝍸	𝍸		𝍸		4
oduct was damaged	𝍸	𝍸	𝍸	𝍸	𝍸	7
Total	13	8	8	12	6	47

2. Fishbone Diagram

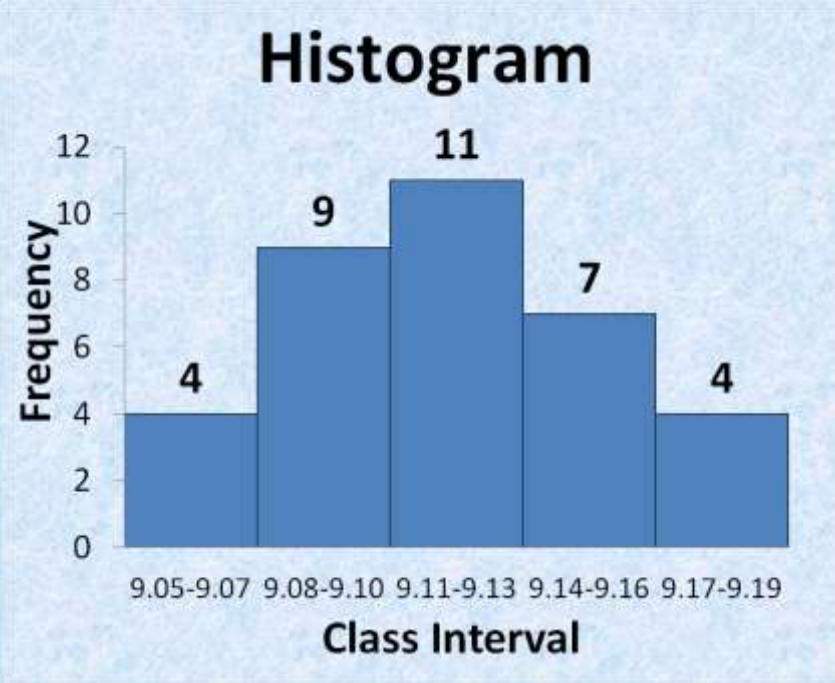
Fishbone diagram is also called as Cause and Effect diagram and Ishikawa diagram. It helps to Identify all possible potential causes and select the real/best potential cause which contributes to the problem/effect. The brainstorming technique is used for pote

In a brainstorming session, all 4M or 6M factors are taken into consideration to identify the potential causes. 4M or 6M factors are – Man, Machine, Method, Material, Measurement, and Mother nature also called Environment.



3. Histogram

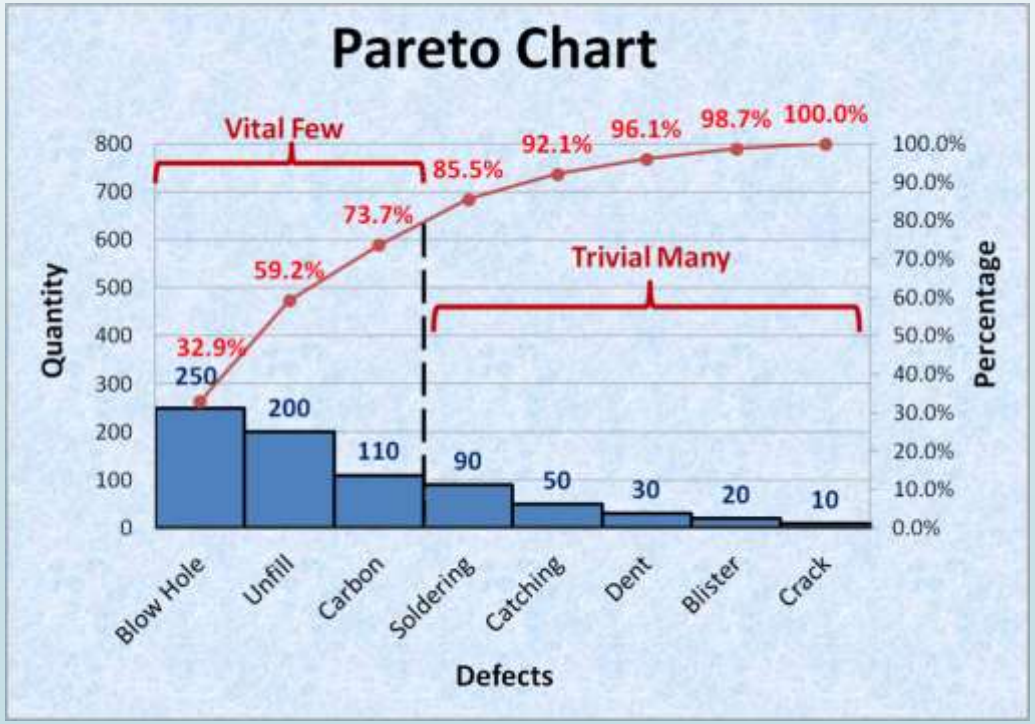
A Histogram is a pictorial representation of a set of data, and the most commonly used bar graph for showing frequency distributions of data/values. Histogram frequency distribution chart is widely used in Six Sigma problem solving process.



4. Pareto Chart

The Pareto chart helps to Narrow the problem area or prioritize the significant problems for corrective measures. The pareto principle is based on the 80-20 rule. It means that 80 percent of the problems/failures are caused by 20 percent of the few major causes/factors which are often referred to as Vital Few.

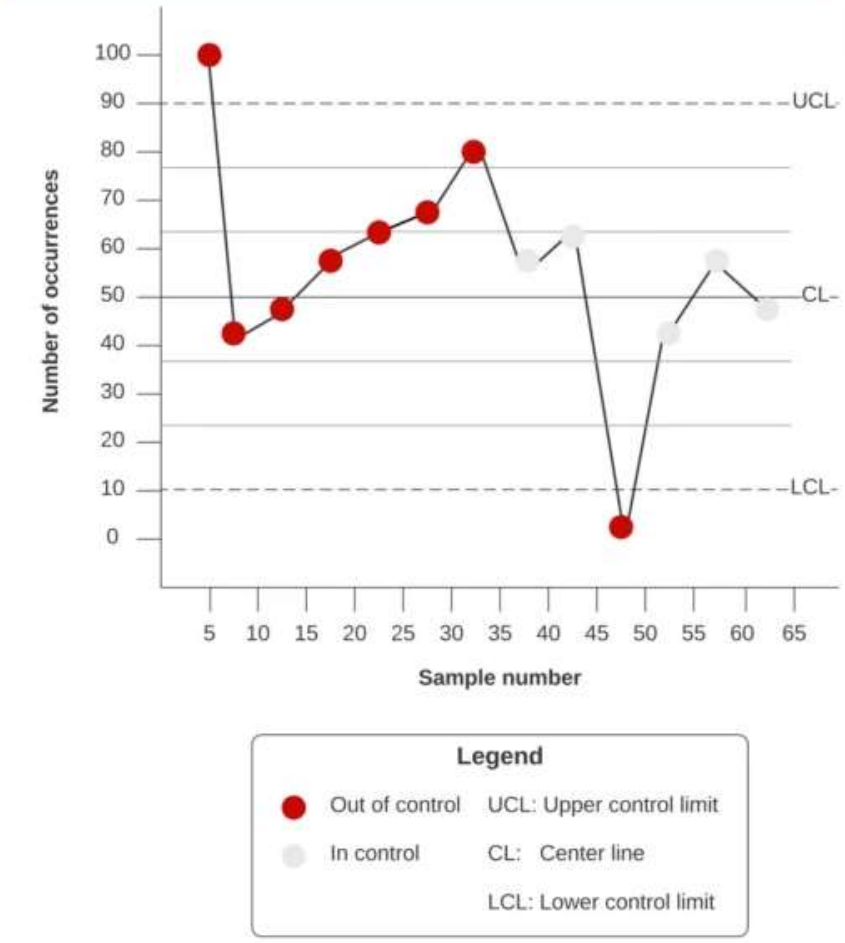
And the remaining 20 percent of the problems are caused by 80 percent of many minor causes which are referred to as **Trivial Many**. Hence, it gives us information about Vital few from Trivial many.



5. Control Chart

Control charts, also known as Shewhart charts, are powerful statistical tools used for monitoring and analyzing process performance over time.

Control chart



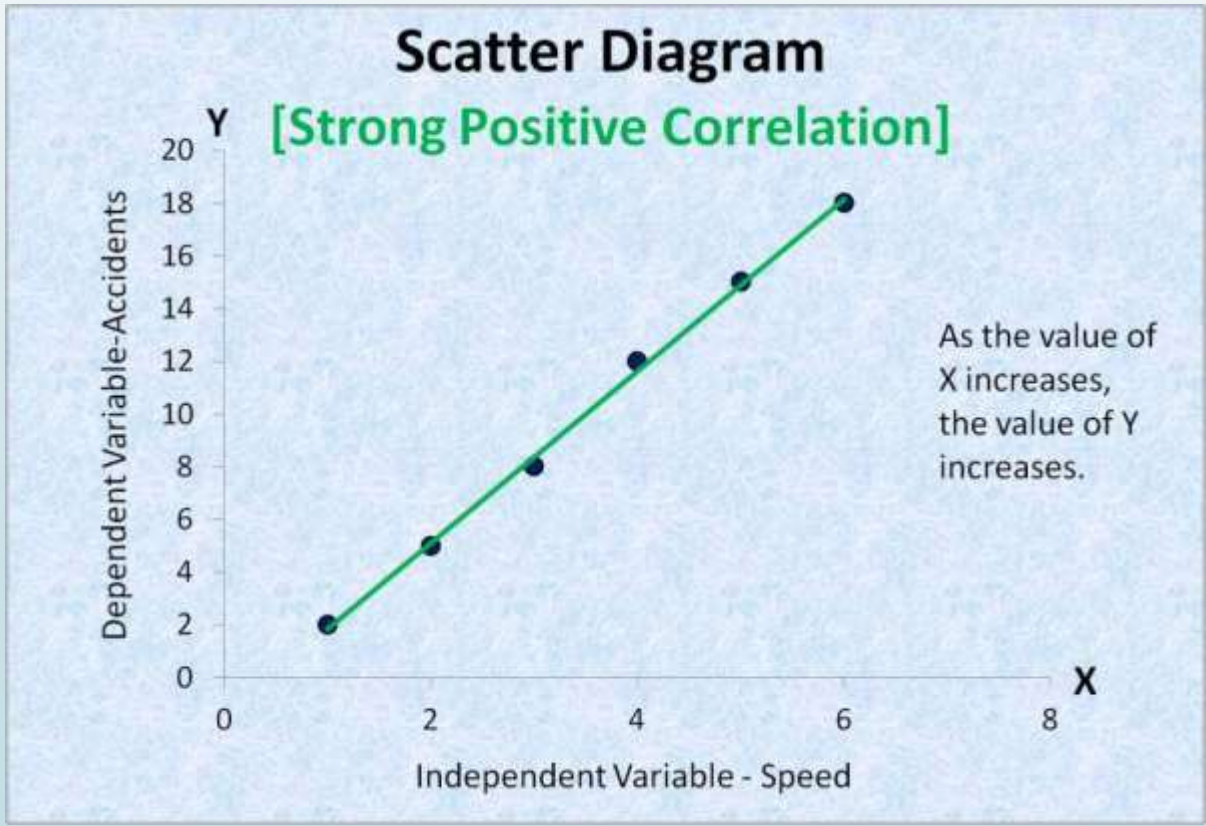
Action plan

Action	Owner	Start date	End date	Required resources	Status
Analyze chart	John	3/15	3/20	Control chart, process analyst	In progress
Investigate the cause for the out-of-control metric	Chris	3/15	3/22	Documentation of expected process, process manger	In progress
Determine if changes need to be made to the process	Peter	3/16	3/22	Process manager	Completed

6. Scatter Diagram

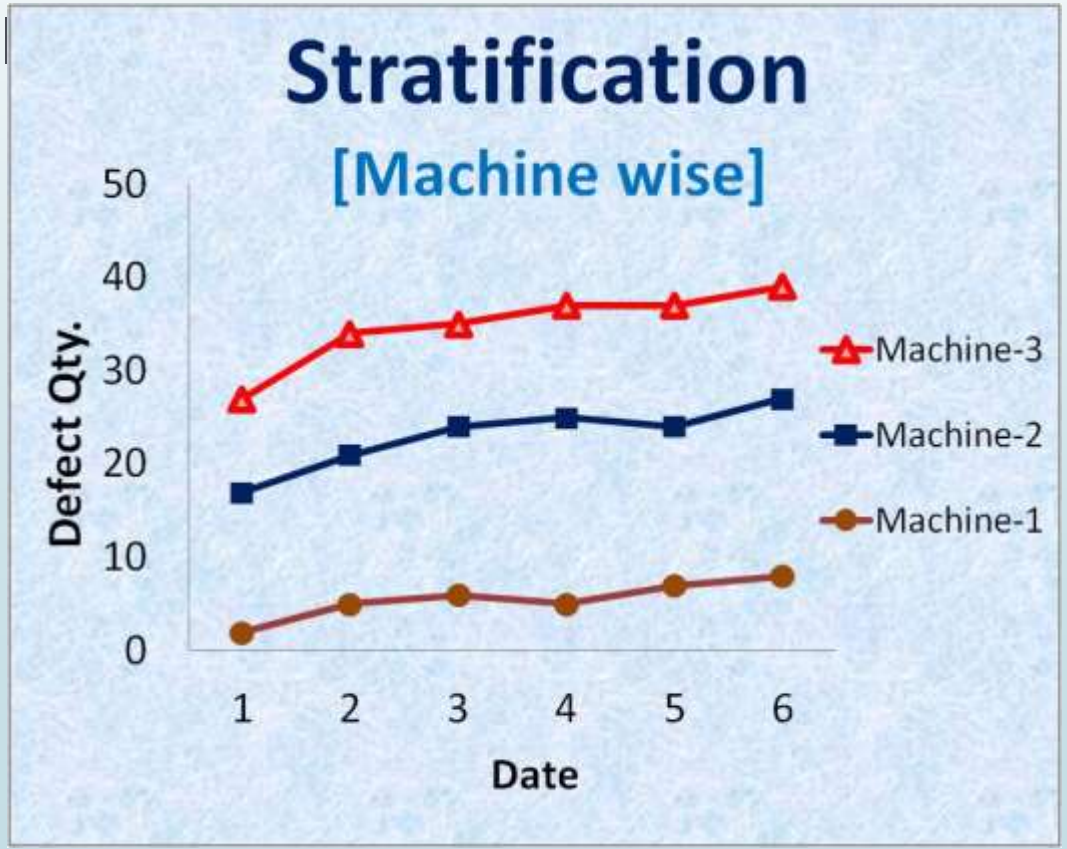
A Scatter diagram is also known as Correlation Chart, Scatter Plot, and Scatter Graph. A Scatter graph is used to find out the relationship between two variables. In other words, it shows the relationship between two sets of numerical data. Scatter graph shows a Positive or Negative correlation between two variables.

Independent variable data and dependent Variable data are customarily plotted along the horizontal X-axis and Vertical Y-axis respectively. Independent variable is also called controlled parameters.



7. Stratification Diagram

A technique used to analyze and divide a universe of data into homogeneous groups is called -Strata. Stratification tools are used when the data come from different sources or conditions, such as data collected from different shifts, machines, people, days, suppliers and popul



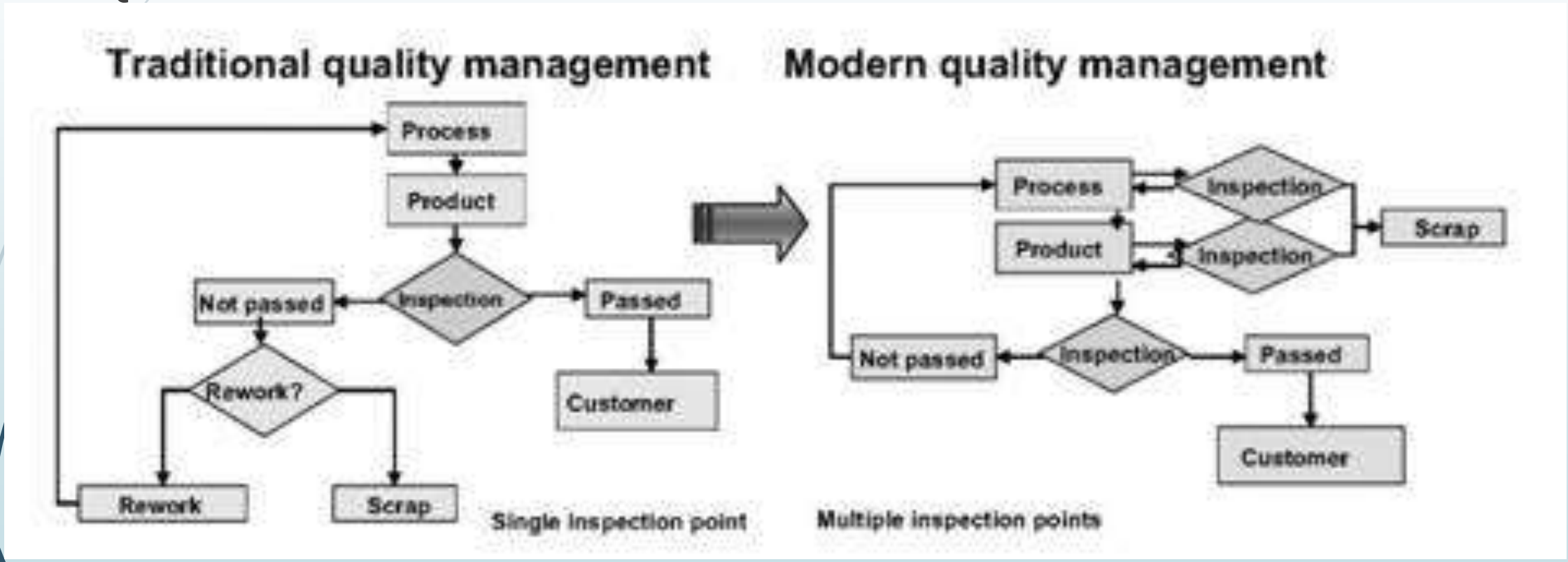
Example:

An Indian software company like **Wipro** or **Infosys** might use these tools for quality improvement. A **Check Sheet** could track types of bugs found during testing. A **Pareto Chart** could identify the modules causing the most bugs. A **Cause-and-Effect Diagram** could explore reasons for project delays. A **Histogram** might analyze the distribution of task completion times. A **Scatter Diagram** could check if developer experience correlates with bug rates. A **Flow Chart** documents the software development lifecycle. **Control Charts** might monitor metrics like code complexity or defect density over time.

Modern Quality Management

Modern Quality Management (MQM) refers to contemporary practices, tools, and principles that ensure high-quality software while emphasizing customer satisfaction, continuous improvement, and collaborative development.

Changes in management concepts corrections to be made before additional work is done. Scrap and rework cost is significantly reduced in MQM.



Key Principles of Modern Quality Management

1. Customer-Centric Approach

- Quality is defined by how well the product meets **customer needs** and expectations.
- Involves **user feedback**, **continuous delivery**, and **fast iterations**.

2. Total Quality Management (TQM)

- Everyone in the organization is responsible for quality.
- Focus on **process improvement**, **employee involvement**, and **long-term success**.

3. Continuous Improvement (Kaizen)

- Regular and incremental improvements in processes, code, and performance.
- Encourages **experimentation**, **learning**, and **feedback loops**.

4 Risk Management and Prevention

- Focus on **defect prevention** rather than detection.
- Emphasizes **early testing**, code reviews, static analysis, etc.

5 Process Metrics & Automation

- Using **automated tools** (e.g., CI/CD pipelines, automated testing).
- Collecting and analyzing metrics like **code coverage**, **defect density**, and **deployment frequency**.

VS Traditional vs Modern Quality Management

Feature/Aspect	□ Traditional Quality Management	🚀 Modern Quality Management
Focus	Product-centric	Customer-centric
Quality Control	After development (post-mortem)	Continuous, during development
Processes	Waterfall	Agile, DevOps, Lean
Responsibility	QA Team only	Everyone (developers, testers, ops)
Feedback	Late	Fast, continuous
Tools	Manual testing, documentation	Automation, CI/CD, analytics
Documentation	Heavy	Lightweight, just enough
Improvement	Periodic	Continuous (Kaizen)
Standards	ISO 9001, CMM	ISO 25010, CMMI, Six Sigma
Customer Involvement	Limited	High (feedback loops, demos)

Thank You