

Unit VI: Software Reliability & Quality Management

6.1 SOFTWARE RELIABILITY

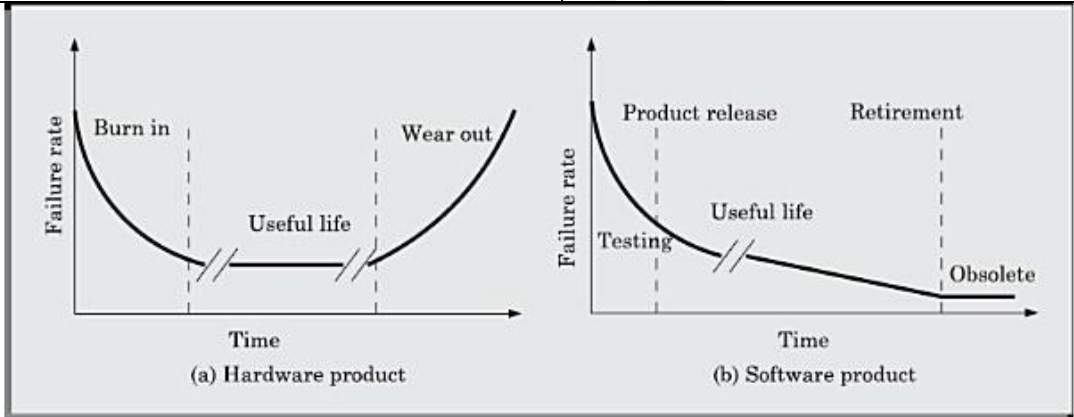
- The reliability of a software product essentially denotes its **trustworthiness or dependability**.
- The reliability of a software product can also be defined as the **probability of the product working “correctly” over a given period of time**.
- Software product having a large **number of defects is unreliable**.
- It is also very reasonable to assume that the **reliability of a system improves**, as the **number of defects in it is reduced**.
- Reliability of a product depends **not only on the number of latent [hidden] errors** but also on the exact **location of the errors**.
- It depends upon **how the product is used**, or on its **execution profile**.
- If the users execute only those features of a program that are **“correctly” implemented**, **none of the errors** will be **exposed** and the **perceived reliability** of the product will be **high**. On the other hand, if only those **functions of the software** which contain **errors are invoked**, then a large number of failures will be observed and the perceived **reliability** of the system will be **very low**.
- For example, for a **Library Automation Software** the **library members** would use **functionalities such as issue book, search book**, etc., on the other hand the **librarian** would normally execute features such as create member, create book record, delete member record, etc. So defects which show up for the librarian, may not show up for the members. so users of these two categories may have different opinions of reliability of the software.

The main reasons that make software reliability more difficult to measure than hardware reliability:

- The reliability improvement due to **fixing a single bug** depends on **where the bug is located in the code**.
- The perceived reliability of a software product is **observer-dependent**.
- The reliability of a product keeps **changing as errors are detected and fixed**.

6.1.1 Hardware versus Software Reliability

Hardware components fail mostly due to wear and tear, whereas software components fail due to bugs.

Sr no	Hardware product	Software product
1	A logic gate may be stuck at 1 or 0, or a resistor might short circuit. To fix a hardware fault, one has to either replace or repair the failed part.	software product would continue to fail until the error is tracked down and either the design or the code is changed to fix the bug
2	When a hardware part is repaired its reliability would be maintained at the level that existed before the failure occurred	when a software failure is repaired, the reliability may either increase or decrease (reliability may decrease if a bug fix introduces new errors).
3	hardware reliability study is concerned with stability (for example, the inter-failure times remain constant)	Aim of software reliability study would be reliability growth (that is, increase in inter-failure times).
4	the plot of change of reliability with time for a hardware component (Figure 6.1(a)) appears like a “bath tub”.	For a software component the failure rate is initially high, but decreases as the faulty components identified are either repaired or replaced.
5	 <p style="text-align: center;">Figure 6.1: Change in failure rate of a product</p>	

- For a **hardware component** the failure rate is **initially high**, but decreases as the faulty components identified are either **repaired or replaced**. The system then enters its useful life, where the **rate of failure** is almost **constant**. After some time (called product life time) the major components wear out, and the failure rate increases. The initial failures are usually covered through manufacturer’s warranty.
- the **software product** show the highest failure rate just after purchase and installation (see the initial portion of the plot in Figure 6.1 (b)). As the system is used, more and more **errors are identified and removed** resulting in reduced failure rate. This error removal continues at a slower pace during the useful life of the product. As the software becomes **obsolete no more error correction occurs** and the failure rate remains unchanged.

6.1.2 Reliability Metrics of Software Products

- The **reliability requirements** for different categories of **software products** may be **different**.
- **level of reliability** required for a software product should be specified in the Software requirements specification (**SRS**) **document**.
- Some **metrics to quantitatively express** the **reliability** of a software product. six metrics that correlate with reliability as follows:

1. Rate of occurrence of failure (ROCOF)

- Measures **the frequency of occurrence of failures**.
- ROCOF measure of a software product can be obtained by **observing the behaviour of a software product** in operation over a specified time interval and then calculating the **ROCOF value as the ratio of the total number of failures observed and the duration of observation**.

2. Mean time to failure (MTTF):

- MTTF is the **time between two successive failures, averaged over a large number of failures**.
- To measure MTTF, we can record the **failure data for n failures**.
- Let the failures occur at the time instants **t1, t2, ..., tn**. Then, MTTF can be calculated as

$$\sum_{i=1}^n \frac{t_{i+1} - t_i}{(n-1)}.$$

3. Mean time to repair (MTTR):

- Once failure occurs, some time is required to fix the error. MTTR measures the **average time it takes to track the errors causing the failure and to fix them**.

4. Mean time between failure(MTBF):

- The MTTF and MTTR metrics can be combined to get the **MTBF metric**:
- **MTBF=MTTF+MTTR**.
- Thus, MTBF of 300 hours indicates that once a failure occurs, the next failure is expected after 300 hours.
- In this case, the time measurements are real time and not the execution time as in MTTF

5. Probability of failure on demand (POFOD):

- **POFOD** measures the **likelihood of the system failing** when a **service request** is made.

- For example, a POFOD of 0.001 would mean that **1 out of every 1000 service requests would result in a failure.**
- 6. **Availability:**
 - Availability of a system is a measure of how **likely would the system be available** for use over a given period of **time**.
 - This metric considers the **number of failures** occurring during a time interval and the **repair time (down time) of a system** when a failure occurs.
 - This metric is important for systems such as **telecommunication systems, and operating systems, and embedded controllers, etc.**

Shortcomings of reliability metrics of software products

- These metrics are centered **around the probability of occurrence of system failures** but take **no account of the consequences of failures**.
- Do not distinguish the **relative severity of different failures**.
- In order to estimate the **reliability of a software product** more accurately, it is necessary to **classify various types of failures**.
- A scheme of classification of failures is as follows:

1. Transient:

- Transient failures occur only for **certain input values** while invoking a function of the system.

2. Permanent:

- Permanent failures occur for **all input values** while invoking a function of the system.

3. Recoverable:

- When a recoverable failure occurs, the system can recover without having to shutdown and restart the system (with or without operator intervention).

4. Unrecoverable:

- In unrecoverable failures, the system may need to be restarted.

5. Cosmetic:

- These classes of failures cause only **minor irritations**, and do not lead to incorrect results.
- An example of a cosmetic failure is the situation where the mouse button has to be clicked twice instead of once to invoke a given function through the graphical user interface.

6.1.3 Reliability Growth Modelling

- A reliability growth model is a **mathematical model** of how **software reliability improves as errors are detected and repaired**.

- A reliability growth model can be used to predict when (or if at all) a particular level of reliability is likely to be attained.
- Reliability growth modelling can be used to **determine when to stop testing to attain a given reliability level.**

Two very simple reliability growth models as given below

1. Jelinski and Moranda model

- The simplest reliability growth model is a **step function model** where it is assumed that the **reliability increases by a constant increment** each time an error is detected and repaired. model is shown in Figure 6.2.
- In this model it is **implicitly assumed that all errors contribute equally to reliability growth**, which is highly **unrealistic** since we know that correction of different errors contribute differently to reliability growth.

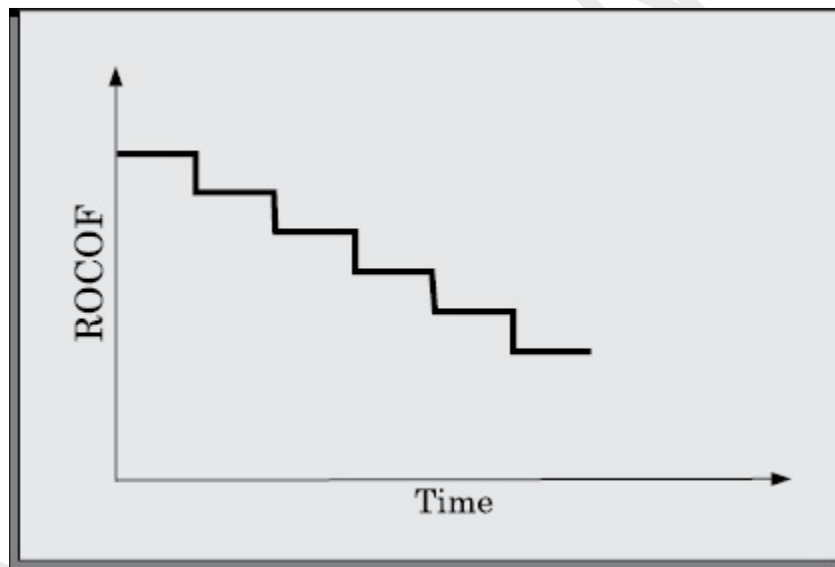


Figure 6.2: Step function model of reliability growth.

2. Littlewood and Verall's model

- This model allows for **negative reliability growth** to reflect the fact that when a **repair is carried out**, it may introduce **additional errors**.
- It also models the fact that **as errors are repaired**, the **average improvement** to the product reliability per **repair decreases**.
- It treats an **error's contribution to reliability improvement** to be an independent **random variable** having Gamma distribution.
- This distribution models the fact that **error corrections with large contributions to reliability growth are removed first**.
- This represents diminishing return as test continues.

6.2 SOFTWARE QUALITY

- Quality of a product is defined in terms of its **fitness of purpose**.
- Fitness of purpose is interpreted in terms of **satisfaction of the requirements laid down in the SRS document**.
- “**Fitness of purpose**” is not a wholly **satisfactory definition of quality for software products**.
- **Example:** Although software product correctly **performs all the functions that have been specified in its SRS** we cannot consider it to be a **quality product**, if it has an almost **unusable user interface**.

The modern view of a quality factors associate with a software product as follows.

1. Portability:

- A software product is said to be **portable**, if it can be easily made to work in different **hardware and operating system environments**, and easily **interface with external hardware devices and software products**.

2. Usability:

- A software product has **good usability**, if different categories of users (i.e., both expert and novice users) can easily invoke the functions of the product.

3. Reusability:

- A software product has good reusability, if different modules of the product can easily be reused to develop new products.

4. Correctness:

- A software product **is correct**, if different requirements as specified in the SRS document have been correctly implemented.

5. Maintainability:

- A software product is maintainable, if errors **can be easily corrected** as and when they show up, new functions can be easily added to the product, and the functionalities of the product can be easily modified, etc.

McCall's quality factors

- McCall distinguishes two levels of **quality attributes** [McCall].
- Higher level attributes, known as **quality factor s or external attributes** can only be measured **indirectly**.
- The second-level quality attributes (**quality criteria**) **can be measured directly, either objectively or subjectively**.
- **Reliability** is a **higher-level quality factor** and number of defects is a low-level quality factor.

ISO 9126

- ISO 9126 defines a set of hierarchical quality characteristics.

6.3 SOFTWARE QUALITY MANAGEMENT SYSTEM

- A quality management system (often referred to as **quality system**) is the **principal methodology** used by **organisations** to ensure that the products they develop have the desired quality.

6.3.1 Issues associated with a quality system:

1. Managerial structure and individual responsibilities:

- A quality system is the **responsibility of the organisation as a whole**.
- Every organisation has a separate **quality department** to perform several quality system activities.
- The quality system of an organisation should have the **full support of the top management**.
- Without support for the **quality system** at a high level in a company, few members of staff will take the quality system seriously.

6.3.2 Quality system activities

The quality system activities encompass the following:

1. Auditing of projects to check if the processes are being followed.
2. Collect **process and product metrics** and analyse them to check if quality goals are being met.
3. **Review of the quality system** to make it more effective.
4. Development of **standards, procedures, and guidelines**.
5. **Produce reports** for the top management summarising the effectiveness of the quality system in the organisation.

6.3.3 Evolution of Quality Systems

- Quality systems of organisations have undergone **four stages of evolution** as shown in Figure 6.3.

1. Inspection method:

- **The initial product inspection** method gave way to quality control (QC) principles.

2. Quality control (QC)

- Focuses not only on **detecting the defective products** and **eliminating** them, but also on determining the **causes behind the defects**, so that the **product rejection rate can be reduced**.

3. Modern quality assurance:

- The basic premise of **modern quality assurance** is that if an organisation's processes are good and are followed rigorously, then the products are bound to be of good quality.
- **It includes** guidance for recognising, defining, analysing, and improving the production process.

4. Total quality management (TQM)

- TQM advocates that the process followed by an organisation.
- must continuously be improved through process measurements.
- TQM goes beyond documenting processes to optimising them through redesign.
- TQM is business process re-engineering.

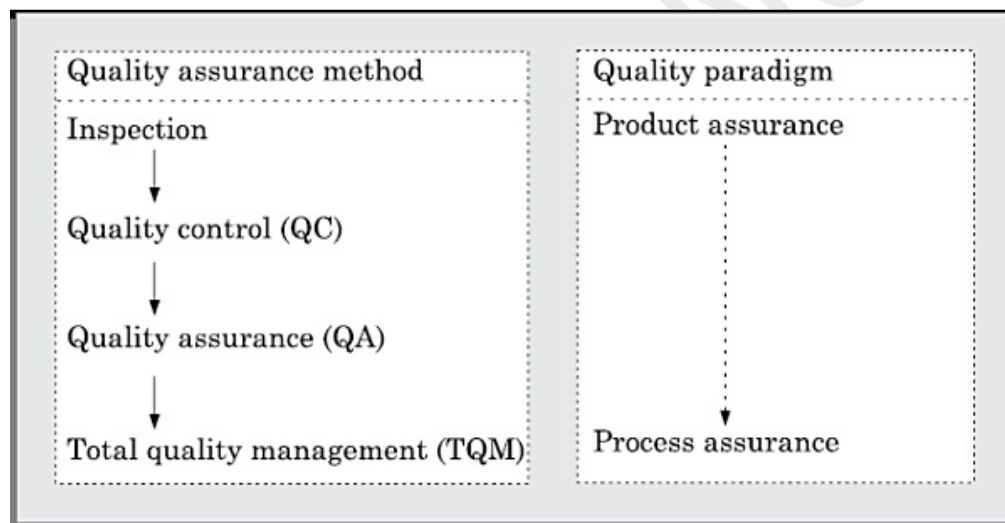


Figure 6.3: Evolution of quality system and corresponding shift in the quality paradigm.

- **Quality paradigm** has shifted from product assurance to process assurance.

6.3.4 Product Metrics versus Process Metrics

- Product metrics help measure the **characteristics of a product being developed**, whereas process metrics help measure **how a process is performing**.

6.4 ISO 9000

- **International standards organisation (ISO)** is a consortium of 63 countries established to formulate and foster standardisation.
- ISO published its **9000 series** of standards in **1987**.

6.4.1 What is ISO 9000 Certification?

- ISO 9000 certification serves as a **reference** for contract between **independent parties**.
- A company awarding a **development contract** can form his opinion about the possible **vendor performance** based on whether the vendor has obtained **ISO 9000 certification or not**.
- ISO 9000 standard specifies **the guidelines for maintaining a quality system**.
- The ISO standard addresses both **operational aspects** (that is, the process) and **organisational aspects** such as responsibilities, reporting, etc.
- It specifies a **set of recommendations** for repeatable and high quality product development.
- is a set of guidelines for the **production process** and is not directly concerned about the **product** it self.
- ISO 9000 is a series of three standards—**ISO 9001, ISO 9002, and ISO 9003**.

The types of software companies to which the different ISO standards apply are as follows:

1. ISO 9001:

- This standard applies to the **organisations engaged in design, development, production, and servicing of goods**.
- Standard is applicable to most **software development organisations**.

2. ISO 9002:

- This standard applies to those organisations **which do not design products but are only involved in production**.
- Examples of this category of industries include **steel and car manufacturing industries** who buy the product and plant designs from external sources and are involved in only manufacturing those products.
- It is not applicable to **software development organisations**.

3. ISO 9003:

- This standard applies to organisations involved only in **installation and testing of products**.

6.4.2 ISO 9000 for Software Industry

- ISO 9000 is a **generic standard** that is applicable to a large amount of industries, starting from a **steel manufacturing industry to a service rendering company**.

- Many of the clauses of the **ISO 9000 documents** are written using **generic terminologies** and it is very **difficult to interpret** them in the context **of software development organisations**.
- differences between software development and development of other kinds of products

Sr.no.	software development	other kinds of products
1	Software is intangible and therefore difficult to control. It means software would not be visible to the user until the development is complete and the software is up and running.	product manufacturing such as car manufacturing, you can see a product being developed through various stages such as fitting engine, fitting doors, etc. Therefore, it becomes easy to accurately determine how much work has been completed and to estimate how much more time will it take
2	During software development, the only raw material consumed is data.	large quantities of raw materials are consumed during the development of any other product.

6.4.3 Why Get ISO 9000 Certification?

There is a mad scramble among software development organisations for obtaining ISO certification due to the benefits it offers.

1. Confidence of customers in an organisation increases when the organisation qualifies for ISO 9001 certification. This is especially true in the international market. In fact, many organisations awarding international software development contracts insist that the development organisation have **ISO 9000 certification**.
2. ISO 9000 requires a well-documented software production process to be in place.
3. A well-documented software production process contributes to repeatable and higher quality of the developed software.
4. ISO 9000 makes the development process focused, efficient, and cost-effective.

6.4.4 How to Get ISO 9000 Certification?

An organisation intending to obtain ISO 9000 certification applies to a ISO 9000 registrar for registration.

The ISO 9000 registration process consists of the following stages:

- 1. Application stage:** Once an organisation decides to go for ISO 9000 certification, it applies to a registrar for registration.
- 2. Pre-assessment:** During this stage the registrar makes a rough assessment of the organisation.
- 3. Document review and adequacy audit:** During this stage, the registrar reviews the documents submitted by the organisation and makes suggestions for possible improvements.
- 4. Compliance audit:** During this stage, the registrar checks whether the suggestions made by it during review have been complied to by the organisation or not.
- 5. Registration:** The registrar awards the ISO 9000 certificate after successful completion of all previous phases.
- 6. Continued surveillance:** The registrar continues monitoring the organisation periodically.

6.4.5 Salient Features of ISO 9001 Requirements

The salient features of ISO 9001 requirements as follows:

1. Document control:

- All documents concerned with the development of a software product should be properly managed, authorised, and controlled. This requires a configuration management system to be in place.

2. Planning:

- Proper plans should be prepared and then progress against these plans should be monitored.

3. Review:

- Important documents across all phases should be independently checked and reviewed for effectiveness and correctness.

4. Testing:

- The product should be tested against specification.

5. Organisational aspects:

- Several organisational aspects should be addressed e.g., management reporting of the quality team.

6.4.6 Shortcomings of ISO 9000 Certification

Shortcoming of the ISO 9000 certification process are the following:

1. ISO 9000 requires a **software production process** to be adhered to, but does not guarantee the process to be of high quality. It also does not give any **guideline for defining an appropriate process**.
2. ISO 9000 certification process **is not fool-proof and no international accreditation agency** exists. Therefore it is likely **that variations** in the norms of **awarding certificates** can exist among the different accreditation agencies and also among the registrars.

3. Organisations getting **ISO 9000 certification** often tend to **downplay domain expertise and the ingenuity of the developers**.

4. ISO 9000 does not automatically lead to **continuous process improvement**. In other words, it does not automatically lead to TQM.

6.5 SEI CAPABILITY MATURITY MODEL

- **SEI capability maturity model (SEI CMM)** was proposed by Software Engineering Institute of the Carnegie Mellon University, USA.
- CMM is patterned after the pioneering work of Philip Crosby who published his maturity grid of five evolutionary stages in adopting quality practices in his book “Quality is Free” [Crosby79].
- CMM is a reference model for apprising the software process maturity into different levels.
- This can be used to predict the most likely outcome to be expected from the **next project that the organisation undertakes**.
- It must be remembered that **SEI CMM** can be used in two ways— **capability evaluation and software process assessment**.
- SEI CMM classifies software development industries into the following five maturity Levels:

Level 1: Initial

- A software development organisation at this level is characterised by **ad hoc activities**.
- Very few or no processes are defined and followed.
- Since software production **processes are not defined**, different engineers follow their own process and as a result development efforts become chaotic.
- The success of projects depends **on individual efforts and heroics**.
- When a developer leaves the organisation, the successor would have great difficulty in **understanding the process** that was followed and the **work completed**.
- **No formal project management** practices are followed. As a result, time pressure builds up towards the end of the delivery time, as a result short-cuts are tried out leading to low quality products.

Level 2: Repeatable

- At this level, the basic project management practices such as tracking **cost and schedule** are established.
- Configuration management tools are used on items identified for **configuration control**.
- **Size and cost estimation** techniques such as function point analysis, COCOMO, etc., are used.
- The necessary **process discipline** is in **place to repeat earlier success** on projects with similar applications.
- Process is not documented.
- **Configuration management practices** are used for all project deliverables.

- If products are very similar, the **success story** on development of one product can be repeated for another.
- In a nonrepeatable software development organisation, a software product development project becomes successful primarily due to the initiative, effort, brilliance, or enthusiasm displayed by certain individuals.

Level 3: Defined

- At this level, the **processes for both management and development activities are defined and documented.**
- There is a common organisation-wide understanding of **activities, roles, and responsibilities.**
- The processes though defined, the **process and product qualities** are not measured.
- At this level, the organisation builds up the **capabilities** of its **employees** through **periodic training programs.**
- **Review techniques** are **emphasized and documented** to achieve phase containment of errors.
- ISO 9000 aims at achieving this level.

Level 4: Managed

- At this level, the focus is on **software metrics.**
- Both **process and product metrics** are collected.
- **Quantitative quality** goals are set for the products and at the time of completion of development it was checked whether the quantitative quality goals for the product are met.
- Various tools like **Pareto charts, fishbone diagrams**, etc. are used to measure the **product and process** quality.
- The **process metrics** are used to check if a **project performed satisfactorily.** Thus, the results of process measurements are used to evaluate project performance rather than improve the process.

Level 5: Optimising

- At this stage, **process and product metrics** are collected.
- Process and product measurement data are **analysed for continuous process improvement.**
- The lessons learned from **specific projects** are **incorporated into the process.**
- Continuous **process improvement** is achieved both by carefully analysing the **quantitative feedback** from the **process measurements** and also from application of innovative ideas and technologies.
- At CMM level 5, an organisation would identify the **best software engineering practices and innovations** (which may be tools, methods, or processes) and would transfer these organisation wide.

- Level 5 organisations usually have a department whose sole responsibility is to **assimilate latest tools and technologies and propagate them organisation-wide.**

Except for level 1, each maturity level is characterised by **several key process areas (KPs)** that indicate the areas an organisation should focus to improve its software process to this level from the previous level. Each of the focus areas identifies a number of key practices or activities that need to be implemented. The focus of each level and the corresponding key process areas are shown in the Table 6.1:

<i>CMM Level</i>	<i>Focus</i>	<i>Key Process Areas (KPs)</i>
Initial	Competent people	
Repeatable	Project management	Software project planning Software configuration management
Defined	Definition of processes	Process definition Training program Peer reviews
Managed	Product and process quality	Quantitative process metrics Software quality management
Optimising	Continuous process improvement	Defect prevention Process change management Technology change management

Table 6.1 Focus areas of CMM levels and Key Process Areas

CMM Shortcomings: CMM does suffer from several shortcomings. The important among these are the following:

- The most frequent complaint by organisations while trying out the CMM-based process improvement initiative is that **they understand what is needed to be improved, but they need more guidance about how to improve it.**
- Another shortcoming (that is common to ISO 9000) is that **thicker documents, more detailed information**, and longer meetings are considered to be better. This is in contrast to the principles of software economics—reducing complexity and keeping the documentation to the minimum without sacrificing the relevant details.
- Getting an accurate measure of an organisation's current maturity level is also an issue.

6.6 SIX SIGMA

6.6.1 History of Six Sigma

Six-Sigma is a set of methods and tools for process improvement. It was introduced by Engineer **Sir Bill Smith** while working at **Motorola** in 1986. In the 1980s, **Motorola** was

developing Quasar televisions which were famous, but the time there was lots of defects which came up on that due to picture quality and sound variations.

6.6.2 Characteristics of Six Sigma

The Characteristics of Six Sigma are as follows:

1. **Statistical Quality Control:** Six Sigma is derived from the **Greek Letter σ (Sigma)** from the Greek alphabet, which is used to denote **Standard Deviation** in statistics. Standard Deviation is used to **measure variance**, which is an essential **tool** for measuring **non-conformance** as far as the **quality** of output is concerned.
2. **Methodical Approach:** The Six Sigma is not a merely quality improvement strategy in theory, as it features a well defined systematic approach of application in DMAIC and DMADV which can be used to improve the quality of production. DMAIC is an acronym for **Design-Measure- Analyze-Improve-Control**. The alternative method DMADV stands for **Design-Measure- Analyze-Design-Verify**.
3. **Fact and Data-Based Approach:** The statistical and methodical aspect of Six Sigma shows the **scientific basis** of the technique. This accentuates essential elements of the Six Sigma that is a fact and data-based.
4. **Project and Objective-Based Focus:** The Six Sigma process is implemented for an organization's project tailored to its **specification and requirements**. The process is flexed to suits the **requirements and conditions** in which the projects are operating to get the best results.
5. **Customer Focus:** The customer focus is fundamental to the Six Sigma approach. The quality improvement and control standards are based on specific customer requirements.
6. **Teamwork Approach to Quality Management:** The Six Sigma process requires organizations to get organized when it comes to controlling and improving quality. Six Sigma involving a lot of **training** depending on the role of an individual in the Quality Management team.

6.6.3 Six Sigma Methodologies

Six Sigma projects follow two project methodologies:

1. DMAIC
2. DMADV

1. **DMAIC:** It specifies a data-driven quality strategy for improving processes. This methodology is used to enhance an existing business process.

The DMAIC project methodology has five phases:

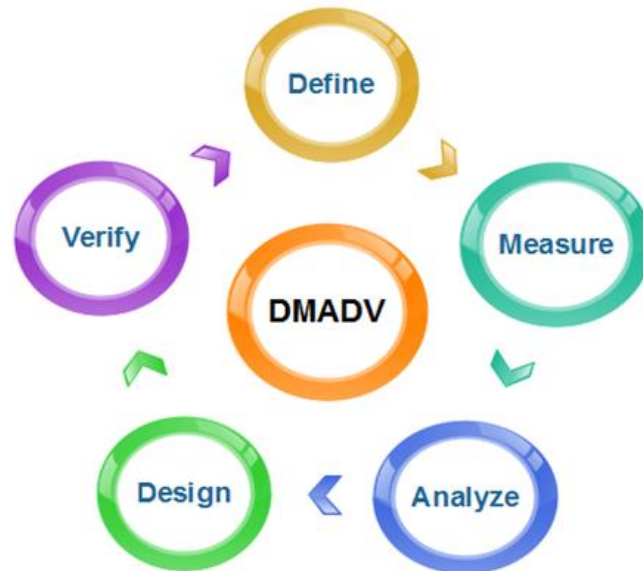


1. **Define:** It covers the process mapping and flow-charting, project charter development, problem-solving tools, and so-called 7-M tools.
2. **Measure:** It includes the principles of measurement, continuous and discrete data, and scales of measurement, an overview of the principle of variations and repeatability and reproducibility (RR) studies for continuous and discrete data.
3. **Analyze:** It covers establishing a process baseline, how to determine process improvement goals, knowledge discovery, including descriptive and exploratory data analysis and data mining tools, the basic principle of Statistical Process Control (SPC), specialized control charts, process capability analysis, correlation and regression analysis, analysis of categorical data, and non-parametric statistical methods.
4. **Improve:** It covers project management, risk assessment, process simulation, and design of experiments (DOE), robust design concepts, and process optimization.
5. **Control:** It covers process control planning, using SPC for operational control and PRE-Control.

2. DMADV

It specifies a data-driven quality strategy for designing products and processes. This method is used to create new product designs or process designs in such a way that it results in a more predictable, mature, and defect free performance.

The DMADV project methodology has five phases:



1. **Define:** It defines the problem or project goal that needs to be addressed.
2. **Measure:** It measures and determines the customer's needs and specifications.
3. **Analyze:** It analyzes the process to meet customer needs.
4. **Design:** It can design a process that will meet customer needs.
5. **Verify:** It can verify the design performance and ability to meet customer needs.

6.7 What is Extreme Programming (XP) and What are its Goals?

- Extreme Programming is a **systematic approach** with a **set of values, rules and practices for rapidly developing high quality software that provides the highest value for customers.**
- The goal of Extreme programming is to attempt **to reduce the cost of changes in requirements** by having multiple short development cycles, rather than a long one.
- Changes are a **natural, inescapable and desirable** aspect of software-development projects, and should be planned for, instead of attempting to define a stable set of requirements.

Extreme Programming (XP) is based on **values**. The following **Extreme Programming Values** are given below.

- **Simplicity:** What is needed will be done and no more, which will maximize value created for investment made to date. Small steps will be taken and failures will be mitigated as they happen. The project developed will be maintained long term at reasonable cost.
- **Communication:** Everyone is part of the team, and daily communication is face to face. Team works together on everything from requirements to code. The best solution to problem will be developed together.

- **Feedback:** Every iteration commitment will be taken seriously to deliver working software. Software is demonstrated early while the developers listen carefully to feedback and make any changes necessary.
- **Respect:** Everyone gives and receives the respect deserved as a valued team member. Management respects our right to accept responsibility and receive authority over our own work.
- **Courage:** We will tell the truth about progress and estimates. We don't document excuses for failure because we plan to succeed. We don't fear the job at hand because no one works alone. We adopt to changes whenever they happen.

Rules of Extreme Programming (XP) Methodology - Promoting Team Collaboration and Empowerment:

The most surprising aspects of Extreme Programming Methodology are its simple rules. The rules for Extreme Programming Methodology are outlined and explained below.

1. Planning:

- In the planning stage **user stories** are written.
- User stories are used to create **time estimates for release planning meeting**.
- In this stage developers are frequently **releasing small releases of iterative versions** of the system to customers.
- And project is divided into **iterations**.
- Iterations are scheduled for **one to three weeks** in length.

2. Managing:

- This rule requires that the work **space be open and barriers** such as **cubicles** that divide people be taken down.
- Setting up pace is having the most completed and production ready software each iteration.
- This rule also requires **to move people around** to avoid serious **knowledge loss and coding bottle necks**.
- This rule also requires a **stand up meeting**.
- most attendees do not **contribute, but attend just to hear the outcome**.
- A large amount of developer time is sacrificed to gain a trivial amount of communication.

3. Designing:

- This rule requires that designing a model must **have simplicity to it**.
- A simple model always takes less to finish than a complex one.

- **Spike solutions** are required by this rule which serve to figure out **answers** to tough technical and design problems.
- A spike solution is a very **simple program** to explore potential solutions.
- This rule also requires for **functionality** to never be added early since only 10% of the extra stuff will ever get used.

4. Coding:

- This rule requires that the customer always be available, not only to help the development team but to be a part of it as well.
- This rule also requires **code to be written to agreed standards**.
- It is required of the **Coding rule** to create the test first before creating the code because it will be much easier to create the code later.
- Collective ownership encourages everyone to contribute new ideas to all segments of the project.

5. Testing:

- All code must undergo **unit tests**, **unit tests** are the corner stones of Extreme Programming.
- When a **bug is found tests** are created to guard against it coming back.
- Acceptance tests are created from user stories. During iteration the user stories selected during the iteration planning meeting will be translated into acceptance tests.

6.8 Agile project management

Agile project management is an **iterative approach** that focuses on **frequent value delivery and getting fast feedback** from the market to adapt to emerging changes quickly. It focuses on:

- working on small batches;
- visualizing processes to create transparency;
- collaborative working with the customer, and
- getting feedback as fast as possible.

This allows you to promptly adapt to changing requirements and produce higher-quality products or services to better satisfy your customer's needs.