Unit I – Introduction

❖ Introduction :-

- Introduction to Quality :-
  - Earlier product was governed by the individual skill and it differed from instance to instance depending upon the creator & the process used to make it at instance.
  - Every product was considered as separate project & every instance of the manufacturing process led to products of different quality attributes.
  - Due to increase in demand for same or similar products which were expected to satisfy similar mechanism of the manufacturing products.
  - The market is changing considering from monopoly to fierce competition nut still maintaining the individual product identity but still maintaining the individual product identity in terms of attributes & characteristics.
  - There are large number of buyers and servers in market providing & demanding similar products which satisfy similar demands.
  - Products may or may not be exactly the same but may be similar or satisfying similar needs/demands of the user.
  - They may differ from one another to some extent on the basis of their cost, deliver : Schedule to acquire it, features, functionality present/absent.
  - These may be termed as attributes of the quality of a product.

- Historical Perspective of Quality :-
  - The term quality means different things to different times, different places and for different products.
  - For example : to some users, a quality product may be one, which has no/less defects and works exactly as expected and matches with his/her concept of cost and delivery schedule along with services offered. Such a though may be a definition of quality. Quality is fitness for use.
  - Quality defined as conformance to specification is a position that people in the engineering industry often promote because they can do very little change the design of a product and have to make product as per the

design which will best suite the user's other expectations like less cost, fast delivery and good service support.

- Quality is the extent to which the customers/ users believe that a product meets or surpasses their needs and expectations.
- Other believe that quality means delivering products that :

1. Meet customers standards either defined by the customer or defined by the normal usage or by some national or international bodies.
2. Meet and fulfill customer needs which include expressed & implied requirements delivered by business analytics and system analysts.
3. One must try to meet customer expectations as maximum as possible. If something is given than the customer requirement, it should be declared before transition so that customer surprises can be avoided.
4. Meet anticipated/unanticipated future needs and aspirations of customers by understanding their business & future plans. One may need to build a software & system considering some future requirements. Every product has some defined life span & one may have to extrapolate future needs accordingly.

- Definitions of Quality :-
  - For achieving quality of product one must define it in some measurable terms which can be used as reference to find whether quality is met or not.
  - There are many views and definitions of quality given by stalwarts working in quality improvements & quality management arena.
  - Some of these are :-

1. Customer – Based Definition of Quality :-

   Quality product must have fitness for up and must meet customer needs, expectations and help in achieving customer satisfaction & possible customer delight.

2. Manufacturing - Based definition of Quality :-

   This definition is mainly derived from engineering product manufacturing where it is not expected that the customer knows all requirement of the product and product level requirements are defined by architects and designers on the basis of customer feedback/survey.

3. Product – Based Definition of Quality :-

The product must have something that the other similar products do not have which can help the customer satisfy his/her need in a better way.

4. Value – Based Definition of Quality :-

   A product is the best combination of price and features or attributed expected by or required by the customer. The customer must get value for this investment by buying the product.

5. Transcendent Quality :-

   To many users/customers it is not clear what is meant by a quality product but as per their perception it is because of some quality present/absent in product.

   - A product must zero/minimum defects so that it does not prohibit normal usage by the uses.
   - A product must be something that people will want to receive as it satisfies their needs and support expectations. It must suffice the purpose of its existence from customer view.

- Core Components of Quality :-
  - Quality of a product must be driven by customer requirements and expectations from the product. Those expectations may be expressed as a part of requirement specifications defined or may be implied which is generally accepted as the requirements.
  - Some postulates of quality are :

1. Quality is based on customer satisfaction by acquiring a product.
   - Quality is something perceived by a customer while using a product.
   - The effect of a quality product delivered and used by a customer, on his satisfaction & delight is the most important factor in determining whether the quality has been achieved or not.
   - It talks about the ability of a product or service to satisfy customer by offering his needs or purpose for which it is acquired.
   - It may come through the attributed of a product, time required for customer to acquire it, price a customer is expected to pay for it & so many other factors associated with the product as well as the organization producing it.
   - Producer must understand the purpose or usage of a product and then devise a quality plan for it accordingly to satisfy the purpose of the product.

2. The organization must define quality parameters before it can be achieved.
   In order to meet some criteria of improvement and ability to satisfy a customer one must follow cycle of define, measure, control & improve.

i. Define : -
   There must be definition of what is required in the product in terms of attributes of a product & in how must quantity it is required to derive customer satisfaction.

ii. Measure :-
   The quantitative measure must be defined as an attribute of a product.

iii. Monitor :-
   Ability of the product to satisfy customer expectations defines the quality of a product.

iv. Control :-
   Control gives the ability to provide desired results & avoid the undesired things to a customer.

v. Improve :-
   Continuous improvements are necessary to maintain ongoing customer satisfaction & overcome the possible competition, customer complaints.

3. Management must lead the organization through Improvement Efforts :
   Quality must be perceived by a customer to realize customer satisfaction.

4. Continuous process (Continual) Improvements is necessary :
   The first step for producing quality is the definition of processes used for producing the product & the cycle of continuous improvement to refine processes to achieve target improvements.

Sr. No	Continuous Improvements	Continual Improvements
1	Continuous improvements is dynamic in nature.	Continual Improvements is dynamic & static change management.
2	The changes are done at every stage & every time.	The changes are done, absorbed baselined & sustained before taking next step of improvement.
3	Continuously striving for the excelence gives a continuously improvement.	Periodic improvements followed by stabilization process.
4	It has high dependence on	Less dependence on people and

	people having innovative skills tending towards inventions.	more dependence on innovation processes.
5	Environment is continuously changed.	Changed in environment are followed by stabilization.

❖ Quality View :-

Stakeholders are the people or entities interested in successful failure of a project or product or organization in general. Quality is viewed differently by different stakeholders.

1. Customer's view of quality :-
   - Customer is the main stakeholder for any product. The customer will be paying for the product interprets customer requirements & expectation for getting a better product at defined schedule, cost & with adequate service along with required features & functionalities :
   i. Delivering right product
   ii. Satisfying customers needs
   iii. Meeting customer expectations.
   iv. Treating every customer with integrity, courtesy & respect.
2. Supplies view of quality :-

   Suppliers give inputs for making product/project. Suppliers can be external or internal to the organization. Supplier expectations may range from profitability, name in market repeat orders, customer satisfaction. These expectations may be different ways.

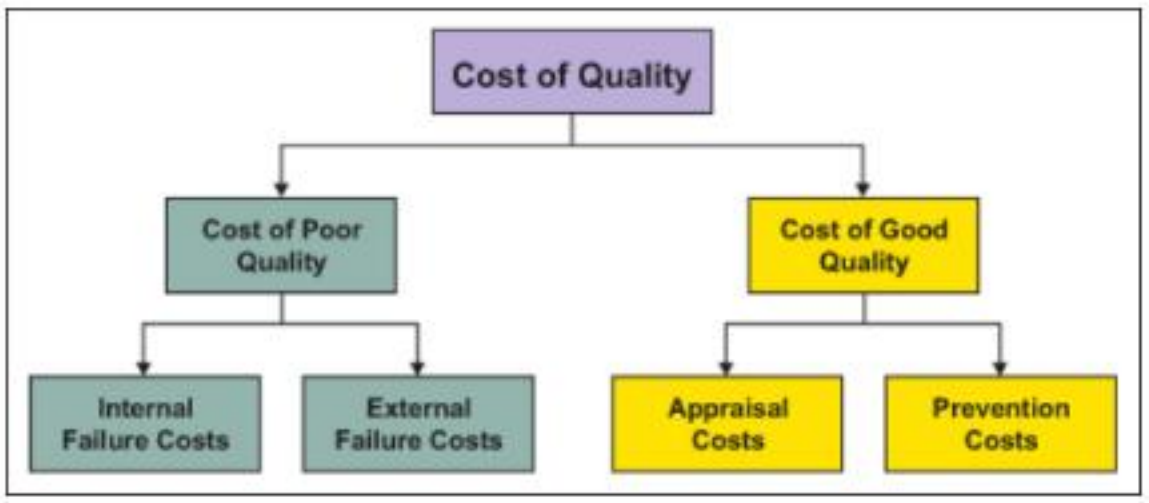
   1. Doing the right things.
   2. Doing the right way.
   3. Doing the right the first time.
   4. Doing it on time.

❖ Financial Aspect of Quality :-

- Earlier people were of the opinion that more price of a more inspection, testing, sorting and ensures that only good part are supplier to customer.
- Sales price was defined as :
  Sales price = cost of manufacturing + cost of quality + profit.

- Reducing sales prices reduces percentage profit. For maintaining profit the producer may try to reduce the cost of production without compromising on the quality aspect.
- Thus in competitive environment the equation changes to ,
  Profit = Sales Price – [cost of manufacturing + cost of quality]

1. Cost of manufacturing :-
   - It is cost required for developing the right product by right method at first time. The money involved in resources like material, people, licenses, etc. forms of cost manufacturing.
   - The cost involved in requirement analysis, designing, development & coding are the cost associated with manufacturing.
2. Cost of quality :-
   - Cost of quality represents the part of cost of production incurred in improving quality of product.
   - Cost of quality includes all the efforts and cost incurred in prevention of defects, appraisal of product to find whether it is suitable to customer or not and fixing of defects or failures at various levels as any when they are reported and conducting any retesting, regression testing etc.
   1. Cost of prevention
   2. Cost of appraisal
   3. Cost of failure
   - On the basis of quality focus organizations may be placed in 2 categories viz. organizations which are less quality conscious (termed as 9) and organizations which are more quality conscious (termed as 'Q'). The distribution of cost of quality for these two types may be represented as below figure.



❖ Customers, Suppliers and Processes :-

- For any organization, there are some suppliers supplying the inputs required and some customers who will be buying the output produced.
- Suppliers and customers may be internal or external to the organization.
- External suppliers provide input to the organization and external customer receive the output of the organization.
- Suppliers may be customers for some other organizations and customers may be acting as suppliers for somebody else down in line.

1. Internal Customer :-

- Internal customers are in the functions and projects serviced and supported by some other functions/projects.
- System administration may have projects as their customer while purchasing as their customer while may have their customer.
- During value chain each function must understand its customers and suppliers.

2. External Customer :-

- External customer are the external people to the organization who will be paying for the services offered by organization.
- There are the people who will be actually buying products from the organization.

❖ Total Quality Management (TQM) :-

- 'Total Quality Management' principle intends to view internal & external customers as well as internal & external suppliers for each process, project and for entire organization as a whole.
- Each supplier eventually also becomes a customer at some other moment and vice versa.
- If one can take care of his/her customer with an intention to satisfy him it may result into customer satisfaction and continual improvement for the organization.

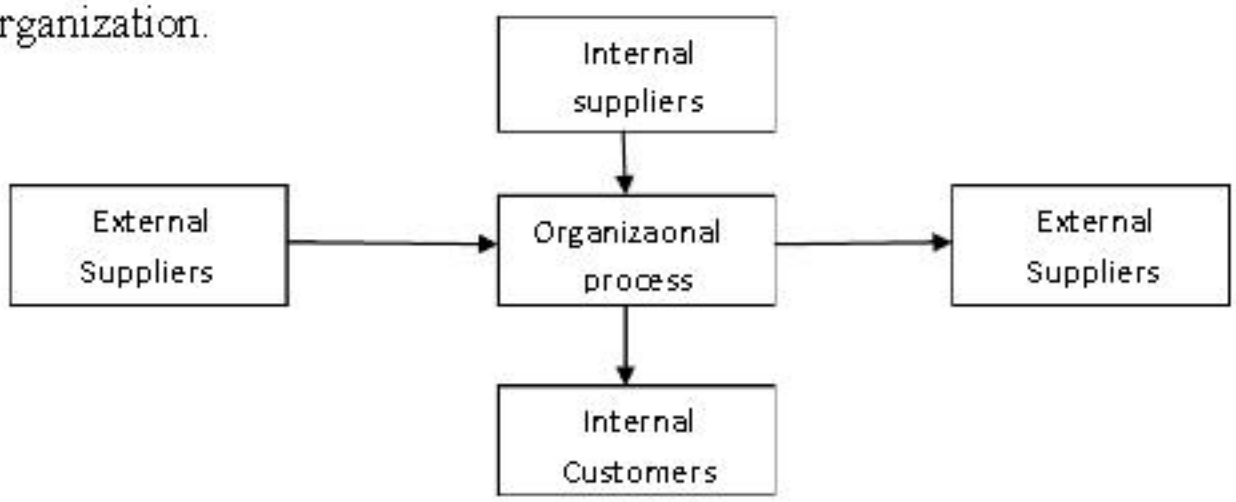


Figure: Supply chain

- TQM is the application of quality principles to all facets and business process of an organization.
- It talks about applying quality methods to the entire organization whether a given function or part of the organization faces external customers or not.

- Quality Practices in TQM :-
  - TQM works on some basic principles of quality management definition and implementation. These have evolved in over a span of exper4imentation and deployment of quality culture in organizations.

1. Develop constancy of purpose of definition & deployment of various initiatives.
2. Adapting to new philosophy of managing people/stakeholders by building confidence & relationships.
3. Declare freedom from mass inspection of incoming produced output.
4. Stop awarding of lowest price tag contracts to suppliers.
5. Improve every process used for development & testing of product.
6. Institutionalize training across the organization for all people.
7. Institutionalize leadership throughout organization at each level.
8. Drive out fear of failure from employees.
9. Break down barriers between functions/department
10. Eliminate exhortations by numbers, goals, targets.
11. Eliminate arbitrary numerical targets which are not supported by process.
12. Permit pride of workmanship for employees.
13. Encourage education of new skills and techniques
14. Top management commitment & action to improve continually.

❖ Quality management through Statistical process control :-

- Dr. Joseph Juran is a pioneer of statistical quality control with a definition of improvements cycle through define, measure, monitor, control and improve (DMMCI).
- There are three approach to understand the interrelationships among customers, suppliers & processes used in development, testing, etc & establish quality management based on metrics program.

1. Quality planning at all level :-

- Quality is not an accident but is a result of deliberate effort to achieve something which is defined in advance.
  - i. Quality planning at organization level :- Quality must be planned first in the form of policy definition and strategic quality plans on the basis of vision, missions and policies set by senior management.
  - ii. Quality planning at unit level :-
    - Quality planning at unit level must be done by the people responsible for managing unit.
    - Operational quality plans must be sink with organizational policies and strategies.

2. Quality Control :-

- Quality Control process attempts to examine the present product at various levels with the defined standards so that an organization may appraise the outcome of the processes.

3. Quality Improvement :-

- Improvement process attempts to continuously improve the quality of the process used for producing products.

❖ Quality management through cultural changes :-

- Quality improvement is based on cultural change in an organization towards total quality management.
- Quality management through cultural change defines quality improvements as cultural change driven by management.
- It involves :

1. Identifying areas in which quality can be improved depending upon process capability measurements and organizational priorities.

2. An organizational must set up cross functional working groups and try to improve awareness about the customer needs, quality and process measurements.
3. Instituting teams representing different functions & areas for quality improvement can help in setting the change of culture.
4. Improving quality of the processes of development testing, managing etc is team work lead by managing directives.
5. Setting measurable goals in each area of an organization can help in improving processes at all level.
6. Giving recognition to achievers of quality goals boost their morale and set a positive competition among the team leading to organizational improvements.
7. Repeating quality improvements cycle continuously by stretching goals further for next phase of improvements is required to maintain & improve status further.
8. The organization must evaluate the goals to be achieved in short term, long term & combination of both to realize organizational vision.

❖ Continual Improvement cycle :-

- Plan, Do, Check and Act (PDCA) Cycle :-
  - Continual (continuous) improvement cycle is based on systematic sequence of plan-do-check activities representing a never ending cycle of improvements.
  - PDCA cycle was initially implemented in agriculture.
  - It was later implemented in electronic industry.
  - TQM has made the PDCA cycle famous in all industries.
  - Stages of continual improvement through PDCA cycle are :-

1. Plan :-

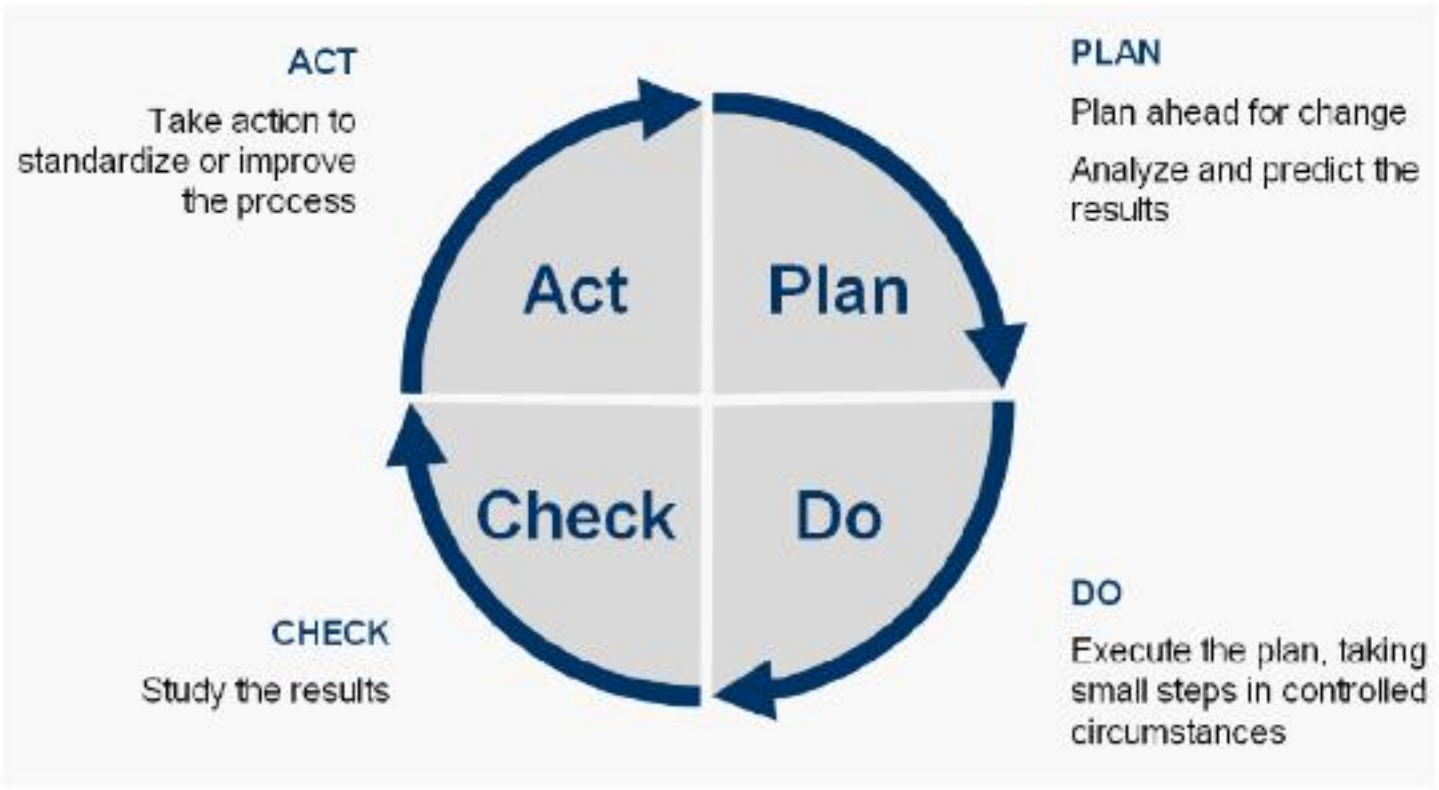
- An organization must plan for improvements of the basis of its vision and mission definition.
- Planning includes answering all questions like who, when, where, why about various activities & setting expectations.

2. Do :-

- An organization must work in the direction set by the plan devised in earlier phase for improvements.
- Plan is not everything but a roadmap.
- It sets a direction but execution is also important.
- Plans sets the tone while execution makes the plan work.
- Check :- An organization must compare actual outcome of Do stage with reference or expected results which are planned.

3. Act :-

- If any deviations (positive or negative) are observed in actual outcome with respect to planned results the organization may need to decide actions to correct situation.
- When expected results and actual match with degree of variation, one may understand plan is going on in right decision. Running faster & slower plan need a action.



❖ Quality in Different areas :-

- Different domains need different quality factors. They may be arrived from customers/users of the domain.
- Definition of quality expectations will vary from instance to instance depending on the domain under consideration.

- Products and expected attributed :

Sr. No.	Product/service Category	Expected attributes
1	Airlines Industry	On time arrival & departure, comfortable journey, low cost service, reliability & safety.
2	Health care Industry	Correct diagnosis and treatment, minimum wait for time, lower cost, safety & security.
3	Food Service Industry	good product, good taste, fast delivery, good ambience, clean environment.
4	Consumer Products Industry	Properly made to suite individuals, defect free products, cost effective.
5	Military Services	Rapid deployment, decreased wages & cost security.
6	Automotive Industry	Defect free product, less fuel consumption, more power, safe journey.
7	Communications Industry	Clear communications, faster access, cheaper service

❖ Benchmarking :-

- Benchmarking :-
  - It is an important concept used in quality function deployment (QFD).
  - IT is the concept of creating qualitative/quantitative metrics or measurable variables which can be used to assess product quality on several scales against a benchmark.
  - Typical variables of benchmarking includes price of a product paid by customer, time require to acquire it, customer satisfaction, defects of product.
- Metrics :-
  - Metrics are defined for collecting information about the product capabilities, process variability & outcome of the process in terms of the attributes of products.
  - Metric is relative measurement of some parameter of a product which are related to the product & processes used to make it.

- An organization must develop consistent set of metric derived from its strategic business plan and performance of benchmark partner.

❖ Problem Solving Techniques :-

- Improving quality of products and services offered to customers requires methods and the techniques of solving problems associated with development and processes used during their lifecycle.
- Problem solving can be accomplished by both qualitative and quantitative methods.

1. Qualitative problem solving :
   - It refers to understanding a problem solution using only qualitative indexes such as high medium low depending on whether something is improving from present status & so forth.
   - This is typical scenario for low maturity organizations where the problems are much broader.
   - It saves time in defining & measuring data accurately & basic maturity can be achieved.
2. Quantitative problem solving :-
   - It requires specification of exact measures in numerical terms such as cost has increased 32.5% during the last quarter or the time required to produce one product unit is reduced by 32 minutes.
   - For highly matured organizations, quantitative analysis is required for further improvements on basis are already done.
   - Measurement of processes and products may need good measuring instruments with high level of accuracy & repeatability.

❖ Problem solving software tools :-

While buying software for data management and statistical analysis many organizations find it to be a big investment in terms of money resources

Advantages of using software tools for analysis & decision making.

1. Accuracy and speed of the tools in much higher compared to performing all transaction & calculation manually.
2. Decision support offered by the tool is independent of personal skills & there is least variation from instance to instance.
3. Tools can implement theoretical means of assessing metrics about quality as defined by business law.

Disadvantages of using computer tools for analysis & decision making.

1. Training incurs cost as well as time.
2. All software/hardware's are prone for defects & these tools are not exceptions to it.
3. Decision has to be taken by human being and by the tool.
4. Tools may more cost & time to learn & implement

Tools :-

- Tools of an organizations analytical assets in understanding a problem through data & try to indicate possible solutions.
- Quality tools are more specific tools which can be applied to solving problems faced by projects.
- Tools may be hardware/software & physical tools.

Techniques :-

- Techniques indicates about a process used in measurement, analysis & decision making during problem solving.
- Techniques are independent of tools but they drive tool usage.

Tools	Techniques
Usage of tools is guided by the technique.	Techniques is independent of any tool.
Different techniques may use the same tool to achieve different results.	Same technique use different tools to achieve the same result.
Tool improvement needs technological change	Technique change can be effected through procedural change
Contribution of tools in improvement is limited	Contribution of techniques is improvement is important.

- Software Quality :-
  - Software Quality is conformance to explicitly stated and agreed functional & non-functional requirements.
  - It termed as quality for the software product offered to customers/final users from their perspective.
  - Customer may or may not be final user & sometimes, developers have to understand requirement of final users in addition to customer.
  - If the customer is a distributor or retail or facilitator who is paying for product directly then the final user's requirements may be interpreted by the customer to make the right product.

- Constraints of software product quality assessment :-
  - Generally requirements specifications are made by the business analysts & system analysis
  - Tester may or may not have direct access to the customer & may get information through requirement statements, queries answer from either customer or business system/analysts etc.
  - There are few limitations of product quality assessment in this scenario.

1. Software is virtual in nature software products cannot be seen, touched or heard.
2. There is a huge communication gap between users of software & testers of product.
3. Software is a product is unique in nature. Similarities between any two products are superficial ones.

- Customer is a King :-
  - The customer is the most important person in any process of developing a product and using it software development is not an exception.
  - All software life cycle processes such as development, testing, maintenance, etc are governed by customer requirements, whether implied or expressed.

- An organization must be dedicated to exceed customer satisfaction with the latter's consent.
- Exceeding customer satisfaction must not be received with surprise by the customer.
- A satisfied customer is an important achievement for an organization and is considered as an investment which may pay back in short & long term.
- Satisfied customers may give references to others & come back with repeat orders.
- Customer references are very important for developing new accounts.
- Organizations should try to implements some of the following measures to achieve customer satisfaction.

1. Internal customer & internal supplier : It is guided by the principles of total quality management.
2. External customer & external supplier : External customers may be the final users purchasers of software.

❖ Quality and Productivity relationships :-

- Quality of a product can be achieved by more inspection or testing, reworking, scrapping, sorting.
- More inspection cycles mean finding more defects mean better quality & ultimately the customer may get product.
- Quality improvements does not talk about product quality only but a process quality used for making such a product.
- If the processes of development & testing a good, a bad product will not be manufactured in the first place.

1. Improvement in quality directly leads to improved productivity.
2. The hidden factor producing scrap, rework, sorting, repair & customer complaint is closed.
3. Effective way to improve productivity is to reduce scrap & rework by improving process
4. Quality improvements leads to cost estimation.
5. Employees participate & contribute in improvement process.
6. Employee shares responsibility for innovation & quality improvement.

❖ Requirements of a product :-

- Everything done in software development, testing & maintenance is driven by the requirements of a product to satisfy customer needs.
- Requirements may be put in different categories.

1. Stated/Implied requirements :-

- Some requirements are specially documented in software requirement specifications while few others are implied ones.
- When we build software which will be there are functional & non-functional requirements specified by a customer/system analyst.

2. General/Specific requirements :-

- Some requirements are generic in nature which are generally accepted for a type of product & for a group of users while some other are very specific for the product under development.

3. Present/Future Requirement :-

- Present requirements are essential when an application is used in present circumstances while future requirements are for future needs which may be required after some time span.
- For projects, present as well as future requirements may be specifically defined by a customer or business/system analysts.

4. 'Must' and 'must not' Requirements or Primary Requirements :-

- Must requirements are primary requirements for which the customer is going to pay for while acquiring a product.
- These are essential requirements and the value of the product is defined on the basis of the accomplishment of 'must requirements'.

5. 'Should be' and 'should not be' Requirements or secondary Requirements :-

- Should be requirements are the requirements which may be appreciated by the customer if they are present/absent and may add some value to the product.
- Customer pay little bit extra for the satisfaction of these requirements but price of the product is not governed by them.

6. 'Could be' and 'could not be' requirements or tertiary requirements :-

- 'Could be' requirements are requirements which may add a competitive advantage to the producer but may not add much value in terms of price paid by a customer.
- If two products have everything same then could be requirements may help in better appreciation of a products by the users.

❖ Organization Culture :-

- An organization has a culture based on its philosophy for existence, management perception and employee involvement in defining future.
- Quality improvement programs are based on the ability of the organization to bring about a change in culture.
- Quality culture of an organization is an understanding of the organizations virtue about its people, customer, supplier and all stakeholders. 'Q' organizations are more quality conscious organization while 'q' organizations are less quality conscious organizations.
- The difference between 'Q' organization & 'q' organization.

Sr. No.	Quality Culture is 'Q'	Quality Culture is not 'q'
1	These organizations believe in listening to customers and determining their customers requirements.	These organizations assume that they know customer requirements
2	These organizations concentrate on identifying cost of quality & focusing on it reduce cost of failure which will reduce cost & price.	These organizations overtook cost of poor quality & hidden factor effect. They believe in more testing to improve product quality.
3	Doing things right for the first time & every time is the motto of success.	Doing things again & again to make them right is their way of working. Inspection, rework, scrap etc and essential.
4	These organizations believe in taking ownership of processes and defects at all levels.	These organizations try to assign responsibility to defects to someone else.
5	They demonstrate leadership & commitment to quality & customer	They believe in assigning responsibility for quality to others.

	satisfaction.	

- Shift in focus from 'q' to 'Q'
  - As the organization grows from 'q' to 'Q' there is cultural change in attitude of the management and employees towards quality & customer.
  - In initial stage at the higher side of 'q' a product subjected to heavy inspection, rework, sorting, scrapping etc to ensure that no defects are present in final deliverable to customer while final stage of 'Q' organization concentrate on defect prevention through process improvements.

Fig. Shift in focus from quality control to quality management.

1. Quality control approach is the oldest approach in engineering when a product was subjected to rigorous inspection for finding & fixing defects to improve it. Organizations at higher end of 'q' believe in finding and fixing defects to the extent possible as they improve quality of product.
2. Quality assurance approach (creation of process framework) is the next stage of improvement from quality control where the focus shifts from testing and fixing the defects to first time right.
3. Quality management approach in there are three kinds of system in the universe, viz completely closed systems, completely open systems and systems with semipermeable boundaries. Completely closed system represent that nothing can enter inside the system and nothing can go out of the system

- Characteristics of software :-

1. There are many products available in the market which are intended to satisfy same or similar demands.
2. There is a vast difference between software products & other products due to their nature.
3. Software cannot be sensed by common methods of inspection or testing. The product is in the form of executable which cannot be checked by any natural method available to mankind like touch, smell, hearing, taste etc. It needs testing in real environment but nobody can do exhaustive testing by anything all permutations & combinations.
4. There are different kinds of software products & their performance, capabilities etc vary considerably from each other. There are no same products though there may be several similar ones or satisfying similar needs.
5. Every product is different in characteristics, performance etc. software is always unique in nature.
6. Every condition defined by the software program gets executed in the same way every time when it gets executed. But the number of condition and algorithm combinations may be very large tending to infinity & testing of all permutations/combinations in practically impossible.

❖ Software Development Process :-

- Software development process defines how the software is being built.
- Some people also refer to SDLC as system development life cycle with a view that system is made of several components & software is one of the components.
- They are various approaches to build software it may be positive & negative points.

1. Waterfall development approach/model
2. Iterative development approach/model
3. Incremental development approach/model
4. Spiral development approach/model
5. Prototyping development approach/model
6. Rapid Application development approach/model
7. Agile development approach/model

1. Waterfall development approach/model :-
   - It is simplest software development model and is used extensively in development process study.
   - It is also termed as classical view of software development as it remains as a primary focus for study purpose or foundation of any development activity.

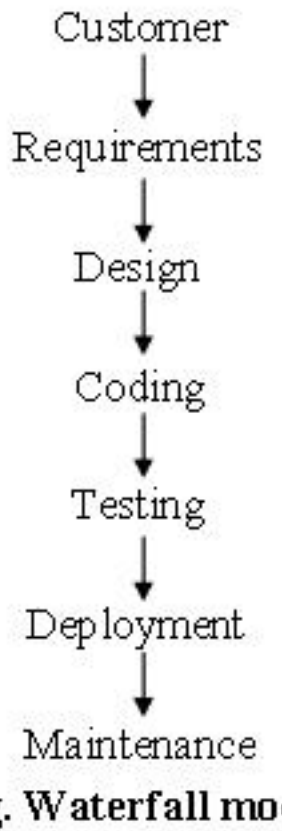


Fig. Waterfall model

- Arrows in the waterfall model are un-direction.
- It assumes that develops shall get all requirements from customer in single go.
- The requirements are converted into high level as well as low level designs.
- Design are implemented through coding.
- Code is integrated & executables are created.
- Executables are tested are per test plan.
- The final output in the form of an executable is deployed at customer premises.
- Waterfall models are used extensively in fixed price/fixed schedule projects where estimation is based on initial requirements. As the requirements changes, estimation is also revised.

- Limitations of Waterfall cycle :-
  - There is no feedback loop available in waterfall model.
  - It is assumed that requirements are stable and no problem is encountered during entire development life cycle.

2. Iterative Development approach/model :-
   - Iterative development process is more practical than the waterfall model.
   - activity.

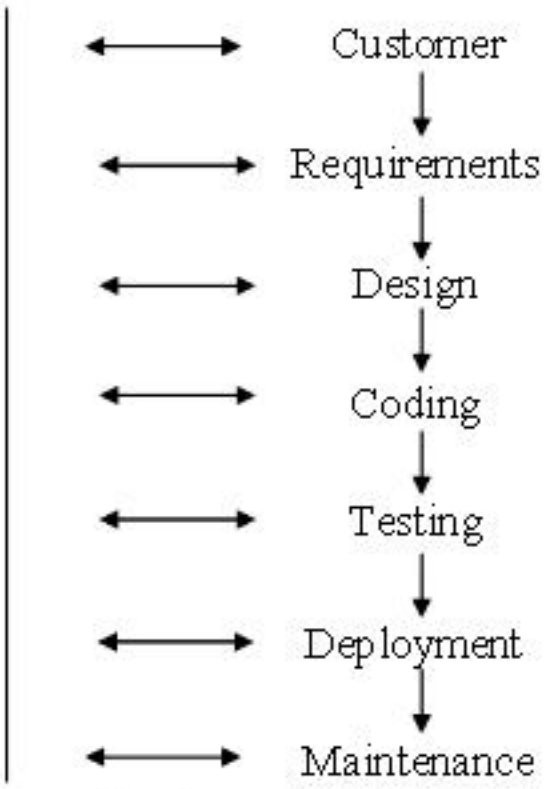


Fig. Iterative development model

   - It does not assumes that the customer gives all requirements in one go & there is complete stability of requirements.
   - It assumes that changes may come from any phase of development to any previous phase.

- Limitation of Iterative Development :-
  - Iterative development consists of many cycles of waterfall model.
  - It gives problems in fixed price projects for estimation.

3. Incremental Development Approach/Model :-
   - It is used in developing huge systems.
   - There are made of several subsystems which in themselves are individual systems.
   - Incremental systems may be considered as a collection of several subsystems.

- These systems may be connected to each other extremely either directly or indirectly.
- The incremental model gives flexibility to a customer.

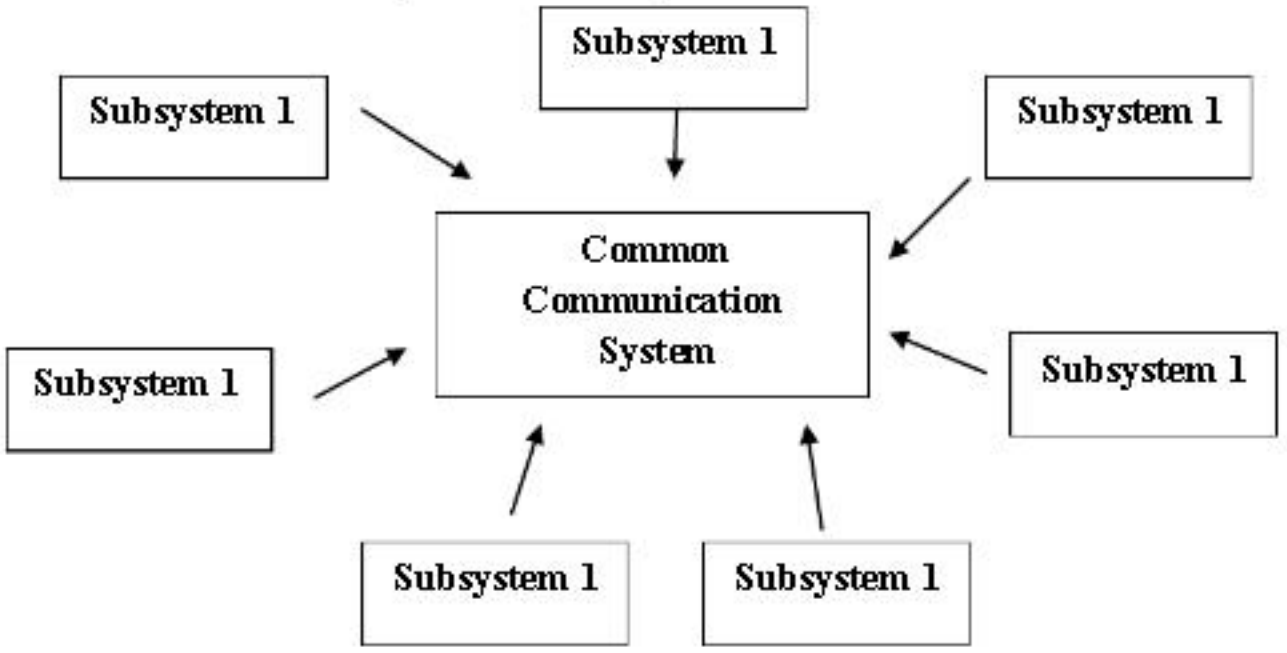


Fig. Incremental Model

- Limitations of Incremental Development :-
  - Incremental models with multivendor producer integration are a major challenge as parameter passing between different systems may be difficult big systems at the cost of loss of flexibility.
  - When a system is incremented with new subsystems, it changes the architecture of that system.

4. Spiral Development Approach/Model :-
   - Spiral development process assumes that customer requirements are obtained in multiple iterations & development also work in iterations.
   - Many big software systems are built by spiral models of ever increasing size.
   - First some functionalities are added then product is created & related to customer.
   - After getting the benefit of first iteration of implementation the customer may add another chunk of requirements to the existing one.
   - Further addition of requirements increase the size of software quality.

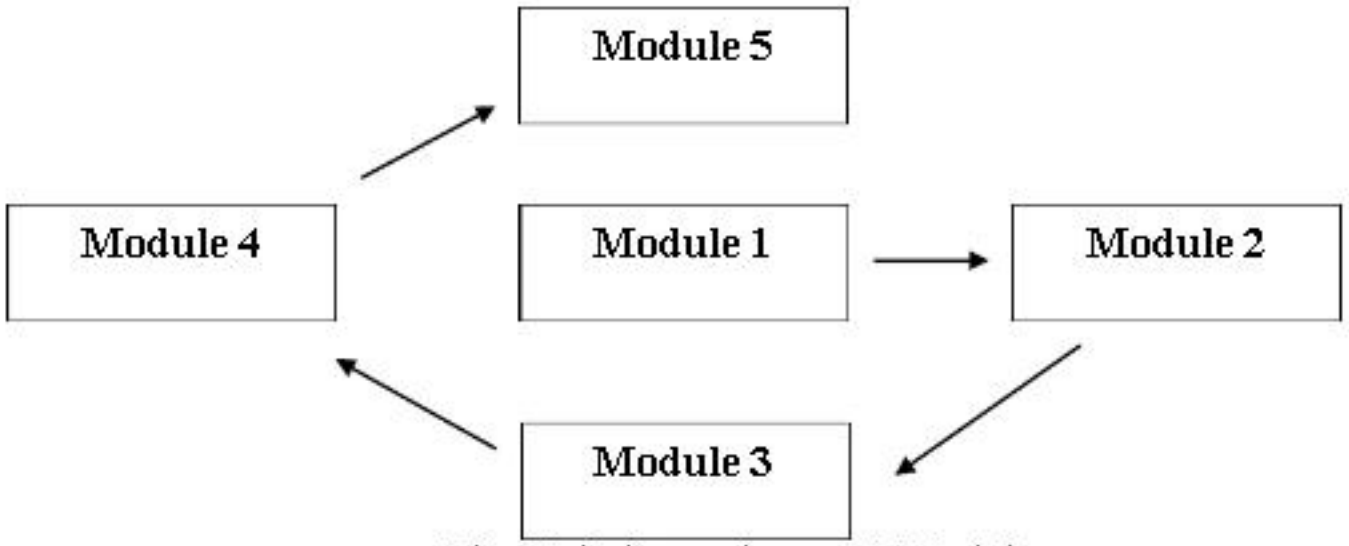


Fig. Spiral Development Model

- Spiral development is considered as a miniature form of the incremental model.

- Limitations of Spiral Development :-
  - Spiral models represent requirements election as the software is being developed.
  - Sometimes it may lead to refactoring & changes in approach where initial structure become non-usable.

5. Prototype Development Approach/Model :-
   - Prototype Development Approach represents top to bottom reverse integration approach.
   - Major problem of software development is procuring and understanding the customers requirement for the product.
   - Prototyping is one of the solution to help them.
   - In prototyping initially a prototype of the system is created this is similar to cardboard model of building.
   - It helps the customer to understand what they can expect from the given set of requirements.
   - It also helps the development team to understand the possible applications look & feel.
   - Once the election is done the logic is build behind it to implement the elicited requirements.

- Limitations of Prototype Development :-
  - Though one may get the feel of system by looking at the prototype one must understand that it is not actual system but a model.
  - The customer may get the feeling that the system is already ready & may deliver it immediately.

❖ Types of Product :-
  - Software development methodologies software products have some peculiarities defined as criticalities of software.
  - Critically of the software defines how much important it is to a user/customer.
  - There are few schemes of grouping the products on the basis of criticality to the user few of them are.

1. Life Affecting Products :-
  - Products which directly/indirectly affect human life considered as most critical products in the world from user's perspective.
  - Such products generally come under the preview or regulatory & safety requirements.
  - This type of product may be further grouped into 5 different categories.

i. Any product failure resulting into death of person. These will be most critical products as they can affect human life directly.
ii. Any product failure which may cause permanent disablements to a patient. These are second level critically products.
iii. Any product failure which may cause temporary disablements to a product.
iv. Any product failure which may cause minor injury & does not result into anything.
v. Other products which do not affect health or safety directly.

2. Product Affecting Huge Sum of Money :-
  - A product which has direct relationship with loss of huge sum of money is second in the list of criticality of the product.
  - Such products may need large testing efforts & have many regulatory & stationary requirements.

- E-Commerce & E-Business software's may be put in this category.
- Security, confidentiality & accuracy are some of the important quality factor for such large products.

3. Product which can be tested only by simulators :-

- Products which can't be tested in real life scenario but need simulated environment for testing are third in the ranking of critically.
- Products used in aeronautics, space research etc may be put in this category.
- Such products also need huge testing, although lesser than earlier two types.

4. Other Products :-

- Unfortunately 'critically' is not very easy to define.
- Let us consider an example of auto piloting software where we have three combinations of critically together.
- It does not affect the life of passengers travelling cost of an airplane is huge & it can't be tested in real environment.
- These all three criticalities coming together make a product most critical.

❖ Critically Definitions :-

- There may be several other ways of classifying criticality of a product.
- It has direct relationship with business dependability & extent of loss to the user organization/person in case of failure

i. From users perspective :-

- This classification mainly discuss dependency of a business on a system.
- Products failure which disrupts the entire business can be very critical from business point of view.
- Products failure which affects business partially as there may be some fallback arrangements is a type of criticality.
- Products failure which does not affect business at all is one of the options. If it fails one may have another method to achieve same result.

ii. Another way of defining users perspective :-

- The classification considers the environment in which the product is operating.

- It may range from very complex environment to very easy user environment to very easy user environment.
- Products where user environment is very complex such as aeronautics it may be considered as critical.
- Products where user environment is comparative less complex may represent the second stage of complexity.

iii. Criticality from developers perspective :-

- This classification defines the complexity of the system on the basis of development capabilities required.
- From based software where user inputs are taken & stored in some database.
- An & when required those inputs are manipulated & shown to user on screen or in the form of product.
- Algorithm based software where huge calculations the system on the basis of outcome of these calculations.

❖ Problematic Areas of Software Development Life cycle :

1. Problems with Requirement phase :-
   - Requirement gathering & elicitation is the most important phase in software development life cycle.
   - Many surveys indicate that the requirement phase introduces maximum defects in the product.
   - Problems associated with gathering are

   i. Requirements are not easily communicated
      - Communication major problem in requirement statement creation. Software development & implementation.
      - The types of requirements are technical requirements, economical requirements, legal requirements operational requirements, system requirements.

   ii. Requirements change very frequently :-
      - Requirements are very dynamic in nature.
      - There are many complaints by development team that requirement change is very frequent.

    - Many times developer get confused because customer requirements change frequently.

2. Generally a unique product is developed at each time.
    - The same implementation done by two different developers may differ from each other thus a software produced may be unique for that instance,
3. Intangible nature of a product, intellectual approach throughout development
    - Software products cannot be felt by normal senses. It existence can be felt by disc space it occupies.
4. Inspection can be exhaustive/impossible :-
    - While defining exhaustive inspection one may tend to include infinite permutations & combinations of testing. Testing of complete software product is impossible.
5. Effect of bad quality is not known immediately :-
    - Quality of software product cannot be improved by testing it again & again & finding & fixing the defects. It needs to be built in the product while development using good processes & methods.
6. Quality is inbuilt in product :-
    - Quality of a software product cannot be improved by testing it again & again & finding & fixing the defects. It needs to be built in the product while development using good processes & methods.
7. Quality objectives vary from product to product/customer to customer :-
    - Quality objectives define the expectations of customer/user and the acceptance level of various parameters which must be present in a given product for accepting/using it.
    - Quality objectives are product dependent time dependent & are mainly driven by customers of final users.

❖ Software Quality Management :-
- Quality management approaches talk about managing quality of a product or service using systematic ways and methods of development & maintenance.
- It is much above achieving quality factors as defined in software requirement specifications.

- Quality management involves management of all inputs & processing to the processes defined as so that the output from the process is as per defined quality criteria.
- There are three levels of handling problems,

i. Correction :-
   - Correction talks about the condition where defects found in the product or service are immediately sorted & fixed.
   - This is natural phenomenon which occurs when a tester defines any problem found during testing.
   - Manu organizations stop at fixing the defect though it may be defined by corrective action by them
   - This is mainly a quality control.

ii. Corrective Actions :-
   - Every defect needs an analysis to find the root causes for introduction of a defect in the system.
   - Corrective action identification & implementation is a responsibility of operations management group.
   - This is a quality assurance approach where process related problems are found & resolved to avoid recurrence of similar problems again & again.

iii. Preventive Actions :-
- On the basis of root cause of the problems other potential weak areas are identified.
- Preventative actions means that there are potential weak areas where defect has not been found till that point but there are potential weak areas weak defect has not been found till that defect. This is a quality management approach where an organization takes preventive action so that there is no defect in the first place.

❖ Why software has defects?
- One very important question about a product is 'why there are defects in the product at all?'
- There is no single answer to this question.

- After taking so much precautions of defining and implementing the processes doing verification & validation of each artifact during SLDC, yet nobody can claim that the product is free of any defects.
- In case of software developments and usage there are many factors responsible for its success failure few of them are

i. There are huge communication losses between different entities as requirements get converted into the actual product. Understanding of requirements is a major issue & majority of the defects can be attributed to this.

ii. Development people are more confident about their technical capabilities & do not consider that they can make mistakes. Sometimes self review & or peer review does not yield any defects.

iii. Requirement changes are very dynamic. As the traceability matrix is not available impact analysis of changing requirements becomes heuristic.

iv. Technologies are responsible for introducing few defects. There are many defects introduced due to browsers, platforms, databases etc.

v. Customer may not be aware of all requirements and idea develop as the product is used. Prototyping is used for clarifying requirements to overcome this

❖ Processes related to software quality :-
- Quality environment is an organization is established by the management.
- Quality management is a temple built by pillars of quality.
- Culture of an organization lays the foundation for quality temple.
- Every organization has different number of tiers or quality management system definition.

Fig. Relationship between vision, mission (s), policy (ies), objective (s), strategy (es), Goal (s) & value (s) of organization.

1. Vision :-
   - The vision of an organization is established by the policy management. Vision defines in brief about what the organization wishes to achieve in the given time horizon. Every organization must have a vision statement, clearly defining the ultimate aim it wishes to achieve with respect to time span.
2. Mission :-
   - In an organization there are several initiatives defined as missions which will eventually help the organization realize its vision.
3. Policy :-

   Policy statements talks about a way of doing business as defined by senior management. There may be several policies to achieve mission. E.g. security policy quality policy & human resource development policy.
4. Objectives :-
   - Strategy defines the way of achieving a particular mission. It talks about the actions required to realize the mission & way of doing things.
5. Goals :-
   - Goals define the milestones to be achieved to make the mission successful.

6. Values :-

- Values can be defined as the principles/way of doing business as perceived by the management.

❖ Quality Management System Structure :-

- Every organization has a different quality management structure depending upon its need & circumstances. Generic view of quality management is

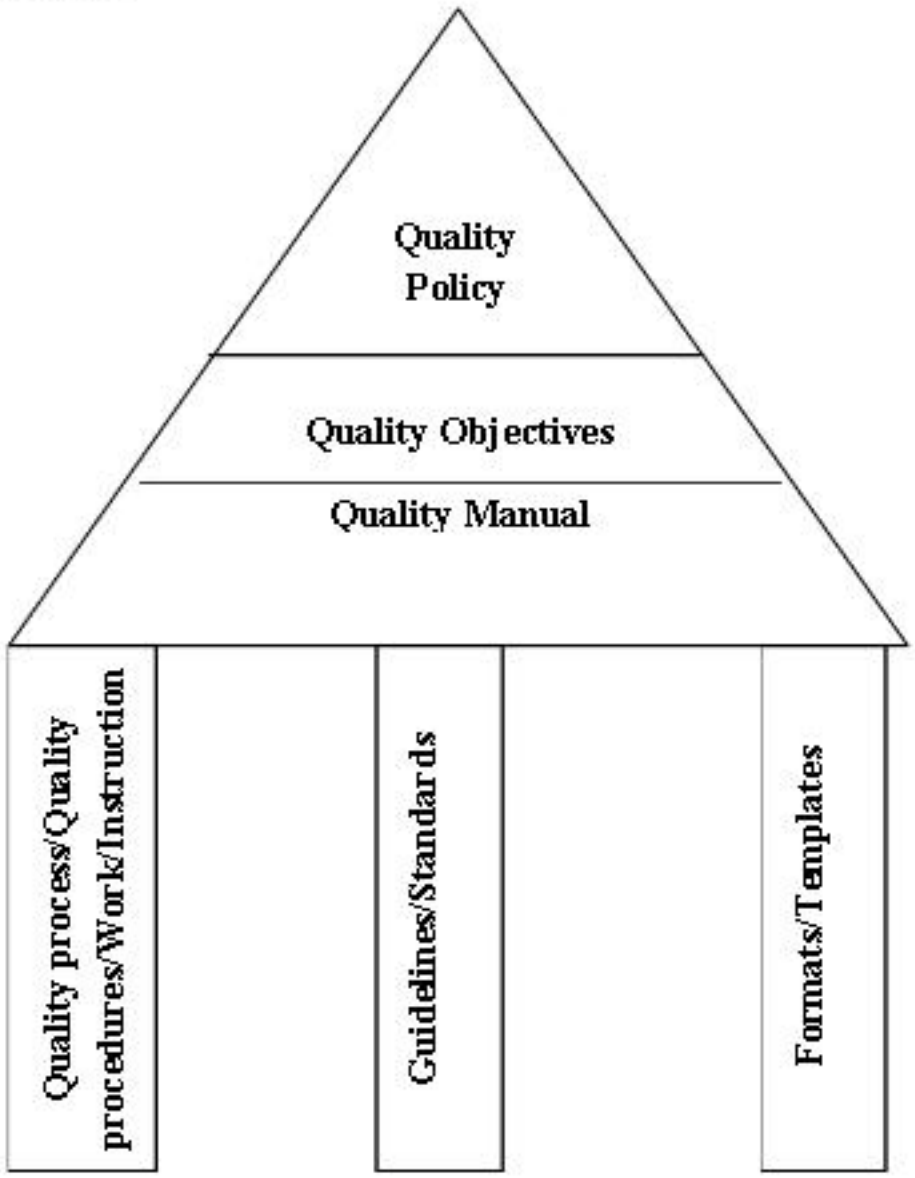


Fig. Quality Management System of a typical organization

i. 1st tier Quality Policy :-

- Quality policy sets the wish intent & direction by the management about how activities will be conducted by the organization. Since management is the strongest driving force in an organization its intents are most important. It is a basic framework on which quality temporary resets.

ii. 2nd tier Quality objectives :-

- Quality objectives are the measurements established by the management to define progress & achievements in a numerical way. An improvement in quality must be demonstrated by improvement in achievements of quality factors in numerical termed as a expected by the management.

iii. 3rd tier quality manual :-

- Quality manual also termed as policy manual is established & published by the management of the organization. It sets a framework for other process definitions & is a foundation of quality planning at organization level,

❖ Pillars of Quality Management System :-

- Top part of the quality temple is build upon the foundation of following parts of pillars.

1. Quality Process/Quality Procedures/Work instructions :

- Quality processes, quality procedures, work instructions methods are defined at an organization level by the functional area, experts and at project and functional level by the experts in those areas separately.
- Organization level processes act as an umbrella where as project & function level processes are in the preview of top level of organizations.
- Quality procedures must be in sync with tone established by quality manual at organization level.

2. Guidelines & standards :-

- Guidelines & standards are used by an organizations project team for achieving quality goals for the products & services delivered to customers.

- Many times guidelines defined by customers are termed as standards for the project as the project team takes the recommendations by customers as mandatory.

3. Formats and Templates :-
   - Common formats & templates are used for tracking a project, function & department information within an organization.
   - It creates same understanding across the board where outputs can be compared for the projects & functions.
   - Generally templates are mandatory while formats are suggestive in nature.

❖ Important Aspects of Quality Management :-

- Quality improvements is not an accident but a planned activity.
- An organization must plan for improvement under the leadership of management & with employee participation.

i. Quality planning at organization level :-
   - An organization creates quality plan at the organization level for achieving quality objectives, goals & vision & mission.

ii. Quality planning at project level.
iii. Resource Management
iv. Work Environment
v. Customer related processes
vi. Quality management system document & data control.
vii. Verification & validation.
viii. Software project management
ix. Software configuration management
x. Software matrix & measurement
xi. Software quality audits
xii. Subcontract management
xiii. Information security management
xiv. Management review

Unit II Test Planning and Management

1. Review of Fundamentals of Software Testing
2. Testing during development life cycle
3. Requirement Traceability matrix Essentials and Work bench
4. Important Features of Testing Process
    - 4.1 Misconceptions
    - 4.2 Principles
5. Salient and policy of Software testing
    - 5.1 Test Strategy
    - 5.2 Test Planning
    - 5.3 Testing Process and number of defects found
    - 5.4 Test team efficiency
    - 5.5 Mutation testing
    - 5.6 Challenges
    - 5.7 Test team approach
6. Cost aspect
7. Establishing testing policy
    - 7.1 Methods
    - 7.2 Structured approach
8. Categories of defect
9. Mistake in software
10. Developing Test Strategy and Plan
    - 10.1 Testing process
    - 10.2 Attitude towards testing
    - 10.3 Approaches
    - 10.4 Challenges
11. Raising management awareness for testing
12. Skills required by tester

1. Review of Fundamentals of Software Testing

- Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation.
- When it comes to testing software, testers all around the world use one fundamental test process that guarantees exceptional defect detection and prevention.
- This process involves continuous monitoring and controlling, which enables software testers to take necessary preventive measures whenever required. Moreover, it allows a tester to verify the compatibility of the process with the plan and ensure that it is being developed as per client's requirements. Hence, the process of testing involves the following activities:

- **Planning & Control:** The first and most important stage of the process, planning and control involves determining the objectives and goals of testing, its various scopes as well as risks, among other things. Performed throughout the life cycle, this process involves constant monitoring, which allows testers to take control of situations, where testing is not in conformance with the stated standards and plan.
- **Analysis & Design:** During this stage, testers review the test basis, such as requirements, standards, design specifications, etc. Additionally, they identify the test conditions, required infrastructure and tools, while designing tests and environment set-up.
- **Implementation & Execution:** This is the stage where the main work is done. Here, the team comes along and executes the test cases with test data. To ensure the accuracy of the outputs, the test environment is checked, while updating traceability between test basis and test cases. At the end of this stage, the output, along with various errors is reported.
- **Evaluating Exit Criteria & Reporting:** The main objective of this stage is to evaluate and assess the results and outputs from the implementation and execution stage. The team checks logs against exit criteria, which was already decided during the initial stage. After the culmination of evaluation, a summary report is created, which offers a better understanding of the test to the concerned stakeholders.
- **Test Closure:** Last, but for sure not the least important stage of the process, test closure is mainly performed at the end of the process, though there are situations where it is implemented earlier, like if the test is terminated for some reason. Other tasks performed here are:

  - Closing incident report.
  - Ensuring all deliverables is delivered.

- Documentation of all the systems and activities performed for future reference

2. Testing during development life cycle

- The Software Development Lifecycle is a systematic process for building software that ensures the quality and correctness of the software built. SDLC process aims to produce high-quality software which meets customer expectations. The software development should be complete in the pre-defined time frame and cost.
- SDLC consists of a detailed plan which explains how to plan, build, and maintain specific software. Every phase of the SDLC lifecycle has its own process and deliverables that feed into the next phase.

Here, are prime reasons why SDLC is important for developing a software system.

- It offers a basis for project planning, scheduling, and estimating
- Provides a framework for a standard set of activities and deliverables
- It is a mechanism for project tracking and control
- Increases visibility of project planning to all involved stakeholders of the development process
- Increased and enhance development speed
- Improved client relations
- Helps you to decrease project risk and project management plan overhead

- **SDLC Phases**

The entire SDLC process divided into the following stages:

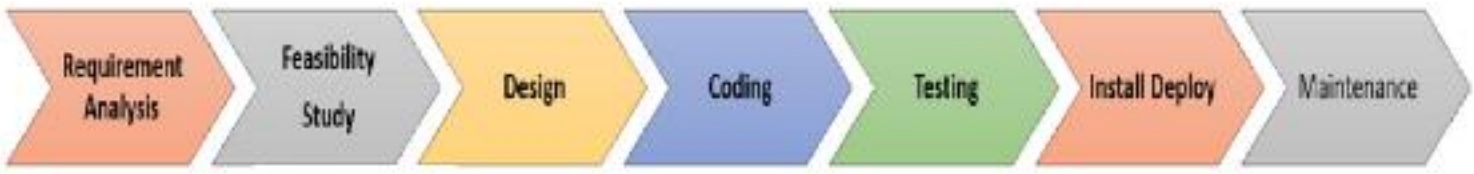


Fig - Testing during development life cycle

- Phase 1: Requirement collection and analysis
- Phase 2: Feasibility study:
- Phase 3: Design:
- Phase 4: Coding:
- Phase 5: Testing:
- Phase 6: Installation/Deployment:
- Phase 7: Maintenance:

- **Phase 1: Requirement collection and analysis:**
  The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and reorganization of the risks involved is also done at this stage.
  This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project. Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

- **Phase 2: Feasibility study:**
  Once the requirement analysis phase is completed the next step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle

- **Phase 3: Design:**
  In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.
  This design phase serves as input for the next phase of the model.

- **Phase 4: Coding:**
  Once the system design phase is over, the next phase is coding. In this phase, developers start builds the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

- **Phase 5: Testing:**
  Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

- **Phase 6: Installation/Deployment:**
  Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

- **Phase 7: Maintenance:**
  Once the system is deployed, and customers start using the developed system, following 3 activities occur
- Bug fixing - bugs are reported because of some scenarios which are not tested at all
- Upgrade - Upgrading the application to the newer versions of the Software
- Enhancement - Adding some new features into the existing software

3. Requirement Traceability matrix Essentials and Work bench

- Requirement Traceability Matrix or RTM captures all requirements proposed by the client or software development team and their traceability in a single document delivered at the conclusion of the life-cycle.

- In other words, it is a document that maps and traces user requirement with test cases. The main purpose of Requirement Traceability Matrix is to see that all test cases are covered so that no functionality should miss while doing Software testing.

- **RTM is Important:**

- The main agenda of every tester should be to understand the client's requirement and make sure that the output product should be defect-free. To achieve this goal, every QA should understand the requirement thoroughly and create positive and negative test cases.

- This would mean that the software requirements provided by the client have to be further split into different scenarios and further to test cases. Each of this case has to be executed individually.

- A question arises here on how to make sure that the requirement is tested considering all possible scenarios/cases? How to ensure that any requirement is not left out of the testing cycle?

- A simple way is to trace the requirement with its corresponding test scenarios and test cases. This merely is termed as 'Requirement Traceability Matrix.

- The traceability matrix is typically a worksheet that contains the requirements with its all possible test scenarios and cases and their current state, i.e. if they have been passed or failed. This would help the testing team to understand the level of testing activities done for the specific product.

- **Requirement Traceability Matrix**

- Let's understand the concept of Requirement Traceability Matrix through a Guru99 banking project.

- On the basis of the Business Requirement Document (BRD) and Technical Requirement Document (TRD), testers start writing test cases.
- Let suppose, the following table is our Business Requirement Document or BRD for Guru99 banking project.

Advantage of Requirement Traceability Matrix

- It confirms 100% test coverage
- It highlights any requirements missing or document inconsistencies
- It shows the overall defects or execution status with a focus on business requirements
- It helps in analyzing or estimating the impact on the QA team's work with respect to revisiting or re-working on the test cases

4. Important Features of Testing Process

- Integrated development environments are designed to maximize programmer productivity by providing tight-knit components with similar user interfaces. IDEs present a single program in which all development is done.
- This program typically provides many features for authoring, modifying, compiling, deploying and debugging software. This contrasts with software development using unrelated tools, such as VI, GCC or make.

4.1 Misconceptions

Software testing is a craft that's often misunderstood by those who are outside the field - and sometimes those who are inside it. Some of the most common misconceptions are also the most damaging to the field, leading to miscommunication, blame, and resentment.

- **Testers Break Software**

- This is a very common misconception that can quickly fall apart with some thought. Testers don't have a magical bug-insertion tool hidden between the developer's machine and the testing system, but somehow, code that worked just fine for the developer doesn't work for the tester.
- The reason isn't that the tester broke it. The reason is that the tester did something the developer didn't expect. That might be "running the software on the test system without installing all the prerequisites first" because the developer didn't realize the test system didn't have the prerequisites on it in the first place.

- It might also be actively looking for weaknesses. This is also where "but it works on my machine" comes from. On the developer's machine everything that's needed has already been installed, the entire configuration is done, and the developer knows what she's doing so her testing usually focuses on what the software should be doing.

- **Testers Don't Have Tech Skills**

- This idea is one that sets my teeth on edge. I've known some very good testers who couldn't code and didn't have particularly deep technical skills. I've also known some very good testers who could out-code every developer in their organization.

- Technical skills are not just about being able to code, and arguably include being able to reproduce an error that's difficult to trace and difficult to reproduce, being able to read or understand an error code, being able to set up the application under test with different configurations, being able to clearly explain both to a developer and to someone with no development experience just what the software is doing and why that behavior is a problem.

- **Testing Is Writing Test Cases Then Performing Them**

- You only have to scroll through job board listings to see this one in spades. Position after position will have descriptions referencing writing test cases and/or performing them. Senior testers write test cases, junior testers perform them. It's enough to make your eyes cross.

- This, like many of these misconceptions, goes back to the factory model of code development and the associated "Any Warm Body" school of testing, with their assumptions that any software can be documented sufficiently and thoroughly enough that it's possible to write exhaustively detailed test cases before the software is completed. It's a very waterfall model assumption that doesn't survive most agile practices.

4.2 Principles

Software testing is an extremely creative and intellectually challenging task. When testing follows the principles given below, the creative element of test design and execution rivals any of the preceding software development steps.

- **Testing shows the presence of bugs**

Testing an application can only reveal that one or more defects exist in the application, however, testing alone cannot prove that the application is error

free. Therefore, it is important to design test cases which find as many defects as possible.

- **Exhaustive testing is impossible**

  Unless the application under test (AUT) has a very simple logical structure and limited input, it is not possible to test all possible combinations of data and scenarios. For this reason, risk and priorities are used to concentrate on the most important aspects to test.

- **Early testing**

  The sooner we start the testing activities the better we can utilize the available time. As soon as the initial products, such the requirement or design documents are available, we can start testing.

  It is common for the testing phase to get squeezed at the end of the development lifecycle, i.e. when development has finished, so by starting testing early, we can prepare testing for each level of the development lifecycle.

  Another important point about early testing is that when defects are found earlier in the lifecycle, they are much easier and cheaper to fix. It is much cheaper to change an incorrect requirement than having to change functionality in a large system that is not working as requested or as designed.

- **Defect clustering**

  During testing, it can be observed that most of the reported defects are related to small number of modules within a system i.e. small number of modules contains most of the defects in the system. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.

- **The pesticide paradox**

  If you keep running the same set of tests over and over again, chances are no more new defects will be discovered by those test cases. Because as the system evolves, many of the previously reported defects will have been fixed and the old test cases do not apply anymore.

  Anytime a fault is fixed or a new functionality added, we need to do regression testing to make sure the new changed software has not broken any other part of the software. However, those regression test cases also need to change to reflect

the changes made in the software to be applicable and hopefully fine new defects.

- **Testing is context dependent**

  Different methodologies, techniques and types of testing are related to the type and nature of the application. For example, a software application in medical device needs more testing than games software.

  More importantly a medical device software requires risk based testing, be compliant with medical industry regulators and possibly specific test design techniques.

  By the same token, a very popular website needs to go through rigorous performance testing as well as functionality testing to make sure the performance is not affected by the load on the servers.

- **Absence of errors fallacy**

  Just because testing didn't find any defects in the software, it doesn't mean that the software is ready to be shipped. Were the executed tests really designed to catch the most defects? Or where they designed to see if the software matched the user's requirements? There are many other factors to be considered before making a decision to ship the software.

5. Salient and policy of Software testing

- There can be a very high level of management at an organizational or corporate level. This might be limited to a definition of how testing is expected to be conducted throughout the organisation.
- Wherever organizations have ISTQB certified Test Managers, the management of software testing can be done either at the programme level or at individual project level.
- For every such level of test management there is an associated document. When structured in a top-down fashion, it is called the test management documentation hierarchy.

5.1 Test Strategy

- Writing a Test Strategy effectively is a skill every tester should achieve in their career. It initiates your thought process

- Which helps to discover many missing requirements? Thinking and test planning activities help a team to define the testing scope and test coverage.
- It helps test managers to get the clear state of the project at any point. The chances of missing any test activity are very low when there is a proper test strategy in place.

Fig - Test Strategy

- Test execution without any plan rarely works. I know teams who write strategy document but never refer it back while test execution. Testing strategy plan must be discussed with the whole team so that the team will be consistent with approach and responsibilities.
- Test strategy means "How you are going to test the application?" You need to mention the exact process/strategy that you are going to follow when you will get the application for testing.
- I see many companies follow test strategy template very strictly. Even without any standard template, you can keep this test strategy document simple but still effective.

5.2 Test Planning

A TEST PLAN is a document describing software testing scope and activities. It is the basis for formally testing any software/product in a project.

- **Test plan**: A document describing the scope, approach, resources and schedule of intended test activities. It identifies amongst others test items, the features to be tested, the testing tasks, who will do each task, degree of tester independence, the test environment, the test design techniques and entry and exit criteria to be used, and the rationale for their choice, and any risks requiring contingency planning. It is a record of the test planning process.
- **Master test plan**: A test plan that typically addresses multiple test levels.

- **Phase test plan**: A test plan that typically addresses one test phase
- **Test Plan Types**: One can have the following types of test plans:
- **Master Test Plan:** A single high-level test plan for a project/product that unifies all other test plans.
- **Testing Level Specific Test Plans:** Plans for each level of testing.

  - Unit Test Plan
  - Integration Test Plan
  - System Test Plan
  - Acceptance Test Plan

- **Testing Type Specific Test Plans:** Plans for major types of testing like Performance Test Plan and Security Test Plan.

5.3 Testing Process and number of defects found

The internet defines Software Testing as the process of executing a program or application with the intent of identifying bugs. I like to define Testing as the process of validating that a piece of software meets its business and technical requirements. Testing is the primary avenue to check that the built product meets requirements adequately.

- **Test Strategy and Test Plan**

Every project needs a Test Strategy and a Test Plan. These artefacts describe the scope for testing for a project:

- The systems that need to be tested, and any specific configurations
- Features and functions that are the focus of the project
- Non-functional requirements
- Test approach—traditional, exploratory, automation, etc a mix
- Key processes to follow – for defects resolution, defects triage
- Tools—for logging defects, for test case scripting, for traceability
- Documentation to refer, and to produce as output

- **Test Design**

- Now that you have a strategy and a plan, the next step is to dive into creating a test suite. A test suite is a collection of test cases that are necessary to validate the system being built, against its original requirements.

- Test design as a process is an amalgamation of the Test Manager's experience of similar projects over the years, testers' knowledge of the system/functionality being tested and prevailing practices in testing at any given point.

- Defect may be seen as the deviation in the actual working of a software product against what was specified and expected by it. A "defect" in a software product reflects its inability or inefficiency to meet the specified requirements and criteria and subsequently prevent the software application to perform its desired and expected working.
- Since, the primary purpose behind testing is to trace out the maximum defects, present in a software product, a tester needs to be aware about the different forms of the defects, which may prevail in a software product.

5.4 Test team efficiency

- Efficiency testing tests the amount of resources required by a program to perform a specific function. In software companies, this term is used to show the effort put in to develop the application and to quantify its user-satisfaction.
- Based on this the efficiency is calculated as
  Efficiency = (ormalized units1....n)*(work items 1...n))/ Total efforts in man hours *100
- Efficiency is one of the parameters. It can never be more than a 100%. By definition, efficiency is the ratio of output to input expressed in percentage.
- It is not just one single formula but a number of calculations at each step and activity of testing. We can take an example of a team where the manager would want to deliver the product on time, within budget, without compromising with the quality.
- To achieve this, all members of a team should work efficiently to give the desired output. Test efficiency is a quality of the QA team, i.e. to carry out all testing activities in an efficient manner. It is not only about test case execution alone. This includes test planning, test cases creation, review, execution, defects tracking, comprehension and most importantly, team work.

5.5 Mutation testing

- Mutation Testing is a type of software testing where we mutate (change) certain statements in the source code and check if the test cases are able to find the errors. It is a type of White Box Testing which is mainly used for Unit Testing. The changes in mutant program are kept extremely small, so it does not affect the overall objective of the program.

- The goal of Mutation Testing is to assess the quality of the test cases which should be robust enough to fail mutant code. This method is also called as Fault-based testing strategy as it involves creating a fault in the program

How to execute Mutation Testing?

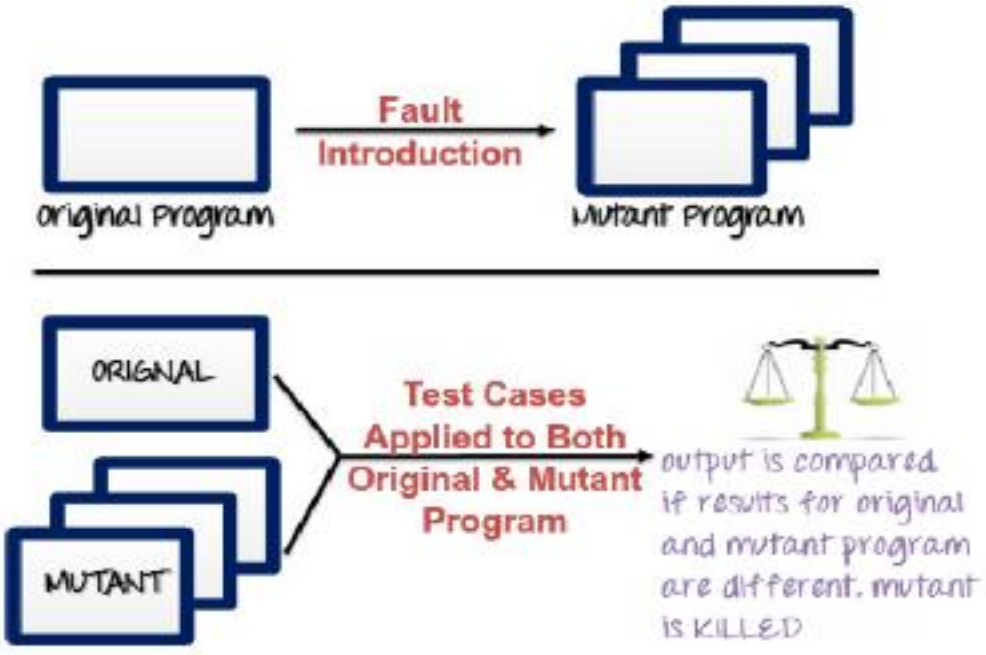


Fig - Mutation testing

Following are the steps to execute mutation testing (mutation analysis):

Step 1: Faults are introduced into the source code of the program by creating many versions called mutants. Each mutant should contain a single fault, and the goal is to cause the mutant version to fail which demonstrates the effectiveness of the test cases.

Step 2: Test cases are applied to the original program and also to the mutant program. A Test Case should be adequate, and it is tweaked to detect faults in a program.

Step 3: Compare the results of an original and mutant program.

Step 4: If the original program and mutant programs generate the different output, then that the mutant is killed by the test case. Hence the test case is good enough to detect the change between the original and the mutant program

Step 5: If the original program and mutant program generate the same output, Mutant is kept alive. In such cases, more effective test cases need to be created that kill all mutants.

Mutation Testing Benefits:

Following benefits are experienced, if mutation testing is adopted:

- It brings a whole new kind of errors to the developer's attention.

- It is the most powerful method to detect hidden defects, which might be impossible to identify using the conventional testing techniques.
- Tools such as Insure++ help us to find defects in the code using the state-of-the-art.
- Increased customer satisfaction index as the product would be less buggy.
- Debugging and maintaining the product would be easier than ever.

5.6 Challenges

- **Complete testing is impossible**
  For any application, there are many different test combinations and resulting scenarios. Trying to test all of them is an unrealistic goal. Having a clear focus on your target users and business goals will help you in prioritizing the app features that need your attention.

- **Setting the right process**
  The company process for testing should be aligned in such way that the process is efficient in capturing the bugs in time. The process should be clearly defined to avoid misunderstandings. Setting up a uniform process to get everyone on the same page becomes harder for teams that are located in different offices in different time zones.

- **Lack of proper communication**
  Right from the start of the project, it is important to make the business requirement and testing goals clear. Any change in app features or requirement must be updated to the QA team in time. This further helps in fixing defects that are detected. A good communication channel between various teams like development & QA will help sync the testing efforts with the quality goals set by the business.

- **Lack of resources**
  Skilled testers with good knowledge of the business domain can help you create more effective test scenarios and scripts. Apart from getting domain specific testing experts on board, you also need to equip them with the right testing tools. The right tools can make it easier for QA teams to execute the project and also delivering results faster.

- **Test coverage**
  In today's competitive world, enterprises need an effective testing strategy to ensure that their app works well, every where all the time. Covering different geographies and configurations for testing applications is challenging for teams that rely solely on a traditional QA approach.

Lab-based testing with limited resources can never recreate a real world environment. Getting an accurate insight in these cases thus becomes more difficult. Though 100% coverage might not be possible, you can maximize your QA efforts by relying on crowd sourced testing platforms.

5.7 Test team approach

A test approach is the test strategy implementation of a project, defines how testing would be carried out. Test approach has two techniques:

- **Proactive** - An approach in which the test design process is initiated as early as possible in order to find and fix the defects before the build is created.
- **Reactive** - An approach in which the testing is not started until after design and coding are completed.

Different Test approaches:

There are many strategies that a project can adopt depending on the context and some of them are:

- Dynamic and heuristic approaches
- Consultative approaches
- Model-based approach that uses statistical information about failure rates.
- Approaches based on risk-based testing where the entire development takes place based on the risk
- Methodical approach, which is based on failures.
- Standard-compliant approach specified by industry-specific standards.

6. Cost aspect

- Cost of Quality (COQ) is a measure that quantifies the cost of control/conformance and the cost of failure of control/non-conformance. In other words, it sums up the costs related to prevention and detection of defects and the costs due to occurrences of defects.

- Definition by ISTQB: cost of quality: The total costs incurred on quality activities and issues and often split into prevention costs, appraisal costs, internal failure costs and external failure costs.

- Definition by QAI: Money spent beyond expected production costs (labor, materials, and equipment) to ensure that the product the customer receives is a quality (defect free) product. The Cost of Quality includes prevention, appraisal, and correction or repair costs.

EXPLANATION

- Cost of Control (Also known as Cost of Conformance)
- Prevention Cost

- The cost arises from efforts to prevent defects.
- Example: Quality Assurance costs
- Appraisal Cost
- The cost arises from efforts to detect defects.
- Example: Quality Control costs
- Cost of Failure of Control (Also known as Cost of Non-Conformance)
- Internal Failure Cost
- The cost arises from defects identified internally and efforts to correct them
- Example: Cost of Rework (Fixing of internal defects and re-testing)
- External Failure Cost
- The cost arises from defects identified by the client or end-users and efforts to correct them
- Example: Cost of Rework (Fixing of external defects and re-testing) and any other costs due to external defects

7. Establishing testing policy

Test Policy is one of the key documents that exist in an organization. These are some of the broad categories of documents that exist in every testing project.

- **Test Policy** – It explains the goals that the organization wishes to achieve through testing activities.
- **Test Strategy** – This document details the general testing methods used by the organization. These test methods are independent of any project.
- **Master Test Plan** – Also called the project test plan, it explains project specific testing strategy and test implementation.
- **Level Test Plan** – Also referred as the phase test plan, this document gives details about the **testing activities** that must be performed for every test level.

7.1 Methods

SOFTWARE TESTING METHODS listed here are the major methods used while conducting various Software Testing Types during various Software Testing Levels:

Method	Summary
Black Box Testing	A software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually

	functional. Test design techniques include Equivalence partitioning, Boundary Value Analysis, Cause-Effect Graphing.
White Box Testing	A software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. Test design techniques include Control flow testing, Data flow testing, Branch testing, Path testing.
Gray Box Testing	A software testing method which is a combination of Black Box Testing method and White Box Testing method.
Agile Testing	A method of software testing that follows the principles of agile software development.
Ad Hoc Testing	A method of software testing without any planning and documentation.

7.2 Structured approach

- The structured design approach helps developers deal with the size and complexity of large scale projects. The structured approach is a process oriented approach, aiming to break a large complex project into a series of smaller, more manageable modules.
- The Structured approach is usually associated with projects that have the following characteristics :
- **Long time periods**: The time involved in a structured approach is generally measured in months, or even years.
- **Large-scale projects**: The structured approach is usually applied to large projects which involve a huge amount of detail and coordination between various parts of the business and other systems.
- **Large budgets**: Large projects will involve a large number of people and will employ them over a long time scale. The budget, consequently, will be large. New hardware and software is often required for completion of the project, elevating the cost of the project - though the amount spent upon personnel is usually the highest cost factor.

This approach, however, may be implemented as part of each of the other design approaches studied in this course. For example, an end user creating their own solution to a problem may follow the steps of the structured approach.

Similarly, the Rapid Application Development paradigm is an iterative approach. Each of the iterations may follow a simplified structured design approach.

The steps of the structured approach are as follows:

- **Defining the problem.** In this stage, the requirements for the software are gathered and analyzed, to produce a complete and unambiguous specification of what the software is required to do. The problem will need careful analysis so that the "true" issues involved are identified. For example, if your car won't start, your "problem" is "My car is not working". However, this is merely an analysis of the "symptoms". Deeper analysis may reveal that there is no petrol in the car, or the battery is flat or some other problem. Until the true cause of the problem is discovered, a useable solution cannot be developed. This stage is sometimes called Requirements Analysis.
- **Planning.** During this phase, software architecture for the implementation of the requirements is designed and specified, identifying the components within the software and the relationships between the components. Inputs, Outputs and Processes required for the new system are identified and a basic plan in the form of flowcharts or pseudo code is developed.
  A "Top Down" approach would almost certainly be used. This involves breaking a larger and complex problem into smaller more manageable modules - each module being coded separately. Thus, a structure chart is an important document developed during this phase. This stage is sometimes called Architectural and Detailed design.
- **Building.** This is the Code and Test phase, in which each component of the software is coded and tested to verify that it faithfully implements the detailed design outlined in the previous phase. Finally, once all of the modules have been coded and tested the software is integrated. Progressively larger groups of tested software components are integrated and tested until the software works as a whole. This process is sometimes called Software Integration.
- **Checking the solution.** This involves Acceptance Testing, where tests are applied and witnessed to validate that the software faithfully implements the specified requirements. Real data may be used and this is compared to outputs from a previous system. The system may also be tested under full load.
  A program may work well for a small data set, but have problems when thousands or millions of records and transactions are required of it. Beta testing may be used during application software development. Users are also involved in this process to ensure that their expectations of the system are fulfilled. In addition to the feedback that the development team receives, users also benefit by being able to use the system. In addition to checking that the program can process the data correctly, the user interface is evaluated for ease of use and intuitive design.

- **Modifying the solution.** During the testing phase, problems or possible improvements (from beta testing) may be found. If problems are found, modifications may be made to solve the problems and another round of testing will occur.

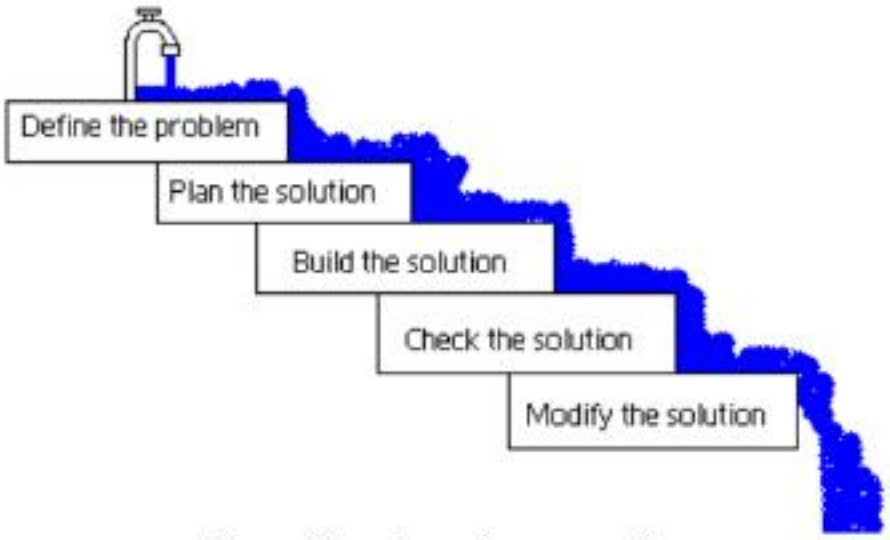


Fig - Structured approach

8. Categories of defect

Types of Defects:

- **Severity Basis:**

Severity defines the degree of impact. Thus, defect's severity reflects the degree or intensity of a particular defect, to impact a software product or its working, adversely. Based on the severity metric, a defect may be further categorized into following:

- **Critical:** The defects termed as 'critical', needs immediate attention and treatment. A critical defect directly affects the critical and essential functionalities, which may affect a software product or its functionality on a large scale, such as failure of a feature/functionality or the whole system, system crash-down, etc.
- **Major:** Defects, which are responsible for affecting the core and major functionalities of a software product. Although, these defects does not results into complete failure of a system, but may bring several major functions of software, to rest.
- **Minor:** These defects produce minor impact, and do not have any significant influence on a software product. The results of these defects may be seen in the product's working, however, it does not stops users to execute it task, which may be carried out, using some other alternative.
- **Trivial:** These types of defects have no impact on the working of a product, and sometimes, it is ignored and skipped, such as spelling or grammatical mistake.

- **Probability Basis:**

  One more angle to see a defect in a software application is on the basis of its probability to occur and getting encountered by the user. Based on the likelihood or the possibility of a defect in a software product in terms of percentage, it may be classified into following forms:

- **High:** Signifies the high probability of getting traced out by almost all the users of the application.
- **Medium:** Half of the users are able to trace out the defects presence in a software product.
- **Low:** Generally, not detected any user or may be detected only few numbers of user.

- **Priority Basis:**

  Defects could also be seen from the business perspective, as which defects needs to be corrected first and which may be fixed at a later stage, based on the current need and demand of the business. Similar to probability basis, priority may also be categorized into following forms:

- **High:** High priority defines the foremost need of a business, to correct a defect, as soon as possible, in the same build.
- **Medium:** Medium priority defects come next to that of high priority, and may be addressed in any next version or a release of a product.
- **Low:** These types of defects, does not needs to be individually corrected. It may or may not be considered or corrected, along with any other defect, which needs to be fixed.

Apart from the above given basis, a software defect may also be considered on the following grounds:

- **Extra Defects:** Defects due to the implementation of the requirements other than the client's specified requirements and specifications.
- **Missing Defects:** These defects, generally arises to due to the non-fulfilment of any of the requirements and specifications, as specified by the client or the user.
- **Wrong Defects:** This defect shows the discrepancies between the specified requirements and what was considered and implemented. A requirement being misunderstood and misinterpreted by the project team, and subsequently incorporated by the development team, results into wrong defects.

9. Mistake in software

- **Hands-on mode**
  A person who has never seen something starts coding without even planning the solution. Just turns on hands-on mode without analysis and detailed planning.

- **Lack of knowledge about the business**
  Without the knowledge about the business itself, just with a punctual requirement and a single vision about the functionality and with no idea about problems that could occur in the future.

- **Unknown infrastructure**
  Many developers don't know or don't care to know about the infrastructure. For example, how does Web host or the memory Recycle Bin work, among other subjects. If the developers knew the basics of an environment; the solutions and programming would be different.

- **Inability to speak user language**
  Most are extremely technical and are unable to relate to the user language. It means that they can understand programming and solving coding issues, but are unable to make an analysis on the user's requirements.

- **Guilt free**
  He never takes the blame. The guilt is always on the person that didn't allow him to test.

- **Estimates hours only for development**.
  The hours estimated are based on the amount of times a fly swings its wings divided by the turtle steps in a minute. Mostly the estimated time has no real basis in fact and they forget that they need hours for an analysis and test beyond the development hours.

- **Fear of changes**
  When a new technology comes out, it's like Oh my god! It's going to take a while to learn, it's hard, I won't be able to deliver.

- **Think that others are better than yourself**
  Developers are likely to think that other developers are better than he is. Nevertheless, he tends to forget that better is not always only about knowledge but the role package, commitment, responsibility, punctuality etc. Relax.

- **Wants to create for the web with desktop concepts**
  For example, the developer who thinks that only someone who knows how to program in C# and has worked with Windows Forms is going to be able to develop a Web Application in MVC. He or she starts programming using desktop concepts and gets side-tracked at the client/Server concept.
- **Code jealousy**
  "What a beautiful code! Wonderful! Can I see?" "Don't touch this! Don't change anything in that part..." When someone is going to change something in the code that someone else created it is always war and the manager must interfere.

- **Wants to create mobile with Web concepts.**
  Another interesting point is when a programmer starts to develop mobile apps with the looks or the concepts of the Web. It's easy to realize that what has been done looks like a responsive Webpage.

- **Under-usage of design patterns.**
  Design patterns were created to be used. They were exhaustively tested before becoming a pattern. Thus, let's not reinvent the wheel. Use design patterns.

- **Doesn't follow any software architecture**
  The developers are the architects of their own ego and ignore the basic principles of separating the responsibilities.

- **View state is a madhouse**
  I have seen lots of developers use and abuse the famous Viewstate of the Web forms, filling it up with datasets and almost a full database.

- Try and find out an error without reading the documentation

- A person who has never seen a developer tries without even looking at the documentation.

- Coding without the comments

- /* NO COMMENTS */

- Copying and pasting the code; unsure of what it does.

- Many programmers say "Copy and paste that it should work!"

- You must ask "What's that for?"

- Answer: "Don't know, but it works!"
- Doesn't follow the best practices
- "Is it working? Yes! Good enough."
- Doesn't follow a pattern or logic
- Do something here in one way and the same there in another completely different way.
- Doesn't know exception error meanings
- Exceptions are hieroglyphs.
- Doesn't handle possible null reference exceptions.

10. Developing Test Strategy and Plan

- A test strategy is an outline that describes the testing approach of the software development cycle. It is created to inform project managers, testers, and developers about some key issues of the testing process.
- This includes the testing objective, methods of testing new functions, total time and resources required for the project, and the testing environment.
- Test strategies describe how the product risks of the stakeholders are mitigated at the test-level, which types of testing are to be performed, and which entry and exit criteria apply. They are created based on development design documents.
- System design documents are primarily used, and occasionally conceptual design documents may be referred to. Design documents describe the functionality of the software to be enabled in the upcoming release.

Test strategy contains:

- **Scope and objective:** The objective of the business and how much testing scope is there is defined under test strategy.
- **Business Issues:** How much is the budget of the project, how much time is required for testing, how much resources are needed etc. are the part of business issues which needs to be considered before the actual testing starts.

- **Testing approach:** What type of testing is needed and whether the testing is only manual or automation or both are some of the crucial points which define the testing approach.
- **Test deliverables:** What are the documents required from the testing team, how they would keep the record of the testing cycles etc. will be included here.
- **Defect tracking approach**: Which tool will be used for tracking the defects and how will the testing team communicate with the development team and how the flow would go for defects are decided at this point in test strategy.
- **Training**: If there is some complex or new tool is introduced in the business then it is helpful if the team members are given proper training.

Test plan contains:

- **Test plan ID:** This is a unique ID which defines the test plan. It can be a number or name or mix of both, as per the convenience.
- **Test environment:** This section defines what kind of environment is needed for the testing to carry out. For e.g. in device testing, usually a virtual set up is made to test emergency calling.
- **Features to be tested/Not tested:** This will have all the details about the features which tester needs to test and what are the feature which are not tested (may be because it is not yet implemented or not tested for that particular release).
- **Entry/Exit criteria:** These are the terms which define when to start or stop the testing. Standards will be defined under test strategy and followed by testers in test plan.
- **Status:** Whether a test case is passed or failed or not tested, all these test results are included in test plan with a proper reason.
- **Types of testing:** The types of testing required such as regression, functional, non-functional, stress etc. are defined and then executed by the respective tester.

10.1 Testing process

- **Test Strategy and Test Plan**
  Every project needs a Test Strategy and a Test Plan. These artefacts describe the scope for testing for a project:

  The systems that need to be tested and any specific configurations
  Features and functions that are the focus of the project
  Non-functional requirements

Test approach—traditional, exploratory, automation, etc.—or a mix
Key processes to follow – for defects resolution, defects triage
Tools—for logging defects, for test case scripting, for traceability
Documentation to refer, and to produce as output
Test environment requirements and setup

- **Test Design**

Now that you have a strategy and a plan, the next step is to dive into creating a test suite. A test suite is a collection of test cases that are necessary to validate the system being built, against its original requirements.

Test design as a process is an amalgamation of the Test Manager's experience of similar projects over the years, testers' knowledge of the system/functionality being tested and prevailing practices in testing at any given point.

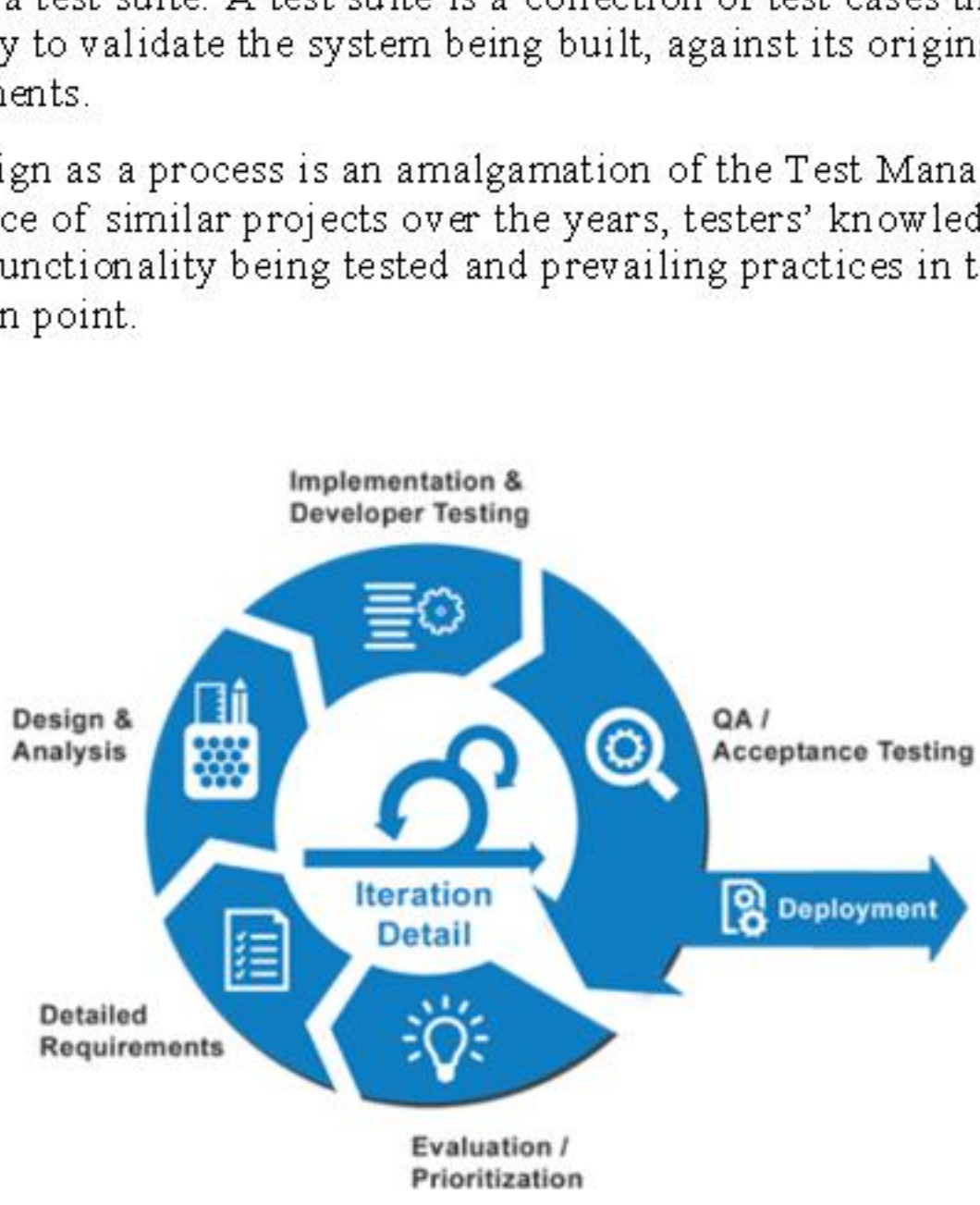


Fig - Agile Testing Life Cycle

- **Test Execution**

  You can execute tests in many different ways—as single, waterfall SIT (System Integration Test) and UAT (User Acceptance Test) phases; as part of Agile sprints; supplemented with exploratory tests; or with test-driven development. Ultimately, you need to do adequate amount of software testing to ensure your system is (relatively) bug-free.
  Let's set methodology aside for a second, and focus on how you can clock adequate testing. Let's go back to the example of building a mobile app that can be supported across operating systems, OS versions, and devices.

- **Test Closure**

  Right so you have done the planning necessary, executed tests and now want to green-light your product for release. You need to consider the exit criteria for signalling completion of the test cycle and readiness for a release. Let's look at the components of exit criteria in general:

- 100% requirements coverage: all business and technical requirements have to be covered by testing.
- Minimum % pass rate: targeting 90% of all test cases to be passed is best practice.
- All critical defects to be fixed: self-explanatory. They are critical for a reason.

10.2 Attitude towards testing

- **Be Sceptical**
  Don't believe that the build given by the developers is a bug-free or quality outcome. Question everything. Accept the build only if you test and find it defect free. Don't believe anyone whatever is the designation they hold, just apply your knowledge and try to find the errors. You need to follow this until the last phase of the testing cycle.

- **Don't Compromise on Quality**
  Don't compromise after certain testing stages. There is no limit for testing until you produce a quality product. Quality is the word made by software testers to achieve more effective testing. Compromising at any level leads to a defective product, so don't do that at any point.

- **Ensure End User Satisfaction**
  Always think what can make an end user happy. How they can use the product with ease. Don't stop by testing the standard requirements alone. The end user can be happy only when you provide an error-free product.

- **Think from the Users Perspective**
  Every product is developed for the customers. Customers may or may not be technical persons. If you don't consider the scenarios from their perspective you will miss many important bugs. So put yourself in their shoes. Know your end users first. Their age, education even the location can matter most while using the product.

  Make sure to prepare your test scenarios and test the data accordingly. After all, the project is said to be successful only if the end user is able to use the application successfully.

- **Prioritize Tests**
  First, identify the important tests and then prioritize the execution based on test importance. Never ever execute test cases sequentially without deciding the priority. This will ensure that all your important test cases get executed early and you won't cut down on these at the last stage of release cycle due to time pressure.

  Also, consider the defect history while estimating test efforts. In most cases, defect count at the beginning is more and goes on reducing at the end of the test cycle.

- **Never Promise 100% Coverage**
  Saying 100% coverage on paper is easy but practically it is impossible. So never promise to anyone including your clients about total test coverage. In business there is a philosophy – "Under promise and over deliver." So don't the goal for 100% coverage but focus on the quality of your tests.

- **Be Open to Suggestions**
  Listen to everyone even though you are an authority on the project having in-depth project knowledge. There is always scope for improvements and getting suggestions from the fellow software testers are a good idea. Everyone's feedback to improve the quality of the project would certainly help you to release a bug-free software.

- **Start Early**
  Don't wait until you get your first build for testing. Start analyzing the requirements, preparing test cases, test plan and test strategy documents in the early design phase. Starting early to test helps to visualize the complete project scope and hence planning can be done accordingly.

Most of the defects can be detected in early design and analysis phase saving huge time and money. Early requirement analysis will also help you to question the design decisions.

- **Identify and Manage Risks**
  Risks are associated with every project. Risk management is a three-step process. Risk identification, analysis, and mitigation. Incorporate risk driven testing process. Priorities of software testing are based on risk evaluation.

- **Do Market Research**
  Don't think that your responsibility is just to validate software against the set of requirements. Be proactive, do your product market research and provide suggestions to improve it. This research will also help you to understand your product and its market.

- **Develop Good Analyzing Skill**
  This is a must for requirement analysis but even further this could be helpful for understanding customer feedback while defining the test strategy. Question everything around you. This will trigger the analysis process and it will help you to resolve many complex problems.

- **Focus on Negative Side as Well**
  Testers should have the test to break attitude. Concentrating only on the positive side will almost certainly create many security issues in your application. You should be the hacker of your project to keep other hackers away from it. Negative testing is equally important. So cover a good chunk of your test cases based on the negative scenarios.

- **Be a Good Judge of Your Product**
  A Judge usually thinks if something is right or wrong. A judge will listen to both the sides. Same is applicable for testing as well. As a software tester if you think something as right, try to prove it why it is not wrong and later accept it. You must have a valid reason for all your decisions.

- **Learn to Negotiate**
  Testers must negotiate with everyone in all the stages of a project lifecycle. Especially negotiation with the developers is more important. Developers can do anything to prove that their code is correct and the defect logged by the testers is not valid. It requires great skills to convince the developers about the defect and get it resolved.

Though some software testers think that this is not our task, explaining the true impact of any issue is very helpful for the developers to quickly understand the overall scenario and its implications. This requires years of practice but once you learn to negotiate you will gain more respect.

- **Stop the Blame Game**
  It's common to blame others for any defects which are not caught in testing. This is even more common when the tester's responsibilities are not defined concretely. But in any situation never blame anyone. If an error occurs, first try to resolve it rather than finding someone to blame.

- **Finally, Be a Good Observer**
  Observe things happening around you. Keep a track of all the major and minor things on your project. Observe the way of developing the code, types of testing and its objective. Observe and understand the test progress and make necessary changes if it is off the track in terms of schedule or testing activities.

10.3 Approaches

A test approach is the test strategy implementation of a project, defines how testing would be carried out. Test approach has two techniques:

- **Proactive** - An approach in which the test design process is initiated as early as possible in order to find and fix the defects before the build is created.
- **Reactive** - An approach in which the testing is not started until after design and coding are completed.
  Different Test approaches:

There are many strategies that a project can adopt depending on the context and some of them are:

- Dynamic and heuristic approaches
- Consultative approaches
- Model-based approach that uses statistical information about failure rates.
- Approaches based on risk-based testing where the entire development takes place based on the risk
- Methodical approach, which is based on failures.
- Standard-compliant approach specified by industry-specific standards.

10.4 Challenges

Software Testing has a lot of challenges both in a manual as well as in automation. Generally in manual testing scenario developers through the build to test team assuming the responsible test team or tester will pick the build and

will come to ask what the build is about? This is the case in organizations not following so-called 'processes'. Tester is the middleman between developing a team and the customers, handling the pressure from both the sides.

- **Testing the complete application:**
  Is it possible? I think impossible. There are millions of test combinations. It's not possible to test each and every combination both in the manual as well as in automation testing. If you try all these combinations you will never ship the product.
- **Misunderstanding of company processes:**
  Sometimes you just don't pay proper attention what the company-defined processes are and these are for what purposes. There are some myths in testers that they should only go with company processes even these processes are not applicable for their current testing scenario. This results in incomplete and inappropriate application testing.
- **Relationship with developers:**
  Big challenge. Requires very skilled tester to handle this relation positively and even by completing the work in testers way. There are simply hundreds of excuses developers or testers can make when they do not agree with some points. For this tester also requires good communication, troubleshooting and analyzing skill.
- **Regression testing:**
  When a project goes on expanding the regression testing work simply becomes uncontrolled. Pressure to handle the current functionality changes, previous working functionality checks, and bug tracking.

11. Raising management awareness for testing

- **Test strategy**

  A test strategy is a high-level document that derives from the Business Requirements Specification document. Usually, a project manager or a business analyst creates a test strategy to define software testing approaches used to achieve testing objectives. A test strategy is driven by the project's business requirements, which is why it meshes with a project manager's responsibilities.

- **Test Plan**

  A test plan is a document that describes what to test, when to test, how to test, and who will do the tests. It also describes the testing scope and activities. The test plan includes the objectives of the tests to be run and helps control the risks. It's a good practice to have a test plan written by an experienced person like a QA lead or manager.

- **Test cases**

  Preparation of effective test cases is an integral part of software testing improvements. According to the definition, given by ISTQB (International Software Testing Qualifications Board, the worldwide leader in the certification of competences in software testing) "a test case is a document which consists of a set of conditions or actions which are performed on the software application in order to verify the expected functionality of the feature". It's one of the key instruments used by testers.

- **Test-driven development**

  Test-driven development (TDD) is a software development process in which tests are written before any implementation of the code. TDD has a test-first approach based on repetition of a very short development cycle. According to it, each new feature begins with writing a test. The developer writes an automated test case before he/she writes enough production code to fulfil that test. This test case will initially fail. The next step will be to write the code focusing on functionality to get that test passed. After these steps are completed, a developer reactors the code to pass all the tests.

- **Optimize the use of automated tests**

  If you really want to improve the quality of your software, then automated testing is definitely worth taking into consideration. According to the World Quality Report 2017–2018 by Capgemini, Sogeti, and Micro Focus, two of three key trends are increasing test automation and widespread adoption of the agile methodologies. It's really a wise recommendation to deploy automated testing throughout the QA process. Automated testing means using automation tools to run the tests.

12. Skills required by tester

- Ability to understand the functionality of the application under test (quick understanding because sometimes people don't have the time to teach you all about the application and you need to figure all that on your own)
- Out of the box thinking to understand how can possibly the application broken and tested
- Working knowledge of Database, you would need to validate data update/delete etc.
- Working knowledge of API calls to pin point where actually the error is happening.
- Ability to write a bug properly (the actual issue and the steps to reproduce the issue)
- SDLC and Testing Methodologies (Types of Testing, Test Plan, Creating Test cases, Executing Test cases)

- Domain Knowledge of Product / Application / Project which you are going to test
- Technical knowledge of Technology which Product / Application / Project using.(Including Backend)
- Bug Tracking Tools - Complete bug life cycle management
- Knowledge of Automation Tool if your Product / Application / Project requires automation testing (E.g. Selenium, QTP, Test complete)

References:

1. Naresh Chauhan, "Software Testing Principles and Practices ", OXFORD, ISBN-10: 0198061846. ISBN-13: 9780198061847.

2. Stephen Kan, "Metrics and Models in Software Quality Engineering", Pearson, ISBN-10: 0133988082; ISBN-13: 978-0133988086.

3. Srinivasan Desikan, Gopalswamy Ramesh, "Software Testing Principles and Practices", Pearson, ISBN-10: 817758121X.

Unit III Software Test Automation

1. What is Test Automation
2. Terms used in automation
3. Skills needed for automation
4. What to automate
5. Scope of automation
6. Design and Architecture of automation
7. Generic requirement for Test Tool
8. Process Model for Automation
9. Automation for XP/Agile model
10. Challenges in Automation
11. Data-driven Testing
12. Automation Tools like JUnit and Jmeter

1. What is Test Automation

- Software Test automation makes use of specialized tools to control the execution of tests and compares the actual results against the expected result. Usually, regression tests, which are repetitive actions, are automated.

- Testing Tools not only helps us to perform regression tests but also helps us to automate data set up generation, product installation, GUI interaction, defect logging, etc. Automation tools are used for both Functional and Non-Functional testing.

  There are many approaches to test automation; however below are the general approaches used widely:

- Graphical user interface testing. A testing framework that generates user interface events such as keystrokes and mouse clicks, and observes the changes that result in the user interface, to validate that the observable behavior of the program is correct.

- API driven testing. A testing framework that uses a programming interface to the application to validate the behaviour under test. Typically API driven testing bypasses application user interface altogether. It can also be testing public (usually) interfaces to classes, modules or libraries are tested with a variety of input arguments to validate that the results that are returned are correct.

- One way to generate test cases automatically is model-based testing through use of a model of the system for test case generation, but research continues into a variety of alternative methodologies for doing so.

- In some cases, the model-based approach enables non-technical users to create automated business test cases in plain English so that no programming of any kind is needed in order to configure them for multiple operating systems, browsers, and smart devices.

- What to automate, when to automate, or even whether one really needs automation are crucial decisions which the testing (or development) team must make. A multi-vocal literature review of 52 practitioner and 26 academic sources found that five main factors to consider in test automation decision are:

- System under Test (SUT)
- The types and numbers of tests
- Test-tool
- Human and organizational topics
- Cross-cutting factors.

- The most frequent individual factors identified in the study were: need for regression testing, economic factors, and maturity of SUT.
- A growing trend in software development is the use of unit testing frameworks such as the xUnit frameworks (for example, JUnit and NUnit) that allow the execution of unit tests to determine whether various sections of the code are acting as expected under various circumstances. Test cases describe tests that need to be run on the program to verify that the program runs as expected.
- Test automation, mostly using unit testing, is a key feature of extreme programming and agile software development, where it is known as test-driven development (TDD) or test-first development. Unit tests can be written to define the functionality before the code is written. However, these unit tests evolve and are extended as coding progresses, issues are discovered and the code is subjected to refactoring.

2. Terms used in automation

- This glossary of testing terms contains a large number of frequently used terms, and is an excerpt of the book Testing and Quality Assurance of IT Systems, by ReQtest founder Ulf Eriksson.

- The glossary is meant to help you get familiar with words and phrases commonly used in testing and requirements work. You can use the glossary as the basis for introducing these terms into your organization or standardizing their use.

- **Acceptance testing**
  the final test level. Conducted by users with the purpose to accept or reject the system before release.
- **Actual result**
  the system status or behaviour after you conduct a test. An anomaly or deviation is when your actual results differ from the expected results.
- **Ad hoc testing**
  Testing carried out informally without test cases or other written test instructions.
- **Agile development**
  a development method that emphasizes working in short iterations. Automated testing is often used. Requirements and solutions evolve through close collaboration between team members that represent both the client and supplier. (Also read: Agile Software Development- 5 Trends to Watch Out For In 2019)
- **Alpha testing**
  Operational testing conducted by potential users, customers, or an independent test team at the vendor's site. Alpha testers should not be from the group involved in the development of the system, in order to maintain their

objectivity. Alpha testing is sometimes used as acceptance testing by the vendor.

- **Anomaly**
  any condition that deviates from expectations based on requirements specifications, design documents, standards etc. A good way to find anomalies is by testing the software.
- **Big-bang integration**
  an integration testing strategy in which every component of a system is assembled and tested together; contrast with other integration testing strategies in which system components are integrated one at a time.
- **Black box testing**
  Testing in which the test object is seen as a "black box" and the tester has no knowledge of its internal structure. The opposite of white box testing.
- **Bottom-up integration**
  an integration testing strategy in which you start integrating components from the lowest level of the system architecture. Compare to big-bang integration and top-down integration.

3. Skills needed for automation

The skills required depends on what generation of automation the company is in.

- **Capture / Playback and Test Harness Tools:**
- One of the most boring and time-consuming activity during testing life cycle is to rerun
- Manual tests number of times. Here, capture/playback tools are of great help to the testers. These tools do this by recording and replaying the test input scripts. As a result, tests can be replayed without attendant for long hours specially during regression testing.
- Also these recorded test scripts can be edited as per need i.e., whenever changes are made to the software. These tools can even capture human operations e.g., mouse activity, keystrokes etc.
- A capture / playback tool can be either intrusive or non-intrusive. Intrusive capture / playback tools are also called native tools as they along with software-under-test (SUT), reside on the same machine.

- **Data-driven Tools:**
- This method help in developing test scripts that generates the set of input conditions and corresponding expected output. The approach takes as much time and effort as the product. However, changes to application do not require the automated test cases to be changed as long as the input conditions and

expected output are still valid. This generation of automation focuses on input and output conditions using the black box testing approach.

- **Action-driven Tools:**
- This technique enables a layman to create automated tests. There are no input and expected output conditions required for running the tests. All actions that appear on the application are automatically tested, based on a generic set of controls defined for automation.

4. What to automate?

- As we know, automation is the driving force behind continuous delivery and agile practices. It's helped change our digital landscape and is shaping businesses into the Modern Software Factories the application era requires. Automation's benefits can be applied to just about any department within an enterprise; from HR to Accounts, Dev to Ops – even the mailroom.
- However, certain processes are more suited to automation than others, and what to automate depends on certain factors. There are telltale signs to watch out for that indicate a process is primed for automation.

- **Medium and high volume**

  Workflows vary dramatically in size. They can consist of simple processes which are composed of few steps, to processes requiring dozens, if not hundreds of items.

  When we think about workflows with minimal steps or items, we should ask 'does it make business sense to automate this process?' Conversely, processes with medium and high volume items are clearly business-pivotal processes, primed for automation.

- **Manual completion requires three or more users**

  Generally, if a repeatable task involves three or more people, the likelihood is that it would be more efficient if it was automated. There is less chance of a communication breakdown, making it more secure and more accurate. Furthermore, by automating such routines, you'll free up the man hours of at least three individuals.

5. Scope of automation

- The purpose of test automation is to provide automated support for testing procedures. Integration, unit, and system testing work well with automated testing.

- A QA team keen on test design and execution develops the automated testing framework. The extent to which automation is used in the testing process defines the scope of automated testing.
- Among system tests, automation best facilitates security, configuration, and load testing. Regression testing also works well with automation during development and at the initial time of release.

Fig – Architecture of test automation

- Test planning, design, builds, execution, analysis, verification, validation, and reporting are phases which include defect tracking, software and system configuration management, and software metrics, all working together and ultimately adhering to each other as a unified process of effective management strategies.
- The appropriate automation works in these testing situations to varying degrees, depending upon how the testing process is managed.

The Team

The QA testing team is the manpower nucleus of a testing project. Strategies that support and enhance communication among team members, allowing the

interconnection of talents and the integration of perspectives, boost the effectiveness of the test procedure.

Automated test management oversees the configurations of test modules, calculates the software testing metrics, and processes test deliverables, allowing teams a more concentrated focus on risk assessment, project design, and planning.

6. Design and Architecture of automation

- Test automation framework architecture efforts are often complete failures. It's true. I've worked with many companies who have given up on creating good test automation framework architecture, because after investing a large amount of time and money and resources in doing it the wrong way, they have incorrectly assumed the entire effort is not cost effective.

- Once you understand those two very important goals and why they are important, it is easier to think about how you should design test automation framework architecture.
- I've created quite a few test automation framework architectures, so I'll give you a basic design I use for most of them.

Take a look at this diagram:

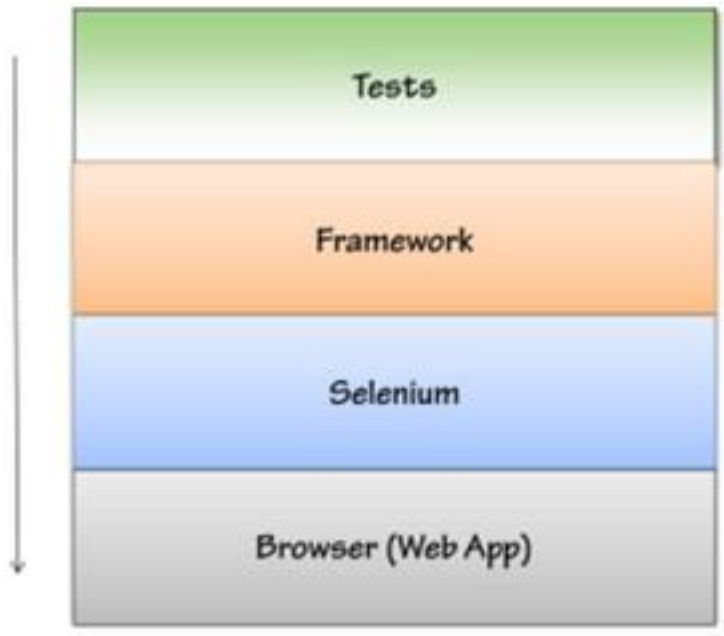


Fig - Architecture of automation

- Here you can see that there are four layers to my test automation framework architecture. First, we have the browser layer or the web app itself. This just represents your actual application.
- Next, we have the Selenium or web driver layer. This layer represents your browser automation tool. Selenium is basically just a framework for automating

a browser. It doesn't know anything about testing; it is an API that lets you interact with the browser programmatically.

- After that, we have the framework layer. This is the actual framework you create which uses Selenium, or whatever web driver you want, to actually automate your application. The framework is acting as an abstraction between your application and the tests. The framework knows about the UI of your application and how to interact with it using Selenium. It is your job to create this layer.
- Finally, we have the actual tests. The tests use the framework to manipulate your application and check the state of the application. These tests should be simple to understand and should not have to know anything about the actual implementation of the UI of your application. These tests should rely on the framework to give them the ability to do whatever they need to do in your application.

7. Generic requirement for Test Tool

Tools from a software testing context can be defined as a product that supports one or more test activities right from planning, requirements, creating a build, test execution, defect logging and test analysis.

Classification of Tools

Tools can be classified based on several parameters. They include:

- The purpose of the tool
- The Activities that are supported within the tool
- The Type/level of testing it supports
- The Kind of licensing (open source, freeware, commercial)
- The technology used

Types of Tools:

Sr.No.	Tool Type	Used for	Used by
1.	Test Management Tool	Test Managing, scheduling, defect logging, tracking and analysis.	testers
2.	Configuration management tool	For Implementation, execution, tracking changes	All Team members
3.	Static Analysis Tools	Static Testing	Developers
4.	Test data Preparation Tools	Analysis and Design, Test data generation	Testers
5.	Test Execution Tools	Implementation, Execution	Testers
6.	Test Comparators	Comparing expected and actual results	All Team members
7.	Coverage measurement tools	Provides structural coverage	Developers
8.	Performance Testing tools	Monitoring the performance, response time	Testers
9.	Project planning and Tracking Tools	For Planning	Project Managers

8. Process Model for Automation

- One of the key reasons that business process automation initiatives fail is that the business will model several hundred as-is and to-be processes, and then "throw them over the fence" to a team of process automation specialists, who ask, "What do I do with these?"

- The process modelled by or in conjunction with the business subject matter experts is a business view of the process. So even though a BPMN model can be executed, the business, or logical, view of the process, isn't necessarily the best executable model.

- Process Modelling for Automation is about taking the business view of the process and optimising it for execution in workflows and automation platforms.

Following steps are followed in an Automation Process.

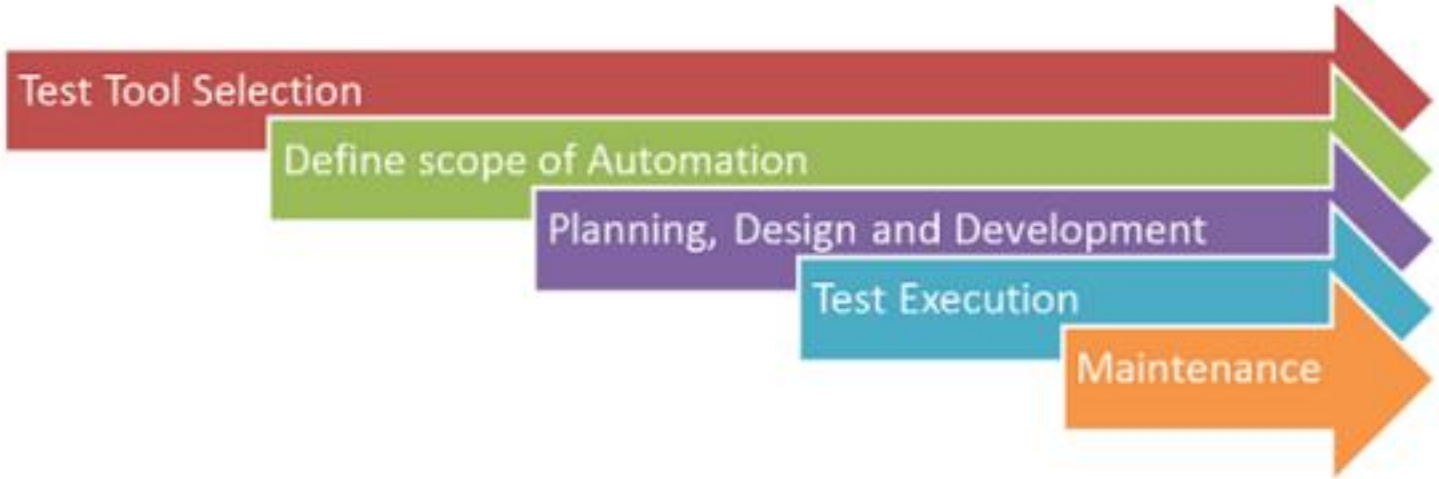


Fig - Process Model for Automation

- **Test tool selection**

  Test Tool selection largely depends on the technology the Application Under Test is built on. For instance, QTP does not support Informatics. So QTP cannot be used for testing Informatics applications.

- **Planning, Design, and Development**

  During this phase, you create an Automation strategy & plan, which contains the following details-

- Automation tools selected
- Framework design and its features
- In-Scope and Out-of-scope items of automation

- Automation tested preparation
- Schedule and Timeline of scripting and execution
- Deliverables of Automation Testing
- **Test Execution**

  Automation Scripts are executed during this phase. The scripts need input test data before there are set to run. Once executed they provide detailed test reports.

  Execution can be performed using the automation tool directly or through the Test Management tool which will invoke the automation tool.

9. Automation for XP/Agile model

- Extreme Programming (XP) is a software engineering methodology, the most prominent of several development methodologies. Like other agile methodologies, Extreme Programming differs from traditional methodologies primarily in placing a higher value on adaptability than on predictability.
- Proponents of XP regard ongoing changes to requirements as an often natural and often inescapable aspect of software development projects; they believe that being able to adapt to changing requirements at any point during the project life is a more realistic and better approach than attempting to define all requirements at the beginning of a project and then expending effort to control changes to the requirements.
- Quality assurance is believed to be a most important aspect of XP. QA can be performed by a separate group, or it can be integrated into the development team itself. In either case, XP promotes the idea that the developers have a high degree of contact with the QA group; this is very much in line with the XP philosophy of closer communications between the stakeholders in a project.
- A typical model of planning & feedback loop for XP as described in Wikipedia is shown below.

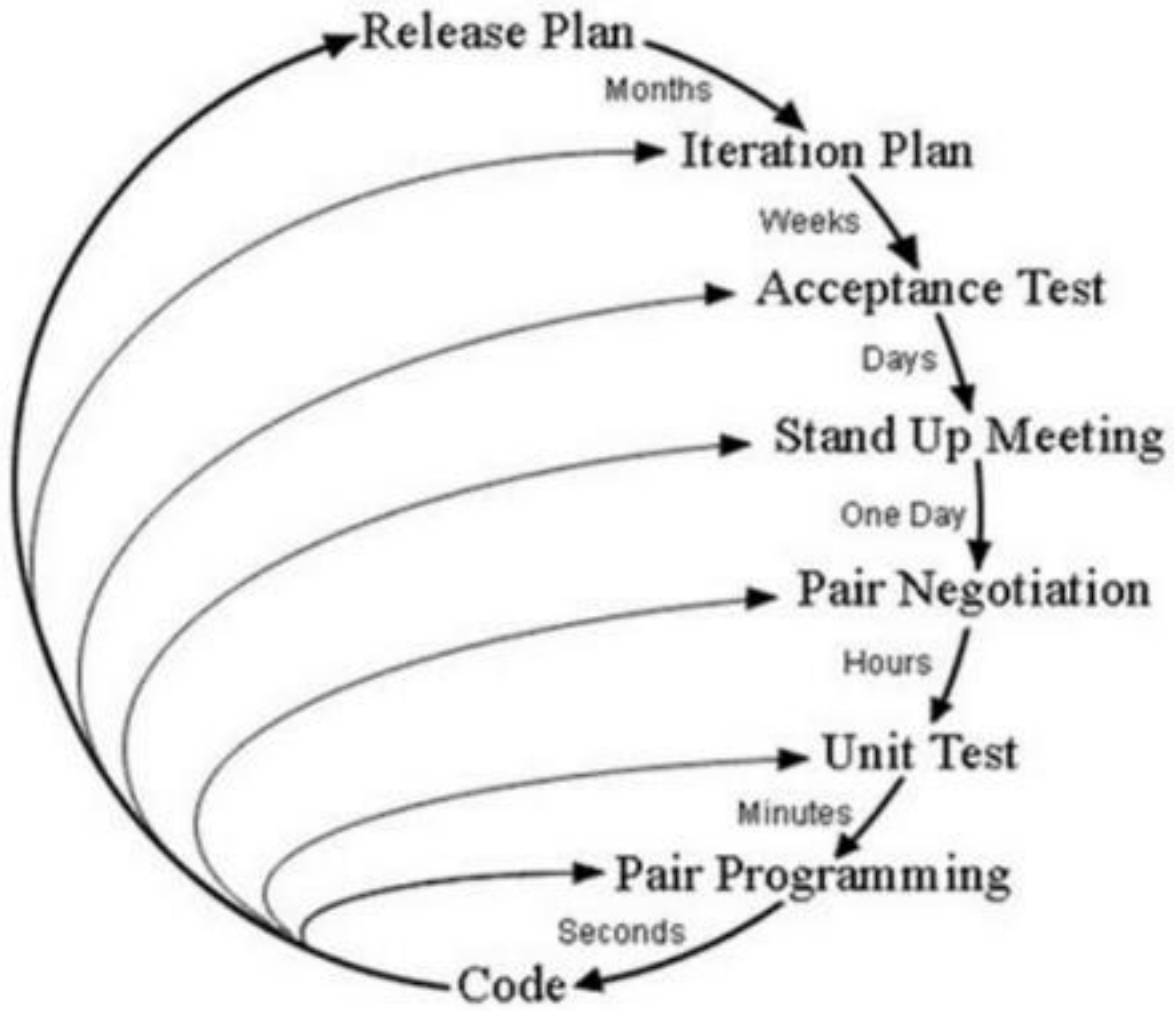


Fig – Typical planning and feedback loop of Agile XP model

- XP prescribes a set of day-to-day practices for managers and developers; the practices are meant to embody and encourage particular values. Proponents believe that the exercise of these practices—which are traditional software engineering practices taken to so-called "extreme" levels—leads to a development process that is more responsive to customer needs ("agile") than traditional methods, while creating software of similar or better quality.

Advantages of deploying XP in Testing:

- XP lays strong focus on customer satisfaction.
- XP encourages early and frequent testing.
- XP looks to make the process as efficient and effective as possible by promoting the use of test automation.
- XP demands that all code have associated tests and that where code is changed that the tests be re-executed.
- XP requires test design to be completed before coding has started, with all code having associated unit tests.

10. Challenges in Automation

Software Testing has a lot of challenges both in a manual as well as in automation. Generally in manual testing scenario developers through the build to test team assuming the responsible test team or tester will pick the build.

So here we go with the top challenges:

- **Testing the complete application:**
  Is it possible? I think impossible. There are millions of test combinations. It's not possible to test each and every combination both in the manual as well as in automation testing. If you try all these combinations you will never ship the product.

- **Misunderstanding of company processes:**
  Sometimes you just don't pay proper attention what the company-defined processes are and these are for what purposes. There are some myths in testers that they should only go with company processes even these processes are not applicable for their current testing scenario. This results in incomplete and inappropriate application testing.

- **Relationship with developers:**
  Big challenge. Requires very skilled tester to handle this relation positively and even by completing the work in testers way. There are simply hundreds of excuses developers or testers can make when they do not agree with some points. For this tester also requires good communication, troubleshooting and analyzing skill.

- **Regression testing:**
  When a project goes on expanding the regression testing work simply becomes uncontrolled. Pressure to handle the current functionality changes, previous working functionality checks, and bug tracking.

- **Lack of skilled testers:**
  I will call this as 'wrong management decision' while selecting or training testers for their project task in hand. These unskilled fellows may add more chaos than simplifying the testing work. This results in incomplete, insufficient and ad-hoc testing throughout the testing life cycle.

- **Testing always under time constraint:**
  Hey tester, we want to ship this product by this weekend, are you ready for completion? When this order comes from the boss, tester simply focuses on task completion and not on the test coverage and quality of work. There is a huge list

of tasks that you need to complete within specified time. This includes writing, executing, automating and reviewing the test cases.

- **Which tests to execute first?**
  If you are facing the challenge stated in point no 6, then how will you take a decision which test cases should be executed and with what priority? Which tests are important over others? This requires good experience to work under pressure.

- **Understanding the requirements:**
  Sometimes testers are responsible for communicating with customers for understanding the requirements. What if tester fails to understand the requirements? Will he be able to test the application properly? Definitely No! Testers require good listening and understanding capabilities.

11. Data-driven Testing

Data-driven testing is creation of test scripts where test data and/or output values are read from data files instead of using the same hard-coded values each time the test runs. This way, testers can test how the application handles various inputs effectively. It can be any of the below data files.

- Data pools
- Excel files
- ADO objects
- CSV files
- ODBC sources

Flow Diagram:

Data Driven Testing can be best understood by the following diagram:

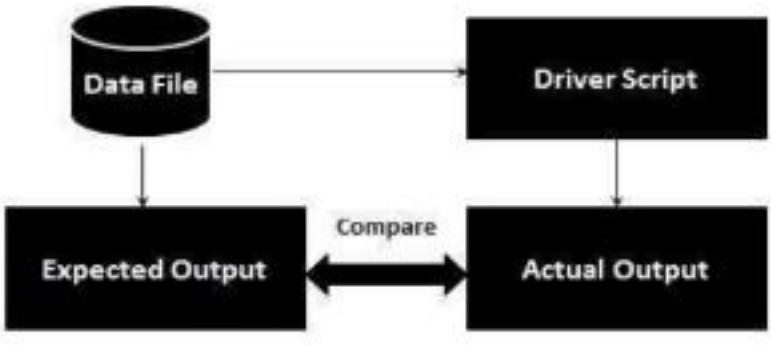


Fig – Data Driven Testing

- Frequently we have multiple data sets which we need to run the same tests on. To create an individual test for each data set is a lengthy and time-consuming process.
- Data Driven testing framework resolves this problem by keeping the data separate from Functional tests. The same test script can execute for different combinations of input test data and generate test results.

Example:

We want to test the login system with multiple input fields with 1000 different data sets.

To test this, you can take following different approaches:

- **Approach 1)** Create 1000 scripts one for each dataset and runs each test separately one by one.
- **Approach 2)** manually change the value in the test script and run it several times.
- **Approach 3)** Import the data from the excel sheet. Fetch test data from excel rows one by one and execute the script.

12. Automation Tools like JUnit and Jmeter

- JUnit is by far the most popular unit testing framework for the Java language. According to a recent survey, more than 30% of GitHub projects use JUnit for unit testing.

- JMeter developers and contributors also use JUnit to verify that a new feature or bug fix won't break any existing functionality.

- If you're already using JMeter, you can use its JUnit Request sampler. This sampler can execute individual JUnit test cases, enabling you to:

- Run JUnit test cases in multiple threads
- Apply JMeter features like:
- Logic Controllers
- Timers
- Assertions
- Listeners
- Control the test execution order and conditions to build an advanced test plan

- JUnit 5 is the next generation of JUnit. The goal is to create an up-to-date foundation for developer-side testing on the JVM. This includes focusing on Java 8 and above, as well as enabling many different styles of testing.
- JUnit 5 is the result of JUnit Lambda and its crowd funding campaign on Indie go

- **JMeter**
  The Apache JMeter is pure Java open source software, which was first developed by Stefano Mazzocchi of the Apache Software Foundation, designed to load test functional behavior and measure performance. You can use JMeter to analyze and measure the performance of web application or a variety of services.
  Performance Testing means testing a web application against heavy load, multiple and concurrent user traffic. JMeter originally is used for testing Web Application or FTP application. Nowadays, it is used for a functional test, database server test etc.

Fig – Diagram of JMeter

JMeter simulates a group of users sending requests to a target server, and return statistics information of target server.

JMeter Advantages

- **Open source license**: JMeter is totally free, allows developer use the source code for the development
- **Friendly GUI**: JMeter is extremely easy to use and doesn't take time to get familiar with it
- **Platform independent**: JMeter is 100% pure Java desktop application. So it can run on multiple platforms
- **Full multithreading framework**. JMeter allows concurrent and simultaneous sampling of different functions by a separate thread group
- **Visualize Test Result:** Test result can be displayed in a different format such as chart, table, tree and log file
- **Easy installation**: You just copy and run the *.bat file to run JMeter. No installation needed.

- **Highly Extensible**: You can write your own tests. JMeter also supports visualization plugins allow you to extend your testing
- **Multiple testing strategies**: JMeter supports many testing strategies such as Load Testing, Distributed Testing, and Functional Testing.
- **Simulation**: JMeter can simulate multiple users with concurrent threads, create a heavy load against web application under test
- **Support multi-protocol**: JMeter does not only support web application testing but also evaluate database server performance. All basic protocols such as HTTP, JDBC, LDAP, SOAP, JMS, and FTP are supported by JMeter

References:

1. Naresh Chauhan, "Software Testing Principles and Practices ", OXFORD, ISBN-10: 0198061846. ISBN-13: 9780198061847.

2. Stephen Kan, "Metrics and Models in Software Quality Engineering", Pearson, ISBN-10: 0133988082; ISBN-13: 978-0133988086.

3. Srinivasan Desikan, Gopalswamy Ramesh, "Software Testing Principles and Practices", Pearson, ISBN-10: 817758121X.

Unit IV Selenium Tool

1. Introducing Selenium
2. Brief History of the Selenium Project
   2.1 Selenium's Tool Suite
   2.2 Selenium IDE
   2.3 Selenium RC
   2.4 Selenium Web drivers
   2.5 Selenium Grid
3. Test Design Considerations

1. Introducing Selenium

- Selenium is a free (open source) automated testing suite for web applications across different browsers and platforms.
- It is quite similar to HP Quick Test Pro (QTP now UFT) only that Selenium focuses on automating web-based applications. Testing done using Selenium tool is usually referred as Selenium Testing.
- Selenium is not just a single tool but a suite of software's, each catering to different testing needs of an organization. It has four components.

- Selenium Integrated Development Environment (IDE)
- Selenium Remote Control (RC)
- Web Driver
- Selenium Grid

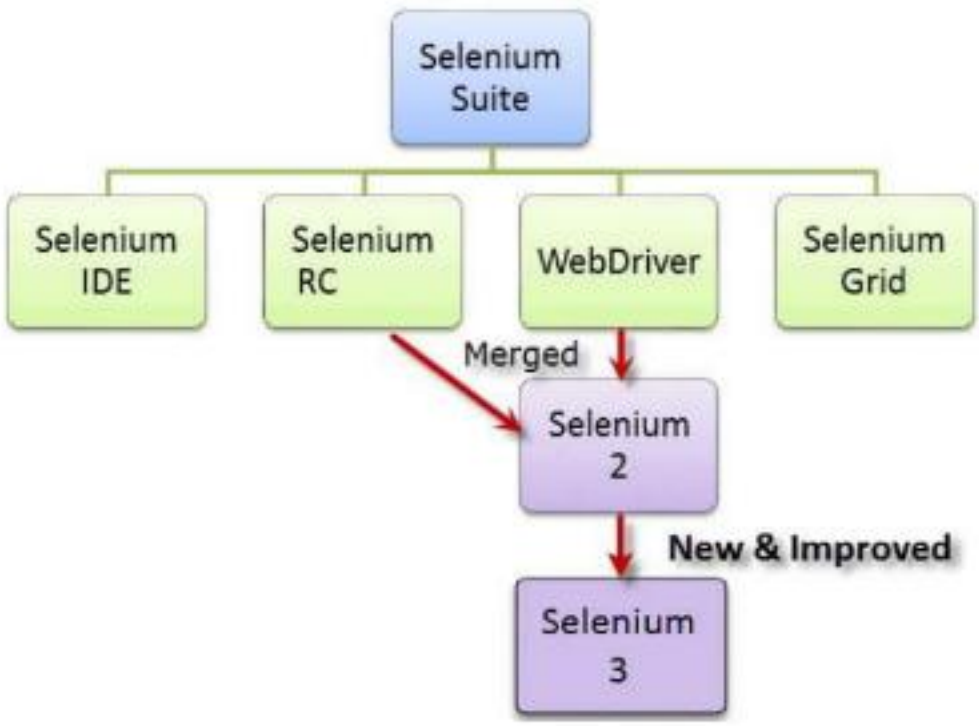


Fig – Architecture of Selenium

- At the moment, Selenium RC and Web Driver are merged into a single framework to form Selenium 2. Selenium 1, by the way, refers to Selenium RC.
- Selenium is a set of different software tools each with a different approach to supporting test automation. Most Selenium QA Engineers focus on the one or two tools that most meet the needs of their project, however learning all the tools will give you many different options for approaching different test automation problems.
- The entire suite of tools results in a rich set of testing functions specifically geared to the needs of testing of web applications of all types. These operations are highly flexible, allowing many options for locating UI elements and

comparing expected test results against actual application behavior. One of Selenium's key features is the support for executing one's tests on multiple browser platforms.

2. Brief History of the Selenium Project

- Selenium first came to life in 2004 when Jason Huggins was testing an internal application at Thought Works. Being a smart guy, he realized there were better uses of his time than manually stepping through the same tests with every change he made.
- He developed a Javascript library that could drive interactions with the page, allowing him to automatically rerun tests against multiple browsers. That library eventually became Selenium Core, which underlies all the functionality of Selenium Remote Control (RC) and Selenium IDE. Selenium RC was ground-breaking because no other product allowed you to control a browser from a language of your choice.
- While Selenium was a tremendous tool, it wasn't without its drawbacks. Because of its Javascript based automation engine and the security limitations browsers apply to Javascript, different things became impossible to do.
- To make things worse, web apps became more and more powerful over time, using all sorts of special features new browsers provide and making these restrictions more and more painful.
- In 2006 a plucky engineer at Google named Simon Stewart started work on a project he called Web Driver. Google had long been a heavy user of Selenium, but testers had to work around the limitations of the product.
- Simon wanted a testing tool that spoke directly to the browser using the 'native' method for the browser and operating system, thus avoiding the restrictions of a sandboxed Java script environment. The Web Driver project began with the aim to solve the Selenium' pain-points.
- Jump to 2008. The Beijing Olympics mark China's arrival as a global power, massive mortgage default in the United States triggers the worst international recession since the Great Depression, The Dark Knight is viewed by every human (twice), still reeling from the untimely loss of Heath Ledger. But the most important story of that year was the merging of Selenium and Web Driver. Selenium had massive community and commercial support, but Web Driver was clearly the tool of the future.
- The joining of the two tools provided a common set of features for all users and brought some of the brightest minds in test automation under one roof. Perhaps the best explanation for why Web Driver and Selenium are merging was

detailed by Simon Stewart, the creator of Web Driver, in a joint email to the Web Driver and Selenium community on August 6, 2009.

2.1 Selenium's Tool Suite

Selenium is composed of multiple software tools. Each has a specific role.

- **Selenium 2 (aka. Selenium Web Driver)**

- Selenium 2 is the future direction of the project and the newest addition to the Selenium toolkit. This brand new automation tool provides all sorts of awesome features, including a more cohesive and objects oriented API as well as an answer to the limitations of the old implementation.
- As you can read in Brief History of The Selenium Project, both the Selenium and Web Driver developers agreed that both tools have advantages and that merging the two projects would make a much more robust automation tool.
- Selenium 2.0 is the product of that effort. It supports the WebDriver API and underlying technology, along with the Selenium 1 technology underneath the WebDriver API for maximum flexibility in porting your tests. In addition, Selenium 2 still runs Selenium 1's Selenium RC interface for backwards compatibility.

- **Selenium 1 (aka. Selenium RC or Remote Control)**

- As you can read in Brief History of The Selenium Project, Selenium RC was the main Selenium project for a long time, before the WebDriver/Selenium merge brought up Selenium 2, the newest and more powerful tool.

2.2 Selenium IDE

- Selenium Integrated Development Environment (IDE) is the simplest framework in the Selenium suite and is the easiest one to learn. It is a Firefox plug in that you can install as easily as you can with other plugins. However, because of its simplicity, Selenium IDE should only be used as a prototyping tool.
- Selenium IDE (Integrated Development Environment) is a prototyping tool for building test scripts. It is a Firefox and Chrome plugin and provides an easy-to-use interface for developing automated tests. Selenium IDE has a recording feature, which records user actions as they are performed and then exports them as a reusable script in one of many programming languages that can be later executed.

- Even though Selenium IDE has a "Save" feature that allows users to keep the tests in a table-based format for later import and execution, it is not designed to run your test passes nor is it designed to build all the automated tests you will need.

- Specifically, Selenium IDE doesn't provide iteration or conditional statements for test scripts. At the time of writing there is no plan to add such thing. The reasons are partly technical and partly based on the Selenium developers encouraging best practices in test automation which always requires some amount of programming.

- Selenium IDE is simply intended as a rapid prototyping tool. The Selenium developers recommend for serious, robust test automation either Selenium 2 or Selenium 1 to be used with one of the many supported programming languages.

2.3 Selenium RC

- Unfortunately, testers using Selenium Core had to install the whole application under test and the web server on their own local computers because of the restrictions imposed by the same origin policy.

- So another Thought Work's engineer, Paul Hamm ant, decided to create a server that will act as an HTTP proxy to "trick" the browser into believing that Selenium Core and the web application being tested come from the same domain. This system became known as the Selenium Remote Control or Selenium 1.

- This is the old, support platform for Selenium 1.0. It should still apply to the Selenium 2.0 release of Selenium-RC.

Browser	Selenium IDE	Selenium 1 (RC)	Operating Systems
Firefox 3.x	Record and playback tests	Start browser, run tests	Windows, Linux, Mac
Firefox 3	Record and playback tests	Start browser, run tests	Windows, Linux, Mac
Firefox 2	Record and playback tests	Start browser, run tests	Windows, Linux, Mac
IE 8	Test execution only via Selenium RC*	Start browser, run tests	Windows

IE 7	Test execution only via Selenium RC*	Start browser, run tests	Windows
IE 6	Test execution only via Selenium RC*	Start browser, run tests	Windows
Safari 4	Test execution only via Selenium RC	Start browser, run tests	Windows, Mac
Safari 3	Test execution only via Selenium RC	Start browser, run tests	Windows, Mac
Safari 2	Test execution only via Selenium RC	Start browser, run tests	Windows, Mac
Opera 10	Test execution only via Selenium RC	Start browser, run tests	Windows, Linux, Mac
Opera 9	Test execution only via Selenium RC	Start browser, run tests	Windows, Linux, Mac
Opera 8	Test execution only via Selenium RC	Start browser, run tests	Windows, Linux, Mac
Google Chrome	Test execution only via Selenium RC	Start browser, run tests	Windows, Linux, Mac
Others	Test execution only via Selenium RC	Partial support possible**	As applicable

Tests developed on Firefox via Selenium IDE can be executed on any other supported browser via a simple Selenium RC command line.

- Selenium RC server can start any executable, but depending on browser security settings there may be technical limitations that would limit certain features.

2.4 Selenium Web drivers

- The Web Driver proves itself to be better than both Selenium IDE and Selenium RC in many aspects. It implements a more modern and stable approach in automating the browser's actions.
- Web Driver, unlike Selenium RC, does not rely on JavaScript for Automation. It controls the browser by directly communicating with it.
- Selenium-Web Driver supports the following browsers along with the operating systems these browsers are compatible with.

- Google Chrome
- Internet Explorer 7, 8, 9, 10, and 11 on appropriate combinations of Vista, Windows 7, Windows 8, and Windows 8.1. As of April 15 2014, IE 6 is no longer supported. The driver supports running 32-bit and 64-bit versions of the browser where applicable
- Firefox: latest ESR, previous ESR, current release, one previous release
- Safari
- Opera
- Html Unit
- phantomjs
- Android (with Selendroid or appium)
- iOS (with ios-driver or appium)

2.5 Selenium Grid

- Selenium-Grid allows the Selenium RC solution to scale for large test suites and for test suites that must be run in multiple environments. Selenium Grid allows you to run your tests in parallel, that is, different tests can be run at the same time on different remote machines.

- This has two advantages. First, if you have a large test suite, or a slow-running test suite, you can boost its performance substantially by using Selenium Grid to divide your test suite to run different tests at the same time using those different machines.

- Also, if you must run your test suite on multiple environments you can have different remote machines supporting and running your tests in them at the same time.

- In each case Selenium Grid greatly improves the time it takes to run your suite by making use of parallel processing.

- Selenium Grid is a tool used together with Selenium RC to run parallel tests across different machines and different browsers all at the same time. Parallel execution means running multiple tests at once.

Features:

- Enables simultaneous running of tests in multiple browsers and environments.
- Saves time enormously.
- Utilizes the hub-and-nodes concept. The hub acts as a central source of Selenium commands to each node connected to it.

3. Test Design Considerations

- **Introducing Test Design**
  The information that will be useful to both those new to test automation and for the experienced QA professional. Here we describe the most common types of automated tests. We also describe 'design patterns' commonly used in test automation for improving the maintenance and extensibily of your automation suite. The more experienced reader will find these interesting if not already using these techniques.

- **Types of Tests**
  What parts of your application should you test? That depends on aspects of your project: user expectations, time allowed for the project, priorities set by the project manager and so on. Once the project boundaries are defined though, you, the tester, will certainly make many decisions on what to test.
  We've created a few terms here for the purpose of categorizing the types of test you may perform on your web application. These terms are by no means standard, although the concepts we present here are typical for web-application testing.

- **Testing Static Content**
  The simplest type of test, a content test, is a simple test for the existence of a static, non-changing, UI element. For instance Does each page have its expected page title? This can be used to verify your test found an expected page after following a link.
  Does the application's home page contain an image expected to be at the top of the page?
  Does each page of the website contain a footer area with links to the company contact page, privacy policy, and trademarks information?
  Does each page begin with heading text using the <h1> tag? And, does each page have the correct text within that header?
  You may or may not need content tests. If your page content is not likely to be affected then it may be more efficient to test page content manually. If, for example, your application involves files being moved to different locations, content tests may prove valuable.

- **Testing Links**
  A frequent source of errors for web-sites is broken links or missing pages behind links. Testing involves clicking each link and verifying the expected page. If static links are infrequently changed then manual testing may be sufficient. However if your web designers frequently alter links, or if files are occasionally relocated, link tests should be automated.

- **Function Tests**
  These would be tests of a specific function within your application, requiring some type of user input, and returning some type of results. Often a function test will involve multiple pages with a form-based input page containing a collection of input fields, Submit and Cancel operations, and one or more response pages. User input can be via text-input fields, check boxes, drop-down lists, or any other browser-supported input.
  Function tests are often the most complex tests you'll automate, but are usually the most important. Typical tests can be for login, registration to the site, user account operations, account settings changes, complex data retrieval operations, among others. Function tests typically mirror the user-scenarios used to specify the features and design or your application.

- **Testing Dynamic Elements**
  Often a web page element has a unique identifier used to uniquely locate that element within the page. Usually these are implemented using the html tag's 'id' attribute or its 'name' attribute. These names can be a static, i.e. unchanging, string constant.
  They can also be dynamically generated values that vary each instance of the page. For example, some web servers might name a displayed document doc3861 on one instance of a page, and 'doc6148' on a different instance of the page depending on what 'document' the user was retrieving.
  A test script verifying that a document exists may not have a consistent identifier to use for locating that document. Often, dynamic elements with varying identifiers are on some type of result page based on a user action. This though certainly depends on the function of the web application.

References:

1. Naresh Chauhan, "Software Testing Principles and Practices ", OXFORD, ISBN-10: 0198061846. ISBN-13: 9780198061847.

2. Stephen Kan, "Metrics and Models in Software Quality Engineering", Pearson, ISBN-10: 0133988082; ISBN-13: 978-0133988086.

3. Srinivasan Desikan, Gopalswamy Ramesh, "Software Testing Principles and Practices", Pearson, ISBN-10: 817758121X.

Unit V Quality Management

1. Software Quality
    - 1.1Software Quality Dilemma
    - 1.2 Achieving Software Quality
2. Software Quality Assurance
    - 2.1 Elements of SQA
    - 2.2 SQA Tasks
    - 2.3 Goals and Metrics
    - 2.4 Formal Approaches to SQA
    - 2.5 Statistical Software Quality Assurance
3. Six Sigma for Software Engineering
4. ISO 9000 Quality Standards
5. SQA Plan

1. Software Quality

- In the context of software engineering, software quality refers to two related but distinct notions:

- Software functional quality reflects how well it complies with or conforms to a given design, based on functional requirements or specifications. That attribute can also be described as the fitness for purpose of a piece of software or how it compares to competitors in the marketplace as a worthwhile product. It is the degree to which the correct software was produced.

- Software structural quality refers to how it meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability. It has a lot more to do with the degree to which the software works as needed.

- Many aspects of structural quality can be evaluated only statically through the analysis of the software inner structure, its source code, at the unit level, the technology level and the system level, which is in effect how its architecture adheres to sound principles of software architecture outlined in a paper on the topic by OMG. But some structural qualities, such as usability, can be assessed only dynamically.

- Other aspects, such as reliability, might involve not only the software but also the underlying hardware, therefore, it can be assessed both statically and dynamically (stress test). Functional quality is typically assessed dynamically but it is also possible to use static tests.

1.1 Software Quality Dilemma

- Any time software and business come together, there is an inherent conflict between "get it done fast" and "do a good job". This conflict often comes to a head when deadlines are missed, whether through unrealistic expectation or underestimation on the part of the developers.

- The dilemma between quantity, speed and feature set isn't going to go away any time soon. It's an inherent dilemma in software. But there are approaches we can take to help solve it.

- Any software project has three core elements: the speed at which the software ships, the fidelity of the feature set, and the quality of the underlying code base. In a perfect world, we'd be right in the middle, like so:

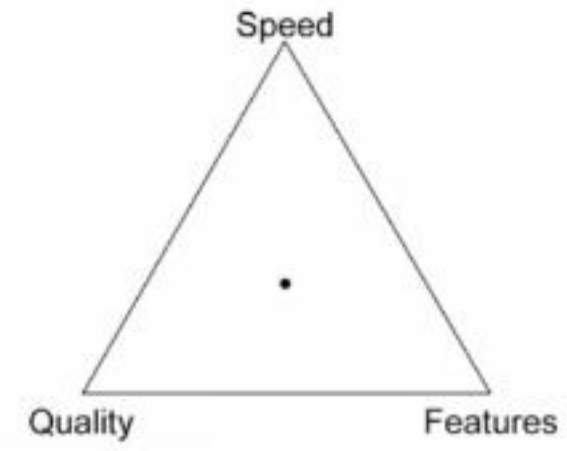


Fig1 - Software Quality Dilemma

- The triangle makes clear that as you move towards one goal, you tend to move away from the others. Pushing for speed, for example, sacrifices quality and feature fidelity to some degree.
- And yet, often it's impossible to live completely in the center of the triangle. After all, there are deadlines to be met, and sometimes we need to push software out, even while incurring technical debt. But leaving the center of the triangle does not necessarily mean project ruin.
- Imagine that the triangle has concentric zones. The outermost zone is a danger zone, where one priority is overemphasized to the detriment of all others. Inside that zone is a warning area, an area where a project can live temporarily, or even for a longer period of time, before things begin to go wrong.
- This area might represent short pushes in one direction or another for the good of the project or the company. And of course in the center of the triangle is a "green zone", where all priorities are largely equal in value to the organization and the team.

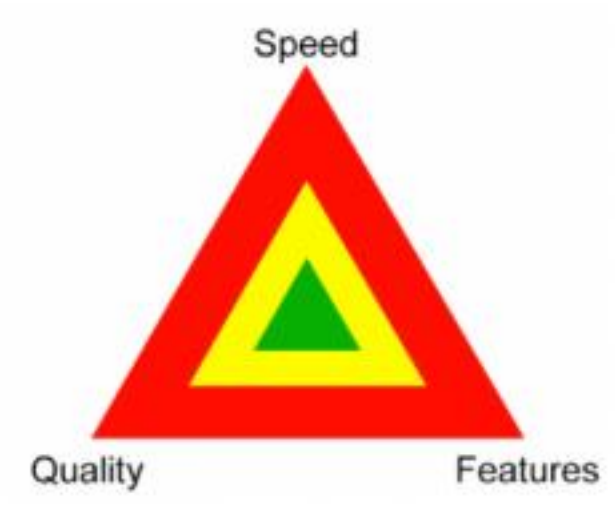


Fig 2 - Software Quality Dilemma

- Notice how the widest and largest area is the danger zone around the outside of the triangle. The inner core of warning and good are smaller. This is only natural: it's easy in a software project to move too far off the safe ground, and too far towards one of the corners of the triangle.

1.2 Achieving Software Quality

- Coverage testing helps the tester create a thorough set of tests and gives a measure of test completeness. The concepts of coverage testing are well-described in the literature.

- However, there are few tools that actually implement these concepts for standard programming languages, and their realistic use on large-scale projects is rare. In this article, we describe the uses of a dataflow coverage-testing tool for C programs-called ATAC for Automatic Test Analysis for C/sup 3/-in measuring, controlling, and understanding the testing process.

- We present case studies of two real-world software projects using ATAC. The first study involves 12 program versions developed by a university/industry fault-tolerant software project for a critical automatic-flight-control system.

- The second study involves a Bell core project of 33 program modules. These studies indicate that coverage analysis of programs during testing not only gives a clear measure of testing quality but also reveals important aspects of software structure.

- Understanding the structure of a program, as revealed in coverage testing, can be a significant component in confident assessment of overall software quality.

- The universal equation for excellence. This theme is rapidly becoming emphasized in the information systems industry, and in particular in software development and integration, due to the increasing cost of software in relation to hardware. Software quality control is a relatively new concept.

- Therefore, effective software quality control programs are not common. Striving for excellence without a plan will not necessarily produce a quality product.

- It is necessary for the efforts of all individuals involved in the software development to be coordinated and the development process analyzed via one software quality plan.

- The unique and original software quality assurance program based on software quality control principles.

2. Software Quality Assurance

- Software quality assurance (SQA) is a process which assures that all software engineering processes, methods, activities and work items are monitored and comply against the defined standards. .
- SQA incorporates all software development processes starting from defining requirements to coding until release. Its prime goal is to ensure quality. It is a set of methods that try to ensure the quality of all projects in the software process.
- This includes standards and procedures that administrators may use to review and audit software products and activities to verify that the software meets standards.
- SQA encompasses the entire software development process including processes such as requirements definition, software design, coding, source code control, code reviews, software configuration management, testing, release management, and product integration. It is organized into: goals, commitments, abilities, activities, measurements, and verification.

- **SQA Activities**

  Given below is the list of SQA activities:

- **Creating an SQA Management Plan:**

  The foremost activity includes laying down a proper plan regarding how the SQA will be carried out in your project.

  Along with what SQA approach you are going to follow, what engineering activities will be carried out, and it also includes ensuring that you have a right talent mix in your team.

- **Setting the Checkpoints:**

  The SQA team sets up different checkpoints according to which it evaluates the quality of the project activities at each checkpoint/project stage. This ensures regular quality inspection and working as per the schedule.

- **Apply software Engineering Techniques:**

  Applying some software engineering techniques aids a software designer in achieving high-quality specification. For gathering information, a designer may use techniques such as interviews and FAST (Functional Analysis System Technique).

2.1 Elements of SQA

There are 10 essential elements of SQA which are enlisted below for your reference:

1. Software engineering Standards
2. Technical reviews and audits
3. Software Testing for quality control
4. Error collection and analysis
5. Change management
6. Educational programs
7. Vendor management
8. Security management
9. Safety
10. Risk management

SQA Techniques

There are several techniques for SQA. Auditing is the chief technique that is widely adopted. However, we have a few other significant techniques as well.

Various SQA Techniques include:

- **Auditing:** Auditing involves inspection of the work products and its related information to determine if the set of standard processes were followed or not.
- **Reviewing**: A meeting in which the software product is examined by both the internal and external stakeholders to seek their comments and approval.
- **Code Inspection:** It is the most formal kind of review that does static testing to find bugs and avoid defect growth in the later stages. It is done by a trained mediator/peer and is based on rules, checklist, entry and exit criteria. The reviewer should not be the author of the code.
- **Design Inspection:** Design inspection is done using a checklist that inspects

2.2 SQA Tasks

- **Description of SQA Task 1**

  The Engine Software Engineer will check with the Requirements Specification on a weekly basis to make sure that what he is coding conforms to the original design. This process will ensure that the product meets the client's expectations and standards and that the engine, up to its current point, is working properly.

- **Work products and documentation for Task 1**

  As a result of Task 1, any major deviations that occur will be expressed to the other group members and documented on a separate defect log. Documentation will ensure that each group member is aware of the change(s) made to the engine so that each part of the project can be adjusted accordingly.

- **Description of SQA Task 2**

  This process will ensure that the product meets the client's expectations and standards and that the user-interface, up to its current point, is working properly.

- **Work products and documentation for Task 2**

  As a result of Task 2, any major deviations that occur will be expressed to the other group members and documented on a separate defect log. Documentation will ensure that each group member is aware of the change(s) made to the interface, so that each part of the project can be adjusted accordingly.

- **Description of SQA Task 3**

  Each member of the group will routinely perform a hands-on evaluation of the user-interface. Noted evaluation criteria will be: ease of use, principle of least astonishment, unobtrusiveness, and overall attractiveness. This is done to ensure that the user-interface is evaluated honestly, and remains easily understandable and attractive.

- **Work products and documentation for Task 3**

  As a result of Task 3, all suggestions or concerns are expressed to the User-Interface Engineer. These are recorded in the defect log. Based on these concerns, the User-Interface Engineer takes note and makes the appropriate

adjustments to the user-interface to make sure that the final product is satisfactory

2.3 Goals and Metrics

Goal	Attribute	Metric
Requirement quality	Ambigully	Number of ambiguous modifiers (e.., many, large, human–friendly)
	Completeness	Number of TBA, TBD
	Understand ability	Number of sections/subsections
	Volatility	Number of changes per requirement Time (by activity) when change is requested
	Traceability	Number of requirements not traceable to design/code
	Model clarity	Number of UML models Number of descriptive pages per model Number of UML errors
Design quality	Architectural integrity	Existence of architectural model
	Component completeness	Number of components that trace to architectural model
	Interface complexity	Complexity of procedural design Average number of pick to get to a typical function or content

	Patterns	Layout appropriateness Number of patterns used
Code quality	Complexity	Cyclomatic complexity
	Maintainability	Percent internal comments
	Understand ability	Variable naming conventions Percent reused components
	Reusability Documentation	Readability index
QC effectiveness	Resource allocation	Staff hour percentage per activity
	Completion rate	Actual vs. budgeted completion time
	Review effectiveness	See review metrics
	Testing effectiveness	Number of errors found and criticality Effort required to correct an error Origin of error

2.4 Formal Approaches to SQA

- Software quality is everyone's job; that it can be achieved through competent analysis, design, coding, and testing, as well as through the application of formal technical reviews, a multitier testing strategy, better control of software work products and the changes made to them, and the application of accepted software engineering standards.

- In addition, quality can be defined in terms of a broad array of quality factors and measured (indirectly) using a variety of indices and metrics.

- Over the past two decades, a small, but vocal, segment of the software engineering community has argued that a more formal approach to software quality assurance is required. It can be argued that a computer program is a mathematical object.

- A rigorous syntax and semantics can be defined for every programming language, and work is underway to develop a similarly rigorous approach to the specification of software requirements. If the requirements model (specification) and the programming language can be represented in a rigorous manner, it should be possible to apply mathematic proof of correctness to demonstrate that a program conforms exactly to its specifications.

- Attempts to prove programs correct are not new. Dijkstra and Linger, Mills, and Witt, among others, advocated proofs of program correctness and tied these to the use of structured programming concepts.

2.5 Statistical Software Quality Assurance

- Traditional compliance testing techniques can sometimes provide limited pass/fail information, which results in insufficient measurements on the batch's quality control, identification of the root cause of failure results and overall quality assurance (QA) in the production process.

- Intertek combines legal, customer and essential safety requirements to customize a workable QA process, called Statistical Quality Assurance (SQA). SQA is used to identify the potential variations in the manufacturing process and predict potential defects on a parts-per-million (PPM) basis. It provides a statistical description of the final product and addresses quality and safety issues that arise during manufacturing.

SQA consists of three major methodologies:

- **Force Diagram** - A Force Diagram describes how a product should be tested. Intertek engineers base the creation of Force Diagrams on our knowledge of foreseeable use, critical manufacturing process and critical components that have high potential to fail.

- **Test-to-Failure (TTF)** - Unlike any legal testing, TTF tells manufacturers on how many defects they are likely to find in every million units of output. This

information is incorporated into the process and concludes if a product needs improvement in quality or if it is being over engineered, which will eventually lead to cost savings.

- **Intervention** - Products are separated into groups according to the total production quantity and production lines. Each group then undergoes an intervention. The end result is measured by Z-value, which is the indicator of quality and consistency of a product to a specification. Intervention allows manufacturers to pinpoint a defect to a specific lot and production line; thus saving time and money in corrective actions.

3. Six Sigma for Software Engineering

- Six Sigma (6σ) is a set of techniques and tools for process improvement. It was introduced by engineer Bill Smith while working at Motorola in 1980.
- Jack Welch made it central to his business strategy at General Electric in 1995. A six sigma process is one in which 99.99966% of all opportunities to produce some feature of a part are statistically expected to be free of defects.
- Six Sigma strategies seek to improve the quality of the output of a process by identifying and removing the causes of defects and minimizing variability in manufacturing and business processes. It uses a set of quality management methods, mainly empirical, statistical methods, and creates a special infrastructure of people within the organization who are experts in these methods.
- Each Six Sigma project carried out within an organization follows a defined sequence of steps and has specific value targets, for example: reduce process cycle time, reduce pollution, reduce costs, increase customer satisfaction, and increase profits.
- The term Six Sigma (capitalized because it was written that way when registered as a Motorola trademark on December 28, 1993) originated from terminology associated with statistical modeling of manufacturing processes.
- The maturity of a manufacturing process can be described by a sigma rating indicating its yield or the percentage of defect-free products it create specifically, within how many standard deviations of a normal distribution the fraction of defect-free outcomes corresponds to. Motorola set a goal of "six sigma" for all of its manufacturing.

Features of Six Sigma:

- Six Sigma's aim is to eliminate waste and inefficiency, thereby increasing customer satisfaction by delivering what the customer is expecting.
- Six Sigma follows a structured methodology, and has defined roles for the participants.

- Six Sigma is a data driven methodology, and requires accurate data collection for the processes being analyzed.
- Six Sigma is about putting results on Financial Statements.
- Six Sigma is a business-driven, multi-dimensional structured approach for –
- Improving Processes
- Lowering Defects
- Reducing process variability
- Reducing costs
- Increasing customer satisfaction
- Increased profits

4. ISO 9000 Quality Standards

- The ISO 9000 series was created by the International Organization for Standardization (ISO) as international requirements and guidelines for quality management systems. It was originally introduced in 1987 and over the years has established itself in the global economy having been adopted in over 178 countries with over one million registrations.
- The phrase "ISO 9000 family" or "ISO 9000 series" refers to a group of quality management standards which are process standards (not product standards).
- ISO 9000 Quality management systems – Fundamentals and Vocabulary, referenced in all ISO 9000 Standards.
- ISO 9001 Quality management systems – Requirements, contains the requirements an organization must comply with to become ISO 9001 certified.
- ISO 9002 – **Guidelines** for the application of ISO 9001:2015
- ISO 9004 – Managing for the sustained success of an organization, provides **guidelines** for sustaining QMS success through evaluation and performance improvement's.
- ISO 9001 lists requirements, while the other standards in the 9000 family provide guidelines and information. People often say "ISO 9000 certified", but what they mean is they have met the requirements of the ISO 9001 standard. Read more about ISO 9001 Certification.
- The series is not industry specific and is applicable to any manufacturing, distribution or service organization. It is managed by Technical Committee (TC) 176, comprised of international members from many industries and backgrounds.

5. SQA Plan

- The SQA plan is a document that specifies the process to be followed in each step of the software development and the procedures to be followed in each activity of such a process.

- The objective of SQA plan is to ensure that the development of the software is based on a course of action and that from time to time the development can be measured controlled and monitored with respect to such a course of action-so that the end product is as per the specifications.

- The plan is governed by several quality standards, policies and models such as IS09000, SEI CMM and Baldrige.

- Abbreviated as SQAP, the software quality assurance plan comprises of the procedures, techniques, and tools that are employed to make sure that a product or service aligns with the requirements defined in the SRS (software requirement specification).

Fig - Software Quality Assurance Plan

- The plan identifies the SQA responsibilities of a team, lists the areas that need to be reviewed and audited. It also identifies the SQA work products.

The SQA plan document consists of the below sections:

- Purpose section
- Reference section
- Software configuration management section

- Problem reporting and corrective action section
- Tools, technologies and methodologies section
- Code control section
- Records: Collection, maintenance and retention section
- Testing methodology

References:

1. Naresh Chauhan, "Software Testing Principles and Practices ", OXFORD, ISBN-10: 0198061846. ISBN-13: 9780198061847.

2. Stephen Kan, "Metrics and Models in Software Quality Engineering", Pearson, ISBN-10: 0133988082; ISBN-13: 978-0133988086.

3. Srinivasan Desikan, Gopalswamy Ramesh, "Software Testing Principles and Practices", Pearson, ISBN-10: 817758121X.

Unit VI Software Quality Tools

1. Total Quality Management
2. Product Quality Metrics
3. In process Quality Metrics
4. Software maintenance
5. Ishikawa's 7 basic tools
   - 5.1 Checklists
   - 5.2 Pareto diagrams
   - 5.3 Histogram
   - 5.4 Control chart
   - 5.5 Cause Effect diagram
6. Defect Removal Effectiveness and Process Maturity Level

1. Total Quality Management

- Total Quality Management is an extensive and structured organization management approach that focuses on continuous quality improvement of products and services by using continuous feedback. Joseph Juran was one of the founders of total quality management just like William E. Deming.
- Total quality management originated in the industrial sector of Japan. Since that time the concept has been developed and can be used for almost all types of organizations such as schools, motorway maintenance, hotel management and churches.
- Nowadays, Total Quality Management is also used within the e-business sector and it perceives quality management entirely from the point of view of the customer. The objective of total quality management is doing things right the first time over and over again.
- Total Quality Management can be set up separately for an organization as well as for a set of standards that must be followed- for instance the International Organization for Standardization (ISO) in the ISO 9000 series. Total Quality Management uses strategy, data and communication channels to integrate the required quality principles into the organization's activities and culture.
- Total Quality Management has a number of basic principles which can be converted to the figure below.

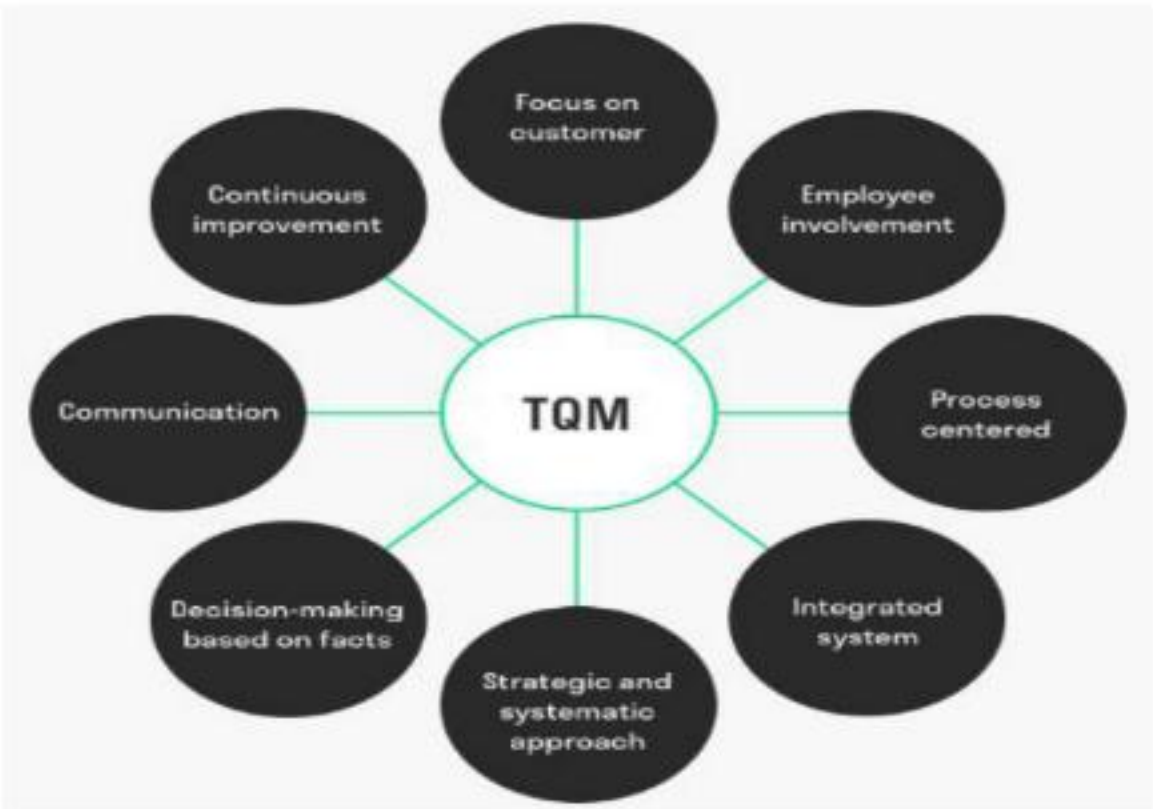


Fig - Total Quality Management

- **Focus on customer**

  When using total quality management it is of crucial importance to remember that only customers determine the level of quality. Whatever efforts are made with respect to training employees or improving processes, only customers determine, for example through evaluation or satisfaction measurement, whether your efforts have contributed to the continuous improvement of product quality and services.

- **Employee involvement**

  Employees are an organization's internal customers. Employee involvement in the development of products or services of an organization largely determines the quality of these products or services. Ensure that you have created a culture in which employees feel they are involved with the organization and its products and services.

- **Process centered**

  Process thinking and process handling are a fundamental part of total quality management. Processes are the guiding principle and people support these processes based on basis objectives that are linked to the mission, vision and strategy.

- **Integrated system**

  Following principle Process centred, it is important to have an integrated organization system that can be modelled for example ISO 9000 or a company quality system for the understanding and handling of the quality of the products or services of an organization.

- **Strategic and systematic approach**

  A strategic plan must embrace the integration and quality development and the development or services of an organization.

- **Decision-making based on facts**

  Decision-making within the organization must only be based on facts and not on opinions (emotions and personal interests). Data should support this decision-making process.

- **Communication**

  A communication strategy must be formulated in such a way that it is in line with the mission, vision and objectives of the organization. This strategy

comprises the stakeholders, the level within the organization, the communications channels, the measurability of effectiveness, timeliness, etc.

- **Continuous improvement**

  By using the right measuring tools and innovative and creative thinking, continuous improvement proposals will be initiated and implemented so that the organization can develop into a higher level of quality.

2. Product Quality Metrics

The customer problems metrics can be regarded as an intermediate measurement between defects measurement and customer satisfaction.

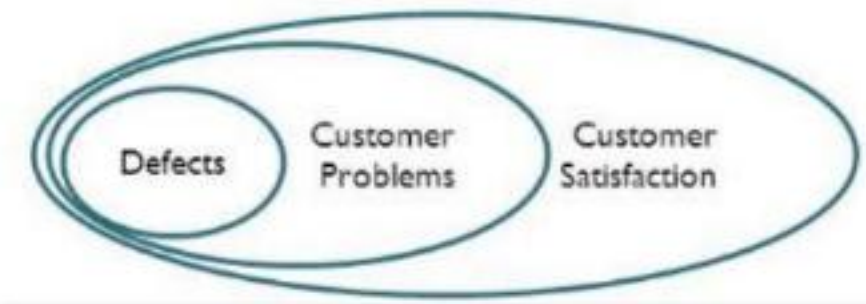


Fig – Representation of Product Quality Metrics

The metrics we discuss here cover both levels:

- Mean time to failure
- Defect density
- Customer problems
- Customer satisfaction.

- Intrinsic product quality is usually measured by the number of "bugs" (functional defects) in the software or by how long the software can run before encountering a "crash." In operational definitions, the two metrics are defect density (rate) and mean time to failure (MTTF).
- The MTTF metric is most often used with safety-critical systems such as the airline traffic control systems, avionics, and weapons. For instance, the U.S. government mandates that its air traffic control system cannot be unavailable for more than three seconds per year. In civilian airliners, the probability of certain catastrophic failures must be no worse than 10^{-9} per hour. The defect density metric, in contrast, is used in many commercial software systems.
- The two metrics are correlated but are different enough to merit close attention. First, one measures the time between failures, the other measures the defects relative to the software size (lines of code, function points, etc.).

Second, although it is difficult to separate defects and failures in actual measurements and data tracking, failures and defects (or faults) have different meanings. According to the IEEE/ American National Standards Institute (ANSI) standard (982.2):

- An error is a human mistake that results in incorrect software.
- The resulting fault is an accidental condition that causes a unit of the system to fail to function as required.
- A defect is an anomaly in a product.
- A failure occurs when a functional unit of a software-related system can no longer perform its required function or cannot perform it within specified limits.
- From these definitions, the difference between a fault and a defect is unclear. For practical purposes, there is no difference between the two terms. Indeed, in many development organizations the two terms are used synonymously.

3. In process Quality Metrics

- Because our goal is to understand the programming process and to learn to engineer quality into the process, in-process quality metrics play an important role. In-process quality metrics are less formally defined than end-product metrics, and their practices vary greatly among software developers.
- On the one hand, in-process quality metrics simply means tracking defect arrival during formal machine testing for some organizations. On the other hand, some software organizations with well-established software metrics programs cover various parameters in each phase of the development cycle.

- **Defect Density During Machine Testing**

- Defect rate during formal machine testing is usually positively correlated with the defect rate in the field. Higher defect rates found during testing is an indicator that the software has experienced higher error injection during its development process, unless the higher testing defect rate is due to an extraordinary testing effort for example, additional testing or a new testing approach that was deemed more effective in detecting defects.
- The rationale for the positive correlation is simple: Software defect density never follows the uniform distribution. If a piece of code or a product has higher testing defects, it is a result of more effective testing or it is because of higher latent defects in the code.

- **Defect Arrival Pattern during Machine Testing:-**

- The pattern of defect arrivals studies number of defect arrives during testing of software. Defect can occur during and after the software development process and it may have pattern of defect.

- It can be calculated in Mean Time to Repair (MTTR), Mean time Between Failure (MTBF), Mean Time to Failure (MTTF) etc

- When we talk about the defect arrival pattern during testing, there are actually three slightly different metrics, which should be looked at simultaneously:

- The defect arrivals (defects reported) during the testing phase by time interval (e.g., week). These are the raw number of arrivals, not all of which are valid defects.

- The pattern of valid defect arrivals—when problem determination is done on the reported problems. This is the true defect pattern.

- The pattern of defect backlog overtime. This metric is needed because development organizations cannot investigate and fix all reported problems immediately. This metric is a workload statement as well as a quality statement.

- **Phase-Based Defect Removal Pattern**

- The phase-based defect removal pattern is an extension of the test defect density metric. In addition to testing, it requires the tracking of defects at all phases of the development cycle, including the design reviews, code inspections, and formal verifications before testing.

- Because a large percentage of programming defects is related to design problems, conducting formal reviews or functional verifications to enhance the defect removal capability of the process at the front end reduces error injection. The pattern of phase-based defect removal reflects the overall defect removal ability of the development process.

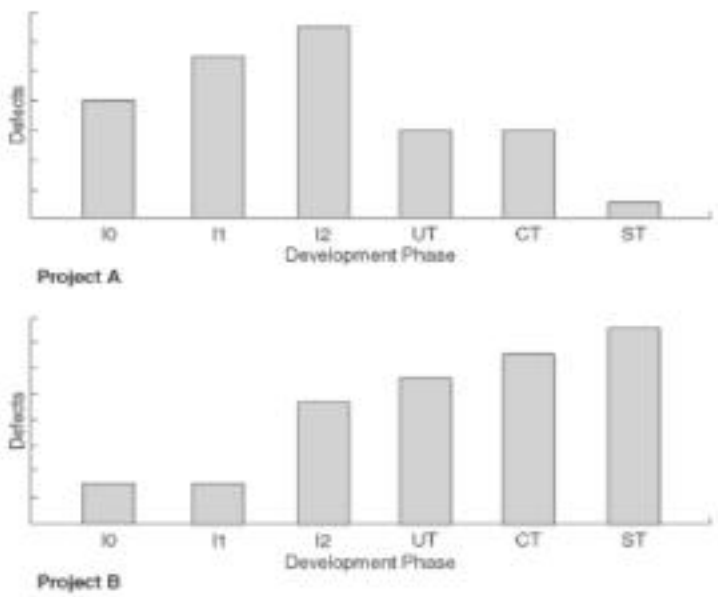


Fig - Defect Removal by Phase for Two Products

With regard to the metrics for the design and coding phases, in addition to defect rates, many development organizations use metrics such as inspection coverage and inspection effort for in-process quality management.

- **Defect Removal Effectiveness**
- Defect removal effectiveness can be defined as follows:
- $$DRE = \frac{\text{Defects removed during a development phase}}{\text{Defects latent in the product}} \times 100\%$$
- Because the total number of latent defects in the product at any given phase is not known, the denominator of the metric can only be approximated. It is usually estimated by:
- Defects removed during the phase + defects found later

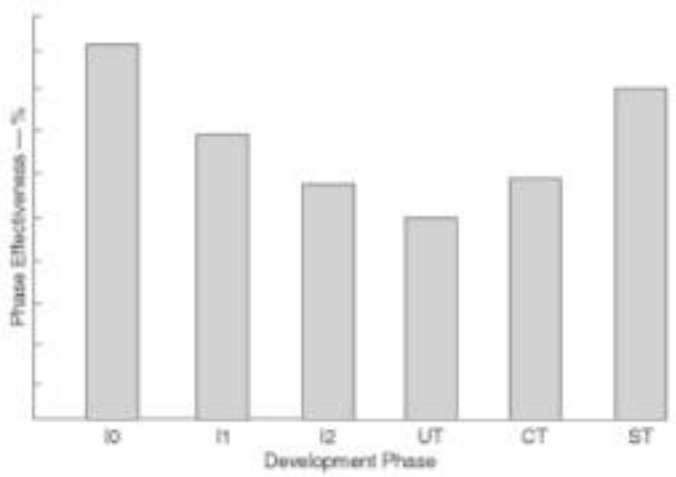


Fig - Phase Effectiveness of a Software Project

- The metric can be calculated for the entire development process, for the front end (before code integration), and for each phase. It is called early defect

removal and phase effectiveness when used for the front end and for specific phases, respectively. The higher the value of the metric, the more effective the development process and the fewer the defects escape to the next phase or to the field. This metric is a key concept of the defect removal model for software development.

4. Software maintenance

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

- **Market Conditions** - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
- **Client Requirements** - Over the time, customer may ask for new features or functions in the software.
- **Host Modifications** - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.
- **Maintenance Activities** - IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.

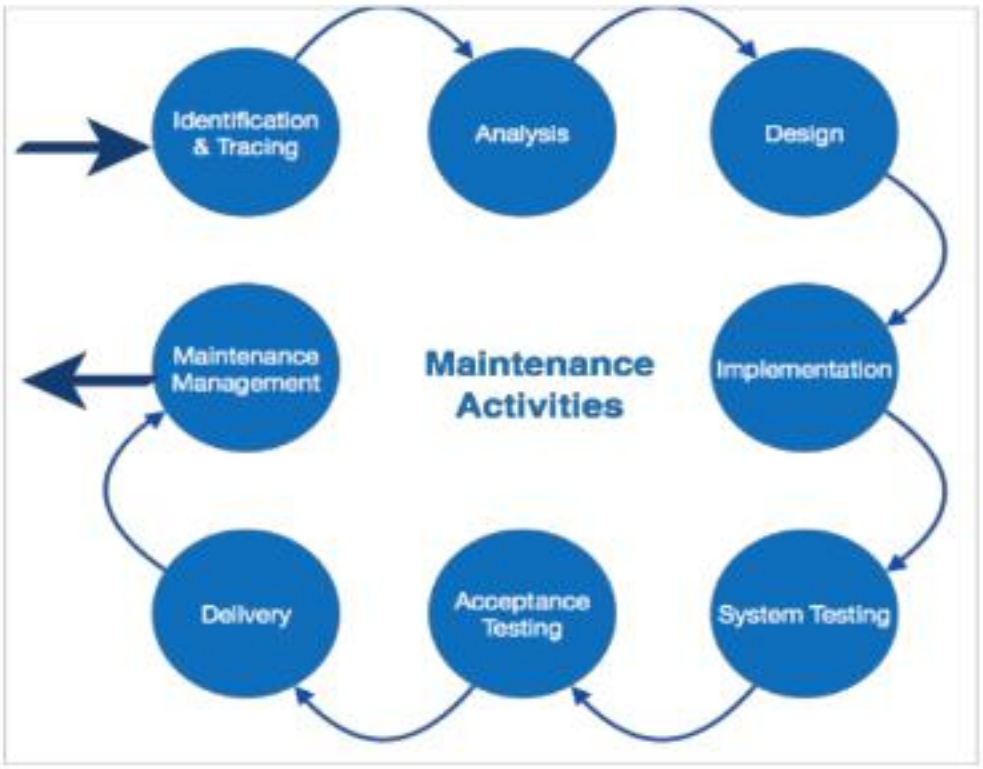


Fig – Representation of maintenance activities

These activities go hand-in-hand with each of the following phase:

- **Identification & Tracing** - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages. Here, the maintenance type is classified also.
- **Analysis** - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.
- **Design** - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.
- **Implementation** - The new modules are coded with the help of structured design created in the design step. Every programmer is expected to do unit testing in parallel.
- **System Testing** - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.
- **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.
- **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

Training facility is provided if required, in addition to the hard copy of user manual.

- **Maintenance management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

5. Ishikawa's 7 basic tools

- The Seven Basic Tools of Quality (also known as 7 QC Tools) originated in Japan when the country was undergoing major quality revolution and had become a mandatory topic as part of Japanese's industrial training program.
- These tools which comprised of simple graphical and statistical techniques were helpful in solving critical quality related issues. These tools were often referred as Seven Basics Tools of Quality because these tools could be implemented by any person with very basic training in statistics and were simple to apply to solve quality-related complex issues.
- 7 QC tools can be applied across any industry starting from product development phase till delivery. 7QC tools even today owns the same popularity and is extensively used in various phases of Six Sigma (DMAIC or DMADV), in continuous improvement process (PDCA cycle) and Lean management.

5.1 Checklists

- A check sheet can be metrics, structured table or form for collecting data and analysing them. When the information collected is quantitative in nature, the check sheet can also be called as tally sheet.
- The very purpose of checklist is to list down the important checkpoints or events in a tabular/metrics format and keep on updating or marking the status on their occurrence which helps in understanding the progress, defect patterns and even causes for defects.

Defect Types ? (Major/ Minor)	Defects in Supplied Items							Total Count
	Sun	Mon	Tue	Wed	Thu	Fri	Sat	
Rusted Items		4	2		2	1		9
Items with Scratch	1							1
Dirty		1		3		2		6
Broken/ Cracks			2				1	3
Main Body Dent					3			3
Missing Components		2		2			1	5
Labelling Error					1	3		4
Damage in Packaging			2					2
Wrong Item Issued					2		1	3
Film on Parts			4					4
Voids in Casting	1					1	2	4
Incorrect Dimensions			2	1	2			5
Failed to pass the quality test		2				1		3
Total Count	2	9	12	6	10	8	5	52

Fig - Table (Check Sheet: Defect types with their occurrence on day of the week)

5.2 Pareto diagrams

- The Pareto principle also known as the 80-20 rule derived from the Italian economist Wilfred Pareto's observations about the factor of sparsity which states that 80% of the effects are coming from 20% of the causes. This also holds true to the business' rule of thumb that 80% of the sales are contributed by only 20% of the customers.

- As a tool in Six Sigma, Pareto is part of the quality control tools that are derived from historical data in order to come up with efficient and most appropriate actions to address most common and impacting failures. With the use of Pareto, scarce resources are efficiently allocated.

- The principle dictates that 80% of the failures are coming from 20% of the causes. It is important to note that this tool wholly bases recommended actions on present data. It does not consider probable increase or decrease and projected movements of any one contributing factor.

- To further illustrate, the Pareto chart has been designed as a visual representation of the vital few against the trivial many. With the help of the chart, it is easy to identify the causes of most of the problems. Below is an example of a Pareto chart:

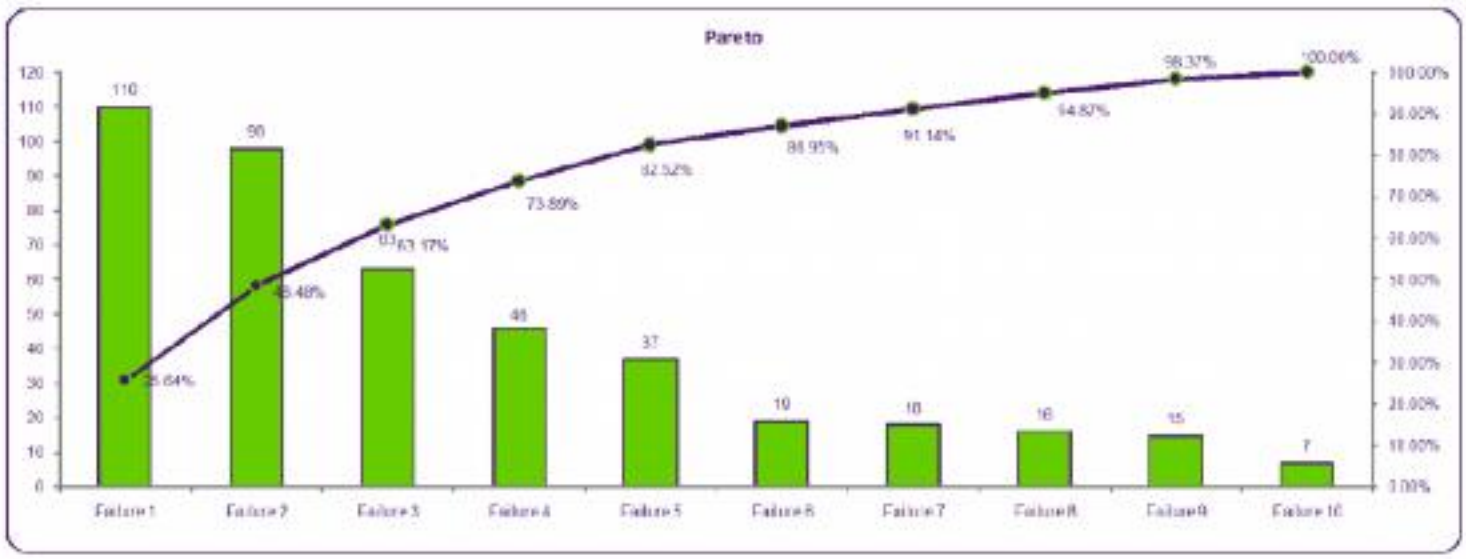


Fig – Representation of Pareto diagrams

5.3 Histogram

- Most of us (Quality Professionals) deal with data in our day to day operations. Often, we come across situations where we have to analyze data and interpret to management about the behavior of certain groups of items. To present the data in a simple and catchy manner, we use graphical tools sometimes.

- There are a lot of graphs like Pie chart, Bar graph, Histogram etc. which we use for different purposes. Histogram is one such tool used to represent the frequency distribution of data falling into different groups of a population or sample.

- It is a graph that represents the frequency (count) of items falling into different categories of a given population or sample. It looks similar to a bar graph. It has vertical bars with different heights. Each group has a corresponding bar representing it in the graph.

A histogram looks like below:

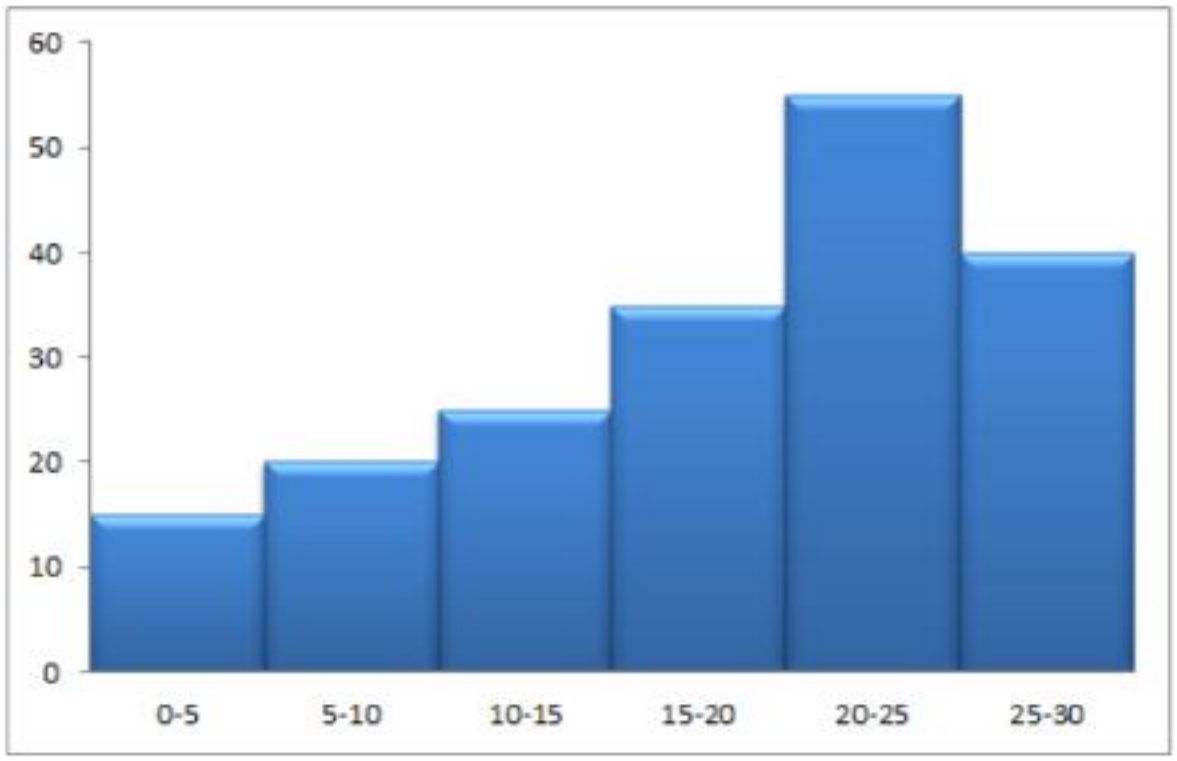


Fig - Histogram

Characteristics of a Histogram

1. Histogram is used to represent categorization of **Continuous data**
2. There is **no gap between the bars** unlike Bar graph, to signify that the data is continuous
3. The **width of the groups is equal**.

$$Width = \frac{Total\ range\ of\ population}{No.\ of\ categories}$$

Example:

Assume the management of an amusement park wants to study the number of visitors in different age group. Based on the visitors they want to include o modify the different 'Rides' or 'Theme shows' for the maximum contributing group. They plotted a histogram to identify the frequency of visitors falling into each of the below age group.

5.4 Control chart

- Control chart is a statistical tool used to monitor whether a process is in control or not. It is a time series graph with the process mean at center and the control limits on both sides of it.

- The values lying outside the control limits show that the process is out of control. There are several other criteria with which the out-of-control nature of the process is detected.

- The variation of the process can be attributed to two causes: Common Cause and Special cause. Common cause variation is the variation that is inherent of the process and no eternal factor can be associated to it. It is indeed very difficult to reduce this type of variation.

- Special cause variation is any variation that is caused by factors that are not a part of the process or system. These are normally outliers, and can be easily detected.

- The use of a control chart helps one to distinguish between a common cause and a special cause. Identifying the type of variation helps in setting up the right improvement path. Therefore the use of control chart becomes very vital in Process control.

- There are different types of Control charts based on the data that we use. One needs to understand these types clearly to use the right chart for the data.

Types of Control Chart

- Before understanding the types, one should know about the concept of 'Rational sub grouping'. Rational sub grouping is the process of selecting samples at various points of time to study the variation. An important point is that all the samples in a subgroup should be selected at the same time.

- There should be enough time-gap between the selection of subsequent subgroups. While plotting a control chart, the Mean and Standard Deviation of each subgroup are also calculated along with the mean of the overall observations. This is done to study whether there is variation with-In subgroups or between subgroups.

5.5 Cause Effect diagram

- Cause–and–effect diagram introduced by Kaoru Ishikawa helps in identifying the various causes (or factors) leading to an effect (or problem) and also helps in deriving meaningful relationship between them.

- The very purpose of this diagram is to identify all root causes behind a problem. Once a quality related problem is defined, the factors leading to the causal of the problem are identified.

- We further keep identifying the sub factors leading to the causal of identified factors till we are able to identify the root cause of the problem. As a result we get a diagram with branches and sub branches of causal factors resembling to a fish bone diagram.

- In manufacturing industry, to identify the source of variation the causes are usually grouped into below major categories:
- People
- Methods
- Machines
- Material
- Measurements
- Environment

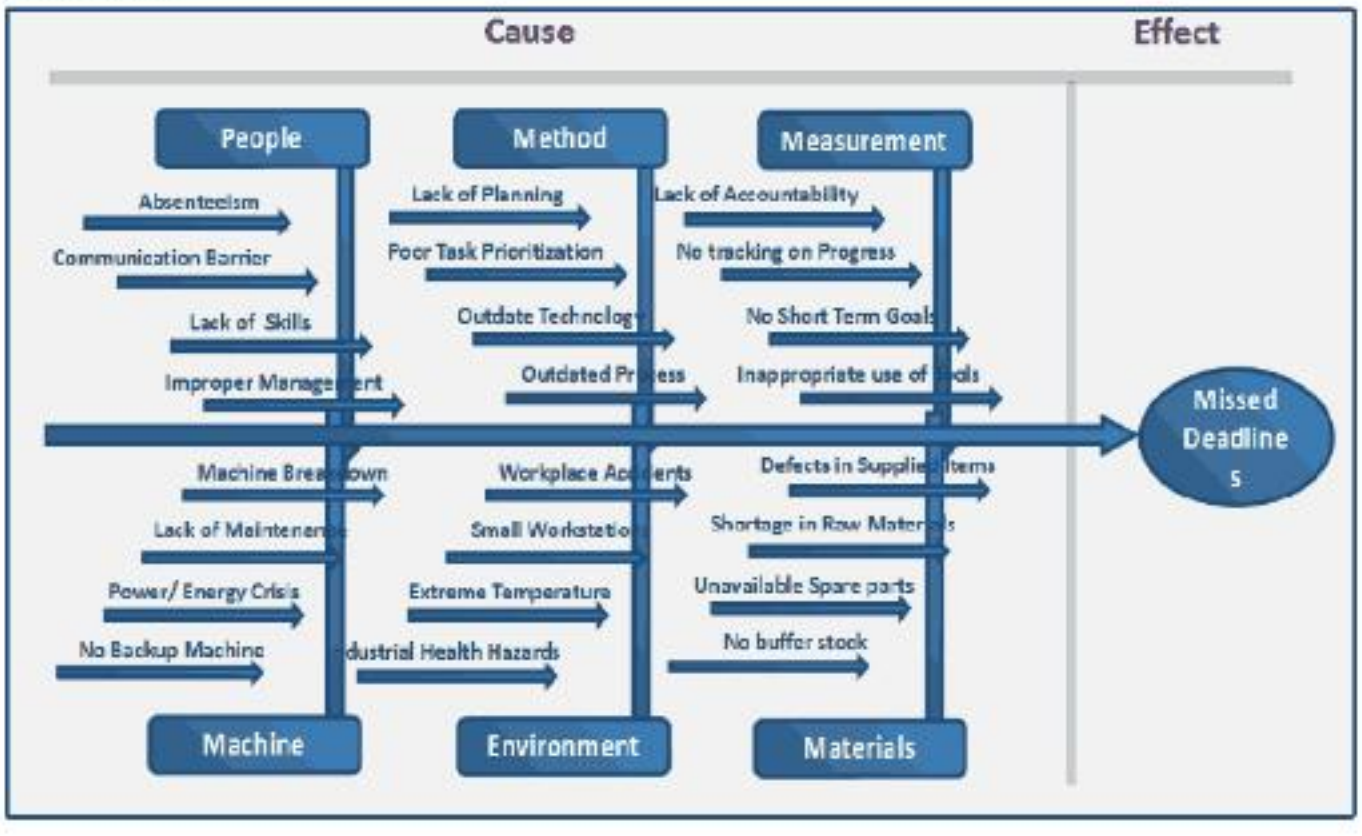


Fig - Cause Effect diagram

6. Defect Removal Effectiveness and Process Maturity Level

The defect removal effectiveness for organizations at different levels of the development process capability maturity model (CMM):

- Level 1: 85%
- Level 2: 89%

- Level 3: 91%
- Level 4: 93%
- Level 5: 95%

These values can be used as comparison baselines for organizations to evaluate their relative capability with regard to this important parameter.

In a discussion on quantitative process management (a process area for Capability Maturity Model Integration, CMMI, level 4) and process capability baselines, Curtis (2002) shows the estimated baselines for defect removal effectiveness by phase of defect insertion.

The cumulative percentages of defects removed up through acceptance test (the last phase before the product is shipped) by phase insertion, for CMMI level 4, are shown in Table 6.4. Based on historical and recent data from three software engineering organizations at General Dynamics Decision Systems, Diaz and King (2002) report that the phase containment effectiveness by CMM level as follows:

- Level 2: 25.5%
- Level 3: 41.5%
- Level 4: 62.3%
- Level 5: 87.3%

References:

1. Naresh Chauhan, "Software Testing Principles and Practices ", OXFORD, ISBN-10: 0198061846. ISBN-13: 9780198061847.

2. Stephen Kan, "Metrics and Models in Software Quality Engineering", Pearson, ISBN-10: 0133988082; ISBN-13: 978-0133988086.

3. Srinivasan Desikan, Gopalswamy Ramesh, "Software Testing Principles and Practices", Pearson, ISBN-10: 817758121X.