Learning Material

(Lesson-1)

Adv. Software Engineering

By

Prof. Dr. Kashif Laeeq

Software Engineering

A software is a computer programs along with the associated documents and the configuration data that make these programs operate correctly.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called software product.

A program is a set of instructions (written in form of human-readable code) that performs a specific task.

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

Software engineering is an engineering discipline that's applied to the development of software in a systematic approach (called a software process).

It's the application of theories, methods, and tools to design build a software that meets the specifications efficiently, cost-effectively, and ensuring quality.

It's not only concerned with the technical process of building a software, it also includes activities to manage the project, develop tools, methods and theories that support the software production.

Different methods and techniques of software engineering are appropriate for different types of systems. For example, games should be developed using series of prototypes, while critical control systems require a complete analyzable specification to be developed.

Types of Software Systems

There are many different types of software systems from simple to complex systems. These systems may be developed for a particular customer, like systems to support a particular business process, or developed for a general purpose, like any software for our computers such as word processors.

Many systems are now being built with a generic product as base, which is then adapted to suit the requirements of a customer such as SAP system.

Successful Software System

A good software should deliver the main required functionality.

Other set of attributes — called quality or non-functional — should be also delivered. Examples of these attributes are, the software is written in a way that can be adapted to changes, response time, performance (less use of resources such as memory and processor time), usable; acceptable for the type of the user it's built for, reliable, secure, safe, …etc.

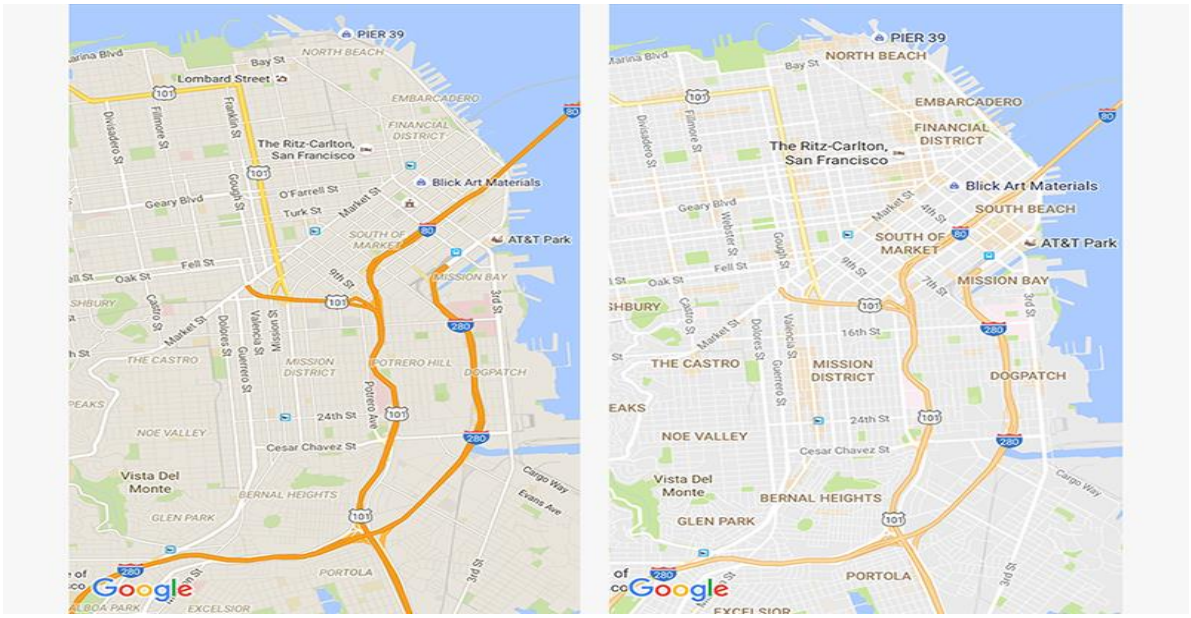
Successful Software Product

Every software engineer wants to build the products that truly change the world we live in. Are there hidden secrets or learnings that we can tap into to build into our own process?

Is there something about how those products were built that all software engineers and product managers can learn from? In this lecture we will go through the Google Maps, one of the most successful software products and, analyze what we can learn from it to be an even better software engineer.

Google Maps

Google Maps launched in 2005, and since then, has totally transformed the way we experience the world around us.



Approximately, 154 million Americans don't just go on the platform every month to get from A-B, this product must be offering more value than simple navigation.

Why it's good

Google Maps is designed not just to get you from A to B, but to make life easy. It's a navigational app, and Google does this basic function better than its competitors. It helps people find local businesses, to learn from other people's experiences, and to make more informed decisions from them. In short, Google Maps is a successful product because it extends so far beyond what it was originally designed to do.

Where other tools were designing around navigation, Google thought about how to join up the entire experience of a journey with community-driven touchpoints. That's why now when you navigate around your surroundings with Google Maps, you don't just see how to get somewhere, you also see reviews, recommendations and can even create lists and contribute your knowledge back to the community. In essence, Google Maps is designed around a self-sustaining ecosystem, in which the more users put into it, the more they get out. By creating a product that is truly user-centric, Google stopped thinking about maps as a means of getting around and started thinking about how spaces can create meaningful experiences for people. It's transcended its original

function to become the default way that we find information about our surroundings so that we can make better choices for our lives. Now Google Street View enriches the way we experience spaces around us on a deeper level than ever before. We can walk streets on the other side of the planet, at the click of a button. Thanks to this crazy ambitious feature, the world has never been so small.

What Software Engineers can learn from Google Maps?

Build a great product

You've probably heard this a million times, but its importance can't be overstated. Compared to other maps, this just happens to be the best product on the market - hands down. There's no use building tons of extraneous (and ambitious) features if you aren't already offering your core service better than anyone else. So start there, always. As software engineer - especially on large products - it's easy to fall victim to soloed thinking. You can get sucked into fixating on your single feature, and forget about how it connects to the overall experience that a product has the potential to provide. In the case of Google Maps, they found ways to connect the dots between dozens of features such as reviews, business listings, lists, check-ins, etc.

The result?

They've created a product in which many features work together to create something exceptional. This is a tactic that every product, and every product manager, should strive to achieve.

Don't be scared to do things that are seemingly impossible

Truly ground-breaking products are ones that fundamentally rethink the way we do things. They take the rule book, rip it up, and rewrite it in a new way. Street View is a perfect example of this. It would have taken a truly creative and visionary set of minds to pitch this idea - and an even more visionary team to trust that idea, even though it might have seemed crazy. What's even crazier is that Street Views probably hasn't made any money from Street View. It's purely a strategic (and extremely long-term) investment to dominate the market with a tool people didn't even know they wanted, but now couldn't live without.

So don't be afraid to take your product in directions that might go against the grain. You could just end up changing the world.

Computer Science Vs Software Engineering

Computer science focuses on the theory and fundamentals, like algorithms, programming languages, theories of computing, artificial intelligence, and hardware design, while software engineering is concerned with the activities of developing and managing a software.

Software Engineers

The job of a software engineer is difficult. It has to balance between different people involved, such as:

- Dealing with users: User don't know what to expect exactly from the software. The concern is always about the ease of use and response time.
- Dealing with technical people: Developers are more technically inclined people so they think more of database terms, functionality, etc.
- Dealing with management: They are concerned with return on their investment, and meeting the schedule.

For success in large software developments, it is important to follow an engineering approach, consisting of a well-defined process. That's what we're going to explore next in the "Software Processes".

-----END-----

Learning Material

(Lesson-2)

Adv. Software Engineering

By

Dr. Kashif Laeeq

Software Process

A software process (also known as software methodology) is a set of related activities that leads to the production of the software. These activities may involve the development of the software from the scratch, or, modifying an existing system.

Any software process must include the following four activities:

1. Software specification (or requirements engineering): Define the main functionalities of the software and the constrains around them.
2. Software design and implementation: The software is to be designed and programmed.
3. Software verification and validation: The software must conforms to its specification and meets the customer needs.
4. Software evolution (software maintenance): The software is being modified to meet customer and market requirements changes.

In practice, they include sub-activities such as requirements validation, architectural design, unit testing …etc.

There are also supporting activities such as configuration and change management, quality assurance, project management, user experience.

Along with other activities aim to improve the above activities by introducing new techniques, tools, following the best practice, process standardization (so the diversity of software processes is reduced), etc.

When we talk about a process, we usually talk about the activities in it. However, a process also includes the process description, which includes:

1. Products: The outcomes of an activity. For example, the outcome of architectural design maybe a model for the software architecture.
2. Roles: The responsibilities of the people involved in the process. For example, the project manager, programmer, etc.
3. Pre and post conditions: The conditions that must be true before and after an activity. For example, the pre-condition of the architectural design is the requirements have been

approved by the customer, while the post condition is the diagrams describing the architectural have been reviewed.

Software process is complex, it relies on making decisions. There's no ideal process and most organizations have developed their own software process.

For example, an organization works on critical systems has a very structured process, while with business systems, with rapidly changing requirements, a less formal, flexible process is likely to be more effective.

Software Process Models

A software process model is a simplified representation of a software process. Each model represents a process from a specific perspective.

We're going to take a quick glance about very general process models. These generic models are abstractions of the process that can be used to explain different approaches to the software development. They can be adapted and extended to create more specific processes.

Some methodologies are sometimes known as software development life cycle (SDLC) methodologies, though this term could also be used more generally to refer to any methodology.

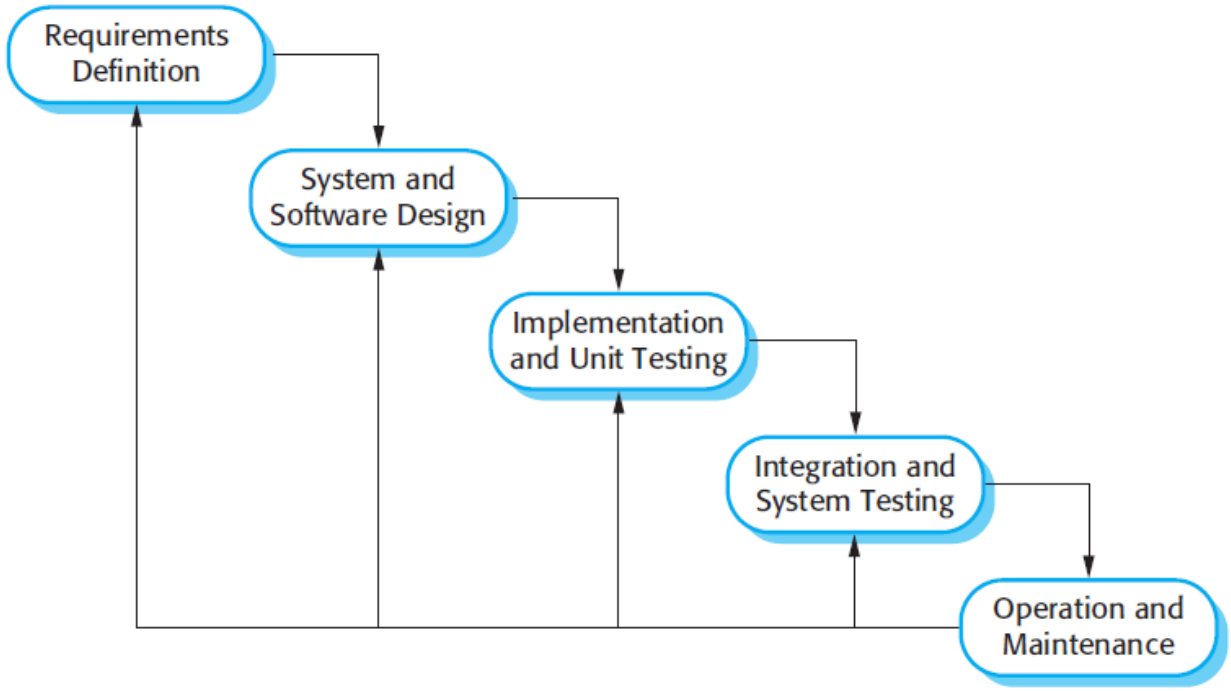
Waterfall Model

The waterfall model is a sequential approach, where each fundamental activity of a process represented as a separate phase, arranged in linear order.

In the waterfall model, you must plan and schedule all of the activities before starting working on them (plan-driven process).

Plan-driven process is a process where all the activities are planned first, and the progress is measured against the plan. While the agile process, planning is incremental and it's easier to change the process to reflect requirement changes.

The phases of the waterfall model are: Requirements, Design, Implementation, Testing, and Maintenance.



The Waterfall Model

The Nature of Waterfall Phases

In principle, the result of each phase is one or more documents that should be approved and the next phase shouldn't be started until the previous phase has completely been finished.

In practice, however, these phases overlap and feed information to each other. For example, during design, problems with requirements can be identified, and during coding, some of the design problems can be found, etc.

The software process therefore is not a simple linear but involves feedback from one phase to another. So, documents produced in each phase may then have to be modified to reflect the changes made.

When to Use?

In principle, the waterfall model should only be applied when requirements are well understood and unlikely to change radically during development as this model has a relatively rigid structure which makes it relatively hard to accommodate change when the process in underway.

Prototyping

A prototype is a version of a system or part of the system that's developed quickly to check the customer's requirements or feasibility of some design decisions.

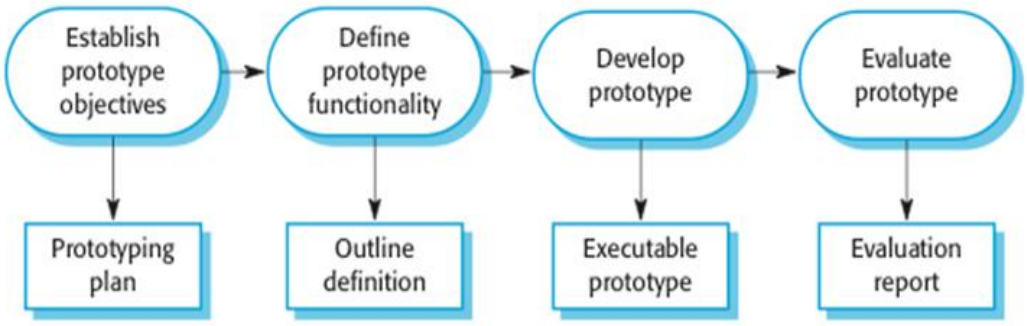
So, a prototype is useful when a customer or developer is not sure of the requirements, or of algorithms, efficiency, business rules, response time, etc.

In prototyping, the client is involved throughout the development process, which increases the likelihood of client acceptance of the final implementation.

While some prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to working system.

A software prototype can be used:

- In the requirements engineering, a prototype can help with the elicitation and validation of system requirements.
  It allows the users to experiment with the system, and so, refine the requirements. They may get new ideas for requirements, and find areas of strength and weakness in the software.
  Furthermore, as the prototype is developed, it may reveal errors and in the requirements. The specification maybe then modified to reflect the changes.
- In the system design, a prototype can help to carry out deign experiments to check the feasibility of a proposed design.
  For example, a database design may be prototype-d and tested to check it supports efficient data access for the most common user queries.



The process of prototype development

The phases of a prototype are:

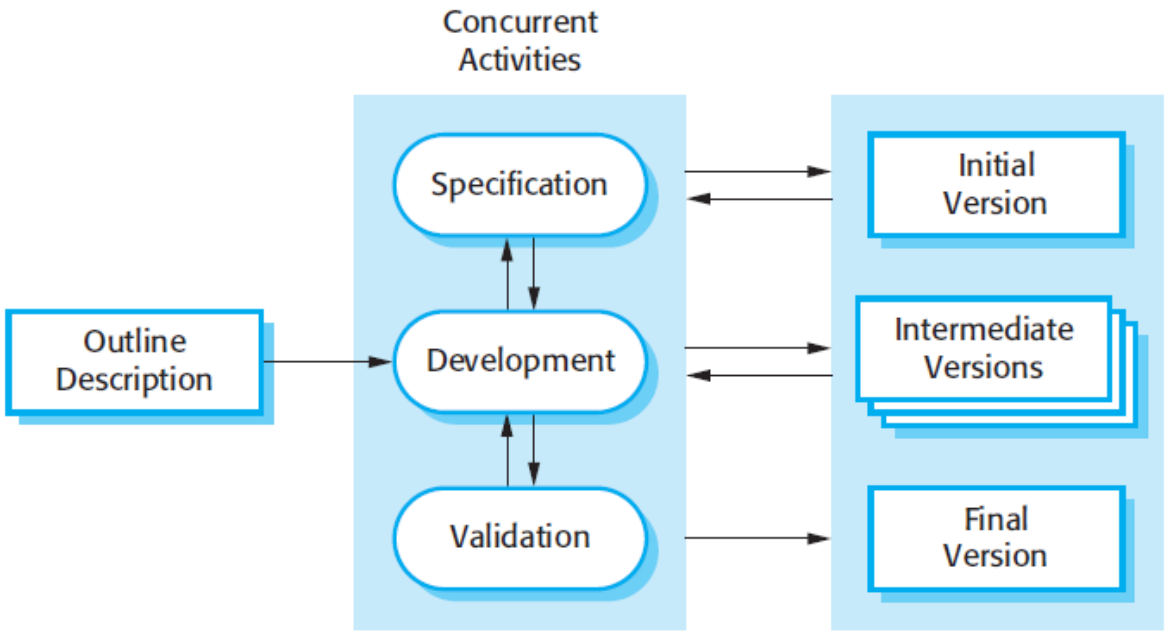
1. **Establish objectives:** The objectives of the prototype should be made explicit from the start of the process. Is it to validate system requirements, or demonstrate feasibility, etc.
2. **Define prototype functionality:** Decide what are the inputs and the expected output from a prototype. To reduce the prototyping costs and accelerate the delivery schedule, you may ignore some functionality, such as response time and memory utilization unless they are relevant to the objective of the prototype.
3. **Develop the prototype:** The initial prototype is developed that includes only user interfaces.
4. **Evaluate the prototype:** Once the users are trained to use the prototype, they then discover requirements errors. Using the feedback both the specifications and the prototype can be improved. If changes are introduced, then a repeat of steps 3 and 4 may be needed.

Prototyping is not a standalone, complete development methodology, but rather an approach to be used in the context of a full methodology (such as incremental, spiral, etc).

Incremental Development

Incremental development is based on the idea of developing an initial implementation, exposing this to user feedback, and evolving it through several versions until an acceptable system has been developed.

The activities of a process are not separated but interleaved with feedback involved across those activities.



The Incremental Development Model

Each system increment reflects a piece of the functionality that is needed by the customer. Generally, the early increments of the system should include the most important or most urgently required functionality.

This means that the customer can evaluate the system at early stage in the development to see if it delivers what's required. If not, then only the current increment has to be changed and, possibly, new functionality defined for later increments.

Incremental Vs Waterfall Model

Incremental software development is better than a waterfall approach for most business, e-commerce, and personal systems.

By developing the software incrementally, it is cheaper and easier to make changes in the software as it is being developed.

Compared to the waterfall model, incremental development has three important benefits:

- The cost of accommodating changing customer requirements is reduced. The amount of analysis and documentation that has to be redone is much less than that's required with waterfall model.
- It's easier to get customer feedback on the work done during development than when the system is fully developed, tested, and delivered.
- More rapid delivery of useful software is possible even if all the functionality hasn't been included. Customers are able to use and gain value from the software earlier than it's possible with the waterfall model.

It can be a plan-driven or agile, or both

Incremental development is one of the most common approaches. This approach can be either a plan-driven or agile, or both.

In a plan-driven approach, the system increments are identified in advance, but, in the agile approach, only the early increments are identified and the development of later increments depends on the progress and customer priorities.

It's not a problem-free

But, it's not a problem-free …

Some organizations have procedures that have evolved over the time, and can't follow informal iterative or agile process. For example, procedures to ensure that the software properly implements external regulations.

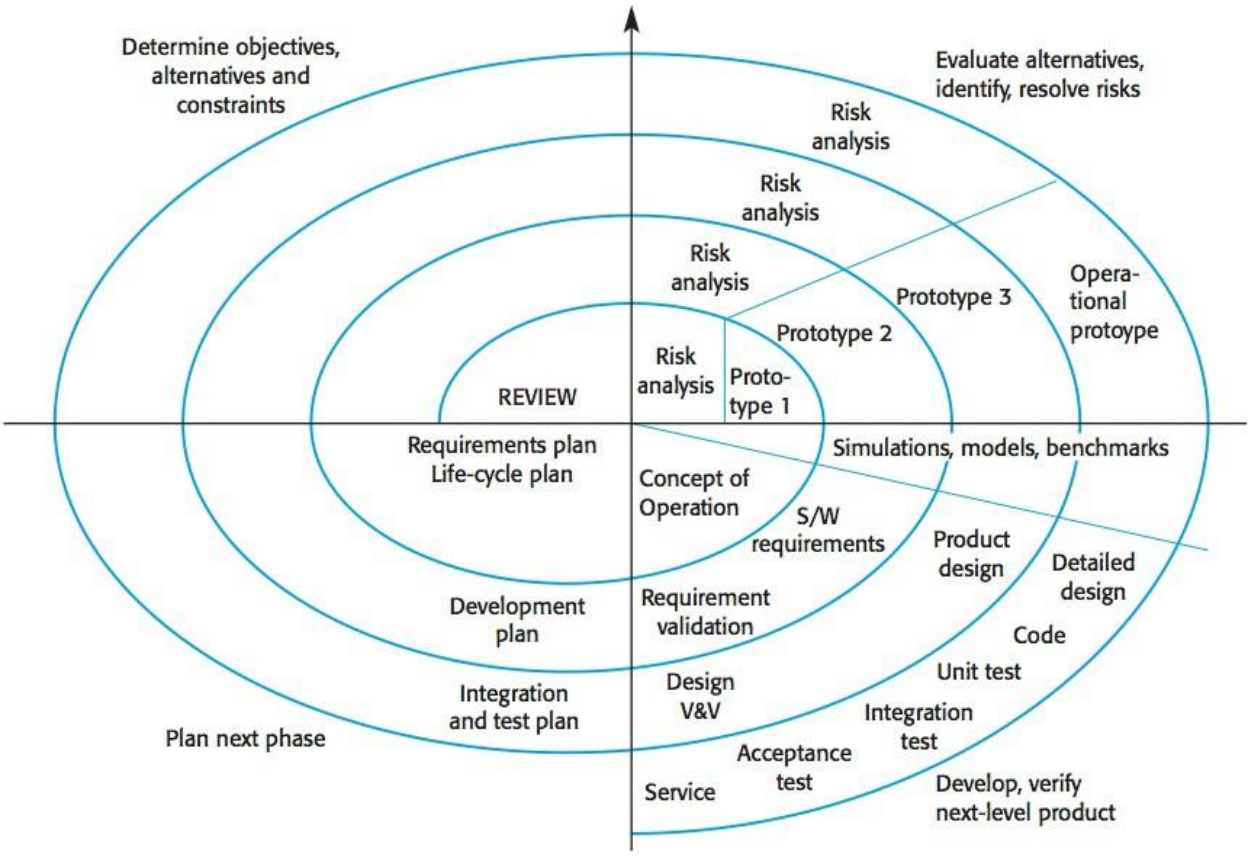
System structure tends to degrades as new increments are added and get corrupted as regular changes are incorporated. Even if time and money spent on refactoring to improve the software, further changes becomes more difficult and costly.

Spiral Model

The spiral model is a risk-driven where the process is represented as spiral rather than a sequence of activities.

It was designed to include the best features from the waterfall and prototyping models, and introduces a new component; risk-assessment.

Each loop (from review till service — see figure below) in the spiral represents a phase. Thus the first loop might be concerned with system feasibility, the next loop might be concerned with the requirements definition, the next loop with system design, and so on.



The spiral model

Each loop in the spiral is split into four sectors:

Objective setting: The objectives and risks for that phase of the project are defined.

Risk assessment and reduction: For each of the identified project risks, a detailed analysis is conducted, and steps are taken to reduce the risk. For example, if there's a risk that the requirements are inappropriate, a prototype may be developed.

Development and validation: After risk evaluation, a process model for the system is chosen. So if the risk is expected in the user interface then we must prototype the user interface. If the risk is in the development process itself then use the waterfall model.

Planning: The project is reviewed and a decision is made whether to continue with a further loop or not.

Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development. In practice, however, the model is rarely used.

Iterative Development

Iterative development model aims to develop a system through building small portions of all the features, across all components.

We build a product which meets the initial scope and release it quickly for customer feedback. An early version with limited features important to establish market and get customer feedback.

In each increment, a slice of system features is delivered, passing through the requirements till the deployment.

The phases of iterative development are:

1. **Inception:** The goal is to establish a business case for the system. We should identify all the external entities that will interact with the system, and define these interactions. Then, uses this information to assess the contribution that the system makes to the business. If the contribution is minor, then the project may be cancelled.

2. **Elaboration:** We develop an understanding of the problem domain and architecture framework, develop the project plan, and identify risks.
3. **Construction:** Incrementally fills-in the architecture with production-ready code produced from analysis, design, implementation, and testing of the requirements. The components of the system are dependent on each other and they're developed in parallel and integrated during this phase. On the completion of this phase, you should have a complete working software.
4. **Transition:** We deliver the system into the production operating environment.

Increment Vs Iterative

You might be asking about the difference between incremental, and iterative models.

Each increment in the incremental approach builds a complete feature of the software, while in iterative, it builds small portions of all the features.

-----END-----

Learning Material

(Lesson-3)

Adv. Software Engineering

By

Dr. Kashif Laeeq

Agile

- Agile development, is a main term for several iterative and incremental software development methodologies.
- Agile means able to move quickly and easily. In software engineering the term Agile means able to adopt the changes from requirements or technology-based.
- Used to software development that is characterized by the division of tasks into short phases of work and frequent reassessment and adaption of plans.
- Agile is nothing new it is a model that based on the best practices from previous models like waterfall, iterative and incremental etc.
- Agile is quite flexible, software team is free to take decision for the betterment of project.
- They are best suited for application where the requirements change rapidly during the development process.
- Agile is not a model, methodology or even framework… in fact, Agile is a set of values and principles.

Agility is flexibility, it is a state of dynamic, adapted to the specific circumstances.

The agile methods refers to a group of software development models based on the incremental and iterative approach, in which the increments are small and typically, new releases of the system are created and made available to customers every few weeks.

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

The principles of agile methods

They involve customers in the development process to propose requirements changes. They minimize documentation by using informal communications rather than formal meetings with written documents.

Agile Manifesto

Developing software better by valuing the item of left side of the list more than the item of the right side of list.

Value More	Value Less
Individuals and interactions	Processes and tools
Working software	Comprehensive documentation
Customer collaboration	Contract negotiation
Responding to change	Following a plan

12 Principles of Agile

Below are the guiding practices that support teams in implementing and executing with agility. The following 12 Principles are based on the Agile Manifesto.

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done–is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

There are a number of different agile methods available such as: Scrum, Crystal, Agile Modeling (AM), Extreme Programming (XP), etc.

Increment Vs Iterative Vs Agile

You might be asking about the difference between incremental, iterative and agile models.

Each increment in the incremental approach builds a complete feature of the software, while in iterative, it builds small portions of all the features.

An agile approach combines the incremental and iterative approach by building a small portion of each feature, one by one, and then both gradually adding features and increasing their completeness.

Six Sigma (only basics)

Six Sigma is a methodology for pursuing continuous improvement in customer satisfaction and profit. It is a management philosophy attempting to improve effectiveness and efficiency. Six Sigma is a highly disciplined process that helps us focus on developing and delivering near-perfect products and services.

Origin of Six Sigma

- Six Sigma originated at Motorola in the early 1980s, in response to achieving 10X reduction in product-failure levels in 5 years.
- Engineer Bill Smith invented Six Sigma, but died of a heart attack in the Motorola cafeteria in 1993, never knowing the scope of the craze and controversy he had touched off.
- Six Sigma is based on various quality management theories (e.g. Deming's 14 point for management, Juran's 10 steps on achieving quality).

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

The word *Sigma* is a statistical term that measures how far a given process deviates from perfection.

The central idea behind Six Sigma: If you can measure how many "defects" you have in a process, you can systematically figure out how to eliminate them and get as close to "zero defects" as possible and specifically it means a failure rate of 3.4 parts per million or 99.9997% perfect.

Key Concepts of Six Sigma

At its core, Six Sigma revolves around a few key concepts.

- **Critical to Quality** − Attributes most important to the customer.
- **Defect** − Failing to deliver what the customer wants.
- **Process Capability** − What your process can deliver.

- **Variation** − What the customer sees and feels.
- **Stable Operations** − Ensuring consistent, predictable processes to improve what the customer sees and feels.
- **Design for Six Sigma** − Designing to meet customer needs and process capability.

Our Customers Feel the Variance, Not the Mean. So Six Sigma focuses first on reducing process variation and then on improving the process capability.

Benefits of Six Sigma

Six Sigma offers six major benefits that attract companies −

- Generates sustained success
- Sets a performance goal for everyone
- Enhances value to customers
- Accelerates the rate of improvement
- Promotes learning and cross-pollination
- Executes strategic change

There are three key elements of Six Sigma Process Improvement −

- Customers
- Processes
- Employees

The Customers

Customers define quality. They expect performance, reliability, competitive prices, on-time delivery, service, clear and correct transaction processing and more. This means it is important to provide what the customers need to gain customer delight.

The Processes

Defining processes as well as defining their metrics and measures is the central aspect of Six Sigma. In a business, the quality should be looked from the customer's perspective and so we must look at a defined process from the outside-in.

By understanding the transaction lifecycle from the customer's needs and processes, we can discover what they are seeing and feeling. This gives a chance to identify weak areas with in a process and then we can improve them.

The Employees

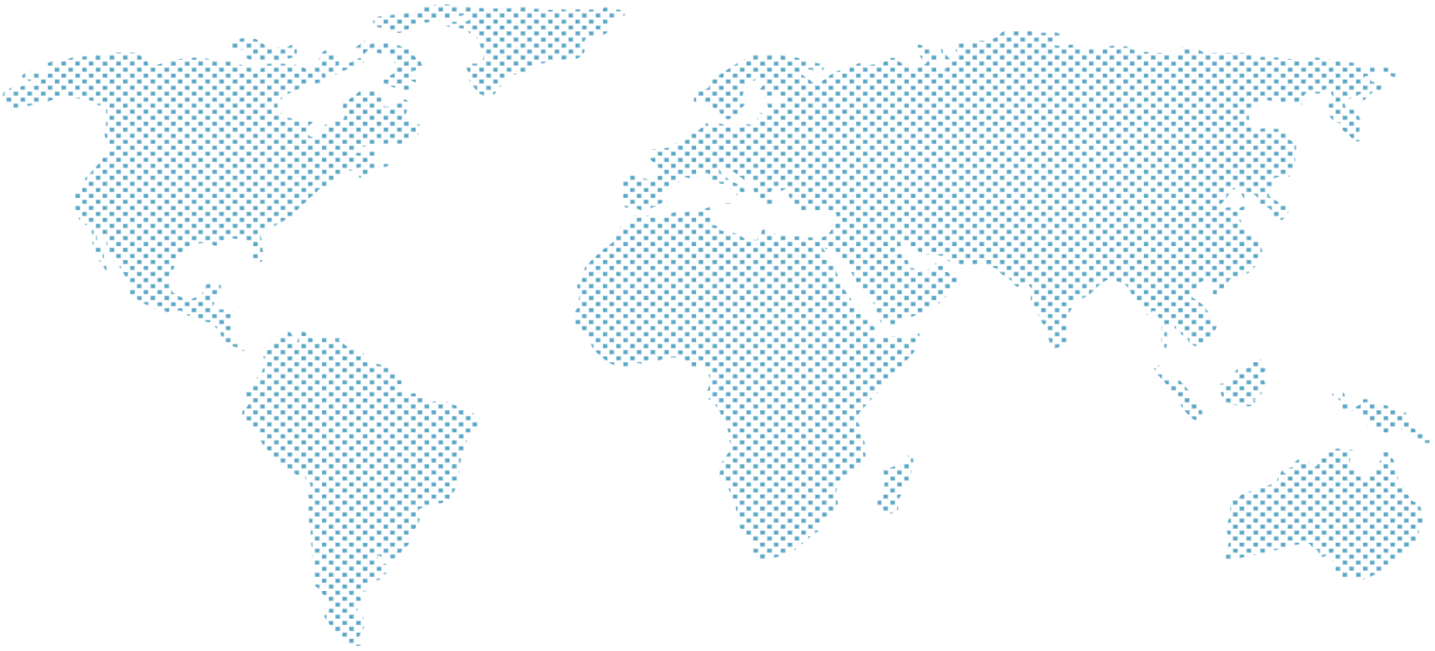
A company must involve all its employees in the Six Sigma program. Company must provide opportunities and incentives for employees to focus their talents and ability to satisfy customers.

It is important to Six Sigma that all the team members should have a well-defined role with measurable objectives.

-----END-----

ADVANCE SOFTWARE ENGINEERING

Lesson No.4



DR. KASHIF LAEEQ

PhD (CS), M.Phil. (CS), MCS (CS), M.Sc. (Math)
Member of IACSIT, IEEE, IEEEP, IJSER, ACM Research Group
Professor, Dept. of Computer Science
Federal Urdu University, Karachi

Software Testing

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.

Software Validation

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

- Validation ensures the product under development is as per the user requirements.
- Validation answers the question – "Are we developing the product which attempts all that user needs from this software?"
- Validation emphasizes on user requirements.

Software Verification

Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

- Verification ensures the product being developed is according to design specifications.
- Verification answers the question– "Are we developing this product by firmly following all design specifications?"
- Verifications concentrates on the design and system specifications.

Target of the test are -

- **Errors** - These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.

- **Fault** - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.
- **Failure** - failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.

Manual Vs Automated Testing

Testing can either be done manually or using an automated testing tool:

- **Manual** - This testing is performed without taking help of automated testing tools. The software tester prepares test cases for different sections and levels of the code, executes the tests and reports the result to the manager. Manual testing is time and resource consuming. The tester needs to confirm whether or not right test cases are used. Major portion of testing involves manual testing.

- **Automated** testing is a testing procedure done with aid of automated testing tools. The limitations with manual testing can be overcome using automated test tools.

A test needs to check if a webpage can be opened in Internet Explorer. This can be easily done with manual testing. But to check if the web-server can take the load of 1 million users, it is quite impossible to test manually.

There are software and hardware tools which helps tester in conducting load testing, stress testing, regression testing.

Testing Approaches

Tests can be conducted based on two approaches –

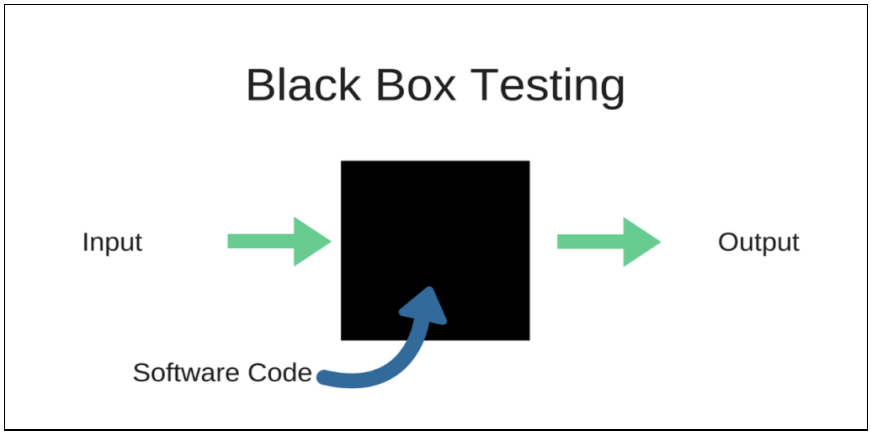
- Functionality testing
- Implementation testing

When functionality is being tested without taking the actual implementation in concern it is known as black-box testing. The other side is known as white-box testing where not only functionality is tested but the way it is implemented is also analyzed.

Exhaustive tests are the best-desired method for a perfect testing. Every single possible value in the range of the input and output values is tested. It is not possible to test each and every value in real world scenario if the range of values is large.

Black-box testing

It is carried out to test functionality of the program. It is also called 'Behavioral' testing. The tester in this case, has a set of input values and respective desired results. On providing input, if the output matches with the desired results, the program is tested 'ok', and problematic otherwise. In this testing method, the design and structure of the code are not known to the tester, and testing engineers and end users conduct this test on the software.



Black-box testing techniques:

Equivalence class - The input is divided into similar classes. If one element of a class passes the test, it is assumed that all the class is passed.

Boundary values - The input is divided into higher and lower end values. If these values pass the test, it is assumed that all values in between may pass too.

Cause-effect graphing - In both previous methods, only one input value at a time is tested. Cause (input) – Effect (output) is a testing technique where combinations of input values are tested in a systematic way.

Pair-wise Testing - The behavior of software depends on multiple parameters. In pairwise testing, the multiple parameters are tested pair-wise for their different values.

State-based testing - The system changes state on provision of input. These systems are tested based on their states and input.

White-box testing

It is conducted to test program and its implementation, in order to improve code efficiency or structure. It is also known as 'Structural' testing.

In this testing method, the design and structure of the code are known to the tester. Programmers of the code conduct this test on the code.

The below are some White-box testing techniques:

Control-flow testing - The purpose of the control-flow testing to set up test cases which covers all statements and branch conditions. The branch conditions are tested for both being true and false, so that all statements can be covered.

Data-flow testing - This testing technique emphasis to cover all the data variables included in the program. It tests where the variables were declared and defined and where they were used or changed.

Testing Levels

Testing itself may be defined at various levels of SDLC. The testing process runs parallel to software development. Before jumping on the next stage, a stage is tested, validated and verified. Testing separately is done just to make sure that there are no hidden bugs or issues left in the software. Software is tested on various levels -

Unit Testing

While coding, the programmer performs some tests on that unit of program to know if it is error free. Testing is performed under white-box testing approach. Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

Integration Testing

Even if the units of software are working fine individually, there is a need to find out if the units if integrated together would also work without errors. For example, argument passing and data updation etc.

System Testing

The software is compiled as product and then it is tested as a whole. This can be accomplished using one or more of the following tests:

- **Functionality testing** - Tests all functionalities of the software against the requirement.
- **Performance testing** - This test proves how efficient the software is. It tests the effectiveness and average time taken by the software to do desired task. Performance testing is done by means of load testing and stress testing where the software is put under high user and data load under various environment conditions.
- **Security & Portability** - These tests are done when the software is meant to work on various platforms and accessed by number of persons.

Acceptance Testing

When the software is ready to hand over to the customer it has to go through last phase of testing where it is tested for user-interaction and response. This is important because even if the software matches all user requirements and if user does not like the way it appears or works, it may be rejected.

Alpha testing - The team of developer themselves perform alpha testing by using the system as if it is being used in work environment. They try to find out how user would react to some action in software and how the system should respond to inputs.

Beta testing - After the software is tested internally, it is handed over to the users to use it under their production environment only for testing purpose. This is not as yet the delivered product. Developers expect that users at this stage will bring minute problems, which were skipped to attend.

Regression Testing

Whenever a software product is updated with new code, feature or functionality, it is tested thoroughly to detect if there is any negative impact of the added code. This is known as regression testing.

Testing Documentation

Testing documents are prepared at different stages -

Before Testing

Testing starts with test cases generation. Following documents are needed for reference –

- **SRS document** - A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform. It also describes the functionality the product needs to fulfill all stakeholders (business, users) needs. A typical SRS includes: A purpose. An overall description. Functional Requirements document…
- **Test Policy document** - This describes how far testing should take place before releasing the product.
- **Test Strategy document** - This mentions detail aspects of test team, responsibility matrix and rights/responsibility of test manager and test engineer.
- **Traceability Matrix document** - This is SDLC document, which is related to requirement gathering process. As new requirements come, they are added to this matrix.

These matrices help testers know the source of requirement. They can be traced forward and backward.

While Being Tested

The following documents may be required while testing is started and is being done:

- **Test Case document** - This document contains list of tests required to be conducted. It includes Unit test plan, Integration test plan, System test plan and Acceptance test plan.
- **Test description** - This document is a detailed description of all test cases and procedures to execute them.
- **Test case report** - This document contains test case report as a result of the test.
- **Test logs** - This document contains test logs for every test case report.

After Testing

The following documents may be generated after testing:

Test summary - This test summary is collective analysis of all test reports and logs. It summarizes and concludes if the software is ready to be launched. The software is released under version control system if it is ready to launch.

Testing vs. Quality Control, Quality Assurance and Audit

We need to understand that software testing is different from software quality assurance, software quality control and software auditing.

Software quality assurance (QA) - These are software development process monitoring means, by which it is assured that all the measures are taken as per the standards of organization. This monitoring is done to make sure that proper software development methods were followed.

Software quality control (QC) - This is a system to maintain the quality of software product. It may include functional and non-functional aspects of software product, which enhance the goodwill of the organization. This system makes sure that the customer is receiving quality product for their requirement and the product certified as ‘fit for use’.

Software audit - This is a review of procedure used by the organization to develop the software. A team of auditors, independent of development team examines the software process, procedure, requirements and other aspects of SDLC. The purpose of software audit is to check that software and its development process, both conform standards, rules and regulations.

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.

Types of maintenance

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based n maintenance size or nature. Following are some types of maintenance based on their characteristics.

Corrective Maintenance - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.

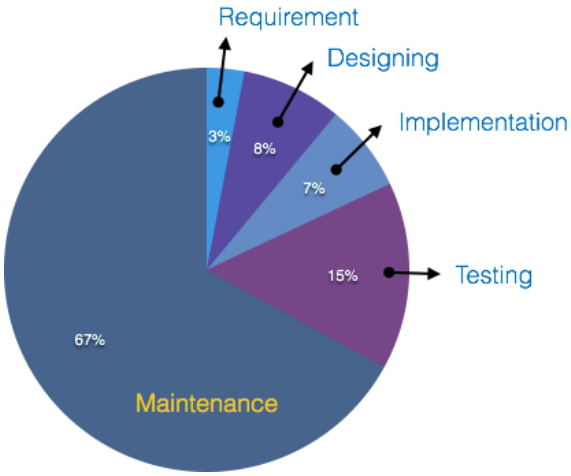
Adaptive Maintenance - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.

Perfective Maintenance - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.

Preventive Maintenance - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Cost of Maintenance

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.



On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:

Real-world factors affecting Maintenance Cost

- The standard age of any software is considered up to 10 to 15 years.
- Older software, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced software on modern hardware.
- As technology advances, it becomes costly to maintain old software.
- Most maintenance engineers are newbie and use trial and error method to rectify problem.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
- Changes are often left undocumented which may cause more conflicts in future.

Factors affecting Maintenance Cost

- Structure of Software Program
- Programming Language
- Dependence on external environment
- Staff reliability and availability

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.

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.

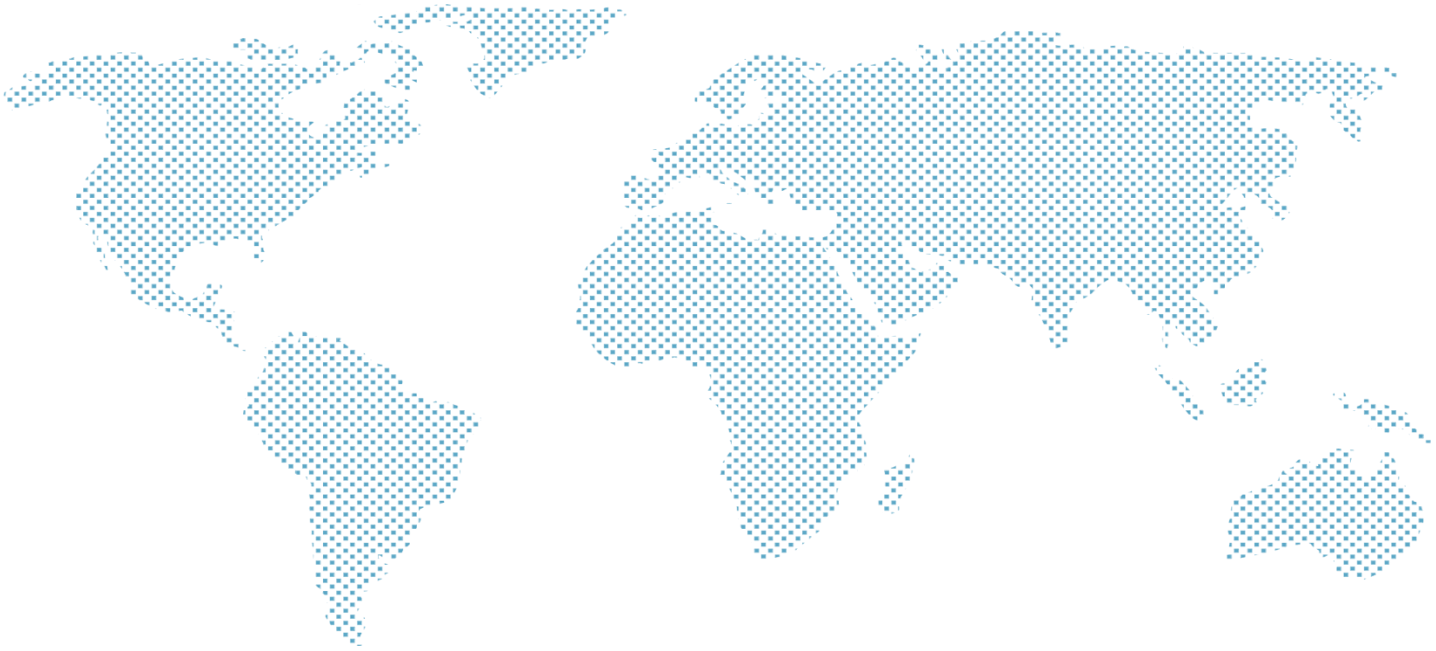
Training facility is provided if required, in addition to the hard copy of user manual.

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.

-----END-----

ADVANCE SOFTWARE ENGINEERING

Lesson No.5



DR. KASHIF LAEEQ

PhD (CS), M.Phil. (CS), MCS (CS), M.Sc. (Math)
Member of IACSIT, IEEE, IEEEP, IJSER, ACM Research Group
Professor, Dept. of Computer Science
Federal Urdu University, Karachi

In previous lecture we learnt…

Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect-free in order to produce the quality product.

Some more concept will be discussed in today's lecture…

Software Testing Definition according to **ANSI/IEEE 1059** standard – A process of analyzing a software item to detect the differences between existing and required conditions (i.e., defects) and to evaluate the features of the software item.

Testing Methods:

1. Static Testing
2. Dynamic Testing

Static Testing: It is also known as Verification in Software Testing. Verification is a static method of checking documents and files. Verification is the process, to ensure that whether we are building the product right i.e., to verify the requirements which we have and to verify whether we are developing the product accordingly or not.
Activities involved here are Inspections, Reviews, and Walkthroughs.

Dynamic Testing: It is also known as Validation in Software Testing. Validation is a dynamic process of testing the real product. Validation is the process, whether we are building the right product i.e., to validate the product which we have developed is right or not.

Testing Approaches:
There are three types of software testing approaches.

1. White Box Testing
2. Black Box Testing
3. Grey Box Testing

Grey Box Testing: Grey box is the combination of both White Box and Black Box Testing. The tester who works on this type of testing needs to have access to design documents. This helps to create better test cases in this process.
It is a technique to test the software product or application with partial knowledge of the internal workings of an application. The purpose of this testing is to search for defects due to improper code structure or improper functioning usage of an application.

- In White Box testing internal structure (code) is known
- In Black Box testing internal structure (code) is unknown
- In Grey Box Testing internal structure (code) is partially known

Types of Black Box Testing:

1. Functional Testing
2. Non-functional Testing

Functional Testing:
In simple words, what the system actually does is functional testing. To verify that each function of the software application behaves as specified in the requirement document. Testing all the functionalities by providing appropriate input to verify whether the actual output is matching the expected output or not. It falls within the scope of black-box testing and the testers need not concern about the source code of the application.

Non-functional Testing:
In simple words, how well the system performs is non-functionality testing. Non-functional testing refers to various aspects of the software such as performance, load, stress, scalability, security, compatibility, etc., The Main focus is to improve the user experience on how fast the system responds to a request.

Why do we need Software Testing?

As per the current trend, due to constant change and development in digitization, our lives are improving in all areas. We access our bank online, we do shopping online, we order food online, and many more. We rely on software and systems. What if these systems turn out to be defective? We all know that one small bug shows a huge impact on business in terms of financial loss and goodwill. To deliver a quality product, we need to have Software Testing in the Software Development Process.
Some of the reasons why software testing becomes a very significant and integral part in the field of information technology are as follows.

1. Cost-effectiveness
2. Customer Satisfaction
3. Security
4. Product Quality

Seven Principles of Software Testing

The seven principles of Software Testing are as follows:

1. Testing shows presence of defects
2. Exhaustive testing is impossible
3. Early testing
4. Defect clustering
5. Pesticide paradox
6. Testing is context dependent
7. Absence of error – fallacy

1. **Testing Shows Presence of Defects:**

Testing shows the presence of defects in the software. The goal of testing is to make the software error-free or fail. Sufficient testing reduces the presence of defects. In case testers are unable to find defects after repeated regression testing doesn't mean that the software is bug-free.

Testing talks about the presence of defects and don't talk about the absence of defects.

2. **Exhaustive Testing is Impossible:**

What is Exhaustive Testing?

Testing all the functionalities using all valid and invalid inputs and preconditions is known as Exhaustive testing.

Why it's impossible to achieve Exhaustive Testing?

Assume we have to test an input field which accepts age between 18 to 20 so we do test the field using 18,19,20. In case the same input field accepts the range between 18 to 100 then we have to test using inputs such as 18, 19, 20, 21, …., 99, 100. It's a basic example, you may think that you could achieve it using automation tool. Imagine the same field accepts some billion values. It's impossible to test all possible values due to release time constraints.

If we keep on testing all possible test conditions then the software execution time and costs will rise. So instead of doing exhaustive testing, risks and priorities will be taken into consideration whilst doing testing and estimating testing efforts.

3. **Early Testing:**

Defects detected in early phases of SDLC are less expensive to fix. So conducting early testing reduces the cost of fixing defects.
Assume two scenarios, first one is you have identified an incorrect requirement in the requirement gathering phase and the second one is you have identified a bug in the fully developed functionality. It is cheaper to change the incorrect requirement compared to fixing the fully developed functionality which is not working as intended.

4. **Defect Clustering:**

Defect Clustering in software testing means that a small module or functionality contains most of the bugs or it has the most operational failures.

As per the Pareto Principle (80-20 Rule), 80% of issues comes from 20% of modules and remaining 20% of issues from remaining 80% of modules. So we do emphasize testing on the 20% of modules where we face 80% of bugs.

5. **Pesticide Paradox:**

Pesticide Paradox in software testing is the process of repeating the same test cases again and again, eventually, the same test cases will no longer find new bugs. So to overcome this Pesticide

Paradox, it is necessary to review the test cases regularly and add or update them to find more defects.

6. **Testing is Context Dependent:**

Testing approach depends on the context of the software we develop. We do test the software differently in different contexts. For example, online banking application requires a different approach of testing compared to an e-commerce site.

7. **Absence of Error – Fallacy:**

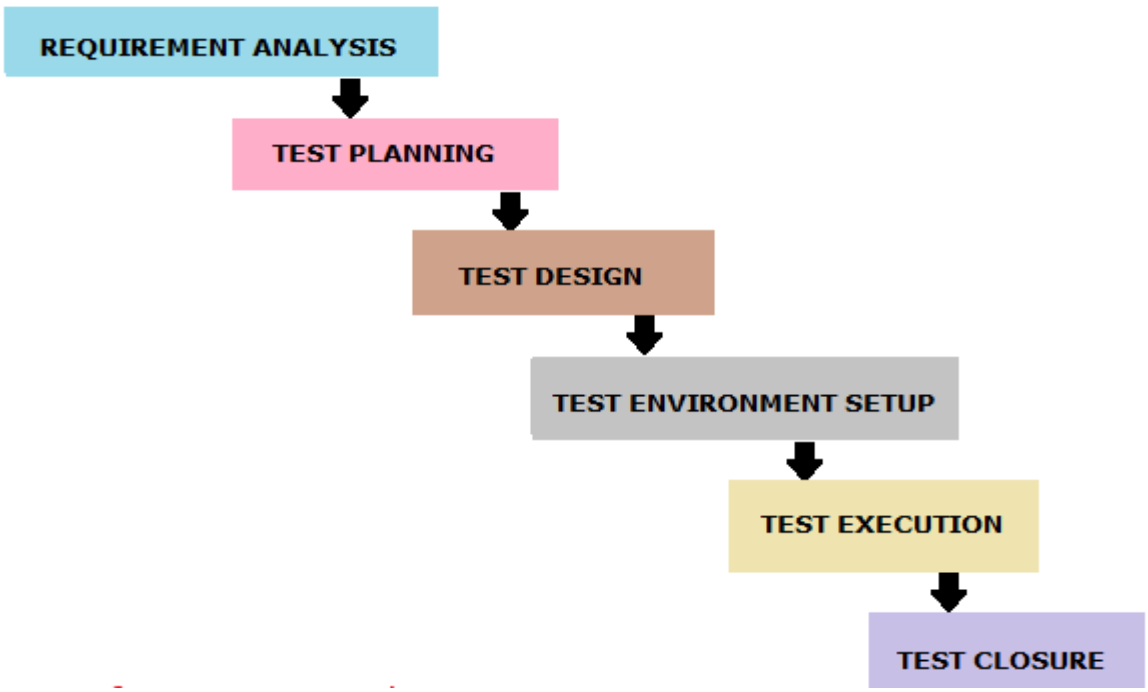
99% of bug-free software may still be unusable, if wrong requirements were incorporated into the software and the software is not addressing the business needs.

The software which we built not only be a 99% bug-free software but also it must fulfill the business needs otherwise it will become an unusable software.

These are the seven principles of Software Testing every professional tester should know.

Software Testing Life Cycle:

Software Testing Life Cycle (**STLC**) identifies what test activities to carry out and when to accomplish those test activities. Even though testing differs between organizations, there is a testing life cycle.



The different phases of Software Testing Life Cycle are:

1. Requirement Analysis
2. Test Planning
3. Test Design
4. Test Environment Setup
5. Test Execution
6. Test Closure

Every phase of STLC (Software Testing Life Cycle) has a definite Entry and Exit Criteria.

1. **Requirement Analysis:**

The entry criteria for this phase is the BRS (Business Requirement Specification) document. During this phase, test team studies and analyzes the requirements from a testing perspective. This phase helps to identify whether the requirements are testable or not. If any requirement is not testable, the test team can communicate with various stakeholders (Client, Business Analyst, Technical Leads, System Architects, etc) during this phase so that the mitigation strategy can be planned.

Entry Criteria: BRS (Business Requirement Specification)
Deliverables: List of all testable requirements, Automation feasibility report (if applicable)

2. **Test Planning:**

Test planning is the first step of the testing process. In this phase typically Test Manager/Test Lead involves determining the effort and cost estimates for the entire project. Preparation of the Test Plan will be done based on the requirement analysis. Activities like resource planning, determining roles and responsibilities, tool selection (if automation), training requirement, etc., carried out in this phase. The deliverables of this phase are Test Plan & Effort estimation documents.

Entry Criteria: Requirements Documents

Deliverables: Test Strategy, Test Plan, and Test Effort estimation document.

3. **Test Design**:

The test team starts with test case development activity here in this phase. Test team prepares test cases, test scripts (if automation), and test data. Once the test cases are ready then these test cases are reviewed by peer members or team lead. Also, the test team prepares the Requirement Traceability Matrix (RTM). RTM traces the requirements to the test cases that are needed to verify whether the requirements are fulfilled. The deliverables of this phase are Test Cases, Test Scripts, Test Data, and Requirements Traceability Matrix

Entry Criteria: Requirements Documents (Updated version of unclear or missing requirement)

Deliverables: Test cases, Test Scripts (if automation), Test data.

4. **Test Environment Setup:**

This phase can be started in parallel with the Test design phase. The test environment setup is done based on the hardware and software requirement list. In some cases, the test team may not be involved in this phase. The development team or customer provides the test environment. Meanwhile, the test team should prepare the smoke test cases to check the readiness of the given test environment.

Entry Criteria: Test Plan, Smoke Test cases, Test Data
Deliverables: Test Environment. Smoke Test Results.

5. **Test Execution:**

The test team starts executing the test cases based on the planned test cases. If a test case result is Pass/Fail then the same should be updated in the test cases. The defect report should be prepared for failed test cases and should be reported to the Development Team through a bug tracking tool (eg., Quality Center) for fixing the defects. Retesting will be performed once the defect was fixed.

Entry Criteria: Test Plan document, Test cases, Test data, Test Environment.
Deliverables: Test case execution report, Defect report, RTM

6. **Test Closure:**

The final stage where we prepare Test Closure Report, Test Metrics. The testing team will be called out for a meeting to evaluate cycle completion criteria based on Test coverage, Quality, Time, Cost, Software, Business objectives. Test team analyses the test artifacts (such as Test cases, Defect reports, etc.,) to identify strategies that have to be implemented in the future, which will help to remove process bottlenecks in the upcoming projects. Test metrics and Test closure report will be prepared based on the above criteria.

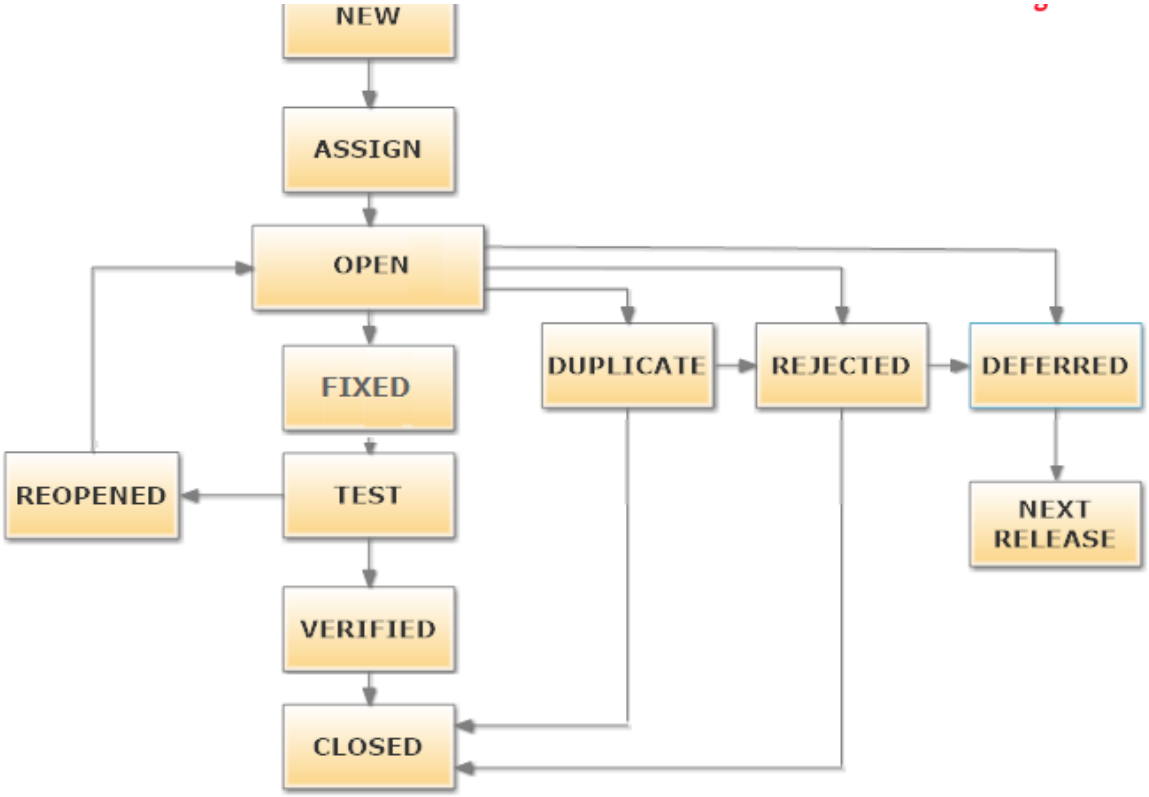
Entry Criteria: Test Case Execution report (make sure there are no high severity defects opened), Defect report
Deliverables: Test Closure report, Test metrics

Bug Life Cycle or Defect Life Cycle:

Bug life cycle is also known as **Defect life cycle**. In Software Development process, the bug has a life cycle. The bug should go through the life cycle to be closed. Bug life cycle varies depends upon the tools (QC, JIRA etc.,) used and the process followed in the organization.

What is a Software Bug?

Software bug can be defined as the abnormal behavior of the software. Bug starts when the defect is found and ends when a defect is closed, after ensuring it is not reproduced.



The different states of a bug in the bug life cycle are as follows:

New: When a tester finds a new defect. He should provide a proper Defect document to the Development team to reproduce and fix the defect. In this state, the status of the defect posted by tester is "New"

Assigned: Defects which are in the status of New will be approved (if valid) and assigned to the development team by Test Lead/Project Lead/Project Manager. Once the defect is assigned then the status of the bug changes to "Assigned"

Open: The development team starts analyzing and works on the defect.

Fixed: When a developer makes the necessary code change and verifies the change, then the status of the bug will be changed as "Fixed" and the bug is passed to the testing team.

Test: If the status is "Test", it means the defect is fixed and ready to do test whether it is fixed or not.

Verified: The tester re-tests the bug after it got fixed by the developer. If there is no bug detected in the software, then the bug is fixed and the status assigned is "verified."

Closed: After verified the fix, if the bug is no longer exits then the status of bug will be assigned as "Closed."

Reopen: If the defect remains same after the retest, then the tester posts the defect using defect retesting document and changes the status to “Reopen”. Again the bug goes through the life cycle to be fixed.

Duplicate: If the defect is repeated twice or the defect corresponds the same concept of the bug, the status is changed to “duplicate” by the development team.

Deferred: In some cases, Project Manager/Lead may set the bug status as deferred. If the bug found during end of release and the bug is minor or not important to fix immediately If the bug is not related to current build If it is expected to get fixed in the next release Customer is thinking to change the requirement In such cases the status will be changed as “deferred” and it will be fixed in the next release.

Rejected: If the system is working according to specifications and bug is just due to some misinterpretation (such as referring to old requirements or extra features) then Team lead or developers can mark such bugs as “Rejected”

Some other statuses are:

Cannot be fixed: Technology not supporting, Root of the product issue, Cost of fixing bug is more

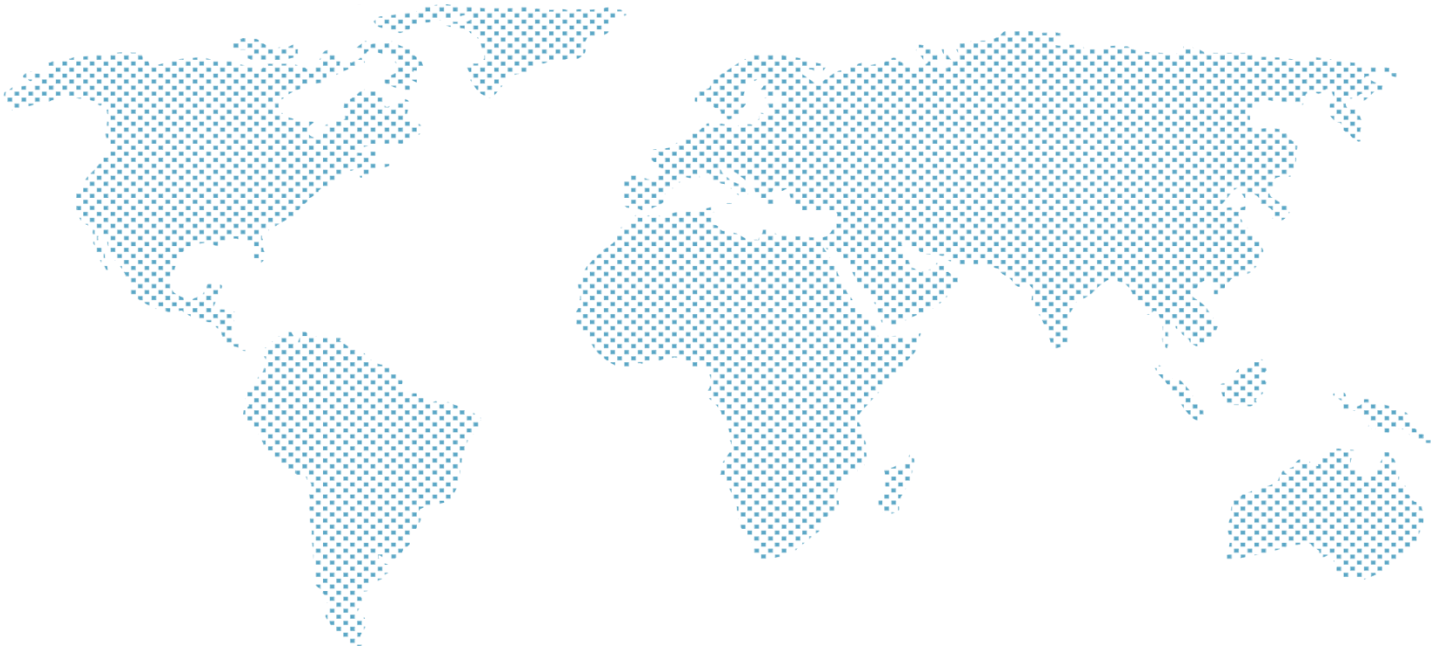
Not Reproducible: Platform mismatch, improper defect document, data mismatch, build mismatch, inconsistent defects

Need more information: If a developer is unable to reproduce the bug as per the steps provided by a tester then the developer can change the status as “need more information’. In this case, the tester needs to add detailed reproducing steps and assign bug back to the development team for a fix. This won’t happen if the tester writes a good defect document.
This is all about Bug Life Cycle / Defect Life Cycle. Some companies use these bug id’s in RTM to map with the test cases.

-----END-----

ADVANCE SOFTWARE ENGINEERING

Lesson No.6



DR. KASHIF LAEEQ

PhD (CS), M.Phil. (CS), MCS (CS), M.Sc. (Math)
Member of IACSIT, IEEE, IEEEP, IJSER, ACM Research Group
Professor, Dept. of Computer Science
Federal Urdu University, Karachi

In lesson 3 we learnt the basic definition of Six Sigma, now in lesson 6 we will cover some more advanced concept of Six Sigma.

Six Sigma (6σ) is a set of techniques and tools for process improvement. It is a disciplined, statistical-based, data-driven approach and continuous improvement methodology for eliminating defects in a product, process or service. It is also a methodology for pursuing continuous improvement in customer satisfaction and profit. It is a management philosophy attempting to improve effectiveness and efficiency.

Six Sigma is a highly well-organized process that helps us focus on developing and delivering near-perfect products and services.

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

The word *Sigma* is a statistical term that measures how far a given process deviates from perfection.

The central idea behind Six Sigma: If you can measure how many "defects" you have in a process, you can systematically figure out how to eliminate them and get as close to "zero defects" as possible and specifically it means a failure rate of 3.4 parts per million or 99.9997% perfect.

Key Concepts of Six Sigma

At its core, Six Sigma revolves around a few key concepts.

- **Critical to Quality** − Attributes most important to the customer.
- **Defect** − Failing to deliver what the customer wants.
- **Process Capability** − what your process can deliver.
- **Variation** − what the customer sees and feels.
- **Stable Operations** − Ensuring consistent, predictable processes to improve what the customer sees and feels.
- **Design for Six Sigma** − Designing to meet customer needs and process capability.

Our Customers Feel the Variance, Not the Mean. So Six Sigma focuses first on reducing process variation and then on improving the process capability.

Wrong Concept about Six Sigma

There are several misunderstandings about Six Sigma. Some of them are given below −

- Six Sigma is only concerned with reducing defects.
- Six Sigma is a process for production or engineering.
- Six Sigma cannot be applied to engineering activities.
- Six Sigma uses difficult-to-understand statistics.
- Six Sigma is just training.

Benefits of Six Sigma

Six Sigma offers six major benefits that attract companies −

- Generates sustained success
- Sets a performance goal for everyone
- Enhances value to customers
- Accelerates the rate of improvement
- Process improvement
- Executes strategic change

Six Sigma - Key Elements

There are three key elements of Six Sigma Process Improvement −

- Customers
- Processes
- Employees

The Customers

Customers define quality. They expect performance, reliability, competitive prices, on-time delivery, service, clear and correct transaction processing and more. This means it is important to provide what the customers need to gain customer delight.

The Processes

Defining processes as well as defining their metrics and measures is the central aspect of Six Sigma.

In a business, the quality should be looked from the customer's perspective and so we must look at a defined process from the outside-in.

By understanding the transaction lifecycle from the customer's needs and processes, we can discover what they are seeing and feeling. This gives a chance to identify weak areas with in a process and then we can improve them.

The Employees

A company must involve all its employees in the Six Sigma program. Company must provide opportunities and incentives for employees to focus their talents and ability to satisfy customers.

It is important to Six Sigma that all the team members should have a well-defined role with measurable objectives.

Six Sigma – Organization

Under a Six Sigma program, the members of an organization are assigned specific roles to play, each with a title. This highly structured format is necessary in order to implement Six Sigma throughout the organization.

There are seven specific responsibilities or "role areas" in a Six Sigma program, which are as follows.

1. Leadership

A leadership team or council defines the goals and objectives in the Six Sigma process. Just as a corporate leader sets a tone and course to achieve an objective, the Six Sigma council sets the goals to be met by the team. Here is the list of leadership Council Responsibilities −

- Defines the purpose of the Six Sigma program
- Explains how the result is going to benefit the customer
- Sets a schedule for work and interim deadlines
- Develops a mean for review and oversight
- Support team members and defend established positions

2. Sponsor

Six Sigma sponsors are high-level individuals who understand Six Sigma and are committed to its success. The individual in the sponsor role acts as a problem solver for the ongoing Six Sigma project. Six Sigma is generally led by a full-time, high-level champion, such as an Executive Vice President.

Sponsors are the owners of processes and systems, who help initiate and coordinate Six Sigma improvement activities in their areas of responsibilities.

3. Implementation Leader

The person responsible for supervising the Six Sigma team effort, who supports the leadership council by ensuring that the work of the team is completed in the desired manner, is the implementation Leader.

Ensuring success of the implementation plan and solving problems as they arise, training as needed, and assisting sponsors in motivating the team are some of the key responsibilities of an implementation leader.

4. Coach

Coach is a Six Sigma expert or consultant who sets a schedule, defines result of a project, and who mediates conflict, or deals with resistance to the program.

Duties include working as a go-between for sponsor and leadership, scheduling the work of the team, identifying and defining the desired results of the project, mediating disagreements, conflicts, and resistance to the program and identifying success as it occurs.

5. Team Leader

It is an individual responsible for overseeing the work of the team and for acting as a go-between with the sponsor and the team members.

Responsibilities include communication with the sponsor in defining project goals and rationale, picking and assisting team members and other resources, keeping the project on schedule, and keeping track of steps in the process as they are completed.

6. Team Member

An employee who works on a Six Sigma project, given specific duties within a project, and has deadlines to meet in reaching specific project goals.

Team members execute specific Six Sigma assignments and work with other members of the team within a defined project schedule, to reach specifically identified goals.

7. Process Owner

The individual who takes on responsibility for a process after a Six Sigma team has completed its work.

Definitions of Roles Belt - Colors

The assignment of belt colors to various roles is derived from the obvious source, the martial arts. Based on experience and expertise following roles have evolved over the year.

The belt names are a tool for defining levels of expertise and experience. They do not change or replace the organizational roles in the Six Sigma process.

Black Belt

The person possessing this belt has achieved the highest skill level and is an experienced expert in various techniques. As applied to the Six Sigma program, the individual designated as a Black Belt has completed a thorough internal training program and has the experience working on several projects.

The black belt holder is usually given the role of a team leader, the person who is responsible for execution and scheduling.

Master Black Belt

A person who deals with the team or its leadership; but is not a direct member of the team itself. This may be equivalent to the role played by the coach, or for more technical and complex projects.

The Master Black Belt is available to answer procedural questions and to resolve the technical issues that come up.

Green Belt

The Green Belt designation can also belong to the team leader or to a member of the team working directly with the team leader.

A Green Belt is less experienced than a Black Belt but is cast in a key role within the team.

Project Selection for Six Sigma

One of the most difficult challenges in Six Sigma is the selection of the most appropriate problem to attack. There are generally two ways to generate projects −

- **Top-down** − This approach is generally tied to business strategy and is aligned with customer needs. The major weakness is they are too broad in scope to be completed in a timely manner (most six sigma projects are expected to be completed in 3-6 months).
- **Bottom-up** − In this approach, Black Belts choose the projects that are well-suited for the capabilities of teams. A major drawback of this approach is that, projects may not be tied directly to strategic concerns of the management thereby, receiving little support and low recognition from the top.

Six Sigma – Methodology

Six Sigma has two key methodologies −

- **DMAIC** (an acronym for **Define, Measure, Analyze, Improve and Control**) − It refers to a data-driven quality strategy for improving processes. This methodology is used to improve an existing business process.

- **DMADV (Define, Measure, Analyze, Design, and Verify)** − It refers to a data-driven quality strategy for designing products & processes. This methodology is used to create new product designs or process designs in such a way that it results in a more predictable, mature and defect free performance.

-----END-----