



WELCOME

Software Engineering Best Practices

by

Girish Godbole

Program Agenda

- 1 • Software Engineering - An Overview
- 2 • Software Development Models
- 3 • Software Requirements Engineering
- 4 • Software Design and Development
- 5 • Software Project Management(SPM)
- 6 • Software Testing Best Practices

Software Engineering - What



Software Engineering - An Overview

Software -

“A set of executable program in an application along with associated libraries and documentations”

“A set of instructions, data or programs used to operate computers and execute specific tasks.”

Engineering -

“On the other hand, is all about developing products, using well-defined, scientific principles and methods.”

Software Engineering -

“Software engineering is the systematic, disciplined and quantifiable approach to the design, development, testing, implementation, and maintenance of application software and computer systems software.”

Broadly, software engineering can be divided into two categories:

Application Engineering - Application engineering is the process to develop software applications for businesses and organizations

Systems Engineering - System engineering refers to the coordination of the development and maintenance of computer systems for businesses and organizations.

Software Engineering - Importance

Large software – In our real life, it is quite more comfortable to build a wall than a house or building. In the same manner, as the size of the software becomes large, software engineering helps you to build software.

Scalability- If the software development process were based on scientific and engineering concepts, it is easier to re-create new software to scale an existing one.

Adaptability: Whenever the software process was based on scientific and engineering, it is easy to re-create new software with the help of software engineering.

Cost- Hardware industry has shown its skills and huge manufacturing has lower the cost of the computer and electronic hardware.

Dynamic Nature– Always growing and adapting nature of the software. It depends on the environment in which the user works.

Quality Management: Offers better method of software development to provide quality software products.

The importance of software engineering is growing day by day as people require a wide range of applications to efficiently manage their businesses and increase productivity. Software engineering helps to handle complex and big projects by applying the principle of modularity. It divides complex projects into different modules and allows developers to work independently on each module. Software engineering itself is a process of developing a perfect plan for software development. Therefore, it reduces the time and cost required to develop software.

Software Engineering - Challenges

Critical Challenges

- In safety-critical areas such as space, aviation, nuclear power plants, etc. the cost of software failure can be massive because lives are at risk.
- Increased market demands for fast turnaround time.
- Dealing with the increased complexity of software need for new applications.
- The diversity of software systems should be communicating with each other.

Characteristics of Good Software

Every software must satisfy the following attributes:

- Operational
- Transitional
- Maintenance

Operational

This characteristic let us know about how well software works in the operations which can be measured on:

- Budget, Efficiency, Usability, Dependability, Correctness, Functionality, Safety, Security

Transitional

This is an essential aspect when the software is moved from one platform to another:

- Interoperability, Reusability, Portability, Adaptability

Maintenance

This aspect talks about how well software has the capabilities to adapt itself in the quickly changing environment:

- Flexibility, Maintainability, Modularity, Scalability

Software Engineering -Objectives

- **Maintainability** – It should be feasible for the software to evolve to meet changing requirements.
- **Efficiency** – The software should not make wasteful use of computing devices such as memory, processor cycles, etc.
- **Correctness** – A software product is correct if the different requirements as specified in the SRS document have been correctly implemented.
- **Reusability** – A software product has good reusability if the different modules of the product can easily be reused to develop new products.
- **Testability** – Here software facilitates both the establishment of test criteria and the evaluation of the software with respect to those criteria
- **Reliability** – It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.
- **Portability** – In this case, the software can be transferred from one computer system or environment to another.
- **Adaptability** – The software allows differing system constraints
- **Interoperability** – Capability of 2 or more functional units to process data cooperatively.

Software Engineering - Classification of Software

Types of Project /Classification of Software

- Web based project development
- Standalone/Desktop based programs/applications
- Jobs/Schedules
- Enterprise projects (ERP systems)
- Database oriented applications
- Reporting/Analytics applications
- Artificial Intelligence Software
- Scientific Software
- Embedded Software
- Cloud based projects

Software Engineering – Model Selection

Which Model to select ?

Before selecting any model, there are few questions which should be addressed other than looking into the differences in the approach of any model. These questions will not just help to find the model but will also play a role in finding the best according to your demands. These questions are:

- What is the size of the project you are going to work on?
- Do you need a prototype?
- Are you using a packaged solution?
- How flexible is your team?
- How much will your customer participate in the process?
- Does your project manager have experience?

Above mentioned important queries need to be answered before going any step ahead.

Software Development Life Cycle –SDLC Model



Software Development Life Cycle –SDLC Model

SDLC –

1. A framework that describes the activities performed at each stage of a software development project.
2. Is a systematic process for building software that ensures the quality and correctness of the software built.

Every phase of the SDLC life Cycle has its own process and deliverables that feed into the next phase.

Why

- 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

Software Development Life Cycle –SDLC Model

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 recognition 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

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.

There are mainly five types of feasibilities checks:

- Economic:** Can we complete the project within the budget or not?
- Legal:** Can we handle this project as cyber law and other regulatory framework/compliances.
- Operation feasibility:** Can we create operations which is expected by the client?
- Technical:** Need to check whether the current computer system can support the software
- Schedule:** Decide that the project can be completed within the given schedule or not.

Software Development Life Cycle –SDLC Model

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. There are two kinds of design documents developed in this phase:

High-Level Design (HLD)

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture diagrams along with technology details

Low-Level Design(LLD)

- Functional logic of the modules
- Database tables, which include type and size
- Complete detail of the interface
- Addresses all types of dependency issues
- Listing of error messages
- Complete input and outputs for every module

Software Development Life Cycle –SDLC Model

Phase 4: Coding

Once the system design phase is over, the next phase is coding. In this phase, developers start build 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.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

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.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

Software Development Life Cycle –SDLC Model

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

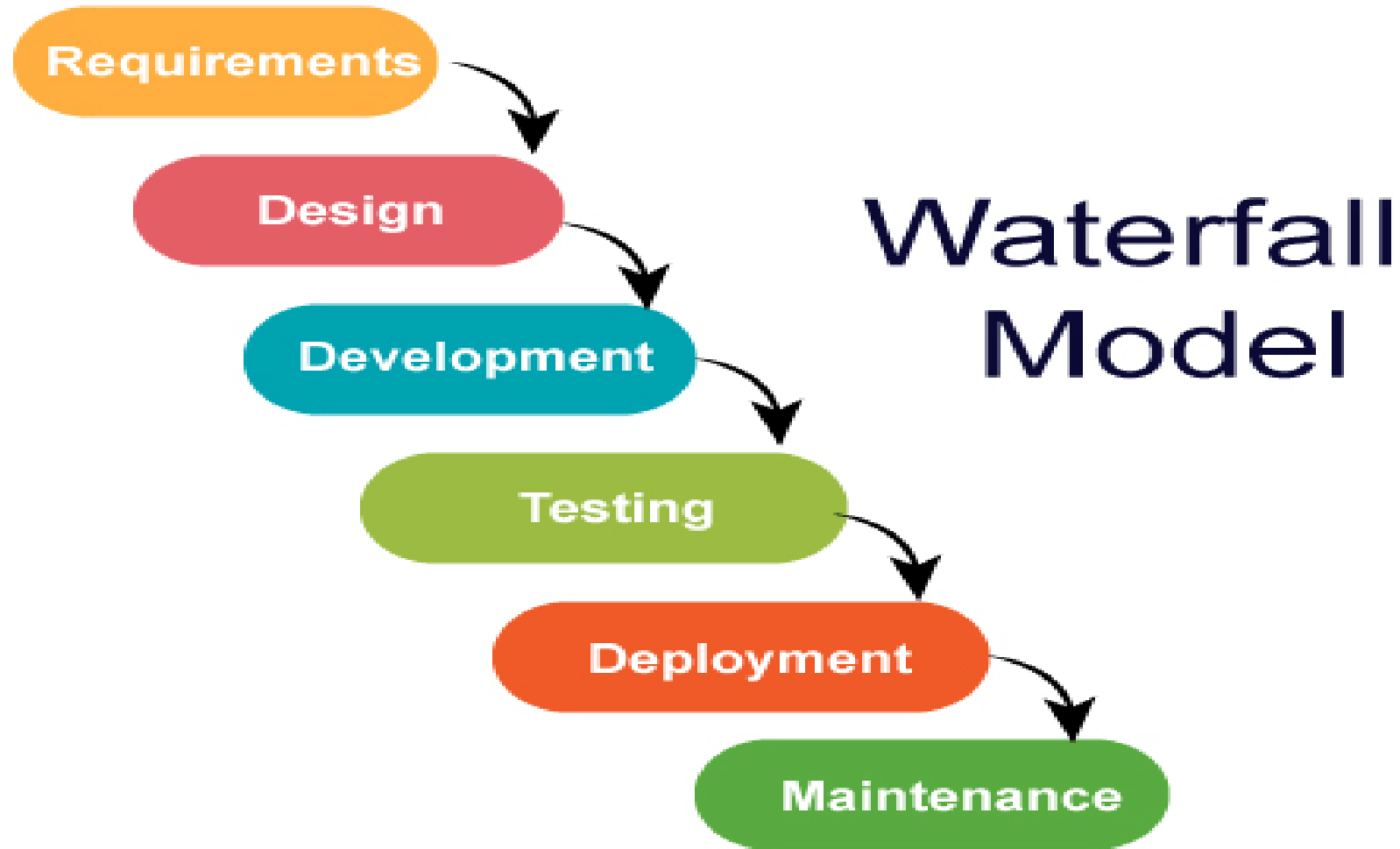
Bug fixing – bugs are reported because of some scenarios which are not tested at all

Upgrade – Upgrading the application to the newer versions of the Software

Enhancement – Adding some new features into the existing software

The focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

SDLC – Waterfall Model



SDLC – Waterfall Model

Waterfall Strengths

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

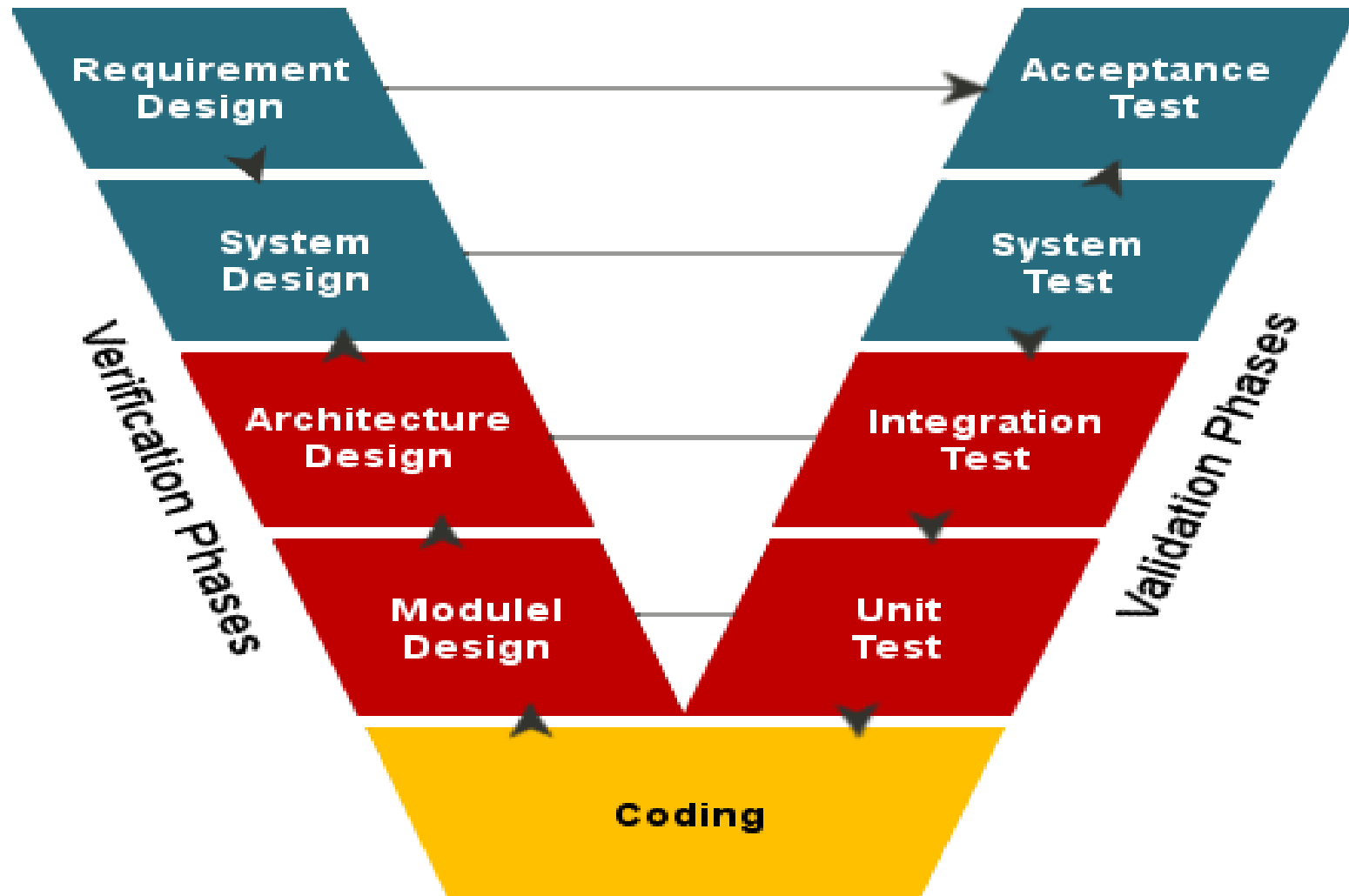
Waterfall Deficiencies

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – inhibits flexibility
- Not ideal for long, complex, or ongoing projects where more flexibility is required
- Does not reflect problem-solving nature of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

When to use

- Requirements are very well known
- Product definition is stable. Technology is understood
- New version of an existing product. Porting an existing product to a new platform.
- Useful for small, one-time projects with few requirements and short timelines because it is simple and can be quickly configured

SDLC – V Model



SDLC – V Model

V Model Strengths

Emphasize planning for verification and validation of the product in early stages of product development

Each deliverable must be testable

Project management can track progress by milestones

Easy to use

V Model Weakness

Does not easily handle concurrent events

Does not handle iterations or phases

Not ideal for long, complex, or ongoing projects

Does not easily handle dynamic changes in requirements

Does not contain risk analysis activities

When to use

Excellent choice for systems requiring high reliability – hospital patient control applications

All requirements are known up-front

When it can be modified to handle changing requirements beyond analysis phase

Solution and technology are known

Verification Vs. Validation

Verification	Validation
Are we building the system, right?	Are we building the right system?
Verification process includes checking of documents, design, code and program	Validation process includes testing and validation of the actual product.
Verification is the static testing.	Validation is the dynamic testing.
Verification uses methods like reviews, walkthroughs, inspections and desk-checking	Validation uses methods like black box testing, white box testing and non-functional testing.
Verification does not involve code execution	Validation involves code execution.
It checks whether the software conforms to specifications or not.	It checks whether the software meets the requirements and expectations of a customer or not.
Cost of errors caught in Verification is less than errors found in Validation.	Cost of errors caught in Validation is more than errors found in Verification.
It can find the bugs in the early stage of the development.	It can only find the bugs that could not be found by the verification process.
The goal of verification is application and software architecture and specification.	The goal of validation is an actual product.
Quality assurance team does verification.	Validation is executed on software code with the help of testing team.
It comes before validation.	It comes after verification.
It consists of checking of documents/files and is performed by human.	It consists of execution of program and is performed by computer.

Static Testing Vs. Dynamic Testing

Static Testing	Dynamic Testing
It is performed in the early stage of the software development.	It is performed at the later stage of the software development.
In static testing whole code is not executed.	In dynamic testing whole code is executed.
Static testing prevents the defects.	Dynamic testing finds and fixes the defects.
Static testing is performed before code deployment.	Dynamic testing is performed after code deployment.
Static testing is less costly.	Dynamic testing is highly costly.
Static Testing involves checklist for testing process.	Dynamic Testing involves test cases for testing process.
It includes walkthroughs, code review, inspection etc.	It involves functional and nonfunctional testing.
It generally takes shorter time.	It usually takes longer time as it involves running several test cases.
It can discover variety of bugs.	It expose the bugs that are explorable through execution hence discover only limited type of bugs.
Static Testing may complete 100% statement coverage in comparably less time.	While dynamic testing only achieves less than 50% statement coverage.
Verification	Validation

Software Engineering – Iterative Model

In a practical software development project, the old waterfall model is not going to be a big help. It has a lot of errors in it when it comes to bigger and complex projects. So, Iterative can be a good alternative. In the Iterative model, the development begins by specifying and implementing part of the software which you will review later to identify the further requirements. According to the iterative model, you can make software by using some of the software specifications and develop the first version of the software. And later, if you or your client realize that you need some modification than it can be easily made using a new iteration.

The process of Iterative model is cyclic, once the initial planning is complete, few of the phases are kept repeating repeatedly, with the completion of each cycle incrementally improving and iterating on the software.

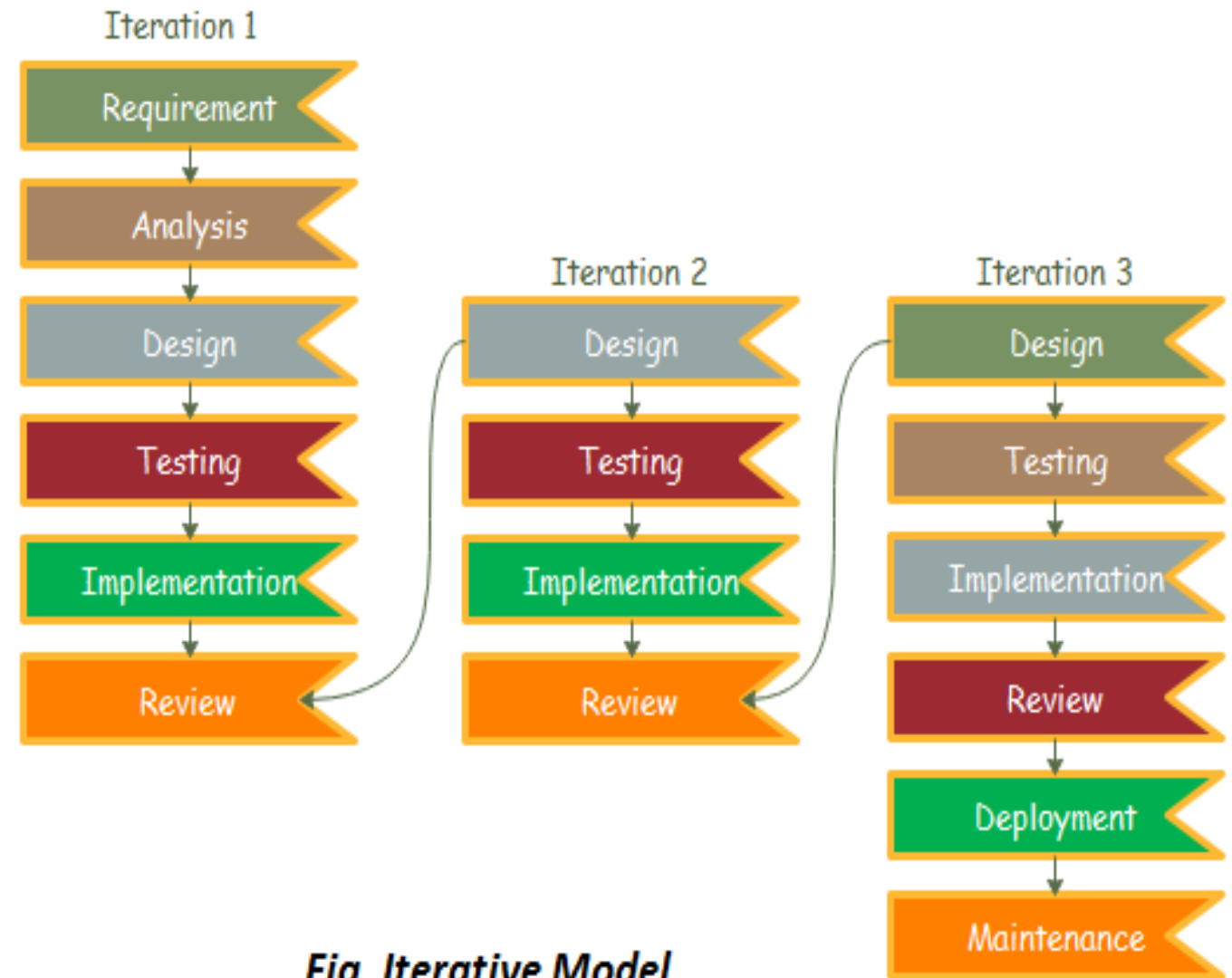


Fig. Iterative Model

Software Engineering – Iterative Model

Advantage(Pros) of Iterative Model:

1. Testing and debugging during smaller iteration is easy.
2. A Parallel development can plan.
3. It is easily acceptable to ever-changing needs of the project.
4. Risks are identified and resolved during iteration.
5. Limited time spent on documentation and extra time on designing.

Disadvantage(Cons) of Iterative Model:

1. It is not suitable for smaller projects.
2. More Resources may be required.
3. Design can be changed again and again because of imperfect requirements.
4. Requirement changes can cause over budget.
5. Project completion date not confirmed because of changing requirements.

The Iterative model can pose more risks with frequent changes, unknown costs and resource requirements, and uncertain deadlines

When to use the Iterative Model?

1. When requirements are defined clearly and easy to understand.
2. When the software application is large.
3. New technology involved
4. Being learned by the development team
5. Big project with high-risk features and goals will change later on.

SDLC – Spiral Model

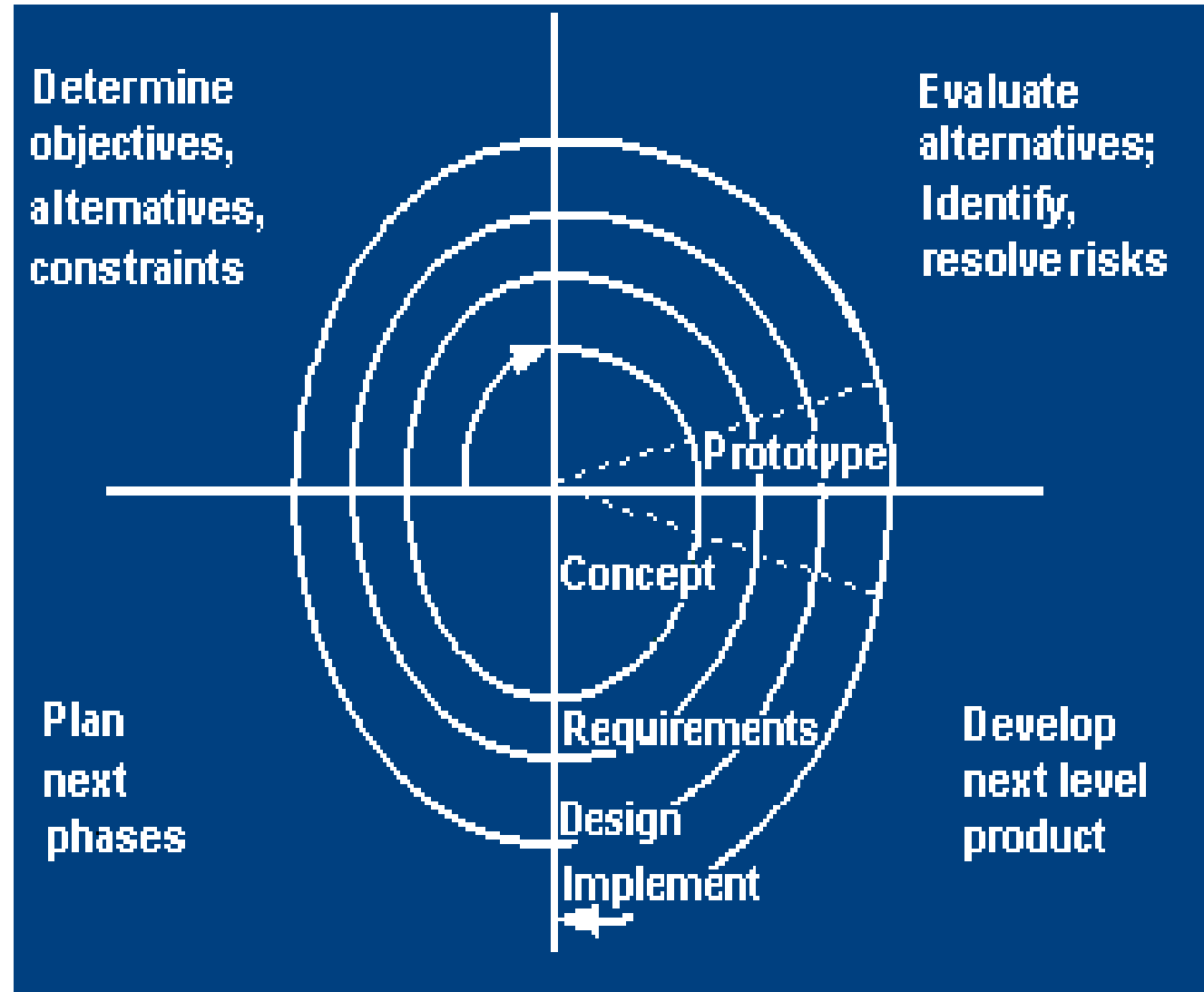
Spiral Model is a risk-driven software development process model. It is a combination of waterfall model and iterative model.

Spiral Model helps to adopt software development elements of multiple process models for the software project based on unique risk patterns ensuring efficient development process.

Each phase of spiral model in software engineering begins with a design goal and ends with the client reviewing the progress.

The development process in Spiral model in SDLC, starts with a small set of requirement and goes through each development phase for those set of requirements.

The software engineering team adds functionality for the additional requirement in every-increasing spirals until the application is ready for the production phase.



SDLC – Spiral Model

Spiral Model Strengths

- Provides early indication of insurmountable risks, without much cost
- Users see the system early because of rapid prototyping tools
- Critical high-risk functions are developed first
- The design does not have to be perfect
- Users can be closely tied to all lifecycle steps
- Early and frequent feedback from users
- Cumulative costs assessed frequently

Spiral Model weakness

- Time spent for evaluating risks too large for small or low-risk projects
- Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive
- The model is complex, Risk assessment expertise is required
- Spiral may continue indefinitely, Developers must be reassigned during non-development phase activities
- May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration

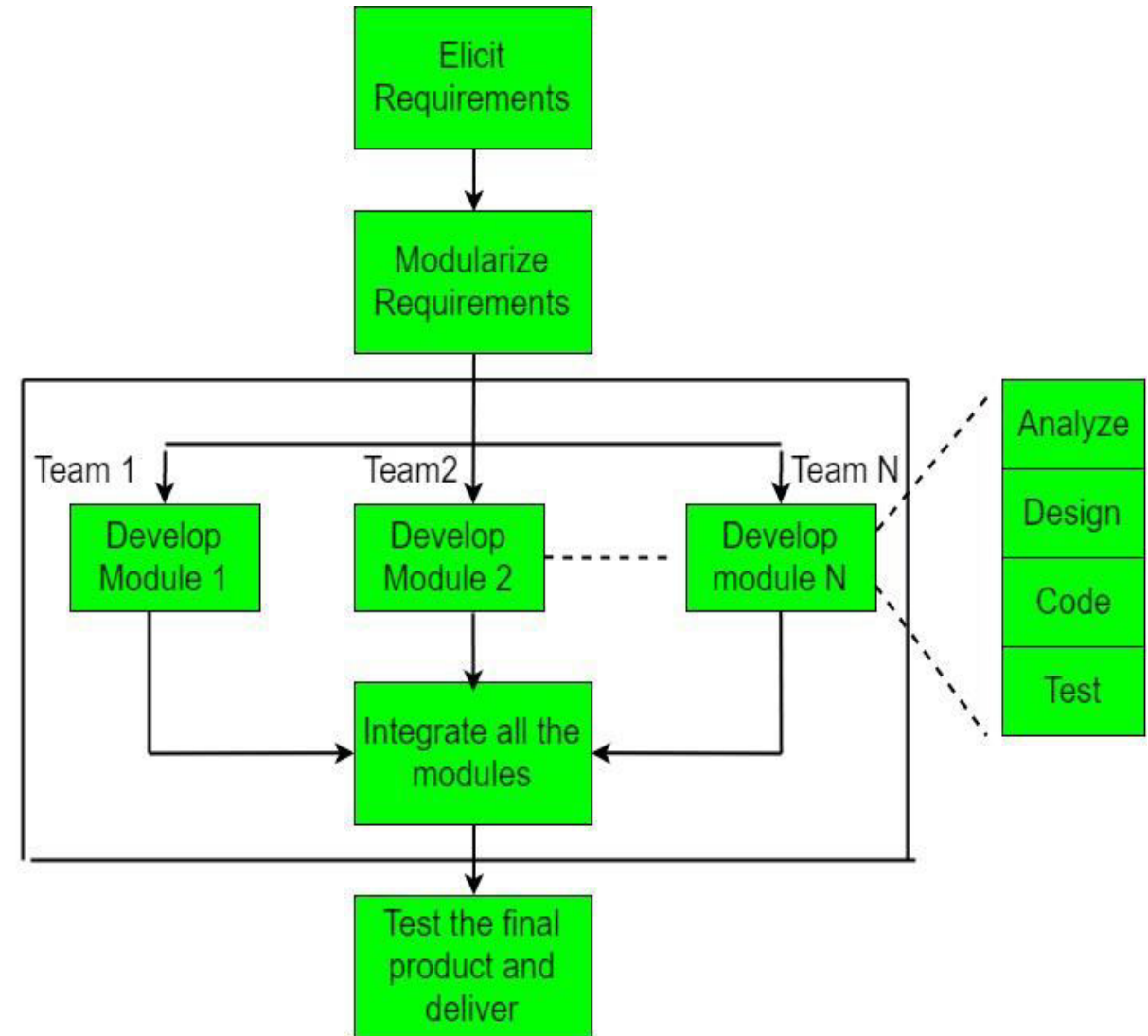
When to use

- A Spiral model in software engineering is used when project is large
- When creation of a prototype is applicable and appropriate
- When costs and risk evaluation is important, Significant changes are expected (research and exploration)
- For medium to high-risk projects.
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs, Requirements are complex, New product line

Software Engineering – RAD Model

RAD is completely different from any other types of engineering because changes can be made almost instantaneously, and it even gives the edge to make amendments at the end of the development process. The key benefit of the rapid application development approach is fast turn out of the project and it really attracts the developers as it is working in a fast-paced environment which is ideal for software development. By minimizing the planning stage and maximizing the prototype development, the rapid pace is achieved in the **RAD Model**

By this approach, it has become easy for the stakeholders and the project managers to accurately measure progress and communicate in real time to focus on changes and take care of error as soon as possible. That brings efficiency, faster development, and effective communication.



Software Engineering – RAD Model

Advantages of Rapid Application Development

- RAD breaks the project into smaller manageable tasks.
- Clients get the best version in a shorter period.
- This model is flexible for change.
- It increases the reusability of features, It reduced development time.
- Regular communication and constant feedback increases the efficiency of the design and build process.

Disadvantages of Rapid Application Development

- In the hurry of completing everything timely, it is very common to leave something important unattended.
- No so cost friendly, depending on the iterations
- It required highly skilled designers.
- All application is not compatible with RAD.
- For smaller projects, we cannot use the RAD model.
- On the high technical risk, it's not suitable.

When to use RAD Model?

- When the system should need to create the project that modularizes in a short span time (2-3 months).
- When the requirements are well-known, When the technical risk is limited.
- When there's a necessity to make a system, which modularized in 2-3 months of period.
- It should be used only if the budget allows the use of automatic code generating tools.

Software Engineering – Agile vs RAD Model

Agile model	RAD model
<p>The Agile model does not recommend developing prototypes but emphasizes the systematic development of each incremental feature at the end of each iteration.</p>	<p>The central theme of RAD is based on designing quick and dirty prototypes, which are then refined into production quality code.</p>
<p>Agile projects logically break down the solution into features that are incrementally developed and delivered.</p>	<p>The developers using the RAD model focus on developing all the features of an application by first doing it badly and then successively improving the code over time.</p>
<p>The Agile team only demonstrate completed work to the customer after each iteration.</p>	<p>Whereas RAD teams demonstrate to customers screen mock up and prototypes, that may be based on simplifications such as table lookup rather than actual computations.</p>
<p>Agile model is not suitable for small projects as it is difficult to divide the project into small parts that can be incrementally developed.</p>	<p>When the company has not developed a almost similar type of project, then it is hard to use RAD model as it is unable to reuse the existing code.</p>

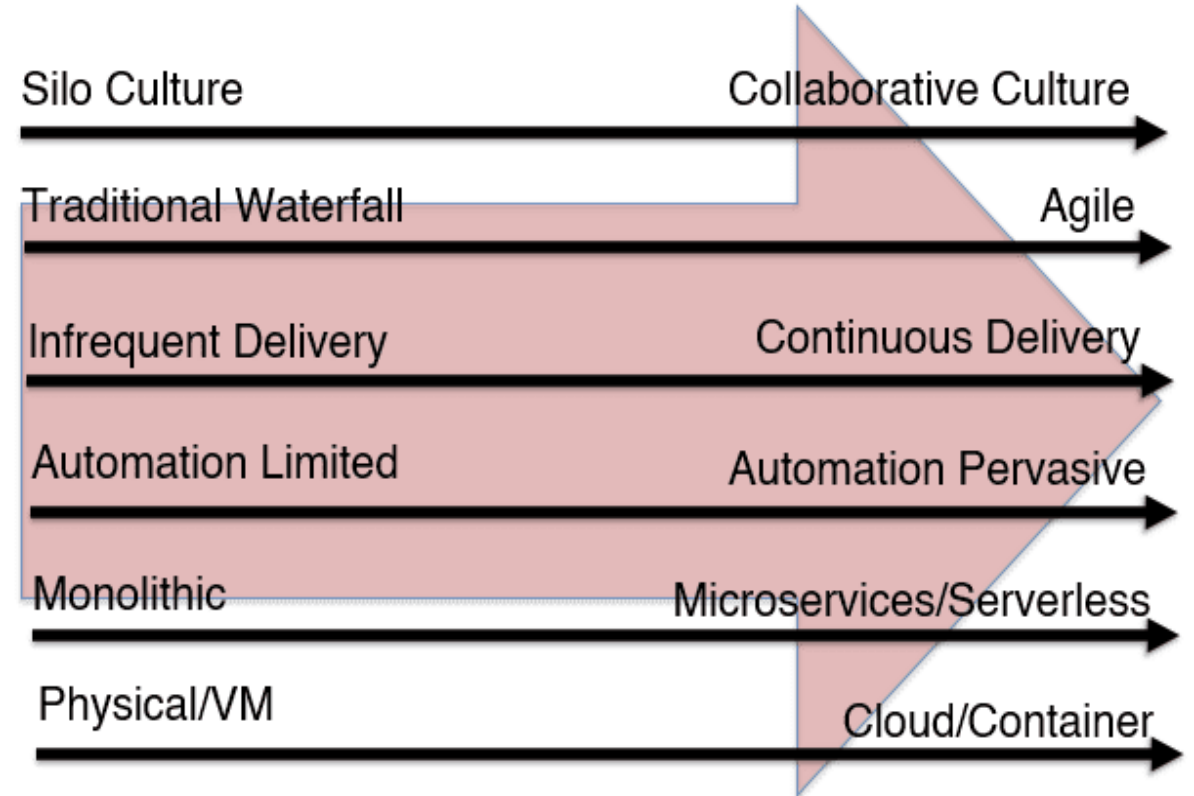
The Shift to Agile Software Development

Every model was created with the purpose of improving the software development and delivery process. Today, every software development model works well for specific types of projects. However, older manual models, like Waterfall, are quickly becoming a thing of the past.

IT teams, and businesses at large, must move faster and more effectively to deliver software, please their end-users, and keep up with the competition. A faster, repeatable, and more secure software development process is anchored in automation.

This level of automation and speed is not attainable with many models. As a result, the Agile methodology has become increasingly popular.

Companies are adopting enabling technologies and processes like continuous integration, continuous delivery, release automation, and DevOps.



Requirement Engineering

What is Requirement ?

- (1) A condition or capability needed by a user to solve a problem or achieve an objective.
- (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.

Type of Requirements –

Business Requirements

Customer Requirements

Functional Requirements

Non-functional Requirements

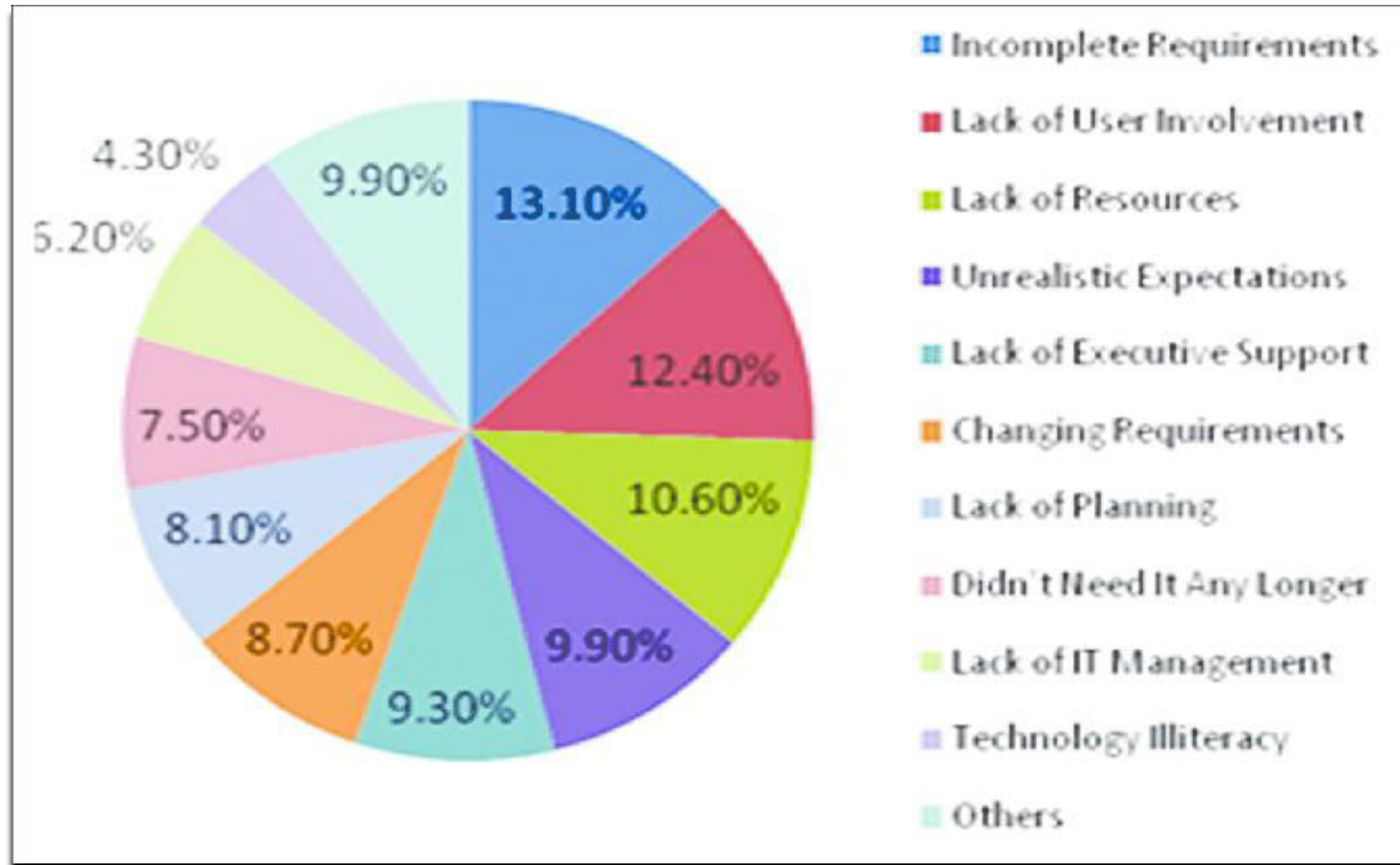
Quality / Testing Requirements

Technology requirement Requirements

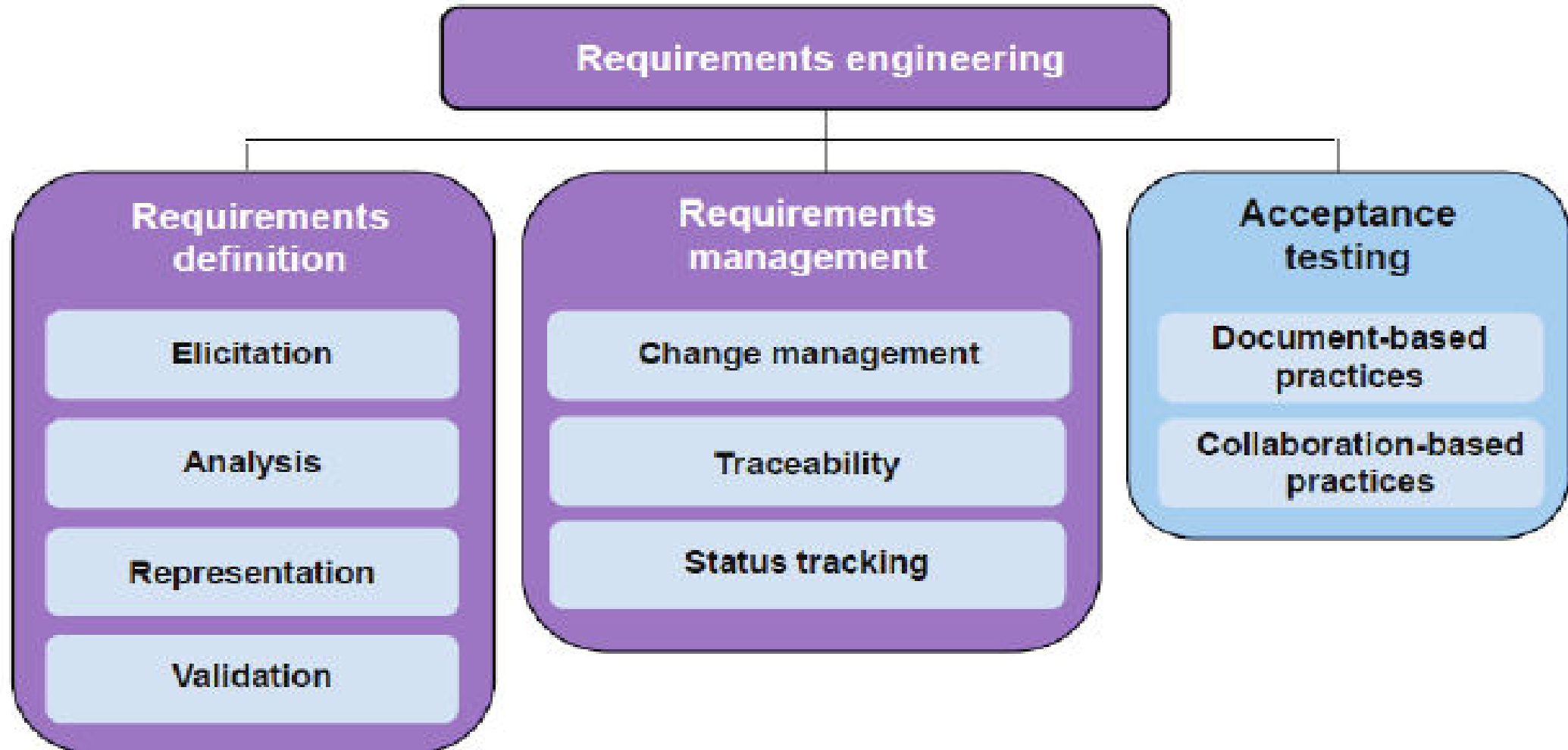
Implementation/Transition Requirements

Requirement Engineering

Why do Projects fail ? – Current Survey



Requirement Engineering



Use Case Elements

Explanation of Use Case Contents

This section will provide explanations for each element of the Use Case.

Name of Use Case: Provide a short name for the use case which should lend itself to the objective of the system.

Description: This section should provide a description of both the reason for using the use case and the expected outcome of the use case.

Actors: Actors may be primary or secondary. Primary actors are the people who will be initiating the system described in the use case. Secondary actors are those will participate in the completion of the use case.

Precondition: This section should describe any conditions that must be true or activities that must be completed prior to executing the use case.

Postcondition: This section should describe the state of the system at the conclusion of the use case. Postconditions may include conditions for both successful and unsuccessful execution of the use case.

Use Case Elements

Explanation of Use Case Contents

Flow: This section should describe all actions of the user and the expected system responses for planned normal execution of the use case. The description should be sequential and provide adequate detail to understand all user actions and system responses.

Alternative Flows: Many use cases have varying or special extensions or conditions which are separate from the main flow but also necessary. Alternative flows are usually the result of options or exceptions built into the use case which may alter the primary flow.

Exceptions: When use cases are executed, there may be various conditions which result in errors. This section should describe any errors that may result during use case execution and how the system will react or respond to those errors.

Requirements: This section should describe any non-functional or special requirements for the system as the use case is executed. These requirements may consist of legal or regulatory requirements, quality standards, or organizational requirements that are outside of the functional requirements the system is expected to perform.

Use Case Diagram -Notations

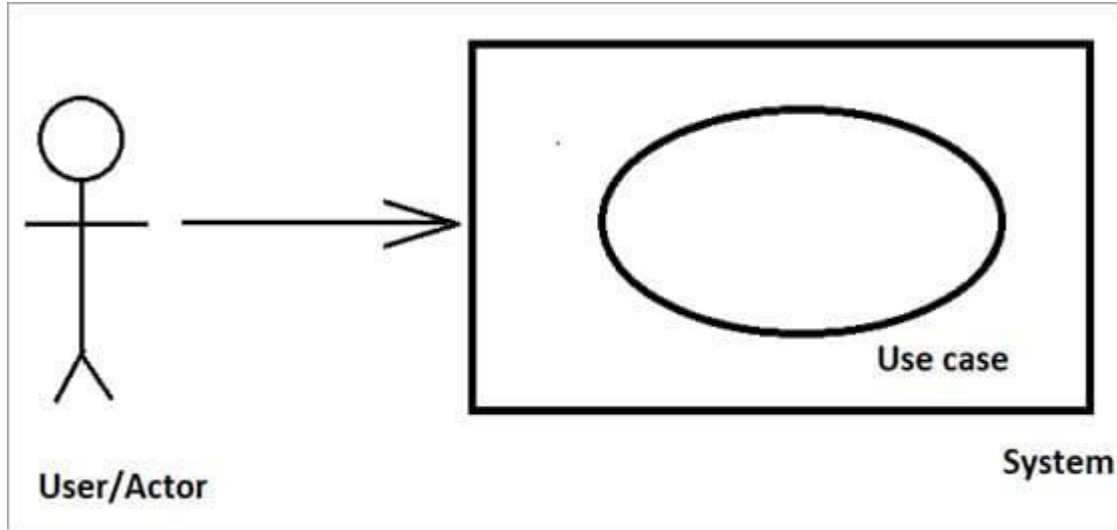
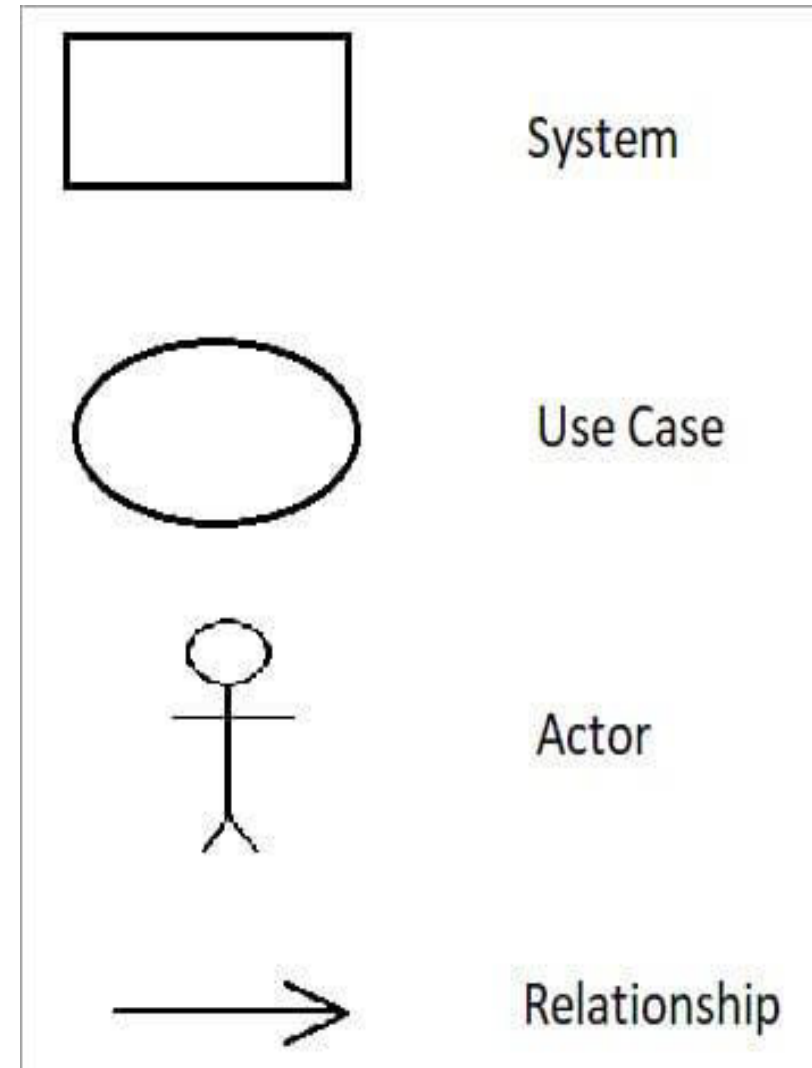


Fig No: UC 01

As shown in the Fig No: UC 01 it represents a diagram where Rectangle represents a 'System', oval represent a 'Use Case', Arrow represents a 'Relationship' and the Man represents a 'User/Actor'. It Shows a system/application, then it shows the organization/people who interact with it and shows the basic flow of 'What the system does?'

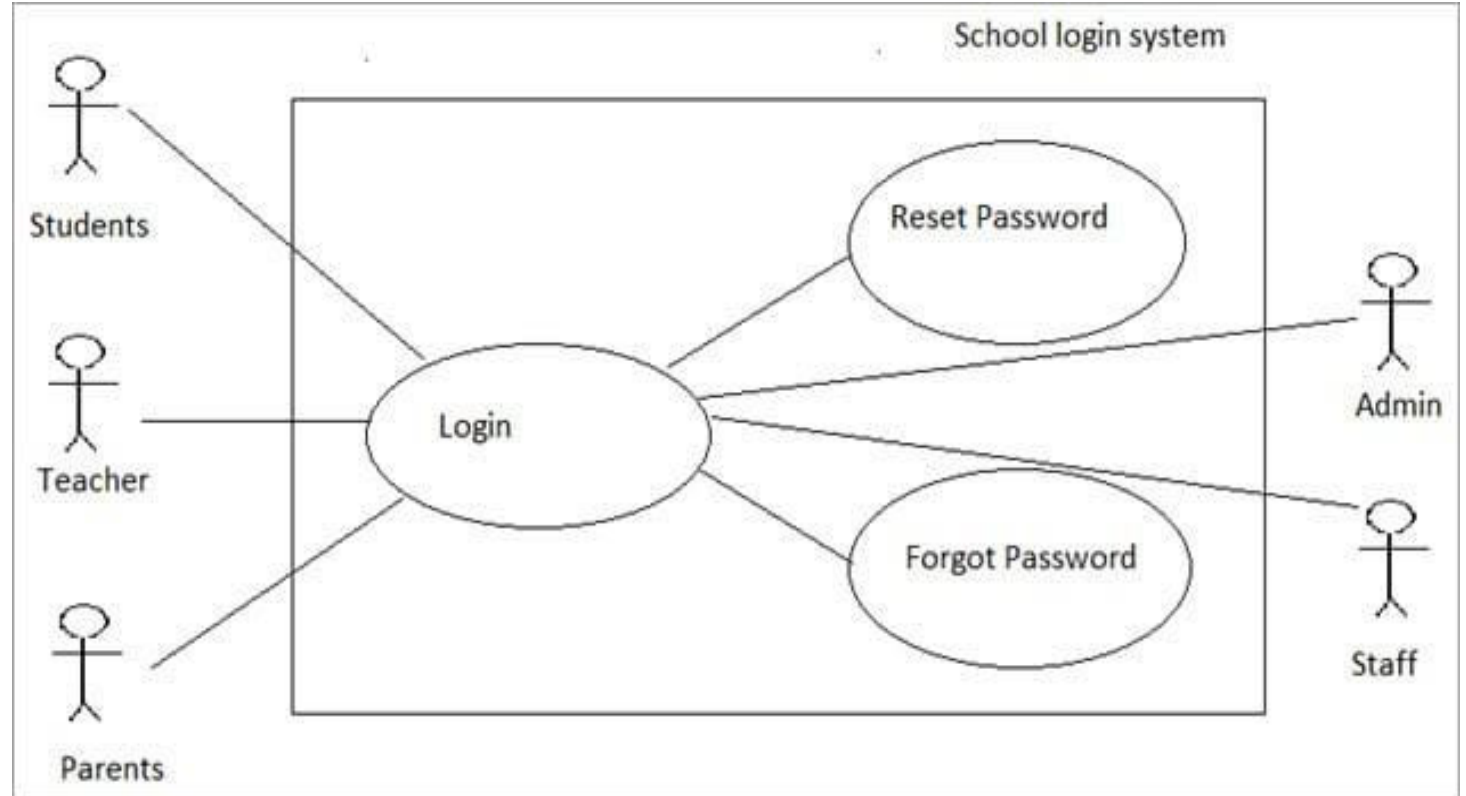


Use case diagram for login

This is the Use case diagram of 'Login' case. Here, we have more than one actor, they are all placed outside the system. Students, teachers, and parents are considered as primary actors. That is why they all are placed on the left side of the rectangle.

Admin and Staff are considered as secondary actors, so we place them on the right side of the rectangle. Actors can log in to the system, so we connect the actors and login case with a connector.

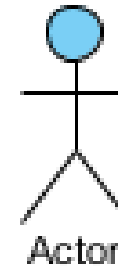
Other functionality found in the system are Reset Password and Forgot password. They are all related to login case, so we connect them to the connector.



Use Case Diagram -Notations

Actor

- Someone interacts with use case (system function).
- Named by noun.
- Actor plays a role in the business
- Similar to the concept of user, but a user can play different roles
- For example:
 - A prof. can be instructor and also researcher
 - plays 2 roles with two systems
- Actor triggers use case(s).
- Actor has a responsibility toward the system (inputs), and Actor has expectations from the system (outputs).



Use Case

- System function (process - automated or manual)
- Named by verb + Noun (or Noun Phrase).
- i.e. Do something
- Each Actor must be linked to a use case, while some use cases may not be linked to actors.



Use Case Diagram -Notations

Communication Link

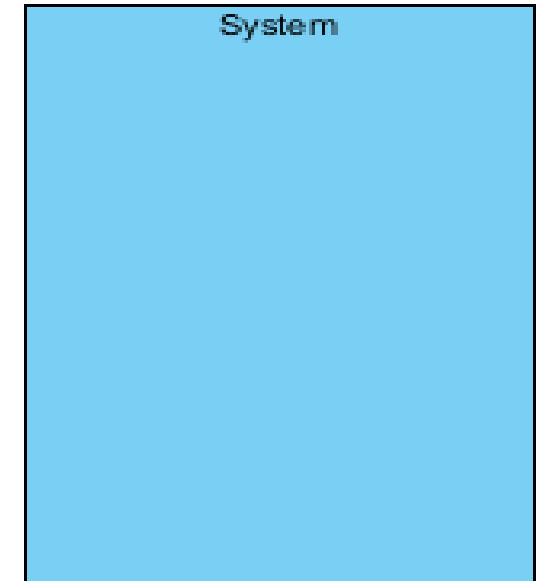
The participation of an actor in a use case is shown by connecting an actor to a use case by a solid link. Actors may be connected to use cases by associations, indicating that the actor and the use case communicate with one another using messages.

Boundary of system

The system boundary is potentially the entire system as defined in the requirements document.

For large and complex systems, each module may be the system boundary. For example, for an ERP system for an organization, each of the modules such as personnel, payroll, accounting can form a system boundary for use cases specific to each of these business functions.

The entire system can span all of these modules depicting the overall system boundary

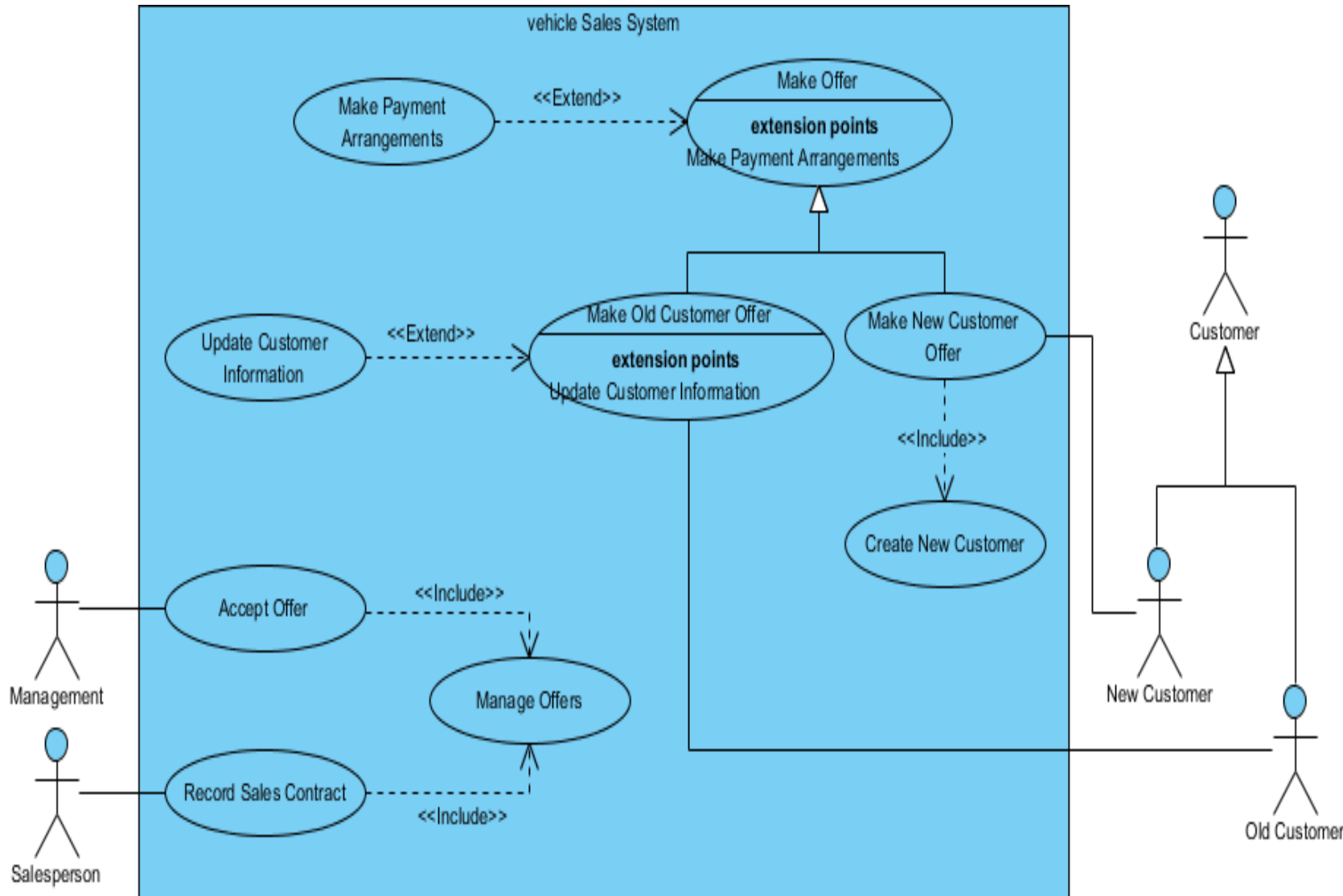


Use Case Example

Login' to a 'School Management System.

Use Case Name	Login	
Use case Description	A user login to System to access the functionality of the system.	
Actors	Parents, Students, Teacher, Admin	
Pre-Condition	System must be connected to the network.	
Post -Condition	After a successful login a notification mail is sent to the User mail id	
Main Scenarios	Serial No	Steps
	1	Enter username Enter Password
	2	Validate Username and Password
	3	Allow access to System
Extensions	1a	Invalid Username System shows an error message
	2b	Invalid Password System shows an error message
	3c	Invalid Password for 4 times Application closed

Use Case Model - Vehicle Sales Systems



Note that:

While a use case itself might drill into a lot of detail about every possibility, a use-case diagram is often used for a higher-level view of the system as blueprints.

It is beneficial to write use cases at a coarser level of granularity with less detail when it's not required.

Requirement Engineering

Software Requirement Specification (SRS) / Requirement Specification

A **software requirement specification (SRS)** is a comprehensive information/description of a product/system to be developed with its functional and non-functional requirements.

The software **requirement specification (SRS)** is developed based on the agreement between customer and supplier. It may include the use cases of how a user is going to interact with the product or software system. SRS helps to reduce the time and effort to develop software. To develop the software system we should clearly understand the Software requirements. To achieve this we need to continuously communicate with clients to gather all related information and requirements.

Characteristics of good SRS Document

- **Concise:** The SRS document should be unambiguous, consistent, and complete.
- **Structured:** A well-structured SRS document is easy to understand and modify.
- **Black-Box View:** SRS should only specify what the system should do and restrict from stating how to do.
- **Conceptual Integrity:** Conceptual integrity in the SRS helps the reader to easily understand it.
- **Traceable:** Traceability means it can tell which code part corresponds to which design component, which test case corresponds to which requirement and which design component corresponds to which requirement etc.
- **Verifiable:** All requirements of the system should be verifiable. This means it should be possible to tell whether or not the requirement has been met as specified in the SRS.

Software Design & Architectural Engineering

Software Design and its Activities

The purpose of Design phase in the Software Development Life Cycle is to produce a solution to a problem (transforming the client requirements) given in the SRS(Software Requirement Specification) document. The output of the design phase is Software Design Document (SDD).

3 Software Design Levels

- 1) Architectural Design
- 2) Preliminary (High-Level) Design
- 3) Detailed Design

Difference between Analysis and Design

The aim of analysis is to understand the problem with a view to eliminate any deficiencies in the requirement specification such as incompleteness, inconsistencies, etc.

The aim of design is to produce a model that will provide a seamless transition to the coding phase, i.e. once the requirements are analyzed and found to be satisfactory, a design model is created which can be easily implemented.

Software Design & Architectural Engineering

Software Design and its Activities

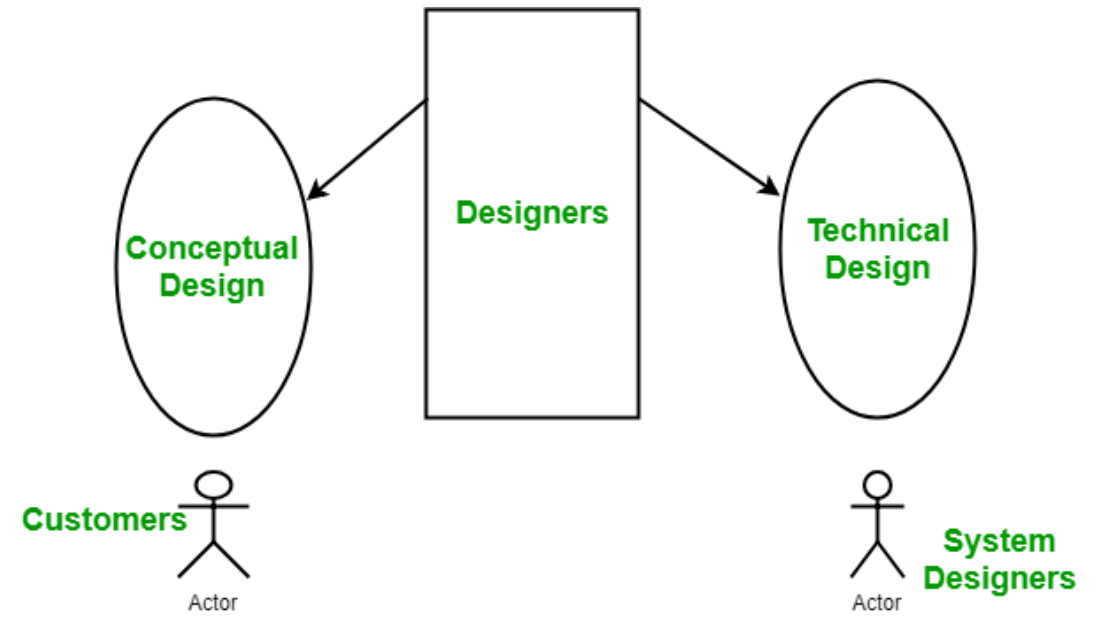
The purpose of Design phase in the Software Development Life Cycle is to produce a solution to a problem (transforming the client requirements) given in the SRS(Software Requirement Specification) document. The output of the design phase is Software Design Document (SDD).

Conceptual Design of the system:

- Written in simple language i.e., customer understandable language.
- Detailed explanation about system characteristics.
- Describes the functionality of the system.
- It is independent of implementation.
- Linked with requirement document.

Technical Design of the system:

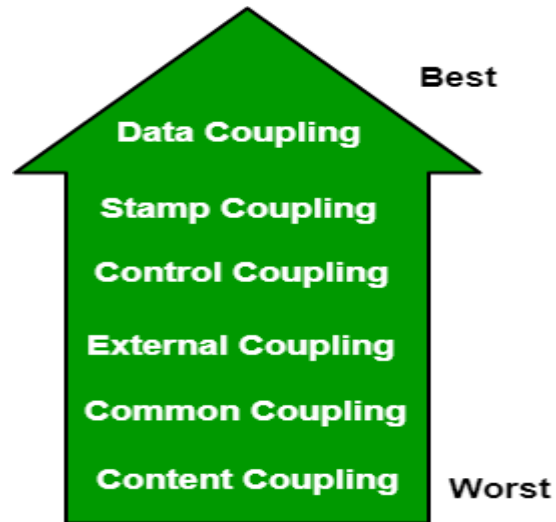
- Hardware component and design.
- Functionality and hierarchy of software components.
- Software architecture
- Network architecture
- Data structure and flow of data.
- I/O component of the system.
- Shows interface.



Software Design & Architectural Engineering

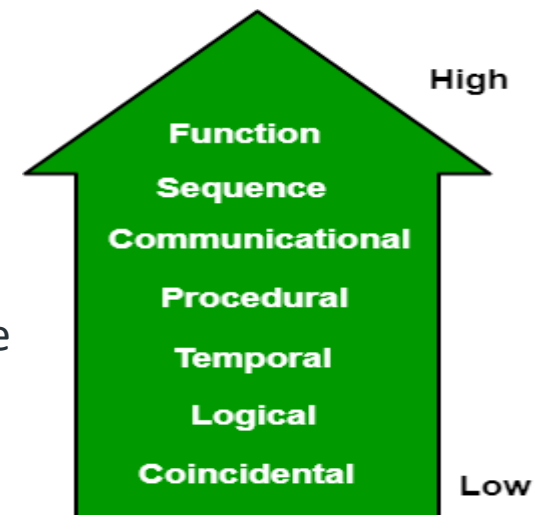
Modularization: Modularization is the process of dividing a software system into multiple independent modules where each module works independently. There are many advantages of Modularization in software engineering. Some of these are given below:

- Easy to understand the system.
- System maintenance is easy.
- A module can be used many times as their requirements. No need to write it again and again.



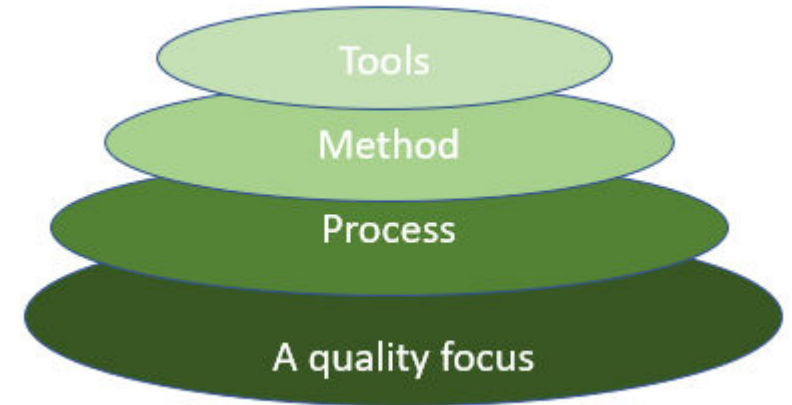
Coupling: Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.

Cohesion: Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.



Software Design & Architectural Engineering

Layering: Software engineering is a fully layered technology, to develop software we need to go from one layer to another. All the layers are connected, and each layer demands the fulfillment of the previous layer



Characteristics of good design

The definition of “a good software design” can vary depending on the application being designed. Most researchers and software engineers agree on a few desirable characteristics that every good software design for a general application must possess. The characteristics are listed below:

Correctness: A good design should correctly implement all the functionalities identified in the SRS document.

Understandability: A good design is easily understandable.

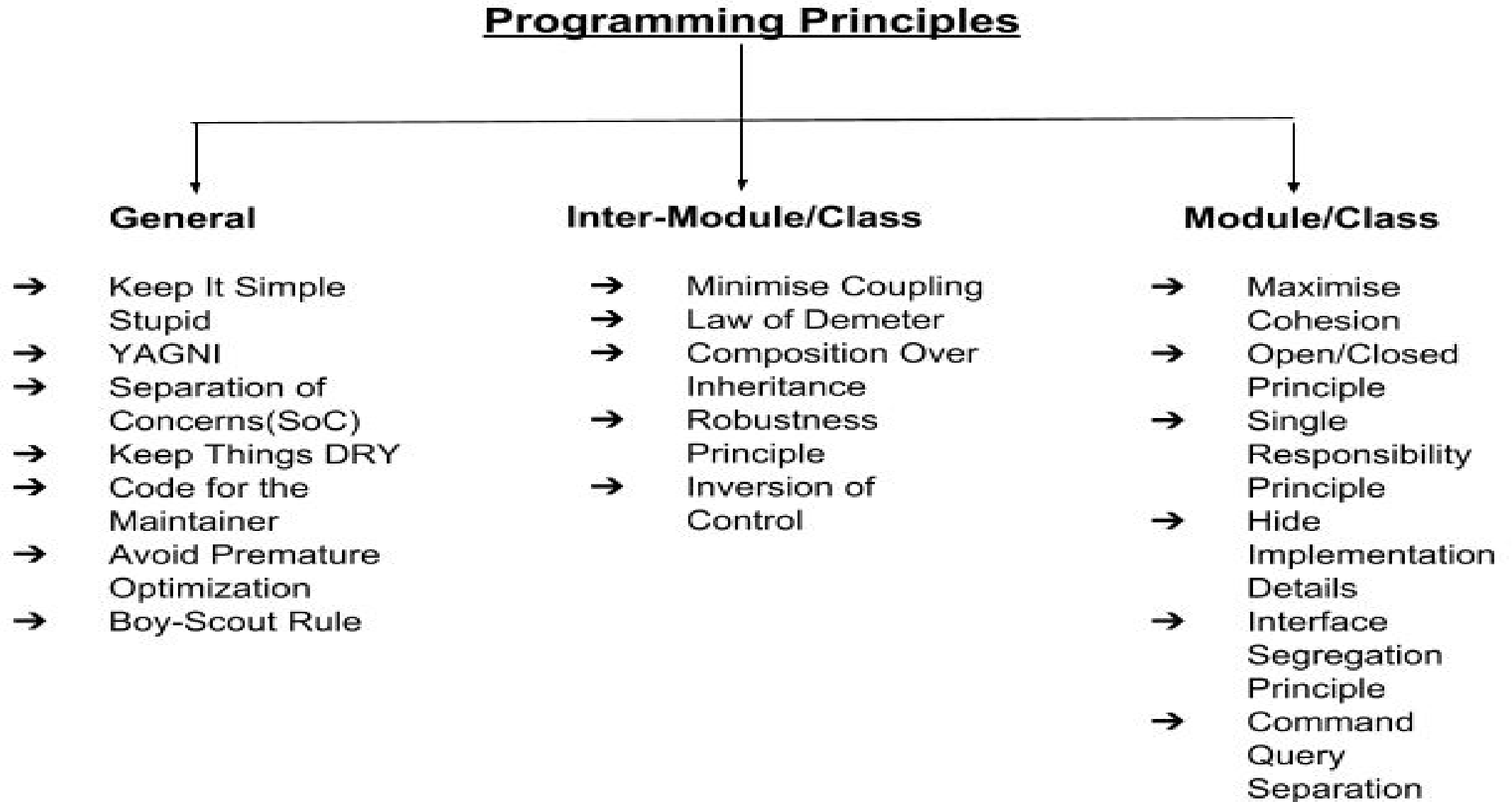
Efficiency: It should be efficient.

Maintainability: It should be easily amenable to change.

Software Design & Architectural Engineering

COMPARISON FACTORS	FUNCTION ORIENTED DESIGN	OBJECT ORIENTED DESIGN
Abstraction	The basic abstractions, which are given to the user, are real world functions.	The basic abstractions are not the real-world functions but are the data abstraction where the real-world entities are represented.
Function	Functions are grouped together by which a higher-level function is obtained.	Function are grouped together on the basis of the data they operate since the classes are associated with their methods.
execute	carried out using structured analysis and structured design i.e, data flow diagram	Carried out using UML
State information	In this approach the state information is often represented in a centralized shared memory.	In this approach the state information is not represented is not represented in a centralized memory but is implemented or distributed among the objects of the system.
Approach	It is a top-down approach.	It is a bottom-up approach.
Begins basis	Begins by considering the use case diagrams and the scenarios.	Begins by identifying objects and classes.
Decompose	In function-oriented design we decompose in function/procedure level.	We decompose in class level.
Use	This approach is mainly used for computation sensitive application.	This approach is mainly used for evolving system which mimics a business or business case.

Programming Principles



Programming Principles

- **Pseudocode**

- Pseudocode is **an informal way of programming description that** does not require any strict programming language syntax or underlying technology considerations.
- It is used for creating an outline or a rough draft of a program. Pseudocode summarizes a program's flow, but excludes underlying details

- E.g.

Get 10,000 records from DB

Filter records based on Category

For loop: Process each record for business algorithm

Business requirement: Check if each record is created during certain period.

If Yes, mark the record as invalid.

End For Loop

UML - Unified Modeling Language (UML)

Design Modeling in Software Engineering

Component level design —

User Interfaces design —

Architectural design —

Data design —



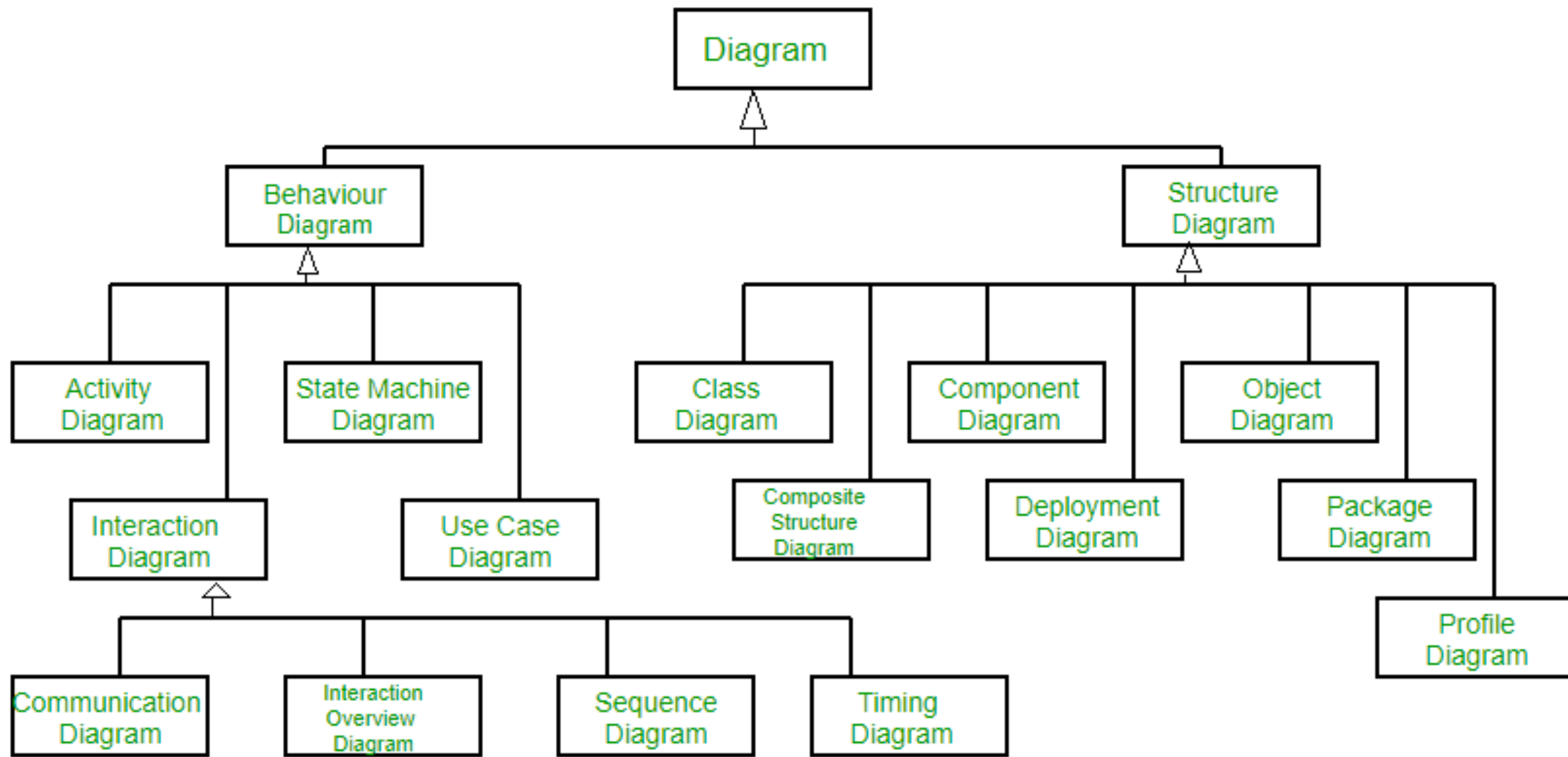
UML - Unified Modeling Language (UML)

This is not a programming language, it is rather a visual language. We use UML diagrams to portray the **behavior and structure** of a system

1. Structural Diagrams – Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.
Describe system's architecture, entities and their properties.

2. Behavior Diagrams – Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.
Describe how the system responds and behaves in different scenarios.

UML



CMM-Capability Maturity Model

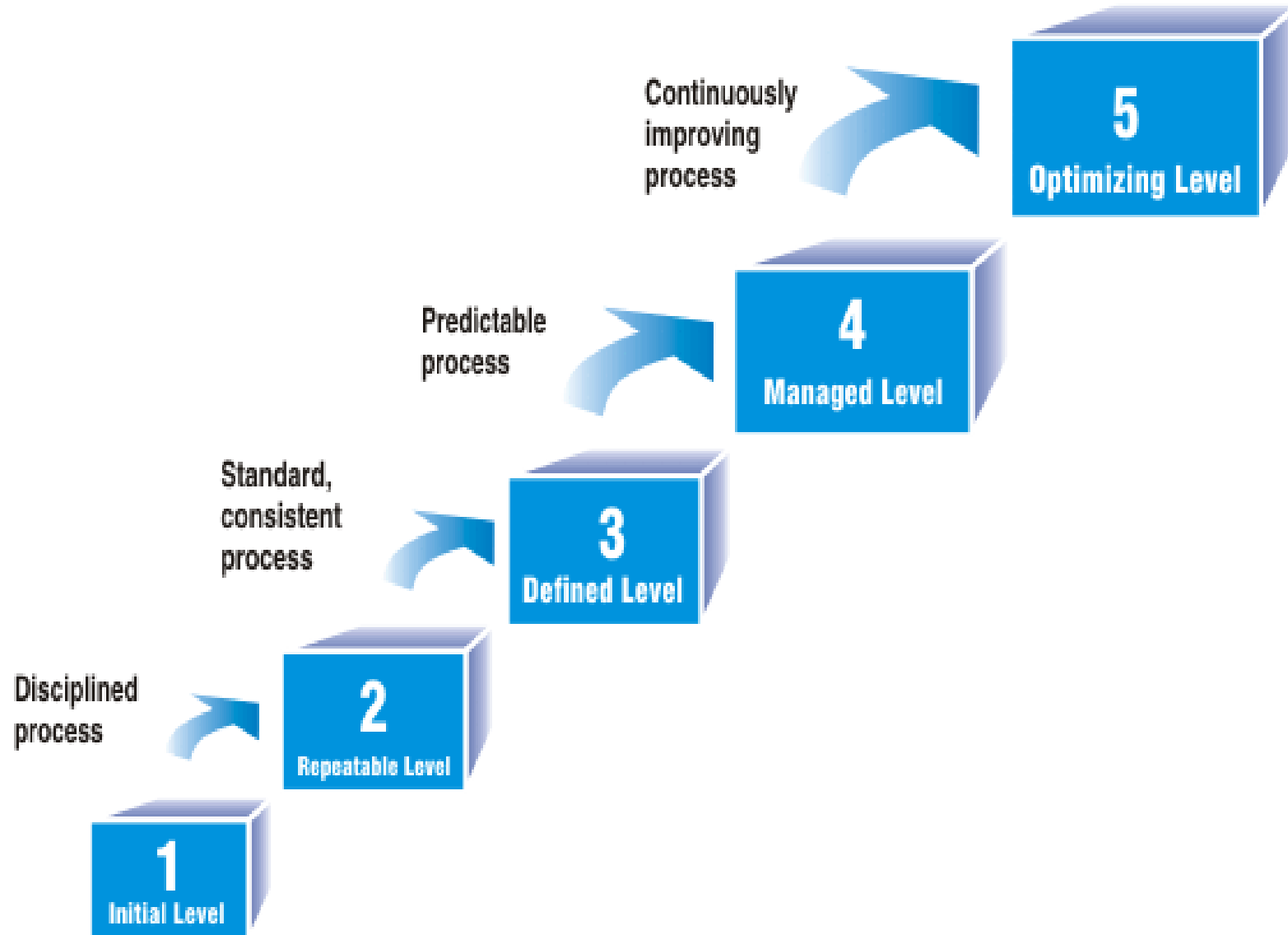
CMM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.

It is not a software process model. It is a framework that is used to analyze the approach and techniques followed by any organization to develop software products.

It also provides guidelines to further enhance the maturity of the process used to develop those software products.

It is based on profound feedback and development practices adopted by the most successful organizations worldwide.

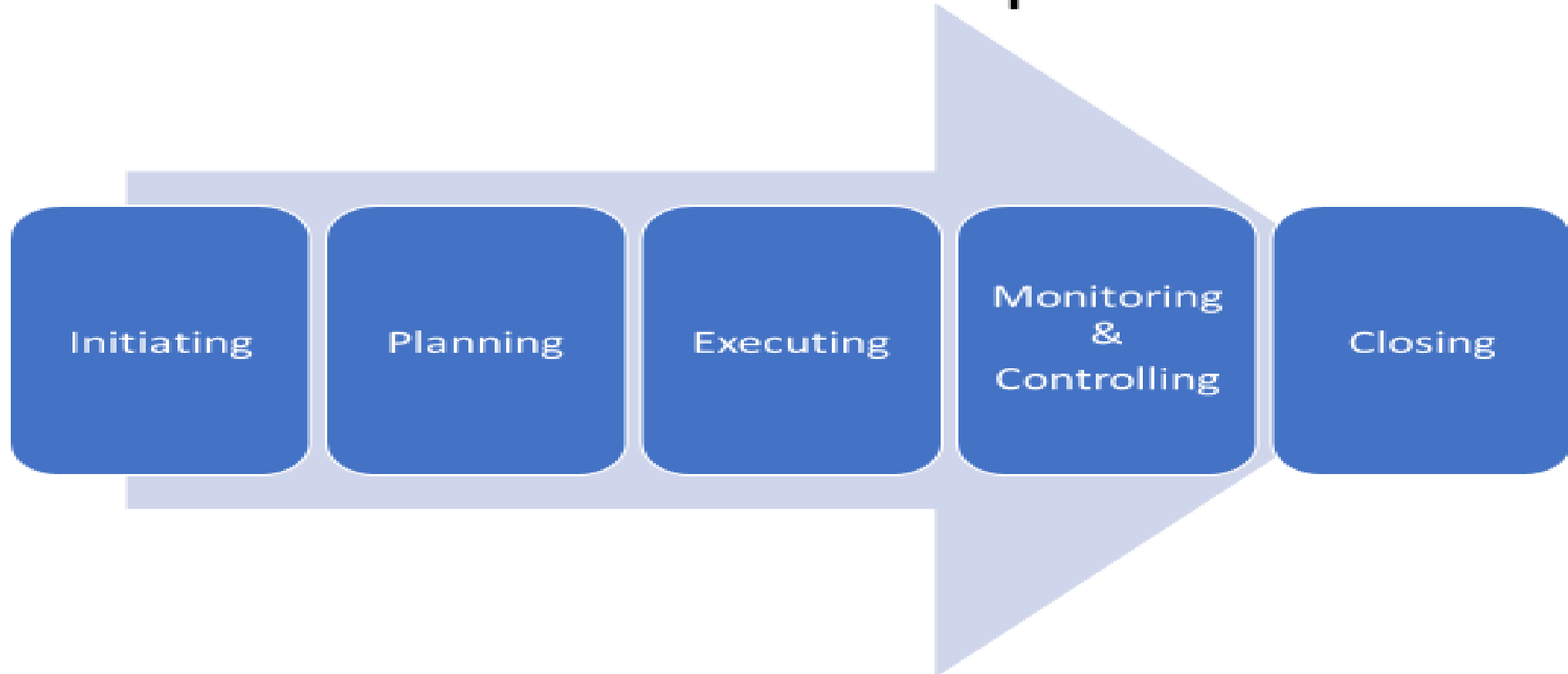
This model describes a strategy for software process improvement that should be followed by moving through 5 different levels.



Software Project Management(SPM)

Project Management is the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements.

Process Groups



Software Estimation Measures

Estimation of the size of the software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation. Some of these are:

- Lines of Code
- Number of entities in ER diagram
- Total number of processes in detailed data flow diagram
- Function points

Software Configuration Management - What

Software Configuration Management(SCM)

SCM is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle.

The primary goal is to increase productivity with minimal mistakes.

SCM is part of cross-disciplinary field of configuration management, and it can accurately determine who made which revision.

Configuration management is considered a subset of systems management, a process for keeping servers, systems, and software functioning consistently within a set of established parameters.

The process ensures the system, and its resources perform as expected, despite updates, additions, and deletions. So, configuration management ensures that all the devices in your network infrastructure march to the same beat, keeping everyone in line.

Software Configuration Management - Features

Enforcement:

With enforcement feature execution daily, ensures that the system is configured to the desired state.

Cooperating Enablement:

This feature helps to make the change configuration throughout the infrastructure with one change.

Version Control Friendly:

With this feature, the user can take their choice of version for their work.

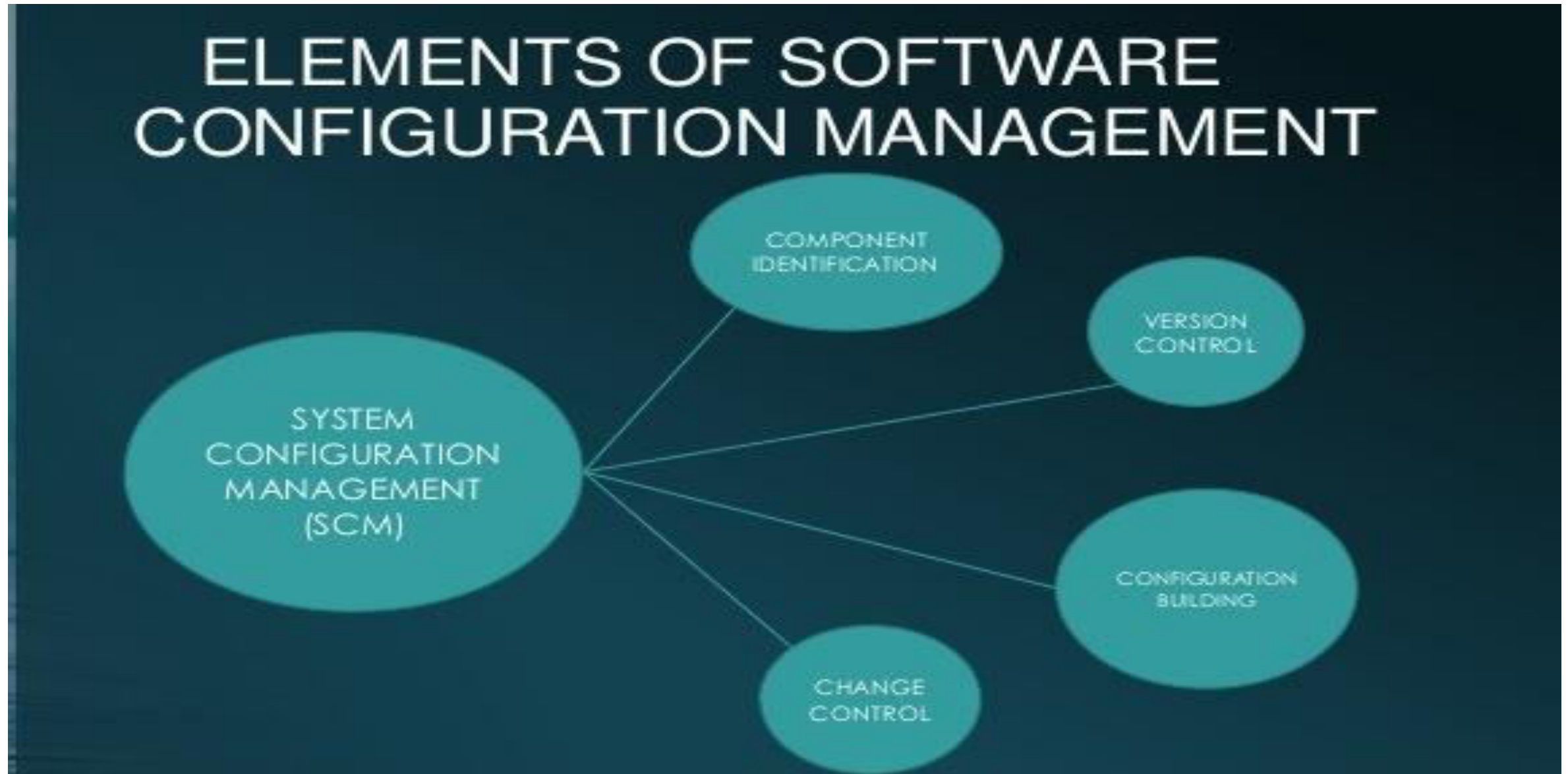
Enable Change Control Processes:

As Software Configuration Management tools are version control and textual friendly, we can make changes in code. Changes can be made as a merge request and send for review.

Tasks in SCM process

1. Planning and Identification Process
2. Version Control Process or Baselines
3. Change Control Process
4. Configuration Release Process
5. Configuration Auditing Process
6. Review and Status Reporting Process

Software Configuration Management - Elements



SDLC – Agile Methodologies

Agile - “quick to move or drive”

Agile Software development methodology.

1. Adaptive Software Development (ASD)
2. Crystal Methods
3. Dynamic Systems Development Method (DSDM)
4. Extreme Programming (XP)
5. Feature-Driven Development (FDD)
6. Lean Software Development (LSD)
7. SCRUM
8. Kanban

Agile – The Manifesto

The agile software development emphasizes on four core values.

Individual and team interactions over processes and tools

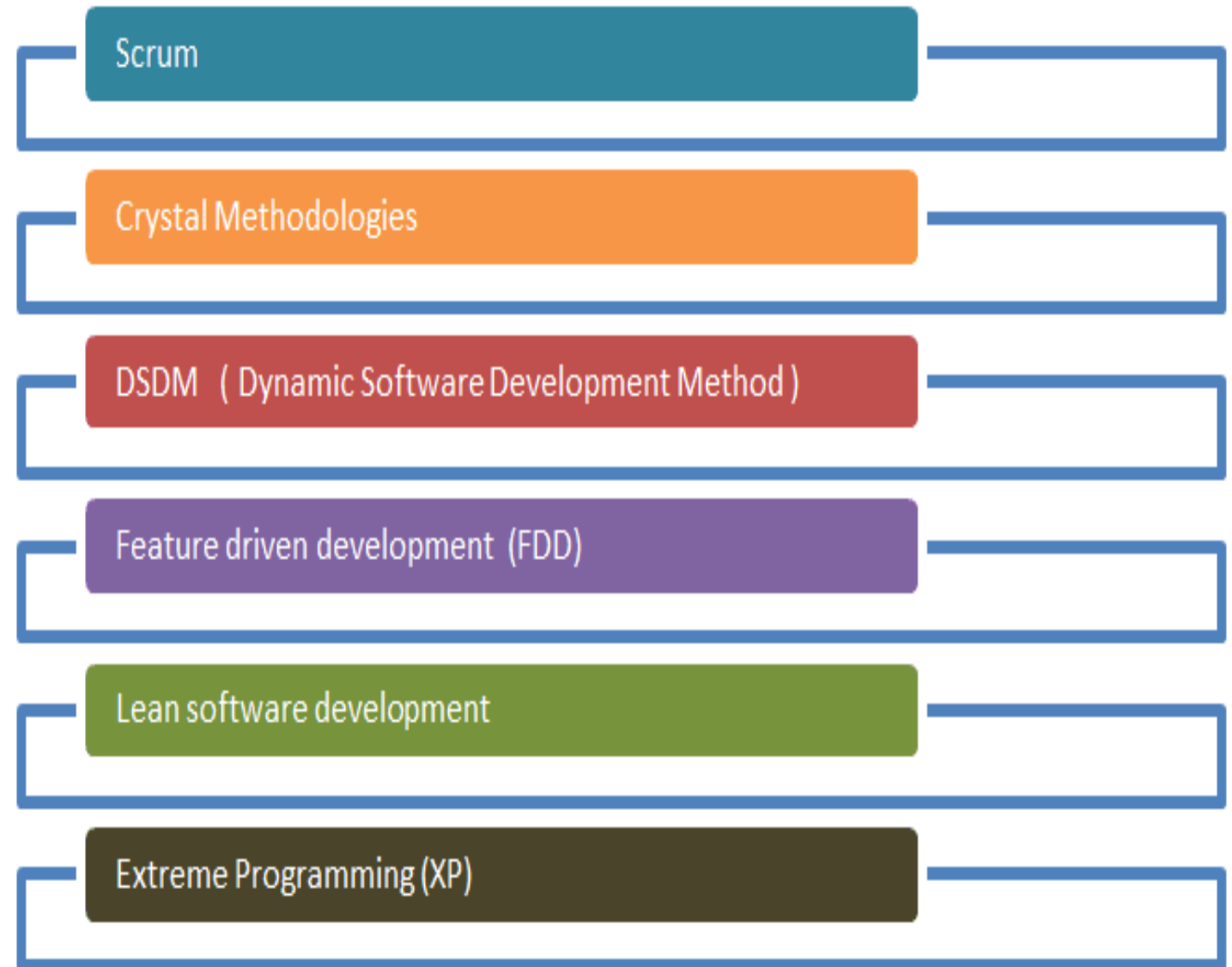
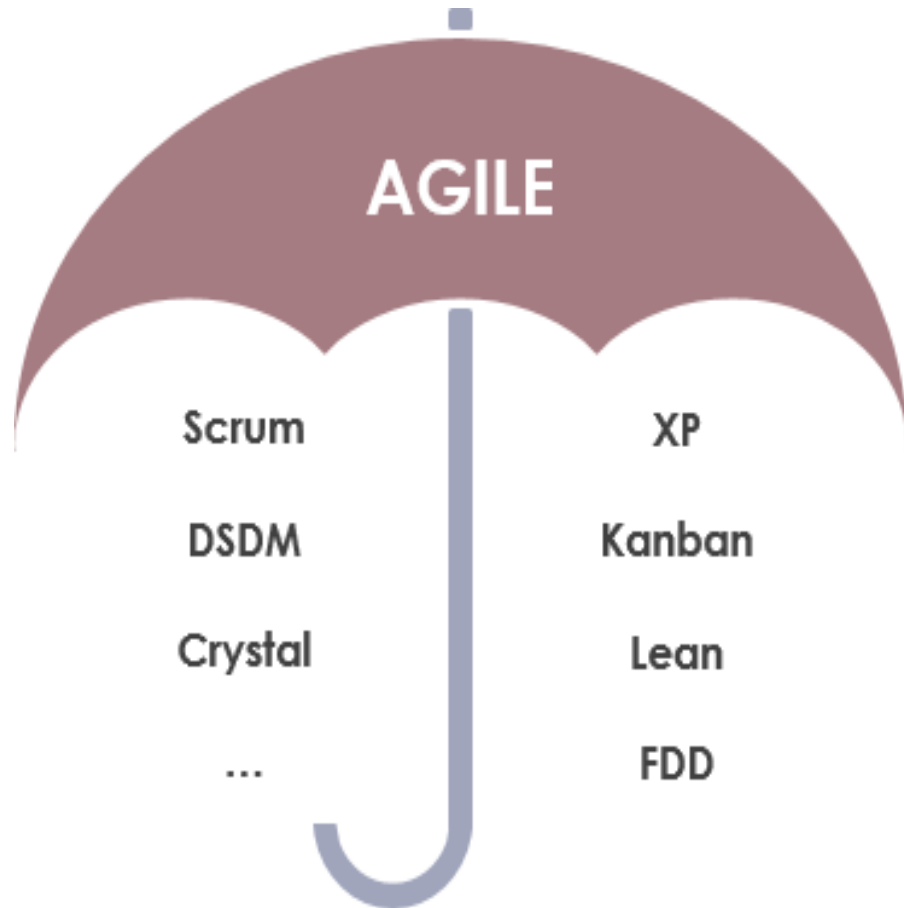
Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan



Agile Methods and Practices

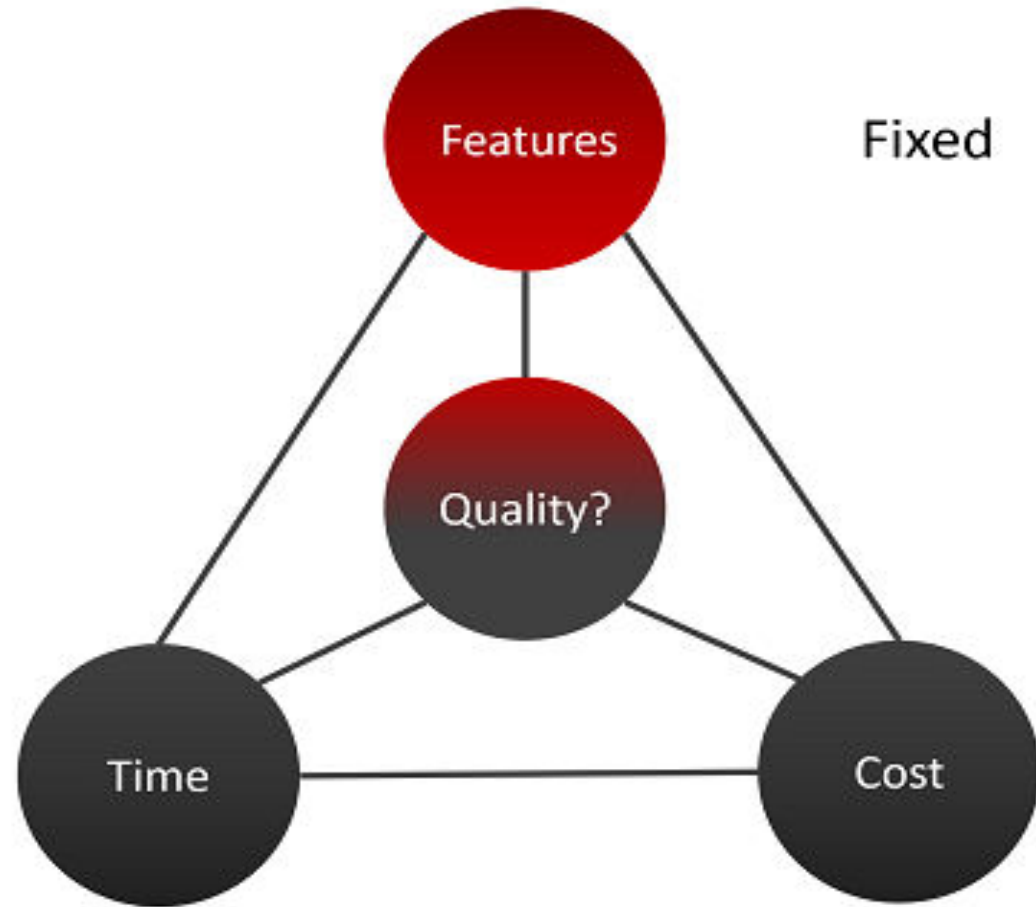


Agile – The Principles

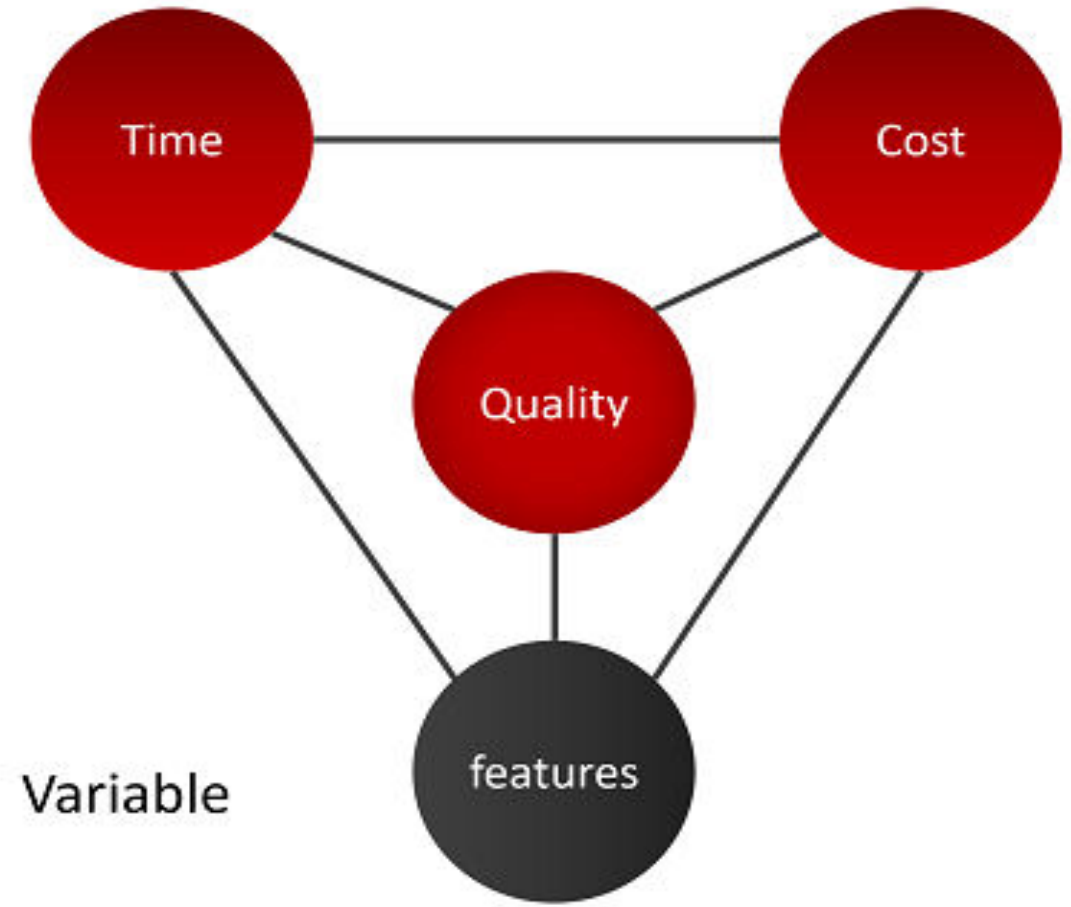
1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Businesspeople 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.

Agile – Approach

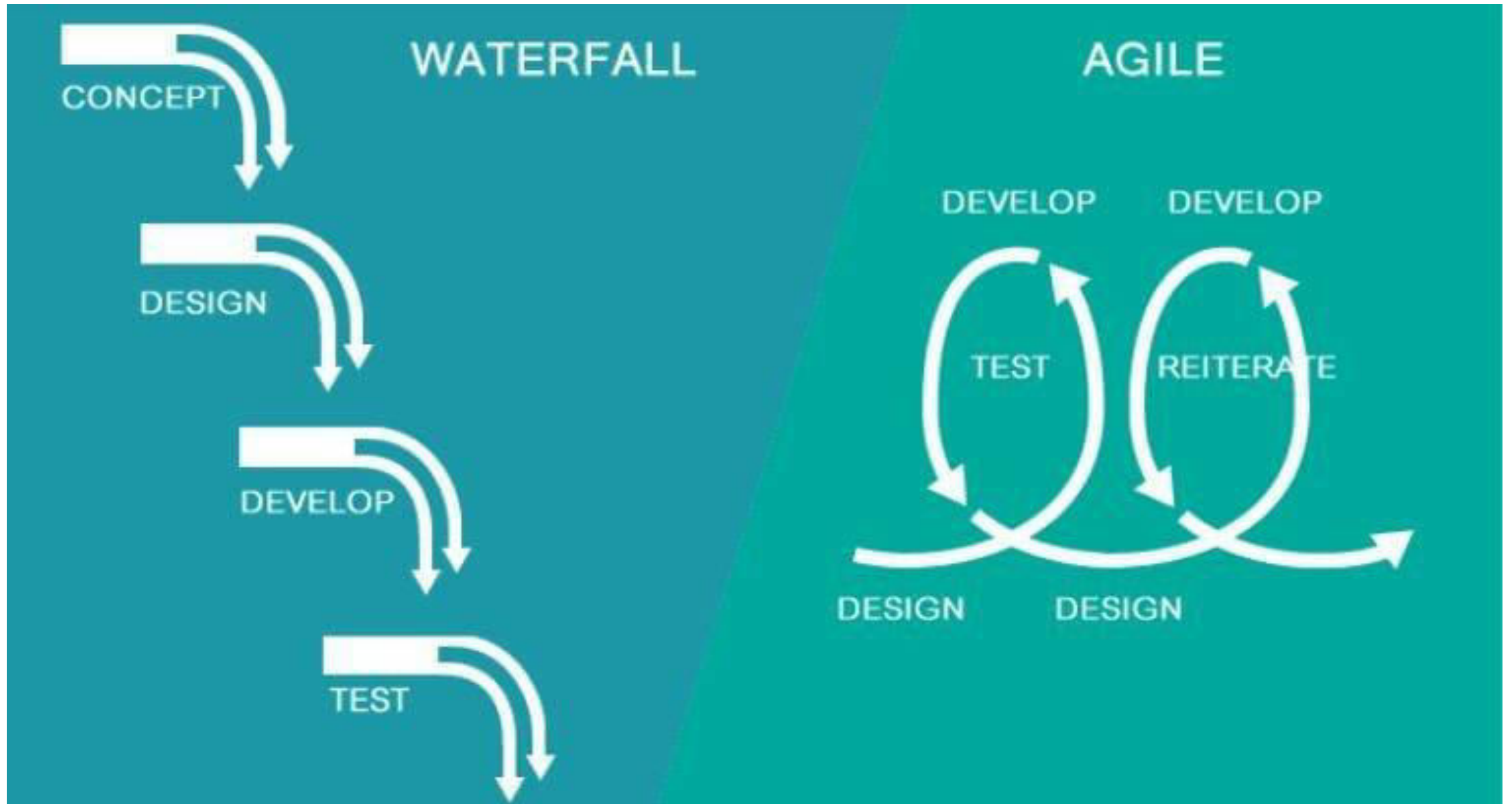
Traditional Approach



Agile Approach



Traditional vs Agile Approach



Agile Methods and Practices

Extreme Programming (XP) –

Extreme Programming (XP) also emphasizes speed and continuous delivery. Like Scrum, XP enables closely-knit teams to deliver working software increments at frequent intervals, usually every 1-3 weeks. It relies on customers to communicate the most useful features of a software product and developers to work towards implementing that feedback.

The five basic component of Extreme Programming are:

- 1.Communication
- 2.Simplicity
- 3.Feedback
- 4.Respect
- 5.Courage

Extreme Programming allow changes in their set timelines.

In Extreme Programming(XP), teamwork for 1-2 weeks only.

Extreme Programming emphasizes strong engineering practices

In Extreme Programming, team must follow a strict priority order or pre-determined priority order

Extreme Programming(XP) can be directly applied to a team. Extreme Programming is also known for its **Ready-to-apply** features.

Agile Methods and Practices

Feature-Driven Development (FDD) –

Feature-Driven Development, or FDD, provides a framework for product development that starts with an overall model and gets progressively more granular. Like other Agile methodologies, FDD aims to deliver working software quickly in a repeatable way. It uses the concept of “just enough design initially” (JEDI) to do so, leveraging two-week increments to run “plan by feature, design by feature, build by feature” iterations.

Organizations that practice Agile like Feature-Driven Development for its feature-centric approach and its scalability.

Five Step Process -

- Developing an overall model

- Building the feature list

- Planning by feature

- Designing by feature

- Building by feature

FDD is amazing for big projects and is quite scalable and prone to get achieve success.

FDD is very effective in helping with complex projects that are in a critical situation.

Agile Methods and Practices

Lean Software Development –

Lean software development is more flexible than Scrum or XP, with fewer strict guidelines, rules, or methods. Lean is based on a set of principles developed to ensure value and efficiency in production in the mid 20th century and has evolved into the software setting. Lean relies on five principles of Lean management:

1. Identify value
2. Value stream mapping
3. Create continuous workflow
4. Create a pull system
5. Continuous improvement

Lean particularly emphasizes eliminating waste. In the context of software development, that includes cutting out wasted time and unproductive tasks, efficiently using team resources, giving teams and individuals decision-making authority, and prioritizing only the features of a system that deliver true value.

Agile Methods and Practices

Kanban –

Kanban focuses on helping teams work together more effectively to enable continuous delivery of quality products. Kanban is unique, however, for offering a highly visual method for actively managing the creation of products.

The Kanban Agile methodology relies on six fundamental practices:

1. Visualize the workflow
2. Limit work in progress
3. Manage flow
4. Make process policies explicit
5. Implement feedback loops
6. Improve collaboratively

Kanban achieves these practices using a Kanban board. The Kanban board facilitates the visual approach to Agile using columns to represent work that is To Do, Doing, and Done. This Agile methodology improves collaboration and efficiency and helps define the best possible team workflow.

To Do, In Progress, In Review, and Done (or Completed).

The most important metrics are lead time and cycle time. Each of these metrics measures the average amount of time it takes for tasks to move through the board.

Agile – Scrum Vs Kanban

	Scrum	Kanban
Cadence	Regular fixed length sprints (i.e., 2 weeks)	Continuous flow
Release methodology	At the end of each sprint	Continuous delivery
Roles	Product owner, scrum master, development team	No required roles
Key metrics	Velocity	Lead time, cycle time, WIP
Change philosophy	Teams should not make changes during the sprint.	Change can happen at any time
Approach	A structured agile approach	Continuous improvement, flexible processes

Agile Methods and Practices

Dynamic Systems Development Method (DSDM)

The Dynamic Systems Development Method (DSDM) rounds out our list of well-known Agile methodologies. DSDM originated in the 1990s to provide a common industry framework for rapid software delivery. Today, it has matured into a comprehensive

Agile methodology that revolves around:

- Business needs and value
- Active user involvement
- Empowered teams
- Frequent delivery
- Integrated testing
- Stakeholder collaboration

The DSDM framework is particularly useful for prioritizing requirements. It also mandates that rework is to be expected, so any development changes must be reversible. DSDM relies on sprints, similar to other Agile methodologies, and is often used in conjunction with approaches like Scrum and XP.

Agile Methods and Practices

Crystal -

The Crystal Agile methodology focuses more on the interactions of the people involved in a project versus the tools and techniques of development. A lightweight model, Crystal emphasizes interaction, people, community, skills, communications, and talents.

Crystal categorizes projects based on three criteria:

- 1.Team size
- 2.System criticality
- 3.Project priorities

The approach is similar to other Agile methodologies in its attention to early and often delivery of software, high involvement of users, and removal of red tape. Crystal's assertion that every project is unique, however, has led to its reputation as one of the most flexible Agile methodologies.

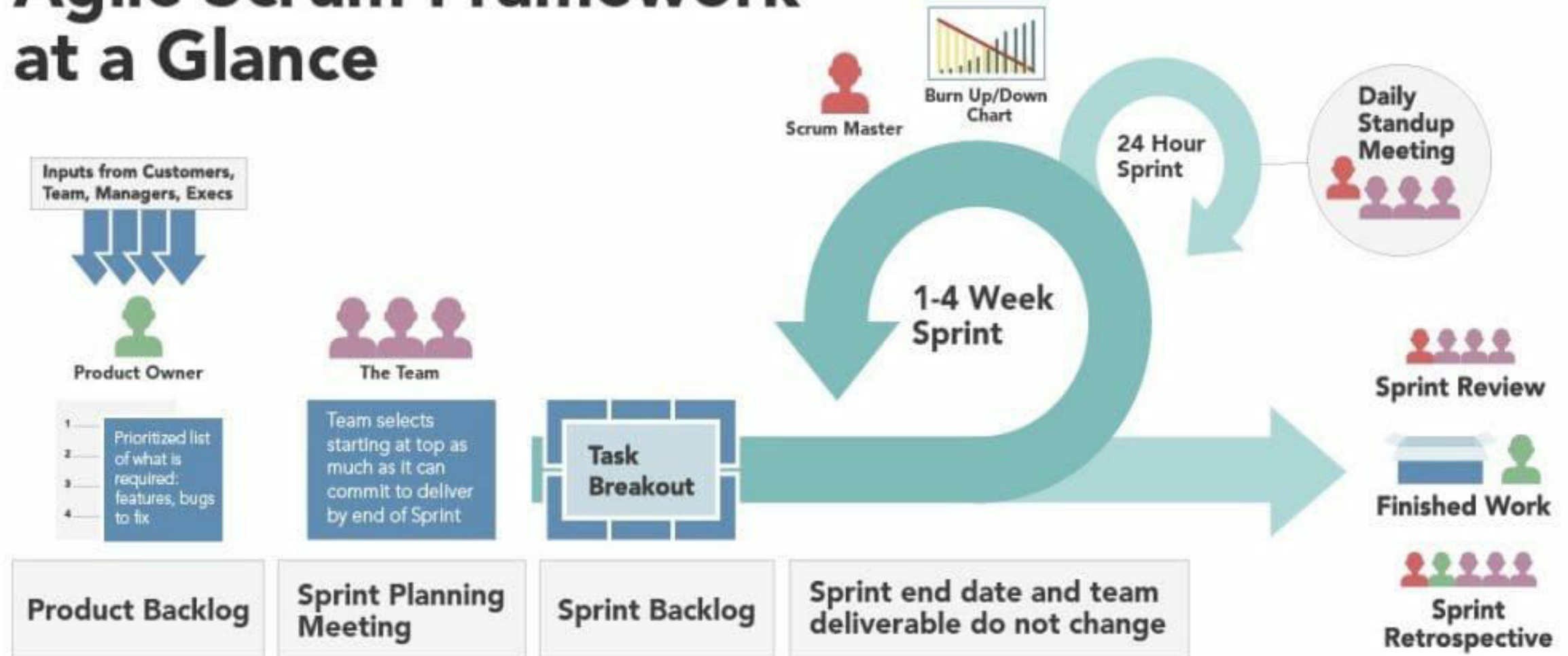
Agile Methods and Practices

Scrum

SCRUM is an agile development method which concentrates specifically on how to manage tasks within a team-based development environment. Basically, Scrum is derived from activity that occurs during a rugby match. Scrum believes in empowering the development team and advocates working in small teams (say- 7 to 9 members). Agile and Scrum consist of three roles, and their responsibilities are explained as follows:

Agile – Scrum Framework

Agile Scrum Framework at a Glance





Agile Concepts –
I know Agile concepts.

All set for Applied Agile.



-
- How do I apply my agile skills ?
 - How do I Manage ?
-

Which Tool will help me ?



Not to Worry.

We have Many Tools
available in Market.

Trello, Wrike, Teamwork,
ClickUp, Asana, Wrike etc.

Let's go for most widely
used tool called - **JIRA**



What is JIRA?

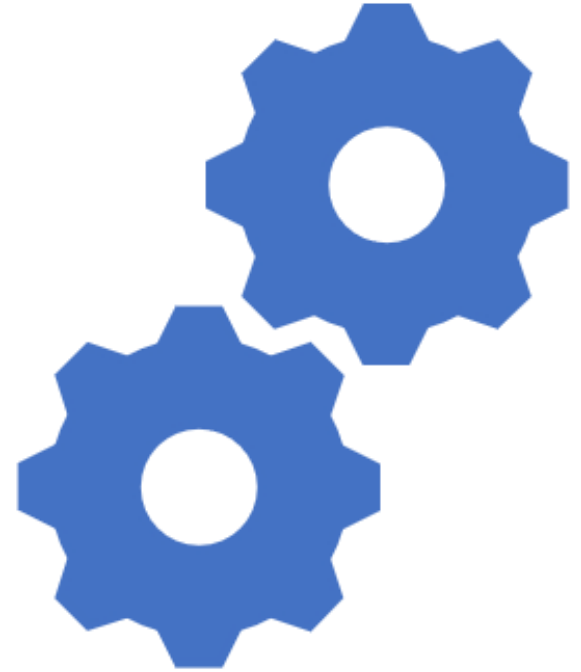


JIRA is powerful Work management Tool.

Jira is great at helping your team plan and track all the work.



It can help teams work faster, communicate more effectively due to its well-managed workflow, Mapping and issue tracking ability with minimal effort.




Jira – Default Roles

- Administrators (people who administer a given project).
- Developers (people who work on issues in each project).
- Users (people who log issues on a given project).



Roles

JIRA enables you to allocate particular people to specific roles in your project. Roles are used when defining other settings, like notifications and permissions.



- Project Lead:  Bryan Rollins 
- Default Assignee: Unassigned 

Project Roles

Users


Groups


Administrators

 Bryan Rollins  Christina
 David


 administrators

Developers

 Penny  Scott

 developers

Users

 Michal Orzechowski

 users

Jira - Activities



Configure Boards

Configure and Generate Reports

Get Active Sprint Data

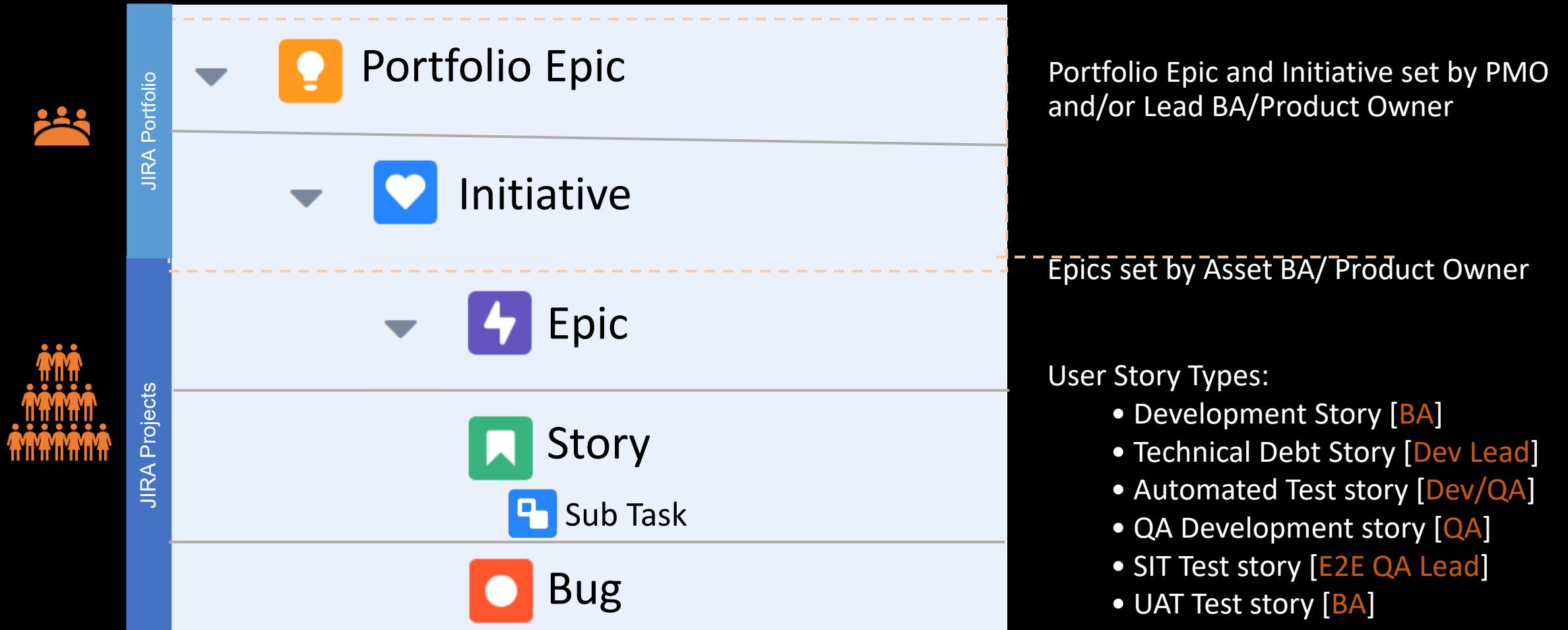
Manage Backlog

Plan and Manage Sprints

Plan and Manage Stories (User Story, Bug)

Setup Project

JIRA Hierarchy and Story definition



No Spike or Chore Issue types

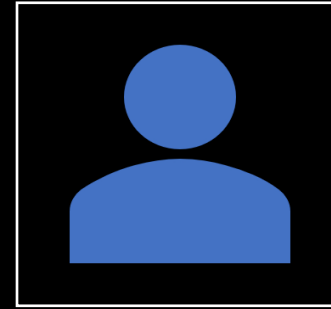
Plan and Manage Sprints



How to create
Sprint



How to Start
Sprint



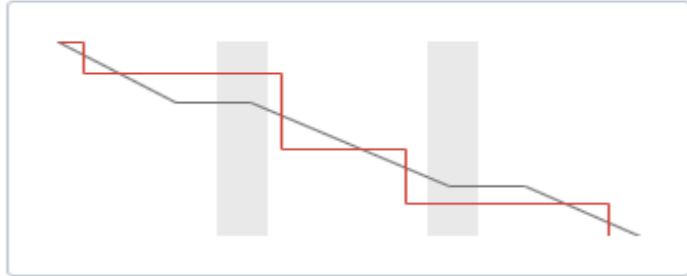
How to Manage
Sprint



How to Manage
Sprint Backlog

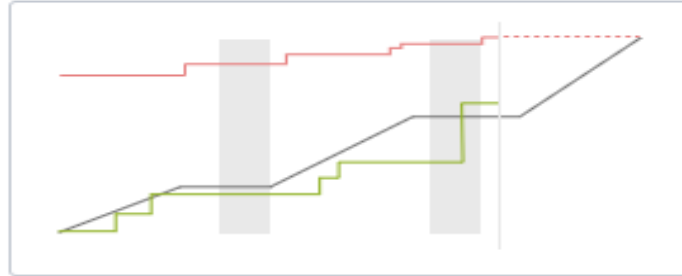
Jira – Configure and Generate Reports

Agile



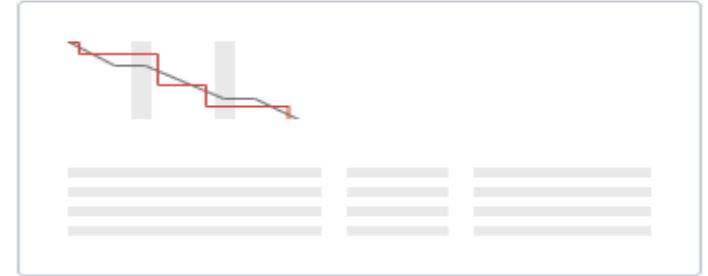
Burndown Chart

Track the total work remaining and project the likelihood of achieving the sprint goal. This helps your team manage its progress and respond accordingly.



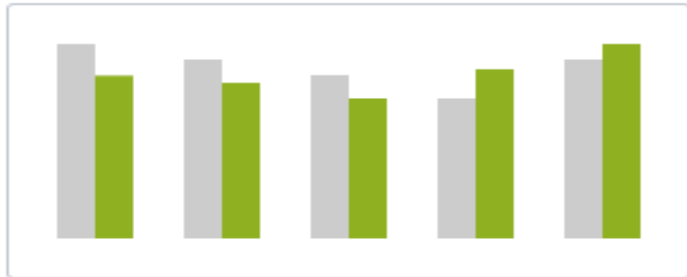
Burnup Chart

Track the total scope independently from the total work done. This helps your team manage its progress and better understand the effect of scope change.

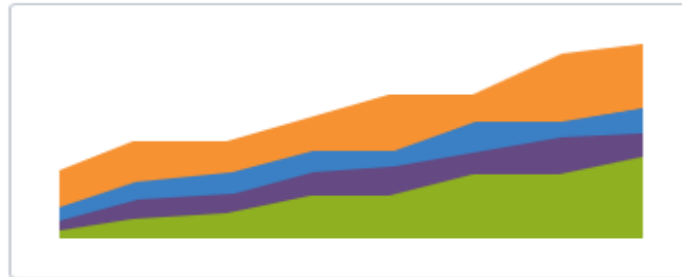


Sprint Report

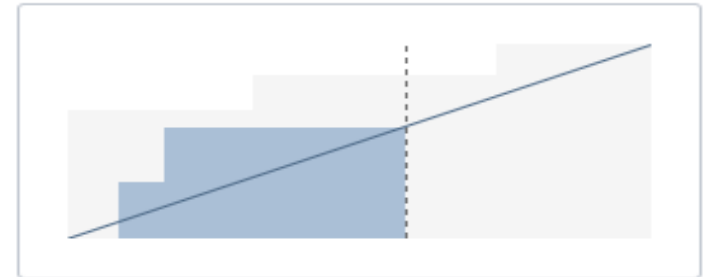
Understand the work completed or pushed back to the backlog in each sprint. This helps you determine if your team is overcommitting or if there is excessive scope creep.



Velocity Chart



Cumulative Flow Diagram



Version Report

Software Testing– An Introduction

Software testing is a process of checking software applications and products is Bugs/Defect free to ensure their performance is efficient. And whether the actual software product matches expected requirements.

It is more important to demonstrate that the software is not doing what it is not supposed to do.

Testing in software engineering is a fundamental process of creating reliable – and usable – software products

The goal of software testing is to find errors, gaps, or missing requirements in comparison to the actual requirements

Testing is a DESTRUCTIVE PROCESS : A CREATIVE DESTRUCTION.

Testing Need a Negative approach

Successful testing requires a methodical approach

Establishing the proper “Test to Break” mental attitude has a profound effect on testing success.

Software Testing - Principles

7 Principles Of Software Testing

Testing Shows the Presence of Defects -

Every application or product is released into production after enough testing by different teams or passes through different phases like System Integration Testing, User Acceptance Testing, and Beta Testing etc.

So, have you ever seen or heard from any of the testing team that they have tested the software fully and there is no defect in the software? Instead of that, every testing team confirms that the software meets all business requirements, and it is functioning as per the needs of the end user.

In the software testing industry, no one will say that there is **no defect** in the software, which is quite true as testing cannot prove that the software is error-free or defect-free.

However, the objective of testing is to find more and more hidden defects using different techniques and methods.

Testing can reveal undiscovered defects and if no defects are found then it does not mean that the software is defect free.

Early Testing –

Testers need to get involved at an early stage of the Software Development Life Cycle (SDLC). Thus, the defects during the requirement analysis phase or any documentation defects can be identified. The cost involved in fixing such defects is very less when compared to those that are found during the later stages of testing.

Software Testing - Importance

Testing is important because software bugs could be expensive or even dangerous.

Software testing is required to identify errors >> Reduce flaws in the component or system.

Software testing is required to Increase the overall quality of the system >> Gain customer. confidence, Satisfaction.

Software testing is required to check software adaptability >> To accelerate software development and adding new features.

Software testing is required to avoid risks.

Software testing is required to avoid extra costs >> To optimise business.

Software testing is required to determining Software performance.

Software testing is required to determining Software Security.

Software testing is required to check the reliability of the software.

Software testing is required to make sure that the final product is user friendly.

Software Testing - Principles

Exhaustive Testing is Not Possible –

It is not possible to test all the functionalities with all valid and invalid combinations of input data during actual testing. Instead of this approach, testing of a few combinations is considered based on priority using different techniques. Exhaustive testing is not possible. Instead, we need the optimal amount of testing based on the risk assessment of the application.

Testing is Context-Dependent –

There are several domains available in the market like Banking, Insurance, Medical, Travel, Advertisement etc. and each domain has several applications. Also, for each domain, their applications have different requirements, functions, different testing purpose, risk, techniques etc.

Different domains are tested differently, thus testing is purely based on the context of the domain or application.

For Example, testing a banking application is different than testing any e-commerce or advertising application. The risk associated with each type of application is different, thus it is not effective to use the same method, technique, and testing type to test all types of application.

Defect Clustering –

Defect Clustering which states that a small number of modules contain most of the defects detected. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.

Software Testing - Principles

Pesticide Paradox -

Pesticide Paradox principle says that if the same set of test cases are executed again and again over the period then these set of tests are not capable enough to identify new defects in the system.

In order to overcome this “Pesticide Paradox”, the set of test cases needs to be regularly reviewed and revised. If required a new set of test cases can be added and the existing test cases can be deleted if they are not able to find any more defects from the system.

Absence of Error -

If the software is tested fully and if no defects are found before release, then we can say that the software is 99% defect free. But what if this software is tested against wrong requirements? In such cases, even finding defects and fixing them on time would not help as testing is performed on wrong requirements which are not as per needs of the end user.

For Example, suppose the application is related to an e-commerce site and the requirements against “Shopping Cart or Shopping Basket” functionality which is wrongly interpreted and tested. Here, even finding more defects does not help to move the application into the next phase or in the production environment.

Quality Concepts – Cost of Quality

COQ -- Cost of Quality

Prevention Cost

Money required to prevent errors and to do the job right the first time. money spent on establishing methods and procedures

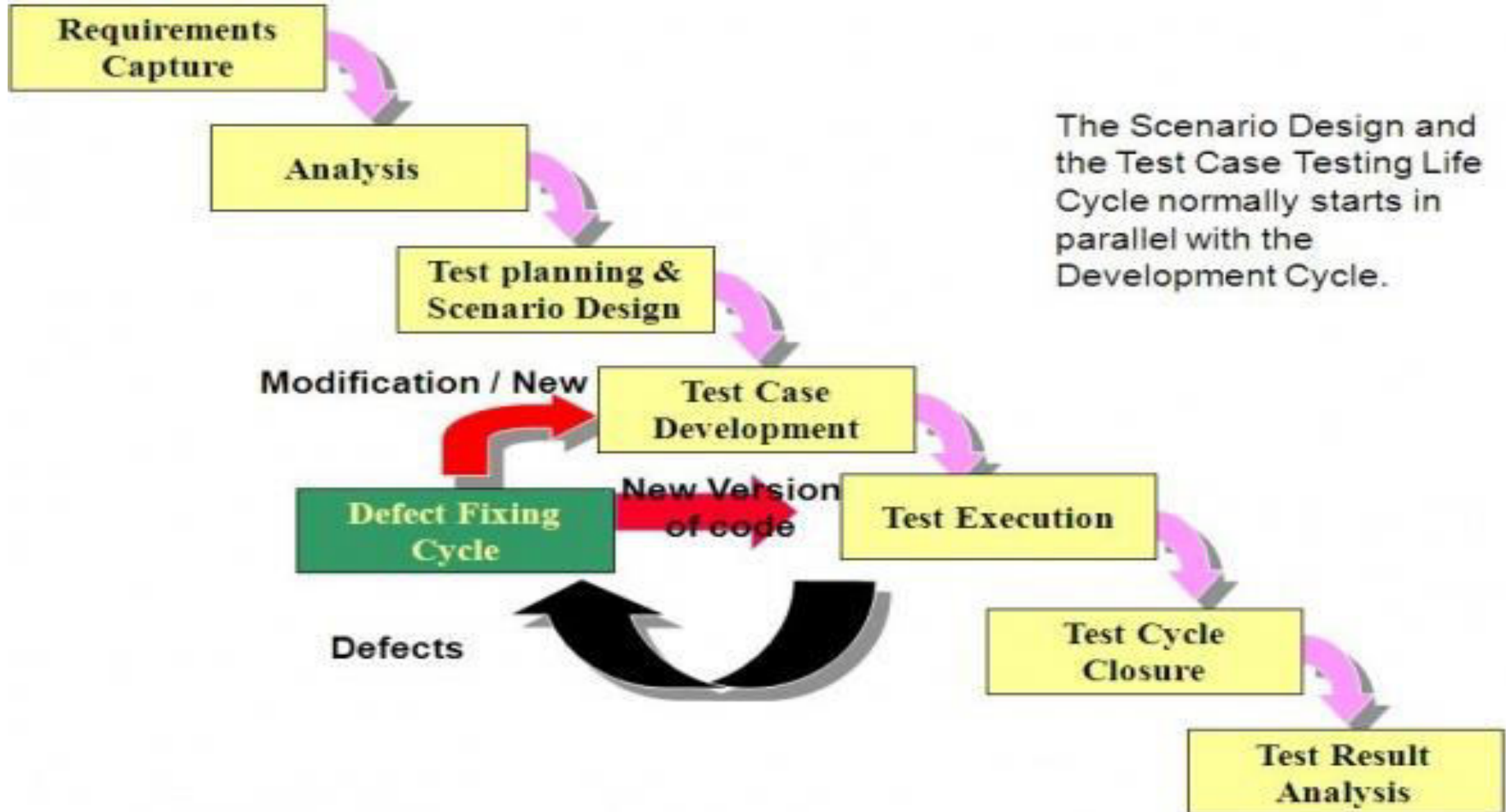
Appraisal Cost

Money spent to review completed products against requirements. Appraisal includes the cost of inspections, testing, and reviews. This money is spent after the product is built but before it is shipped to the user or moved into production.

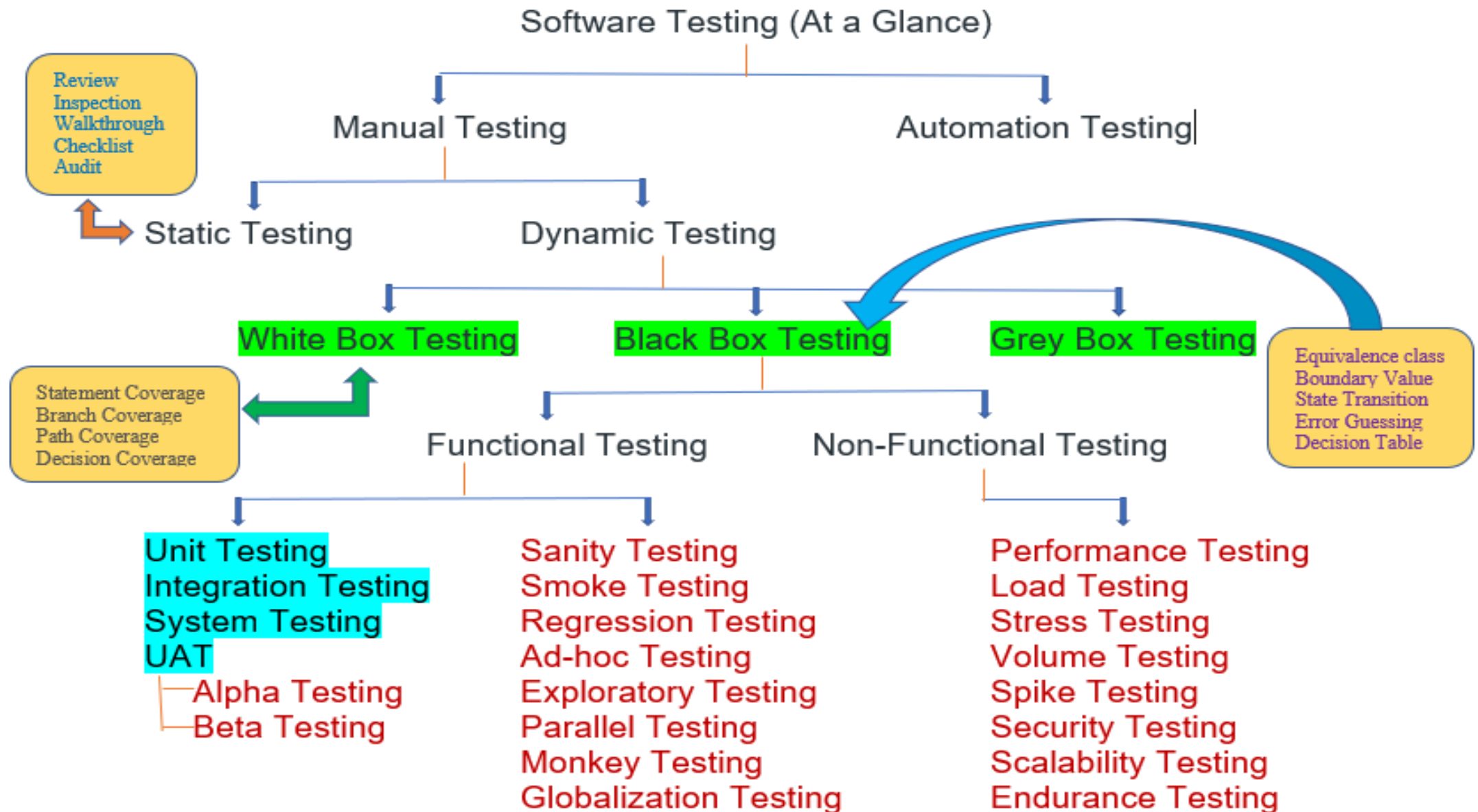
Failure Cost

All costs associated with defective products that have been delivered to the user or moved into production.

Software Testing Life Cycle



Software Testing – At a Glance



Software Testing Methodologies

White-box testing

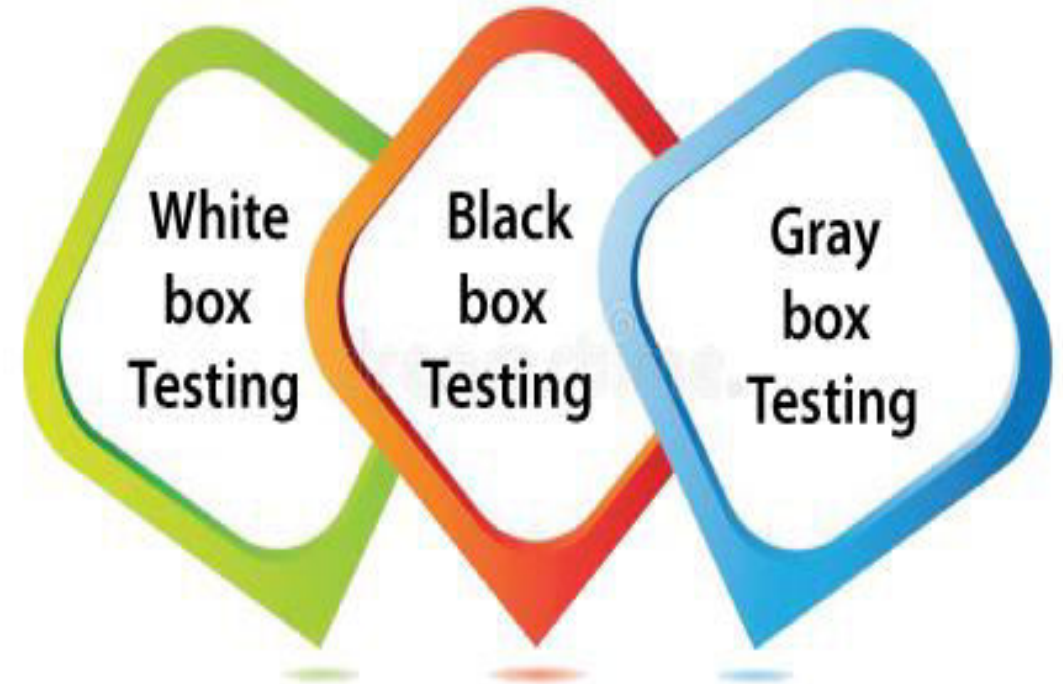
The white box testing is done by Developer, where they check every line of a code before giving it to the Test Engineer. Since the code is visible for the Developer during the testing, that's why it is also known as White box testing.

Black box testing

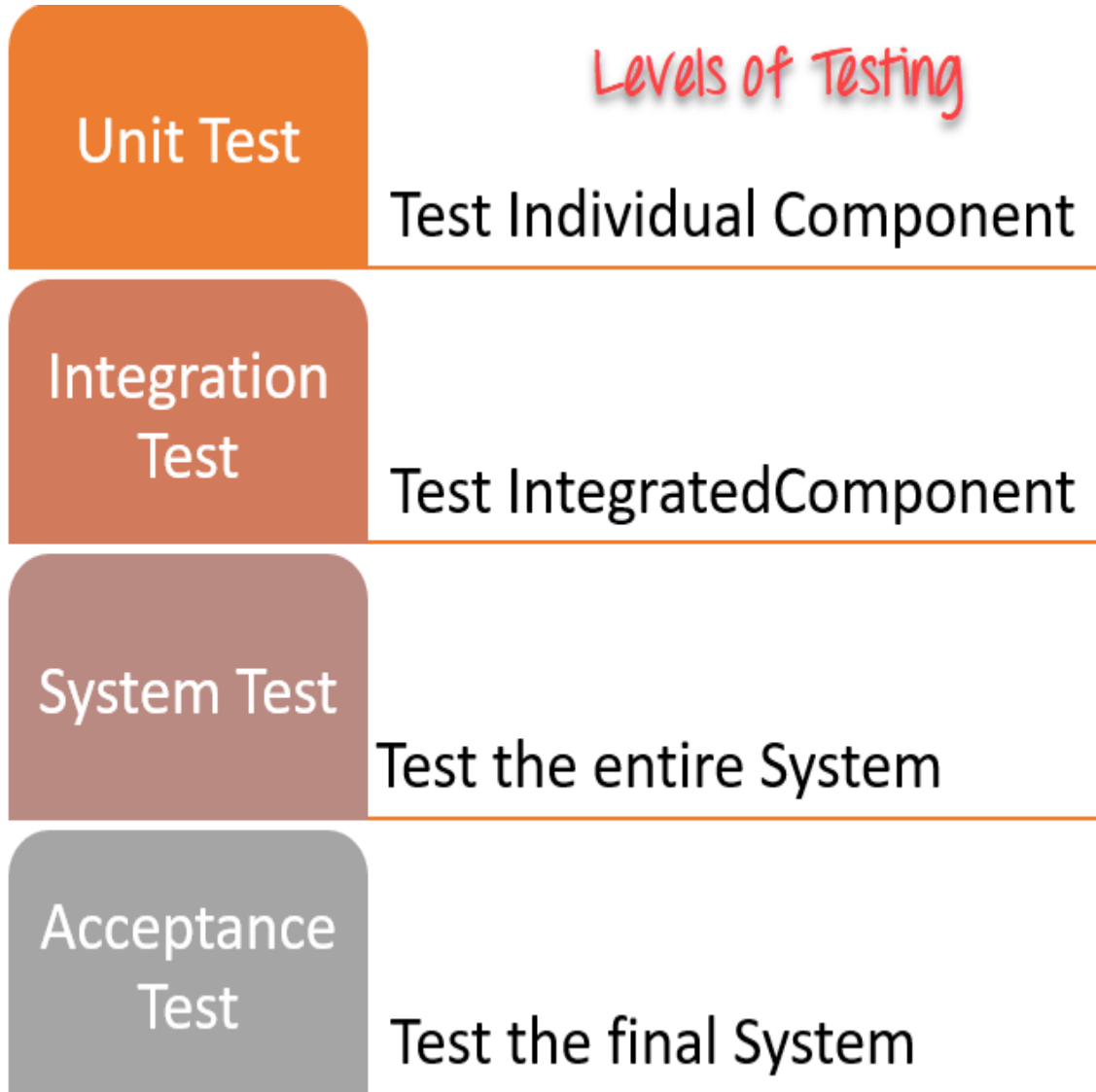
The black box testing is done by the Test Engineer, where they can check the functionality of an application or the software according to the customer /client's needs. In this, the code is not visible while performing the testing; that's why it is known as black-box testing.

Gray Box testing

Gray box testing is a combination of white box and Black box testing. It can be performed by a person who knew both coding and testing. And if the single person performs white box, as well as black-box testing for the application, is known as Gray box testing.



Levels of Software Testing



Unit Testing :

In this type of testing, errors are detected individually from every component or unit by individually testing the components or units of software to ensure that if they are fit for use by the developers. It is the smallest testable part of the software.

Integration Testing :

Two or more modules which are unit tested are integrated to test i.e., technique interacting components and are then verified if these integrated modules work as per the expectation or not and interface errors are also detected.

System Testing :

In system testing, complete and integrated Software's are tested i.e., all the system elements forming the system is tested as a whole to meet the requirements of the system.

Acceptance Testing :

It is a kind of testing conducted to ensure whether the requirement of the users are fulfilled prior to its delivery and the software works correctly in the user's working environment.

Types of Testing

Sanity Testing

Sanity testing is a stoppage to check whether testing for the build can proceed or not. The focus of the team during sanity testing process is to validate the functionality of the application and not detailed testing. Sanity testing is generally performed on build where the production deployment is required immediately like a critical bug fix.

Smoke Testing

This test is done to make sure that software under testing is ready or stable for further testing. It is called a smoke test as the testing an initial pass is done to check if it did not catch the fire or smoke in the initial switch on.

Regression Testing

Every time a new module is added leads to changes in the program. This type of testing makes sure that the whole system works properly even after adding or changing components to the complete program. (Impact)

Parallel Testing: Testing technique which has the purpose to ensure that a new application which has replaced its older version has been installed and is running correctly. It is conducted by the testing team.

Exploratory Testing: Black box testing technique performed without planning and documentation. It is usually performed by manual testers.

Ad-hoc Testing:

Types of Testing

Usability Testing: Testing technique which verifies the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component. It is usually performed by end users.

Mutation Testing: Method of software testing which involves modifying programs' source code or byte code in small ways in order to test sections of the code that are seldom or never accessed during normal tests execution. It is normally conducted by testers

End to End Testing

Like system testing, involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate. It is performed by QA teams:

Localization Testing

Types of Testing

Load Testing: Testing technique that puts demand on a system or device and measures its response. It is usually conducted by the performance engineers.

Stress Testing

In this, we give unfavorable conditions to the system and check how they perform in those conditions.

Performance Testing

It is designed to test the run-time performance of software within the context of an integrated system. It is used to test the speed and effectiveness of the program. It is also called load testing. In it we check, what is the performance of the system in the given load.

Volume Testing: Testing which confirms that any values that may become large over time (such as accumulated counts, logs, and data files), can be accommodated by the program and will not cause the program to stop working or degrade its operation in any manner. It is usually conducted by the performance engineer.

Recovery Testing: Testing technique which evaluates how well a system recovers from crashes, hardware failures, or other catastrophic problems. It is performed by the testing teams

Security Testing

Security testing unveils the vulnerabilities of the system to ensure that the software system and application are free from any threats or risks. These tests aim to find any potential flaws and weaknesses in the software system that could lead to a loss of data, revenue, or reputation per employees or outsiders of a company.

Types of Testing

Scalability Testing: Part of the battery of non-functional tests which tests a software application for measuring its capability to scale up – be it the user load supported, the number of transactions, the data volume etc. It is conducted by the performance engineer.

Compatibility Testing: Testing technique that validates how well a software performs in a particular hardware/software/operating system/network environment. It is performed by the testing teams.

Penetration Testing: Testing method which evaluates the security of a computer system or network by simulating an attack from a malicious source. Usually, they are conducted by specialized penetration testing companies.

Acceptance Testing: Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system. It is usually performed by the customer.

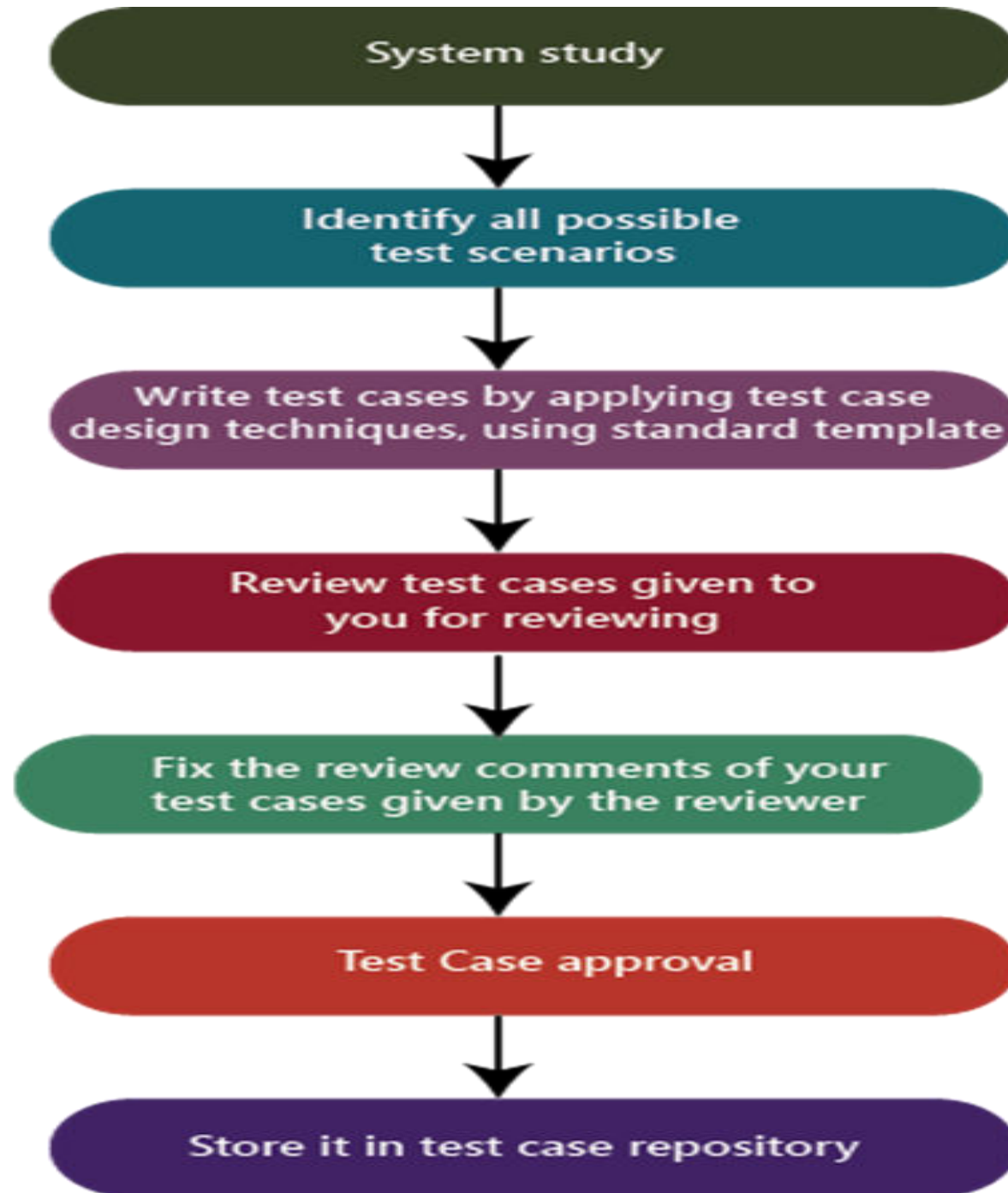
Alpha Testing

Alpha Testing is a type of software testing performed to identify bugs before releasing the software product to the real users or public. It is a type of acceptance testing which is done before the product is released to end user and performed in controlled environment. It is typically done by QA people.

Beta Testing

The beta test is conducted at one or more customer sites by the end-user of the software. This version is released for a limited number of users for testing in a real-time environment.

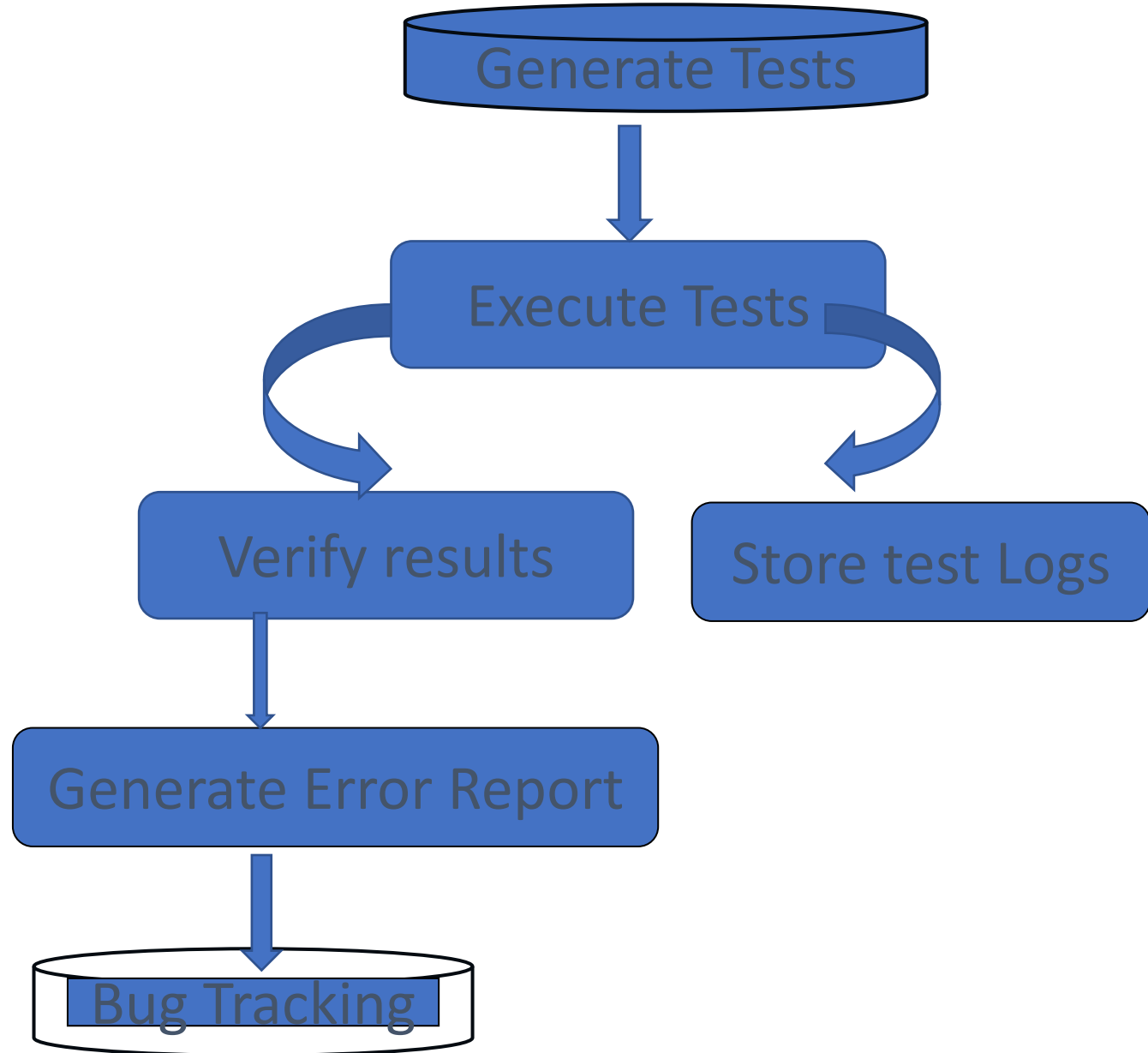
Test Case - Process



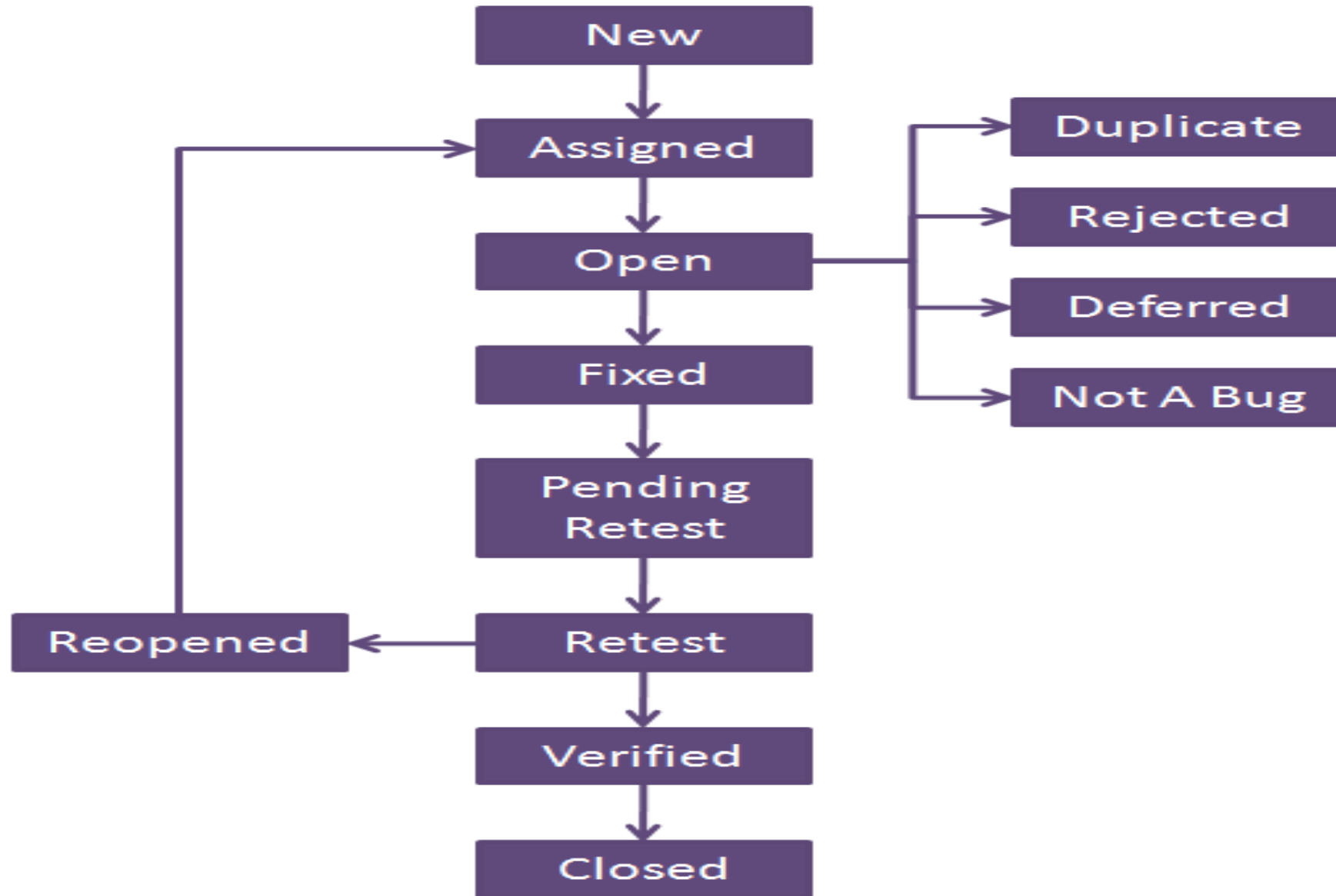
Best Practices To Write Good Test Case

- Easy to understand and execute
- Create Test Cases with End User's perspective
- Unique Test case Identifiers must be used. It allows us to track them easily.
- Prerequisites should be listed clearly. Helps to execute the test case without any issues.
- Test data should be defined to evaluate each functional area.
- Test case description should be concise.
- Test Steps should be in detail and clear.
- Specify the exact expected result.
- Position condition should be listed if any.
- Test cases should neither too simple nor too complex.
- Test cases must be distinctive. There should not be no repeated test cases.
- Test cases should be written by following test case design techniques
- Test cases must be comprehensible. So that any tester (even a newly appointed testers) can understand them by perusing once.
- Needs to provide clear environment details where we need to execute them.
- Test cases should be reusable & maintainable
- Get peer review.

Test Case –Execution Flow



Defect Life Cycle



Defect - Priority and Severity

Defect Priority provides a perspective for the order of the defect fixes. Priority in other words tells us , how soon the defect must be fixed.

For Example, the priority can be divided into P1, P2, P3,P4 and so on.

P1 – Fix the defect on highest priority, fix it before the next build

P2 – Fix the defect on high priority before the next test cycle

P3 – Fix the defect on moderate priority when time permits , before the release

P4- Postpone the defect for the next release or live with this defect.

Defect Severity provides the perspective of the impact of that defect in product functionality.

Defect Severity levels can be:

S1,S2,S3, S4 or Extreme, Critical, important, Minor, Cosmetic

Extreme – Product Crashes or is unusable

Critical - Basic functionality of the product is not working.

Important- Extended functionality of the product not working

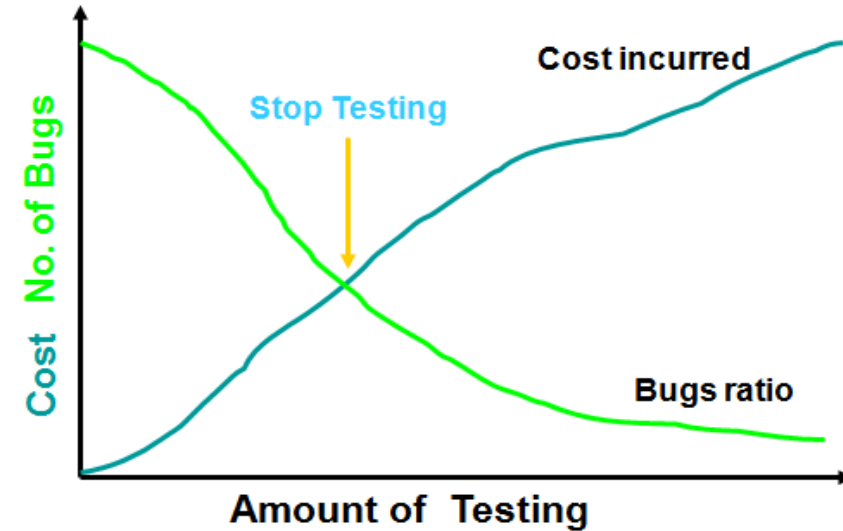
Minor – Product behaves differently

Cosmetic – GUI related

When to Stop Testing

Test Manager will consider the following factors

- Deadlines, e.g. release deadlines, testing deadlines;
- Test cases completed with certain percentage passed;
- Test budget has been depleted;
- Coverage of code, functionality or requirements reaches a specified point;
- Bug rate falls below a certain level
- Beta or alpha testing period ends



Thank You

