Welcome!

Software Engineering Best Practices

by

Girish Godbole

Program Agenda

1. Software Engineering - An Overview
2. Software Development Models & Architecture
3. Software Requirements Engineering
4. Software Project Management(SPM)
5. Software Testing and Debugging

Software Engineering - What

Software Engineering - An Overview

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

“A set of executable program in an application”

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: applications engineering and systems engineering. Applications engineering is the process to develop software applications for businesses and organizations, whereas systems 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

1.Maintainability – It should be feasible for the software to evolve to meet changing requirements.

2.Efficiency – The software should not make wasteful use of computing devices such as memory, processor cycles, etc.

3.Correctness – A software product is correct if the different requirements as specified in the SRS document have been correctly implemented.

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

5.Testability – Here software facilitates both the establishment of test criteria and the evaluation of the software with respect to those criteria

6.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.

7.Portability – In this case, the software can be transferred from one computer system or environment to another.

8.Adaptability – The software allows differing system constraints

9.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

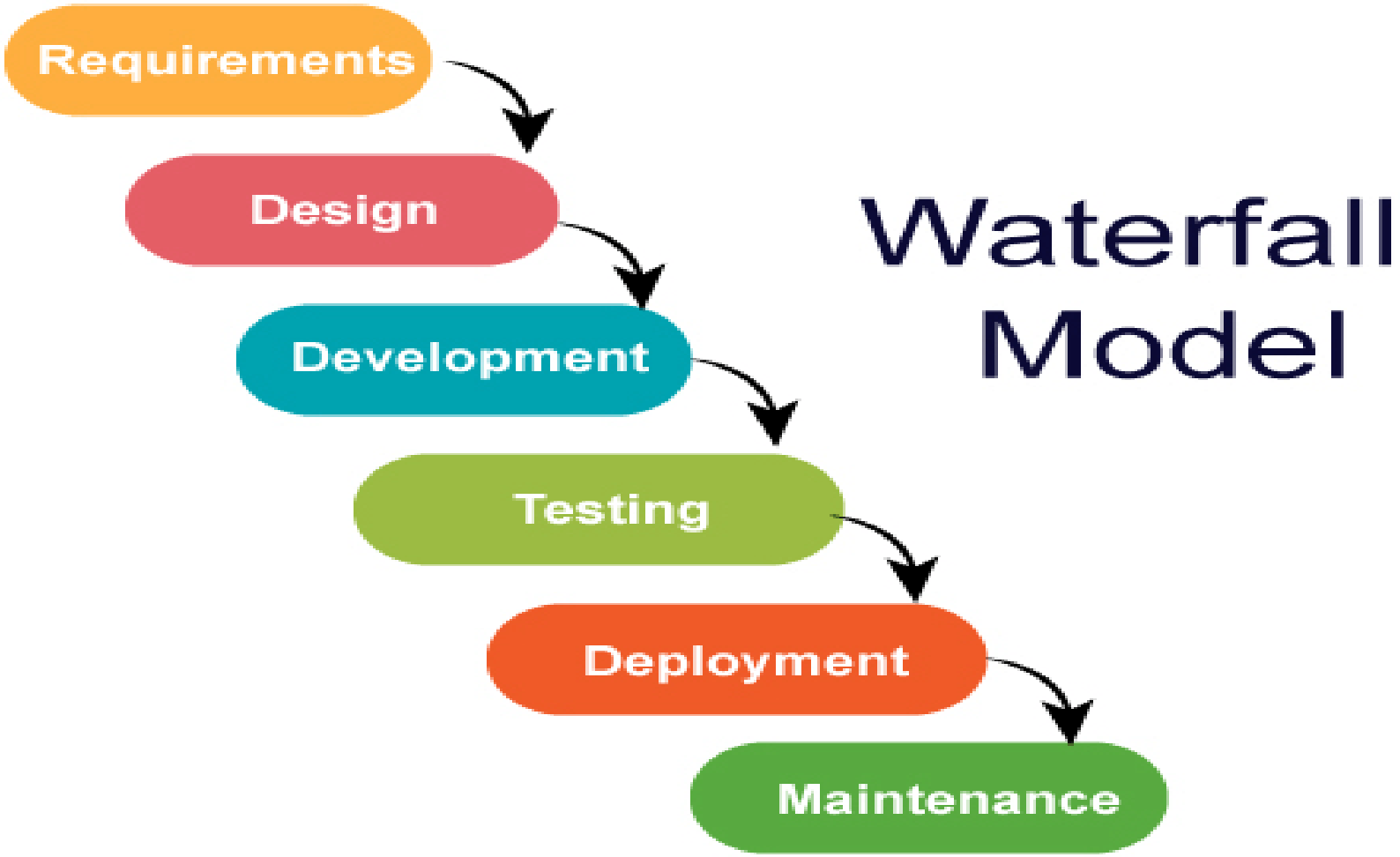
Bug fixing – bugs are reported because of some scenarios which are not tested at all

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

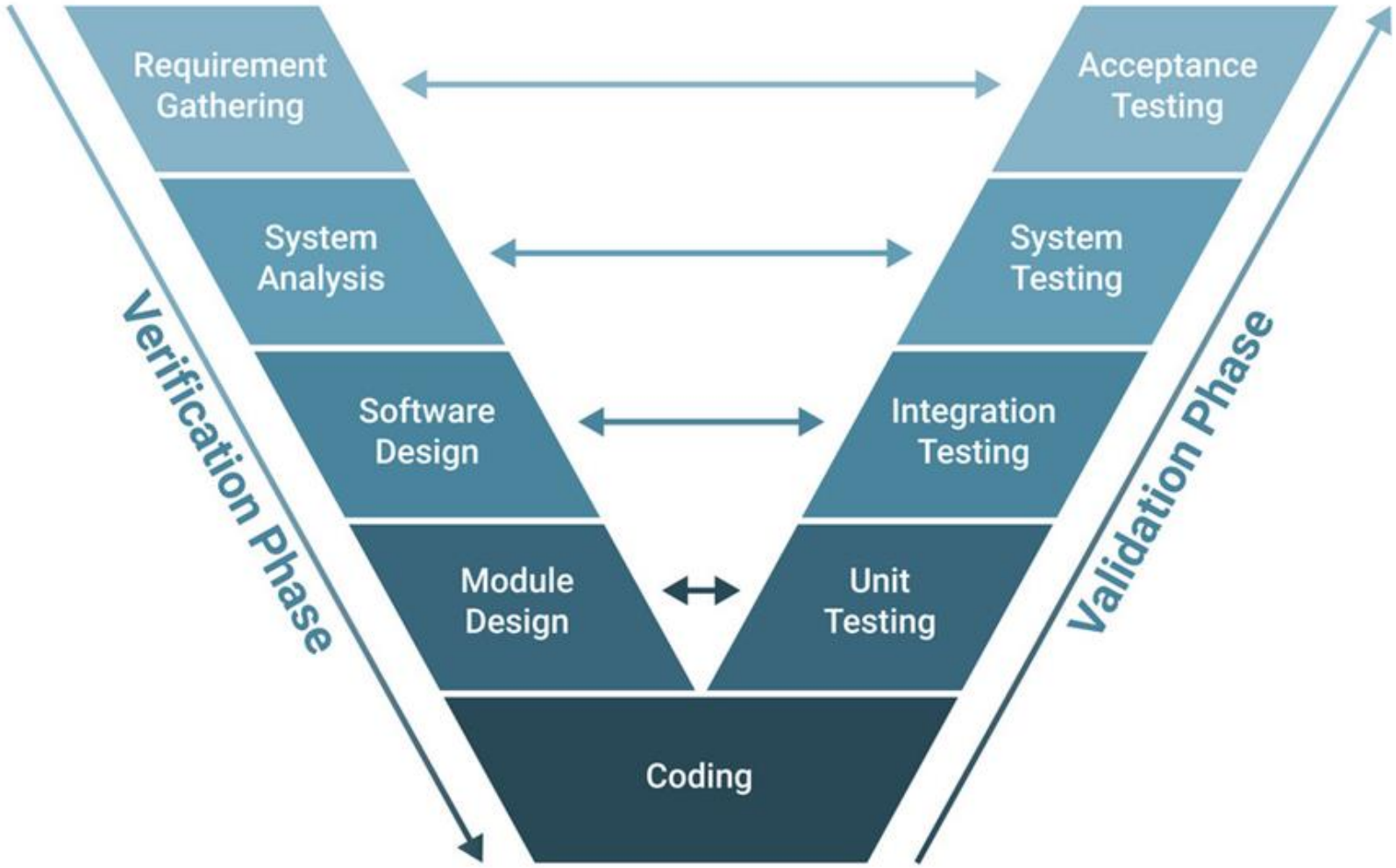
Waterfall Deficiencies

All requirements must be known upfront
Deliverables created for each phase are considered frozen – inhibits flexibility
Can give a false impression of progress
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.

SDLC – V Model


Requirement Gathering
System Analysis
Software Design
Module Design
Coding
Unit Testing
Integration Testing
System Testing
Acceptance Testing
Verification Phase
Validation Phase

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
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

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.

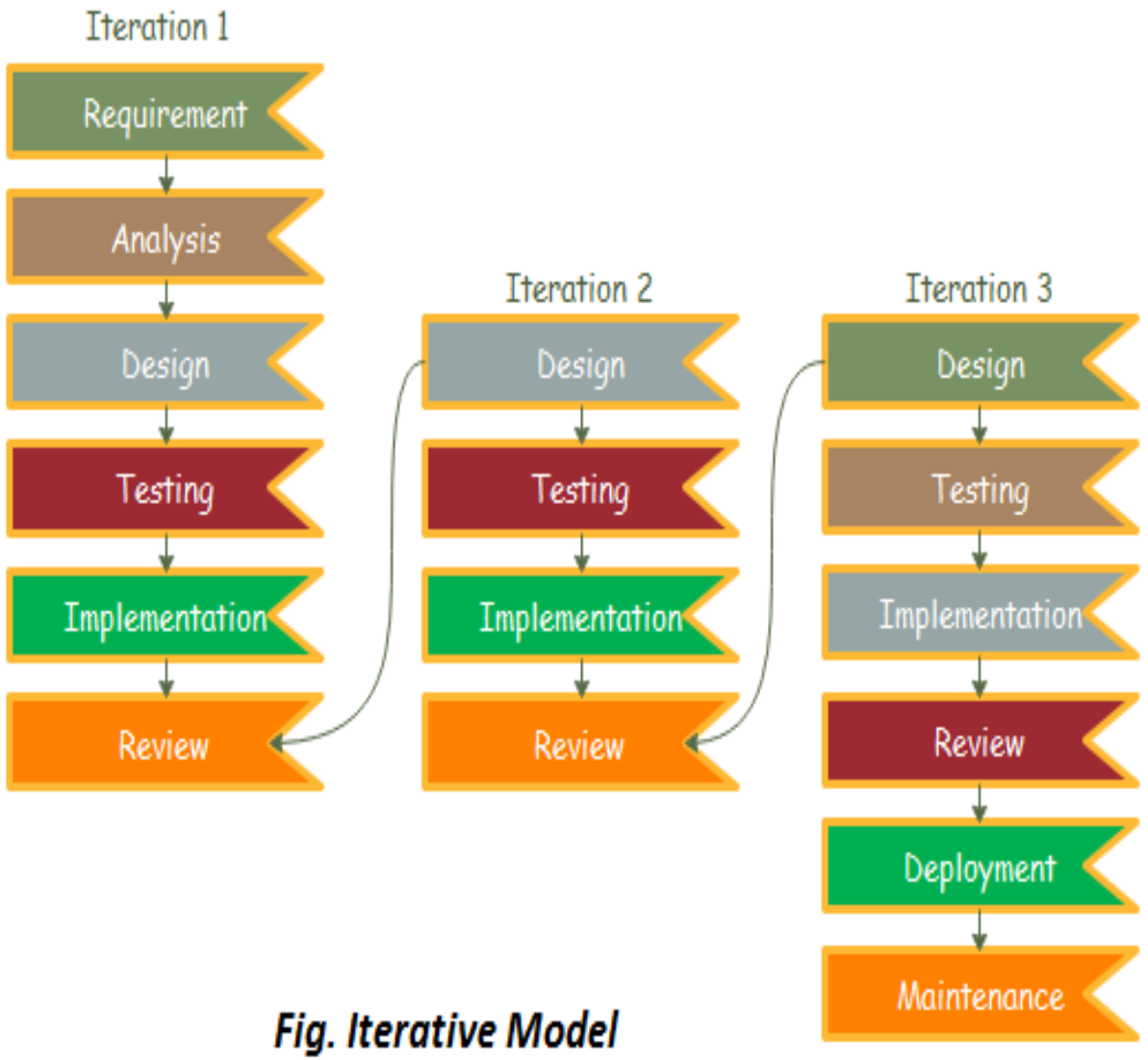


Fig. Iterative Model

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.

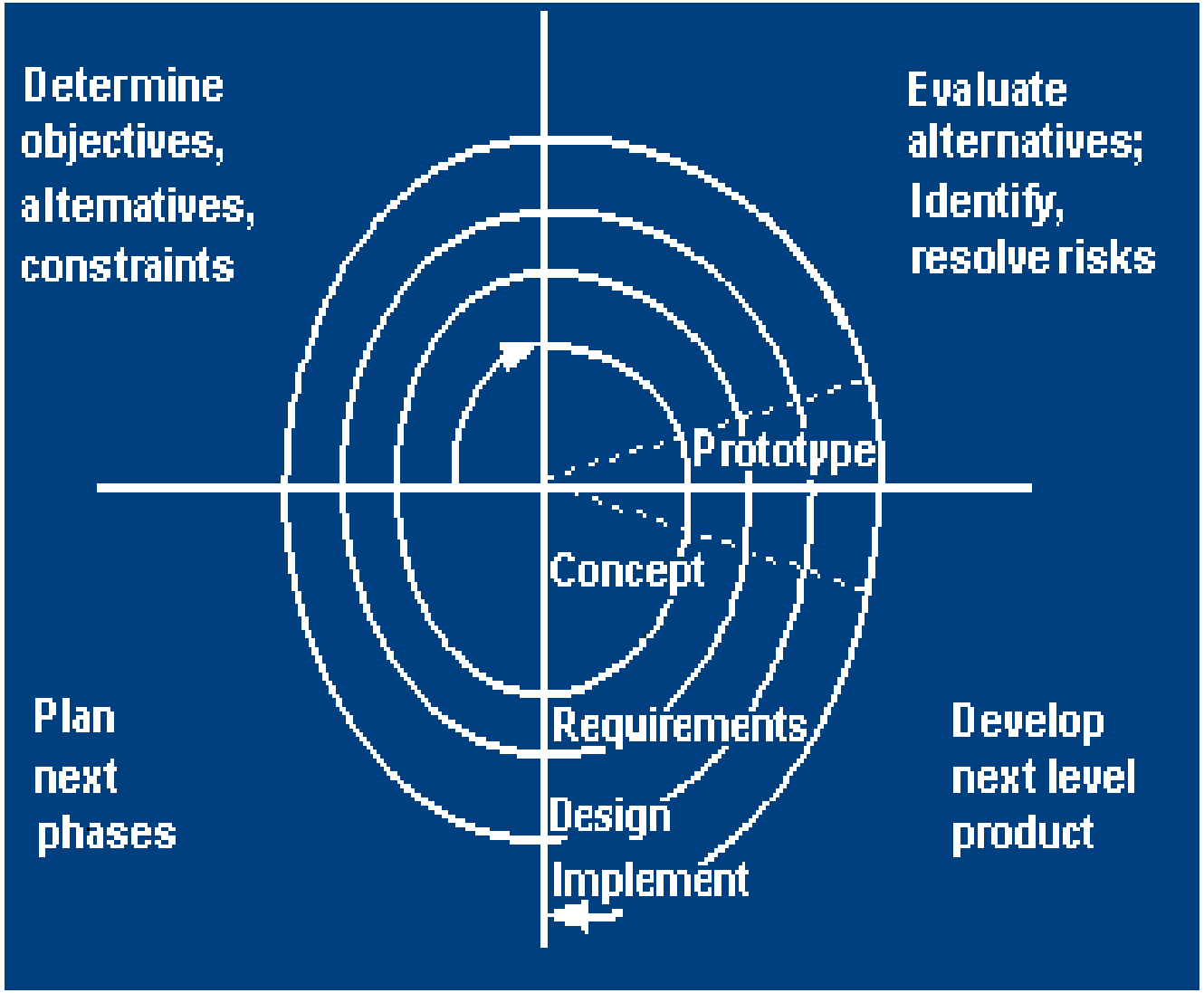
When to use the Iterative Model?

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

Determine objectives, alternatives and constraints
Objectives: functionality, performance, hardware/software interface, critical success factors, etc.
Alternatives: build, reuse, buy, sub-contract, etc.
Constraints: cost, schedule, interface, etc.

Evaluate alternatives, identify and resolve risks
Study alternatives relative to objectives and constraints
Identify risks (lack of experience, new technology, tight schedules, poor process, etc.
Resolve risks (evaluate if money could be lost by continuing system development

Develop next-level product
Typical activities:
Create a design
Review design
Develop code
Inspect code
Test product

SDLC – Spiral Model

Plan next phase
Typical activities
Develop project plan
Develop configuration management plan
Develop a test plan
Develop an installation plan

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

SDLC – Spiral Model

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

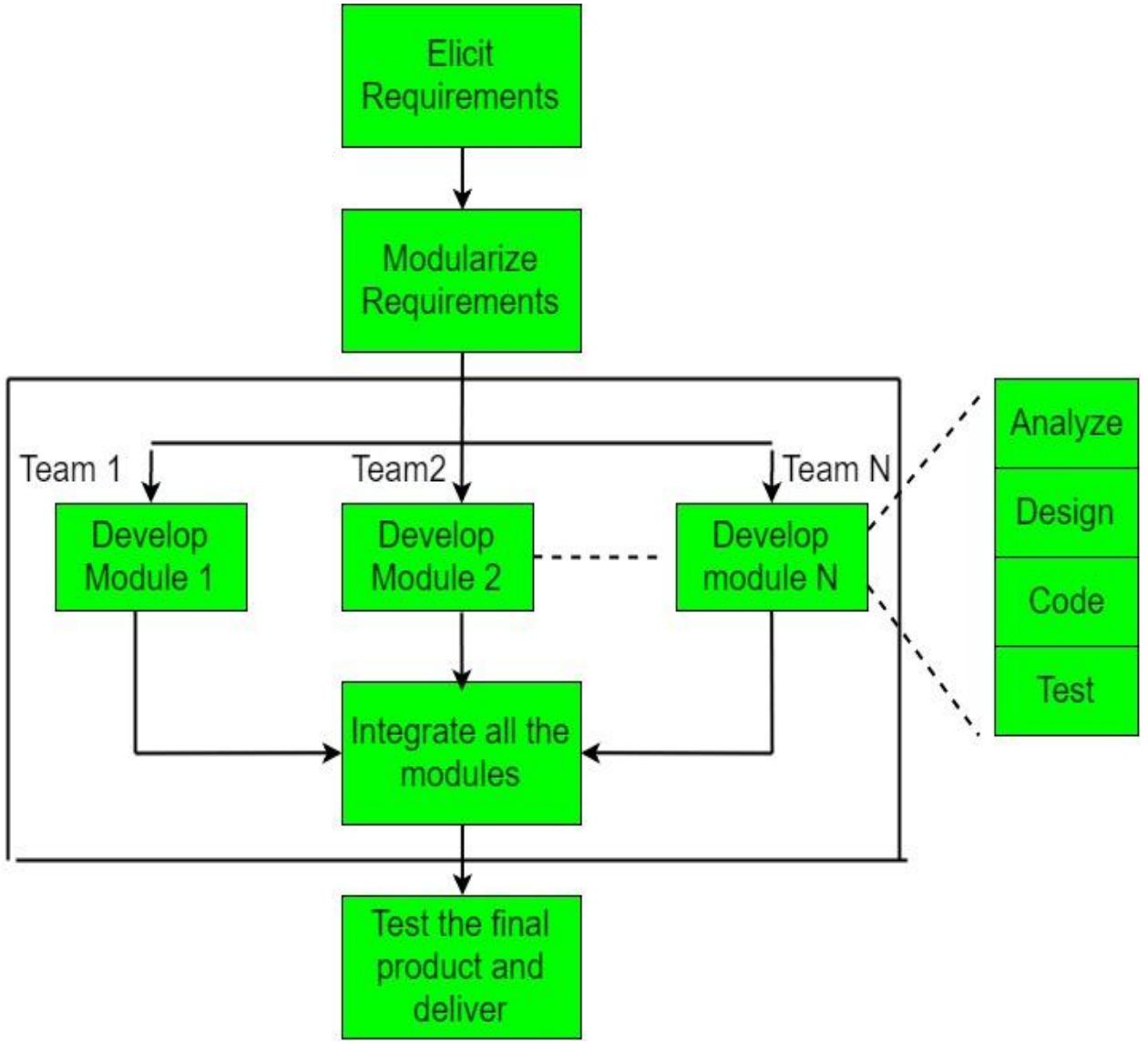
When to use

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
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
Significant changes are expected (research and exploration)

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
The Agile model does not recommend developing prototypes but emphasizes the systematic development of each incremental feature at the end of each iteration.	The central theme of RAD is based on designing quick and dirty prototypes, which are then refined into production quality code.
Agile projects logically break down the solution into features that are incrementally developed and delivered.	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.
The Agile team only demonstrate completed work to the customer after each iteration.	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.
Agile model is not suitable for small projects as it is difficult to divide the project into small parts that can be incrementally developed.	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.

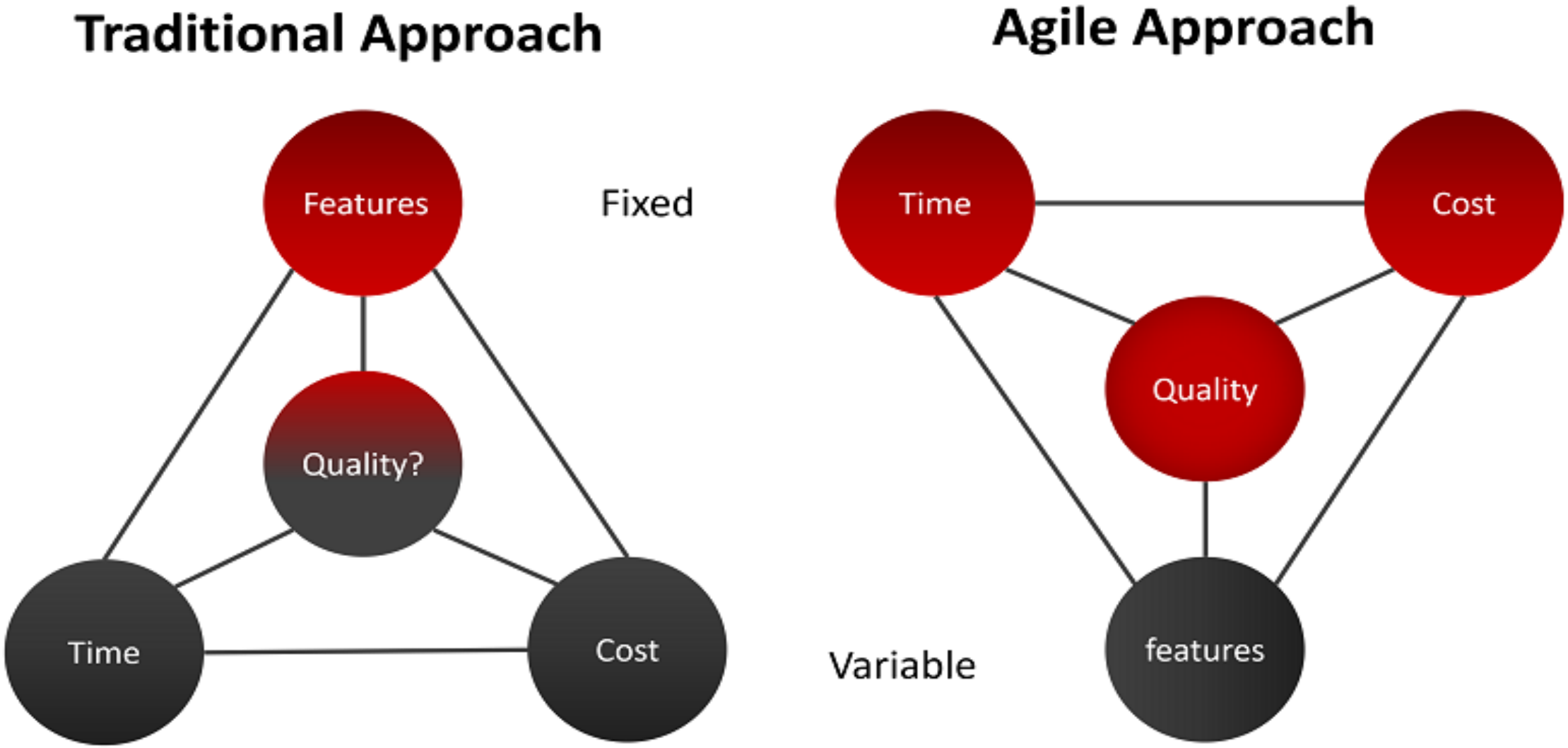
SDLC – Agile Methodologies

Agile - “quick to move or drive”

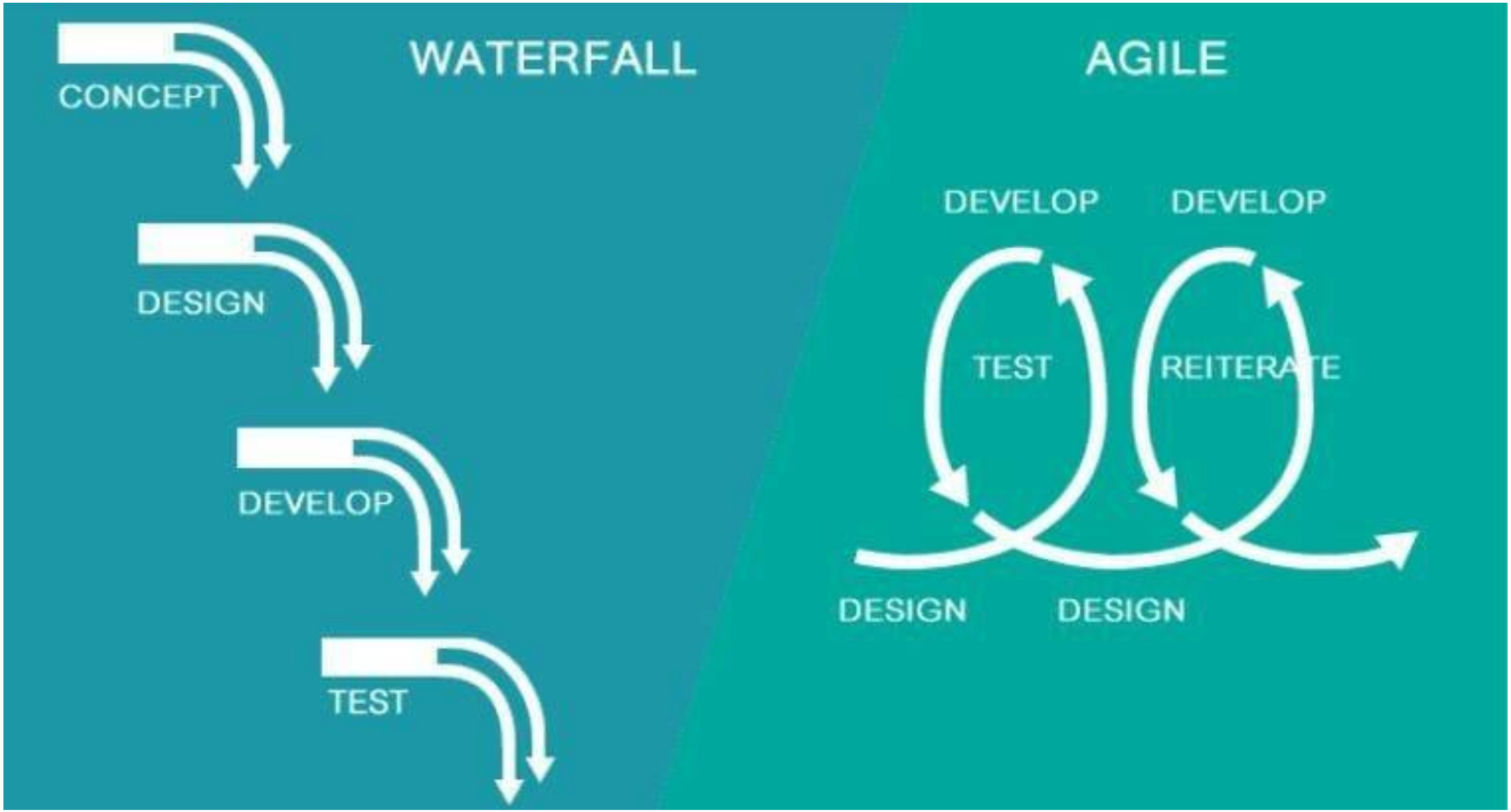
Agile Software development methodology.

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

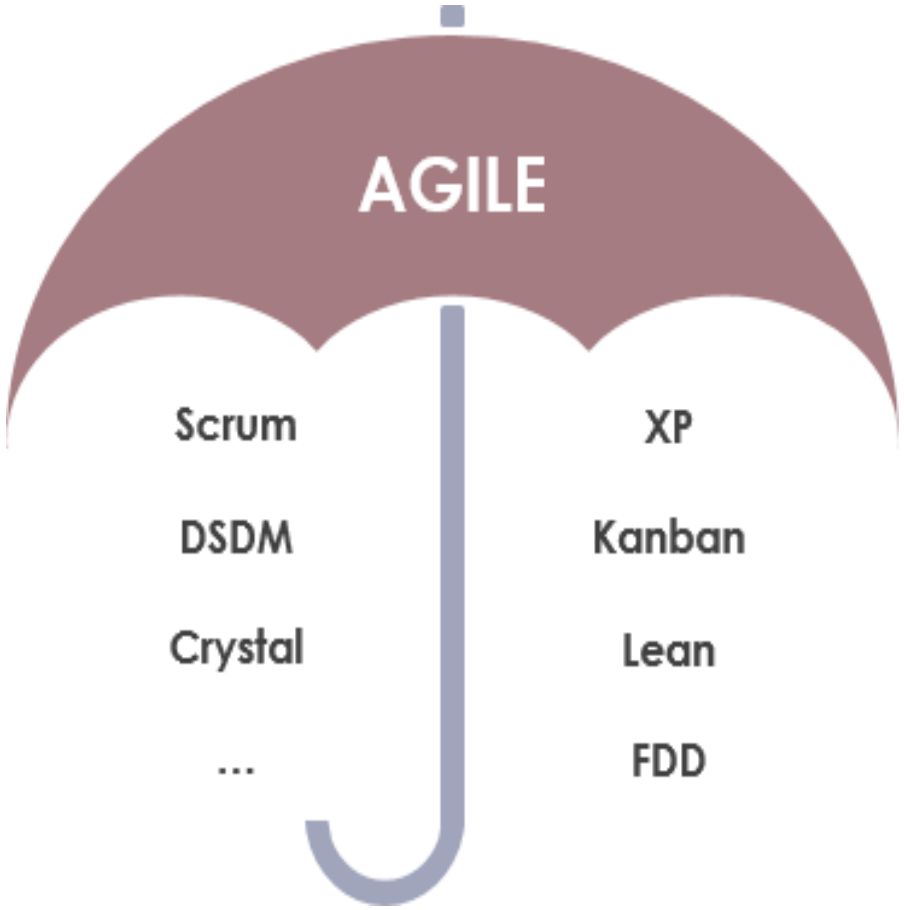
Agile – Approach



Traditional vs Agile Approach



Agile Methods and Practices



- Scrum
- Crystal Methodologies
- DSDM (Dynamic Software Development Method)
- Feature driven development (FDD)
- Lean software development
- Extreme Programming (XP)

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

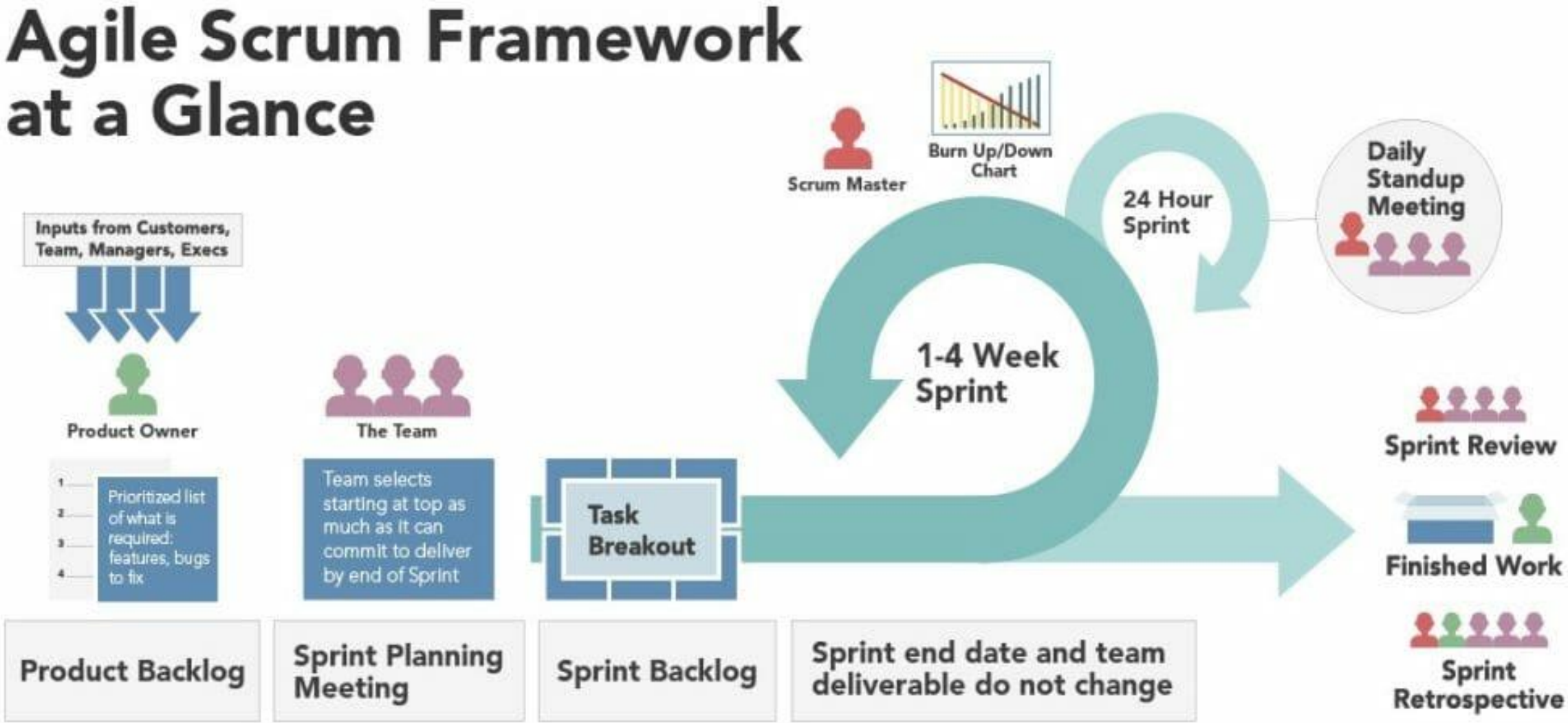
The approach is similar to other Agile methodologies in its attention to early and often delivery of software, high involvement of users, and removal of red tape. Crystal's assertion that every project is unique, however, has led to its reputation as one of the most flexible Agile methodologies.

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



Requirement Engineering

What is Requirement ?

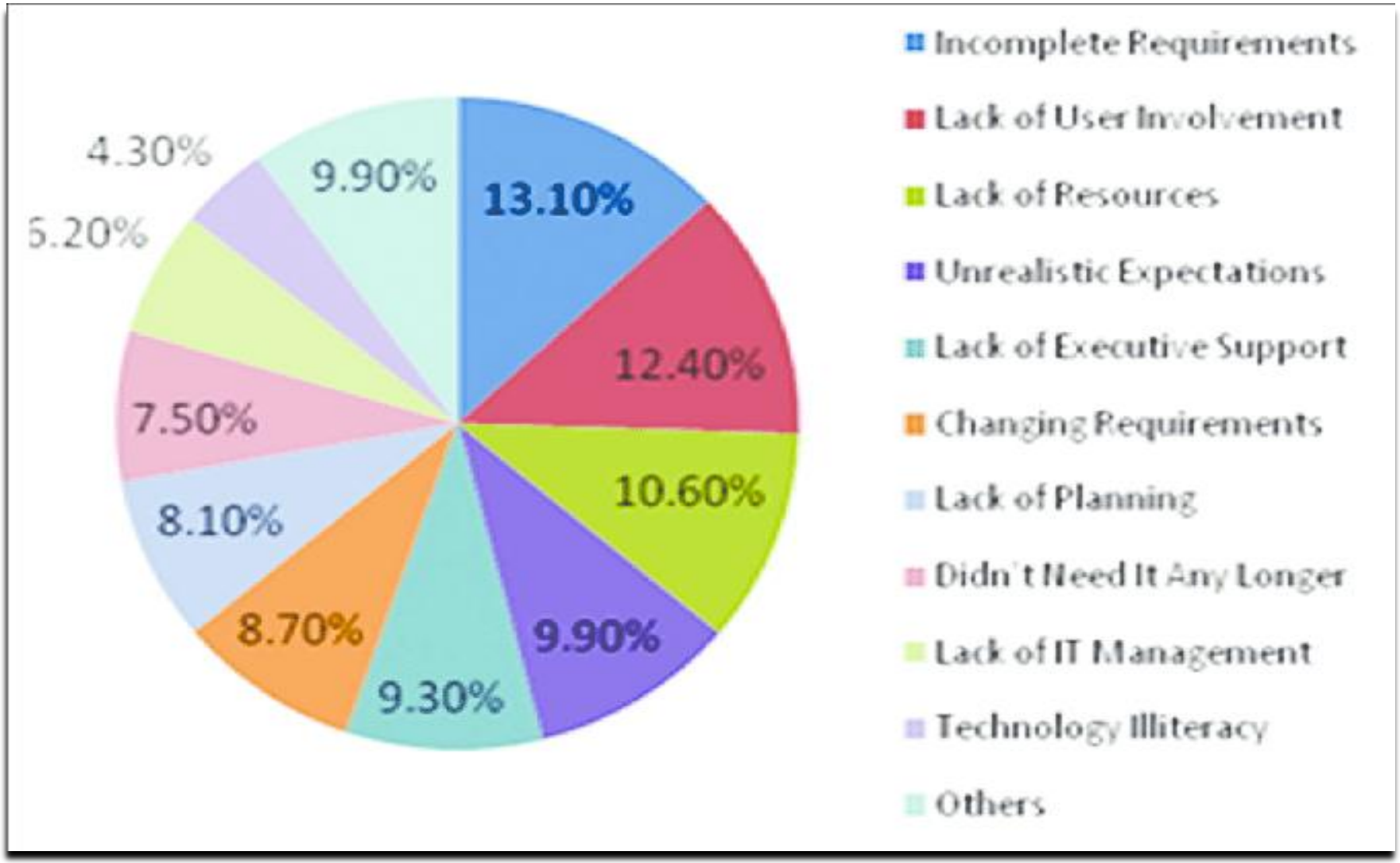
(1) A condition or capability needed by a user to solve a problem or achieve an objective.
(2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.

Type of Requirements –

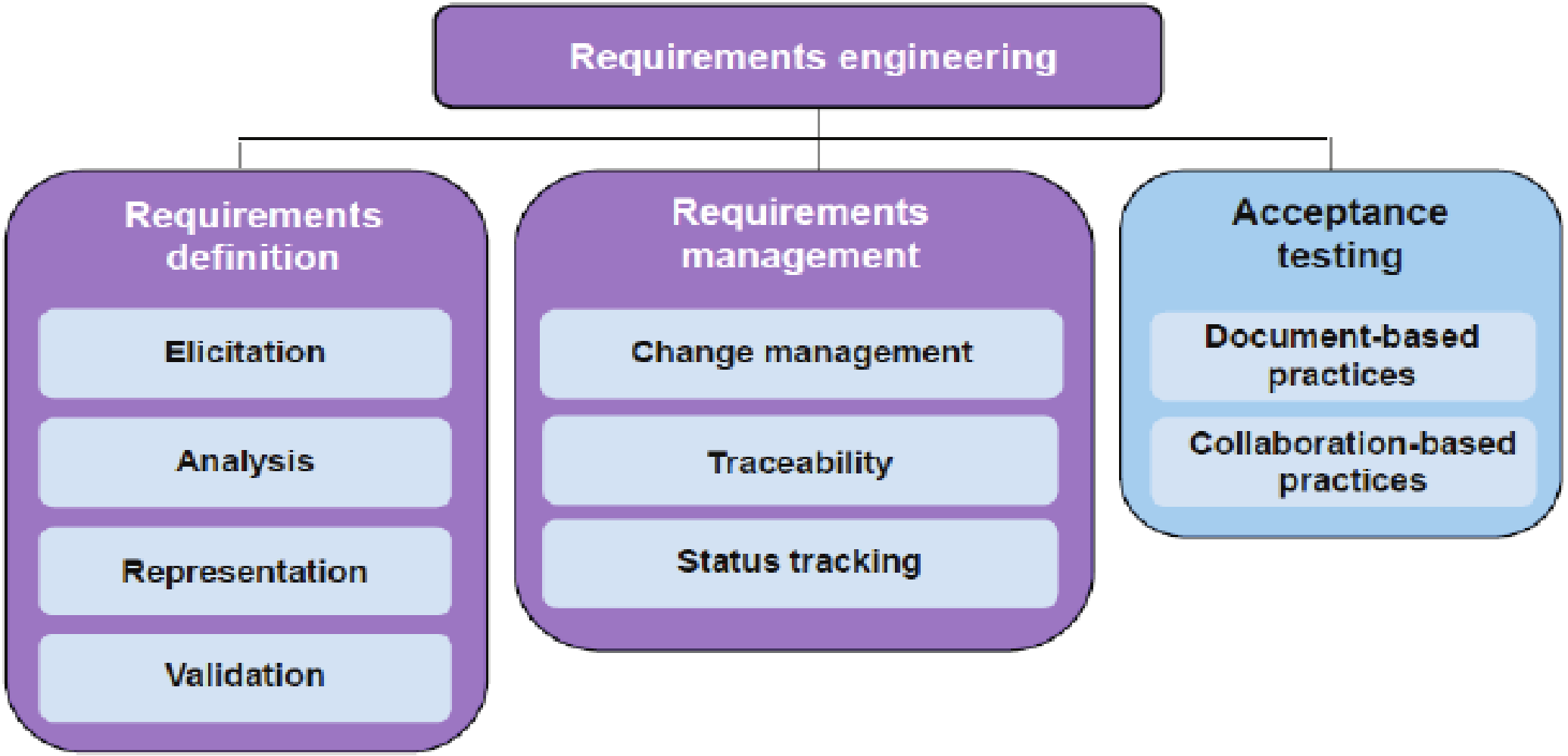
Business Requirements
Customer Requirements
Functional Requirements
Non-functional Requirements
Quality / Testing Requirements
Technology requirement Requirements
Implementation/Transition Requirements

Requirement Engineering

Why do Projects fail ? – Current Survey



Requirement Engineering



CMM-Capability Maturity Model

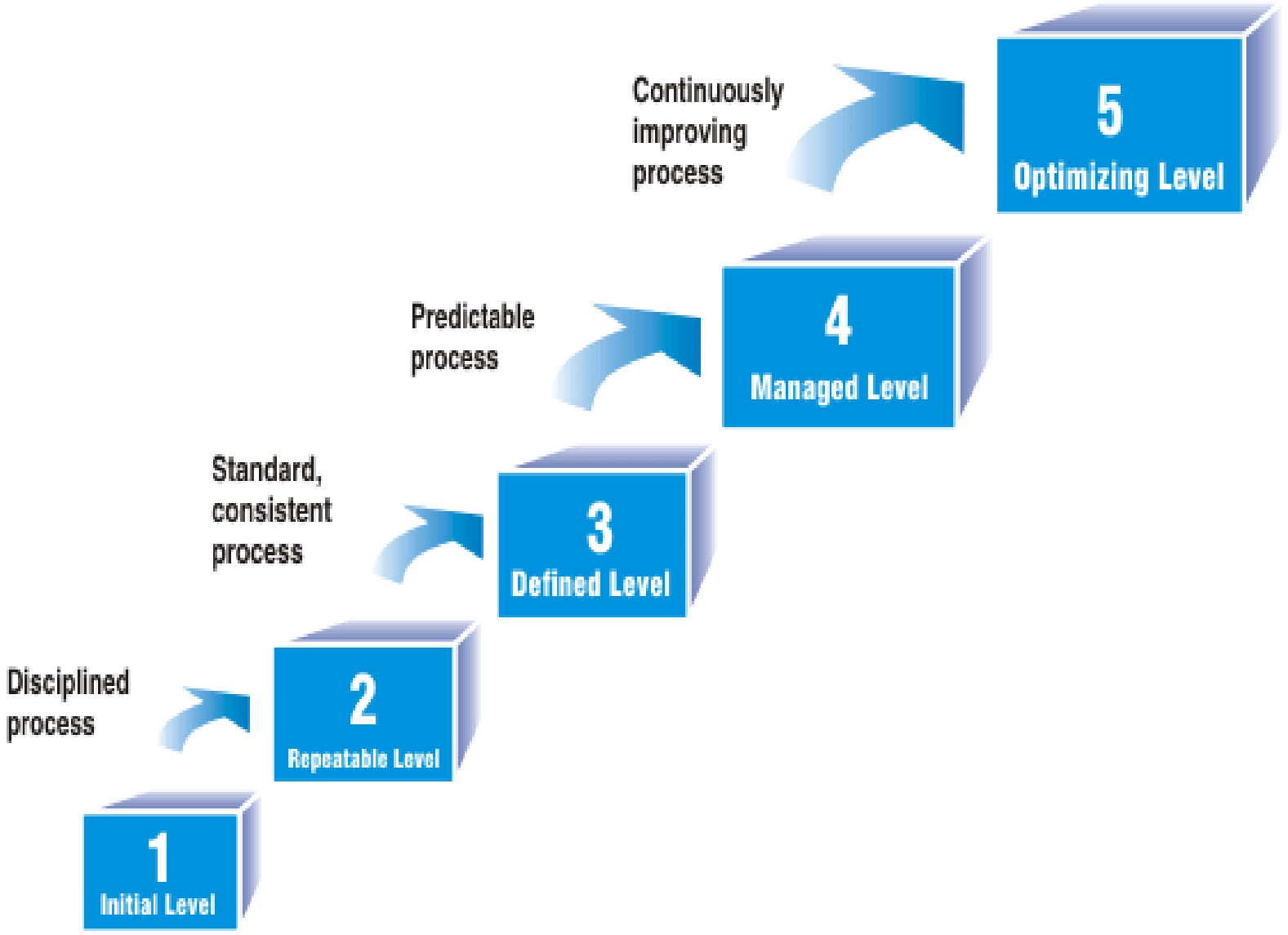
CMM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.

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.

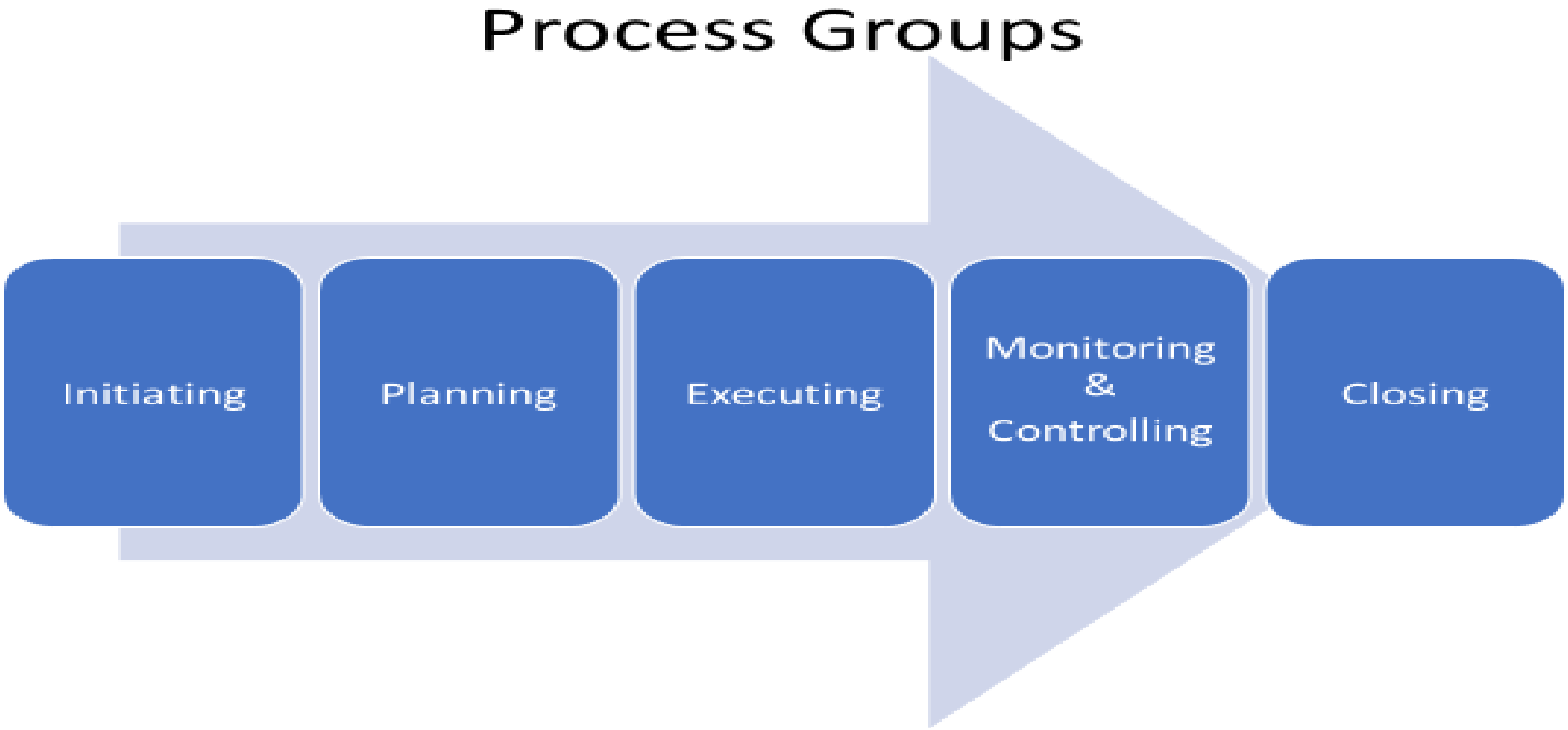
It is based on profound feedback and development practices adopted by the most successful organizations worldwide.

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.



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

Project Manager Roles and Skills

Role of a software project manager:

- Planning, Monitoring and Controlling the Project health.
- Lead the team
- Motivate the team-member
- Tracking the progress
- Liaison
- Documenting project report
- Managing the Risks

Necessary skills of software project manager:

- Communication Skills
- Listening Abilities
- Knowledge of project estimation techniques
- Good decision-making abilities at the right time
- Previous experience of managing a similar type of projects
- A project manager must encourage all the team members to successfully develop the product
- Risk management Capabilities
- Conflict management Capabilities

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.

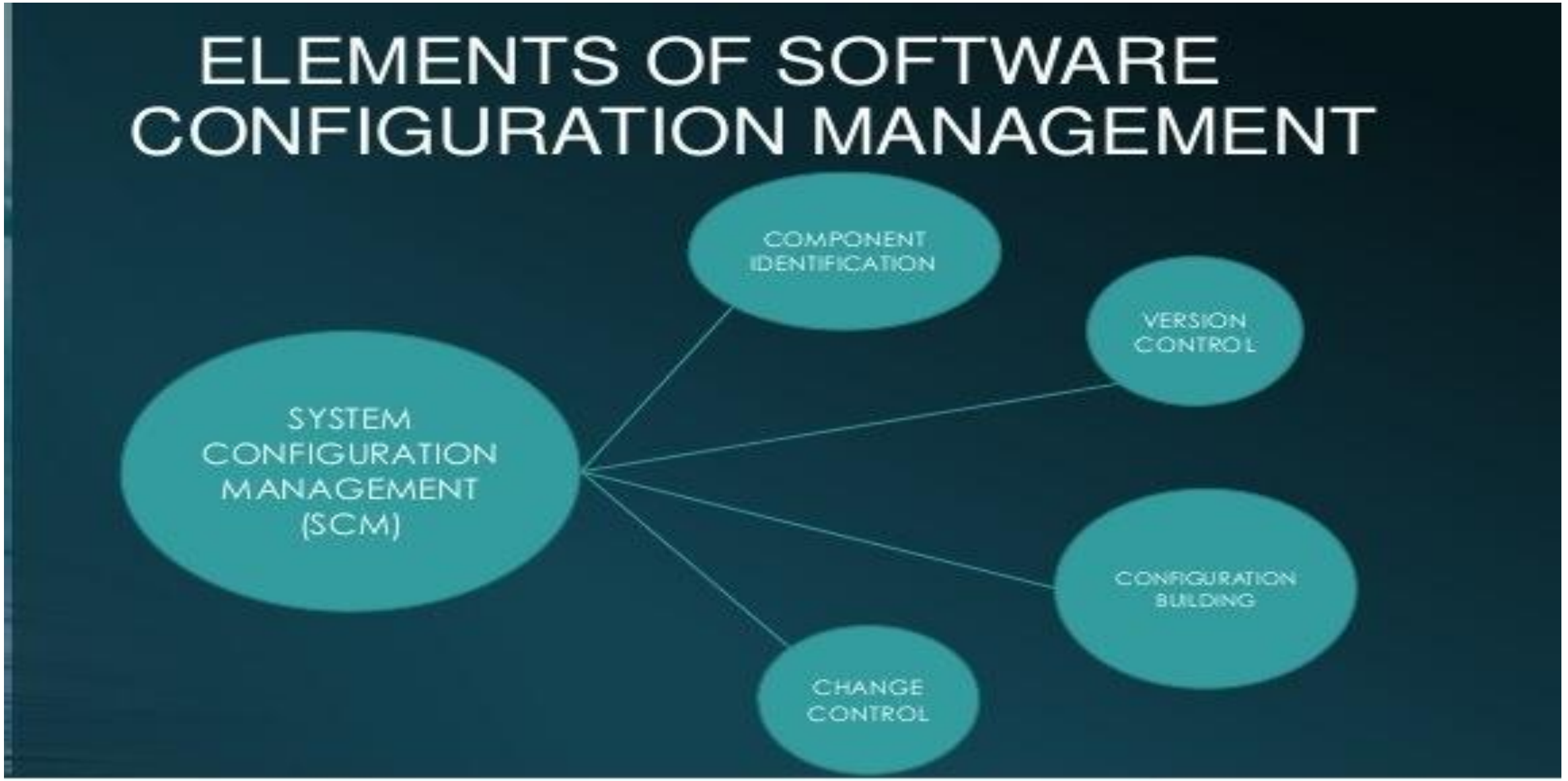
Version Control Friendly:
With this feature, the user can take their choice of version for their work.

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



COCOMO Model

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e number of Lines of Code. It is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality. It was proposed by Barry Boehm in 1981 and is based on the study of 63 projects, which makes it one of the best-documented models.

The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:

- Effort: Amount of labor that will be required to complete a task. It is measured in person-months units.
- Schedule: Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, months.

These are types of COCOMO model:
1.Basic COCOMO Model
2.Intermediate COCOMO Model
3.Detailed COCOMO Model

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 - 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

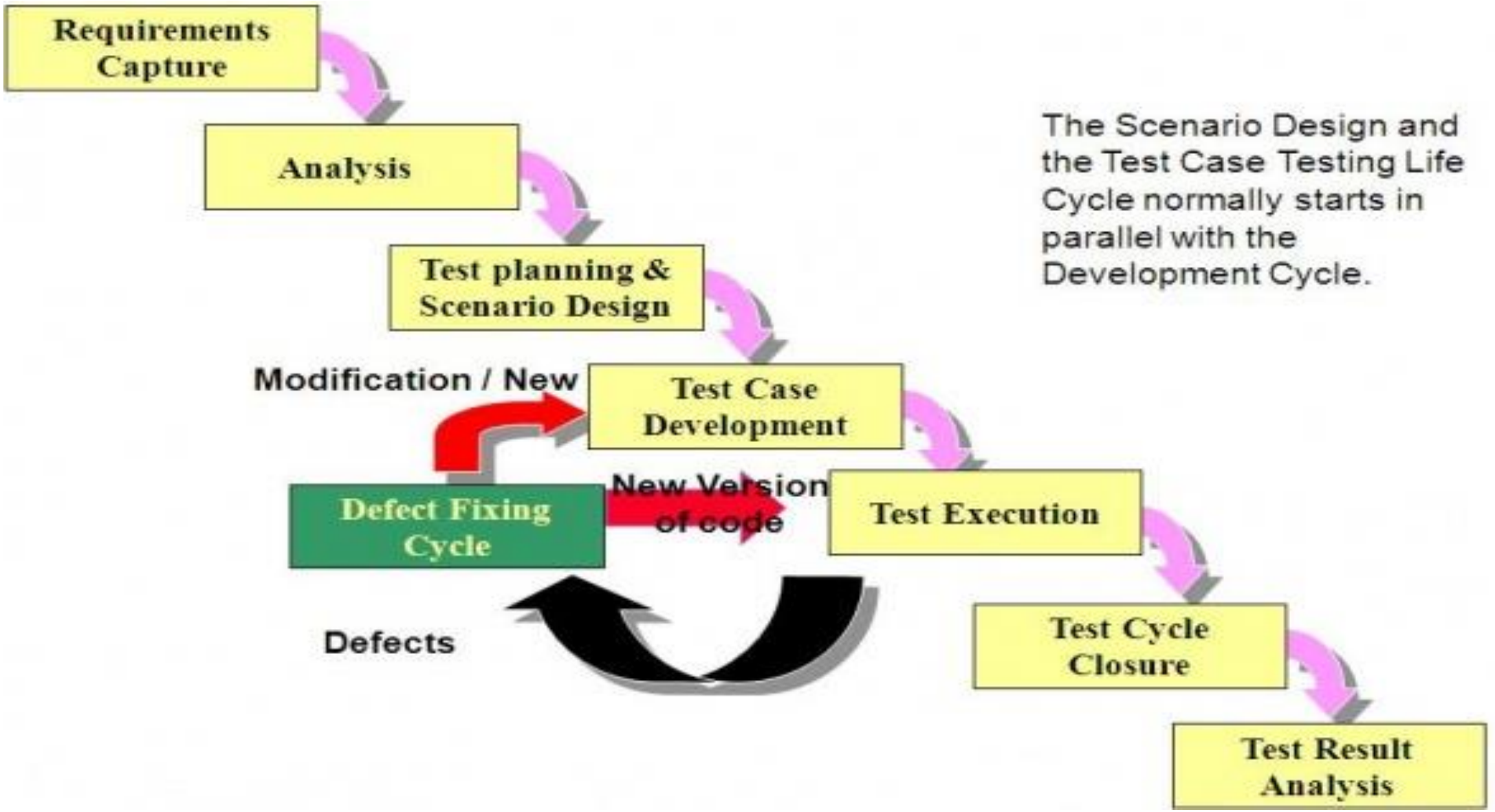
Appraisal Cost

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.

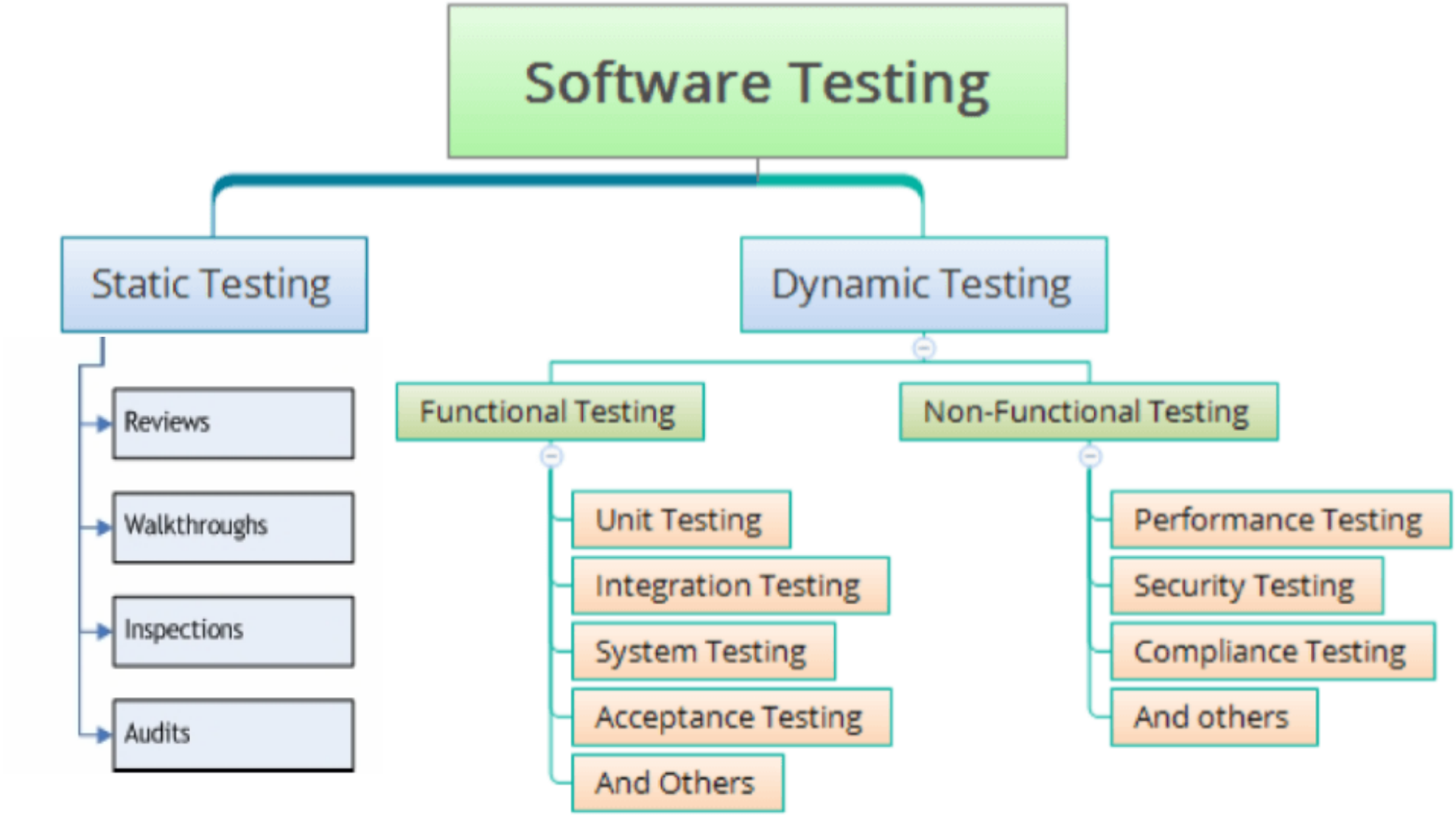
Failure Cost

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

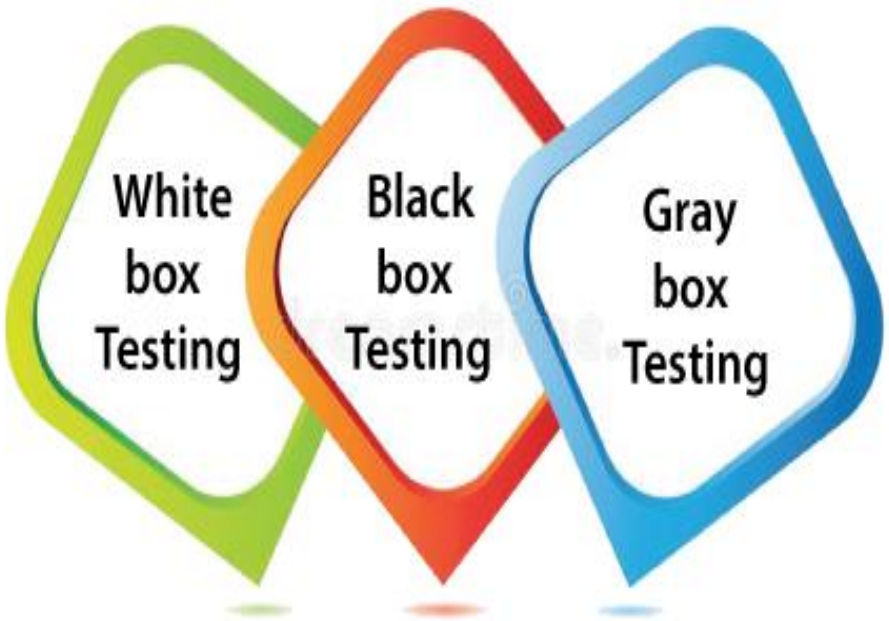


Software Testing Methodologies

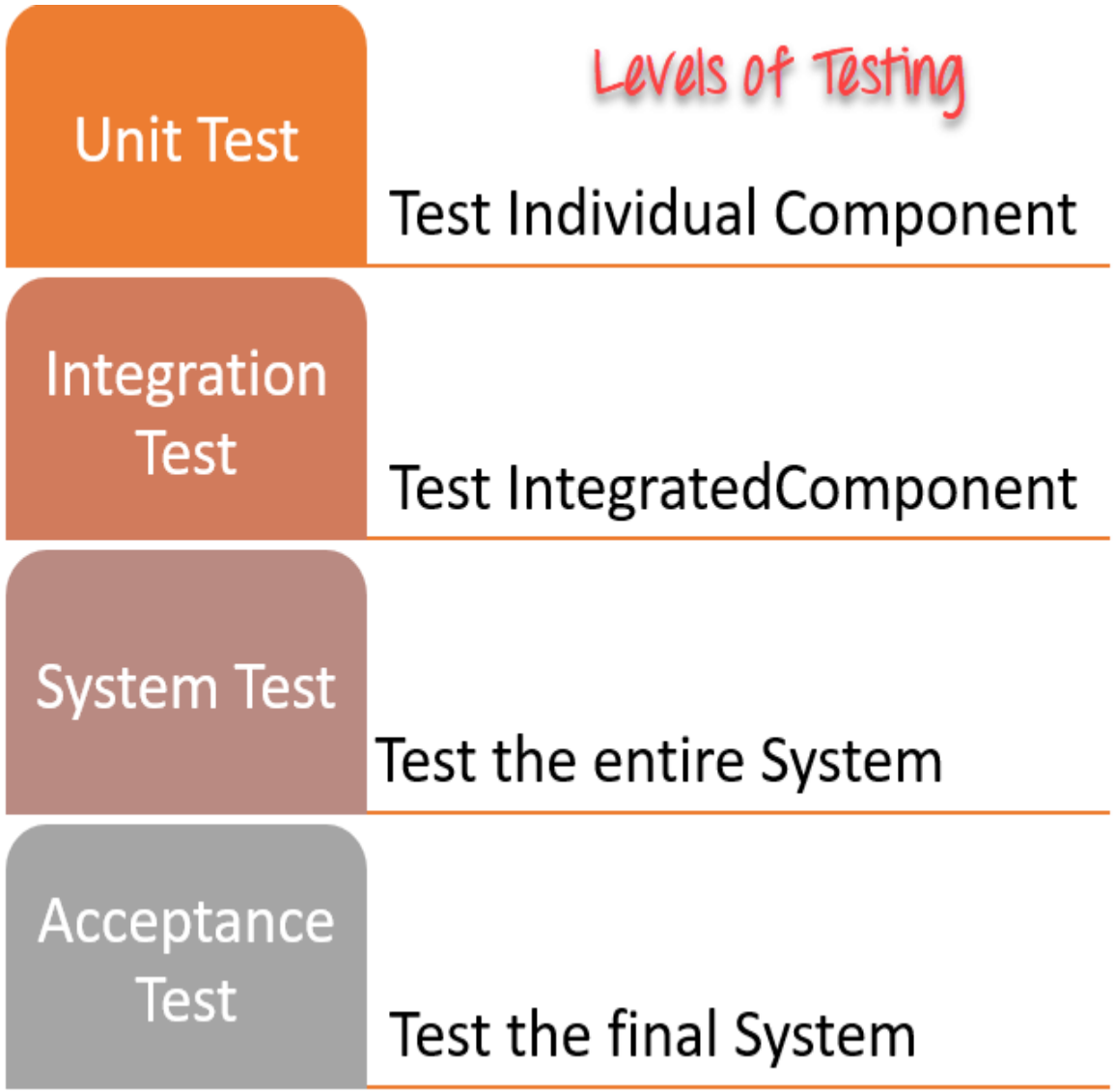
White-box testing
The white box testing is done by Developer, where they check every line of a code before giving it to the Test Engineer. Since the code is visible for the Developer during the testing, that's why it is also known as White box testing.

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.

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.

Thank You