MOHAN BABU UNIVERSITY
Sree Sainath Nagar, Tirupati 517 102

MODULE 1: SOFTWARE ENGINEERING AND SOFTWARE PROCESS

Engineering Layers; The Software Process, Software Engineering Practice, Software myths.
***Process models**: A Generic Process Model, Prescriptive Process Models-The Waterfall Model, Incremental Process Models, Specialized Process Models; The Unified Process, Agile Development-Agility, Agile Process, Extreme Programming (XP), Scrum, Lean Software Development (LSD), Dynamic System Development Method, Agile Modeling (AM), Agile Unified Process (AUP), The Cleanroom strategy.*

INTRODUCTION TO SOFTWARE ENGINEERING

1.1 Introduction

* As the importance of software has risen, millions of computer programmes have been developed and will need to be fixed, changed, and enhanced. Nobody could have predicted that these upkeep tasks would use up more time and energy than developing brand new software. Therefore, people in the software industry have been hard at work creating tools that make it easier, faster, and cheaper to build and maintain high-quality computer programmes. However, the software community has not yet succeeded in developing a single technology that can accomplish all of these goals. Therefore, in order for us to accomplish this, we require a framework that includes a procedure, a group of procedures, and a variety of instruments. The term "Software Engineering" refers to this particular framework.

1.2 The Evolving Role of Software

* Today software takes a dual role

=> As a Product

=> A vehicle for delivering a product

(1) As a Product:

- Hardware local connection to a network of computers makes available the processing power already present in the hardware.

- A programme is an information transformer if it produces, manages, acquires, modifies, or transfers information; this is true whether the programme is in a mobile phone or a supercomputer.

(2) <u>A vehicle for delivering a product</u>:

* Software provides one of the most important products of our time, information, by doing things like:

=> It transforms personnel data [for example, an individual financial transaction]

=> It manages business information to improve competitiveness

=> It serves as a gateway to global information networks [for example, the internet]

=>It provides a means of acquiring information in all of its forms

* Software provides one of the most significant products of our day, information, by doing things like:

=> It changes personnel data [for example, an individual financial transaction]

=> It handles business

1.3 <u>Adaptation of Software Engineering Practices:</u>

* Extraordinary advances in hardware performance,

=> deep changes in computing design,

=> vast increases in memory and storage capacity,

=> exotic input and output possibilities

- All of these factors have contributed to the evolution of increasingly complex and sophisticated computer-based systems.
- Sophistication and complexity yield desirable outcomes if the system works as intended, but can pose serious challenges if the opposite is true.
- Large software companies now employ entire groups of specialists, each of whom works on a specific aspect of the technology needed to complete a single application.
- The inquiries made of the lone programmer are identical to those made throughout the development of contemporary computer systems.

They are,

(1) What factors contribute to the protracted time required to finish software?

(2) What aspects of development add to the staggering costs?

(3) Why is it that we are unable to detect all of the faults in the software before releasing it to our clients?

(4) Why do we put forth a lot of time and energy to keep the programmes that are currently in place going?

(5) Why do we still struggle to accurately measure the amount of progress made when the software is being built and maintained?

- The fact that these inquiries are being made demonstrates that businesses are worried about software and its creation process.
- This worry has prompted the growth of software engineering practices.

1.4 Software:

Definition:

" It is a set of instruction that when executed provide desired features, functions and performance"

(Or)

" It is a data structure that enable the programs to adequately manipulate information"

(Or)

" It is a documents that describe the operation and use of programs"

1.5 Software Characteristics:

The following are some ways in which software features differ from hardware ones:

(1) Software is not produced in the conventional sense; rather, it is developed (or engineered).

(2) In both software and hardware production, high quality is accomplished through good design.

(3) Whereas software rarely has quality issues, hardware often does throughout the production phase.

(4) Although human beings are essential to the success of both endeavours, the correlation between manpower and output is very different.

5) In both cases, something must be built.

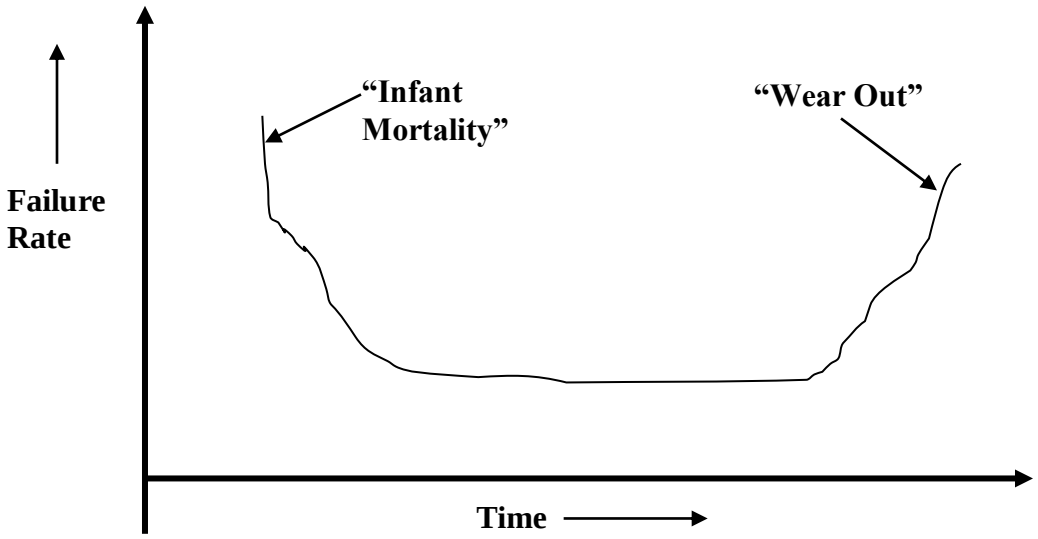
(6) There is an investment of time and energy in both

(2) Software doesn't "wear out"

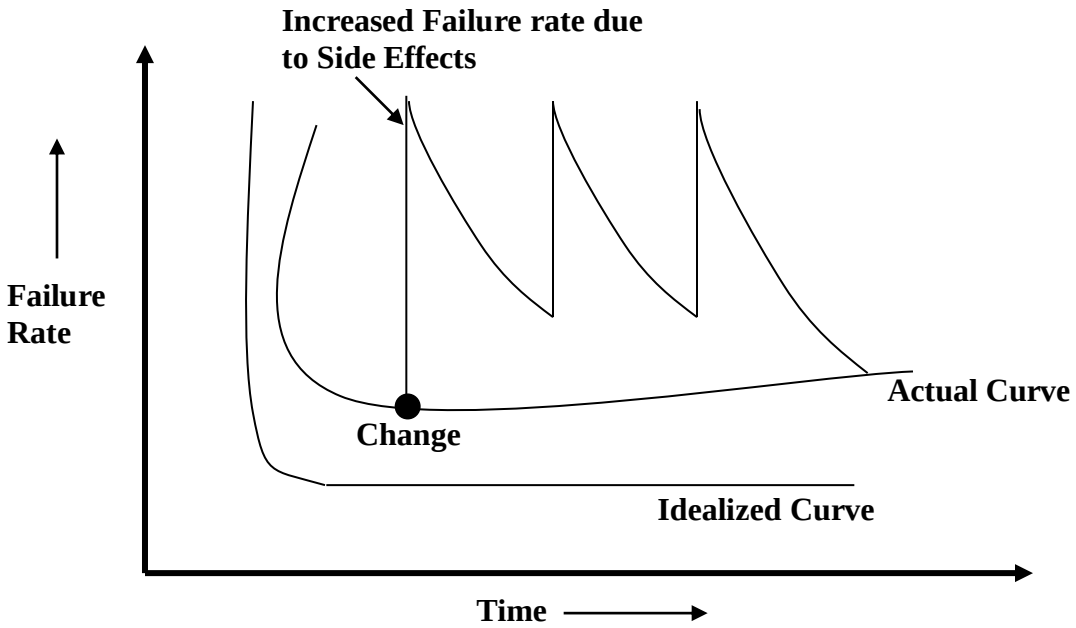
Once problems are addressed, the failure rate stabilises for a while.

Dust, Vibration, Abuse, Extreme Temperatures, and Time all add up to shorten the lifespan of hardware components.

This leads to an increase in the failure rate; alternatively, the hardware "wears out." The "Bathtub Curve" below illustrates this point.



* The failure rate curve for software should take the form of a "idealised curve," as it is not affected by environmental conditions in any way.



* In the beginning stages of a program's life, significant failure rates are caused by mistakes that have not yet been found.

* The curve will become flatter once these faults have been rectified (provided that no new flaws are introduced).

Therefore, it is abundantly evident that:

- Software undergoes deterioration rather than complete obsolescence; modifications occur throughout its lifespan; and the implementation of changes introduces errors.
- In order for the curve to revert back to its initial steady state failure, an additional modification is requested that induces another surge in the curve, thereby escalating the failure rate. Consequently, this degradation impacts the software's quality.
- Hardware components are replaceable with spare parts when they fail; however, software components are not replaceable. This distinction renders software maintenance a more challenging endeavour.
- Hardware maintenance is less complex in comparison to software maintenance.

(3) Although the industry is moving toward component – based construction, most software continues to be custom built

Component reuse is a natural element of the engineering process in the world of hardware, but in the world of software, it has to be performed on a large scale to be effective.

* It is recommended to build and develop a reusable software component that may be utilized in a variety of different programmes.

Example:

Present-day user interfaces are constructed using reusable components, which facilitate the following:

=> Window creation;

=> Pull-down menus; and

=> An extensive range of interaction mechanisms.

1.6 The Changing nature of Software

* There are seven primary classifications of computer software, each of which presents its own unique set of obstacles to software developers.

(1) System Software: It is a collection of programmes that were written in order to provide support for other programmes.

=> Some pieces of system software process information structures that are complex yet determinable, whereas other pieces of system software process data that is mostly undetermined.

As an example: Compilers, editors, and utilities for managing and organizing files

Characteristics:

=> Extensive connection with computer hardware

=> Prolonged and intensive use by a number of individuals

=> Structures of data that are complex

= Multiple connections to the outside world

=> Working in parallel at the same time

(2) Application Software:

* It is a self-contained application that caters to a particular organizational requirement.

* This piece of software can be used for either business or technical decision making. In addition to this, it can be used to exercise real-time control over the operations of a corporation.

Here's an example:

=> Processing of sales transactions at the point of sale

=> Real-time monitoring and management of the production process

(3) Engineering / Scientific Software:

Several applications utilise this software, including but not limited to

- astronomy,
- molecular biology, and a
- utomated manufacturing.

Contemporary scientific and engineering applications are increasingly diverging from conventional numerical algorithms.

Aspects of system simulation, computer-aided design, and other interactive applications have begun to incorporate real-time functionality.

(4) Embedded Software:

* This software is incorporated into a product.

* This software is integrated into a system. It is employed to execute and regulate functionalities and features for both the system and the end user. For example:

=> A microwave oven's keypad control corresponds to a variety of digital functions found in automobiles, including the gas lever and dashboard.

=> Such components as displays and the braking system are encompassed.

(5) Product-line Software:

* It is intended to give a certain competence, which may then be utilised by a wide variety of consumers.

* This software addresses both the general consumer market as well as the more specialized niche markets.

Here's an example:

=> Word processing

=> Spreadsheets

=> Presentation software Computer Graphics

=>Multimedia, Entertainment, and Other Things, etc.

(6) Web Applications:

* The software's uses are extremely varied.

* Web applications are transforming into complex computing environments that incorporate corporate database and business application in addition to providing end users with stand-alone features, computing capabilities, and content.

(7) Artificial Intelligence Software:

* This piece of software employs methods that are not numerical in nature in order to solve difficult situations. Applications that fall under this category include:

1) Robotics;

2) Expert Systems;

3) Pattern Recognition; and

4) Pattern Recognition.

The demonstration of theorems and the practise of games

1.7 Software Myths:

* Beliefs about software

* The procedure used to build it, which can be traced back to the earliest days of computers

* The myth – has a number of characteristics that have contributed to them becoming insidious [i.e. proceeding inconspicuously but harmfully]

* For example, myths give the impression of being factual claims that are rational [and sometimes do contain aspects of truth].

(1) Management Myths:

* Managers in most fields are frequently under pressure to keep budgets in check, prevent schedules from falling behind, and enhance quality

* Software managers frequently cling to the assumption that certain software myths are true in the hopes that this will reduce the amount of pressure they are under.

Myth 1:

There is already a book in our possession that is loaded with guidelines and processes for the construction of software....Won't that provide the information that my people require to make informed decisions?

Reality:

=> The book of standards might very well exist.....However, is it put to use?

=> Are those who work in the software industry aware of its existence?

=> Does it adhere to the practices of contemporary software engineering?

=> Does it have everything? Is it amenable to change?

=> Is it possible to streamline it in order to shorten the amount of time it takes to provide while still concentrating on the quality?

* The answer to each of these questions will most likely be "No" in the majority of instances.

Myth 2:

If we fall behind schedule, we have the ability to add more programmers and make up the time [this strategy is sometimes referred to as the "Mongolian Horde Concept"].

Reality:

* Adding personnel to a software project that has already been delayed results in the project being further behind schedule.

* By educating the newly added members, the time allocated for productive development is diminished.

* While it is possible to add personnel, it is crucial that such additions are conducted in a methodical and coordinated fashion.

Myth 3: By delegating the software development to a third-party firm, I can simply sit back and unwind while the project is executed.

Reality:

Failure to comprehend internal software project management and control will inevitably result in difficulties for an organisation when it attempts to outsource software projects.

(2) Customer Myths:

The clientele requesting computer software might consist of,"
=> A person"
=> An technical crew
=> A sales or marketing department (Or)
=> an external provider
Customer misconceptions regarding software are common.
* Myths result in erroneous anticipations (on the part of the clientele) and, ultimately, developer discontentment

Myth 1:
It suffices to commence programme writing with a broad statement of objectives. The details can be completed later.
Reality:

- A statement that is equivocal, or has two meanings, gives rise to a multitude of complications.
- However, unambiguous statements can only be generated via consistent and efficient communication between the client and the developer.
- Consequently, it is not always feasible to formulate statements of requirements that are exhaustive and consistent.

Myth 2:

Project requirements are subject to constant evolution; however, modifications can be readily integrated due to the adaptable nature of software.
Reality:

- Early requests for requirements modifications, prior to the commencement of design or code, result in a comparatively minimal cost impact.
- Requests for requirement modifications made after design (or code) has commenced have an excessively high cost impact.

- The proposed change has the potential to induce disruption, such as violent change or disturbance, which may necessitate the allocation of supplementary resources and substantial modifications to the design.

(3) Practitioners Myths

Myth 1:

After completing the program's development and testing, our work is complete.

Reality:

***** According to industry data, between sixty and eighty percent of all software development efforts will be repeated subsequent to the initial delivery of the product to the client.

Myth 2:

Before the programme is operational, it is impossible for me to evaluate its quality.

Reality:

* Implement one of the effective software quality mechanisms during the project's inception.

Software quality evaluations exhibit greater efficacy in identifying specific categories of software errors when compared to testing.

Myth 3:

The working programme is the sole deliverable work product that ensures a successful undertaking.

Reality:

Work programmes are components of software configuration, which comprises a multitude of elements.

However, documentation merely serves as a cornerstone for effective software engineering and support.

Myth 4:

Inevitably, software engineering will force us to generate copious amounts of superfluous documentation, which will impede our progress.

Reality:

Software engineering focuses on producing high-quality software rather than mere document creation.

Thus, improved quality results in decreased rework.

* Decreased rework results in expedited delivery

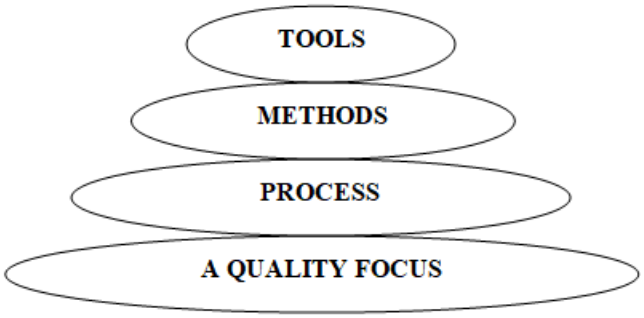
In closing,

The fact that numerous software professionals acknowledge the fallacy (erroneous belief) of software myths will indirectly contribute to the propagation of ineffective management and technical practices.

However, acknowledging the realities of software is the initial stride in developing pragmatic resolutions for software engineering.

A GENERIC VIEW OF PROCESS

1.8 Software Engineering – A Layered Technology:



(i) Quality Focus:

* Every engineering approach, software engineering included, must be founded upon an organisational dedication to quality. Total quality management facilitates ongoing process improvement, which in turn fosters the creation of efficient software engineering methodologies. A quality-oriented approach serves as the foundation that bolsters software engineering.

(ii) Process:

The process layer serves as the cornerstone of software engineering, facilitating the logical and expeditious development of computer software by acting as a cohesive agent between the technology layers.

It establishes a framework that is necessary for the following:

=> Effective delivery of software engineering technology

* The software process serves as the foundation for management control.

It provides the framework within which

=> Technical methods are applied

=> Work products are produced [i.e. models, documents, data, reports, forms etc..]

=> Milestones are established

=> Quality is ensured and change is probably managed

(iii) Methods:

The document encompasses a wide range of tasks, which comprise:

=> Communication

=> Requirement analysis

=> Design modeling

=> Program Construction

=> Testing and support

* It depends on a set of basic principles that

=> Govern each area of the technology

=> Include modeling activities and other descriptive techniques

(iv) Tools:

* It offers automated or partially automated assistance for the procedures and processes.

* When tools are incorporated, data generated by one tool becomes accessible to another.

1.9 A Process Framework:

* It finds a small set of framework tasks that can be used on all software projects, no matter how big or complicated they are.

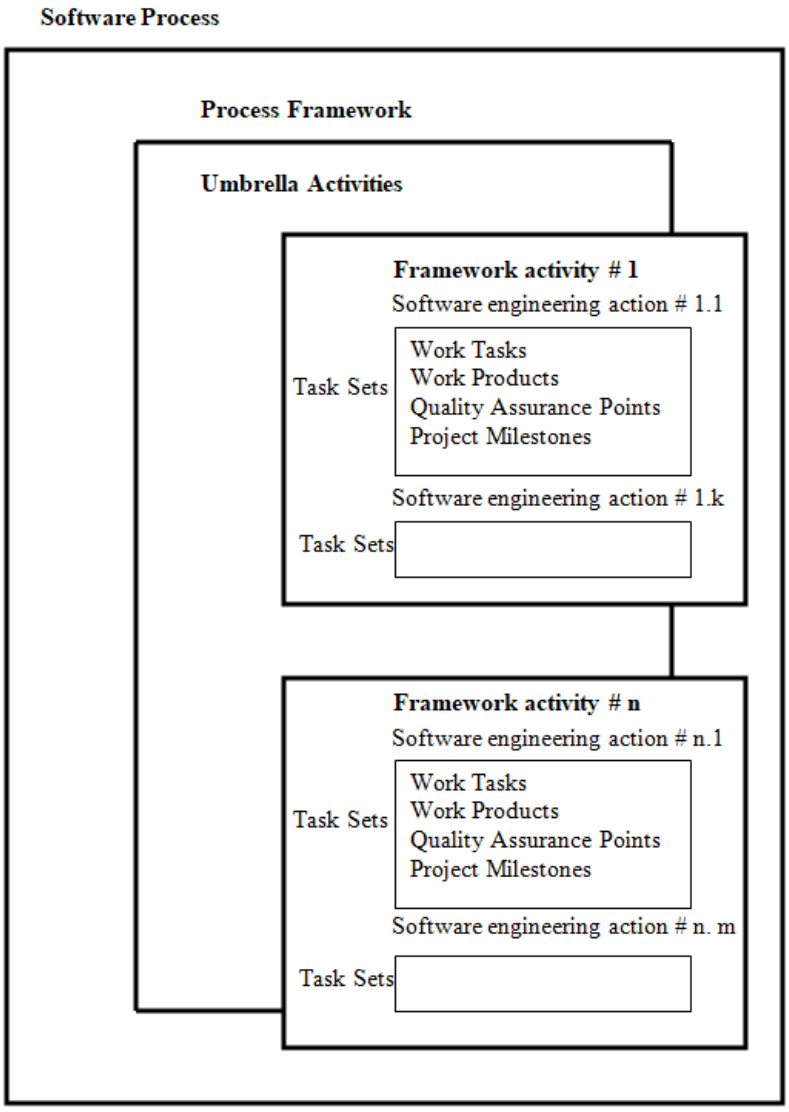
*The process framework has a group of tasks that can be used throughout the whole software development process.

Framework Activity:

* It has a set of software engineering actions [a group of related jobs that come together to make a big piece of software engineering work].

Design is an action in software engineering.

Each action has its own set of tasks that need to be done. These tasks do some of the work that the action implies.



A Process Framework

1.10 Generic Framework Activities:

(1) Communication:

Constant interaction and cooperation with the client is required, as well as the gathering of requirements and other relevant tasks.

(2) Planning:

* It describes the

=> Technical tasks to be conducted

=> The risks that are expected

=> Resources that will be required

=> Work products to be produced

=> Work Schedule

(3) Modeling:

It explains how to build models that help both developers and clients visualise and discuss software's desired features and functionality.

(4) Construction:

* It combines

=> Code generation [either manual (Or) automated]

=> Testing [Required uncovering errors in the code]

(5) Deployment:

* The customer receives the software and evaluates it.

The customer then gives feedback based on the evaluation.

These general framework tasks can be used during the

=> Development of small programs

=> Creation of large web applications

=> Engineering of large complex computer based systems

*The way the software is made is different each time, but the tasks that make up the framework stay the same.

1.11 Umbrella Activities:

(1) Software project tracking and control:

* Let the software team make work on the project plan. If you need to, take steps to stay on schedule.

(2) Risk Management:

*Look for risks that could affect how the job turns out or the quality of the product.

(3) Software quality assurance:

*It lists and carries out the tasks needed to make sure the quality of software

(4) Formal Technical Reviews:

* Get rid of any mistakes that you find before moving on to the next action (Or) activity.

(5) Measurements:

To aid the team in delivering software, it specifies and collects process, project, and product measures, and it can be used in conjunction with other frameworks and overarching tasks.

(6) Software configuration management:

Effectively handles change impact management for software development.

(7) Reusability management:

* It sets up a way to make parts that can be used again and again.
* It sets rules for reusing work products.

(8) Work product preparation and production:

* It includes the things that need to be done to make work goods, like

=> Models

=> Documents

=> Logs, forms and lists

PROCESS MODELS

2.0 Process Models – Definition

* It's a separate collection of things you have to do, accomplish, and produce in order to create high-quality software.
* While not flawless, these process models do provide a helpful framework for software engineering projects.

2.1 THE WATER FALL MODEL

* The term "classic life cycle" is occasionally used to describe it.

It recommends a methodical and sequential approach to software development, starting with a detailed specification of the needs of the end user and continuing all the way through

=> Planning

=> Modeling

=> Construction and

=> Deployment

Problems encountered in waterfall model:

(1) In practise, projects rarely progress in a linear fashion. Consequently, the team's progress is muddled by the constant stream of changes.

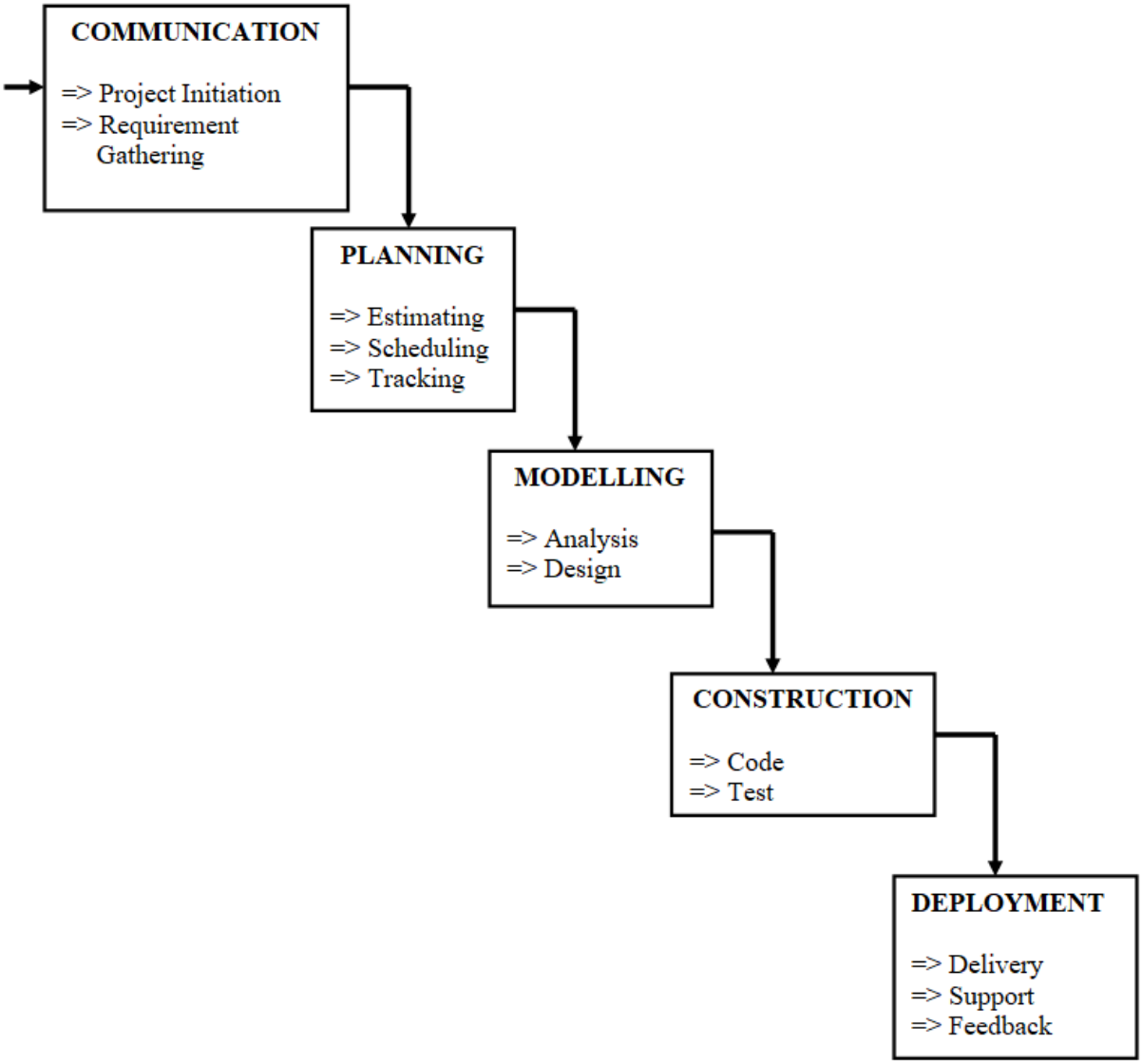
(2) The consumer often has trouble articulating their needs in detail;

(3) The customer must be patient

* Because of the sequential structure of the water fall model, "Blocking State" occurs when certain members of the project team must wait for others to finish dependent tasks.

* In certain contexts, the water-fall model can be utilized effectively as a model for the process.

=> Requirements are fixed and work is to proceed to completion in a linear manner



2.2 INCREMENTAL PROCESS MODELS

* This model applies linear sequences in a staggered form as calendar time progresses

* This model combines aspects of the water fall model that are applied in an iterative fashion

* The incremental model combines elements of the water fall model

* Each linear sequence results in the production of software "increments" that can be delivered.

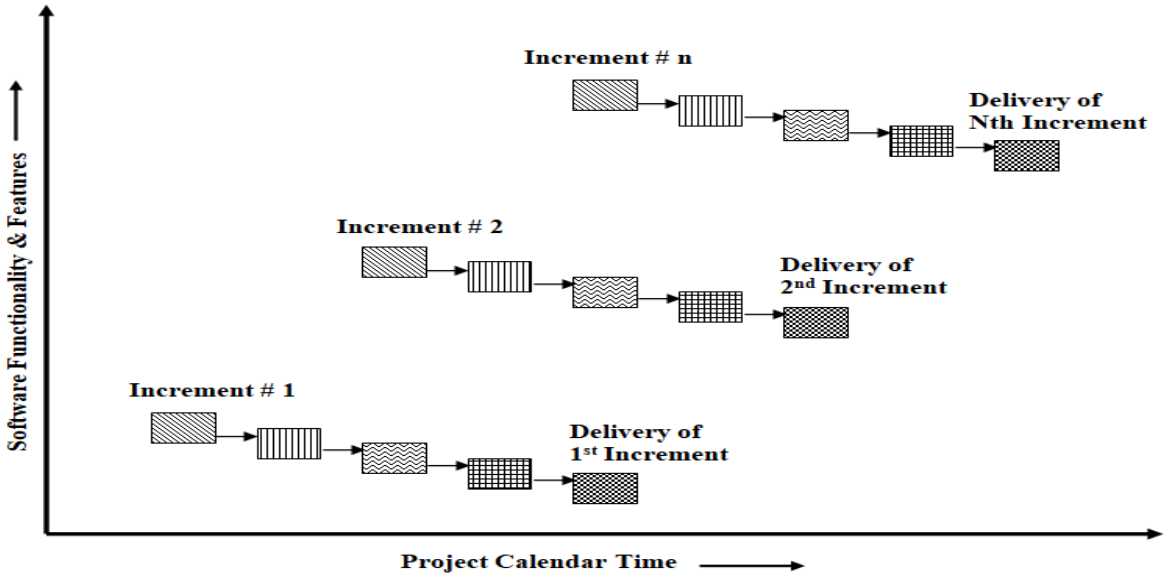
Main Point:

* The initial iteration of an incremental model is referred to as the "CORE PRODUCT" when the model is used.

* meaning that the fundamental needs have been met, but the primary extra features have not been given

* Either the fundamental product is put through extensive testing by the customer, or the customer uses it.

* As a direct outcome of the evaluation, a strategy for the subsequent increment is prepared.



=> Communication

=> Planning

=> Modeling {Analysis, Design}

=> Construction {Code, Test}

=> Deployment {Delivery, Feedback}

Unlike the prototyping model, which prioritises the delivery of a functional product with each increment, the incremental model focuses on the delivery of an operational product with each increment, ensuring that the product fully meets the needs of the customer before it is considered complete.

In contrast to the prototype methodology, the incremental model focuses on adapting the original product to new circumstances.

* When personnel is unavailable for a comprehensive implementation by the business deadline that has been imposed for the project, this strategy is particularly effective.
* It is possible to implement early increments with a smaller number of individuals. In the event that the core product is favourably received, extra personnel may be added in order to implement the subsequent increase. It is possible to plan for increments in order to mitigate technological risks.
* For instance, the production of a significant quantity of brand new hardware is now underway, although the exact date of its release is unknown.
* Therefore, it is important to arrange early increments in a way that prevents the use of this hardware. This will make it possible for end-users to receive partial functionality without an excessive amount of delay.

AN AGILE VIEW OF PROCESS

Agile software engineering is a methodology that fosters both the early incremental creation of software and the pleasure of customers.

Software engineers and other project stakeholders, such as managers, end-users, and consumers, collaborate as part of an agile team. This means that the team is responsible for its own organisation and is in charge of its own fate.

The members of an agile team are better able to communicate with one another and work together more quickly.

• Agile software engineering is an acceptable alternative to conventional software engineering for certain categories of software and certain types of software projects. This is because agile software engineering prioritises collaboration and iterative development.

• Customers and Software Engineers that have accepted the agile concept share the same point of view, which is that the only really important work product is an Operational "Software increment" that is provided to the customer on or before the appropriate commitment date.

• If the agile team comes to the conclusion that the process is successful and the team is able to produce increments of software that the customer finds satisfactory.

What Is Agility ?

• Modifications to the software under development, adjustments to team members, modifications brought about by new technology, and modifications of any kind that could

affect the product they created or the project that develops the project are all instances of the kinds of modifications to which an agile team can adapt in a way that is appropriate.

• An agile team is aware that software is created by people working in groups, and that the success of the project depends on the abilities and talents of these people cooperating.

• Agility encompasses more than just the ability to adapt quickly to change. In addition to that, it incorporates the agile way of thinking.

What Is An Agile Process?

The bulk of software development projects are based on three key assumptions, and an AGILE SOFTWARE PROCESS is characterised by the way it handles these assumptions.

1. It is impossible to determine in advance which software requirements will continue to be necessary and which will be replaced by new ones. It is similarly challenging to anticipate how the priorities of a customer will shift as a project moves forward.

2. The phases of design and production are frequently combined in the creation of different kinds of software. i.e. It is recommended that both processes be carried out simultaneously so that design models can be validated as they are being developed. It is challenging to make an accurate estimate of the amount of design work that must be completed before construction can be used to validate the design.

3. The phases of analysis, design, construction, and testing are not as predicable as we would like them to be (based on the planning).

• Based on these three presumptions, we are able to assert that the process's success resides in its adaptability (to rapidly shifting technical conditions and project parameters).

• Flexibility is an absolute requirement for an agile process.

• The iterative approach to software development known as agile needs to change.

• The agile team needs feedback from customers in order to achieve incremental goals.

• The iterative methodology enables the customer to frequently evaluate the software increment, provide the software team with any necessary input, and have some say in the process changes that are made to meet the feedback provided by the customer.

Those who wish to attain agility are required to adhere to the following 12 principles, as defined by the Agile Alliance:

1. The earliest and most consistent delivery of useful software is our first and foremost concern in order to fulfil the requirements of our customers.
2. Be prepared to modify plans in response to changing needs, particularly as development progresses. Agile processes give the client a competitive edge by allowing them to adapt to change.
3. Regularly deliver functional software, giving attention to completion in the least amount of time. A few weeks or a few months could pass in this case.
4. Business experts and developers work together every day for the duration of the project.
5. Focus on individuals with a strong sense of motivation. Have faith in their abilities to do the task and give them the environment and assistance they need.
6. Direct, in-person communication is the most effective and beneficial means of sharing information with other team members and members of a development team.
7. The best measure of success is having software that functions as intended.
Using agile methodologies promotes sustainable development. The speed is anticipated to be steady indefinitely, and sponsors, developers, and consumers should all be able to maintain it.
9. You can improve agility by keeping an eye on sound design and technical excellence.
Simplifying involves maximizing the amount of work that is not done, which is a key element.
11. The best architectures, specifications, and designs are created by self-organizing teams.
12. On a regular basis, the team reflects on how it may become more efficient and then adapts and changes its behavior to take those ideas into account.

13. Using an agile approach can be advantageous for any software development process. The method places emphasis on incremental delivery, with the goal of providing clients with functional software as soon as feasible while considering the nature of the product and the operational context.

AGIL E PROCESS MODELS

- There are thousands of defunct process descriptions and techniques, modelling approaches and notations, tools, and technologies littered throughout the history of software engineering.
- With the introduction of a wide variety of agile process models, each of which is competing for approval within the community of software developers.
- The term "agile" can refer to a variety of different process paradigms. There are a lot of parallels to be drawn between these different approaches. the aspects of each approach that set them apart from one another.
- The Manifesto for Agile Software Development and the concepts of agility are adhered to by each and every one of the Agile models.

- **EXTREME PROGRAMMING(XP)**

• XP, short for Extreme Programming, is the widely adopted agile development methodology that follows an object-oriented approach. It consists of a collection of principles and practices that are carried out in the context of four distinct framework activities: planning, designing, coding, and testing.

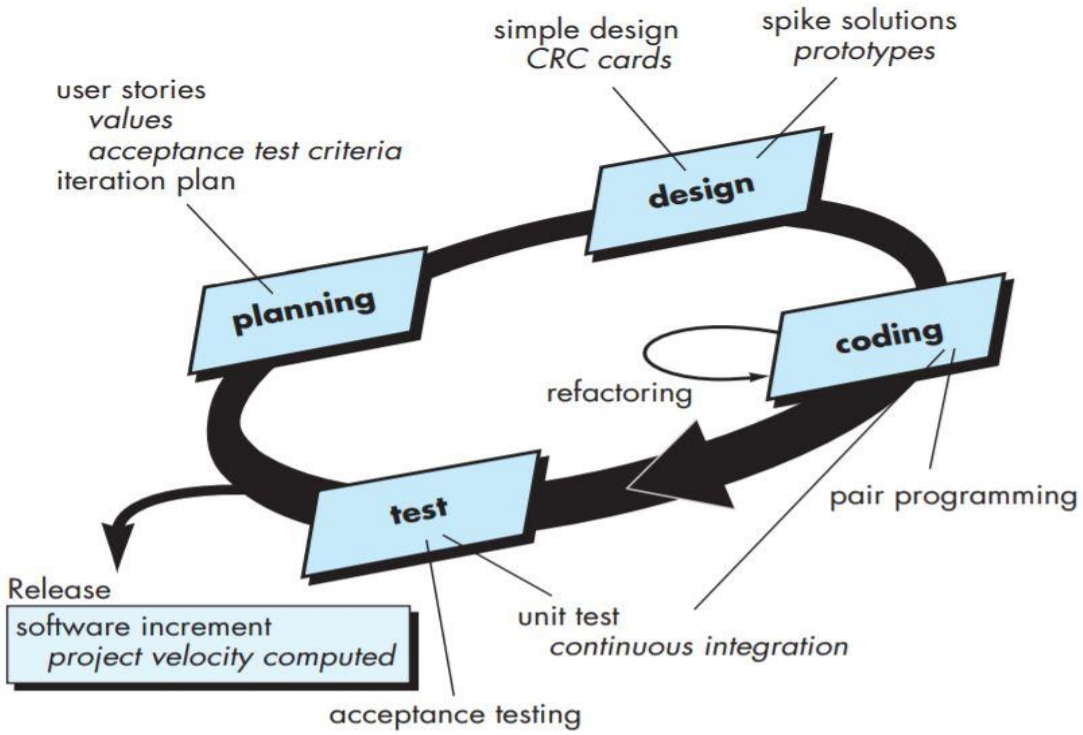
PLANNING

• The process of planning begins with the preparation of a set of Stories, which are also referred to as User stories and that explain the features and functionality that are necessary for software that is to be produced.

• The customer is responsible for writing each tale, often known as a "Use-Case," and placing it on an index card.

• The customer gives the narrative a VALUE (priority) based on the total business value of the feature or function being discussed.

• After that, members of the XP team evaluate each story and assign a cost for it, measured in terms of the number of weeks required for its creation.



• If the narrative will require more than three weeks of development time, the customer will be asked to break the tale up into smaller stories, and the value and cost will be assigned to each of those stories individually once again.

• The new stories may be written whenever the author chooses.

• The XP team and its customers collaborate to decide how to organise individual user stories into the next release, also known as the next software increment. This decision is made by the XP team.

• Once a release has been committed to, the XP team will place an order for the stories that will be developed in one of the following three ways:

1. All of the stories will be put into action right away (within a few weeks).
2. The tasks associated with the stories that have the greatest potential impact will be prioritised higher in the timetable and completed first.

3. The stories with the highest potential for failure will be placed to the front of the schedule and implemented first.

DESIGN

• The design of XP adheres to the "Keep it Simple" (KIS) approach. A less complicated representation is preferable over a more complicated design.

• The design offers implementation assistance for a story exactly as it is stated; nothing less and nothing more than that.

• XP promotes the utilization of CRC cards, which stand for "Class-Responsible Collaboration." These cards identify and organize the object-oriented classes that are pertinent to the currently active software increment.

• The only design work products that were generated as a result of the XP process were the CRC cards.

• Refactoring, a building approach that doubles as a design technique, is encouraged by the XP game mode.

• The term "REFACTORING" refers to a design process that is ongoing during the construction of the system.

CODING

• According to the XP approach, once a team has finished working on the stories and the preliminary design work, they should not move on to coding but rather develop a set of unit tests that are included into the software increment that is presently being worked on. The XP methodology proposes that once the team has finished working on the stories and the preliminary design work, they should not go on to work on coding.

• Once the unit test has been constructed, the developer will have a much easier time focusing on the requirements that need to be met in order for the unit test to be passed. • Once the programming has been completed, the code may be instantly put through unit testing, which provides the developers with immediate feedback.

• PAIR PROGRAMMING, XP recommends having two people cooperate on the production of the story's code while working at the same computer work station. These two people should be working together. This provides a means for addressing issues in real time and checking the quality of work in real time while it is being performed.

EX: It's possible that one person will focus on the specifics of the coding for a segment of the design while another looks over their shoulder to make sure the coding standards are being adhered to.

• The created code will "FIT" into the larger framework of what the tale is about.

• The integration work is the duty of the pair programmers. This technique of continuous integration helps to avoid problems with compatibility and interfacing, and it creates a "SMOKE TESTING" environment, which helps to expose defects at an earlier stage.

TESTING

- The newly developed unit tests ought to be put into practise by making use of a framework that gives rise to the possibility of their being automated. When code is modified in any way, this should encourage the use of a regression testing technique.
- Integration and validation testing of the system can be performed on a daily basis now that the unit tests have been arranged into a "Universal Testing suit." The XP team receives a continuous indication of progress from it, and it also has the potential to raise early warning signs if things start to deteriorate.
- XP acceptance tests, also known as customer tests, are tests that are specified by the customer and concentrate on the overall features and functionality of the system that are reviewed by the client.
- Acceptance tests are created from user stories after they have been incorporated into a product release.
- Once the XP team has completed the delivery of the first release of the project, they will compute PROJECT VELOCITY, which is the number of customer stories that were implemented during the first release. After then, one may make advantage of Project Velocity to

1. Contribute to the estimation of delivery dates and the release schedule for subsequent versions and

2. Determine whether an over commitment has been made for all of the stories that are part of the overall development project. In the event that an over commitment takes place, either the substance of the release or the end-delivery dates will be adjusted.

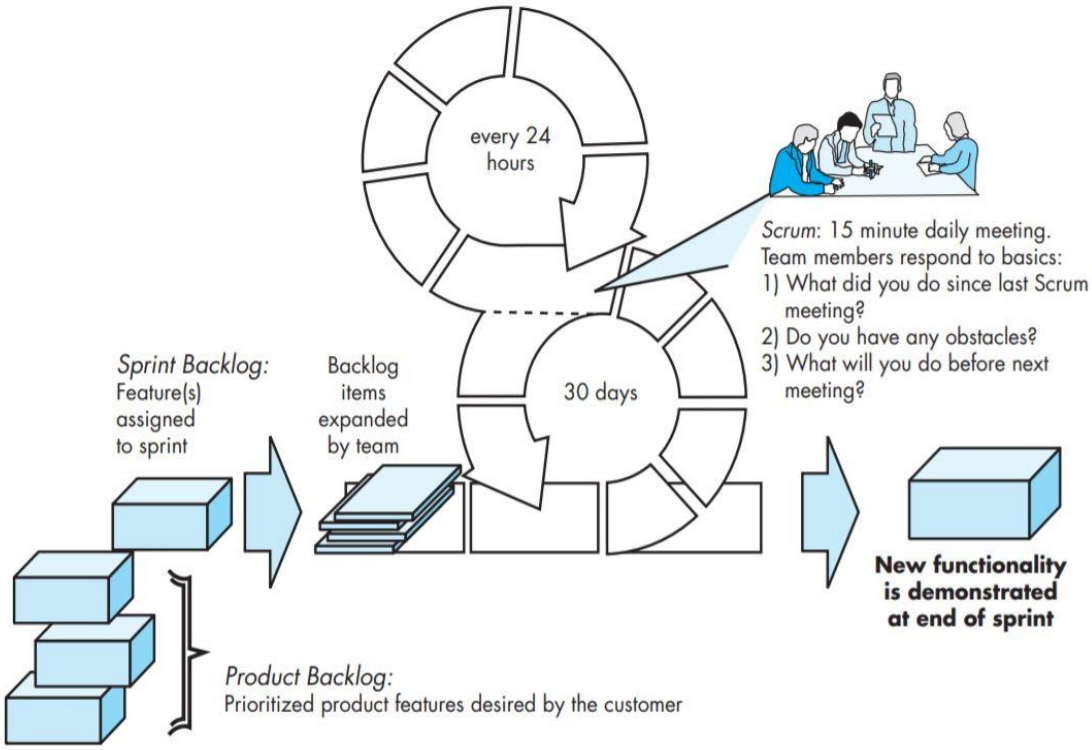
- As the development process progresses, the client has the option of adding new tales, altering the value of an existing narrative, splitting existing stories, or removing existing stories. After then, the XP team reviews all of the releases that are still to come and adjusts its plan accordingly.

SCRUM

SCRUM principles are consistent with the agile manifesto :

- Small working teams are organised to make the most of their communication, minimise their overhead costs, and make the most of their opportunity to share their knowledge.
- In order to "ensure the best product is produced," the process needs to be flexible enough to accommodate both changes in technology and in business.
- The procedure results in frequent software increments "that can be inspected, adjusted, tested, documented, and built upon."
- The work of development and the individuals who carry it out are separated "into clean, low coupling partitions, or packets"
- As the product is being assembled, testing and documentation are carried out in a continuous manner.
- The SCRUM methodology affords users the "flexibility to declare a product 'done' whenever it is necessary to do so."
- The SCRUM principles are utilised to direct the development activities that take place within a process that also includes the activities of the following framework: requirements, analysis, design, evolution, and delivery.
- Within each activity of the framework, work tasks take place according to a pattern of processes known as Sprint.
- The work that is completed during a Sprint (the number of sprints necessary for each framework activity will vary based on the complexity and scale of the product) is suited to the problem that is currently being worked on. Additionally, the SCRUM team defines the work and frequently modifies it in real time. The following diagram outlines the general steps involved in the SCRUM process.
- SCRUM places an emphasis on the utilisation of a collection of "Software process patterns," which have been demonstrated to be effective for projects that have tight timeframes, fluctuating requirements, and high business criticality.

- A group of different kinds of development work is outlined by each of these process patterns.
- A prioritised list of project requirements or features that generate business value for the customer is referred to as a backlog. At any time, additional items may be added to the Backlog. The Backlog is evaluated by the product manager, who then updates the priorities as necessary.
- Sprints are comprised of work units that must be completed within a specified amount of time in order to fulfil a need that has been outlined in the Backlog. During a Sprint, the items in the Backlog are frozen, which enables the members of the team to operate in an environment that is both short-term and consistent.



• Scrum meetings are brief meetings that are held every day by the scrum team. During the meeting, there are three important issues that are discussed, and each member of the team provides an answer.

o Since our previous gathering, what have you been up to?

o What kinds of challenges are you facing right now?

o What do you hope to have accomplished by the time we get together again as a team?

• The gathering is directed by a group leader known as a "Scrum master," who also evaluates the contributions made by each individual. The team is able to identify potential difficulties at the earliest possible stage thanks to the Scrum sessions.

• The regular meetings facilitate "knowledge socialization," which in turn contributes to the development of a structure that allows the team to organise itself.

• Demos - Deliver the software increment to the customer so that the customer can showcase and evaluate the functionality that has been built. This allows the customer to provide feedback on the functionality.

• The demonstration might not have all of the functionality that was anticipated, but it should be possible to implement these features within the time constraint that was set.

DYNAMIC SYSTEMS DEVELOPMENT METHOD(DSDM)

- The DYNAMIC SYSTEMS DEVELOPMENT METHOD (DSDM) is an agile software development approach that offers a structured framework for creating and maintaining systems that adhere to strict time restrictions. This is achieved through the implementation of incremental prototyping inside a controlled project environment. This is achieved through the utilization of the DYNAMIC SYSTEMS DEVELOPMENT METHOD (DSDM) agile software development approach.
- One arrives at the DSDM principle by deriving it from the pareto concept.
- Much like XP, the DSDM employs an iterative software development process. This allows for the delivery of 80% of an application in the same amount of time as it takes to produce the full product. In each iteration, the DSDM methodology adheres to the 80% rule, which states that only enough work is done to meet the requirements of the next increment. The remaining work is finished at a later time when more business requirements are discovered or modifications need to be accommodated.
- The DSM consortium, which is an association of enterprises for some specific purpose, is a global organization. This organization has devised an iterative process model that is referred to as the DSM life cycle.
- The DSM life cycle outlines three distinct repeating cycles, which are preceded by two more activities that are part of the life cycle.

Feasibility study – In order to determine if an application is a good fit for the DSDM process, it is necessary to first establish the fundamental business requirements and constraints connected with the application.

Business Study – Establishes the functional and information requirements that must be met in order for the application to be able to give value to the company, as well as specifies the fundamental architecture of the application and outlines the needs that must be met for the application to be maintainable.

Implementation – Installs the most recent software update available into the environment in which it is now running. It is essential to keep in mind that

1) The increment might not be completely done, or

2) Changes might be required while the incremental is being implemented. Both of these scenarios are crucial to keep in mind.

- DSDM and XP can be coupled to give a strategy that defines a stable process model with the nuts and bolts practises (XP) that are used to construct software increments. This combination method is known as a combination approach.

AGILE MODELING(AM)

Software engineers often find themselves in the position of having to construct massive, mission-critical systems for businesses.

Modelling the scope and complexity of such systems is necessary in order to achieve the following goals:

1. ensuring that all stakeholders have a better understanding of what has to be achieved.

2. The individuals who are responsible for finding a solution to the problem can be divided up into groups that are more likely to be successful. And

3. The quality can be evaluated at each stage of the engineering and construction processes for the system.

- Many other software engineering modelling methods and notations have been suggested for use in the process of analysis and design; however, despite the major virtues of these methods, it has been found that they are difficult to implement and demanding to maintain.

• The "Weight" of these modelling methodologies is a contributing factor to the problem. When we refer to this, we are referring to the amount of notation that is required, the degree of formalism that is advised, the complexity of the models for larger projects, as well as the difficulty in sustaining the model as change occurs.

• The only methods that provide a sustainable benefit for larger projects are the analysis and the design modelling.

• The AGILE MODELLING was implemented to make the projects intellectually more manageable.

A description of the AGILE MODELLING (AM) is as follows:The acronym "AGILE MODELLING" (AM) refers to a methodology that is based on practises and is used to efficiently model and describe software-based systems. Agile modelling is a set of concepts, ideals, and practises for modelling software that may be utilised in a software development project in an efficient and unburdensome manner. These can be utilised in the context of agile software development.

AM proposes a wide variety of "CORE" and "SUPPLIMENTARY" modelling principles, which are what set AM apart from other modelling approaches.
Design with a clear end in mind A developer using AM should have a goal in mind (such as "to communicate information to the customer") before commencing work on the model. Once the purpose of the model has been established, both the form of notation that will be applied to it and the level of detail that it must contain will become more clear.
Make use of a variety of models. There is a wide variety of both models and notations that can be utilized in the process of describing software. Only a small portion of the whole is required for the majority of tasks.
Don't pack too much As the work on software engineering progresses, you should only employ those models that will only deliver a value over the long run. Every

piece of work that is preserved must be brought up to date whenever there is a change.

The actual content is more essential than how it is represented. The audience that the modelling is meant for should receive information from it. A model with correct syntax and notation that does not transfer any information to the target audience is not as valuable as one with incorrect syntax and notation that does not target audience at all.

Be familiar with the models you generate and the tools you use to do so. Gain an understanding of the techniques that are used to develop each model, as well as the benefits and drawbacks of each model.

Adapt on a local level. The modelling strategy ought to be modified so that it caters to the requirements of the agile team.

AGILE UNIFIED PROCESS

- Agile Unified Process follows a serial "in the large" and an iterative "in the small" methodology.
- By utilising the traditional activities of the UP stages, which are "Inception," "Elaboration," "Construction," and "Transition," respectively.
- Iteration is used throughout each of the actions by the team in order to achieve agility and to ensure that end users receive meaningful software increments as quickly as feasible.
- The actions listed below are taken into consideration by this edition of the AUP:

- Modeling
- Implementation
- Testing
- Deployment
- Configuration and Project Management
- Environment Management

SPECIALIZED PROCESS MODELS :

• These models are typically utilised if a specialised or narrowly defined method of software engineering is selected as the strategy of choice.

Component-Based Development

1. 1. Research and analysis are conducted on the component-based products that are currently available on the market for the application domain in question.
2. Problems with component integration are taken into consideration.
3. A software architecture that can accommodate the components is built.
4. The architecture incorporates the components into its structure.
5. Extensive testing is carried out to validate the correct operation of the component.

The use of a component-based development methodology results in increased software reuse and reusability.

The Formal Methods Model

- Using formal techniques, you can describe, build, and validate a computer-based system by employing a stringent mathematical language. This is made possible by the use of formal methods.
- The creation of formal models now requires a significant amount of time and money due to their complexity.
- Extensive training is necessary since only a small percentage of software engineers have the appropriate experience to apply formal approaches.
- Customers who are not technically savvy will have a tough time understanding how to use the models as a communication channel.

Aspect-Oriented Software Development

- Aspectual requirements are used to define the concerns that span across multiple layers of the software architecture and have an effect on each.
- Aspect-oriented software development (AOSD) is a modern approach to software engineering that offers a systematic and methodical approach for defining, specifying, designing, and constructing aspects. It is alternatively referred to as aspect-oriented programming (AOP) and aspect-oriented component engineering (AOCE).
- The process will incorporate features from both evolutionary and concurrent process models.
- Aspect-oriented software development (AOSD) is an acronym that stands for "aspect-oriented software development."

THE UNIFIED PROCESS

It's an effort to incorporate many of the finest practises of agile software development while drawing on the strengths of traditional software process models.

The streamlined procedure highlights the significance of software architecture and directs the architect's attention to where it's needed most.

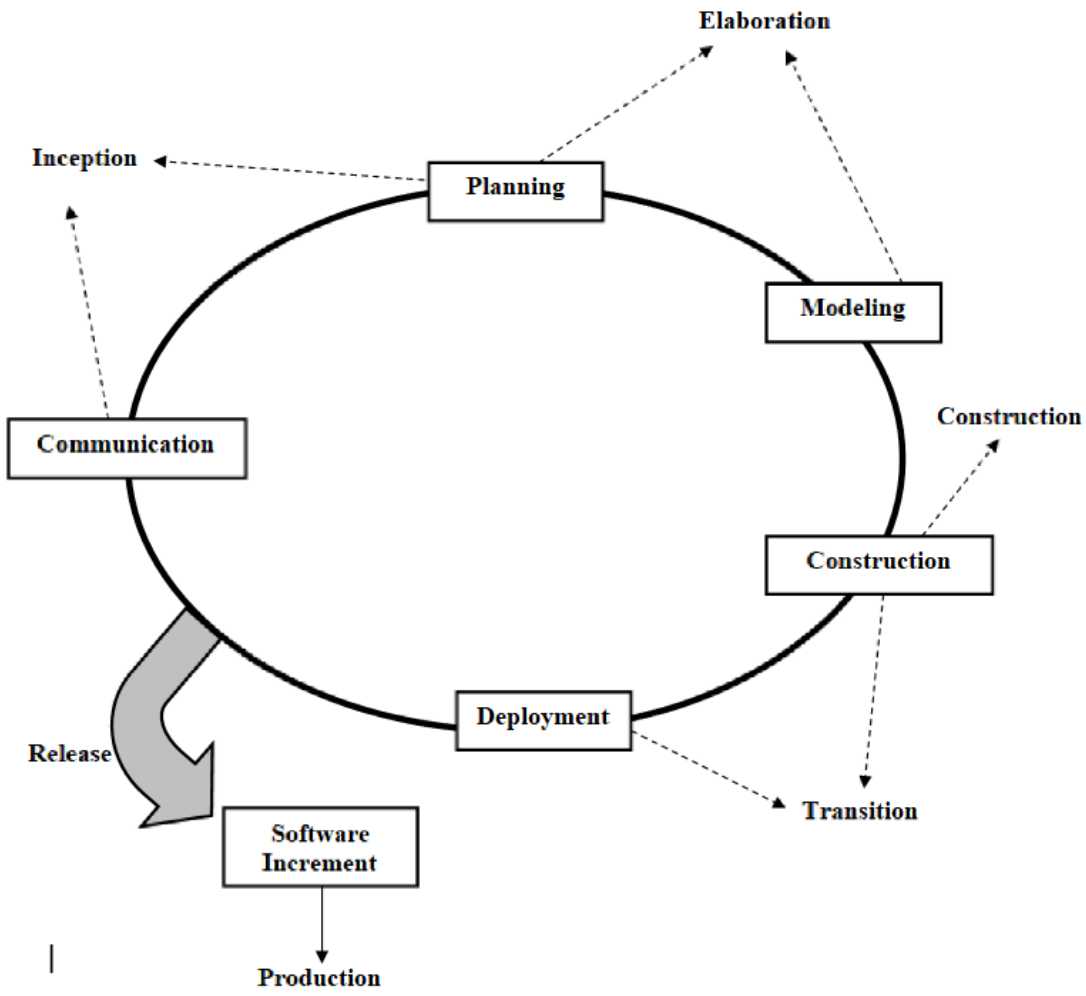
=> Appropriate objectives

=> Clarity

=> Adaptability

=> Reusability

* It suggests at a process flow that is iterative and gradual, which creates the appearance that evolution is taking place.



Phases of the Unified Process:

(1) Inception Phase:

1) In this stage, you'll be communicating with customers and making plans at the same time. One can determine the software's business needs by combining the two above;

2) The system's general layout is proposed; and

3) the project's iterative and incremental nature will be taken into account as the plan is produced.

A use-case is a generic term for a description of a set of steps taken by an actor (whether human or machine).

* Use-cases lay the groundwork for project planning and help define the scope of the project.

(2) Elaboration Phase:

It improves and extends the initial use cases created during the inception stage.

It adds five more perspectives to the software's architectural representation:

=> Use-case model

=> Analysis Model

=> Design Model

=> Implementation Model

=> Deployment Model

* The modification to the plan may be made at this time

(3) Construction Phase:

* The software building blocks that will make each use case usable by actual people are created (or purchased) here.

* The code for the software contains all of its necessary and required features and functions. Unit tests are developed and executed for individual components after their implementation.

(4) Transition Phase:

During this stage, the software team can create all of the documentation it will need to run well.

* In the Beta testing phase, the programme is made available to actual end users. Feedback on reported issues and necessary tweaks comes directly from customers.

Methods of Assembly

(5) Production Phase:

* During these phase the

The infrastructure of the running environment is supported, and the software's continued usage is monitored.

• Problem and enhancement reports are being reviewed as they are submitted.