JEPPIAAR INSTITUTE OF TECHNOLOGY

"Self-Belief | Self Discipline | Self Respect"

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING

LECTURE NOTES

CS8484-SOFTWARE ENGINEERING

(Regulation 2017)

Year/Semester: II/04/ CSE

2020 – 2021

Prepared by

Mrs.R.Dayana

Assistant Professor/CSE

UNIT- I

SOFTWARE PROCESS AND AGILE DEVELOPMENT

Introduction to Software Engineering, Software Process, Perspective and Specialized ProcessModels –Introduction to Agility – Agile process – Extreme programming – XP Process.

1.1 INTRODUCTION TO SOFTWARE ENGINEERING

The Evolving Role of Software:

- Software can be considered in a dual role. It is a **product** and, a vehicle for delivering a product.
- As a product, it delivers the computing potential in material form of computer hardware.

Example

A network of computers accessible by local hardware, whether it resides within a cellular phone or operates inside a mainframe computer.

i) As the vehicle, used to deliver the product. Software delivers the most important product of our time- **Information.**

ii) Software transforms personal data, it manages business information to enhance competiveness, it provides a gateway to worldwide information networks (e.g., Internet) and provides the means for acquiring information in all of its forms.

iii) Software acts as the basis for operating systems, networks, software tools and environments.

1.1.1 software

Software Characteristics

Software is a logical rather than a physical system element. Therefore, software has characteristics that are considerably different than those of hardware:

1. *Software is developed or engineered; it is not manufactured in the classical sense.*

- Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different.
- In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software.

2. Software doesn't "wear out."

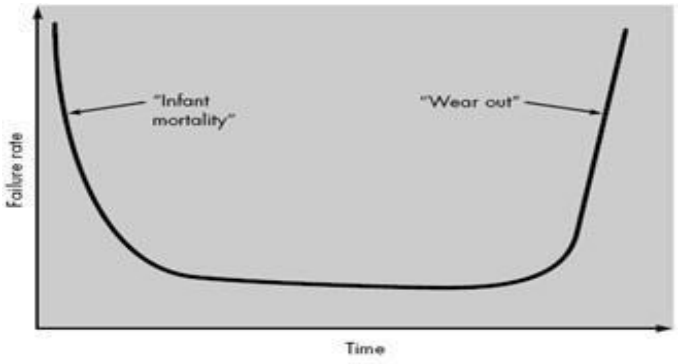


Figure 1.1 Failure curve for hardware

- Figure 1.1 depicts **failure rate as a function of time for hardware**.
- The relationship, often called the "bathtub curve", indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing

defects); defects are corrected and the failure rate drops to a steady-state level (ideally, quite low) for some period of time.

- As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies.
- Stated simply, the hardware begins to wear out.

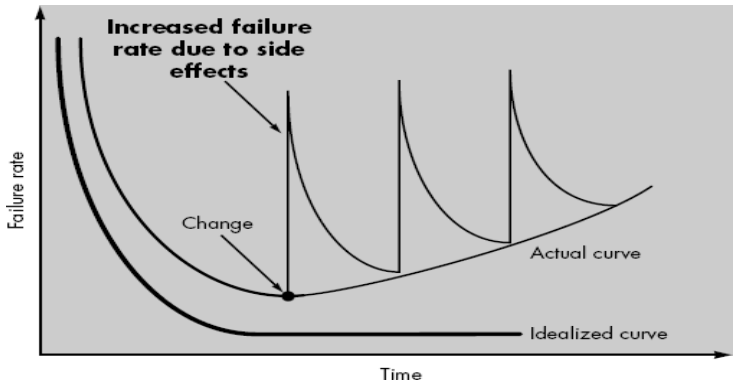


Figure 1.2 Idealized and actual failure curves for software

- The failure rate curve for software should take the form of the **"idealized curve**" shown in Fig 1.2.
- **Undiscovered defects will cause high failure rates** early in the life of a program.
- However, these are corrected (ideally, without introducing other errors) and the curve flattens as shown.
- The idealized curve is a gross oversimplification of actual failure models the implication is clear—software doesn't wear out.
- During its life, software will undergo change (maintenance).
- As changes are made, it is likely that some **new defects will be introduced, causing the failure rate curve to spike** as shown in Figure 1.2.
- Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again.
- Slowly, the minimum failure rate level begins to rise—the software is deteriorating due to change.

3. Although the industry is moving toward component-based assembly, most software Continues to be custom built.

- Software component should be designed and implemented so that it can be reused in many different programs.
- For example, today's graphical user interfaces are built using reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction mechanisms.
- The data structure and processing detail required to build the interface are contained with a library

of reusable components for interface construction.

1.1.2. Software Application Domains

The following categories of computer software present continuing challenges for software engineers.

a) System software:

- System software is a collection of programs written to service other programs.
- Example: compilers, editors, and file management utilities, operating system components, drivers, telecommunications processors, process largely indeterminate data.

b) Real-time software:

Elements of real-time software includes

- a data gathering component that collects and formats information from an external environment
- an analysis component that transforms information as required by the application
- a control/output component that responds to the external environment
- a monitoring component that coordinates all other components so that real-time response (typically ranging from 1 millisecond to 1 second) can be maintained.

c) Business software:

- Business information processing is the largest single software application area.
- Example: payroll, accounts receivable/payable, inventory.
- Applications in this area restructure existing data in a way that facilitates business operations or management decision making. In addition to conventional data processing application, business software applications also encompass interactive computing
- Example: point of-sale transaction processing.

d) Engineering and scientific software:

- This is the software using “number crunching" algorithms.
- Example: System simulation, computer-aided design.

e) Embedded software:

- Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets.
- Example: keypad control for a microwave oven
- It provides significant function and control capability
- Example: Digital functions in an automobile such as fuel control, dashboard displays, and braking systems.

f) Personal computer software:

- Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications, external network, and database access are only a few of hundreds of applications.

g) Web-based software:

- The Web pages retrieved by a browser are software that incorporates executable instructions (Ex: CGI, HTML, Perl, or Java), and data (EX: hypertext and a variety of visual and audio formats).

- Expert systems, also called knowledgebase systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing are representative of applications within this category.

SOFTWARE ENGINEERING

In order to build software that is ready to meet the challenges and it must recognize a few simple realities:

- It follows that a concerted effort should be made to understand the problem before a software solution is developed.
- It follows that design becomes a pivotal activity.
- It follows that software should exhibit high quality.
- It follows that software should be maintainable.

These simple realities lead to one conclusion: software in all of its forms and across all of its application domains should be engineered.

- Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.
- Software engineering encompasses a process, methods for managing and engineering software, and tools.

1.1.3 Software Engineering: A Layered Technology

Software engineering is a layered technology as shown in below Figure 1.3

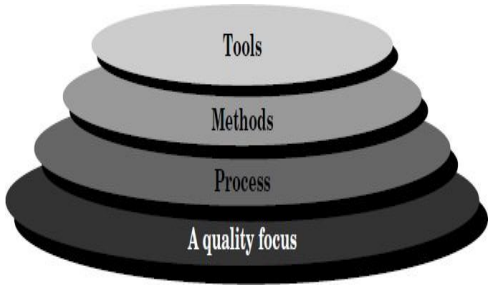


Figure1.3 Layered Technology

- The foundation for software engineering is the **process layer**.
- The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software.
- **Process** defines a framework that must be established for effective delivery of software engineering technology.

Software process:

- The software process forms the basis for management control of software projects and establishes the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured, and change is properly managed.

Software engineering methods:

- Software engineering methods provide the technical how-to's for building software.

- Software engineering methods rely on a set of basic principles that govern each area of the technology and include modelling activities and other descriptive techniques.

Software engineering tools:

- Software engineering tools provide automated or semi-automated support for the process and the methods.
- When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established.

1.1.4 The Software Process

- A **process** is a collection of activities, actions, and tasks that are performed when some work product is to be created.
- An **activity** strives to achieve a broad objective and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.
- An **action** (e.g., architectural design) encompasses a set of tasks that produce a major work product (e.g., an architectural design model).
- A **task** focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.
- ✓ A process framework establishes the foundation for a complete software engineering process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
- ✓ In addition, the process framework encompasses a set of **umbrella activities** that are applicable across the entire software process. A generic process framework for software engineering encompasses five activities:

The five generic process framework activities:

a) Communication:

- The intent is to understand stakeholders' objectives for the project and to gather requirements that help define software features and functions.

b) Planning:

- Software project plan—defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

c) Modelling:

- A software engineer does by creating models to better understand software requirements and the design that will achieve those requirements.

d) Construction:

- This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.

e) Deployment:

- Software engineering process framework activities are complemented by a number of umbrella activity.
- In general, umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk. Typical umbrella activities include:

i) **Software project tracking and control**—allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.
ii) **Risk management**—assesses risks that may affect the outcome of the project or the quality of the product.
iii) **Software quality assurance**—defines and conducts the activities required to ensure software quality.
iv) **Technical reviews**—access software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.
v) **Measurement**—defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs; can be used in conjunction with all other framework and umbrella activities.
vi) **Software configuration management**—manages the effects of change throughout the software process.
vii) **Reusability management**—defines criteria for work product reuse(including software components) and establishes mechanisms to achieve reusable components.
viii) **Work product preparation and production**—encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

1.1.5 Software Engineering Practice:

- A basic understanding of the generic concepts and principles that apply to framework activities
- The essence of problem solving, and consequently, the essence of software engineering practice:

1. Understand the problem (communication and analysis).
2. Plan a solution (modeling and software design).
3. Carry out the plan (code generation).
4. Examine the result for accuracy (testing and quality assurance).

1.1.6. Software Myths

1.Management myths.

A software manager often believes that myths will lessen the pressure

Myth: We already have a book that's full of standards and procedures for building Software, won't that provide my people with everything they need to know?

Reality: The book of standards may very well exist, but is it used? Are software Practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it streamlined to improve time to delivery while still maintaining focus on quality? In many cases, the answer to all of these questions is "no."

Myth: My people have state-of-the-art software development tools, after all, we buy them the newest computers.

Reality: It takes much more than the latest model mainframe, workstation, or PC to do high-quality software development. Computer-aided software engineering

(Sometimes called the Mongolian horde concept).

Reality: "Adding people to a late software project makes it later." As new people are added, People who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort. People can be added but only in a planned and well-coordinated manner.

Myth: If I decide to outsource3 the software project to a third party, I can just relax and let that firm build it.

Reality: If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

2. Customer myths. In many cases, the customer believes myths about software because Software managers and practitioners do little to correct misinformation. Myths lead to false expectations (by the customer) and ultimately, dissatisfaction with the developer.

Myth: A general statement of objectives is sufficient to begin writing programs—
We can fill in the details later.

Reality: A poor up-front definition is the major cause of failed software efforts. A formal and detailed description of the information domain, function, behavior, performance, interfaces, design constraints, and validation criteria is essential. These characteristics can be determined only after thorough communication between customer and developer.

Myth: Project requirements continually change, but change can be easily accommodated because software is flexible.

Reality: It is true that software requirements change, but the impact of change
Varies with the time at which it is introduced.

If serious attention is given to up-front definition, early requests for change can be accommodated easily. When changes are requested during software design, the cost impact grows rapidly. Change can cause upheaval that requires additional resources and major design modification, that is, additional cost. Changes in function, performance, interface, or other characteristics during implementation (code and test) have a severe impact on cost.

3. Practitioner's myths. Myths that are still believed by software practitioners have been fostered by 50 years of programming culture. During the early days of software, programming was viewed as an art form. Old ways and attitudes die hard.

Myth: Once we write the program and get it to work, our job is done.

Reality: Someone once said that "the sooner you begin 'writing code', the longer it'll take you to get done." Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Myth: Until I get the program "running" I have no way of assessing its quality.

Reality: One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the formal technical review. Software reviews are a "quality Filter" that have been found to be more effective than testing for finding certain classes of Software defects.

Myth: The only deliverable work product for a successful project is the working program.

Reality: A working program is only one part of a software configuration that includes

1.1.7 Software Engineering Paradigm:

- ✓ A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.
- ✓ Manufacturing software can be characterized by a series of steps ranging from concept exploration to final retirement; this series of steps is generally referred to as a software lifecycle.
- ✓ Steps or phases in a software lifecycle fall generally into these categories:
- Requirements
- Specification (analysis)
- Design
- Implementation
- Testing
- Integration
- Maintenance
- Retirement
- ✓ Software engineering employs a variety of methods, tools, and paradigms.
- ✓ Paradigms refer to particular approaches or philosophies for designing, building and maintaining software. Different paradigms each have their own advantages and disadvantages.
- ✓ A method (also referred to as a technique) is heavily depended on a selected paradigm and may be seen as a procedure for producing some result. Methods generally involve some formal notation and process(es).
- ✓ Tools are automated systems implementing a particular method.
- ✓ Thus, the following phases are heavily affected by selected software paradigms
- Design
- Implementation
- Integration
- Maintenance

The software development cycle involves the activities in the production of a software system. Generally the software development cycle can be divided into the following phases:

a) Requirements analysis and specification

- **Design**
- ✓ Preliminary design
- ✓ Detailed design
- **Implementation**
  - Component Implementation
  - Component Integration
  - System Documenting
- **Testing**
- Unit testing
- Integration testing
- System testing

- ✓ Change requirements and software upgrading

b) Verification - Validation

- Verification: "Are we building the product right"
- The software should conform to its specification
- Validation: "Are we building the right product"
- The software should do what the user really requires

1.2 SOFTWARE PROCESS A Generic Process Model

- A process was defined as a collection of work activities, actions, and tasks that are performed when some work product is to be created.
- Each of these activities, actions, and tasks reside within a framework or model that defines their relationship with the process and with one another.
- ✓ The software process is represented schematically in Figure 1.4. Referring to the figure 1.4, each framework activity is populated by a set of software engineering actions.

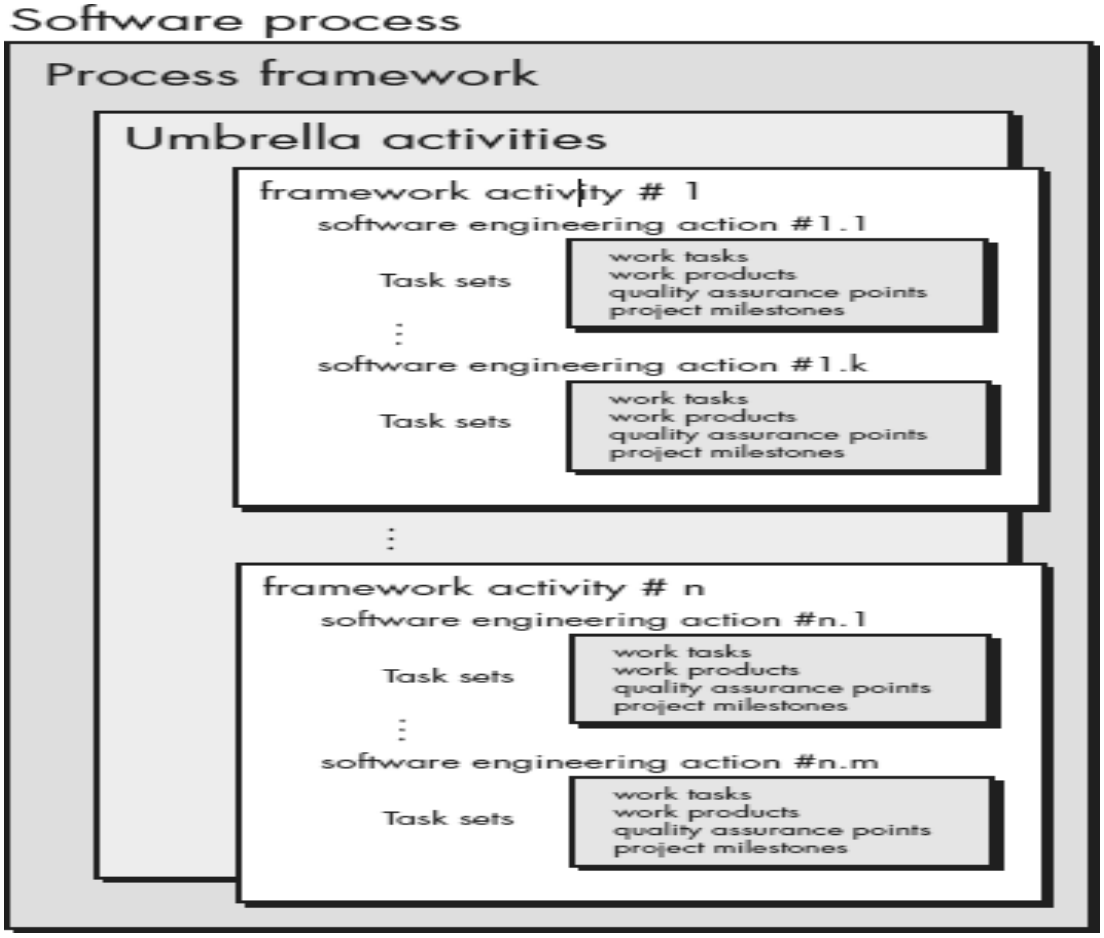


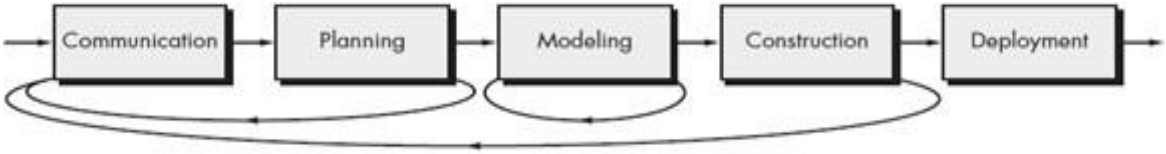
Figure 1.4 Software Process Framework

Process flow—describes how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to sequence and time.

(a) A **linear process flow** executes each of the five framework activities in sequence, beginning with communication and culminating with deployment.

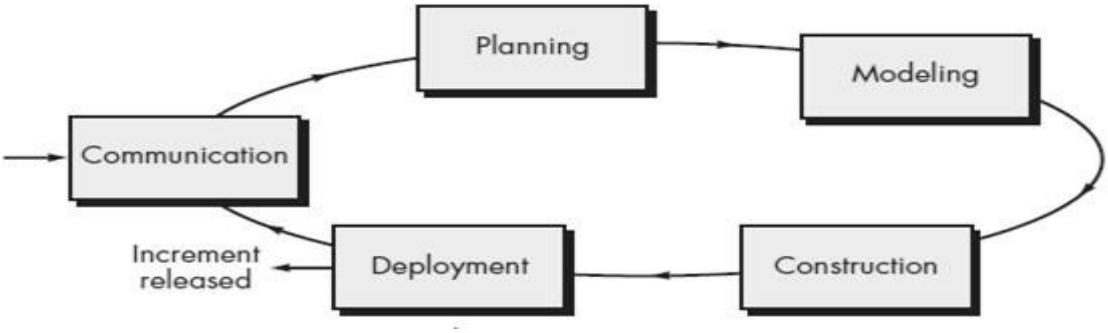
a) Linear Process Flow

(b) An **iterative process flow** repeats one or more of the activities before proceeding to the next.



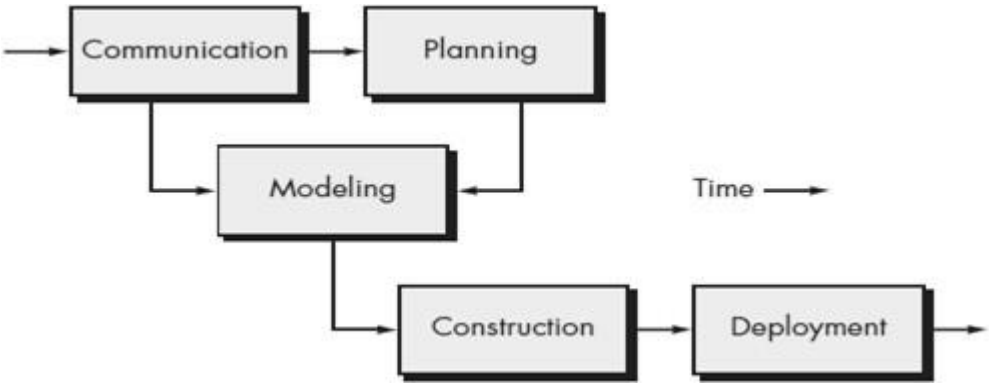
b) Iterative process flow

(c) An **evolutionary process flow** executes the activities in a "circular"manner. Each circuit through the five activities leads to a more complete version of the software.



c) Evolutionary process flow

(d) A **parallel process flow** executes one or more activities in parallel with other activities (e.g., modelling for one aspect of the software might be executed in parallel with construction of another aspect of the software).



d) Parallel process flow

Identifying a Task Set:

- A task set defines the actual work to be done to accomplish the objectives of a software engineering action.

- For example, elicitation (more commonly called "requirements gathering") is an important software engineering action that occurs during the communication activity.
- The goal of requirements gathering is to understand what various stakeholders want from the software that is to be built.

Process Patterns:

- A process pattern describes a process-related problem that is encountered during software engineering work, identifies the environment in which the problem has been encountered, and suggests one or more proven solutions to the problem.
- Patterns can be defined at any level of abstraction. In some cases, a pattern might be used to describe a problem (and solution) associated with a complete process model (e.g., prototyping).
- In other situations, patterns can be used to describe a problem (and solution) associated with a framework activity (e.g., planning) or an action within a framework activity (e.g., project estimating).
- Ambler [Amb98] has proposed a template for describing a process pattern:

Pattern Name: The pattern is given a meaningful name describing it within the context of the software process (e.g., Technical Reviews).
Forces: The environment in which the pattern is encountered and the issues that make the problem visible and may affect its solution.
Type: The pattern type is specified. Ambler [Amb98] suggests three types:
1. Stage pattern—defines a problem associated with a framework activity for the process. Since a framework activity encompasses multiple actions and work tasks, a stage pattern incorporates multiple task patterns (see the following)that are relevant to the stage (framework activity).

- ✓ An example of a stage pattern might be **Establishing Communication.** This pattern would incorporate the task pattern **Requirements Gathering** and others.

2. Task pattern—defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice (e.g., **Requirements Gathering** is a task pattern).
3. Phase pattern—define the sequence of framework activities that occurs within the process, even when the overall flow of activities is iterative in nature.

- ✓ An example of a phase pattern might be **Spiral Model or Prototyping.**

Initial context: Describes the conditions under which the pattern applies.
Prior to the initiation of the pattern:
(1) What organizational or team-related activities have already occurred?
(2) What is the entry state for the process?
(3) What software engineering information or project information already exists?
For example, the Planning pattern (a stage pattern) requires that
(1) customers and software engineers have established a collaborative communication ;

(2) successful completion of a number of task patterns [specified] for the Communication pattern has occurred; and
(3) the project scope, basic business requirements, and project constraints are known.
Problem: The specific problem to be solved by the pattern.
Solution:

- Describes how to implement the pattern successfully.
- This section describes how the initial state of the process (that exists before the pattern is implemented) is modified as a consequence of the initiation of the pattern.
- It also describes how software engineering information or project information that is available before the initiation of the pattern is transformed as a consequence of the successful execution of the pattern.

Resulting Context: Describes the conditions that will result once the pattern has been successfully implemented. Upon completion of the pattern:
(1) What organizational or team-related activities must have occurred?
(2) What is the exit state for the process?
(3) What software engineering information or project information has been developed?
Related Patterns:

i) Provide a list of all process patterns that are directly related to this one. This may be represented as a hierarchy or in some other diagrammatic form.
ii) For example, the stage pattern Communication encompasses the task patterns:

- ✓ **ProjectTeam,**
- ✓ **CollaborativeGuidelines,**
- ✓ **ScopeIsolation,**
- ✓ **RequirementsGathering,**
- ✓ **ConstraintDescription, and**
- ✓ **ScenarioCreation.**

Known Uses and Examples:

- Indicate the specific instances in which the pattern are applicable.
- For example, Communication is mandatory at the beginning of every software project, is recommended throughout the software project, and is mandatory once the deployment activity is under way.

1.3 PRESCRIPTIVE PROCESS MODELS

- ✓ Prescriptive process models were originally proposed to bring order to the chaos of software development.
- ✓ Prescriptive process models define a prescribed set of process elements anda predictable process work flow.

- Prescriptive Process Models
- The Waterfall Model
- Incremental Process Models
- Evolutionary Process Models

1.3.1 The Waterfall Model

The waterfall model, sometimes called the classic life cycle, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modelling, construction, and deployment, culminating in ongoing support of the completed software.

- A variation in the representation of the waterfall model is called the **V-model.**
- Represented in Figure 1.5, the V-model [Buc99] depicts the relationship of quality assurance actions to the actions associated with communication, modelling, and early construction activities.

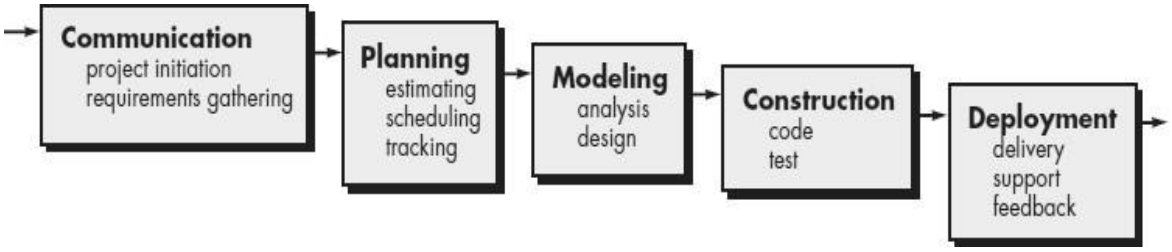
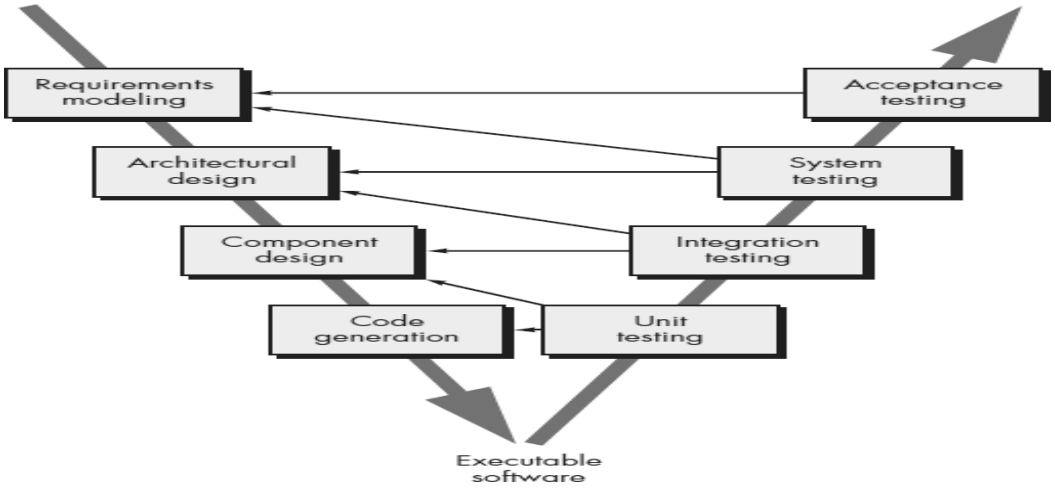


FIGURE 1.5The waterfall model

- As a software team moves down the left side of the V, basic problem requirements are refined into progressively more detailed and technical representations of the problem and its solution.
- Once code has been generated, the team moves up the right side of the V, essentially performing a series of tests (quality assurance actions) that validate each of the models created as the team moved down the left side.
- In reality, there is no fundamental difference between the classic life cycle and the V-model.
- The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work.

Benefits of waterfall model:

- The waterfall model is simple to implement
- For implementation of small systems waterfall model is useful.



Drawbacks of waterfall model:

There are some problems that are encountered if we apply the waterfall model and those are:

- It is difficult to follow the sequential flow in software development process. If some changes are made at some phases then it may cause some confusion.
- The requirement analysis is done initially and sometimes it is not possible to state all the requirements explicitly in the beginning. This causes difficulty in the project.
- The customer can see the working model of the project only at the end. After reviewing of the working model; if the customer gets dissatisfied then it causes serious problems.
- Linear nature of waterfall model induces **blocking states**, because certain tasks may be dependent on some previous tasks. Hence it is necessary to accomplish all the dependant tasks first. It may cause long waiting time.

1.3.2 Incremental Process Models:

- The incremental model delivers a series of releases, called increments that provide progressively more functionality for the customer as each increment is delivered.
- The incremental model applies linear sequences in a staggered fashion as calendar time progresses.
- Each linear sequence produces deliverable "increments" of the software [McD93] in a manner that is similar to the increments produced by an evolutionary process flow.
- The first increment is called core product. In this release the basic requirements are implemented and then in subsequent increments new requirements are added.
- The core product is used by the customer (or undergoes detailed evaluation).
- As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.

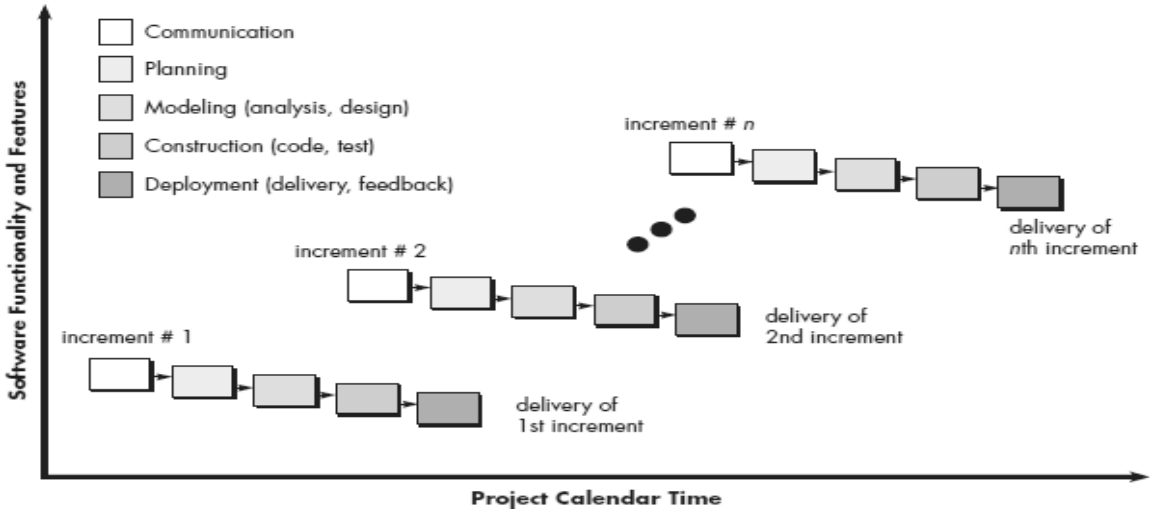


Figure 1.7 Incremental Process Model

i) In the second increment, more sophisticated document producing and processing facilities, file management functionalities are given.

Incremental process Model advantages

1. Produces working software early during the lifecycle.
2. More flexible as scope and requirement changes can be implemented at low cost.
3. Testing and debugging is easier, as the iterations are small.
4. Low risks factors as the risks can be identified and resolved during each iteration.

Incremental process Model disadvantages

1. This model has phases that are very rigid and do not overlap.
2. Not all the requirements are gathered before starting the development; this could lead to problems related to system architecture at later iterations.

1.3.2.1 The RAD Model

- Rapid Application Development is a linear sequential software development process model that emphasizes an extremely short development cycle.
- Rapid application achieved by using a component based construction approach.
- If requirements are well understood and project scope is constrained the RAD process enables a development team to create a ―fully functional system.

RAD phases:

- Business modeling
- Data modeling
- Process modeling
- Application generation
- Testing and turnover

Business modelling:

- What information drives the business process?
- What information is generated?
- Who generates it?
- Where does the information go?
- Who processes it?

Data modelling:

- The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business.
- The characteristics (called attributes) of each object are identified and the relationships between these objects are defined.

Process modelling:

- The data modelling phase are transformed to achieve the information flow necessary to implement a business function.
- Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

Application generation:

- RAD assumes the use of 4 generation techniques.

- Possible) or created reusable components (when necessary).

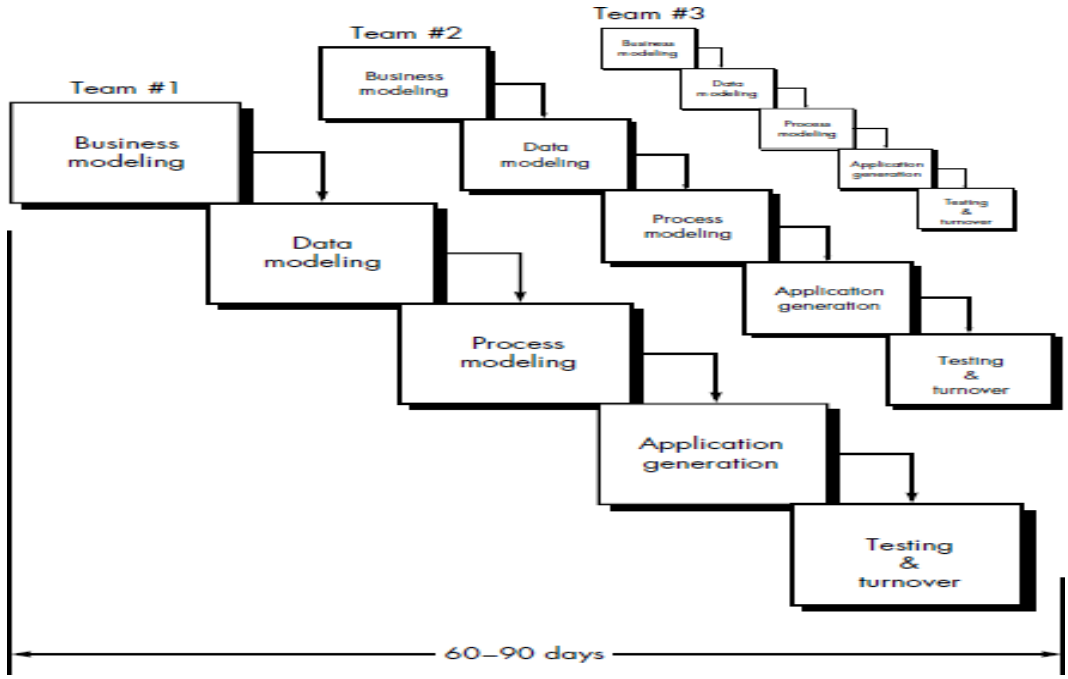


Figure1.8: RAD Process model

Testing and Turnover:

- Since the RAD process emphasizes reuse, many of the program components have already been testing.
- This reduces over all testing time.
- However, new components must be tested and all interfaces must be fully exercised.

Advantages &Disadvantages of RAD:

Advantages

- Extremely short **development** time**.**
- Uses component-based construction and emphasizes reuse and code generation

Disadvantages

- Large human resource requirements (to create all of the teams).
- Requires strong commitment between developers and customers for "rapid-fire" activities.
- High performance requirements can't be met (requires tuning the components).

1.3.3 Evolutionary Process Models

- Evolutionary process models produce an increasingly more complete version of the software with each iteration.

1.3.3.1 The Prototyping Model:

- The prototyping paradigm (Figure 1.9) begins with communication.Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known,

- A **quick design** focuses on a representation of those aspects of the software that will be visible to the customer/user (e.g. Input approaches and output formats). The quick design leads to the construction of a prototype.
- The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.
- Ideally, the prototype serves as a mechanism for identifying software requirements.
- If a working prototype is built, the developer attempts to use existing program fragments or applies tools (e.g., report generators, window managers) that enable working programs to be generated quickly.

Advantages:

- Requirements can be set earlier and more reliably.
- Customer sees results very quickly**.**
- Customer is educated in what is possible helping to refine requirements**.**
- Requirements can be communicated more clearly and completely.
- Between developers and clients Requirements and design options can be investigated quickly and cheaply.

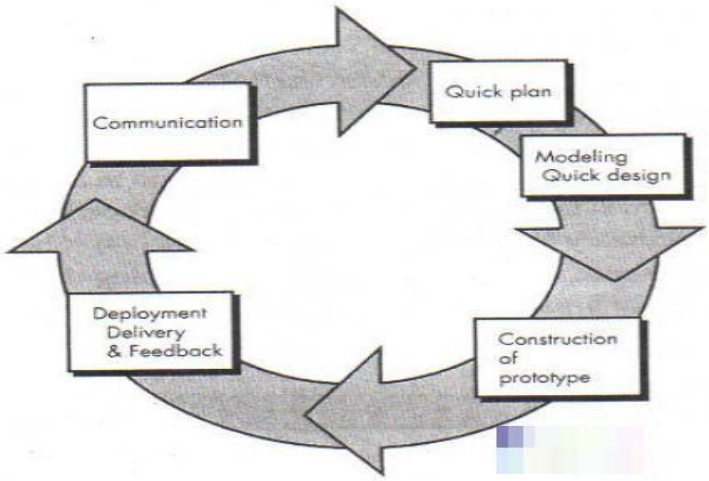


Figure 1.9: Prototyping Model

Drawbacks of prototyping:

- In the first version itself, customer often wants "few fixes" rather than rebuilding of the system whereas rebuilding of new system maintains high level of quality.
- The first version may have some compromises.
- Sometimes developer may make implementation compromises to get prototype working quickly. Later on developer may become comfortable with compromises and forget why they are inappropriate.

1.3.3.2 Spiral Model:

- The spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model.

- The spiral development model is a risk-driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems.
- It has two main distinguishing features.
- One is a cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk.
- The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.
- Using the spiral model, software is developed in a series of incremental releases.
- A spiral model is divided into a number of framework activities, also called **task regions.**
- Typically, there are between three and six task regions. Figure1.10depictsspiral model that contains six task regions:

  Customer communication—tasks required to establish effective communication between developer and customer.

  Planning—tasks required to define resources, timelines, and other project related information.

  Risk analysis—tasks required to assess both technical and management risks.

  Engineering—tasks required to build one or more representations of the application.

  Construction and release—tasks required to construct, test, install, and provide user support (e.g., documentation and training).

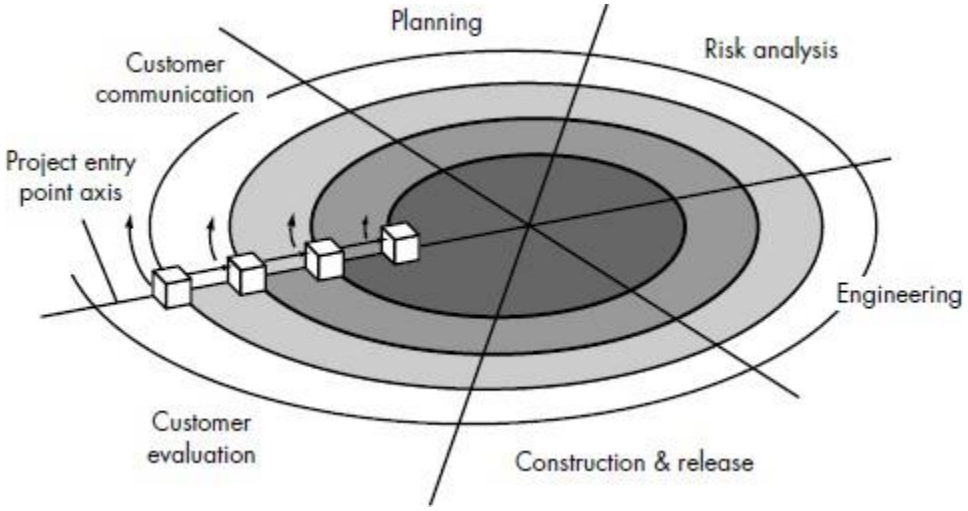


Figure 1.10: Spiral model

Customer evaluation—tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage

- As this evolutionary process begins, the software engineering team moves aroundthe spiral in a clockwise direction, beginning at the centre.
- **Anchor point milestones—**a combination of work products and conditions that are attained along the path of the spiral
- The first circuit around the spiral might result in the development of a product specification;

- Each cube placed along the axis can be used to represent the starting point for different types of projects A "concept development project" starts at the core of the spiral and will continue until concept development is complete.
- If the concept is to be developed into an actual product, the process proceeds through the next cube (new product development project entry point) and a "new development project" is initiated. The new product will evolve through a number of iterations around the spiral, following the path that bounds the core region.
- Spiral model is realistic approach to development of large-scale systems and software. Because customer and developer better understand the problem statement at each evolutionary level. Also risks can be identified or rectified at each such level.

Spiral Model Advantages:

- Requirement changes can b made at every stage.
- Risks can be identified and rectified before they get problematic.

Spiral Model disadvantages:

- It is based on **customer communication.** If the communication is not proper then the software product that gets developed will not be up to the mark.
- It demands considerable **risk assessment.** If the risk assessment is done properly then only the successful product can be obtained.

1.3.4 Concurrent Models

- The concurrent development modelis also called as **concurrent engineering**.
- It allows a software team to represent iterative and concurrent elements of any of the process models.
- In this model, the framework activities or software development tasks are represented as **states.**
- For example, the **modeling or designing** phase of software development can be in one of the states like under development, waiting for modification, under revision or under review and so on.
- All software engineering activities exist concurrently but reside in different states.
- These states make transitions. That is during **modeling,** the transition from **under development** state to **waiting for modification** state occurs.
- Customer indicates that changes in requirements must be made, the modeling activity moves from the under development state into the awaiting changes state.
- This model basically defines the **series of events** due to which the transition from one state to another state occurs. This is called **triggering.** These series of events occur for every software development activity, action or task.

Advantages:

- All types of software development can be done using concurrent development model.
- This model provides accurate picture of current state of project.
- Each activity or task can be carried out concurrently. Hence this model is an efficient process model.

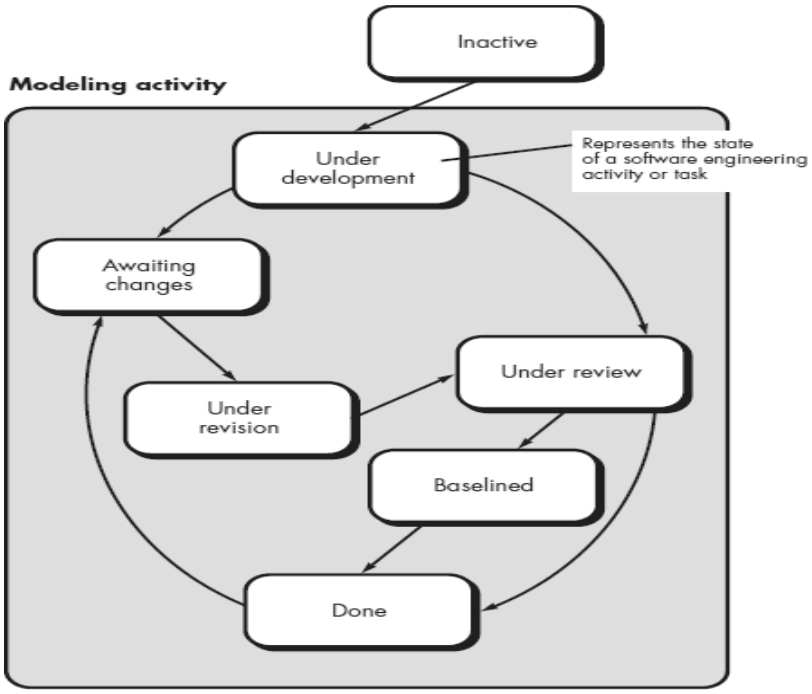


Figure 1.11 One element of the concurrent process model

1.4 SPECIALIZED PROCESS MODELS:

The specialized models are used when only collections of specialized technique or methods are expected for developing the specific software.

Various types of specialized models are-

1. Component based development
2. Formal methods model
3. Aspect oriented software development

Component based development:

- The commercial off-the-shelves components that are developed by the vendors are used during the software built.
- These components have specialized targeted functionalities and well defined interfaces. Hence it is easy to integrate these components into the existing software.
- The component-based development model incorporates many of the characteristics of the spiral model. It is evolutionary in nature.
- Modeling and construction activities begin with the identification of candidate components. These components can be designed as either conventional software modules or object-oriented classes or packages of classes

Following steps are applied for component based development

1. Available component-based products are researched and evaluated for theapplication domain in question.
2. Component integration issues are considered.
3. A software architecture is designed to accommodate the components.

- ✓ **Software reusability** is the major advantage of component based development.
- ✓ The **reusability** reduces the development cycle time and overall cost**.**

Formal methods model:

- The formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software.
- Formal methods enable you to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation. A variation on this approach, called **cleanroom software engineering.**
- The **advantage** of using formal methods model is that it overcomes many problems that we encounter in traditional software process models.
- **Ambiguity, Incompletenessand Inconsistency** are those problems that can be overcome if we use formal methods model.

.

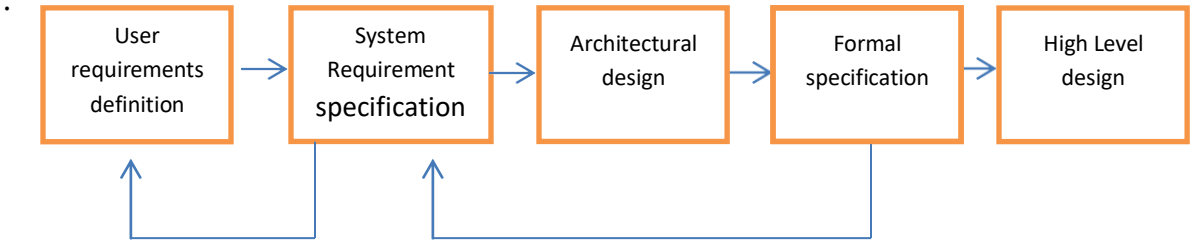


Figure 1.12 Steps involved in Formal Method Model

- The formal methods model offers detect-free software. However there are some **drawbacks of this model** which resists it from getting used widely.
- These drawbacks are
- ✓ The development of formal models is currently quite time consuming and expensive.
- ✓ Because few software developers have the necessary background to apply formal methods, extensive training is required.
- ✓ It is difficult to use the models as a communication mechanism for technically unsophisticated customers.

Aspect oriented software development:

- AOSD defines "aspects" that express customer concerns that cut across multiple system functions, features, and information.
- In traditional software development process the system is decomposed into multiple units of primary functionality.
- When concerns cut across multiple system functions, features, and information, they are often referred to as **crosscutting concerns.**
- **Aspectual requirements** define those crosscutting concerns that have an impact across the software architecture.
- **Aspect-oriented software development (AOSD),** often referred to as **aspect-oriented programming (AOP),** is a relatively new software engineering paradigm that provides a process and methodological approach for defining, specifying, designing, and constructing aspects—"mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concern".

1.5 INTRODUCTION TO AGILITY:

1.5.1 What is Agility?

- Agility is the ability to **respond quickly to changing needs.** It encourages team structures and attitudes that make **effective communication among all stakeholders.**
- It emphasizes **rapid delivery of operational software** and de-emphasizes the importance of intermediate work products.
- It adopts the **customer as a part of the development team**.
- It helps in **organizing a team so that it is in control of the work performed.**

Yielding

- Agility results in **rapid, incremental delivery of software.**

1.5.2 Agility and the Cost of Change:

- The cost of change in software development increases nonlinearly as a project progresses (Figure 1.13, solid black curve).
- It is relatively easy to accommodate a change when software team gathered its requirements.
- The costs of doing this work are minimal, and the time required will not affect the outcome of the project.
- Cost varies quickly, and the cost and time required to ensure that the change is made without any side effects is nontrivial.
- **An agile process reduces the cost of change because software is released in increments and changes can be better controlled within an increment.**
- Agile process "flattens" the cost of change curve (Figure 1.11, shaded, solid curve), allowing a software team to accommodate changes late in a software project without dramatic cost and time impact.
- When incremental delivery is coupled with other agile practices such as continuous unit testing and pair programming, the cost of making a change is attenuated.

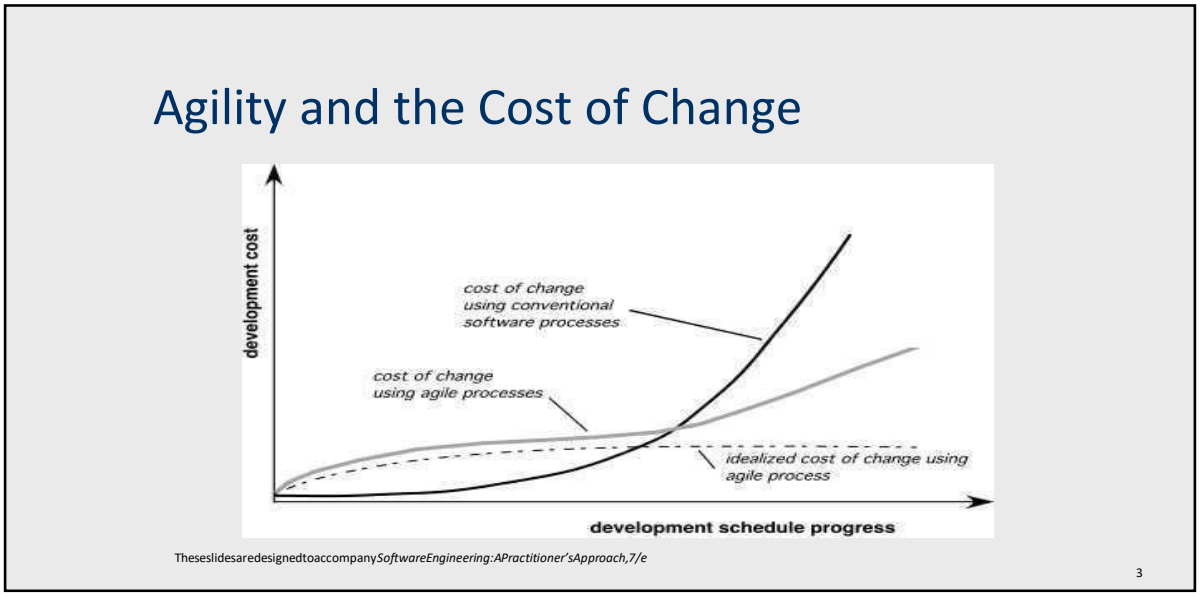


Figure 1.13Change costs as a function of time in development

1.6 AN AGILE PROCESS

An Agile Process is characterized in a manner that addresses a number of key assumptions about the majority of software project:

1. It is difficult to predict which software requirements will persist and which will change.
2. It is difficult to predict how customer priorities will change.
3. It is difficult to predict how much design is necessary before construction.
4. Analysis, design, construction, and testing are not as predictable.

1.6.1 Agility Principles:

1. To **satisfy the customer through early and continuous delivery of software**.
2. **Welcome changing requirements**, even late in development.
3. **Deliver working software frequently**, from a couple of weeks to a couple of months.
4. **'Customers and developers must work together daily** throughout the project.
5. **Build projects around motivated individuals**.
6. **Emphasis on face-to-face communication.**
7. **Working software is the primary measure of progress**.
8. Agile processes promote **sustainable development**.
9. Continuous attention to **technical excellence and good design enhances agility**.
10. **Simplicity**--the art of maximizing the amount of work not done--**is essential**.
11. **Self-organizing teams produce the best architectures/requirements/design**.
12. **The team reflects on how to become more effective** at regular intervals.

1.6.2 Human Factors:

- Agile development focuses on **the talents and skills of individuals, molding the process to specific people and teams.**
- ***The process molds to the needs of the people and team***, not the other wayaround.
- A number of key traits must exist among the people on an agile team and the team itself:
  - ✓ Competence.
  - ✓ Commonfocus.
  - ✓ Collaboration.
  - ✓ Decision-makingability.
  - ✓ Fuzzy problem-solvingability.
  - ✓ Mutual trust andrespect.
  - ✓ Self-organization**.**

1.7 EXTREME PROGRAMMING (XP):

- The best-known and a very influential agile method, Extreme Programming (XP) takes an 'extreme' approach to iterative development.
  - ✓ New versions may be built several times per day;
  - ✓ Increments are delivered to customers every 2 weeks;
  - ✓ All tests must be run for every build and the build is only accepted if tests run successfully.

This is how XP supports **agile principles**:

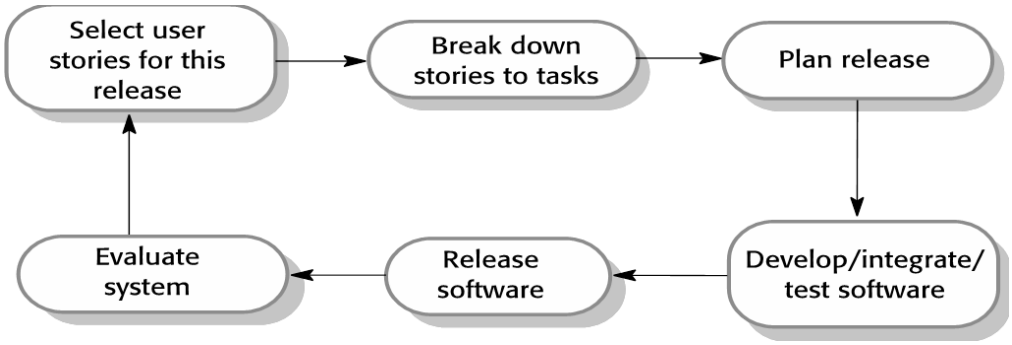


Figure 1.14The extreme programming release cycle

- People not process through **pair programming, collective ownership** and a process that avoids long working hours.
- Change supported through **regular system releases**.
- Maintaining simplicity through **constant refactoring of code.**

1.7.1 XP values:

- ✓ XP is comprised of five values such as:

i. Communication
ii. Simplicity
iii. Feedback
iv. Courage
v. Respect.

- ✓ Each of these values is used as a driver for specific XP activities, actions, and task.
- ✓ In order to achieve effective **communication** between **software engineers and other stakeholders**, XP emphasizes close, yet informal(verbal) collaboration between customers and developers, the establishment of effective metaphors for communicating important concepts, continuous feedback, and the avoidance of voluminous documentation as a communication medium.
- ✓ To consider **simplicity**, XP restricts developers to design only for immediate needs, rather than future needs.
- ✓ **Feedback** is derived from three sources: the **software, the customer and other team members**.
- ✓ By designing and implementing an effective testing strategy, the software provides the agile team with feedback.
- ✓ The team develops a **unit test** for each class being developed, to exercise each operation according to its specified functionality.
- ✓ The **user stories or use cases** are implemented by the increments being used as a basis for acceptance tests. The degree to which software implements the **output, function, and behavior of the test case** is a form of feedback.
- ✓ An agile XP team must have the courage (discipline) to design for today, recognizing that future requirements may change dramatically, thereby demanding substantial rework of the design and implemented code.
- ✓ For example, there is often significant pressure to design for future requirements.

1.8 The XP Process:

- Extreme programming uses an **object-oriented approach** for software development.
- There are four framework activities involved in XP Process are shown in Figure 1.15.

1. Planning
2. Designing
3. Coding
4. Testing

1. Planning:

- Begins with the creation of a set of stories (also called user stories).
- Each story is written by the customer and is placed on an index card.
- The customer assigns a value (i.e. a priority) to the story.
- Agile team assesses each story and assigns a cost.
- Stories are grouped to for a deliverable increment.

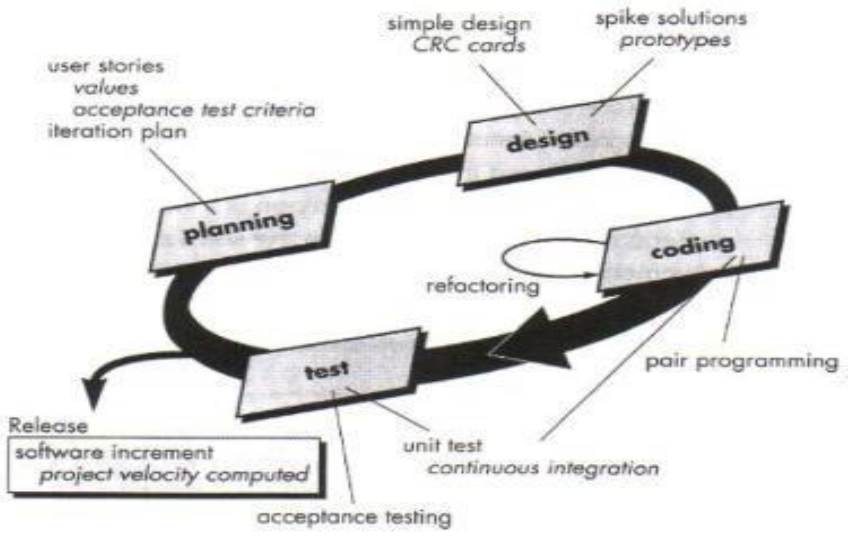


Figure 1.15 The Extreme Programming Process

- A commitment is made on delivery date.
- After the first increment "project velocity" is used to help define subsequent delivery dates for other increments.

2. Design:

- Follows the keep it simple principle.
- Encourage the use of CRC (class-responsibility-collaborator) cards.
- For difficult design problems, suggests the creation of "spike solutions"—a design prototype.
- Encourages "refactoring"—an iterative refinement of the internal program design
- Design occurs both before and after coding commences.

3. Coding:

- Recommends the construction of a series of unit tests for each of the stories before coding

- Encourages "pair programming"
  - Developers work in pairs, checking each other's work and providing the support to always do a good job.
  - Mechanism for real-time problem solving and real-time quality assurance.
  - Keeps the developers focused on the problem at hand.
- Needs continuous integration with other portions (stories) of the s/w, which provides a "smoke testing" environment.

4. Testing:

- The creation of unit test before coding is the key element of the XP approach.
- The unit tests that are created should be implemented using a framework that enables them to be automated.
- This encourages regression testing strategy whenever code is modified.
- Individual unit tests are organize into a "Universal Testing Suit", integration and validation testing of the system can occur on daily basis. This provides the XP team with a continual indication of progress and also can raise warning flags early if things are going away.
- XP acceptance test, also called customer test, are specified by the customer and focus on the overall system feature and functionality that are visible and reviewable by the customer.

1.8.1 Industrial XP:

i) IXP is an organic evolution of XP.
ii) It is imbued with XP's minimalist, customer –centric, test-driven spirit.IXP differs most from the original XP in its greater inclusion of management, its expanded role for customers, and its upgraded technical practices.
iii) IXP incorporates six new practices that are designed to help ensure that an XP project works successfully for significant projects within a large organization.

Readiness assessment: The organization should conduct a readiness assessment prior to the initiation of an IXP project. The assessment ascertains whether

i) an appropriate development environment exists
ii) the team will be populated by the proper set of stakeholders.
iii) the organization has a distinct quality program and supports continuous improvement.
iv) the organizational culture will support the new values of an agile team, and
v) the broader project community will be populated appropriately.

Project community:

i) People on the team must be well-trained, adaptable and skilled, and have the proper temperament to contribute to a self-organizing team.
ii) When XP is to be applied for a significant project in a large organization, the concept of the "team" should morph into that of a community.
iii) A community may have a technologist and customers who are central to the success of a project as well as many other stakeholders may play important roles on the project.

Project chartering:

i) The IXP team assess the project itself to determine whether the project exists and whether the project will further the overall goals and objectives of the organization.
ii) It also determines how it complements, extends, or replaces existing systems or process.

i) Test driven management establishes a series of measurable "destinations" and then defines mechanisms for determining whether or not these destinations have been reached. **Retrospectives:**

i) An IXP team conducts a technical review after software increment is delivered called retrospective.

ii) This review examines "issues,events,and lessons-learned" across a software increment and/or the entire software release.

iii) The intent is to improve the IXP process.

Continuous learning:

i) Learning is a vital product of continuous process improvement, members of the XP team are encouraged to learn new methods and techniques that can lead to a higher quality product.

ii) In addition to these six new practices, IXP modifies a number of existing XP practices.

✓ **Story-driven development (SDD)** insists that stories for acceptance tests be written before a single line of code is developed.

✓ **Domain-driven design (DDD):**

i) It is an improvement on the "system metaphor" concept used in XP.

ii) It suggests the creation of a domain model that accurately represents how domain experts think about their subject.

 Pairing extends the XP pair-programming concept to include managers and their stakeholders. The intent is to improve knowledge sharing among XP team members who may not be directly involved in technical development.

 Iterative usability discourages front-loaded interface design in favor of usability design that evolves as software increments are delivered and users' interaction with the software.

1.8.2 The XP Debate:

 Extreme Programming has done heated debate for both new process models and methods.

 This examines the efficacy of XP, but Stephens and Rosenberg argue that many XP practices are worthwhile, but others have been overhyped, and a few are problematic.

 The authors suggest that the codependent natures of XP practices are both its strength and its weakness.

 Because many organizations adopt only a subset of XP practices, they weaken the efficacy of the entire process.

 Proponents counter that XP is continuously evolving and that many of the issues raised by critics have been addressed as XP practice matures.

 Among the issues that continue to trouble some critics of XP are:

 ***Requirements volatility*.**

✓ Because the customer is an active member of the XP team, changes to requirements are requested informally.

✓ As a consequence, the scope of the project can change and earlier work may have to be modified to accommodate current needs**.**

 Conflicting customer needs.

✓ Many projects have multiple customers, each withhisown set of needs.Requirements are expressed informally.

- ✓ Critics argue that amore formal model or specification is often needed to ensure that omissions,inconsistencies, and errors are uncovered before the system is built.

 Lack of formal design.

- ✓ XP deemphasizes the need for architectural design andin many instances, suggests that design of all kinds should be relativelyinformal.
- ✓ Critics argue that when complex systems are built, design must beemphasized to ensure that the overall structure of the software will exhibit quality and maintainability.
- ✓ XP proponents suggest that the incremental nature of the XP process limits complexity (simplicity is a core value) and therefore reduces the need for extensive design.

1.8.3 Other Agile Process Models

1. Adaptive Software Development (ASD)
2. Dynamic Systems Development Method (DSDM)
3. Scrum
4. Crystal
5. Feature Driven Development (FDD)
6. Agile Modeling (AM)
7. Lean Software Development (LSD)
8. Agile Unified Process (AUP)

1.8.3.1 Adaptive Software Development (ASD)

- Adaptive Software Development (ASD) is a technique for building complex software and ASD incorporates three phases Speculation, Collaboration, and Learning systems
- ASD focus on human collaboration and team self-organization.

Speculation:

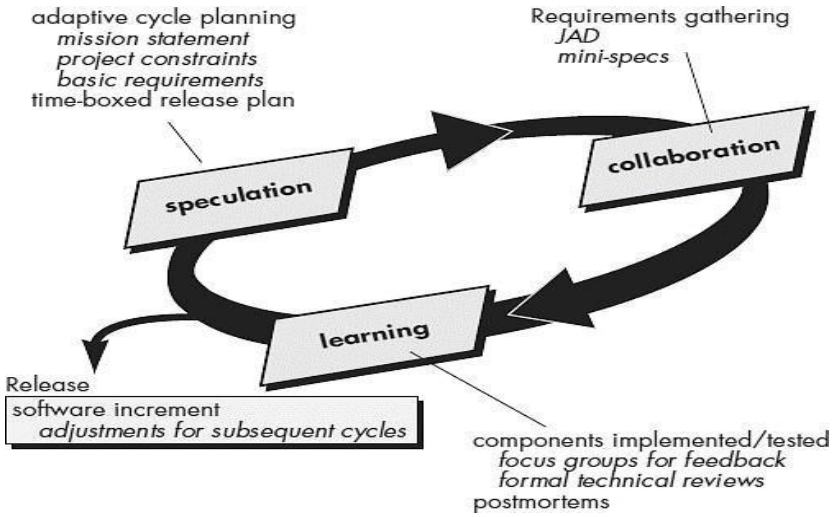


Figure 1.16Adaptive Software Development

- "Speculate" refers to the planning paradox—outcomes are unpredictable, therefore, endless suppositions on a product's look and feel are not likely to lead to any business value.
- The big idea behind speculate is when we plan a product to its smallest detail as in a requirements

- In the ASD mindset, planning is to speculation as intention is to need.

Collaboration:

- Collaboration represents a balance between managing the doing and creating and maintaining the collaborative environment.
- Speculation says we can't predict outcomes. If we can't predict outcomes, we can't plan. If we can't plan, traditional project management theory suffers.
- Collaboration weights speculation in that a project manager plans the work between the predictable parts of the environment and adapts to the uncertainties of various factors—stakeholders, requirements, software vendors, technology, etc.

Learning:

- "Learning" cycles challenge all stakeholders and project team members.
- Based on short iterations of design, build and testing, knowledge accumulates from the small mistakes we make due to false assumptions, poorly stated or ambiguous requirements or misunderstanding the stakeholders' needs.
- Correcting those mistakes through shared learning cycles leads to greater positive experience and eventual mastery of the problem domain**.**

1.8.3.2 Dynamic Systems Development Methods (DSDM)

- The Dynamic Systems Development Method is an agile software development approach that "provides a framework for building and maintaining systems which meet tight time constraints through the use of incremental prototyping in a controlled project environment".
- DSDM is an iterative software process in which each iteration follows the 80 percent rule.
- That is, only enough work is required for each increment to facilitate movement to the next increment.
- The remaining detail can be completed later when more business requirements are known or changes have been requested and accommodated.
- DSDM life cycle that defines three different iterative cycles, preceded by two additional life cycle activities:

  Feasibility study—establishes the basic business requirements and constraints associated with the application to be built and then assesses whether the application is a viable candidate for the DSDM process.

  Business study—establishes the functional and information requirements that will allow the application to provide business value; also, defines the basic application architecture and identifies the maintainability requirements for the application.

  Functional model iteration—produces a set of incremental prototypes that demonstrate functionality for the customer.

  Design and build iteration—revisits prototypes built during functional model iteration to ensure that each has been engineered in a manner that will enable it to
  provide operational business value for end users.

  Implementation—places the latest software increment into the operational environment.
- DSDM can be combined with XP to provide a combination approach that defines a solid process model (the DSDM life cycle) with the nuts and bolts practices (XP) that are required to build

combined process model.

1.8.3.3 Scrum

- Scrum principles are consistent with the agile manifesto and are used to guide development activities within a process that incorporates the five framework activities: requirements, analysis, design, evolution, and delivery.
- Within each framework activity, work tasks occur within a process pattern called a sprint
- The work conducted within a sprint (the number of sprints required for each framework activity will vary depending on product complexity and size) is adapted to the problem at hand and is defined and often modified in real time by the Scrum team.
- Scrum emphasizes the use of a set of software process patterns that have proven effective for projects with tight timelines, changing requirements, and business criticality.
- Each of these process patterns defines a set of development actions: Backlog—a prioritized list of project requirements or features that provide business value for the customer.
- Items can be added to the backlog at any time (this is how changes are introduced).
- The product manager assesses the backlog and updates priorities as required.

1.8.3.4 Crystal

- The Crystal methodology is one of the most lightweight, adaptable approaches to software development. Crystal is actually comprised of a family of agile methodologies such as Crystal Clear, Crystal Yellow, Crystal Orange and others, whose unique characteristics are driven by several factors such as team size, system criticality, and project priorities.
- This Crystal family addresses the realization that each project may require a slightly tailored set of policies, practices, and processes in order to meet the project‘s unique characteristics.
- Several of the key tenets of Crystal include teamwork, communication, and simplicity, as well as reflection to frequently adjust and improve the process.
- Like other agile process methodologies, Crystal promotes early, frequent delivery of working software, high user involvement, adaptability, and the removal of bureaucracy or distractions.

1.8.3.5 Feature Driven Development(FDD)

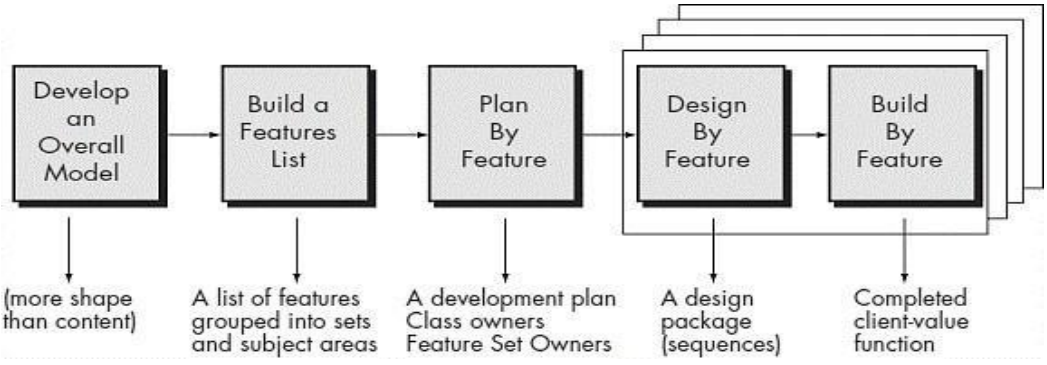


Figure1.17: Feature Driven Development Model

- FDD is a model-driven, short-iterationprocess.

- The features are small, "useful in the eyes of the client" results.
- FDD designs the rest of the development process around feature delivery using the following eight practices:

✓ Domain Object Modelling
✓ Developing by Feature
✓ Component/Class Ownership
✓ Feature Teams
✓ Inspections
✓ Configuration Management
✓ Regular Builds
✓ Visibility of progress and results

- FDD recommends specific programmer practices such as "Regular Builds" and "Component/Class Ownership".
- Unlike other agile methods, FDD describes specific, very short phases of work, which are to be accomplished separately per feature.
- These include Domain Walkthrough, Design, Design Inspection, Code, Code Inspection, and Promote to Build.

1.8.3.6 Agile Modelling (AM)

- Agile Modeling (AM) is a practice-based methodology for effective modeling and documentation of software-based systems.
- Simply put, Agile Modeling (AM) is a collection of values, principles, and practices for modeling software that can be applied on a software development project in an effective and light-weight manner.
- Although AM suggests a wide array of "core" and "supplementary" modeling principles, those that make AM unique are:

Use multiple models.

✓ There are many different models and notations that can be used to describe software.
✓ AM suggests that to provide needed insight, each model should present a different aspect of the system and only those models that provide value to their intended audience should be used.

Travel light.

✓ As software engineering work proceeds, keep only those models that will provide long-term value and jettison the rest.

Content is more important than representation.

✓ Modeling should impart information to its intended audience.
✓ A syntactically perfect model that imparts little useful content is not as valuable as a model with flawed notation that nevertheless provides valuable content for its audience.

Know the models and the tools you use to create them.

✓ Understand the strengths and weaknesses of each model and the tools that are used to create it.

Adapt locally.

✓ The modelling approach should be adapted to the needs of the agile team.

1.8.3.7 Lean Software Development (LSD):

- Lean Software Development (LSD) has adapted the principles of lean manufacturing to the world of software engineering.
- The lean principles that inspire the LSD process can be summarized as ***eliminate waste, build quality in, create knowledge, defer commitment, deliver fast, respect people, and optimize the whole.***
- For example, eliminate waste within the context of an agile software project can be interpreted to mean
- ✓ Adding no extraneous features or functions
- ✓ Assessing the cost and schedule impact of any newly requested requirement
- ✓ Removing any superfluous process steps
- ✓ Establishing mechanisms to improve the way team members find information
- ✓ Ensuring the testing finds as many errors as possible,
- ✓ Reducing the time required to request and get a decision that affects the software or the process that is applied to create it,
- ✓ Streamlining the manner in which information is transmitted to all stakeholders involved in the process.

1.8.3.8 Agile Unified Process (AUP):

- AUP adopts a "serial in the large" an "iterative in the small" philosophy for building computer-based systems.
- By adopting the classic UP phased activities –**inception, elaboration, construction, andtransition.**
- It enables a team to visualize the overall process flow for a software project.
- Each AUP iteration addresses the following activities:

(i) **Modelling**. It represents the business and problem domains.
(ii) **Implementation**. Models translated into source code.
(iii) **Testing**. Executes a series of tests to uncover errors and ensures that the source code meets its requirements.
(iv) **Deployment**. Focus on the delivery of software increment and the acquisition of feedback
(v) from end users.
(vi) **Configuration and project management**. Configuration management addresses change management, risk management, and the control of any persistent work products that are produced by the team.
(vii) **Environment management**. It coordinates a process infrastructure that includes standards, tools, and other support technology available to the team.

Agile Methods Applicability:

- Product development where a software company is developing a small or medium-sized product.
- Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are not a lot of external rules and regulations that affect the software.
- Because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems.

Problems with agile methods:

- It can be difficult to keep the interest of customers who are involved in the process.

- Maintaining simplicity requires extra work.
- Contracts may be a problem as with other approaches to iterative development.

Important questions:

Compare and Contrast the different life cycle models.

Waterfall model	**Spiral model**	**Prototyping model**	**Incremental model**
Requirements must be clearly understood and defined at the beginning only.	The Requirements analysis and gathering can be done in iterations because requirements get changed quite often.	Requirement analysis can be made in the later stages of development cycle, because requirements get changed quite often.	Requirement analysis can be made in the later stages of development cycle
The development team having the adequate experience of working on the similar project is chosen to work on this type of process model.	The development team having the adequate experience of working on the similar project is allowed in this process model.	The development team having the adequate experience of working on the similar project is allowed in this process model.	The development team having the adequate experience of working on the similar project is chosen to work on this type of process model.
There is no user involvement in all the phases of development process.	There is no user involvement in all the phases of development process.	There is user involvement in all the phases of development process.	There is user involvement in all the phases of development process.
When the requirements are reasonably well defined and the development effort suggests a purely linear effort then the **waterfall model is chosen.**	Due to iterative nature of this model the risk identification and rectification is done before they get problematic. Hence for handling real time problems the **spiral model is chosen.**	When developer is unsure about the efficiency of an algorithm or the adaptability of an operating system then **the Prototyping model is chosen.**	When the requirements are reasonably well defined and the development effort suggests a purely linear effort and when limited set of software functionality is needed quickly then the **incremental model is chosen.**

Compare and Contrast waterfall model with spiral model.

S.No	Waterfall model	Spiral model
	It requires well understanding of requirements and familiar technology.	It is developed in iterations. Hence the requirement can be identified at new iterations.
	Difficult to accommodate changes after the process has started.	The required changes can be made at every stage of new version.
	Can accommodate iteration but indirectly.	It is iterative model.
	Risks can be identified at the end which may cause failure to the product.	Risks can be identified and reduced before they get problematic.
	The customer can see the working model of the project only at the end. After reviewing of the working model, if the customer gets dissatisfied then it causes serious problems.	The customer can see the working product at certain stages of iterations.
	Customers prefer this model.	Developers prefer this model.
	This model is good for small systems.	This model is good for large systems
	It has sequential nature.	It has evolutionary nature.