1 Introduction to Software Engineering

1.1 Introduction

Today, software engineering is the most important technology with rapid development. National economy is also dependant on this technology. Day by day more and more systems are getting controlled by software. Hence it becomes necessary to understand the software development as an engineering discipline. This chapter is for understanding fundamental concepts. We will get introduced with software engineering by understanding what exactly mean by software, what are the objectives of software engineering. In this chapter, we will also discuss various categories of software. Finally we will discuss various challenges in software engineering.

1.2 Evolving Role of Software

The evolving role of software means changing role of software. Basically any software appears in two roles :

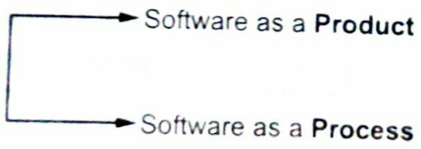


Fig. 1.1

Being a **product** the role of software can be recognised by its computing potentials, hardware capabilities and accessibility of network computers by the local hardware. Being a product it also acts as an **information transformer** i.e. producing, managing, modifying and conveying the source of information.

Being a **process** the software acts as a vehicle for driving the product. In this role the duty of software is to **control the computer** or to establish **communications** between the computers.

The role of software is significantly changing over past decade. There are many factors affecting the role of software and those are –

- Changes in computer architectures (Right from Pentium I to supercomputers)
- Improvement in hardware performance.
- Vast increase in amount of memory
- Wide variety of input and outputs(Ranging from simple text to multimedia videos)

Following table presents the different roles of software in different era of computing.

Era of computing	Software
Early Years	Batch processing Custom software
Second Era	Multi user Real time systems Database systems Product software
Third Era	Distributed systems
Forth Era	Object oriented systems Expert systems Parallel computing Network computers
Fifth Era	Web technologies mobile computing

But this evolutionary role of software brings some crucial problems. Here are some sample **problems** encountered due to evolution on software

1. Advances in hardware demand for more capable software.
2. Ability to build new programs can not meet the demand for new programs and such programs are not sufficient for business and market needs.
3. Vast use of computer based systems brings less use of manpower and ultimately society becomes more dependant on machine and not on man.
4. Constant struggle for high reliability and quality software.

[illegible]

1.3 What is Software Engineering ?

> *"Software engineering is a discipline in which theories, methods and tools are applied to develop professional software."* ✓
>
> process

In software engineering a systematic and organized approach is adopted. Based on the nature of problem and development constraints various tools and techniques are applied in order to develop quality software.

1.4 Software

Software is nothing but a collection of computer programs and related documents that are intended to provide desired features, functionalities and better performance.

Software products may be

1. Generic - That means developed to be sold to a range of different customers.
2. Custom - That means developed for a single customer according to their specification.

1.4.1 Software Characteristics

Software development is a logical activity and therefore it is important to understand basic characteristics of software. Some important characteristics of software are

- Software is engineered, not manufactured

Software development and hardware development are two different activities. A good design is a backbone for both the activities. Quality problems that occur in hardware manufacturing phase can not be removed easily. On the other hand, during software development process such problems can be rectified. In both the activities, developers are responsible for producing qualitative product.

- Software does not ware out

In early stage of hardware development process the failure rate is very high because of manufacturing defects. But after correcting such defects the failure rate gets reduced. The failure rate remains constant for some period of time and again it starts increasing because of environmental maladies (extreme temperature, dusts, and vibrations).

On the other hand software does get affected from such environmental maladies. Hence ideally it should have an *"idealized curve"*. But due to some undiscovered errors the failure rate is high and drops down as soon as the errors get corrected. Hence in failure rating of software the *"actual curve"* is as shown below.

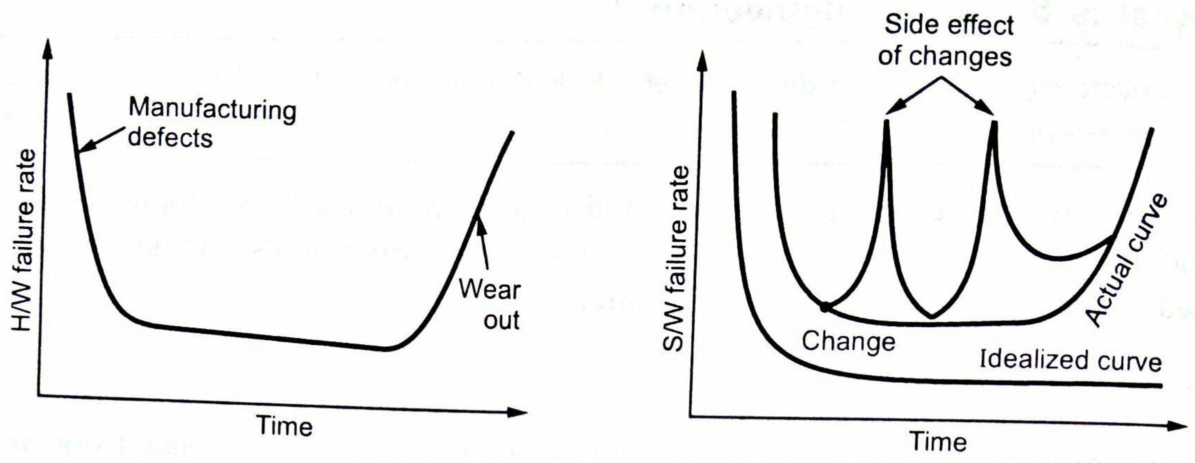


Fig. 1.2 Failure curves for Hardware and Software

During the life of software if any change is made, some defects may get introduced. This causes failure rate to be high. Before the curve can return to original steady state another change is requested and again the failure rate becomes high. Thus the failure curve looks like a spike. Thus frequent changes in software cause it to deteriorate.

Another issue with software is that there are *no spare parts for software.* If hardware component wears out it can be replaced by another component but it is not possible in case of software. Therefore software maintenance is more difficult than the hardware maintenance.

- Most software is custom built rather than being assembled from components

While developing any hardware product at first the circuit design with desired functioning properties is created. Then required hardware components such as ICs, Capacitors, and registers are assembled according to the design, but this is not done while developing software product. Most of the software is custom built.

However, now the software development approach is getting changed and we look for reusability of software components. It is practiced to reuse algorithms and data structures. Today software industry is trying to make library of reusable components. For example: in today's software, GUI is built using the reusable components such as message windows, pull down menus and many more such components. The approach is getting developed to use in-built components in the software. This stream of software is popularly known as *component engineering.*

1.4.2 Changing Nature of Software

Software can be applied in a situation for which a predefined set of procedural steps (algorithm) exists. Based on a complex growth of software it can be classified into following categories.

- **System software** - It is collection of programs written to service other programs. Typical programs in this category are compiler, editors, and assemblers. The purpose of the system software is to establish a communication with the hardware.
- **Application software** - It consists of standalone programs that are developed for specific business need. This software may be supported by database systems.
- **Engineering/scientific software** - This software category has a wide range of programs from astronomy to volcanology, from automative stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing. This software is based on complex numeric computations.
- **Embedded software** - This category consists of program that can reside within a product or system. Such software can be used to implement and control features and functions for the end-user and for the system itself.
- **Web applications** - Web application software consists of various web pages that can be retrieved by a browser. The web pages can be developed using programming languages like JAVA, PERL, CGI, HTML, DHTML.
- **Artificial Intelligence software** - This kind of software is based on knowledge based expert systems. Typically, this software is useful in robotics, expert systems, image and voice recognition, artificial neural networks, theorem proving and game playing.

1.4.3 Software Myths

There are some misbelieves in the software industry about the software and process of building software. For any software developer it is a must to know such beliefs and reality about them. Here are some typical myths-

- **Myth :** Using a collection of standards and procedures one can build software.

  Reality : Eventhough we have all standards and procedures with us for helping the developer to build software, it is not possible for software professionals to build desired product. This is because – the collection which we have should be complete, it should reflect modern techniques and more importantly it should be adaptable. It should also help the software professional to bring quality in the product.
- **Myth :** Add more people to meet deadline of the project.

  Reality : Adding more people in order to catch the schedule will cause the reverse effect on the software project i.e. software project will get delayed.

Because, we have to spend more time on educating people or informing them about the project.

- **Myth :** If a project is outsourced to a third party then all the worries of software building are over.

  Reality : When a company needs to outsource the project then it simply indicates that the company does not know how to manage the projects. Sometimes the outsourced projects require proper support for development.

- **Myth :** Even if the software requirements are changing continuously it is possible to accommodate these changes in the software.

  Reality : It is true that software is a flexible entity but if continuous changes in the requirements have to be incorporated then there are chances of introducing more and more errors in the software. Similarly the additional resources and more design modifications may be demanded by the software.

- **Myth :** We can start writing the program by using general problem statements only. Later on using problem description we can add up the required functionalities in the program.

  Reality : It is not possible each time to have comprehensive problem statement. We have to start with general problem statements; however by proper communication with customer the software professionals can gather useful information. The most important thing is that the problem statement should be unambiguous to begin with.

- **Myth :** Once the program is running then its over!

  Reality : Even though we obtain that the program is running major part of work is after delivering it to customer.

- **Myth :** Working program is the only work product for the software project.

  Reality : The working program/software is the major component of any software project but along with it there are many other elements that should be present in the software project such as documentation of software, guideline for software support.

- **Myth :** There is no need of documenting the software project; it unnecessarily slows down the development process.

  Reality : Documenting the software project helps in establishing ease in use of software. It helps in creating better quality. Hence documentation is not wastage of time but it is a must for any software project.

1.5 Goals/Objectives of Software Engineering

While developing software following are common objectives.

1. **Satisfy user requirements** - Many programmers simply don't do what the end user wants because they do not understand user requirements. Hence it becomes necessary to understand the demand of end user and accordingly software should be developed.
2. **High reliability** - Mistakes or bugs in a program can be expensive in terms of human lives, money, and customer relation. For instance Microsoft has faced many problems because earlier release of Windows has many problems. Thus software should be delivered only if high reliability is achieved.
3. **Low maintenance costs** - Maintenance of software is an activity that can be done only after delivering the software to the customer. Any small change in software should not cause restructuring of whole software. This indicates that the design of software has poor quality.
4. **Delivery on time** - It is very difficult to predict the exact time on which the software can be completed. But a systematic development of software can lead to meet the given deadline.
5. **Low production costs** - The software product should be cost effective.
6. **High performance** - The high performance software are expected to achieve optimization in speed and memory usage.
7. **Ease of reuse** - Use same software in different systems and software.

Environments reduce development costs and also improve the reliability. Hence reusability of developed software is an important property.

1.6 Challenges in Software Engineering

The key challenges facing software engineering are :

- Coping with legacy systems

  Old, valuable systems must be maintained and updated. Hardware is evolved faster than software. If original developer have moved on managing, maintaining or integrating of software becomes a critical issue.

different groups ←

- Heterogeneity challenge

  Sometimes systems are distributed and include a mix of hardware and software. This implies that software systems must cleanly integrate with other different software systems, built by different organizations and teams using different hardware and software platforms.

- Delivery time challenge

  There is increasing pressure for faster delivery of software. As the complexity of systems that we develop increases, this challenge becomes harder.

As software is an integral part of computer based systems, it is essential to apply software engineering principles and practices while developing software. Hence the main objective of software engineering is to adopt systematic, disciplined approach while building high quality software.

Review Questions

1. *What is software engineering ?*
2. *Define the term software.*
3. *Explain characteristics of software.*
4. *Give the evolving role of software.*
5. *Give the classification of software based on changing nature of software.*
6. *Explain five software myths.*
7. *What are the goals of software engineering.*
8. *What are the key challenges in software engineering.*

❑❑❑

2 A Generic View of Software Process

2.1 Introduction

Software engineering is the establishment and sound engineering principles applied to obtain reliable and efficient software in an economical manner.

Software engineering includes process, management techniques, technical methods, and the use of tool. While building any software, the software process provides the interaction between user and developer. In this chapter we will understand the basic concept of process and process models.

2.1.1 Layered Technology

- Software engineering is a layered technology. Any software can be developed using these layered approaches. Various layers on which the technology is based are quality focus layer, process layer, methods layer, tools layer.

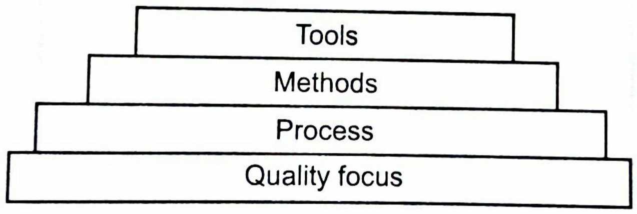


Fig. 2.1 Software engineering layers

- A disciplined quality management is a backbone of software engineering technology.
- Process layer is a foundation of software engineering. Basically, process defines the framework for timely delivery of software.
- In method layer the actual method of implementation is carried out with the help of requirement analysis, designing, coding using desired programming constructs and testing.
- Software tools are used to bring automation in software development process.

Thus software engineering is a combination of process, methods, and tools for development of quality software.

2.2 Software Process

Software process can be defined as the structured set of activities that are required to develop the software system.

The fundamental activities are

- Specification
- Design and implementation
- Validation
- Evolution

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

2.2.1 Common Process Framework

The process framework is required for representing the common process activities. It is as shown below.

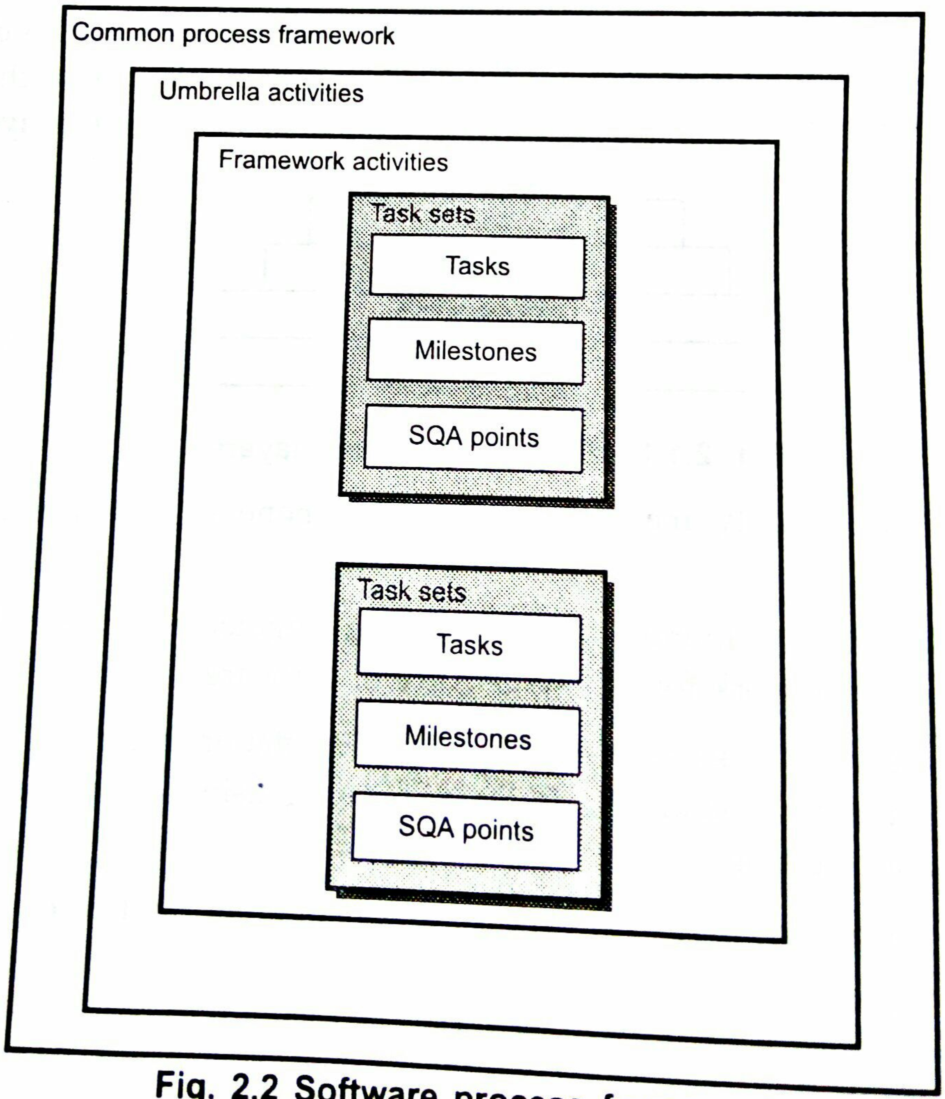


Fig. 2.2 Software process framework

As shown in figure the software process is characterized by process framework activities, task sets and umbrella activities.

Process Framework Activities

- Communication
- By communicating customer requirement gathering is done.
- Planning – Establishes engineering work plan, describes technical risks, lists resource requirements, work products produced, and defines work schedule.
- Modelling – The software model is prepared by:
    - Analysis of requirements
    - Design
    - Construction – The software design is mapped into a code by:
    - Code generation
    - Testing
- Deployment – The software delivered for customer evaluation and feedback is obtained.

Task sets – The task set defines the actual work done in order to achieve the software objective. The task set is used to adopt the framework activities and project team requirements using

- Collection of software engineering work tasks
- Project milestones
- Software quality assurance points

Umbrella activities – The umbrella activities occur throughout the process. They focus on project management, tracking and control. The umbrella activities are

1. **Software project tracking and control** – This is an activity in which software team can assess progress and take corrective action to maintain schedule.
2. **Risk management** – The risks that may affect project outcomes or quality can be analyzed.
3. **Software quality assurance** – These are activities required to maintain software quality.
4. **Formal technical reviews** – It is required to assess engineering work products to uncover and remove errors before they propagate to next activity.
5. **Software configuration management** – Managing of configuration process when any change in the software occurs.

6. **Work product preparation and production** – The activities to create models, documents, logs, forms, and lists are carried out.
7. **Reusability management** – It defines criteria for work product reuse.
8. **Measurement** – In this activity, the process can be defined and collected. Also project and product measures are used to assist the software team in delivering the required software.

2.2.2 Capability Maturity Model (CMM)

- The Software Engineering Institute (SEI) has developed a comprehensive process meta-model emphasizing process maturity. It is predicated on a set of system and software capabilities that should be present when organizations reach different levels of process capability and maturity.
- The Capability Maturity Model (CMM) is used in assessing how well an organization's processes allow to complete and manage new software projects.
- Various process maturity levels are

Level 1 : Initial – Few processes are defined and individual efforts are taken.

Level 2 : Repeatable – To track cost schedule and functionality basic project management processes are established. Depending on earlier successes of projects with similar applications necessary process discipline can be repeated.

Level 3 : Defined – The process is standardized, documented and followed. All the projects use documented and approved version of software process which is useful in developing and supporting software.

Level 4 : Managed – Both the software process and product are quantitatively understood and controlled using detailed measures.

Level 5 : Optimizing – Establish mechanisms to plan and implement change. Innovative ideas and technologies can be tested.

Thus CMM is used for improving the software project.

2.3 Process Patterns

As we know, process is defined as a series of activities in which one or more inputs are used to produce one or more outputs. Now in this section we will discuss one more important concept that is **process pattern**. The process pattern is a template which appears as a general solution to a common problem. Process pattern acts as a consistent method for describing an important characteristic of software process. Using process pattern we can easily build the process that satisfies all the requirements.

Process pattern describes –

- Complete process
- Important framework activity
- Task within the framework activity

Scott Ambler – an object oriented consultant has proposed a template for process pattern as follows

- Pattern name
- Intent
- Type
- Initial context
- Problem
- Solution
- Resulting context
- Known uses

The description of process pattern is as follows –

Pattern name

The pattern name should be a meaningful name given to the pattern. From pattern name one can guess its functionality.

Intent

The objective or the purpose of the pattern should be described here.

Type

The type of pattern should be specified here.

Ambler has suggested three types of patterns

1) *Task Pattern* - It represents the software engineering action or work task which is a part of process. For example **Formal Technical review** is a task pattern.
2) *Stage Pattern*- It defines the process framework activity. A framework activity has multiple work tasks; hence stage pattern consists of multiple task patterns. For example **Coding phase** is a stage pattern.

3) *Phase Patterns*- It defines the sequence of framework activities. For example the phrase pattern can be **spiral model or rapid prototype model.**

Initial context

In this section the conditions under which the pattern applies are described.

Sometimes the **entry conditions** must be true before the process begins.

In this section following issues need to described

1. The set of organisational or team related activities that have already occurred.
2. The list of entry state processed.
3. Already existing software engineering or project related information.

Problem

Under this section the problem is mentioned for which the pattern is to be described. For example *insufficient requirements* is a problem. That means customers are not sure about what they want exactly. They could not specify the requirements in proper manner.

Solution

Every problem for which pattern has to be described should be accompanied with some solution. For example: The problem of *insufficient requirements* has solution. That is - Establish effective communication with the customer. Ask questions in order to obtain meaningful requirements. Conduct timely reviews to modify/redefine the requirements. This solution will help the software developer to get useful information before the actual work starts.

Resulting context

It describes the results after successful implementation of pattern. The resulting context should have following type of information on successful completion of pattern –

1. The team-related or organizational activities that must have occurred.
2. Exit state for the process.
3. The software engineering information or project information that has been developed.

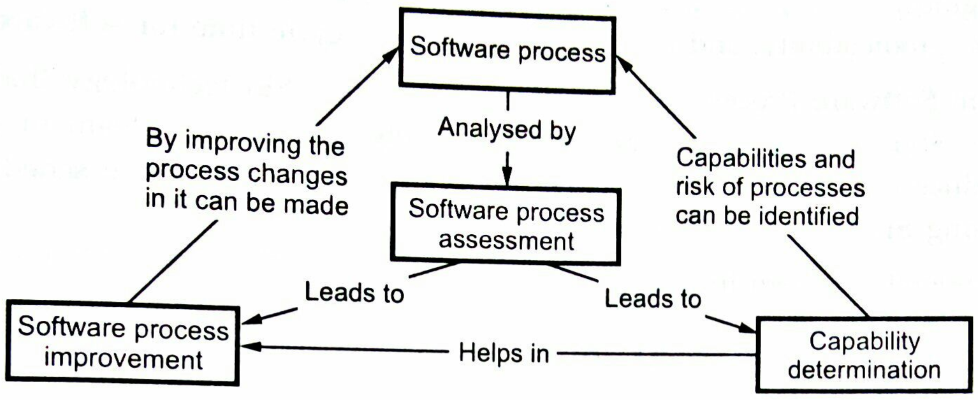
Known uses/Examples

The specific instances or applications in which the described pattern is useful should be mentioned under this section. In other words we describe applicability of the pattern. For example spiral model is useful for the large scale projects in which work products must be examined in several iterations.

2.4 Process Assessment

Normally process is suffered by following problems –

1. The software has to be delivered on time.
2. The software should satisfy customer needs and requirements.
3. The software should posses the long term quality characteristics.



Software process
Analysed by
Software process assessment
By improving the process changes in it can be made
Capabilities and risk of processes can be identified
Leads to
Leads to
Software process improvement
Helps in
Capability determination

Fig. 2.3 Software process assessment

Process assessment is an activity in which it is ensured whether the software meets the basic criteria for successful software engineering project.

Following approaches are used for software assessment –

Standard CMMI assessment method for process improvement

It is a five step process assessment model. These five steps are initiating, diagnosing, establishing, acting and learning. This model makes use of SEI CMM as the base model.

CMM-based appraisal for internal process improvement

This method provides the diagnostic technique for internal process assessment.

SPICE

Using this standard all the requirements are analysed for software process assessment. This standard helps in doing an objective evaluation on efficiency of any process.

ISO 9001:2000

This is a popularly used standard for improving the overall quality of the organization. The International Organization for Standardization i.e. ISO has developed this standard.

2.5 Personal and Term Process Models

Watts Humphrey developed PSP and TSP at the SEI at Carnegie Mellon in the mid-1990's.

The Personal Software Process (PSP) is a SEI (Software Engineering Institute) technology that brings discipline to the software development habits of individual software engineer. It helps in dramatically improving product quality, increasing cost and schedule predictability, and reducing development cycle time for software.

The Team Software Process (TSP) is a complementary SEI technology that **enables teams** to develop software products more effectively. TSP helps a team of engineers how to produce quality products for planned costs and on aggressive schedules. PSP is like applying **Six Sigma** to Software Development.

Let us understand them in detail.

2.5.1 Personal Software Process (PSP)

The computer software is built using various processes developed by every individual. There are different kinds of software processes; random, intended for specific purpose or may be changing each time. These processes are developed by individuals in an organization. Watts Humphrey suggested that to make each process effective every individual who is developing them has to change themselves first. Under PSP every practitioner is made responsible for controlling the quality of corresponding processes. Under PSP model five framework activities are suggested as follows –

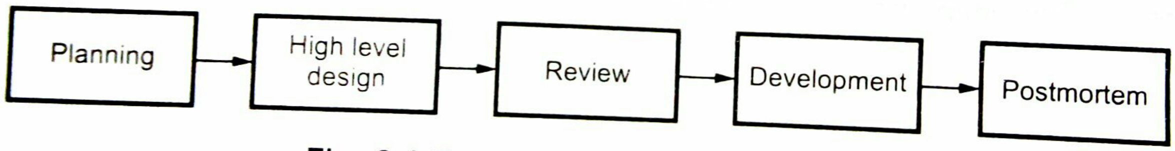


Fig. 2.4 Framework activities in PSP

Planning

In this activity, requirements are identified based on these requirements size, resource estimates and defect estimates are made. All metrics are arranged in template form. Development task are identified and project schedule is created.

High level design

The specification is created first and then component level design is created. To resolve the uncertainty prototypes are created.

Review

Formal technical reviews are made for uncovering the errors. Metrics are formed for important tasks and recorded.

Development

Under this activity code is generated. The generated code is then reviewed, modified and tested. Metrics are formed for important tasks and recorded.

Postmortem

Using the collected measures and metrics the effectiveness of the process is analysed. These measures and metrics should direct certain changes in the process in order to improve its efficiency.

Thus the emphasis of PSP is to identify **errors** in the **early stage** of software development. Using the systematic approach of PSP every work product is assessed with great care.

2.5.2 Term Process Models (TSP)

The term process model (TSP) is designed to produce strategy and set of operational procedures for using disciplined software methods at **team levels**. The goal of TSP is to have **self directed project team** for producing high quality software. Following are the objectives of Term Process models.

1. Build the self directed team for planning the software project in systematic manner. The normal size of team should be 3 to 20 software engineers.
2. Indicate the project manager for the coaching needed by the team members for the performance improvement.
3. Using CMM level 5 (optimizing level), improve the software process.
4. Provide the improvement guidance for the software team.
5. Provide the university teaching.

There are five framework activities in TSP

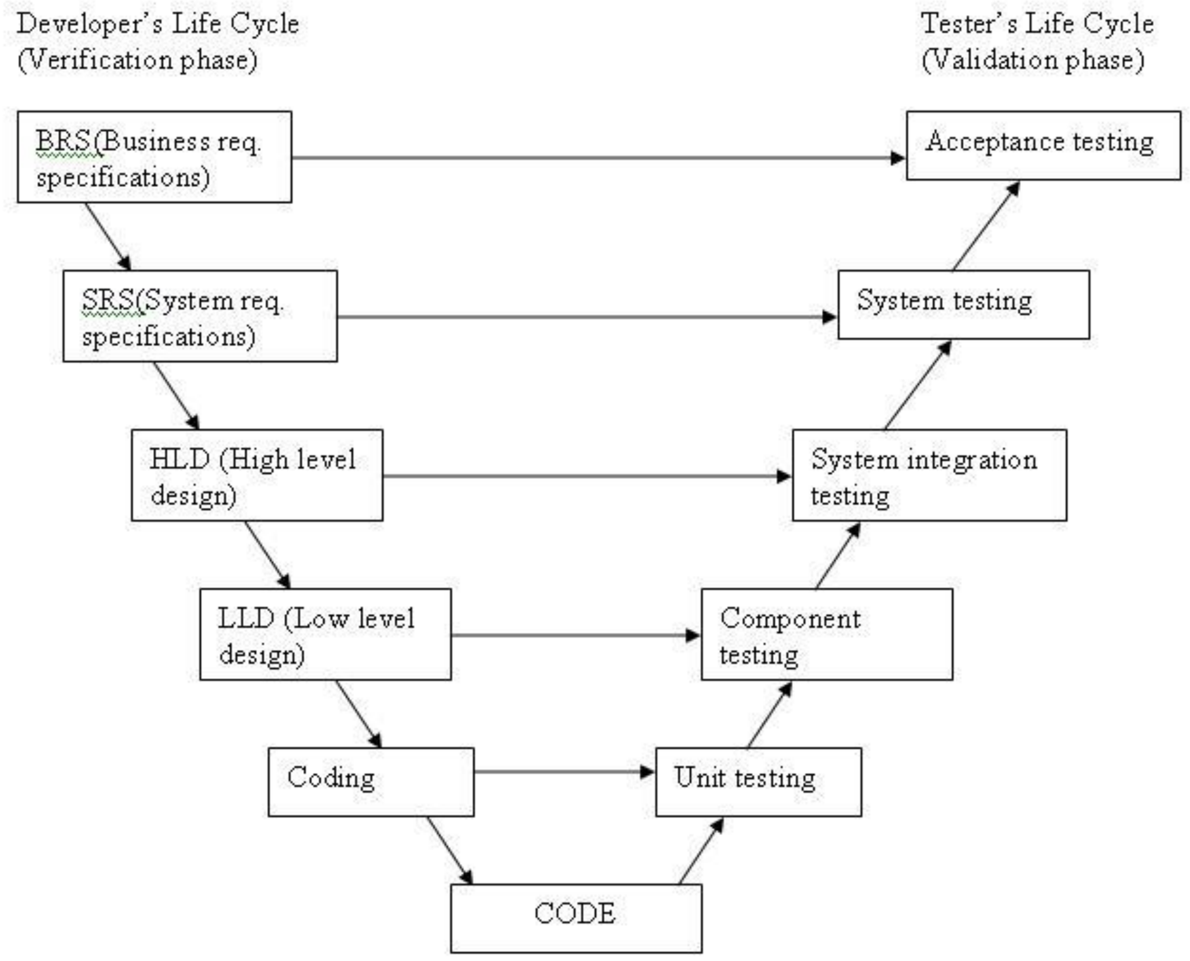
- Launch
- High level design
- Implementation
- Integration and test
- Postmortem

In TSP there is great use of scripts, standards and forms. These documents help the team to seek guidance in their work and improve the overall quality of the work product.

Thus in this chapter we have discussed all the process related concepts.

V MODEL

V- model means Verification and Validation model. , the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. **V-Model** is one of the **many software development models**. Testing of the product is planned in parallel with a corresponding phase of development in **V-model**.



The various phases of the V-model are as follows:

Requirements like BRS and SRS begin the life cycle model just like the waterfall model. But, in this model before development is started, a **system test** plan is created. The **test plan** focuses on meeting the functionality specified in the requirements gathering.

The high-level design (HLD) phase focuses on system architecture and design. It provide overview of solution, platform, system, product and service/process. An **integration test** plan is created in this phase as well in order to test the pieces of the software systems ability to work together.

The low-level design (LLD) phase is where the actual software components are designed. It defines the actual logic for each and every component of the system. Class diagram with all the methods and relation between classes comes under LLD. **Component tests** are created in this phase as well.

The implementation phase is, again, where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.

Coding: This is at the bottom of the V-Shape model. Module design is converted into code by developers. **Unit Testing** is performed by the developers on the code written by them.

Advantages of V-model:

- Simple and easy to use.
- Testing activities like planning, **test designing** happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
- Proactive defect tracking – that is defects are found at early stage.
- Avoids the downward flow of the defects.
- Works well for small projects where requirements are easily understood.

Disadvantages of V-model:

- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- If any changes happen in midway, then the test documents along with requirement documents has to be updated.

When to use the V-model:

- The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
- The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise.

- High confidence of customer is required for choosing the V-Shaped model approach. Since, no prototypes are produced, there is a very high risk involved in meeting customer expectations.

3 Process Models

3.1 Introduction

Process models are proposed in order to adopt systematic approach in software development. These models define the distinct set of activities, tasks and work products that are required to create high quality software. There are different process models with different terminologies but their generic framework activities are almost same. To be more specific these process models can also be called as **prescriptive process model** because they prescribe the process elements such as framework activities, software engineering actions, tasks, work products and quality. All software process models can accommodate the general framework activities hence they are also recognised as **generic software process models**.

In this chapter we discuss various generic process models along with their merits and demerits and applicability.

3.2 Process model

The process model can be defined as an abstract representation of process. The process model is chosen based on nature of software project.

Generic software process models are -

- The Waterfall Model - Separate and distinct phases of specification and development;
- Prototyping Model - A quick design approach is adopted.
- Incremental Models – It emphasizes on short development cycle.
  - Rapid Application and Development (RAD) Model
- Evolutionary Process Models - Specification, development and validation are interleaved.
  - Incremental Model
  - Spiral Model
  - WINWIN spiral model
  - Concurrent Development

3.3 Life Cycle Models

A life cycle is the sequence in which a project specifies, prototypes, designs, implements, tests, and maintains a piece of software. In software engineering, the life cycle model depicts various stages of software development process. Using life cycle model various development issues can be solved at the appropriate time.

3.3.1 Waterfall Model

- The waterfall model is also called as '**linear-sequential model**' or '**classic life cycle model**'. It is the oldest software paradigm. This model suggests a systematic, sequential approach to software development.
- The software development starts with requirements gathering phase. Then progresses through analysis, design, coding, testing and maintenance. Following figure illustrates waterfall model.

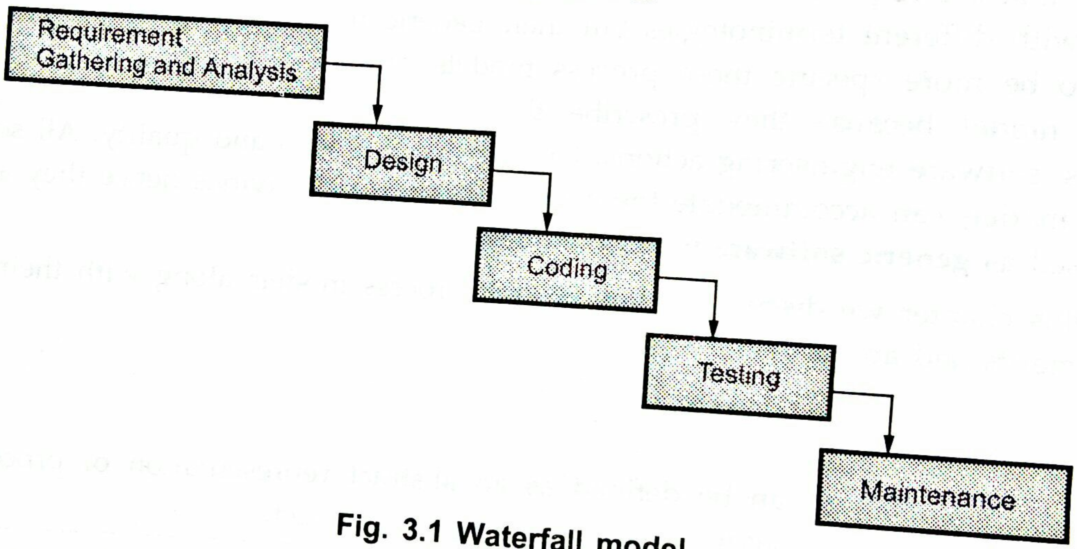


Fig. 3.1 Waterfall model

- In **requirement Gathering and analysis** phase the basic requirements of the system must be understood by software engineer, who is also called **Analyst**. The information domain, function, behavioural requirements of the system are understood. All these requirements are then well documented and discussed further with the customer, for reviewing.
- The **design** is an intermediate step between requirements analysis and coding.

Design focuses on program attributes such as -

- Data structure
- Software architecture
- Interface representation
- Algorithmic details.

The requirements are translated in some easy to represent form using which coding can be done effectively and efficiently. The design needs to be documented for further use.

- **Coding is a step in which design is translated into machine-readable form. If design is done with sufficient detail then coding can be done effectively.** Programs are created in this phase.
- **Testing** begins when coding is done. While performing testing the major focus is on **logical internals** of the software. The testing ensures execution of all the paths, functional behaviours. The purpose of testing is to uncover errors, fix the bugs and meet the customer requirements.
- **Maintenance** is the longest life cycle phase. When the system is installed and put in practical use then errors may get introduced, correcting such errors and putting it in use is the **major purpose** of maintenance activity. Similarly enhancing system's services as new requirements are discovered is again maintenance of the system.

This model is widely used model, although it has many drawbacks. Let us discuss drawbacks of waterfall model.

Drawbacks of waterfall model

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

1. 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.
2. 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.
3. 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.
4. Linear nature of waterfall model induces **blocking states,** because certain tasks may be dependant on some previous tasks. Hence it is necessary to accomplish all the dependant tasks first. It may cause long waiting time.

3.4 Prototyping Model

- In prototyping model initially the requirement gathering is done.
- Developer and customer define overall objectives; identify areas needing more requirement gathering.

- Then a quick design is prepared. This design represents what will be visible to user- in input and output format.
- From the quick design a prototype is prepared. Customer or user evaluates the prototype in order to refine the requirements. Iteratively prototype is tuned for satisfying customer requirements. Thus prototype is important to identify the software requirements.
- When working prototype is built, developer use existing program fragments or program generators to throw away the prototype and rebuild the system to high quality.

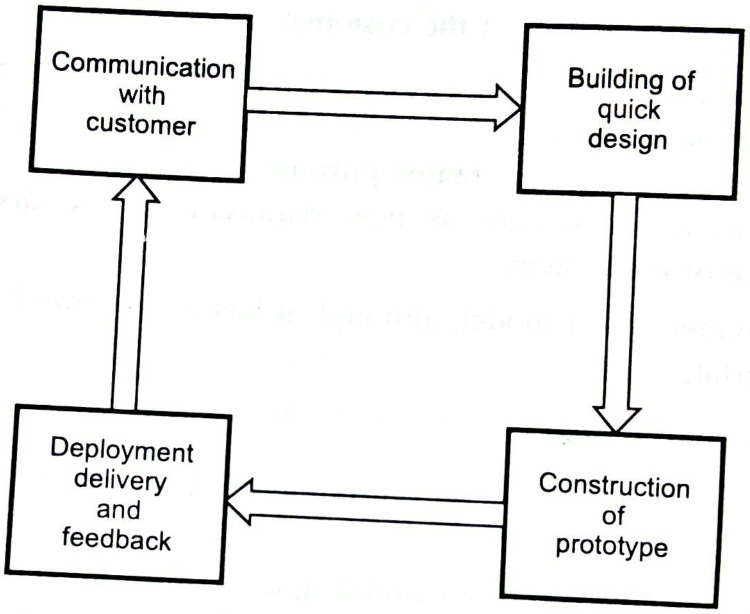

Communication with customer
Building of quick design
Construction of prototype
Deployment delivery and feedback

Fig. 3.2 Prototyping

- Certain classes of mathematical algorithms, subset of command driven systems and other applications where results can be easily examined without real time interaction can be developed using prototyping paradigm.

When to choose it ?

- Software applications that are relatively easy to prototype almost always involve human-machine interaction (HCI) the prototyping model is suggested.
- A general objective of software is defined but not detailed input, processing or output requirements. Then in such a case prototyping model is useful.
- When the developer is unsure of the efficiency of an algorithm or the adaptability of an operating system then prototype serves as a better choice.

Drawbacks of Prototyping

1. 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.
2. The first version may have some compromises.
3. 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.

3.5 Incremental Model

- The incremental model has same phases that are in waterfall model. But it is iterative in nature. The incremental model has following phases.

  1. Analysis 2. Design

  3. Code 4. Test

See Fig. 3.3 on next page.

- The incremental model delivers series of releases to the customer. These releases are called **increments**. More and more functionality is associated with each increment.
- 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 word processing software package can be considered as an example of incremental model. In the first increment only the document processing facilities are available. In the second increment, more sophisticated document producing and processing facilities, file management functionalities are given. In the next increment spelling and grammar checking facilities can be given. Thus in incremental model progressive functionalities are obtained with each release.

When to choose it?

1. When requirements are reasonably well-defined.
2. When overall scope of the development effort suggests a purely linear effort.
3. When limited set of software functionality needed quickly.

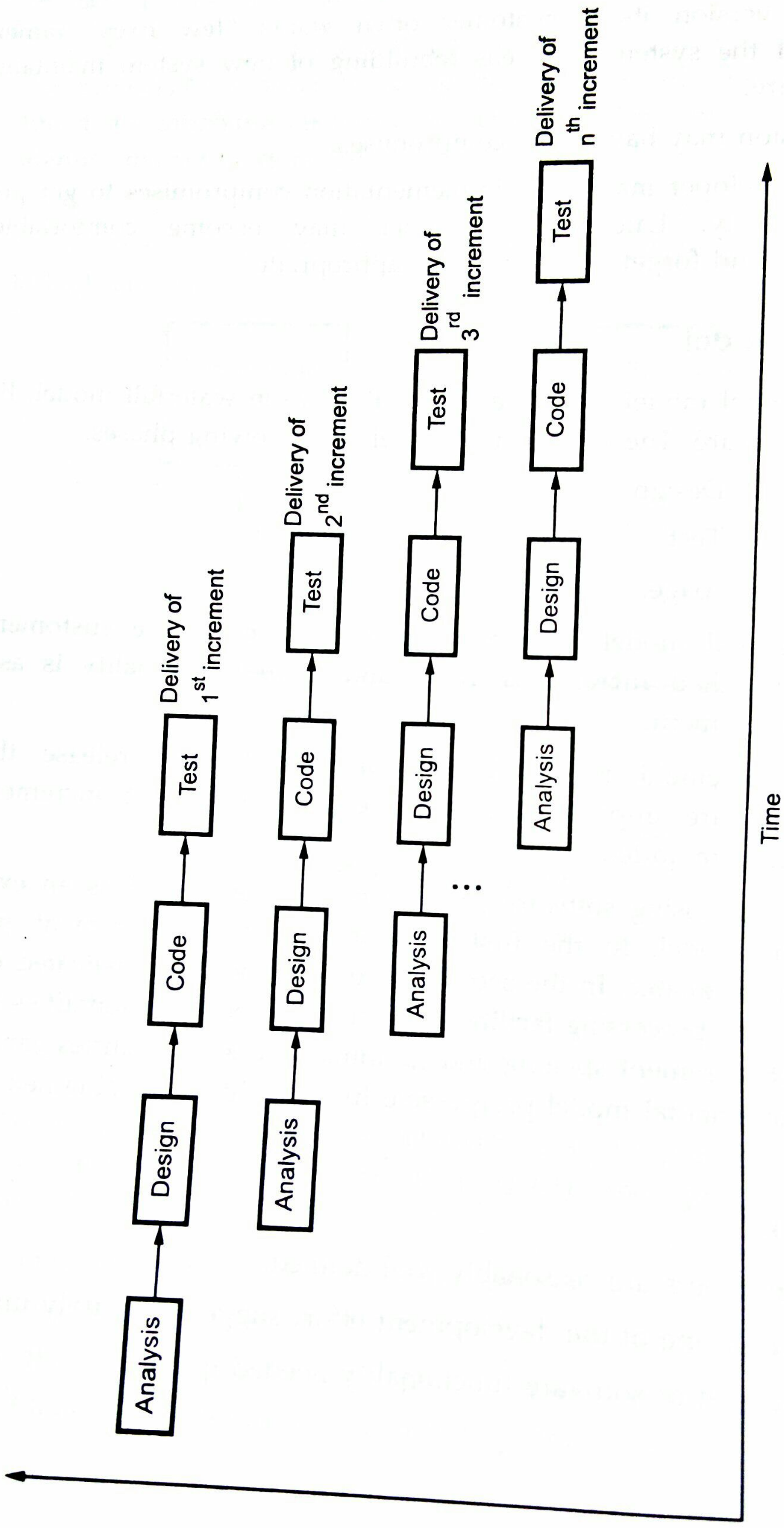

Analysis
Design
Code
Test
Delivery of 1st increment
Analysis
Design
Code
Test
Delivery of 2nd increment
Analysis
Design
Code
Test
Delivery of 3rd increment
...
Analysis
Design
Code
Test
Delivery of nth increment
Time

Fig. 3.3 The incremental model

Merits of incremental model

1. The incremental model can be adopted when there are less number of people involved in the project.
2. Technical risks can be managed with each increment.
3. For a very small time span, at least core product can be delivered to the customer.

3.5.1 Rapid Application Development (RAD) Model

- The rapid application development model is type of incremental software process model in which there is extremely short development cycle.
- This model is similar to waterfall model which achieves the high speed development using **component based construction.**
- To develop the fully functional system within short time period using this model it is necessary to understand the requirements fully and to have a restricted project scope.

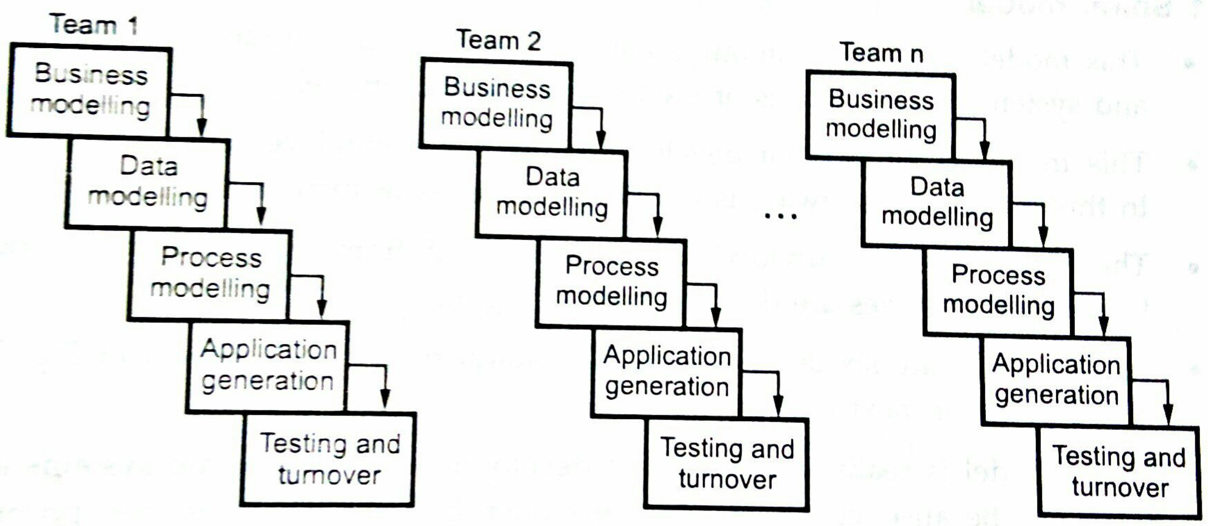


Fig. 3.4 Rapid application development model

- Various phases of RAD model are

1) **Business modeling** – In business modeling, the information flow is modeled into various business functions. These business functions collect following information.

- Information that drives the business process.
- The type of information being generated.
- The generator of information.
- The information flow.
- The processor of information.

2) **Data modeling** – In this phase the information obtained in business model is classified into data objects. The characteristics of data objects (attributes) are identified. The relationship among various data objects is defined.

3) **Process modeling** – In this phase the data objects are transformed into processes. These processes are to extract the information from data objects and are responsible for implementing business functions.

4) **Application generation** – For creating software various automation tools can be used. RAD also makes use of reusable components or creates reusable components to have rapid development of software.

5) **Testing and turnover** – As RAD uses reusable components the testing efforts are reduced. But if new components are added in software development process then such components need to be tested. It is equally important to test all the interfaces.

3.6 Evolutionary process Model

3.6.1 Spiral model

- This model possess the iterative nature of prototyping model and controlled and systematic approaches of the linear sequential model.
- This model gives efficient development of incremental versions of software. In this model, the software is developed in series of increments.
- The sprial model is divided into a number of framework activities. These framework activities are denoted by task regions.
- Usually there are six tasks regions. The spiral model is as shown in Fig. 3.5. (See Fig. 3.5 on next page).
- 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.
- In the initial pass, product specification is built and in subsequent passes around the spiral the prototype gets developed and then more improved versions of software gets developed.
- During planning phase, the cost and schedule of software can be planned and adjusted based on feedback obtained from customer evaluation.
- In spiral model, project entry point axis is defined. This axis represents starting point for different types of projects.

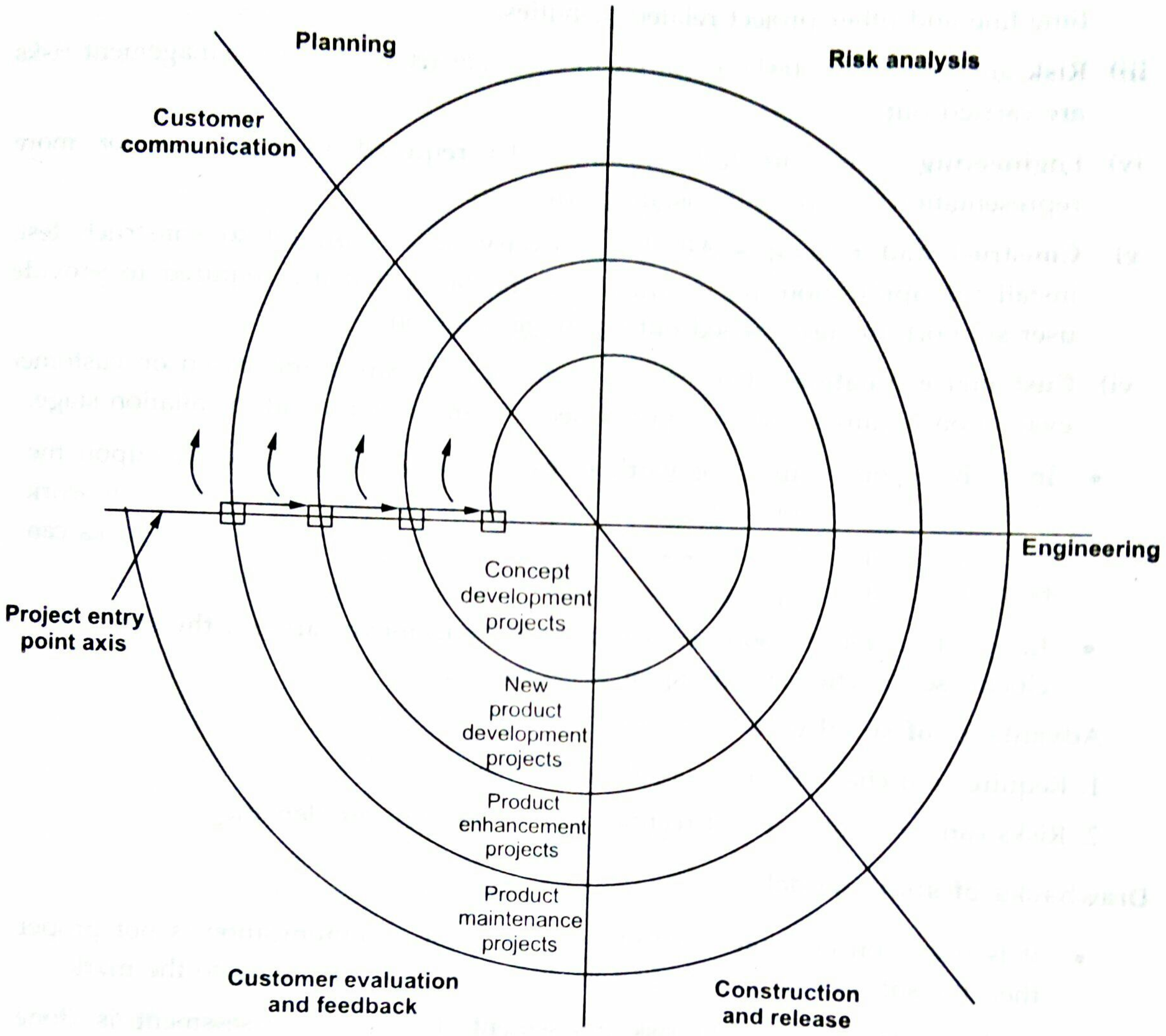

Planning
Risk analysis
Customer communication
Engineering
Project entry point axis
Concept development projects
New product development projects
Product enhancement projects
Product maintenance projects
Customer evaluation and feedback
Construction and release

Fig. 3.5 Spiral model

For instance concept development project will start at core of spiral and will continue along the spiral path. If the concept has to be developed into actual project then at entry point 2 the product development process starts. Hence entry point 2 is called product development project entry point. The development of the project can be carried out in iterations.

- The task regions can be described as,

i) Customer communication – In this region it is suggested to establish customer communication.

ii) **Planning** – All planning activities are carried out in order to define resources time line and other project related activities.

iii) **Risk analysis** – The tasks required to calculate technical and management risks are carried out.

iv) **Engineering** – In this task region, tasks required to build one or more representations of applications are carried out.

v) **Construct and release** – All the necessary tasks required to construct, test, install the application are conducted. Some tasks that are required to provide user support are also carried out in this task region.

vi) **Customer evaluation** – Customer's feedback is obtained and based on customer evaluation required tasks are performed and implemented at installation stage.

- In each region, number of **work tasks** are carried out depending upon the characterisics of project. For a small project relatively small number of work tasks are adopted but for a complex project large number of work tasks can be carried out.
- In spiral model, the software engineering team **moves around the spiral** in a clockwise direction beginning at the core.

Advantages of spiral model

1. Requirement changes can be made every stage.
2. Risks can be identified and rectified before they get problematic.

Drawbacks of spiral model

- 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.

3.7 The Unified Process

The unified process is a framework for object oriented models. This model is also called as **Rational Unified Process model (RUP)**. It is proposed by Ivar Jacobson, Grady Bootch and James Rumbaugh. This model is iterative and incremental by nature. Let us discuss various phases of unified process.

There are 5 phases of unified process model and those are

- Inception
- Elaboration
- Construction

- Transition
- Production

Let us understand each of these phases in detail.

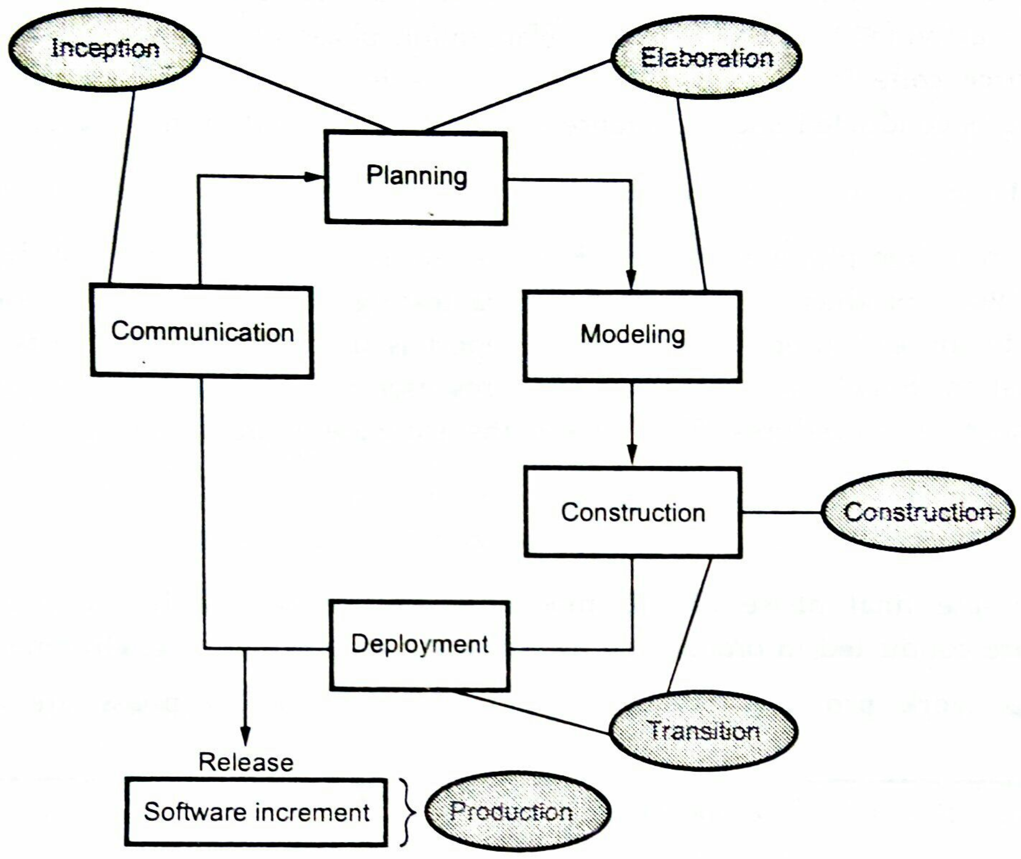


Fig. 3.6 Unified Process Model

Inception

In this phase there are two major activities that are conducted: ***communication*** and ***planning***. By having customer communication business requirements can be identified. Then a rough architecture of the system is proposed. Using this rough architecture it then becomes easy to make a plan for the project. **Use cases** are created which elaborates the user scenario. Using use cases the sequence of actions can be identified. Thus use cases help to identify the scope of the project which ultimately proves to be the basis for the plan.

Elaboration

Elaboration can be done using two activities: ***planning*** and ***modelling***. In this phase the use cases are redefined. And an architectural representation is created using five models such as use-case model, analysis model, design model, implementation model and deployment model. Thus **executable baseline** gets created. Then a plan is created carefully to check whether scope, risks and delivery dates are reasonable.

Construction

The main activity in this phase is to make the use cases **operational**. The analysis and design activities that are started in elaboration phase are completed in this phase and a **source code** is developed which implements all desired functionalities. Then unit testing is conducted and acceptance testing is carried out on the use cases.

Transition

In the transition phase all the activities that are required at the time of deployment of the software product are carried out. **Beta testing** is conducted when software is delivered to the end user. **User feedback** report is used to remove defects from the created system. Finally software team prepares user manuals, installation guides and trouble shooting procedures. This makes the software more usable at the time of release.

Production

This is the final phase of this model. In this phase mainly the maintenance activities are conducted in order to support the user in operational environment.

Various **work products** that may get generated in every phase are as given below –

Inception Phase	Elaboration Phase	Construction Phase	Transition phase
• Initial use case model • Initial risk assessment • Project plan	• Use case model • Requirements Analysis model • Architecture model • Preliminary design model • Risk list • Iterative project plan • Preliminary user manual	• Design model • Software components • Test plan • Test cases • User manual • Installation manual	• Delivered software increments • Beta test report • User feedback report.

Thus the generic software process models are applied for many years in software development process in order to reduce chaos in the process of development. Each of these models suggest different process flow but the insist on performing the same set of generic framework activities.

Solved Exercise

Q. 1 *Explain an advantage of adhering to life cycle model for software development.*

Ans. : A life cycle is the sequence in which a project specifies, prototypes, designs, implements, tests, and maintains a piece of software. In software engineering, the life cycle model depicts various stages of software development process. Using life cycle model various development issues can be solved at the appropriate time. The technical risks can be managed during development of the software. The requirement elicitation can be done before actual product gets developed. If the life cycle model is used for developing the software product then the testing and maintenance can be carried out effectively and efficiently. For developing some software systems there is need for parallel development of different parts of the system. In such cases life cycle model is preferred.

Q. 2 *Why software architecture is important in software process ?*

Ans. : The software architecture gives the hierarchical structure of software components and their interactions. In software architecture the software model is designed and structure of that model is partitioned horizontally or vertically. This helps in implementing a quality software product.

Q. 3 *What are the drawbacks of rapid application development life cycle model?*

Ans. :
1. It requires multiple teams or large number of people to work on the scalable projects.
2. This model requires heavily committed developer and customers. If the commitment is lacking then RAD projects will fail.
3. The projects using RAD model require heavy resources.
4. If there is no appropriate modularization then RAD projects fail. Performance can be a problem to such projects.
5. The projects using RAD model find it difficult to adopt new technologies.

Q. 4 *How does a spiral model represent a process suitable to represent a real time problem?*

Ans. : Spiral model represents a process suitable to represent a real time problem because of following reasons -

1. Software evolves as the project progresses. And at every evolutionary level the risks are identified and managed and risks are reduced at every stage.
2. It enables the developer to apply the prototype approach at any stage in the evolution of the product. It helps in adopting the approach systematic stepwise development of the product.
3. The iterative frameworks help in analyzing the product at every evolutionary stage.

4. The spiral model demands a direct consideration of technical risks at all stages of project. The risks are reduced before they get problematic.

Q. 5 *Compare and contrast waterfall model with spiral model.*

Ans. :

Waterfall model	Spiral model
It requires well understanding of requirements and familiar technology.	It is developed in iterations. Hence the requirements 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.

Q. 6 *List out the problems encountered in Linear sequential model.*

Ans. : The problems encountered in linear sequential model are -

1. 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.
2. 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.
3. 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.
4. Linear nature of this model induces blocking states, because certain tasks may be dependent on some previous tasks. Hence it is necessary to accomplish all the dependent tasks first. It may cause long waiting time.

Q. 7 *Which type of applications suit RAD model? Justify your answer.*

Ans. : The RAD model is suitable for information system applications, business applications and the for systems that can be modularized because of following reasons -

1. This model is similar to waterfall model but it uses very short development cycle.
2. It uses component-based construction and emphasises reuse and code generation.
3. This model uses multiple teams on scaleable projects.
4. The RAD model is suitable for the projects where technical risks are not high.
5. The RAD model requires heavy resources.

Q. 8 *What is meant by 'blocking states' in linear sequential model?*

Ans . : The linear nature of linear sequential model brings a situation in the project that some project team members have to wait for other members of the team to complete the dependent tasks. This situation is called "blocking state" in linear sequential model. For example after performing the requirement gathering and analysis step the design process can be started. Hence the team working on design stage has to wait for gathering of all the necessary requirements. Similarly the programmers can not start coding step unless and until the design of the project is completed.

Q. 9 *Identify in which phase of the software life cycle the following documents are delivered.*

(a) Architectural design

(b) Test plan

(c) Cost estimate

(d) Source code document

Ans:

	Document	Phase
(a)	Architectural Design	Design
(b)	Test plan	Testing
(c)	Cost estimate	Project management and planning
(d)	Source code document	Coding

Q. 10 *Define the terms product and process in software engineering.*

Ans. : The product in software engineering is a standalone entity that can be produced by development organization and sold on the open market to any customer who is able to buy them. The software product consists of computer programs, procedures, and associated documentation (*documentation can be in hard copy form or it may be in visual form*). Some of the examples of software product are databases, word processors, drawing tools.

The process in software engineering can be defined as the structured set of activities that are required to develop software system. Various activities under software process are –

- Specification
- Design and implementation
- Validation
- Evolution

Q. 11 *For each types of process models, identify the types of project suitable to implement.*

Ans. :

- **Waterfall Model** is useful for the projects in which the requirements are well understood and unlikely to change radically during the system development.
- **Incremental model** is used when within a small time span at least a core product needs to be delivered to the customer. The incremental model can be adopted when there are less number of people involved in the project and when a limited set of software functionality is needed quickly.
- **Rapid application development model** is useful for the component based projects which can be developed within extremely short time span (60 to 90 days)
- **Spiral model** is suitable for a projects in which large scale systems can be developed.
- **WIN WIN spiral model** is suitable for the large scale projects in which proper communication with customer can be possible.
- **Prototyping model** is useful for developing the systems efficiency of algorithm or adaptability of an operating system is not criteria and developer and customer together define the overall objective of the software. Certain classes of mathematical algorithms, subset of command driven systems can be developed using prototyping model.

Q. 12 *Discuss the major differences between software life cycle model and a process model.*

Ans. :

Software life cycle model	Process model
This model is based on common four activities : Analysis, design, code and testing.	This model is based on problem definition, technical development, software integration and existing status.
The software development process can be celarly and systematically defined in phases.	The software development process can not be clearly defined in phases.
Customer interaction is possible in every stage of software development process in this model.	Customer interaction is only at initial stage of software development.
Large scale projects can be handled using this approach.	Large scale projects may be caught in chaotic situation using this approach.

Review Questions

1. *What is the need of process model? What are different process models that are used commonly ?*
2. *What are the drawbacks of waterfall model?*
3. *Explain the incremental model.*
4. *Compare waterfall model and spiral model*
5. *Describe the rapid application development model*
6. *Describe the demerits of prototyping model.*
7. *Explain the unified process model. What are the work products that may get generated in this model?*

❑❑❑

CHAPTER 4 – AN AGILE VIEW OF PROCESS

2005 McGraw-Hill Higher Education
http://highered.mcgraw-hill.com/sites/0072853182/student_view0/chapter4/chapter_summary.html

Overview

- Agile software engineering represents a reasonable compromise between to conventional software engineering for certain classes of software and certain types of software projects
- Agile development processes can deliver successful systems quickly
- Agile development stresses continuous communication and collaboration among developers and customers
- Agile software engineering embraces a philosophy that encourages customer satisfaction, incremental software delivery, small project teams (composed of software engineers and stakeholders), informal methods, and minimal software engineering work products
- Agile software engineering guidelines stress on-time delivery of an operational software increment over analysis and design

Manifesto for Agile Software Development

- Proposes that it may be better to value:
    - Individuals and interactions over processes and tools
    - Working software over comprehensive documentation
    - Customer collaboration over contract negotiation
    - Responding to change over following a plan
- While the items on the right are still important the items on the left are more valuable under this philosophy
- Note: although most practitioners agree with this philosophy in theory, many pragmatic issues surface in the real world that may cause items on the right to be as important as items on the left

Agility

- An agile team is able to respond to changes during project development
- Agile development recognizes that project plans must be flexible
- Agility encourages team structures and attitudes that make communication among developers and customers more facile
- Agility eliminates the separation between customers and developers
- Agility emphasizes the importance of rapid delivery of operational software and de-emphasizes importance of intermediate work products
- Agility can be applied to any software process as long as the project team is allowed to streamline tasks and conduct planning in way that eliminate non-essential work products

Agile Processes

- Are based on three key assumptions
    1. It is difficult to predict in advance which requirements or customer priorities will change and which will not
    2. For many types of software design and construction activities are interleaved (construction is used to prove the design)
    3. Analysis, design, and testing are not as predictable from a planning perspective as one might like them to be
- Agile processes must be adapted incrementally to manage unpredictability
- Incremental adaptation requires customer feedback based on evaluation of delivered software increments (executable prototypes) over short time periods

Agility Principles

- Highest priority is to satisfy customer through early and continuous delivery of valuable software
- Welcome changing requirements even late in development, accommodating change is viewed as increasing the customer's competitive advantage
- Delivering working software frequently with a preference for shorter delivery schedules (e.g., every 2 or 3 weeks)
- Business people and developers must work together daily during the project
- Build projects around motivated individuals, given them the environment and support they need, trust them to get the job done
- Face-to-face communication is the most effective method of conveying information within the development team
- Working software is the primary measure of progress
- Agile processes support sustainable development, developers and customers should be able to continue development indefinitely
- Continuous attention to technical excellence and good design enhances agility
- Simplicity (defined as maximizing the work not done) is essential
- The best architectures, requirements, and design emerge from self-organizing teams
- At regular intervals teams reflects how to become more effective and adjusts its behavior accordingly

Human Factors

- Traits that need to exist in members of agile development teams:
  - Competence
  - Common focus
  - Collaboration
  - Decision-making ability
  - Fuzzy-problem solving ability
  - Mutual trust and respect
  - Self-organization

Agile Process Models

- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Scrum
- Crystal
- Feature Driven Development (FDD)
- Agile Modeling (AM)

Extreme Programming

- Relies on object-oriented approach
- Key activities
  - Planning (user stories created and ordered by customer value)
  - Design (simple designs preferred, CRC cards and design prototypes are only work products, encourages use of refactoring)
  - Coding (focuses on unit tests to exercise stories, emphasizes use of pairs programming to create story code, continuous integration and smoke testing is utilized)
  - Testing (unit tests created before coding are implemented using an automated testing framework to encourage use of regression testing, integration and validation testing done on daily basis, acceptance tests focus on system features and functions viewable by the customer)

Adaptive Software Development

- Self-organization arises when independent agents cooperate to create a solution to a problem that is beyond the capability of any individual agent
- Emphasizes self-organizing teams, interpersonal collaboration, and both individual and team learning
- Adaptive cycle characteristics
- Phases
    - Mission-driven
    - Component-based
    - Iterative
    - Time-boxed
    - Risk driven and change-tolerant
    - Speculation (project initiated and adaptive cycle planning takes place)
    - Collaboration (requires teamwork from a jelled team, joint application development is preferred requirements gathering approach, minispecs created)
    - Learning (components implemented and tested, focus groups provide feedback, formal technical reviews, postmortems)

Dynamic Systems Development Method

- Provides a framework for building and maintaining systems which meet tight time constraints using incremental prototyping in a controlled environment
- Uses Pareto principle (80% of project can be delivered in 20% required to deliver the entire project)
- Each increment only delivers enough functionality to move to the next increment
- Uses time boxes to fix time and resources to determine how much functionality will be delivered in each increment
- Guiding principles
    - Active user involvement
    - Teams empowered to make decisions
    - Fitness foe business purpose is criterion for deliverable acceptance
    - Iterative and incremental develop needed to converge on accurate business solution
    - All changes made during development are reversible
    - Requirements are baselined at a high level
    - Testing integrates throughout life-cycle
    - Collaborative and cooperative approach between stakeholders
- Life cycle activities
    - Feasibility study (establishes requirements and constraints)
    - Business study (establishes functional and information requirements needed to provide business value)
    - Functional model iteration (produces set of incremental prototypes to demonstrate functionality to customer)
    - Design and build iteration (revisits prototypes to ensure they provide business value for end users, may occur concurrently with functional model iteration)
    - Implementation (latest iteration placed in operational environment)

Scrum

- Scrum principles
    - Small working teamed used to maximize communication, minimize overhead, and maximize sharing of informal knowledge
    - Process must be adaptable to both technical and business challenges to ensure bets product produced
    - Process yields frequent increments that can be inspected, adjusted, tested, documented and built on
    - Development work and people performing it are partitioned into clean, low coupling partitions
    - Testing and documentation is performed as the product is built
    - Provides the ability to declare the product done whenever required

- Process patterns defining development activities
    - Backlog (prioritized list of requirements or features the provide business value to customer, items can be added at any time)
    - Sprints (work units required to achieve one of the backlog items, must fir into a predefined time-box, affected backlog items frozen)
    - Scrum meetings (15 minute daily meetings) addressing these questions: What was done since last meeting? What obstacles were encountered? What will be done by the next meeting?
    - Demos (deliver software increment to customer for evaluation)

Crystal

- Development approach that puts a premium on maneuverability during a resource-limited game of invention and communication with the primary goal of delivering useful software and a secondary goal of setting up for the next game
- Crystal principles
    - Its always cheaper and faster to communicate face-to-face
    - As methodologies become more formal teams become weighed down and have trouble adapting to project work vagaries
    - As projects grow in size, teams become larger and methodologies become heavier
    - As projects grow in criticality some degree of formality will need to be introduced in parts of the methodology
    - As feedback and communication become more efficient the need for intermediate work products is reduced
    - Discipline, skills, and understanding counter process, formality, and documentation
    - Team members not on the critical project path can spend their excess time improving the product or helping people who are on the critical path
- Incremental development strategy used with 1 to 3 month time lines
- Reflection workshops conducted before project begins, during increment development activity, and after increment is delivered
- Crystal methodologies
    - Clear (small, low criticality projects)
    - Orange (larger, moderately critical projects)
    - Orange Web (typical e-business applications)

Feature Driven Development

- Practical process model for object-oriented software engineering
- Feature is a client-valued function, can be implemented in two weeks or less
- FDD Philosophy
    - Emphasizes collaboration among team members
    - Manages problem and project complexity using feature-based decomposition followed integration of software increments
    - Technical communication using verbal, graphical, and textual means
    - Software quality encouraged by using incremental development, design and code inspections, SQA audits, metric collection, and use of patterns (analysis, design, construction)
- Framework activities
    - Develop overall model (contains set of classes depicting business model of application to be built)
    - Build features list (features extracted from domain model, features are categorized and prioritized, work is broken up into two week chunks)
    - Plan by feature (features assessed based on priority, effort, technical issues, schedule dependencies)
    - Design by feature (classes relevant to feature are chosen, class and method prologs are written, preliminary design detail developed, owner assigned to each class, owner responsible for maintaining design document for his or her own work packages)
    - Build by feature (class owner translates design into source code and performs

unit testing, integration performed by chief programmer)

Agile Modeling

- Practice-based methodology for effective modeling and documentation of software systems in a light-weight manner
- Modeling principles
    - Model with a purpose
    - Use multiple models
    - Travel light (only keep models with long-term value)
    - Content is more important than representation
    - Know the models and tools you use to create them
    - Adapt locally
- Requirements gathering and analysis modeling
    - Work collaboratively to find out what customer wants to do
    - Once requirements model is built collaborative analysis modeling continues with the customer
- Architectural modeling
    - Derives preliminary architecture from analysis model
    - Architectural model must be realistic for the environment and must be understandable by developers

AGILE MODEL	WATERFALL MODEL
Agile model is an incremental delivery process where each incremental delivered part is developed through an iteration after each time box.	The waterfall model is highly structured and systematically steps through requirements gathering, analysis, SRS document preparation, design, coding and testing in a planned manner. These phases of the Waterfall model follow a sequential order.
While using an agile model, progress is measured in terms of the developed and delivered functionalities.	In Waterfall model, progress is generally measured in terms of the number of completed and reviewed artifacts such as requirement specifications, design documents, test plans, code reviews, etc. for which review is complete.
Lack of proper formal documentation leaves ample scope confusion and important decisions taken during various phases can be misinterpreted at later phases.	In Waterfall model proper documentation is very important, which gives a clear idea what should be done to complete the project and it also serve as a agreement between the customer and development team.
Customer interaction is very high. After each iteration, an incremental version is deployed to the customer.	Customer interaction is very less. The product is delivered to the customer after the overall development is completed.

AGILE MODEL	RAD MODEL
The Agile model does not recommend developing prototypes but emphasizes the systematic development of each incremental feature at the end of each iteration.	The central theme of RAD is based on designing quick and dirty prototypes, which are then refined into production quality code.
The Agile team only demonstrate completed work to the customer after each iteration.	Whereas RAD teams demonstrate to customers screen mock up and prototypes, that may be based on simplifications such as table lookup rather than actual computations.
Agile model is not suitable for small projects as it is difficult to divide the project into small parts that can be incrementally developed.	When the company has not developed a almost similar type of project, then it is hard to use RAD model as it is unable to reuse the existing code.

AGILE MODEL	SPIRAL MODEL
The main principle of the Agile model is to achieve agility by removing unnecessary activities that waste time and effort.	The main principle of the Spiral model is risk handling.
The Agile model focuses on the delivery of an increment to the customer after each Time-box, so customer interaction is more frequent.	Spiral model mainly deals with various kinds of unanticipated risks but customer interaction is less.
Agile model is suitable for large projects that are easy to divide into small parts that can be easily developed incrementally over each iteration.	The Spiral model is suitable for those projects that are prone to various kinds of risks that are difficult to anticipate at the beginning of the project.
Agile model does not rely on documentation.	Proper documentation is required for Spiral model.