v-a-days the software becomes more sophisticated vell as complex because of various aspects such as ctacular improvements in hardware performance, nendous up-gradations in computing architectures, e enhancements in memory and storage capacity, extensive diversity of exotic input and output ions.

phistication as well as complexity leads to amazing ults on success of systems, but they may also lead to ge problems for those who want to develop complex stems.

the economies of the industrialized world, the big ftware industry is considered as a dominant factor.

an earlier era, software was developed individually. ow-a-days an entire team works for it in which every rson is responsible for one part of the technology hich is necessary to deliver a complex application.

.2.1 Defining Software

Define Term Software. **(2 Marks)**

oftware can be defined in different ways :

ition :

Software is set of instructions (computer programs) that when executed provide desired features, function, and performance.

Software is data structures that enable the programs to adequately manipulate information.

Software is descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.

Now to understand the software exactly we have to examine some characteristics of it which make it different from remaining human made things.

Software is considered as logical rather than a physical system element. Hence, there are characteristics of software which are significantly different than those of hardware.

1.2.1(A) Characteristics of Software

Explain characteristics of Software.
SPPU - Dec.19, 3 Marks

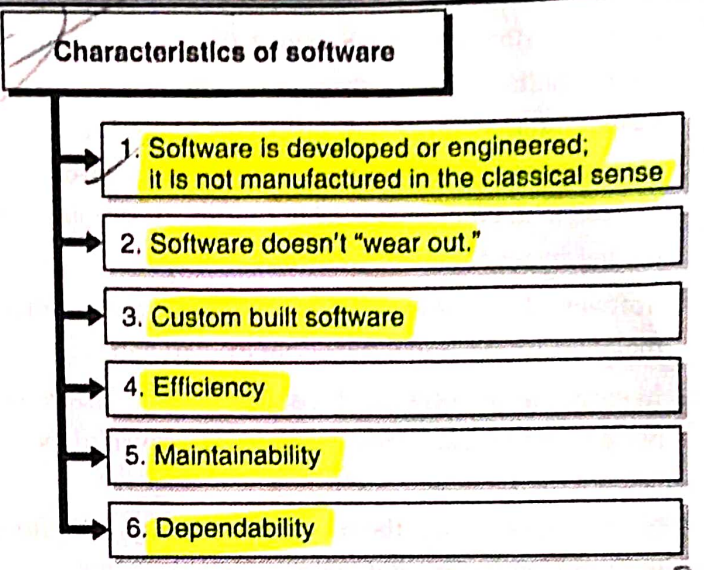


Fig. 1.2.1 : Characteristics of software

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

GQ. How software engineering is different from hardware engineering? Justify. **(3 Marks)**

- Even though there are some similarities found in software development and hardware manufacturing, both the activities are considered as fundamentally different.
- In both of the activities, good design is base for high quality, but in the process of manufacturing there may be quality problems in hardware which may not present in case of software.
- In both of the activities there is prominent dependency on people but there is vast difference in the relationship between people and work accomplished.
- The main aim of both the activities is construction of a "product", but the perspectives are not same.
- Software costs are concentrated in engineering which indicates that it is difficult to manage software projects similar to manufacturing projects.

2. Software doesn't "wear out"

GQ. "Software does not wear out". State whether this statement is true or false. Justify your answer. **(3 Marks)**

- In case of hardware, the failure rates are high as compared to software early in its life. Such failures are mainly concerned with design or manufacturing defects. It is possible to correct the defects and drop the failure rate to a steady-state level for some time period.

However after some time period, there is again increase in the failure rate as there are cumulative effects of dust, vibration, mishandling, tremendous high or low temperature, and number of other environmental maladies on components of hardware. In simple words, the hardware begins to *wear out.*

- Software does not have any risks of such environmental maladies which lead to wear out of hardware.
- In early life of software there may be high failure rates because of undiscovered defects. However, these can be corrected.
- Now it is clear that there is no possibility of software wear out but it does deteriorate.
- In it life change is an integral part of software. Whenever any change is made, there is possibility of introduction of errors which increase the failure rate.
- Before the software gets consistent state by corrections, any new change get introduced which again increases the failure rate.
- Gradually, the lowest failure rate level starts to increase which indicates that the software is deteriorating due to change.
- There is one important difference in hardware and software regarding wear aspect is that in case of hardware, a failed component can be replaced by spare parts. This facility is not available in software as there are no such spare parts.
- All the software failures point out that there is an error in design or in the method by which the respective design was transformed into machine executable code.
- Hence the tasks of software maintenance which handles requests for change has significantly more complexity as compared to hardware maintenance.

3. Custom built software

- Component reusability is an important aspect in software industry.
- It is responsibility of software engineer to design and implement a software component in such a way that it should be reused easily in many different programs.
- Latest reusable components summarize both data as well as the processing which is applied to the data, which helps the software engineer to develop new applications from existing components.
- For example, reusable components are used in the development of modern interactive user interfaces that enable the generation of graphics windows, pull-down menus, as well as wide range of interaction mechanisms.

4. Efficiency

Software is said to be efficient if it uses the available resources in the most efficient manner and produce desired result in timely manner.

5. Maintainability

- If customer requirements changes, programmers need to modify the software to fulfill those requirements.
- Software engineering provides the ability to maintain the software through modifying the software rather than changing whole product like hardware.

6. Dependability

- Dependability is the ability of the software that provides services which can be trusted by users.
- Dependability provides services to fulfill the customer's requirements, so that it is helpful to achieve trust of customers on the system.

1.2.2 Software Application Domain

UQ. Explain any 5 types of software along with an example **SPPU - May 18, 5 Marks**

- Virtually on all computer platforms, software can be grouped into few broad categories.

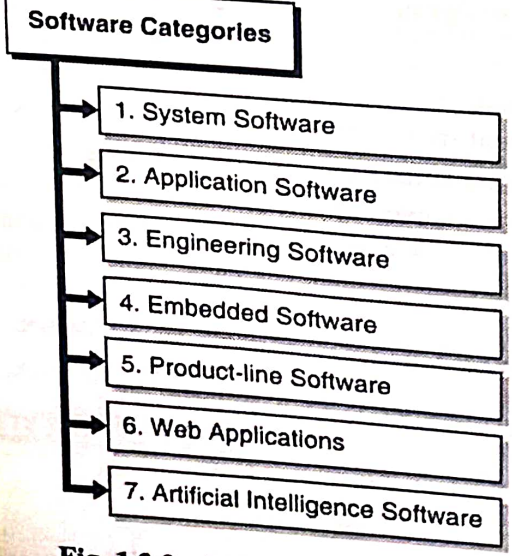


Fig. 1.2.2 : Software Categories

1. System software

- This is a set of programs used to provide service to other programs. Few of them such as compilers, editors, and file management utilities are used to process complex but determinate information structures.
- Some other system software such as components of OS, several drivers, networking related software, and telecommunications processors are generally used to process heavily indeterminate data.
- In all the situations, the system software area is broadly depends upon the interaction with computer hardware.

2. Application software

- These are the stand-alone programs which are specially developed for specific business needs.
- The Application Software helps to process business or technical data in such a way that it should be useful to manage business operations as well technical decision making.
- With traditional processing applications, one can also use application software to keep control on business functions in real time.

3. Engineering/scientific software

- These types of software are characterized through the "number crunching" algorithms. There are number of engineering software in various fields like astronomy, space shuttle orbital dynamics, automotive stress analysis, automated manufacturing, molecular biology etc.
- However, there are drastic changes in modern applications within the engineering/scientific area as they are shifting from traditional numerical algorithms.

4. Embedded software

These are the software which are merged within electronic devices and are used for the purpose of implementing and controlling the features as well as functions for the end user as well as system itself.

5. Product-line software

These software are developed to provide a particular functionality for use by several customers.

The focus of product-line software may be on a restricted marketplace like inventory control applications or can focus on mass consumer markets such as word processing, spreadsheets, graphics based applications, multimedia, database management, and personal as well as business financial products.

6. Web applications

- These applications are also called as "WebApps". This is a network-centric software category which covers large number of applications.
- In shortest form, WebApps are same as of the group of linked hypertext files which provide information through text and limited graphics.
- Now-a-days the modern WebApps are evolving into environments of sophisticated computing which offer stand-alone features, computing functions, as well as content to the end user.
- It is also possible to integrate WebApps with corporate databases and business applications.

7. Artificial intelligence software

- In these applications non-numerical algorithms are generally used for the purpose of solving complex problems which cannot be handled by computation or straightforward analysis.
- There are various applications in this category such as robotics, games, expert systems, artificial neural networks, proving theorem, pattern matching like image and voice.

☞ Difference between Hardware and software

UQ. What is the difference between hardware and software? **SPPU - Aug.17, 4 Marks**

Parameter	Hardware	Software
Concept	Hardare is a physical part computer that cause processing of data.	Software is a set of instructions that tells a computer exactly what to do.
Developed or manufactures	It is manufactured.	It is developed and engineered.
Interdependence	Hardware cannot perform any task without software.	Software cannot be executed without hardware.
Can see or not	As Hardware is physical electronic device, we can see and touch hardware.	We can see and also use the software but can't actually touch them.

Parameter	Hardware	Software
Categories	It has four main categories: input device, output devices, storage, and internal components.	It is mainly divided into System software, Programming software and Application software.
Virus effect	Hardware is not affected by computer viruses.	Software is affected by computer viruses.
Destruction	If hardware is damaged, it is replaced with new one.	If software is damaged, its backup copy can be reinstalled.
Examples	Ex: Keyboard, Mouse, Monitor, Printer, CPU, Hard disk, RAM, ROM etc.	Ex: Ms Word, Excel, Power Point, Photoshop, MySQL etc.

1.3 SOFTWARE ENGINEERING PRACTICE

UQ. What is Software Engineering ? **SPPU - Dec.19, 3 Marks**

Software Engineering combines two concepts, Software and Engineering.

Software

- As we have already seen Software is set of executed programs related to specific functionality. Set of executable instructions is called as a program.
- Software is more than just a program code; it also includes data structures which allow programs to manipulate information, and documentation related to software product which defines the functionality and guidance for the user to use this software. Software engineers design and build the software product.

Engineering

- Engineering is an application of knowledge and well defined principles to develop products.

Software Engineering

Q. Define Software Engineering. **(2 Marks)**

Definition 1 : Software Engineering *is the method of applying* scientific and technological knowledge, procedures, and rules to design, develop, and maintain the software product.

OR

Definition 2 : Applying technological, scientific, and administrative approach to designing, developing, testing and maintaining the software product in order to meet customers' requirements with best quality of product referred as software engineering.

1.3.1 Layered Approach

GQ. Elaborate how software engineering is layered technology **SPPU - Dec.18, 5 Marks**

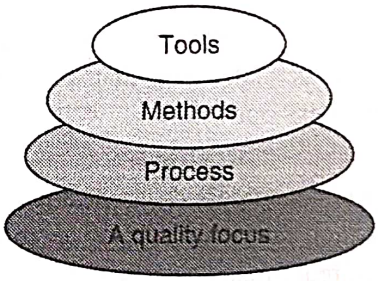


Fig. 1.3.1 : Layered Approach of software Technology

This approach is divided into 4 layers :

1. **A quality focus**
   - Any engineering approach must rest on the quality.
   - The most important aspect in software Engineering is Quality Focus.
2. **Process**
   - Foundation for SE is the Process Layer.
   - SE process is the GLUE that holds all the technology layers together and enables the timely development of computer software.
   - It forms the base for management control of software project.
3. **Methods**
   - SE methods provide the "Technical Questions" for building Software.
   - Methods contain a broad array of tasks that include communication requirement analysis, design modeling, program construction testing and support.

4. **Tools**
   - SE tools provide automated or semi-automated support for the "Process" and the "Methods".
   - Tools are integrated so that information created by one tool can be used by another.

1.3.2 Activities of Software Engineering

GQ. Enlist Activities of Software Engineering. (2 Marks)

Software Engineering includes many activities as follows :

- Understanding the user requirements is the first most important activity in software engineering.
- According to the requirements, the process of designing the system and taking decisions is implemented, which further leads to successful completion of development of product.
- Constructing the software product based upon designs and decisions made earlier is the next activity in software engineering.
- To verify the performance and quality of software, testing the software product is essential after it is constructed by software engineers.
- The next step is to provide maintenance for software product which is deployed to the customer.

1.3.3 Need of Software Engineering

GQ. Explain need of Software Engineering. (2 Marks)

- As the technology changes, the user requirements and environment on which software is working also changes. So every organization is ranked based on the software engineering principles used by that organization.
- Implementing and managing large size of software, programmer requires a specific method to modularize the tasks so that size of software can't harm the software quality. Software engineering provides methodology for implementing complex software systems with high quality.
- Without any standard method or management, it is difficult to address defects in the product and correct them as early as possible. Software Engineering provides this functionality.
- Extending the previous software to add new functionality requires more cost in terms of time to develop and efforts taken by people, as compare to the process of developing new software to provide that functionality. Software engineering provides a way in which software system can be able to scale as needed in future.

1.4 SOFTWARE ENGINEERING PRINCIPLES

GQ. What are the core principles of software engineering ? Explain. (5 Marks)

- Software engineering is considered as guided by a set of core principles that help in the application of a significant software process and the implementation of efficient software engineering methods.
- At the process level, the important use of core principle is to establish a philosophical foundation which guides members of a software team as they perform framework and umbrella activities, navigates the process flow, and generates a collection of software engineering related work products.
- At the practice level, core principles create a set of values and rules which serve as a guide when there is analysis of a problem, designing of a solution, implementing and testing of the solution, and finally deployment of the software in the user community.
- There is a set of general principles which span software engineering process and practice. Those principles are as follows :
  (a) Provide value to end users,
  (b) Keep it simple,
  (c) Maintain the vision (of the product and the project),
  (d) Recognize that others consume (and must understand) what you produce,
  (e) Be open to the future,
  (f) Plan ahead for reuse, and
  (g) Think
- Although these are important general principles, their characterization is done at such a high level of abstraction that in some situations, it becomes complicated to implement them in day-to-day software engineering practice.
- In the next sections we are going to see core principles in more detail which guide process and practice.

1.4.1 Principles that Guide Process

GQ. What are the principles which guide process? **(5 Marks)**

- Up till now we have seen the significance of the several process models which are used in software engineering work.
- In spite of whether a model is linear or iterative, prescriptive or agile, it is possible to characterize it with the help of generic process framework which is applicable for all process models.
- Here is a set of core principles which can be applied to the framework, and by extension, to any of the software process.

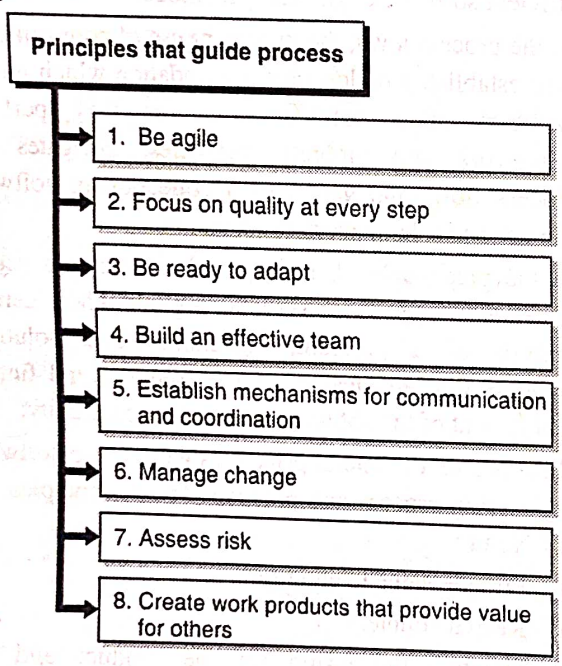


Fig. 1.4.1 : Principles that guide process

Principle 1 : Be agile

- Whether the selected process model is prescriptive or agile, the basic view of agile development must be able to govern the approach.
- Each and every part of the work done should highlight economy of action - keep the technical approach as maximum easy, keep the generated products as concise as possible, and try to make decisions locally whenever it is possible.

Principle 2 : Focus on quality at every step

The exit condition regarding all the process activities, actions, and tasks must concentrate on the quality of the work of generated product.

Principle 3 : Be ready to adapt

Process cannot be considered as a religious experience, and belief has no place in it. Whenever there is need, adapt the approach to constraints imposed by the problem, the people, and the project itself.

Principle 4 : Build an effective team

The Software engineering process and practice are definitely important, but the most important thing is people. It is important to establish a self-organizing team which has mutual trust and respect.

Principle 5 : Establish mechanisms for communication and coordination

- The reason of project failure is that the critical information falls into the cracks and/or stakeholders are not able to coordinate their efforts to produce a successful end product.
- These are considered as management level concerns and must be addressed.

Principle 6 : Manage change

The approach sometimes is formal while sometimes informal, but there is need of mechanisms to handle the way changes are requested, assessed, approved, and implemented.

Principle 7 : Assess risk

There is possibility of concerns in several things as there is progress in software development. It's important that you set contingency plans.

Principle 8 : Create work products that provide value for others

- Create only such types of work products which offer value for other process activities, actions, or tasks. Each and every produced work product which is a part of software engineering practice will be passed on to others.
- A list of necessary functions as well as features will be provided to the individual who will develop a design, the design will be provided to the individual who generate code, and so on.

1.4.2 Principles That Guide Practice

GQ. What are the principles which guide practice? (5 Marks)

- The important aim of Software engineering practice is to deliver on-time, high quality, operational software that consists of functions and features which meet the requirements of all the stakeholders.
- To make it possible, we should adopt a set of core principles which are able to guide the technical work. There are merits in these principles despite of the applied analysis and design methods, the used construction techniques (e.g., programming language, automated tools), or the selected verification and validation approach.
- Here is a set of core principles which are fundamental to the practice of software engineering :

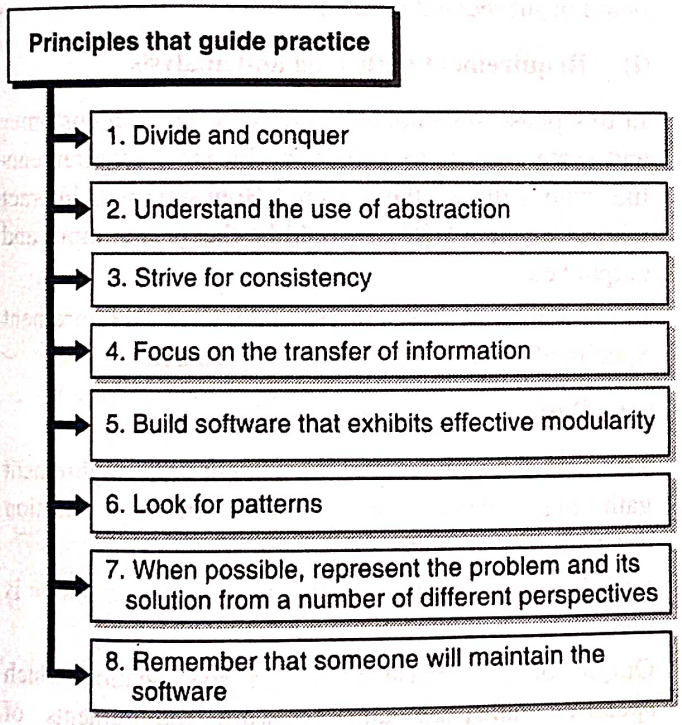


Fig. 1.4.2 : Principles that guide practice

Principle 1: Divide and conquer

- In a more technical way, analysis and design should always emphasize Separation of Concerns (SoC).
- To solve a large problem easily, it is better to subdivide it into a set of elements (or concerns).
- Ideally, each and every concern provides distinct functionality which can be developed, and sometimes validated, separately of other concerns.

Principle 2 : Understand the use of abstraction

- An abstraction is considered as a simplification of any complex element of a system which helps to communicate meaning in a single phrase.
- In software engineering practice, several levels of abstraction are used, all of which imparting or implying meaning that must be communicated.

Principle 3 : Strive for consistency

- Whether it's making of a requirements model, building a software design, creating source code, or generating test cases, the principle of consistency imply that a familiar context makes software easier to use.
- E.g. think a design of a user interface for a WebApp. Consistent position of menu options, consistent color scheme use, and the consistent use of recognizable icons; all these elements make the interface very sound.

Principle 4 : Focus on the transfer of information

- Software is regarding data transfer-from a database to an end user, from a legacy system to a WebApp, from an end user into a graphical user interface (GUI), from an operating system to an application, from one software component to another - the list is almost endless.
- In all these cases, information flows across an interface, and as a result, there is possibility of error, or omission, or ambiguity.
- This principle implies that one must give special attention to the analysis, design, construction, and testing of interfaces.

Principle 5 : Build software that exhibits effective modularity

- Separation of concerns (Principle 1) describes a philosophy for software. Modularity gives a mechanism for realizing the particular philosophy.
- It is possible to divide any complex system into modules (components), but there are more expectations from good software engineering practice.
- Modularity must be effective. That means, each and every module should concentrate solely on one well-constrained aspect of the system.
- Also, the connections between modules should be relatively simple - each module must show low coupling with other modules, to data sources, and to other environmental aspects.

Principle 6 : Look for patterns

- The aim of patterns within the software community is to generate a structure of literature to facilitate software developers resolve recurring problems occurred in the entire process of software development.
- Patterns help to generate a shared language for the purpose of communicating insight and experience regarding these problems with their solutions.
- Formally codifying these solutions and their relationships allows us to effectively confine the body of knowledge which sets our understanding of good architectures which satisfy the user requirements.

Principle 7 : When possible, represent the problem and its solution from a number of different perspectives

- When a problem with its solution is inspected from several perspectives, there is possibility of better insight and the errors and omissions will be uncovered.
- For example, data oriented viewpoint can be used to represent a requirements model.
- Data oriented viewpoint is a function-oriented viewpoint, or a behavioural viewpoint.
- Each provides a unique view of the problem with its requirements.

Principle 8 : Remember that someone will maintain the software

- Over the long period of time, software will be corrected as issues are resolved, adapted as its environment changes, and improved as stakeholders expect more capabilities.
- To facilitate these maintenance activities, there is need of solid software engineering practice throughout the software process.

1.5 SOFTWARE PROCESS

GQ. Explain in brief the software process. **(4 Marks)**

GQ. Explain classic life cycle paradigm for software engineering and problems encountered when it is applied. **(5 Marks)**

- A software engineering process is the model selected for managing the creation of software from customer initiation to the release of the finished product. The selected process typically involves methods such as

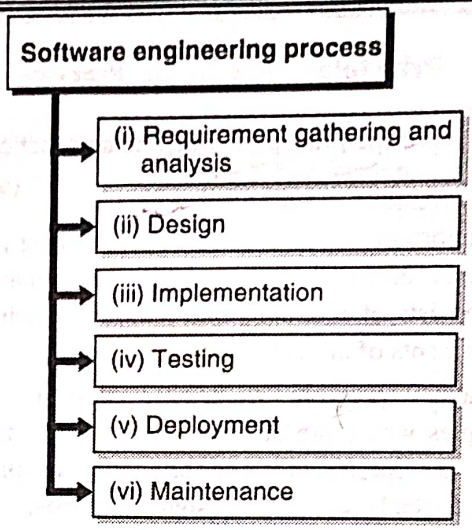


Fig. 1.5.1 : Software engineering process

- The **software life cycle** follows the sequential flow in order to develop software.
- We will see each and every stage of this life cycle in detail in subsequent points :

(i) Requirement gathering and analysis

- In this phase stakeholders communicate with customer and system users to gather the business requirements like who will use the system? How user will interact with the system? What should be the system input and output? etc.
- Depending on these requirements, Requirement Specification Document (SRS) is prepared.

(ii) Design

- This phase makes use of output of requirement gathering phase i.e. requirement specification document as an input.
- Based on requirement specifications software design is prepared.
- Output of this phase is design specification which specifies hardware and software requirements of system.
- Data Flow Diagram, flowcharts etc. are included by design specification document.
- Design specification acts as an input for implementation phase.

(iii) Implementation

- Implementation phase of development process decomposes the system work into various smaller parts called **modules**.

- These modules are assigned to development team members and actual coding is started.
- This is longest phase of system development.
- Output of this phase is actual code using specific programming language.

(iv) Testing

- Testing phase involves testing of actual code of system against requirements of user in order to ensure that system will satisfy all needs of users.
- Various testing strategies used are unit testing, system testing, integration testing etc.
- Output of testing phase is corrected and modified software code.

(v) Deployment

- In this phase the system is deployed at users' site for their use.
- In this phase customer uses the software and give feedback to development team for any changes or modifications in system if any.

(vi) Maintenance

- Once the software is deployed actual problems from user's perspective will come up. It is responsibility of development team to solve user's problems time to time. This process is called as maintenance of system.
- In this phase some additional functionality may need to add in the application as per user's requirement.

1.5.1 Software Process Framework

UQ. Explain Software Process Framework **SPPU - Dec.17, 5 Marks**

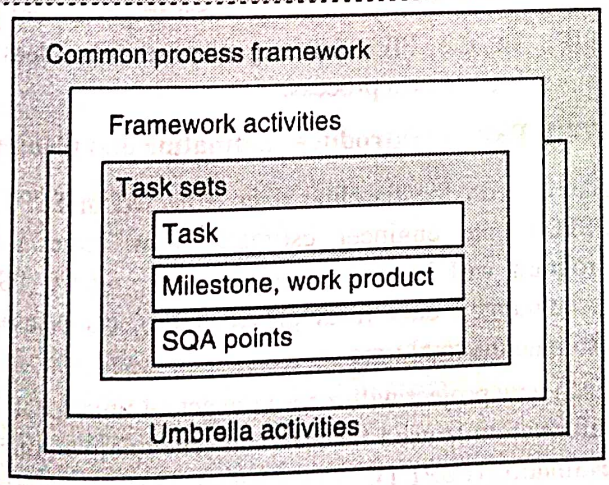


Fig. 1.5.2 : Software process framework

- The process of framework consists if several activities which are applicable to all types of projects.
- The software process framework is considered as a group of various types of task sets.
- In the task sets there is collection of small work tasks, project milestones, work productivity as well as software quality assurance points.

1.5.1(A) Umbrella Activities

GQ. Explain different activities in software process framework. **(5 Marks)**

Typical umbrella activities are :

1. Software project tracking and control

- This activity consists of accessing the project plan and comparing it with the predefined schedule. This is done by the development team.
- If there is no match in project plans and the predefined schedule, then the necessary measures are taken to maintain the schedule.

2. Risk management

- Risk is considered as an event which may or may not occur. If the event occurs, then it may leads to some unwanted outcome. Hence, there is need of proper risk management.

3. Software Quality Assurance (SQA)

- Software Quality Assurance is the planned as well as systematic pattern of activities which are necessary to provide an assurance of software quality.
- **For example,** in the process of software development, several meetings are arranged at each and every stage to find out the defects and imply improvements for the purpose of producing good quality software.

4. Formal Technical Reviews (FTR)

- FTR is a meeting conducted by the technical staff.
- The main aim of such meeting is to detect problems regarding quality and suggest improvements.
- The technical person takes customer point of view on consideration to maintain the quality of the software.

5. Measurement

- Measurement involves the effort needed to measure the software.
- It is difficult to measure the software straightforwardly. Direct and indirect measures are used to measure it.

- Direct measures consist of cost, lines of code, size of software etc.
- Indirect measures like quality of software are measured by some different factors. Hence, it is considered as an indirect measure of software.

6. Software Configuration Management (SCM)

It is used to handle the effect of change during the process of software development.

7. Reusability management

- It involves describing of the criteria regarding reuse of the product.
- The quality of software can be considered as good when it is found that the components of the software which are developed for any specific application are also seem to be useful for developing other types of applications.

8. Work product preparation and production

It involves the activities which are necessary to generate the documents, forms, lists, logs and user manuals for the purpose of developing software.

☞ Framework Activities

- The Framework activities which we have already seen are as follows :
  - Requirement gathering and analysis Design
  - Implementation
  - Testing
  - Deployment
  - Maintenance

1.5.2 Personal Software Process (PSP)

- The Personal Software Process (PSP) is a structured software development process that is intended (planned) to help software engineers better understand and improve their performance by bringing discipline to the way they develop software and tracking their predicted and actual development of the code.
- It clearly shows developers how to manage the quality of their products, how to make a sound plan, and how to make commitments.
- It also offers them the data to justify their plans. They can evaluate their work and suggest improvement direction by analyzing and reviewing development time, defects, and size data.
- The PSP was created by Watts Humphrey to apply the underlying principles of the Software Engineering Institute's (SEI) Capability Maturity Model (CMM) to the software development practices of a single developer.
- It claims to give software engineers the process skills necessary to work on a team software process (TSP).

☞ Objective of PSP

GQ. What is objective of Personal Software Process(PSP) ? **(5 Marks)**

- The PSP aims to provide software engineers with disciplined methods for improving personal software development processes. The PSP helps software engineers to :
  1. Improve their estimating and planning skills.
  2. Make commitments they can keep.
  3. Manage the quality of their projects.
  4. Reduce the number of defects in their work

☞ Activities / Phases of PSP

GQ. What are the activities of PSP Model ? **(5 Marks)**

1. PSP0, PSP0.1 (Introduces process discipline and measurement)

- PSP0 has 3 phases - planning, development (design, code, compile, test) and a post mortem. A baseline is established of current process measuring - time spent on programming, faults injected/removed, size of a program. In a post mortem, the engineer ensures all data for the projects has been properly recorded and analysed. PSP0.1 advances the process by adding a coding standard, a size measurement and the development of a personal process improvement plan (PIP). In the PIP, the engineer records ideas for improving his own process.

2. PSP1, PSP1.1 (Introduces estimating and planning)

- Based upon the baseline data collected in PSP0 and PSP0.1, the engineer estimates how large a new program will be and prepares a test report (PSP1). Accumulated data from previous projects is used to estimate the total time.
- Each new project will record the actual time spent. This information is used for task and schedule planning and estimation (PSP1.1).

3. **PSP2, PSP2.1 (Introduces quality management and design)**

- PSP2 adds two new phases: design review and code review. Defect prevention and removal of them are the focus at the PSP2.
- Engineers learn to evaluate and improve their process by measuring how long tasks take and the number of defects they inject and remove in each phase of development. Engineers construct and use checklists for design and code reviews. PSP2.1 introduces design specification and analysis techniques.

1.5.3 Comparison with Conventional Engineering Process

GQ. Compare Software engineering with Conventional Engineering Process. **(5 Marks)**

- Software engineering has some similar things as the principles of conventional engineering.
- It is assumed that software engineering is associated to the design of software or data processing products and belongs to its problem solving domain, including the class of problems related to software and data processing.
- This idea is expanded by drawing similarity from the methods that are generally used in engineering.
- It is observed that, just as the famous scientific method is used in the field of scientific research, the steps of engineering design process are used in the process of problem solving in the field of engineering.
- Following steps are mostly repetitive :
  1. Problem formulation,
  2. Problem analysis,
  3. Search for alternatives,
  4. Decision,
  5. Specification,
  6. Implementation
- It is advised that these steps are applicable to the field of software engineering as well.
- Software engineering is considered as a result of integration of hardware and systems engineering, including a set of 3 key elements - methods, tools and procedures which facilitate the manager to control the process of software development.
- The methods give the technical "how to" for building software; tools offer automated or semi-automated support for methods; and procedures describe the series of applying methods, the deliverables, the controls, and the objectives.
- Compared to traditional engineering disciplines, software engineering demonstrates some remarkable differences.
- In conventional engineering, one moves from an abstract design to a concrete product. On other hand, in software engineering, one moves from design to coding.

Software Engineering	Abstract Design	→	More Abstract Code
Manufacturing Engineering	Abstract Design	→	Concrete Products

- The problem domains of software engineering can be anything, from word processing to real-time control and from games to robotics.
- In contrast to another engineering discipline, it is therefore much larger in scope and thus provides bigger challenges.
- Traditional manufacturing engineering that normally emphasize mass production is loaded with production features. Thus, it is highly production concentrated. Software engineering, in contrast, is inherently design concentrated.
- Product standardization assists in cost reduction in manufacturing, whereas such a possibility is remote in software engineering.
- The possibility of process standardization, however, is very high in the latter.
- Conventional engineering process makes the use of unlimited number of domain and application-specific ideas which are proved. Software engineering, in contrast, makes use of limited, but global concepts such as loops, conditions etc.

1.6 SOFTWARE MYTHS

UQ. State and explain any 3 software engineering myths and reality **SPPU - Oct 19, 6 Marks**

UQ. State and explain any Five Myths of software development with reality. **SPPU - May 19, 5 Marks**

All people who come into contact with software may suffer from various myths associated with developing and using software. Here are a few common ones.

☞ Management myths

- **Our company has books full of standards, procedures, protocol, and so on, related to programming software. This provides everything that our programmers and managers need to know.** While company standards may exist, one must ask if the standards are complete, reflect modern software practice, and are importantly actually used.
- **If we fall behind schedule in developing software, we can just put more people on it.** If software is late, adding more people will merely make the problem worse. This is because the people already working on the project now need to spend time educating the newcomers, and are thus taken away from their work.
- The newcomers are also far less productive than the existing software engineers, and so the work put into training them to work on the software does not immediately meet with an appropriate reduction in work.

☞ Customer / end-user myths

UQ. Discuss the software myths and realities in customer perspective
SPPU - May 18, Dec.19, 5 Marks

- **A vague collection of software objectives is all that is required to begin programming. Further details can be added later.** If the goals / objectives for a piece of software are vague enough to become ambiguous, then the software will almost certainly not do what the customer requires.
- **Changing requirements can be easily taken care of because software** is so flexible. This is not true: the longer that development on the software has proceeded for, the more work is required to incorporate any changes to the software requirements.

☞ Programmer / Developer / Practitioner myths

UQ. Discuss software myths and realities in developer perspective. **SPPU - Dec.18, 4 Marks**

- **Once the software is written, and works, our job is done.** A large portion of software engineering occurs after the customer has the software, since bugs will be discovered, missing requirements uncovered, and so on.
- **The only deliverable for a project is the working program.** At the very least there should also be documentation, which provides support to both the software maintainers, and to the end-users.
- **Software engineering will make us create a lot of unnecessary documentation, and will slow us down.** Software engineering is not about producing documents. Software engineering increases the quality of the software. Better quality reduces work load and speeds up software delivery times.

⏭ 1.7 PROCESS MODELS : A GENERIC PROCESS MODEL

UQ. Explain the generic process model of software development with the diagram.
SPPU - Aug.17, 5 Marks

- In section 1.5 we have already seen the important phases of a software process which can also be termed as Software Development Life Cycle.
- Now we are going to discuss all these phases in detail for a generic process model with some important aspects.

☞ System Engineering

- As software approximately always makes the modules of a larger system, one can starts work by forming requirements for all components of the system, recognizing not only the role performed by the software but, more importantly, its interface and interaction with the outside world.
- This 'system view' is necessary when software must interface with other elements such as hardware, people, databases and computers outside of, and ahead of the control of the system designer.
- In essence Systems engineering contains discovering some of the following issues :
  (i) Where does the software solution fit into the overall picture?
  (ii) What does the system do? Is it necessary to examine some process or activity or is it simply performing analysis of data?
  (iii) What environment will the system be placed in? That is system is on-line, real-time, embedded etc.
  (iv) What user interface is required and how is it used?
  (v) How systems communicate with other computers?
  (vi) Does the system provide the required performance ?

(vii) What time period has been planned and how serious it is. What budget does the customer have and is it realistic? What are the cost/benefits tradeoffs to the user in automating some manual procedure?

- When answers of these questions are received, the developer is in a good position to evaluate the RISK involved in implementing the system.

(1) Requirements Gathering and Analysis

- Requirements Analysis is the first important step towards creating a specification and a design.
- Trying to design a solution to a problem without totally understanding the nature and requirements of the user will always fails down the solution.
- It has been shown that more than ninety percent of software that are rejected by the customers after delivery is because the software does not meet the customer needs, expectation or requirements so it is important to understand those requirements completely.
- Requirements analysis is an iterative process carried out together by an analyst and the customer and represents an attempt, to discover the customer's needs, even as at the same time assessing the viability of a software solution.
- Analysis gives the designer the demonstration of the information which is processed by the system and the nature of the processing. i.e., what does the system do with the information which it processes?
- Analysis enables the designer to identify the software's function and performance. Also where the customer is not sure about how the system will eventually behave; the analyst may discover the concept of a prototype.

(2) Analysis Summary

- The aim of requirements analysis is to generate a "requirements specification document" or a "target document" that illustrates great extent detail in an unambiguous a manner regarding exactly what the product should do. This requirements document will then form the basis for the succeeding design phase.
- The requirements document may consists of :
  - A Project plan containing details of delivery times and cost estimates.
  - A model of the software's functionality and behavior in the form of 'data flow/UML diagrams' and, where suitable, any performance and data considerations, any input/output details etc.
  - The results of any prototyping so that the emergence of the software and how it should behave can be exposed to the designer. This might contain a user's manual.
  - Any formal Z or VDM specifications for the system requirements.
  - Any test data that may be used to find out whether the end product is build according to the agreed specification or not.
  - The Fig. 1.7.1 illustrates the activities that are sum-up by the analysis.

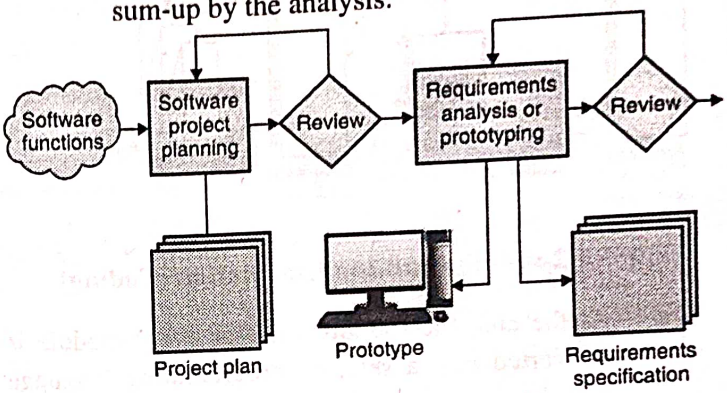


Fig. 1.7.1 : Activities in Analysis

(3) Design and Implementation (Coding)

- When the analysis of the system has been completed, design or development can start. This is an attempt to convert a group of requirements and program/data models that are specified in the "requirements document" into a well-organized, designed and engineering software solution.
- Design can be best summarized with the help of following steps :
  - The data flow/UML (Unified Modeling Language) diagrams that signify the system model are transformed into appropriate hierarchical, modular program and data structure/architecture.
  - Every program module is transformed into a correct cohesive function subroutine or class that is designed to do an individual task which is well-defined.
  - Design further focus on the implementation of individual module/class. The user interfaces of module and its communication with other modules/objects is also considered and evaluated for good design.
  - The algorithm of module can afterwards converted into a flowchart, which is a step-by-step graphical

representation of the actions which are held by the module demonstrated in terms of sequence, selection and repetition.

- The flowchart can then be converted into Pseudo code, which conveys the same information as a flowchart, but presents it in a way that is more acceptable to translate into program code.

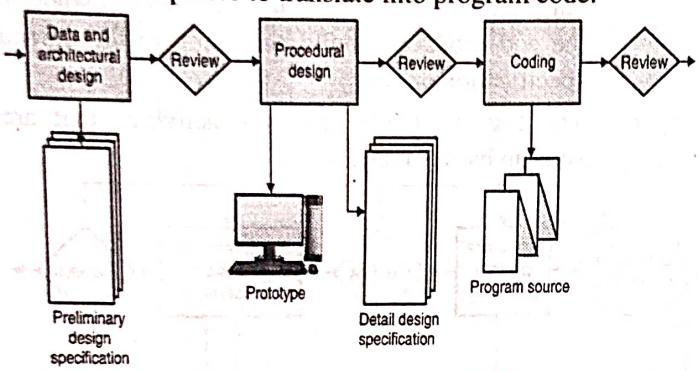


Fig. 1.7.2 : Design and Implementation (Coding)

- At the end, the Pseudo code for every module is converted into a selected programming language and the number of modules entered are compiled and integrated into a system which is ready for testing.
- The final result of design and coding contains the architecture, data structures, flowcharts, Pseudo code, algorithms and all program source code listings, as depicted as shown in Fig. 1.7.2.

(4) Software Testing

- When the part of the software components/modules has been written, testing and debugging can be started.
- Following techniques are involved in testing :

(i) Verification and Validation : This technique is useful for checking that the software meets the agreed specification and verifying that the software is correct in its operation.

(ii) Black and white box testing techniques : It is helpful for testing the insides of the modules for correct operation and testing the interfaces of the module.

(iii) Integration Testing : Testing that the modules all working together properly.

(iv) Acceptance Testing : Allowing the customer to test the product.

- Debugging can be defined as it is an art of recognizing the cause of failure in a part of software and correcting it.

(5) Deployment

After completion of software development, we have to install it at client site. This process is known as deployment.

(6) Software Maintenance

Software maintenance reapplies all of the previous life cycle steps to an existing program instead of a new one in order to correct existing or insert new functionality.

1.8 LINEAR SEQUENTIAL DEVELOPMENT MODEL / WATERFALL MODEL

GQ. What is waterfall model ? State the practical situations in which it can be used. **(5 Marks)**

GQ. Explain the waterfall model. **(5 Marks)**

- **Waterfall model** is the first approach used in software development process.
- It is also called as classical life cycle model or **linear sequential model.**
- In waterfall model any phase of development process begins only if previous phase is completed.

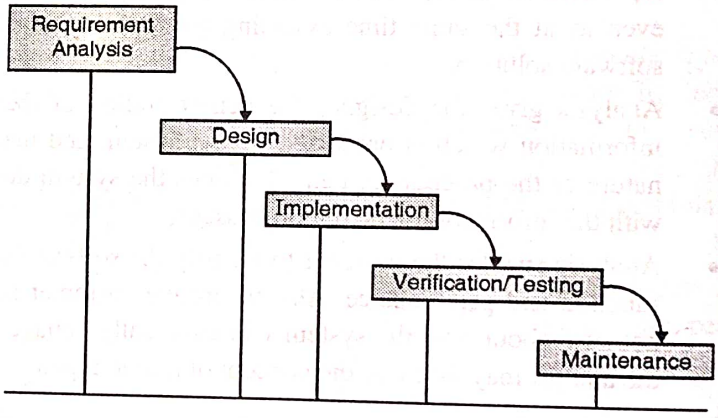


Fig. 1.8.1 : Waterfall Model

1. Requirement Analysis

- In this phase all business requirements of system are gathered and analyzed by communication between stakeholders and customer or user.
- At the end of this phase Requirement Specification Document (SRS) is created.

2. Design

- Based on requirement specification document, design of system is created called software architecture.

- It is blue print of system representing system's internal structure and behaviour.

3. Implementation

- In implementation phase, actual coding is constructed for software architecture using hardware and software requirements of system.
- It is responsibility of developer.

4. Verification / Testing

- Here coding or job done by developer is verified against requirements of user in order to ensure that software will satisfy all business requirements of user.
- After the successful verification, software is deployed at user's site for their use.

5. Maintenance

- While using software if user faces some problems, then those problems must be solved time to time by development team. This task comes under maintenance of software.
- Maintenance also includes adding new functionalities in software as per user requirements.

☞ Advantages of waterfall model

(i) It is very simple to understand and easy to use.

(ii) Phases of waterfall model do not overlap with each other.

(iii) It is useful for small projects in which requirements are clear at the beginning.

(iv) Since development is linear it is easy to manage development process.

☞ Disadvantages of waterfall model

UQ. Why waterfall model of the software engineering is not an accurate reflection of software development activities? Justify. **SPPU - Oct. 19, 5 Marks**

Following are the reasons because of which waterfall model of the software engineering is not considered an accurate reflection of software development activities :

(i) It is not useful for large projects.

(ii) Not suitable for projects in which requirements are not clear initially.

(iii) System or product is available only at the end of development process.

(iv) It is very difficult to modify system requirements in the middle of development process.

☞ Practical situations in which Waterfall model can be used

- This model is used only when the requirements are very well known, clear and fixed.
- Product definition is stable.
- Technology is understood.
- There are no ambiguous requirements.
- Ample resources with required expertise are available freely.
- The project is short.

⏭ 1.9 ITERATIVE DEVELOPMENT MODEL

UQ. Explain incremental model in detail. **SPPU - Dec. 18, 5 Marks**

GQ. Discuss benefits of Incremental process model. **(5 Marks)**

- It can also be called as *Incremental Process Model*. In iterative enhancement, extensions and design modifications in the project can be made in increments i.e. at each step.
- The *iterative enhancement life cycle model* removes the limitations of the waterfall model.
- The incremental Process model combines elements of waterfall models applied in iterative fashion.
  1. It adapts all the phases of waterfall model.
  2. It accepts linear process.
  3. It tries to solve one by one problems of the customer, or it accomplishes one by one requirement of the user, which is called as incremental iterations.
  4. Generally first increment is nothing but core product of the customer.
  5. As time grows, it is incremented from one requirement to next requirement or simultaneously it performs both the requirements (i.e. old requirement as well as starts new requirements)

☞ Advantages of Iterative Enhancement

1. It can result in better testing, since testing each increment is likely to be easier than testing entire system like in the waterfall model.

2. As in prototyping, the increment provides feedback to the client, which is useful for determining the final requirements of the system.

☞ **Iterative Enhancement model includes two process models**

1. Incremental Model
2. Rapid Application Development model (RAD)

1.9.1 The Incremental Development Model

GQ. Explain with neat diagram Increment Model and state its advantages and disadvantages. **(5 Marks)**

- The incremental model applies the waterfall model incrementally.
- The series of releases is referred to as "increments", with each increment providing more functionality to the customers.
- After the first increment, a core product is delivered, which can already be used by the customer.
- Based on customer feedback, a plan is developed for the next increments, and modifications are made accordingly.
- This process continues with increments being delivered until the complete product is delivered. The incremental philosophy is also used in the agile process model.

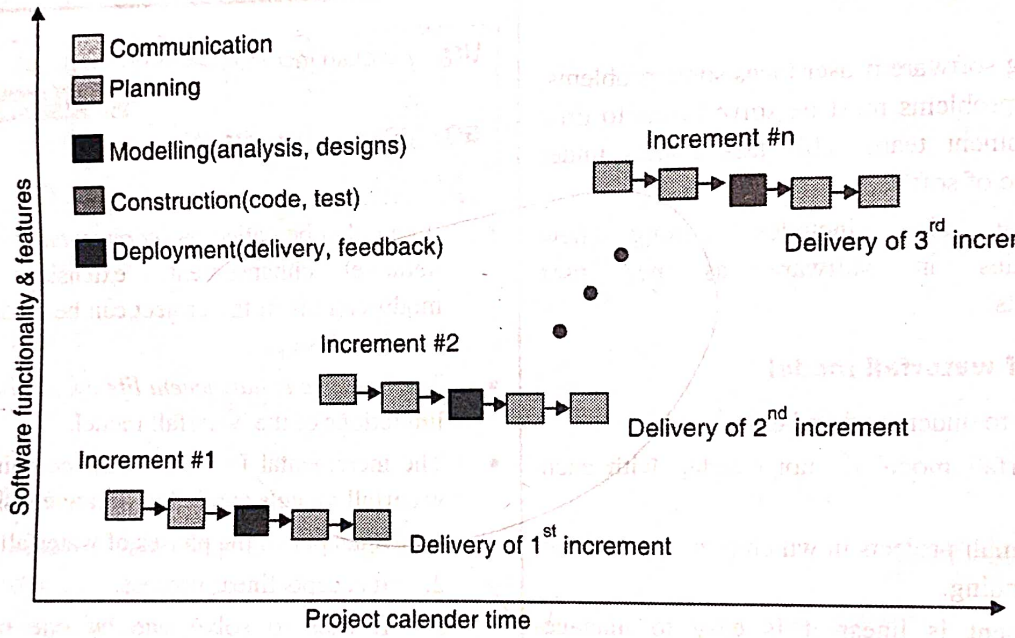


Fig. 1.9.1 : Incremental Model

Following tasks are common to all the models

1. **Communication :** Helps to understand the objective.
2. **Planning :** Required as many people (software teams) work on the same project but different functions at same time.
3. **Modelling :** Involves business modelling, data modelling, and process modelling.
4. **Construction :** This involves the reuse of software components and automatic code.
5. **Deployment :** Integration of all the increments.

☞ **Characteristics of Incremental Model**

1. System is broken down into many mini development projects.
2. Partial systems are built to produce the final system.
3. First tackled highest priority requirements.
4. The requirement of a portion is frozen once the incremented portion is developed.

Advantages of Incremental Model

1. After each iteration, regression testing should be conducted. During this testing, faulty elements of the software can be quickly identified because few changes are made within any single iteration.
2. It is generally easier to test and debug than other methods of software development because relatively smaller changes are made during each iteration. This allows for more targeted and rigorous testing of each element within the overall product.
3. Customer can respond to features and review the product for any needed or useful changes.
4. Initial product delivery is faster and costs less.

Disadvantages of Incremental Model

1. Resulting cost may exceed the cost of the organization.
2. As additional functionality is added to the product, problems may arise related to system architecture which were not evident in earlier prototypes.

1.9.1(A) DIFFERENCE BETWEEN WATERFALL MODEL AND INCREMENTAL MODEL

GQ. Differentiate between waterfall model and incremental model. **(5 Marks)**

Parameter	Waterfall Model	Incremental Model
Simplicity	Simple	Intermediate
Risk Involvement	High	Easily manageable
Flexibility to change	Difficult	Easy
User involvement	Only at beginning	Intermediate
Flexibility	Rigid	Less Flexible
Maintenance	Least	Promotes maintainability
Duration	Long	Very Long

1.9.1(B) Comparison between Evolutionary and Incremental Models.

UQ. Compare between Evolutionary and Incremental Models. **(SPPU - May 19, 5 Marks)**

Parameter	Evolutionary Model	Incremental Model
Requirements	The requirements are not clear and it evolves during the development cycle.	The requirements are clear to the development team.
Functionality	At the very initial stage, based on their understanding of the customer's requirements they developed the core modules and functionalities and deliver to the customer for feedback.	The requirements are split into the slices and one by one, slices are picked based on selection criteria.
Called as	This is called an iteration. In each iteration they release a working software after performing integration, and level of testing.	It is called an increment and in each increment a deliverable product is given to the user.
Developer team knowledge	The development team does not know how much has completed and how much need to be complete in the near future.	The development team knows how much has completed and how much need to be complete in the near future.
Picking criteria	Picking criteria for the evolutionary model: when the requirements are not clear.	Picking criteria for the incremental model: when the requirements are more or less fixed and clearer to the customer.

1.9.2 RAD Model

GQ. Write short note on RAD. **(5 Marks)**

RAD is Rapid Application Development. It is high speed adaptation of waterfall model. It is the example of incremental software process model.

The main disadvantage of incremental model is - the quality of product is very poor as well as it also requires more time to develop. Therefore, RAD model is only designed to produce a good quality product in very short duration of time.

Method

1. It also adapts generic framework activities like waterfall model that means five phases.
2. This allots sufficient time for study of the existing system.
3. It accepts sufficient number of staff for development of the system.
4. Staff is distributed into various teams (i.e. senior programmer, junior programmer, team leader, project leader etc)
5. **Communication** phase is used to understand business problem as well as it tries to identify all type of problems and gather all type of information.
6. **Planning** is essential because it divides work into manageable different parts and distributed among the various teams.
7. **Modeling** will be accomplished by different teams simultaneously. It includes various phases - Business modelling (BM), Data modeling (DM), Process modeling (PM), Application Generation (AG) and Testing & Turnover (TT).
   - *Business modeling* collects all essential information about the product from market view or from the business point of view such as - what information drives the business process?, what information is generated and how is it done and where does it go and all such questions.
   - *Data modeling* shows the flow of data using some techniques like DFD.
   - *Process modeling* performs each process which handles the flow of data. It decides the activities performed in each process.
   - *Application Generation* is about using a set of automated tools to facilitate the construction of the software say for example, the 4th GL techniques.
   - *Testing and Turn over* : Many of the components that will be used in the development of the proposed system might have already been tested since RAD focuses on reuse which reduces overall testing time. It is needed to concentrate on testing the new components.

   Once modeling activity of each team is over, it immediately starts construction phase.

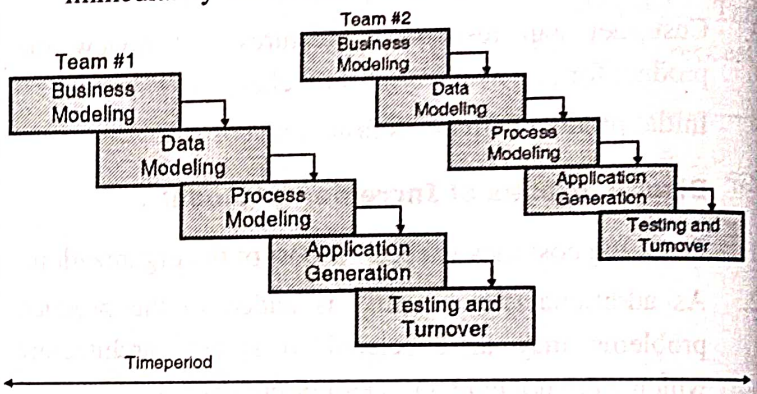


Fig. 1.9.2 : Modeling phase of RAD Model

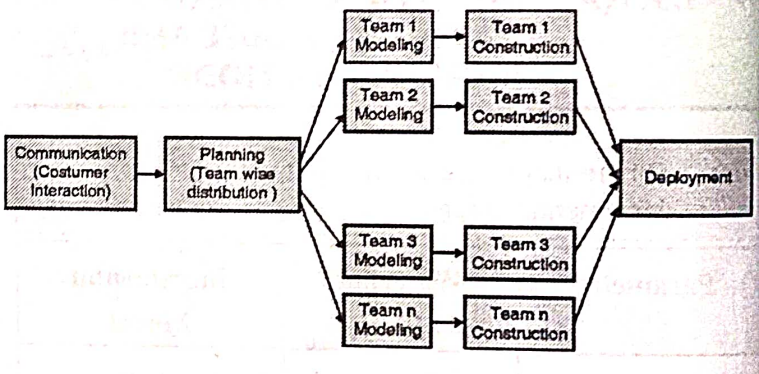


Fig. 1.9.3 : RAD model

8. In **construction** phase, each team develops the code of our product as well as performs testing. It reuses old software components and develops new functions which are called as automatic code generation.
9. **Deployment** collects each code from different team and clubs them into single project implement it and if required then perform iteration among the previous phases.

1.10 ADVANTAGES AND DISADVANTAGES OF RAD MODEL

Advantages

1. It requires very less time to develop the product.

2. Planning and Design is performed before construction so it is not lengthy.
3. Product may be produced before its delivery date.

☞ **Disadvantages**

1. Large projects require sufficient or more human resources.
2. If developers and customers do not interact with each other then whole project may elapse.
3. If system is to be modularized properly then RAD is problematic.
4. If high performance is the Issue then RAD does not work.
5. It also doesn't work in high technical risk.

1.11 EVOLUTIONARY PROCESS MODELS

- Normally the Evolutionary Development Model is based on the principle that "stages consist of growing increments of an operational software product, with the way of evolution being discovered by operational experience".
- According to the business need and the changing nature of the market there are lot of improvements required in the software product over a time.
- Due to this lot of improvement is required in the product hence evolutionary developments model is iterative in nature.
- There are two types of models in Evolutionary process. They are :

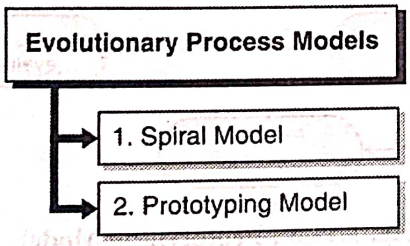


Fig. 1.11.1 : Evolutionary Process Models

1.11.1 Spiral Model

UQ. Explain with an example spiral model with its merits and demerits SPPU - Aug.17

UQ. Explain spiral model in detail. SPPU - May 18, 5 Marks

- Spiral model is a combination of iterative model and waterfall model.
- Spiral model has four phases of development each of these phases is called as spiral.

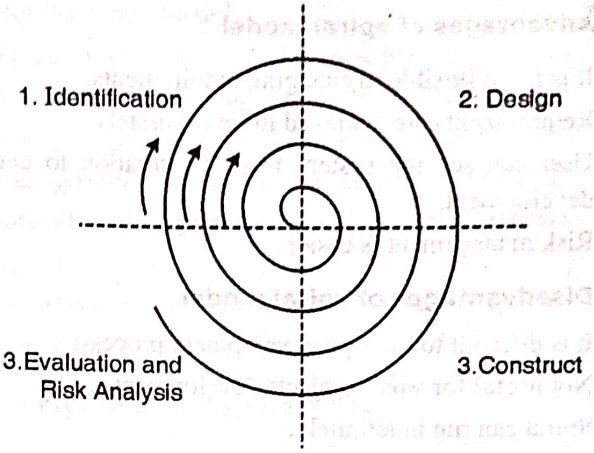


Fig. 1.11.2 : Spiral Model

(i) Identification

- This phase identifies all business requirements of system at the beginning. It involves clear understanding of requirements by communication between stakeholders and customer.
- In the subsequent iterations all subsystem requirements and unit requirements are identified.

(ii) Design

- In first iteration, design phase develops conceptual design of system based on initially gathered requirements.
- In further spirals or iterations, it develops logical design, physical design, architectural design and final design of system.

(iii) Construct

- Initially construct phase develops a code for conceptual design to get user feedback.
- In next subsequent spirals, detailed working model of software is constructed with increment number and are delivered to customer for feedback.

(iv) Evaluation and risk analysis

- In this phase management risks like cost overrun are identified and monitored, technical feasibility of system is also done.
- At end of each spiral customer evaluates software for potential risks in system and provides feedback.
- Spiral model can be used for projects where budget and risk evaluation are important factors.

- If customers are not sure about their requirements then spiral model is useful than waterfall model.

Advantages of spiral model

(i) It is more flexible to changing requirements.
(ii) Requirements are achieved more accurately.
(iii) User can see the system from 1st iteration to end of development.
(iv) Risk management is easier.

Disadvantages of spiral model

(i) It is difficult to manage development process.
(ii) Not useful for small projects development.
(iii) Spiral can run indefinitely.
(iv) It requires excessive documentation work as documentation is prepared for each iteration.

Table 1.11.1 : Comparison between Waterfall and Spiral Model

Parameter	Waterfall Model	Spiral Model
Basic	Process flows from top to bottom like a flow of water from a hill to ground. Flow of water can't be reversed - similarly any new changes cannot be incorporated in the middle of the project development.	Best suitable for projects associated with risks.
Next Phase	Process goes to the next phase only after the completion of the previous phase. Here, end user feedback is not taken into consideration for any change in SRS. This will result in restarting the work from beginning.	Each and every step goes through testing which makes it easy to recover any error and fix it then and there itself. In this model we don't have to start work from beginning.
Nature	Purely a pre planned /strategic in nature.	Used to build a product which doesn't have adequate requirement gathering.
Focus	Focuses more on requirement gathering.	Focuses more on risk analysis which is not much considered in waterfall model.
Customer feedback	Customer feedback is not considered at every step of project development.	Customer feedback is considered at every step of project development.

1.11.2 Prototyping Model

UQ. Explain prototyping model along with their relative merits and demerits. **SPPU - Dec.17, 5 Marks**

- Prototyping model refers to developing software application prototypes (early approximation / version) which displays the behaviour of product under development but may not actually contain the exact logic of the original application.
- Prototyping allows user to evaluate developers' proposal and try it before actual implementation.
- Prototyping model is widely used popular software development model as it helps to understand user requirements in early stage of development process.

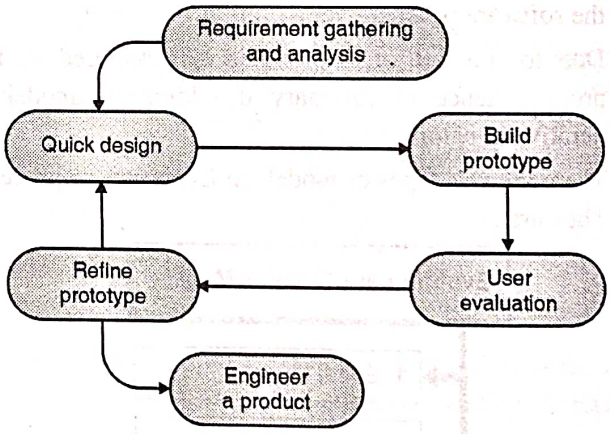


Fig. 1.11.3 : Prototyping Model

Phases of prototyping model are as follow :

(i) Requirement gathering and analysis

- Building of prototype begins with requirement gathering and analysis.
- In this phase various users of system are interviewed in order to know their system requirements or expectations from system.
- Requirement specification report is generated as an output of this phase.

(ii) Quick Design

- After gathering and analysis of user requirements quick design (prototype design) of system is developed.
- Quick design is not detailed design of system; it contains only necessary characteristics of the system which gives basis idea of system to the end users.

(iii) Build prototype

- Based on feedback received from user about quick design of system, the first prototype of system is created.
- Prototype is working model of system under development.

(iv) Use Evaluation

- Once prototype is built, proposed system is presented to end user of system for its evaluation.
- User determines strengths and weaknesses of prototype i.e. what needs to be added or what is to be removed.
- All the comments and suggestions given by user in feedback are passed to developer.

(v) Refining prototype

- Depending on comments and suggestions came from user, developers refine previous prototype to form new prototype of system.
- New prototype is again evaluated just like previous prototype.
- The process of evaluation and refining prototype continues until all user requirements are met by prototype.

(vi) Engineer a product

- Once evaluation and refining of prototype completes i.e. when user accepts final prototype of system the final system is evaluated thoroughly and deployed at user's site.
- Deployment of engineered product is further followed by regular maintenance of system.

When to use Prototype model

- Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
- Typically online system in which web interfaces have a very high amount of interaction with end users, are best suited for Prototype model.
- It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
- Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system.
- They are excellent for designing good human computer interface systems.

☞ Advantages of prototyping model

(i) It enables early evaluation of system by providing working model to end users at early stage of development.

(ii) Refining of prototype results in better implementation of system requirements.

(iii) Communication between developer and user reduces ambiguity.

(iv) More involvement of user in development process results in meeting user's requirement at greater extent.

☞ Disadvantages of prototyping model

UQ. What are the potential problems of Prototyping Model ? **SPPU - Dec. 17, 2 Marks**

(i) Refining of prototype continues until user is completely satisfied, thus it is time consuming and expensive process.

(ii) More involvement of user in development process is not accepted by developer always.

(iii) Refining of prototype again and again may disturb the working of development team.

(iv) Practically prototyping model results in increasing the complexity of system as scope of system extends beyond original plan.

☞ Difference between classic life cycle and prototyping model

GQ. Compare the Classical Life Cycle and Prototyping Models. **(5 Marks**

Table 1.11.2 : Comparison between Classic life cycle and Prototyping model

Parameter	Classic Life Cycle Model	Prototyping Model
Process	It is not an iterative process.	It is an evolutionary i.e. iterative process model.
Development	During development, mostly product extension or addition of new requirements is not encouraged.	During development, product extension may be possible.
Cost	Testing is not conducted from the initial phases of the SDLC – therefore when errors are detected after the coding phase, it costs a lot for refining the product.	Building and refining of product is involved before the actual engineering process – therefore, much cost is not involved
Includes	Included Manual coding	Uses readymade tools such as CASE tools for coding
Criteria	Communication with the users is done at the start i.e. while requirement gathering and then while testing phase. No ongoing communication with the users is encouraged	Develops the product through ongoing communication with the user

1.12 AGILE SOFTWARE DEVELOPMENT : AGILE MANIFESTO, AGILITY PRINCIPLES, AGILE METHODS

UQ. Explain in brief Agile Manifesto **SPPU - Dec.17, 5 Marks**

UQ. Explain Agile manifesto and agility principles **SPPU - May 18, 8 Marks**

UQ. What are the key principles for agile requirements ? **SPPU - Dec.18, 5 Marks**

UQ. State the Agility principles **SPPU - Dec.18, 5 Marks**

UQ. Write the manifesto for agile software development. **SPPU - May 19, 4 Marks**

- Agile software development or **agility** describes an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customers (end users).
- It advocates adaptive planning, evolutionary development, early delivery and continual improvement, and it encourages rapid and flexible response to change.
- The term Agile was popularized, in this context, by the Manifesto for Agile Software Development.
- The values and principles adopted in this manifesto were derived from and underpin a broad range of software development frameworks, including Scrum and Kanban.
- There is significant subjective evidence that adopting agile practices and values improves the agility of software professionals, teams and organizations.

1.12.1 Plan-driven and Agile Development

Agile software development values

- Based on their combined experience of developing software and helping others do that, the seventeen signatories to the manifesto proclaimed that they 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.
- As per the view of Scott Ambler (Canadian software engineer, consultant and author) :
- Tools and processes are important, but it is more important to have competent people working together effectively.
- Good documentation is useful in helping people to understand how the software is built and how to use it, but the main point of development is to create software, not documentation.

- A contract is important but is no substitute for working closely with customers to discover what they need.
- A project plan is important, but it must not be too rigid to accommodate changes in technology or the environment, stakeholders' priorities, and people's understanding of the problem and its solution.

1.12.2 Agility Principles

UQ. State and explain agility principles
SPPU - Dec.17, 5 Marks

The Manifesto for Agile Software Development is based on twelve principles :

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Working software is delivered frequently (weeks rather than months).
4. Close, daily cooperation between business people and developers.
5. Projects are built around motivated individuals, who should be trusted.
6. Face-to-face conversation is the best form of communication (co-location).
7. Working software is the primary measure of progress.
8. Sustainable development, able to maintain a constant pace.
9. Continuous attention to technical excellence and good design.
10. Simplicity is essential.
11. Best architectures, requirements, and designs emerge from self-organizing teams.
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly.

1.12.3 Advantages of Agile Process

GQ. What are Advantages of Agile Processes ? **(5 Marks)**

There are number of advantages of Agile Process :

1. Stakeholder Engagement,
2. Transparency,
3. Early and Predictable Delivery,
4. Predictable Costs and Schedule,
5. Allows for Change,
6. Focuses on Business Value,
7. Focuses on Users,
8. Improves Quality.

1.12.4 Difference between Prescriptive Process Model and Agile Process Model

GQ. Differentiate between prescriptive process model and agile process model. (any four points) **(5 Marks)**

Parameter	Prescriptive Process Model	Agile Process Model
Basic aim	Developed to bring order and structure to the software development process.	These models satisfy customer through early and continuous delivery.
Functionality	It can accommodate changing requirements	Defines a distinct set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software.
Popularity	It is more popular.	Comparatively less popular.
Examples	Water fall model, Incremental models	Scrum, extreme Programming (XP), Feature Driven Development (FDD), Dynamic Systems Development Method (DSDM), Adaptive Software Development (ASD).

1.13 MYTH OF PLANNED DEVELOPMENT

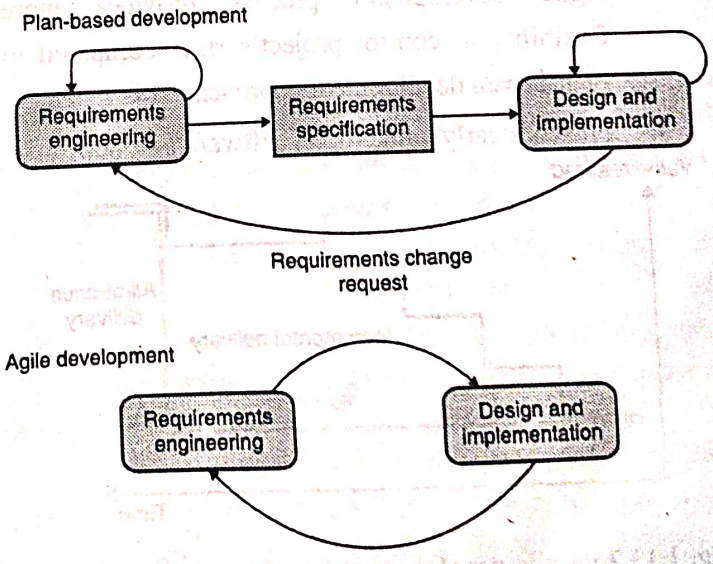


Fig. 1.13.1 : Plan-driven and Agile development

The **plan-driven development** approach usually uses the Waterfall model, which consists of the phases such as: requirement gathering, analysis, design, coding, testing, and implementation.

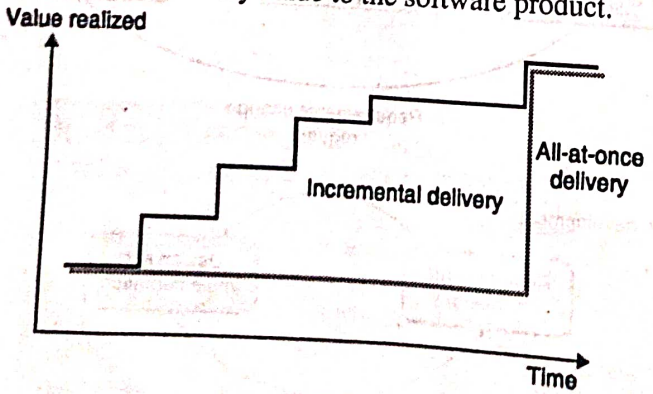
These phases are executed sequentially one by one.

Problems with plan-driven (traditional software development / waterfall) approach are :

1. Excessively long time to deliver the product
2. Customer involvement is less
3. Over developed product of which 60% features are never used
4. Cost of delivering software is too high
5. Too much time taken and unplanned wastage of efforts on fixing defects and rework designs due
6. Poor quality software developed
7. Ability to respond to changes is low
8. Project failure rate is too high i.e. 70% or more

- Hence Agile development approach is chosen to overcome these drawbacks :

1. Agile development evolved in mid-90's – the word "Agile" was adopted in 2001.
2. Iterative development, done in small incremental chunks, validating requirements at each step.
3. Is also similar to "spiral models", except spiral iterations are longer and rely on "prototypes, where Agile emphasizes "working software".
4. Agile development process provides more flexibility to control project's risks compared to Plan-driven development approach.
5. Provides early value to the software product.



g. 1.13.2 : Agile development provides early value to the software product

1.14 INTRODUCTION TO EXTREME PROGRAMMING

GQ. Explain how extreme programming process supports agility with its framework activities? **(5 Marks)**

- Extreme Programming (XP) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements.
- As a type of agile software development, it advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.
- Other elements of extreme programming include : Programming in pairs or doing extensive code review, unit testing of all code, avoiding programming of features until they are actually needed, a flat management structure, code simplicity and clarity, expecting changes in the customer's requirements as time passes and the problem is better understood, and frequent communication with the customer and among programmers.

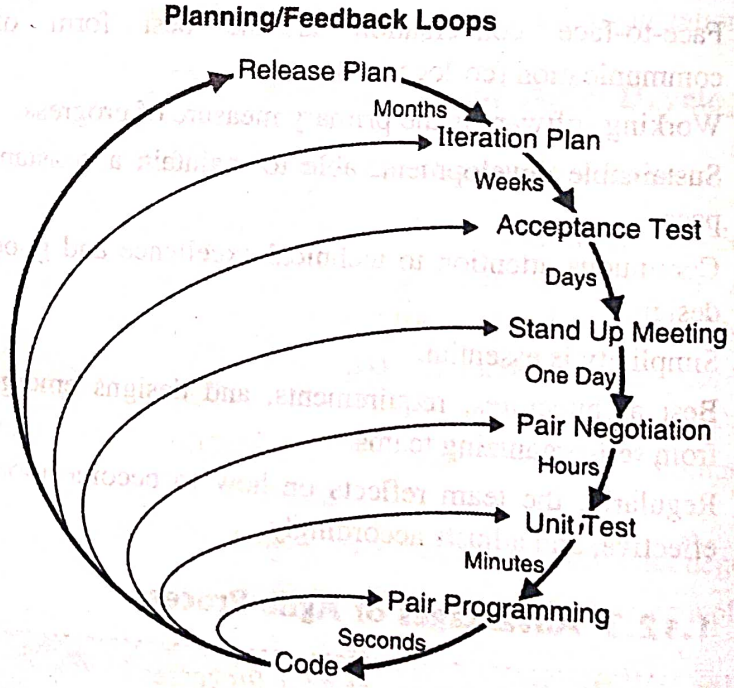


Fig. 1.14.1 : Extreme Programming

- The methodology takes its name from the idea that the beneficial elements of traditional software engineering practices are taken to "extreme" levels.
- As an example, code reviews are considered a beneficial practice; taken to the extreme, code can be reviewed continuously, i.e. the practice of pair programming.

1.14.1 XP Values

Extreme programming initially recognized four values in 1999 : communication, simplicity, feedback, and courage. A new value, respect, was added in the second edition of Extreme Programming Explained. Those five values are described below.

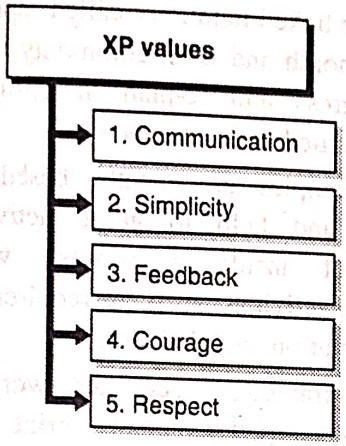


Fig. 1.14.2 : XP values

1. Communication

- Building software systems requires communicating system requirements to the developers of the system. In formal software development methodologies, this task is accomplished through documentation.
- Extreme programming techniques can be viewed as methods for rapidly building and disseminating institutional knowledge among members of a development team.
- The goal is to give all developers a shared view of the system which matches the view held by the users of the system.
- To this end, extreme programming favors simple designs, common metaphors, collaboration of users and programmers, frequent verbal communication, and feedback.

2. Simplicity

- Extreme programming encourages starting with the simplest solution. Extra functionality can then be added later.
- The difference between this approach and more conventional system development methods is the focus on designing and coding for the needs of today instead of those of tomorrow, next week, or next month.
- This is sometimes summed up as the "You aren't gonna need it" (YAGNI) approach.
- Proponents of XP acknowledge the disadvantage that this can sometimes entail more effort tomorrow to change the system; their claim is that this is more than compensated for by the advantage of not investing in possible future requirements that might change before they become relevant.
- Coding and designing for uncertain future requirements implies the risk of spending resources on something that might not be needed, while perhaps delaying crucial features.
- Related to the "communication" value, simplicity in design and coding should improve the quality of communication. A simple design with very simple code could be easily understood by most programmers in the team.

3. Feedback

- Within extreme programming, feedback relates to different dimensions of the system development :
- **Feedback from the system :** By writing unit tests, or running periodic integration tests, the programmers have direct feedback from the state of the system after implementing changes.
- **Feedback from the customer :** The functional tests (aka acceptance tests) are written by the customer and the testers. They will get concrete feedback about the current state of their system. This review is planned once in every two or three weeks so the customer can easily steer the development.
- **Feedback from the team :** When customers come up with new requirements in the planning game the team directly gives an estimation of the time that it will take to implement.
  - Feedback is closely related to communication and simplicity. Flaws in the system are easily communicated by writing a unit test that proves a certain piece of code will break.
  - The direct feedback from the system tells programmers to recode this part.
  - A customer is able to test the system periodically according to the functional requirements, known as user stories.
  - As per Kent Beck (American software engineer and the creator of extreme programming), "Optimism is an occupational hazard of programming. Feedback is the treatment".

4. Courage

- Several practices represent courage. One is the commandment to always design and code for today and not for tomorrow.
- This is an effort to avoid getting delayed in design and requiring a lot of effort to implement anything else. Courage enables developers to feel comfortable with refactoring their code when necessary.
- This means reviewing the existing system and modifying it so that future changes can be implemented more easily.
- Another example of courage is knowing when to throw code away: courage to remove source code that is obsolete, no matter how much effort was used to create that source code.
- Also, courage means persistence: a programmer might be stuck on a complex problem for an entire day, then solve the problem quickly the next day, but only if they are persistent.

5. Respect

- The respect value includes respect for others as well as self-respect.
- Programmers should never commit changes that break compilation, that make existing unit-tests fail, or that otherwise delay the work of their peers.
- Members respect their own work by always striving for high quality and seeking for the best design for the solution at hand through refactoring.
- Adopting the four earlier values leads to respect gained from others in the team.
- Nobody on the team should feel unappreciated or ignored. This ensures a high level of motivation and encourages loyalty toward the team and toward the goal of the project. This value is dependent upon the other values, and is oriented toward teamwork.

1.15 SCRUM

UQ. Explain in brief the SCRUM process with the help of diagram? **SPPU - Dec.17, 5 Marks**

UQ. Draw and explain concept of SCRUM **SPPU - May 18, 8 Marks**

UQ. Explain how scrum works with basic diagram. **SPPU - May 19, 6 Marks**

UQ. Write and explain the role of the scrum master **SPPU - May 19, 4 Marks**

UQ. Explain SCRUM with the help of diagram. **SPPU - Dec.19, 8 Marks**

- **Scrum** is an agile framework for managing knowledge work, with an emphasis on software development. It is designed for teams of three to nine members, who break their work into actions that can be completed within time boxed iterations, called "sprints", no longer than one month and most commonly two weeks, then track progress and re-plan in 15-minute stand-up meetings, called daily scrums.
- Scrum principles are mostly based on the agile manifesto and help to guide activities regarding development inside a process which includes framework activities such as requirements, analysis, design, evolution, and delivery.
- In all the framework activities, work tasks happen within a process pattern called a sprint.
- The work implemented in a sprint is tailored to the problem at hand and is defined and usually tailored in real time by the respective Scrum team.
- Scrum mostly focus on the use of a bunch of software process patterns which are proved effective for projects with restrictive timelines, changing requirements, and business criticality. All such process patterns defines several sets of development actions :
  - **Backlog** - A prioritized list regarding project needs or features which offer business value for the customer. It is possible to add items to the backlog at any time. The respective product manager assesses the backlog and changes the priorities as per necessity.
  - **Sprints** - Includes work units which are necessary to gain a requirement defined in the backlog which should be incorporated in predefined time-box (usually thirty days).
- Modifications (such as backlog work items) are not involved during the sprint. Therefore, team members are allowed by the sprint to work in a short-term, but stable environment.
- Scrum meetings are short (usually fifteen minutes) meetings conducted on daily basis by the Scrum team. Three most important key questions which are asked and must be answered by all team members are :
  1. What did you do since the last team meeting ?
  2. What obstacles are you encountering ?
  3. What do you plan to accomplish by the next team meeting ?

A team leader, who is known as a **Scrum master**, handles the meeting and evaluates the responses from each person.

The important use of Scrum meeting to the team is that it can uncover critical issues as early as possible.

One more benefit of these daily meetings is that they lead to "knowledge socialization" and thus encourage a self-organizing team structure.

- **Demos** - implement the software increment at the client side (customer) so that the newly developed functionality can be demonstrated as well as evaluated by the customer.
- It is also possible that the demo may not able to contain all planned functionality, but instead such functions which can be delivered within the established time-box.

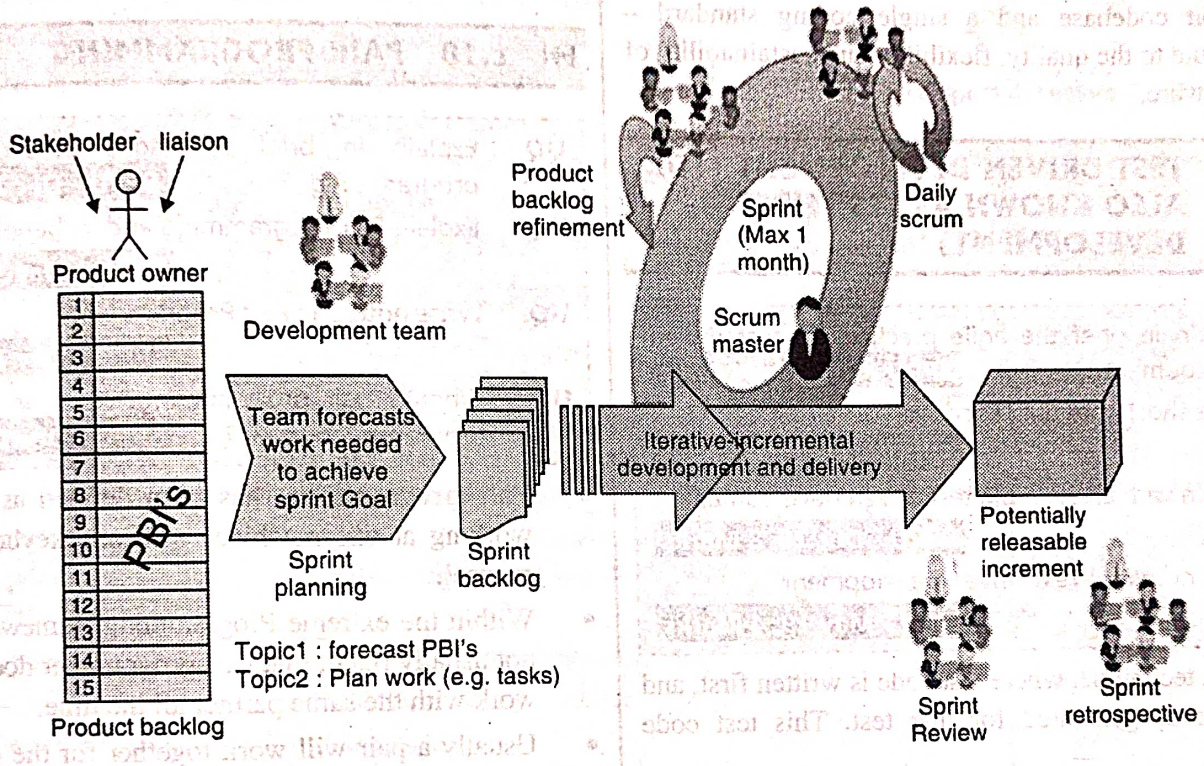


Fig. 1.15.1 : Scrum Framework

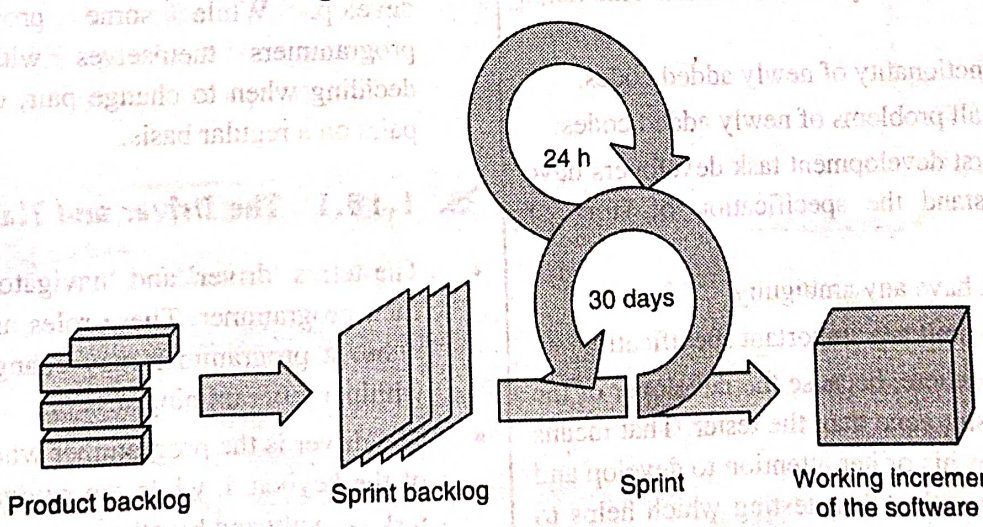


Fig. 1.15.2 : The scrum process

1.16 AGILE PRACTICES

- Using an iterative framework, the Agile methodology relies upon the interaction of self-organizing teams of people who have the cross-functional skill sets required to develop tested, working software.
- The most commonly used programming practices are test-driven development, code refactoring, continuous integration, simple code design, pair programming, a common codebase and a single coding standard – contribute to the quality, flexibility and sustainability of the software.

1.17 TEST DRIVEN DEVELOPMENT (TDD ALSO KNOWN AS TEST FIRST DEVELOPMENT)

UQ. Explain in brief the agile practices of test driven development. **SPPU - Dec.17, 3 Marks**

UQ. Write short note on: Test driven development. **SPPU - Dec.17, 6 Marks**

UQ. Explain with an example test driven development **SPPU - Dec.18, 8 Marks**

UQ. Write note on : Test Driven Development **SPPU - May 19, 4 Marks**

- **TDD** is a technique, where test code is written first, and then code is generated for that test. This test code includes :

  It consists of quick and easy executed tests. This helps to;
  - Checks all functionality of newly added codes.
  - Finds almost all problems of newly added codes.
- Because of test, first development task developers have thoroughly understand the specification of story of customers.
  - Code does not have any ambiguity.
  - Code does not omit any important specification.
- It avoids **'test-lag'** where, because the developer of the system works at faster pace than the tester. That means developer only gives his or her attention to develop and implement the code rather than testing which helps to extreme programming.

Disadvantages of the Test First Code

1. Programmers always prefer to write a code for tasks not for test.
2. Some time test codes are incomplete.
3. Some test codes are not checks any exceptional cases.
4. Some tests are very difficult to write because of complexity of the code.
5. Difficult to judge completeness of the test code.
6. Crucial part of the system may not be executed so it remains untested.
7. Customer does not give sufficient time for development.

1.18 PAIR PROGRAMMING

UQ. Explain in brief the agile practices of pair programming. **SPPU - Dec.17, 3 Marks**

UQ. Explain Pair Programming and its benefits **SPPU - May 18, 8 Marks**

UQ. Write note on : Pair Programming **SPPU - May 19, 4 Marks**

UQ. What is pair programming? What is role of pair programming in XP ? **SPPU - Dec.19, 8 Marks**

- **Pair programming** has been defined as 'two people working at one machine, with one keyboard and one mouse'.
- Within the extreme Programming framework, a pair is not usually fixed, that is, a programmer does not tend to work with the same partner all the time.
- Usually a pair will work together for the duration of a single task that might most often take a day or two to develop. While some projects empower the programmers themselves with responsibility for deciding when to change pair, others enforce rotating pairs on a regular basis.

1.18.1 The Driver and Navigator Roles

- The terms 'driver' and 'navigator' describe the role of each programmer. These roles are by no means fixed, rather a programmer may change roles several times within a programming session.
- The driver is the programmer who currently has control of the keyboard, while the navigator contributes to the task verbally and by other means.
- The navigator role could be considered something of a mystery. Some suggest that the navigator provides a constant design and code review, others consider him/her to be working at a higher level of abstraction than the driver.

- This could also be described as the driver working tactically while the navigator works strategically.
- Research on commercial programming teams has also shown that rather than take a 'divide and conquer' approach to a programming task, the driver and navigator work together on each aspect of the problem.

1.18.2 The Pair Programming Team

- Commercial pair programming generally takes place within the larger context of a programming 'team'.
- A number of studies have considered the environment within which pair programming takes place.
- In particular, Sharp and Robinson (2003) focus on XP in their ethnographic work, providing useful insights into what it means to be a member of an XP team in a number of different commercial companies.
- Kessler and Williams (2003) consider pair programming as providing a form of 'legitimate peripheral participation'.
- Here, a trainee programmer can learn the 'craft' of programming by working as part of a programming pair, playing a useful but controlled part in the production of software while being immersed in the project environment, in which (s)he can learn through observation.
- Similarly, Bryant, Romero et al. (2006) discuss the advantages of the pair's conversation making their work visible to others on the team. T
- They cite occasions where overhearing has resulted in advice and guidance from others outside the pair, or indeed, where pairs have been reformed according to whom on the team is best equipped to work on the task at hand.

1.19 CONTINUOUS INTEGRATION IN DEVOPS

GQ. What is DevOps in software engineering ? **(2 Marks)**

- **DevOps** is a set of practices that automates the processes between software development and IT teams, in order that they can build, test, and release software faster and more reliably.
- The word 'DevOps' is a grouping of two words 'development' and 'operations.'
- DevOps helps to maximize an organization's speed to deliver applications and its services. It always allows organizations to serve their customers better and compete more strongly in the market.
- In other words, DevOps is an alignment of development and IT operations with better communication and collaboration.

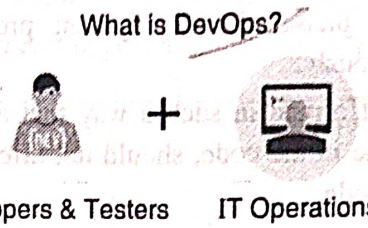


Fig.1.19.1 : DevOps

1.19.1 DevOps

Lifecycle

- DevOps is combination of development and operations.
- Understanding of DevOps is not possible without knowing DevOps lifecycle.
- Here is a brief information about the Continuous DevOps life-cycle:

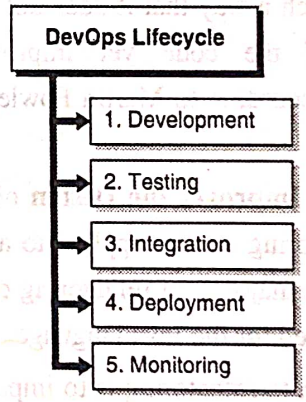


Fig. 1.19.2

1. Development

- In this DevOps stage the development of software takes place constantly.
- In this stage, the whole development process is divided into small development cycles. This benefits DevOps team to speed up software development and delivery process.

2. Testing

QA team use tools like Selenium to identify and fix bugs in the new piece of code.

3. Integration

In this phase, new functionality can be integrated with the previous code, and testing takes place. Constant development is only possible due to continuous integration and testing.

4. Deployment

- In this phase, the deployment process takes place continuously.
- It is performed in such a way that any changes made any time in the code, should not affect the functioning of the code.

5. Monitoring

In this phase, operation team will take care of the inappropriate system behavior or bugs which are found in production.

1.20 REFACTORING

GQ. What do you mean by refactoring ? **(5 Marks)**

- **Refactoring** is "the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure," according to Martin Fowler, the "father" of refactoring.
- **Refactoring Improves the Design of Existing Code.** While refactoring can be applied to any programming language, the majority of refactoring current tools have been developed for the Java language.
- One approach to refactoring is to improve the structure of source code at one point and then extend the same changes systematically to all applicable references throughout the program.
- The result is to make the code more efficient, scalable, maintainable or reusable, without actually changing any functions of the program itself.

Advantages of Refactoring

GQ. *Give the importance of refactoring in improving quality of service.* **(3 Marks)**

There are several **advantages of refactoring** *which shows its importance in improving quality of service :*

1. Refactoring improves objective attributes of code (length, duplication, coupling and cohesion, cyclomatic complexity) that correlate with ease of maintenance
2. Refactoring helps code understanding
3. Refactoring encourages each developer to think about and understand design decisions, in particular in the context of collective ownership/collective code ownership
4. Refactoring favours the emergence of reusable design elements (such as design patterns) and code modules

1.21 CASE STUDIES : AN INFORMATION SYSTEM – LIBRARY MANAGEMENT SYSTEM

- A library management information system can also be called a library management system (LMS) or integrated library system (ILS).
- It is a system that makes use of information technology (IT) to carry out managerial objectives.
- The main goal of a library management information system is to store, organize, share and retrieve vital information needed to carry out daily operational functions of the library.

Function

1. A library management system (LMS) involves three basic elements: hardware, software and the users.
2. LMS is a network of computers that uses a certain program to facilitate technical functions of the library.
3. One such function is electronic cataloguing. With LMS, library users can trace desired books electronically without going through shelves.
4. LMS also facilitates the lending process by keeping records of items lent and borrowers' information. LMS supports other administrative tasks such as inventory and data processing.

Benefits

1. LMS makes everyday library tasks more efficient. This means more work can be done in less time.
2. Consequently, this decreases operational costs. This also minimizes paperwork and manual tasks, thus allowing library personnel to concentrate on other things such as interaction with users.

3. LMS also reinforces users' loyalty and satisfaction as it provides fast and reliable library services.

☞ Disadvantages

1. Libraries may need to change their LMS every now and then to avoid lagging behind the technology.
2. Although a library management system's end goals include cutting costs, setting up a new system initially requires a substantial amount of money and resources.
3. Libraries not only pay for the software but may also spend for new computers, installations, hosting and maintenance.
4. In addition, libraries must hire or train an IT support team to deal with computer network glitches.
5. As LMS becomes more sophisticated and costly, there may be less need for traditional library staff, leading to a decrease in employment opportunities in the industry.

...Chapter Ends

❑❑❑