Introduction to Software Engineering, Software Process Models

University Prescribed Syllabus

Software Engineering Fundamentals : Nature of Software, Software Engineering Principles, The Software Process, Software Myths.

Process Models : A Generic Process Model, Prescriptive Process Models : The Waterfall, Incremental Process(RAD), Evolutionary Process, Unified Process, Concurrent.

Advanced Process Models & Tools : Agile software development : Agile methods, Plan-driven and agile development, Extreme programming Practices, Testing in XP, Pair programming. Introduction to agile tools : JIRA, Kanban,

Case Studies : An information system (mental health-care system), wilderness weather system.

Syllabus Topic : Software Engineering Fundamentals

1.1 Software Engineering Fundamentals

LQ. 1.1.1 What is software? (2 Marks)

- Computer **software**, or simply software, is a generic term that refers to a collection of data or computer instructions that tell the computer how to work, in contrast to the physical hardware from which the system is built, that actually performs the work.
- In computer science and software engineering, computer software is; all the information processed by computer systems, programs and data.
- Computer software includes computer programs, libraries and related non-executable data, such as online documentation or digital media.
- Computer hardware and software require each other and neither can be realistically used on its own.
- At the lowest level, executable code consists of machine language instructions specific to an individual processor-typically a Central Processing Unit (CPU).
- A machine language consists of groups of binary values signifying processor instructions that change the state of the computer from its preceding state.
- For example, an instruction may change the value stored in a particular storage location in the computer - an effect that is not directly observable to the user.
- An instruction may also (indirectly) causes something to appear on a display of the computer system - a state change which should be visible to the user.
- The processor carries out the instructions in the order they are provided, unless it is instructed to "jump" to a different instruction, or is interrupted by the operating system.
- The majority of software is written in high-level programming languages that are easier and more efficient for programmers to use because they are closer than machine languages to natural languages.
- High-level languages are translated into machine language using a compiler or an interpreter or combination of both.
- Software may also be written in a low-level assembly language, which has strong correspondence to the computer's machine language instructions and is translated into machine language using an assembler.
- Fig. 1.1.1 shows how the user interacts with application software on a typical desktop computer. The application software layer interfaces with the operating system, which in turn communicates with the hardware. The arrows indicate information flow.

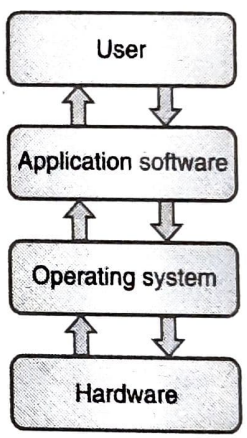


Fig. 1.1.1 : User interaction with application

Syllabus Topic : Nature of Software

1.2 Nature of Software

LQ. 1.2.1 Explain changing nature of software. (5 Marks)

- Now-a-days the software is not only remains a product, it also becomes an interface for delivering a product.
- As a product, software is able to deliver the computing potential which is basically embodied by computer hardware or precisely we can say, by a network of computers which one can access through local hardware.

- Software is considered as an information transformer or producer irrespective of whether it resides within a smart phone or functions inside a mainframe computer.
- Just like any transport vehicle, software can work as the basis for the control of the computer (operating systems), the transport of information (networks) and the generation and management of other programs (such as various software tools).
- Information delivery is the most important aspect of software. To make any data more useful in a local context, software can transform the data. It also helps in improving the competitiveness of a business by managing the business information.
- A gateway is provided by software to worldwide information networks (e.g., the Internet), and offer the way to gather information in several different forms.
- Over the last half-century, significant changes are occurred in the role of computer software.
- Now-a-days the software becomes more sophisticated as well as complex because of various aspects such as spectacular improvements in hardware performance, tremendous up-gradations in computing architectures, huge enhancements in memory and storage capacity, and extensive diversity of exotic input and output options.
- Sophistication as well as complexity leads to amazing results on success of systems, but they may also lead to huge problems for those who want to develop complex systems.
- In the economies of the industrialized world, the big software industry is considered as a dominant factor.
- In an earlier era, software was developed individually. Now-a-days an entire team works for it in which every person is responsible for one part of the technology which is necessary to deliver a complex application.

1.2.1 Defining Software

UQ. 1.2.2 Define Term Software.

SPPU - Aug. 17, 1 Mark

Software can be defined in different ways :

Definitions

1. ***Software** is set of instructions (computer programs) that when executed provide desired features, function, and performance.*
2. ***Software** is data structures that enable the programs to adequately manipulate information.*
3. ***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

LQ. 1.2.3 Describe the characteristics of software.

(5 Marks)

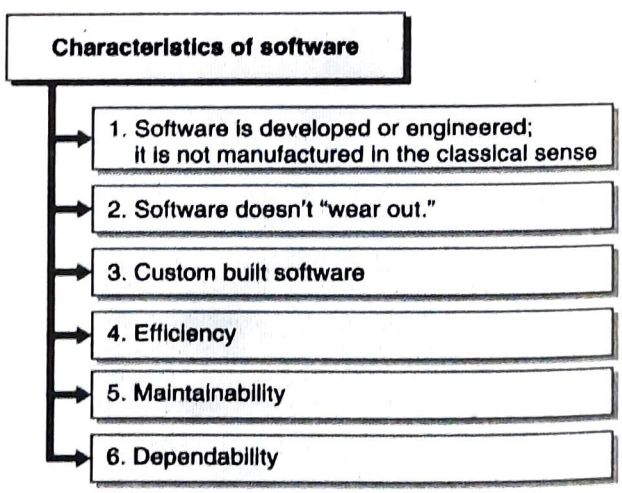


Fig. 1.2.1 : Characteristics of software

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

UQ. 1.2.4 How software engineering is different from hardware engineering? Justify.
SPPU - Aug. 17, 2 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"

UQ. 1.2.5 "Software does not wear out". State whether this statement is true or false. Justify your answer. SPPU - Aug. 17, 1 Mark

- 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

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

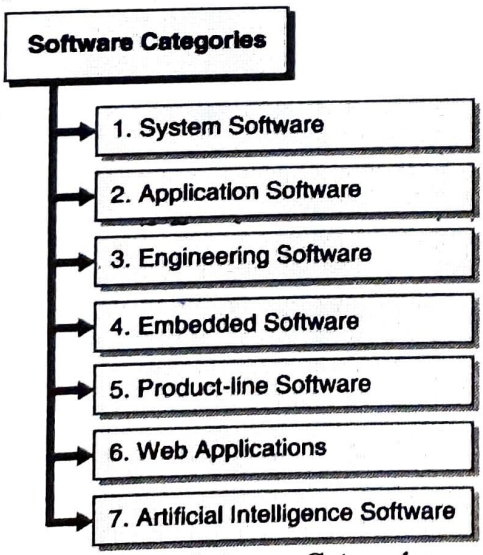


Fig. 1.2.2 : Software Categories

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.

1.3 Software Engineering

LQ. 1.3.1 Explain the term 'Software Engineering'. **(5 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

UQ. 1.3.2 Define term Software Engineering. **SPPU - Aug. 17, 1 Mark**

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

LQ. 1.3.3 Describe 4 layers of software engineering. **(5 Marks)**

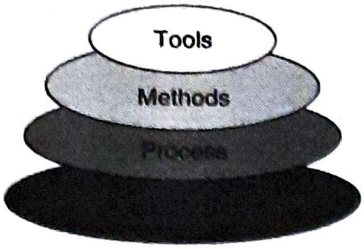


Fig. 1.3.1 : Layered Approach of software Technology

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

LQ. 1.3.4 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

LQ. 1.3.5 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.

Syllabus Topic : Software Engineering Principles

1.4 Software Engineering Principles

LQ. 1.4.1 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

LQ. 1.4.2 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.

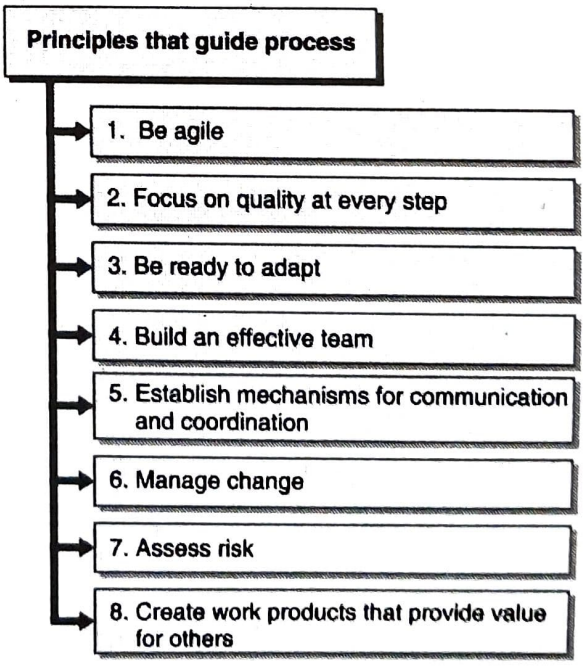


Fig. 1.4.1 : Principles that guide process

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

LQ. 1.4.3 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 :

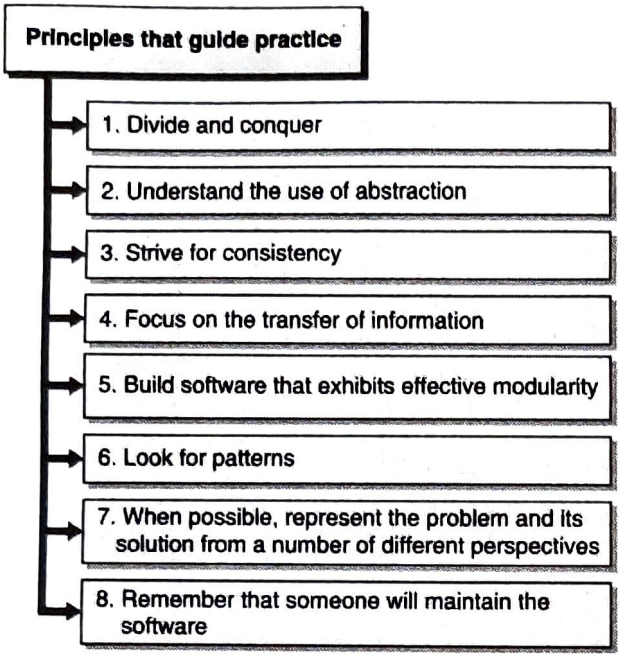


Fig. 1.4.2 : Principles that guide practice

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.

Syllabus Topic : Software Process

1.5 Software Process

LQ. 1.5.1 Explain in brief the software process. **(4 Marks)**

UQ. 1.5.2 Explain classic life cycle paradigm for software engineering and problems encountered when it is applied. **SPPU - Dec. 18, 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

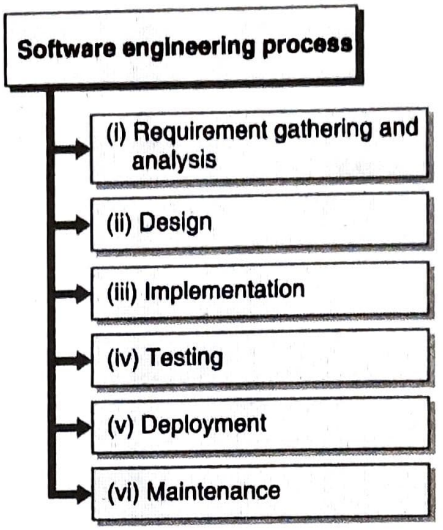


Fig. 1.5.1 : Software engineering process

- 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. 1.5.3 Explain different activities in software process framework

SPPU - Aug. 17, 4 Marks

- 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.

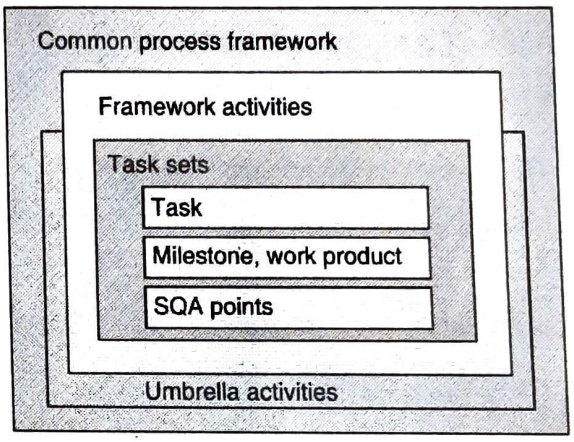


Fig. 1.5.2 : Software process framework

1.5.1(A) Umbrella Activities

UQ. 1.5.4 Explain different activities in software process framework.

SPPU - Aug. 17, 4 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

UQ. 1.5.5 What is objective of Personal Software Process(PSP) ? **SPPU - Dec. 17, 2 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

UQ. 1.5.6 What are the activities of PSP Model ? **SPPU - Dec. 17, 3 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

LQ. 1.5.7 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.

Syllabus Topic : Software Myths

1.6 Software Myths

- 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. 1.6.1 What are customer myths? Discuss the reality of these myths.
SPPU - Aug. 17, 3 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. 1.6.2 What are practitioner's myths? Discuss the reality of these myths.
SPPU - Dec. 17, Dec. 18, 5 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.

Syllabus Topic : Process Models : A Generic Process Model

1.7 Process Models : A Generic Process Model

LQ. 1.7.1 Write note on Generic Process Model. (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.

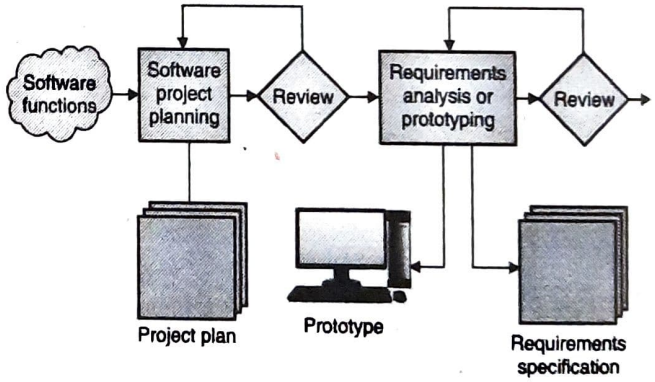


Fig. 1.7.1 : Activities in Analysis

(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.

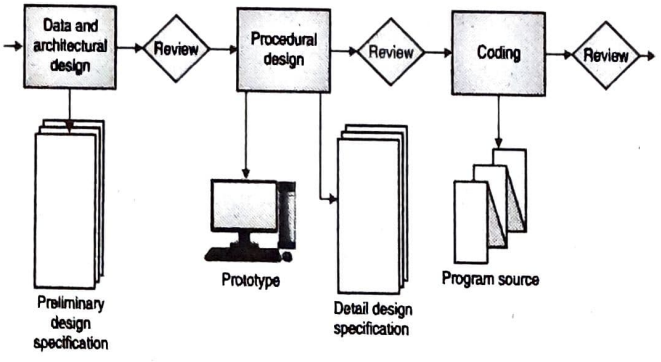


Fig. 1.7.2 : Design and Implementation (Coding)

  - 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.

Syllabus Topic : Prescriptive Process Models

1.8 Prescriptive Process Models

LQ. 1.8.1 Explain Prescriptive Process Models. **(5 Marks)**

- The following framework activities are performed irrespective of the process model selected by the organization.
  1. Communication
  2. Planning
  3. Modelings
  4. Construction
  5. Deployment
- The name 'prescriptive' is given because the model prescribes a set of activities, actions, tasks, quality assurance and change the mechanism for every project.

Syllabus Topic : Waterfall Model

1.8.1 Waterfall Model

LQ. 1.8.2 What is waterfall model ? State the practical situations in which it can be used. **(5 Marks)**

LQ. 1.8.3 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.

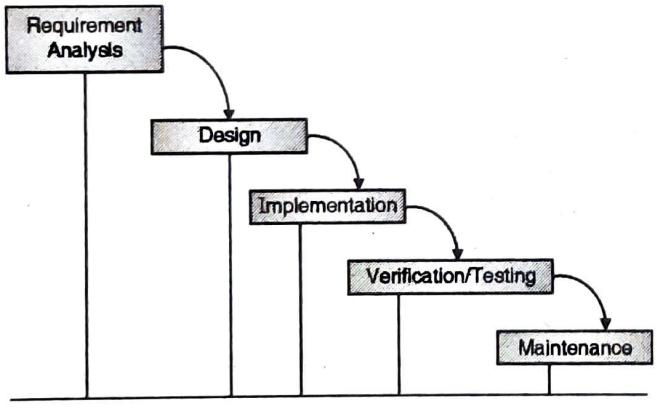


Fig. 1.8.1 : Waterfall Model

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 behavior.

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 initially.

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

☞ Disadvantages of waterfall model

(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.

Syllabus Topic : Incremental Process (RAD) Model

1.9 Incremental Process Model

UQ. 1.9.1 Explain with neat diagram Increment Model and state its advantages and disadvantages.

SPPU - May 18, 6 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.

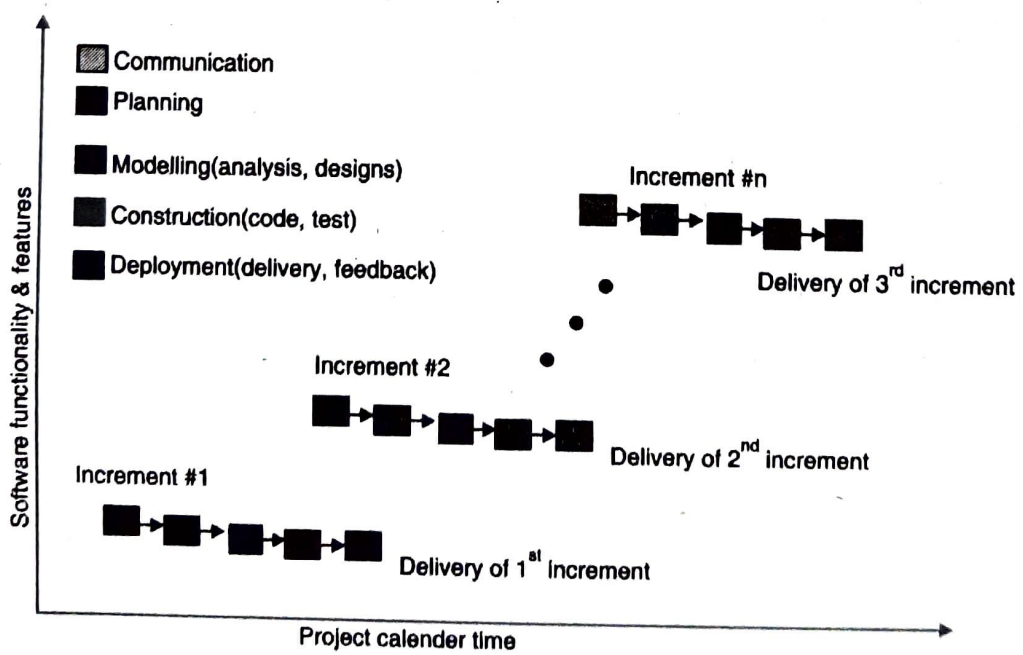


Fig. 1.9.1 : Incremental Model

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 Difference between Waterfall Model and Incremental Model

LQ. 1.9.2 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

Syllabus Topic : Evolutionary Process Model

1.10 Evolutionary Process Model

- 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.

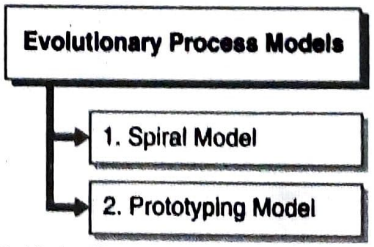


Fig. 1.10.1 : Evolutionary Process Models

1.10.1 Spiral Model

LQ. 1.10.1 With a neat diagram explain the Spiral model of software development. **(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.

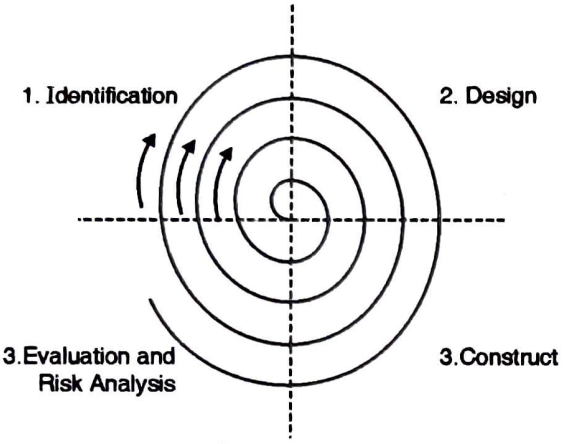


Fig. 1.10.2 : Spiral Model

(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.

1.10.2 Prototyping Model

LQ. 1.10.2 Discuss prototyping model for software development with merits and demerits. **(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.

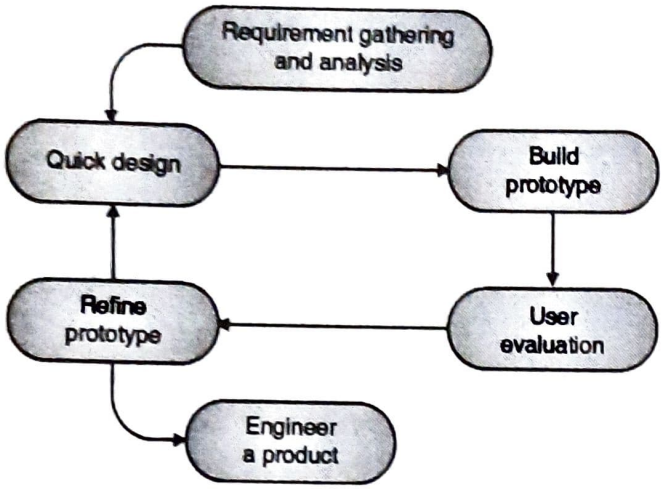


Fig. 1.10.3 : Prototyping Model

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

LQ. 1.10.3 What are the potential problems of Prototyping Model ? **(5 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.

Syllabus Topic : Concurrent Models

1.11 Concurrent Models

LQ. 1.11.1 Write note on Concurrent Models. **(5 Marks)**

- The concurrent development model is also known as concurrent engineering.
- This model helps software team in the representation of iterative as well as concurrent elements of various process models.
- For example, consider the modeling activity which has been defined for the spiral model is carried out by calling one or more software engineering actions such as prototyping, analysis, and design.
- In Fig. 1.11.1 we can observe schematic representation regarding a software engineering activity in the scope of the modeling activity with the help of concurrent modeling approach.

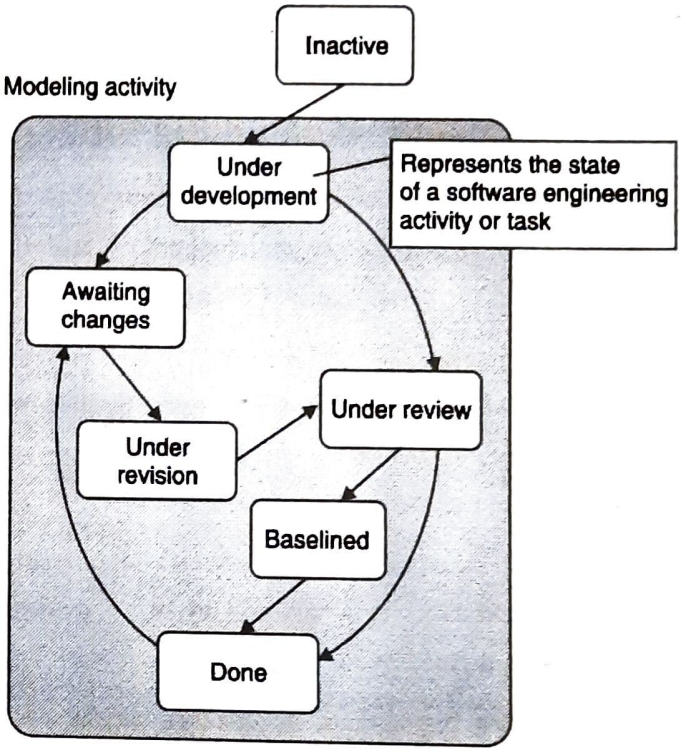


Fig. 1.11.1 : Concurrent Models

- The state of activity **modeling** may be any out of the states which are noted at any particular time.
- In the same way, it is possible to represent remaining activities, actions, or tasks such as **communication**, **construction** etc. in an analogous manner.
- All the activities regarding software engineering are executed simultaneously but exist in different states.
- For example, consider in the early phase of a project, first iteration of the communication activity (not displayed in the diagram) has been completed and exists in the **awaiting changes** state.
- The modeling activity whose state is **inactive** at the time of completion of initial communication, now changes its state to **under development**.

- However, if the customer enforces that changes in requirements should be necessarily done, the modeling activity shifts from the **under development** state into the **awaiting changes** state.
- Concurrent modeling describes a sequence of events which will activate transitions in between states for all the software engineering activities, actions, or tasks.
- For example, in the previous phase of design (an important action regarding software engineering which occurs in the process of modeling activity), an inconsistency in the requirements model is uncovered.
- Because of it the event **analysis model correction,** is generated that will activate the requirements analysis action from the **done** state into the **awaiting changes** state.
- **Concurrent modeling is proved to be compatible for all the types of software development and offers an exact picture of the ongoing state of a project.**
- **Instead of encircling all the software engineering actions, and tasks to a series of events, a process network is defined by the concurrent modeling.**
- All the activities, actions, or tasks on the network execute concurrently with other activities, actions, or tasks.
- **In the process network, events which are generated at one point activate transitions between the states.**

Syllabus Topic : Advanced Process Models & Tools: Agile software development: Agile methods

1.12 Advanced Process Models and Tools: Agile Software Development: Agile Methods

UQ. 1.12.1 Describe Agile Manifesto. SPPU - Aug. 17, 4 Marks

UQ. 1.12.2 What is agility? Explain about Agile Process Model. SPPU - Dec. 17, 5 Marks

UQ. 1.12.3 What is important of Agile/XP methodology for project development ? SPPU - Dec. 18, 5 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.

Syllabus Topic : Plan-driven and Agile Development

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. 1.12.4 Explain principles of Agile Methodology.
SPPU - May 15, 4 Marks

LQ. 1.12.5 Describe and discuss characteristics of Agile Process Model. **(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

LQ. 1.12.6 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

LQ. 1.12.7 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).

Syllabus Topic : Extreme Programming (XP) Practices

1.13 Extreme Programming (XP) Practices

UQ. 1.13.1 Explain how extreme programming process supports agility with its framework activities?

SPPU - May 18, 5 Marks

- Extreme Programming (XP) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements.

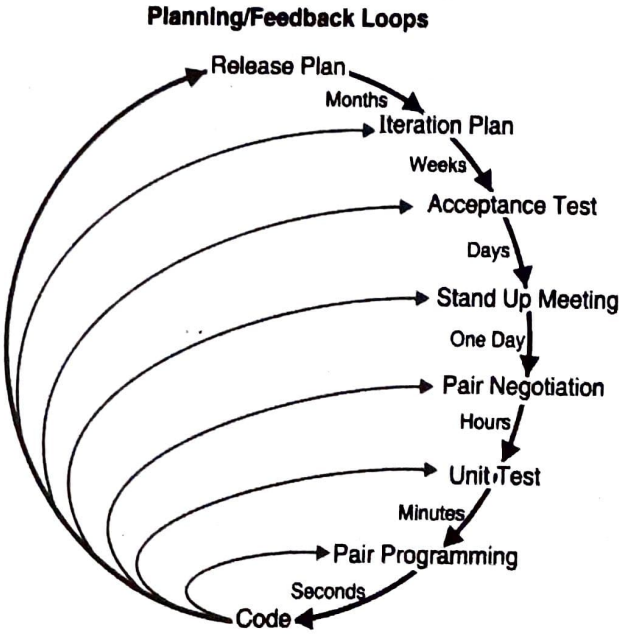


Fig. 1.13.1 : Extreme Programming

- 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.
- 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.

1.13.1 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.

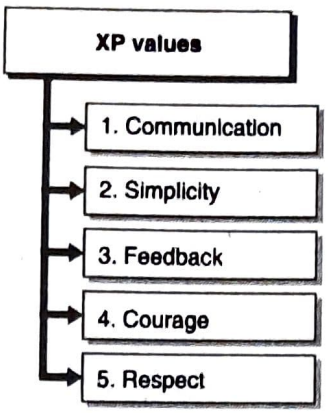


Fig. 1.13.2 : XP values

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.

Syllabus Topic : Testing in XP

1.14 Testing in XP

LQ. 1.14.1 Write note on "Testing in XP". (5 Marks)

- XP testing is different in many ways from 'traditional' testing. The biggest difference is that on an XP project, the entire development team takes responsibility for quality.
- This means the whole team is responsible for all testing tasks, including acceptance test automation. When testers and programmers work together, the approaches to test automation can be pretty creative!
- Testing is an integrated activity on an XP team. The development team needs continual feedback, with the customer expressing their needs in terms of tests, and programmers expressing design and code in terms of tests.
- On an XP team, the tester will play both the customer and programmer 'roles'. He'll focus on acceptance testing and work to transfer testing and quality assurance skills to the rest of the team.

XP Tester Activities

Here are some activities testers perform on XP teams.

1. Negotiate quality with the customer (it's not the standard of quality, it's what the customer desires and is willing to pay for!).

2. Clarify stories, flush out hidden assumptions.
3. Enable accurate estimates for both programming and testing tasks.
4. Make sure the acceptance tests verify the quality specified by the customer
5. Help the team automate tests
6. Help the team produce testable code
7. Form an integral part of the continuous feedback loop that keeps the team on track.

The Nature of XP Testing

- The biggest difference between XP projects and most 'traditional' software development projects is the concept of test-driven development.
- With XP, every chunk of code is covered by unit tests, which must all pass all the time.
- The absence of unit-level and regression bugs means that testers actually get to focus on their job: making sure the code does what the customer wanted.
- The acceptance tests define the level of quality the customer has specified (and paid for!)
- Testers who are new to XP should keep in mind the XP values: communication, simplicity, feedback and courage.
- Courage may be the most important. As a tester, the idea of writing, automating and executing tests in speedy two or three week iterations, without the benefit of traditional requirements documents, can be daunting.
- Testers need courage to let the customers make mistakes and learn from them. They need courage to determine the minimum testing that will prove the successful completion of a story.
- They need courage to ask their teammates to pair for test automation. They need courage to remind the team that we are all responsible for quality and testing.

Syllabus Topic : Pair Programming

1.15 Pair Programming

LQ. 1.15.1 Explain Pair Programming. (5 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.

1.15.1 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.

1.15.2 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.

Syllabus Topic : Introduction to Agile Tools : JIRA

1.16 Introduction to Agile Tools : JIRA

LQ. 1.16.1 Write note on JIRA. **(5 Marks)**

- JIRA is a web-based "Issue tracking system" or "Bug tracking system". It is mainly used for agile project management. JIRA is a proprietary based tool, developed by Atlassian. The product name 'JIRA' is shortened from the word 'Gojira', which means Godzilla in Japanese.
- JIRA helps us to manage the project effectively and smoothly. It is a powerful tool to track the issues, bugs, backlogs of the project. JIRA is more customizable than Bugzilla.
- It helps the team to strive hard towards the common goal. JIRA is widely used by many organizations around the world.

Key features of JIRA includes

1. Issue tracking
2. Scrum boards
3. Project planning
4. Project tracking
5. Reporting
6. Notifications
7. Advantages of JIRA
8. Improves collaboration
9. Improves tracking
10. Better planning
11. Increase productivity
12. Improves customer satisfaction
13. Flexible to use

Syllabus Topic : Kanban Model

1.17 Kanban Model

LQ. 1.17.1 Explain the Kanban Model in detail. **(5 Marks)**

- Kanban is a popular framework used to implement agile software development. It requires real-time communication of capacity and full transparency of work.
- Work items are represented visually on a kanban board, allowing team members to see the state of every piece of work at any time.
- Kanban is enormously prominent among today's agile software teams, but the kanban methodology of work dates back more than 50 years.
- In the late 1940s Toyota began optimizing its engineering processes based on the same model that supermarkets were using to stock their shelves.

- Supermarkets stock just enough products to meet consumer demand, a practice that optimizes the flow between the supermarket and the consumer.
- Because inventory levels match consumption patterns, the supermarket gains significant efficiency in inventory management by decreasing the amount of excess stock it must hold at any given time. Meanwhile, the supermarket can still ensure that the given product a consumer needs is always in stock.
- When Toyota applied this same system to its factory floors, the goal was to better align their massive inventory levels with the actual consumption of materials.
- To communicate capacity levels in real-time on the factory floor (and to suppliers), workers would pass a card, or "kanban", between teams.
- When a bin of materials being used on the production line was emptied, a kanban was passed to the warehouse describing what material was needed, the exact amount of this material, and so on.
- The warehouse would have a new bin of this material waiting, which they would then send to the factory floor, and in turn send their own kanban to the supplier.
- The supplier would also have a bin of this particular material waiting, which it would ship to the warehouse. While the signaling technology of this process has evolved since the 1940s, this same "just in time" (or JIT) manufacturing process is still at the heart of it.

1.17.1 Kanban for Software Teams

- Agile software development teams today are able to leverage these same JIT principles by matching the amount of work in progress (WIP) to the team's capacity.
- This gives teams more flexible planning options, faster output, clearer focus, and transparency throughout the development cycle.
- While the core principles of the framework are timeless and applicable to almost any industry, software development teams have found particular success with the agile practice.
- In part, this is because software teams can begin practicing with little to no overhead once they understand the basic principles.
- Unlike implementing kanban on a factory floor, which would involve changes to physical processes and the addition of substantial materials, the only physical things a software teams need are a board and cards, and even those can be virtual.

1.17.2 Kanban Boards

- The work of all kanban teams revolves around a kanban board, a tool used to visualize work and optimize the flow of the work among the team.
- While physical boards are popular among some teams, virtual boards are a crucial feature in any agile software development tool for their traceability, easier collaboration, and accessibility from multiple locations.
- Regardless of whether a team's board is physical or digital, their function is to ensure the team's work is visualized, their workflow is standardized, and all blockers and dependencies are immediately identified and resolved.
- A basic kanban board has a three-step workflow: To Do, In Progress, and Done. However, depending on a team's size, structure, and objectives, the workflow can be mapped to meet the unique process of any particular team.
- The kanban methodology relies upon full transparency of work and real-time communication of capacity, therefore the kanban board should be seen as the single source of truth for the team's work.

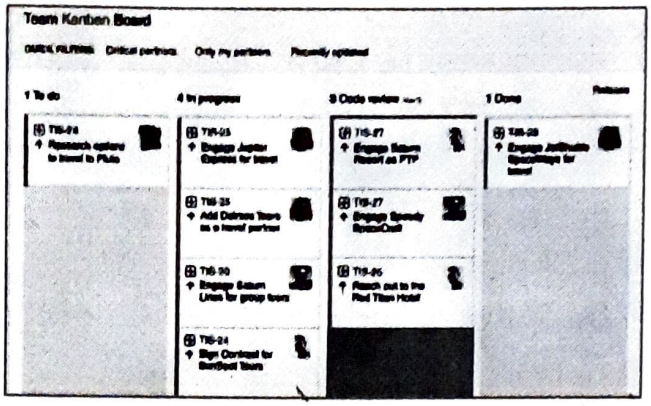


Fig. 1.17.1 : Kanban Board

1.17.3 Kanban Cards

- In Japanese, kanban literally translates to "visual signal." For kanban teams, every work item is represented as a separate card on the board.
- The main purpose of representing work as a card on the kanban board is to allow team members to track the progress of work through its workflow in a highly visual manner.
- Kanban cards feature critical information about that particular work item, giving the entire team full visibility into who is responsible for that item of work, a brief description of the job being done, how long that piece of work is estimated to take, and so on.
- Cards on virtual kanban boards will often also feature screenshots and other technical details that are valuable to the assignee.
- Allowing team members to see the state of every work item at any given point in time, as well as all of the associated details, ensures increased focus, full traceability, and fast identification of blockers and dependencies.

1.17.4 Advantages of Kanban

- Kanban is one of the most popular software development methodologies adopted by agile teams today.
- Kanban offers several additional advantages to task planning and throughput for teams of all sizes :

Advantages of Kanban

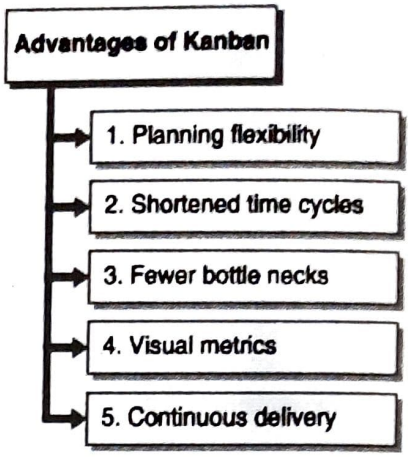


Fig. 1.17.2 : Advantages of Kanban

1. Planning flexibility

- A kanban team is only focused on the work that's actively in progress. Once the team completes a work item, they pluck the next work item off the top of the backlog.
- The product owner is free to reprioritize work in the backlog without disrupting the team, because any changes outside the current work items don't impact the team.
- As long as the product owner keeps the most important work items on top of the backlog, the development team is assured that they are delivering maximum value back to the business. So there's no need for the fixed-length iterations that are found in scrum.

2. Shortened time cycles

- Cycle time is a key metric for Kanban teams. Cycle time is the amount of time it takes for a unit of work to travel through the team's workflow - from the moment work starts to the moment it ships.
- By optimizing cycle time, the team can confidently forecast the delivery of future work.
- Overlapping skill sets lead to smaller cycle times. When only one person holds a skill set, that person becomes a bottleneck in the workflow. So teams employ basic best practices like code review and mentoring help to spread knowledge.

- Shared skills mean that team members can take on heterogeneous work, which further optimizes cycle time. It also means that if there is a backup of work, the entire team can swarm on it to get the process flowing smoothly again. For instance, testing isn't only done by QA engineers. Developers pitch in, too.
- In a Kanban framework, it's the entire team's responsibility to ensure work is moving smoothly through the process.

3. Fewer bottlenecks

- Multitasking kills efficiency. The more work items in flight at any given time, the more context switching, which hinders their path to completion.
- That's why a key tenant of Kanban is to limit the amount of work in progress (WIP). Work-in-progress limits highlight bottlenecks and backups in the team's process due to lack of focus, people, or skill sets.
- For example, a typical software team might have four workflow states: To Do, In Progress, Code Review, and Done.
- They could choose to set a WIP limit of 2 for the code review state. That might seem like a low limit, but there's good reason for it: developers often prefer to write new code, rather than spend time reviewing someone else's work.
- A low limit encourages the team to pay special attention to issues in the review state, and to review others work before raising their own code reviews. This ultimately reduces the overall cycle time.

4. Visual metrics

- One of the core values is a strong focus on continually improving team efficiency and effectiveness with every iteration of work.
- Charts provide a visual mechanism for teams to ensure they're continuing to improve. When the team can see data, it's easier to spot bottlenecks in the process (and remove them).
- Two common reports Kanban teams use are control charts and cumulative flow diagrams.
- A control chart shows the cycle time for each issue as well as a rolling average for the team.

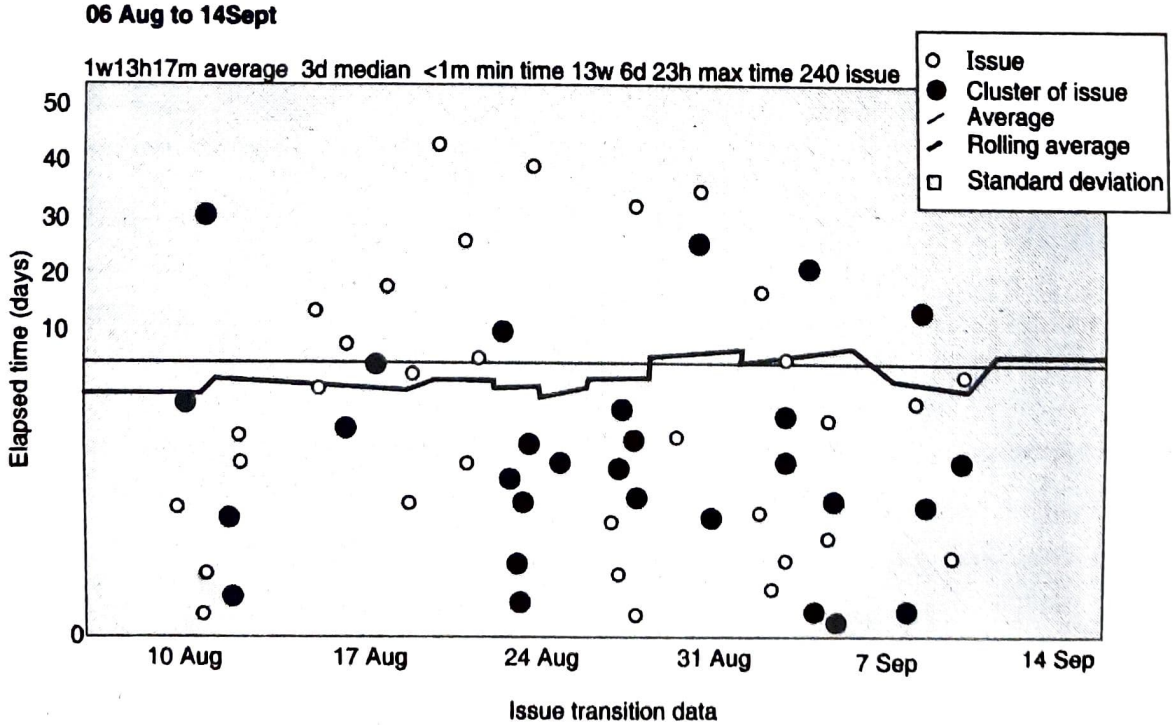


Fig. 1.17.3 : Control Chart

- A cumulative flow diagram shows the number of issues in each state. The team can easily spot blockages by seeing the number of issues increase in any given state. Issues in intermediate states such as "In Progress" or "In Review" are not yet shipped to customers, and a blockage in these states can increase the likelihood of massive integration conflicts when the work does get merged upstream.

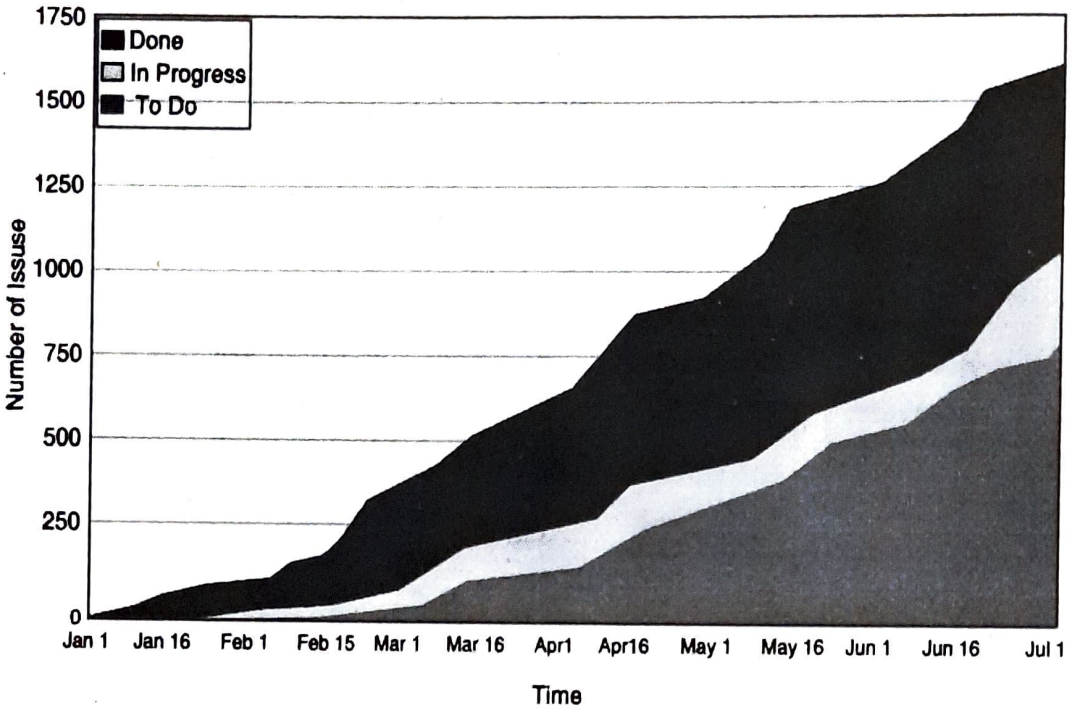


Fig. 1.17.4 : Cumulative Flow Diagram

5. Continuous Delivery

- We know that continuous integration the practice of automatically building and testing code incrementally throughout the day is essential for maintaining quality. Now it's time to meet continuous delivery (CD).
- CD is the practice of releasing work to customers frequently even daily or hourly.
- Kanban and CD beautifully complement each other because both techniques focus on the just-in-time (and one-at-a-time) delivery of value.
- The faster a team can deliver innovation to market, the more competitive their product will be in the marketplace. And Kanban teams focus on exactly that: optimizing the flow of work out to customers.

Syllabus Topic : Case Studies : An Information system (mental health-care system) / Case study - Mental health care patient management system (MHC-PMS)

1.18 Case Studies : An Information System (Mental Health-care System) / Case Study - Mental Health Care Patient Management System (MHC-PMS)

LQ. 1.18.1 Write Case study on "MHC-PMS". **(5 Marks)**

- A patient information system to support mental health care is considered as a medical information system which maintains information regarding patients who are suffering from mental health problems and the treatments received by them.

- Most of the times, the mental health patients do not need dedicated hospital treatment, rather required to attend specialist clinics often where they can meet a doctor having detailed knowledge of their problems.
- To make it simple for patients to attend, these clinics are not just run in hospitals. They may also be held in local medical practices or community centres.
- The MHC-PMS (Mental Health Care-Patient Management System) is an information system which is basically intended for use in clinics.
- It takes reference of a centralized database system of patient information but has also been designed to run on a single computer, in order that it may be accessed and used from sites which without secure network connectivity.
- When there is secure network access in local systems, they use patient information in the database but it is also possible for them to download and use local copies of patient records when they are not connected.

1.18.1 MHC-PMS Goals

1. To produce management information which helps the health service managers in assessing performance against local and government targets.
2. To give medical staff with timely data for the purpose of supporting the treatment of patients

The Organization of the MHC-PMS

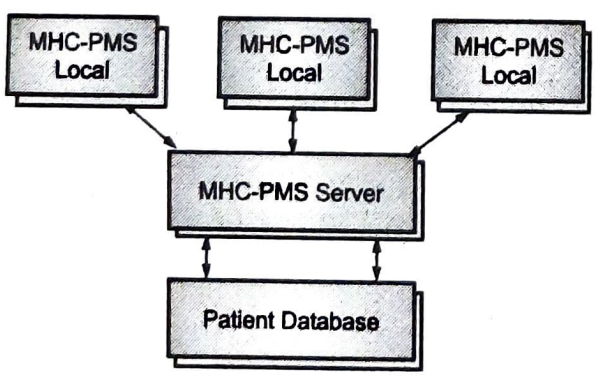


Fig. 1.18.1

1.18.2 MHC-PMS Key Features

- Individual care management – Clinicians are able to generate records regarding patients, update the information in the respective system, see patient history, etc. The system has an efficient supports for data summaries because of which doctors are able to quickly learn about the key issues and treatments which have been prescribed.
- Patient monitoring - The system is able to monitor the records of patients which are concerned with treatment and issues warnings if there is detection of possible problems.
- Administrative reporting - Monthly management reports is generated by the system which shows the total number of patients treated at every clinic, the number of patients who have entered and left the care system, number of patients sectioned, the drugs prescribed and their costs, etc.

1.18.3 MHC-PMS Concerns

- Privacy - It is necessary that patient information is confidential and is never open to anyone beside authorised medical staff and the patient themselves.
- Safety - Some type of mental illnesses may cause patients to become suicidal or a threat to other people. Wherever it is possible, the system should be able to warn medical staff regarding potentially suicidal or dangerous patients.
- It is very important that the system must be available whenever there is need else safety may be compromised and it may be impossible to prescribe the accurate medication to the respective patients.

Syllabus Topic : Case Studies : Wilderness Weather System

1.19 Case Studies : Wilderness Weather System

LQ. 1.19.1 Write Case study on "Wilderness Weather System". **(5 Marks)**

- The government of a nation having large areas of wilderness makes a decision to deploy several hundred weather stations in remote areas.
- Weather stations gather information from a set of instruments which measure temperature and pressure, sunshine, rainfall, wind speed and wind direction.
- The weather station has several instruments which are able to measure weather parameters such as the wind speed and direction, the ground and air temperatures, the barometric pressure and the rainfall over a 24-hour period.
- All those instruments are controlled by a software system which accepts parameter readings periodically and manages the data gathered from the instruments.

1.19.1 The Weather Station's Environment

- Fig. 1.19.1 illustrates the environment of a Weather Staion.

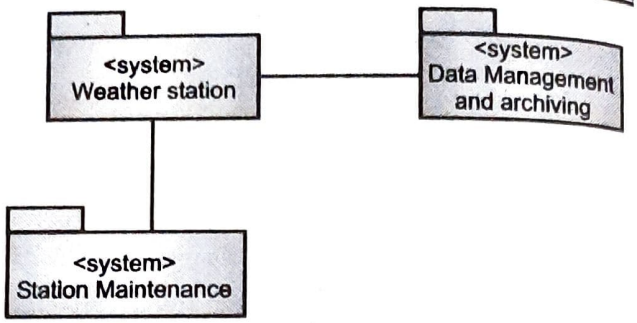


Fig. 1.19.1 : Environment of a Weather Staion

1.19.2 Weather Information System

- The weather station system - This is responsible for collecting weather data, carrying out some initial data processing and transmitting it to the data management system.
- The data management and archiving system - This system collects the data from all of the wilderness weather stations, carries out data processing and analysis and archives the data.
- The station maintenance system - This system can communicate by satellite with all wilderness weather stations to monitor the health of these systems and provide reports of problems.

...Chapter Ends

□□□

Software Requirements Engineering and Analysis

University Prescribed Syllabus

Requirements Engineering : User and system requirements, Functional and non-functional requirements, Types & Metrics, A spiral view of the requirements engineering process.

Software Requirements Specification (SRS) : The software requirements Specification document, The structure of SRS, Ways of writing a SRS, structured & tabular SRS for an insulin pump case study,

Requirements elicitation and Analysis : Process, Requirements validation, Requirements management.

Case Studies : The information system. Case study - Mental health care patient management system (MHC-PMS).

2.1 Requirement Engineering

UQ. 2.1.1 Explain the importance of requirement engineering. **SPPU - Aug. 17. 2 Marks**

☞ **Requirement** The information which describes the user's expectation about the system performance is called as requirement.

- The requirement must be clear and unambiguous. Some requirement definition has to face problem of lack of clarity.

☞ Characteristics of requirements

- There are various characteristic of requirements. They are as follows :

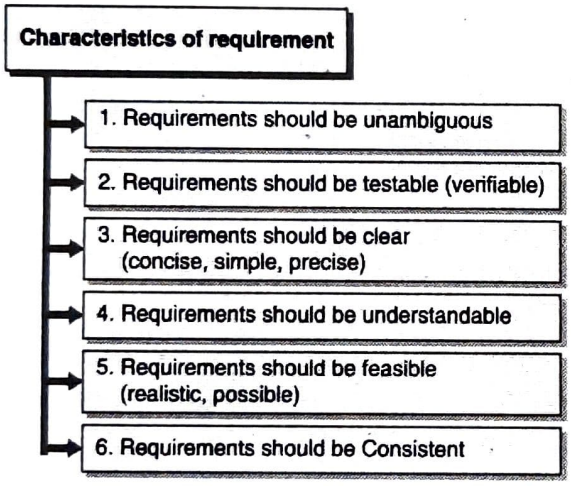


Fig. 2.1.1 : Characteristics of requirements

1. Requirements should be unambiguous

- Ambiguous means the single word or statement has more than one meaning. If requirements contain ambiguity then it is difficult to fulfill the requirements correctly.
- Therefore the requirements should be unambiguous means non confusing.

2. Requirements should be testable (verifiable)

The requirements should be testable means tester should be able to easily test the requirement whether they are implemented successfully or not. For easy test, the requirement should be clear and unambiguous.

3. Requirements should be clear (concise, simple, precise)

- The requirements should be clear. They should not contain any unnecessary information.
- If there is any unnecessary information then it becomes difficult to achieve the appropriate fulfillment of the requirement.

4. Requirements should be understandable

- The requirements should be understandable means when anyone read it then it should be easily understood by that person. To make it understandable proper conventions are used.
- It should be grammatically correct.

5. Requirements should be feasible (realistic, possible)

- The requirement should be completed within the given time and budget.
- Specified requirement said to be feasible if it can be implemented using existing technology, with estimated budget and time.

6. Requirements should be Consistent

- Consistency is an important feature of requirements. It means that all inputs must be processed similarly.
- It should not happen that processes produce different outputs for same inputs coming from different sources.

☞ Requirement Engineering

- The procedure which collects the software requirements from customer, analyze and document them is called as **requirement engineering**.
- The aim of requirement engineering is to create and maintain 'System Requirements Specification' document.

- Requirements engineering is the process of understanding and defining which services are required and identifying the constraints on these services.
- Requirement engineering processes ensures your software will meet the user expectations and ending up with high quality software.
- It's a critical stage of the software process as errors at this stage will reflect later on in the next stages, which definitely will cause you higher costs.
- At the end of this stage, a requirement document that specifies the requirements will be produced and validated with the stakeholders.

2.1.1 Activities Involved in Requirements Engineering

UQ. 2.1.2 List the tasks involved in requirement engineering. **SPPU - Aug. 17, 2 Marks**

UQ. 2.1.3 What are the requirement engineering tasks ? Explain in detail. **SPPU - Dec. 17, 5 Marks**

- The **activities/tasks** involved **in requirements engineering** vary widely, depending on the type of system being developed and the specific practices of the organization(s) involved.
- These may include :

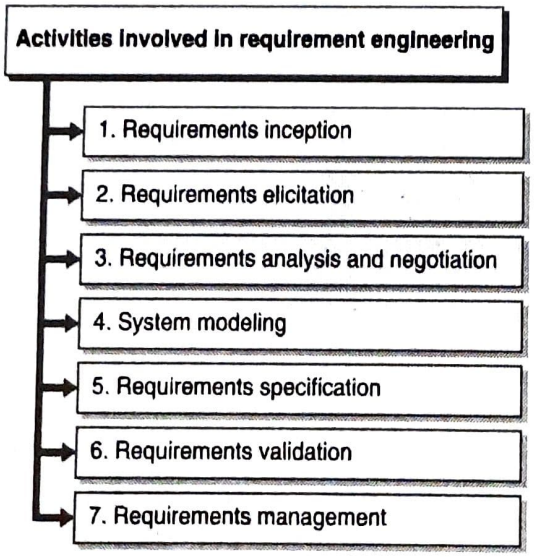


Fig. 2.1.2 : Activities involved in requirements engineering

1. Requirements Inception

- Inception is a task where the requirement engineering asks a set of questions to establish a software process.
- In this task, it understands the problem and evaluates with the proper solution.
- It collaborates with the relationship between the customer and the developer.
- The developer and customer decide the overall scope and the nature of the question.

2. Requirements Elicitation

- In requirements engineering, requirements elicitation is the practice of researching and discovering the requirements of a system from users, customers, and other stakeholders. The practice is also sometimes referred to as "requirement gathering".
- We will see more details regarding requirement gathering in section 2.2.

3. Requirements Analysis and Negotiation

- Requirements are identified (including new ones if the development is iterative) and conflicts with stakeholders are solved. Both written and graphical tools (the latter commonly used in the design phase but some find them helpful at this stage, too) are successfully used as aids.
- Examples of written analysis tools : use cases and user stories.
- Examples of graphical tools : UML and LML.

4. System modelling

- Some engineering fields (or specific situations) require the product to be completely designed and modeled before its construction or fabrication starts and, therefore, the design phase must be performed in advance.
- For instance, blueprints for a building must be elaborated before any contract can be approved and signed.

- Many fields might derive models of the system with the Lifecycle Modeling Language, whereas others might use UML.

Note : In many fields, such as software engineering, most modeling activities are classified as design activities and not as requirement engineering activities.

5. Requirements specification

- Requirements are documented in a formal artifact called Requirements Specification (RS).
- Nevertheless, it will become official only after validation. A RS can contain both written and graphical (models) information if necessary.
- **Example :** Software Requirements Specification (SRS).

6. Requirements validation

It is the process of checking whether the documented requirements and models are consistent and meet the needs of the stakeholder. Only if the final draft passes the validation process, the RS becomes official.

7. Requirements management

Managing all the activities related to the requirements since inception, supervising as the system is developed and, even until after it is put into use (e.g., changes, extensions, etc.)

2.1.2 Difference between Requirement Inception and Requirement Elicitation

UQ. 2.1.4 What is the difference between Requirement Inception and Requirement Elicitation?

SPPU - Aug. 17, 2 Marks

Parameter	Requirement Inception	Requirement Elicitation
Concept	Inception is a task where the requirement engineering asks a set of questions to establish a software process.	In requirements engineering, requirements elicitation is the practice of researching and discovering the requirements of a system from users, customers, and other stakeholders.
Function	In this task, it understands the problem and evaluates with the proper solution. It collaborates with the relationship between the customer and the developer. The developer and customer decide the overall scope and the nature of the question.	This is basically function of requirement gathering.

Syllabus Topic : User and System Requirements

2.2 User and System Requirements

UQ. 2.2.1 Identify and briefly explain four types of requirements that may be defined foe computer system.

SPPU - May 18, 4 Marks

- User requirements, often referred to as user needs, describe what the user does with the system, such as what activities that users must be able to perform.
- User requirements are generally documented in a User Requirements Document (URD) using narrative text. User requirements are generally signed off by the user and used as the primary input for creating system requirements.
- An important and difficult step of designing a software product is determining what the user actually wants it to do.

- This is because the user is often not able to communicate the entirety of their needs and wants, and the information they provide may also be incomplete, inaccurate and self-conflicting.
- The responsibility of completely understanding what the customer wants falls on the business analyst. This is why user requirements are generally considered separately from system requirements.
- The business analyst carefully analyzes user requirements and carefully constructs and documents a set of high quality system requirements ensuring that that the requirements meet certain quality characteristics.
- System requirements are the building blocks developers use to build the system. These are the traditional "shall" statements that describe what the system "shall do."
- System requirements are classified as either functional or supplemental requirements. A functional requirement specifies something that a user needs to perform their work.
- For example, a system may be required to enter and print cost estimates; this is a functional requirement.
- Supplemental or non-functional requirements specify all the remaining requirements not covered by the functional requirements.

Syllabus Topic : Functional Requirements

2.3 Functional Requirements

UQ. 2.3.1 What are functional requirements of software? **SPPU - Aug. 17, 2 Marks**

UQ. 2.3.2 Identify and briefly explain four types of requirements that may be defined for computer system. **SPPU - May 18, 4 Marks**

- The Functional requirements specification create document regarding the operations as well as activities which a system must be able to carry out.
- In software engineering, a functional requirement is used to define a function regarding a system or its component, in which a function is described as a specification of behavior between outputs and inputs.
- In the functional requirements there may be elements such as calculations, technical details, data manipulation and processing, and other specific functionality which define what a system is supposed to implement.
- As defined in requirements engineering, functional requirements is used to mention specific results of a system.
- This should be contrasted with non-functional requirements that mention overall characteristics like as cost and reliability.
- Functional requirements drive the application architecture of a system, while non-functional requirements drive the technical architecture of a system.

☞ Functional Requirements should include

- Descriptions of data to be entered into the system.
- Descriptions of operations performed by each screen.
- Descriptions of work-flows performed by the system.
- Descriptions of system reports or other outputs.
- Persons who can enter the data into the system.
- Way the system meets applicable regulatory requirements.

☞ Examples of Functional Requirements

- Functional requirements must contain functions performed by specific screens, outlines of work-flows performed by the system, and other business or compliance requirements the system must meet.

☞ Interface requirements

- Field 1 accepts numeric data entry.
- Field 2 only accepts dates before the current date.
- Screen 1 can print on-screen data to the printer.

Business Requirements

- Data must be entered before a request can be approved.
- Clicking the Approve button moves the request to the Approval Workflow.
- All personnel using the system will be trained.

Regulatory/Compliance Requirements

- The database will have a functional audit trail.
- The system will limit access to authorized users.
- The spreadsheet can secure data with electronic signatures.

Security Requirements

- Members of the Data Entry group can enter requests but cannot approve or delete requests.
- Members of the Managers group can enter or approve a request but cannot delete requests.
- Members of the Administrators group cannot enter or approve requests but can delete requests.

Syllabus Topic : Non Functional Requirements

2.4 Non Functional Requirements (NFR)

UQ. 2.4.1 What are non-functional requirement of software? **SPPU - Aug. 17, 2 Marks**

UQ. 2.4.2 Identify and briefly explain four types of requirements that may be defined for computer system. **SPPU - May 18, 4 Marks**

- These are basically the quality constraints that the system must satisfy according to the project contract.
- The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements. They basically deal with issues like :
  - Portability
  - Maintainability
  - Scalability
  - Reusability
  - Security
  - Reliability
  - Performance
  - Flexibility
- NFRs are classified into following types :
  - Interface constraints
  - Performance constraints : response time, security, storage space, etc.
  - Operating constraints
  - Life cycle constraints : maintainability, portability, etc.
  - Economic constraints
- The process of specifying non-functional requirements needs the knowledge of the functionality of the system, and also the knowledge of the context within which the system will operate.

Syllabus Topic : Types and Metrics

2.4.1 Types and Metrics

Types of Non-Functional Requirements

UQ. 2.4.3 Identify and briefly explain four types of requirements that may be defined for computer system. **SPPU - May 18, 4 Marks**

Fig. 2.4.1 : Types of non-functional requirements

- **Product requirements :** Requirements which mention that the delivered product should have behavior in a specific way, e.g. execution speed, reliability etc.
- **Organizational requirements :** Requirements that are a result of organizational policies as well as procedures, such as process standards which has been used, implementation necessities etc.
- **External requirements :** Requirements which are generated from the factors that are external to the system and its development process, e.g. interoperability requirements, legislative requirement.

Metrics for specifying non-functional requirements

- Property Measure - Speed Processed transactions/second User/event response time Screen refresh time
- Size Mbytes - Number of ROM chips
- Ease of use - Training time Number of help frames.
- Reliability - Mean time to failure Probability of unavailability Rate of failure occurrence Availability
- Robustness - Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
- Portability - Percentage of target dependent statements, Number of target systems

2.5 Domain Requirements

UQ. 2.5.1 Identify and briefly explain four types of requirements that may be defined for computer system. **SPPU - May 18, 4 Marks**

- Domain requirements are used to describe system characteristics as well as features which impact the domain. Those may be new functional requirements, constraints on present requirements or may define particular computations.
- If domain requirements are not satisfied, the system may be unworkable.
- **Example :** Library system.

Syllabus Topic : A Spiral View of Requirements Engineering Process

2.6 A Spiral View of Requirements Engineering Process

Following figure demonstrates a Spiral View of Requirements Engineering Process :

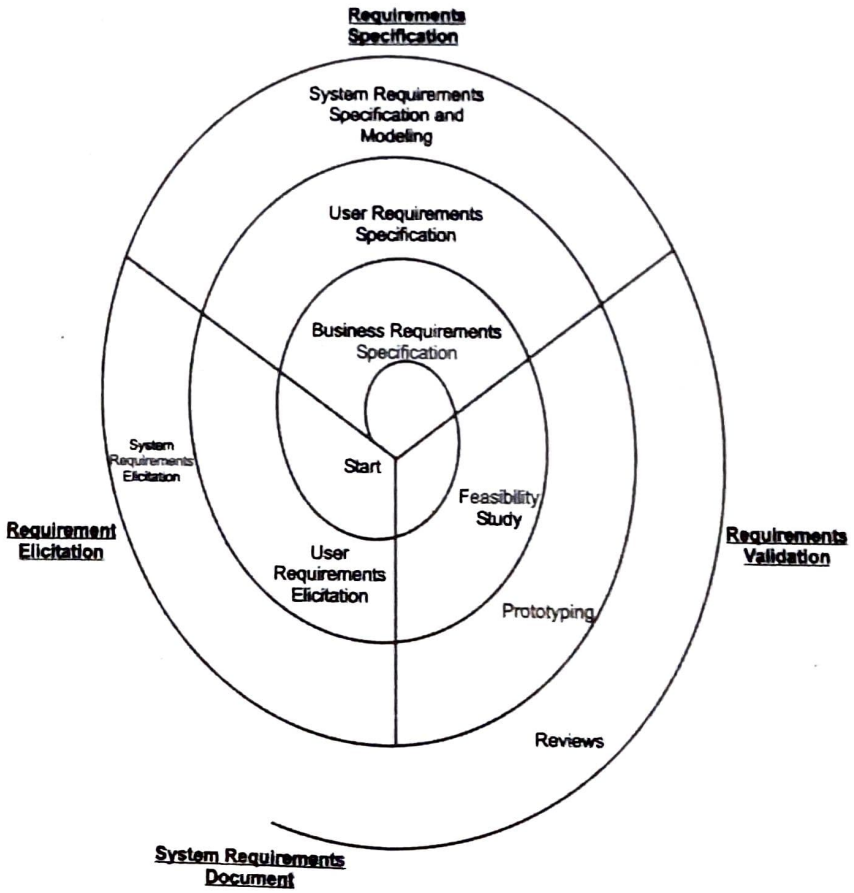


Fig. 2.6.1 : A spiral view of the requirements engineering process

- In requirements engineering processes there are mainly 4 high-level activities. These are in general concentrating on assessing whether the system is constructive to the business (feasibility study), determining requirements (elicitation and analysis), converting the respective requirements into any standard form (specification), and verifying that the requirements in reality define the system which the customer (end user) wants (validation).
- Practically, requirements engineering is considered as an iterative process where the activities are interleaved.
- This interleaving is illustrated in Fig 2.6.1. The respective activities are arranged as an iterative process around a spiral, in which the output is considered as a system requirements document.
- The amount of time as well as effort required in every activity in each iteration is based on the stage of the overall process and the type of the software system which is being developed.
- In the beginning of the process, most of the effort will be exhausted on understanding high-level business as well as non-functional requirements, and the user requirements for the system.
- In the next phase, in the outer rings of the spiral, more effort will be exhausted on eliciting and understanding system requirements in detail.
- This spiral model covers approaches to development in which the respective requirements are developed to diverse levels of detail.
- There may be variation in number of iterations around the spiral because of which the spiral can be exited after elicitation of some or all of the user requirements.
- It is possible to use Agile development instead of prototyping because of which it becomes easy to develop the requirements and the system implementation together.
- Sometimes requirements engineering is considered as the process of applying a structured analysis method. It includes analyzing the system and designing a several graphical system models, for example use case models, which then can be further used as a system specification.
- The set of several different models illustrates the behavior of the system and is interpreted with extra information describing, for example, the system's expected performance or reliability.
- Even if structured methods have a significant role to play in the process of requirements engineering, there is much more to requirements engineering than is covered by these methods.

Syllabus Topic : Software Requirement Specification (SRS) : Software Requirement Specification Document

2.7 Software Requirement Specification (SRS) : Software Requirement Specification Document

LQ. 2.7.1 What is SRS document ? (4 Marks)

- SRS stands for Software/System Requirement Specification. SRS is a special kind of document which contains user requirements for a system which states properties and constraints that must be satisfied by a software system.
- IEEE defines software requirements specification as, 'a document that clearly and precisely describes each of the essential requirements (functions, performance, design constraints and quality attributes) of the software and the external interfaces.
- The outcome of the requirements gathering and analysis phase of the SDLC is **Software Requirements Specification (SRS).**
- SRS is also called as **requirements document.**
- SRS is base for software engineering actions and is generated when all requirements of software project are gathered and analyzed.

- It is generally signed at the end of requirements engineering phase.
- Following are six requirements stated by Heninger, which SRS document should follow :
  1. SRS document should specify external system behaviour.
  2. SRS document should specify implementation constraints.
  3. It should be easily changeable if any changes occur.
  4. It should act as reference tool for maintaining the system.
  5. SRS document record forethought about the lifecycle of the system i.e. predicts changes.
  6. It must include acceptable response to undesired events.

2.7.1 Advantages of SRS

LQ. 2.7.2 What are advantages of SRS ? (4 Marks)

(1) SRS contains the base for agreement among the client and the company who develop the product as per expectations.

(2) A SRS gives a base for verification of the completely developed product.

(3) Using high-quality SRS, we can develop high-quality software product.

(4) A high-quality SRS decreases the cost of development.

2.7.2 Characteristics of SRS

UQ. 2.7.3 What are the characteristics of good SRS ?
SPPU - Aug. 17, 3 Marks

A good software requirement specification must satisfy following characteristics.

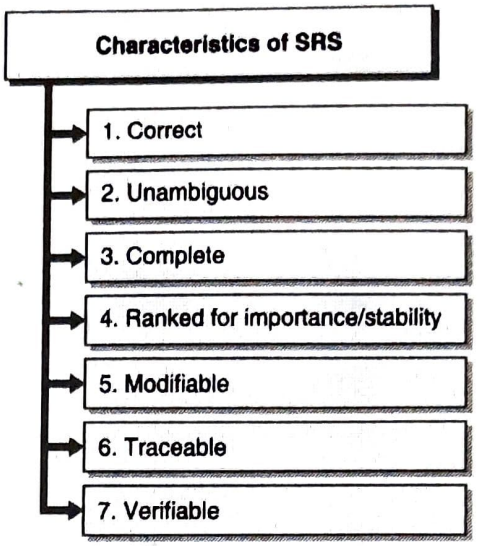


Fig. 2.7.1 : Characteristics of SRS

1. Correct

- SRS is correct when all requirements of user are described in the requirement document.
- The listed requirements must be matched with desired system.
- This depict that every requirement is analyzed to give assurance that it (SRS) contains user requirements.
- Remember that there is no specific tool or process to ensure the correctness of SRS.
- Correctness gives assurance that all stated requirements are worked as expected.

2. Unambiguous

- SRS does not contain any confusion when each specified requirement has single interpretation.
- This characteristic state that every requirement is individually interpreted.
- In situation, where one term has number of meanings, then its meaning must be specified in the SRS so that it will be non-confusing and simple to understand.

3. Complete

- SRS is complete when the requirements undoubtedly specified that which work the software product is required to perform.

- This contains each and every requirement associated to performance, design and functionality.

4. Ranked for importance/stability

- Each and every requirement has not same importance, so every requirement is recognized to make differentiation between requirements.
- For this purpose, it is needed to undoubtedly recognize every requirement.
- Stability refers to the probability of further modifications in the requirement.

5. Modifiable

- The requirements given by user are changeable, so requirement document must be generated in such a way that those modifications can be included easily in SRS by preserving consistently the structure and style of the SRS.

6. Traceable

- SRS is observable when the source of every requirement is unambiguous and facilitates the description of every requirement in future.
- Forward tracing and backward tracing methods are used for this purpose.
- Forward tracing state that every requirement must be referencing to design and code of components.
- Backward tracing state every requirement explicitly addressing its source.

7. Verifiable

- SRS is testable when the stated requirements can be tested with a cost-effective procedure to verify whether the final software fulfills those requirements.
- The requirements are tested through reviews.
- Remember that clear requirement is needed for verifiability.

Syllabus Topic : Structure of SRS

2.7.3 Structure of SRS

LQ. 2.7.4 Explain general format of Software Requirement Specification (SRS). **(5 Marks)**

- Structure of SRS document includes following sections with various subsections in it :

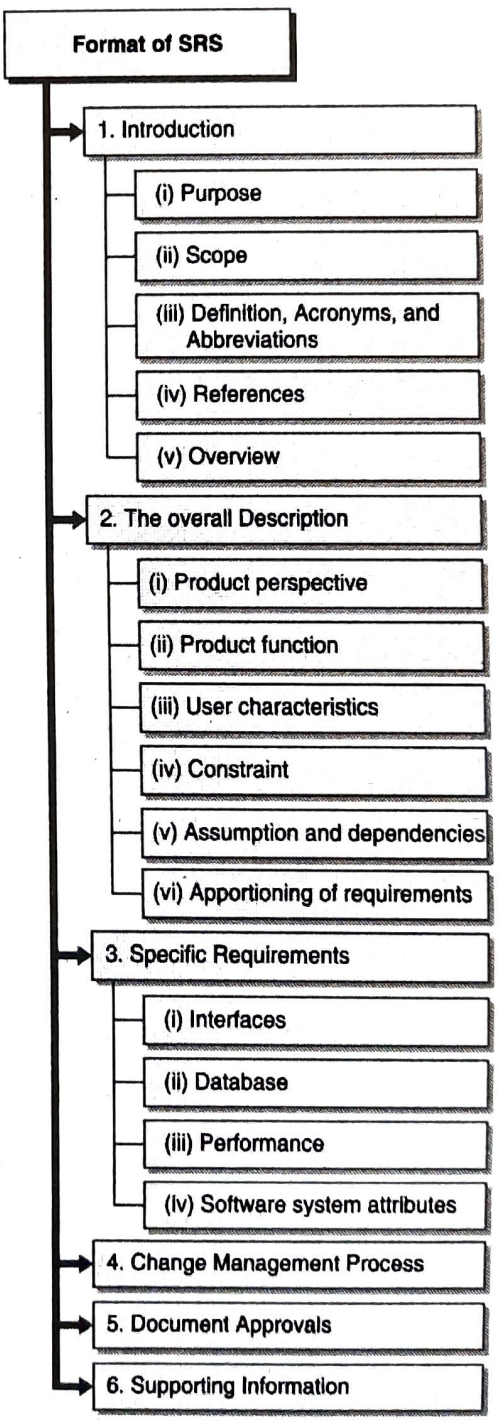


Fig. 2.7.2 : Format of SRS

1. Introduction

Introduction contains brief information about the SRS. This part is further divided into following subsections :

(i) Purpose

This section specifies the main purpose of SRS document with its intended audience for whom SRS is constructed.

(ii) Scope

- This section specifies scope of the system for which SRS is built.
- Scope defines not only benefits, objectives and behaviour of software product but also limitations of software so as to understand the boundary of software.

(iii) Definition, Acronyms, and Abbreviations

To avoid ambiguity of terms specified in requirements, SRS provides definition, abbreviation and acronyms about these terms.

(iv) References

It contains list of all documents which are referred by this document.

(v) Overview

It gives overview of the document including its goals and objectives of the system.

2. The overall Description

- It provides overall information about the requirements. Customers/users of the system are concerned about this section. It contains following subsections :

(i) Product perspective

This section contains the information which states the benefits of current product over other existing products.

(ii) Product function

- The functionality of software product summarizes in this section of SRS document.
- It is possible to use diagrams to summarize the software functionality and logical relationship among variables present in the system. This information is provided in simple way so that users of the system can understand.

(iii) User characteristics

To use any system, it is mandatory that users of system should have some basic knowledge i.e. at least the educational qualification or basic knowledge related to the field. This section provides the criteria about the users of the system.

(iv) Constraint

It includes limitations of components used in the system for example, hardware limitations.

(v) Assumption and dependencies

It contains the list of factors on which SRS is depending. That is if the factor changes it leads to change in the SRS too.

(vi) Apportioning of requirements

It states the order in which the specified requirements are fulfilled.

3. Specific Requirements

- This section of SRS actually specifies the Requirements briefly by taking help of information stated in above section. This section is divided into following subsections :

(i) Interfaces

- This section of SRS contain the information about various interfaces that are needed for product, like hardware/software interface, system interface, user interfaces, and communication interfaces.

- It means that each and every input device, output device and the communication medium used by system is introduced in this section of SRS document.

(ii) Database

To store data required and generated by system, database is required. Information about the database used in the system is given in this section.

(iii) Performance

It describes the system performance which includes the required speed, task completion time, response time and throughput of the system.

(iv) Software system attributes

- System attributes are : reliability, security, maintainability.
- This section of SRS provides the assurance that all system attributes will be provided by product and the way in which terms these attributes are satisfied/ fulfilled.

4. Change Management Process

- Due to additional user requirements, changes in the system are occurred. This section provides a way that will help to handle such kind of changes.
- As the system make any changes according to user requirements, SRS document should be changed accordingly.

5. Document Approvals

- The SRS document should be accepted by both the parties, including the customers for whom the system is developing and the developer who develop the system.
- Document Approvals section include the approval date time and sign of both the parties.

6. Supporting Information

- It includes guidelines about how to use SRS, index of SRS, references, etc.

Syllabus Topic : Ways of Writing SRS

2.7.4 Ways of Writing SRS

Notation Description

- **Natural language sentences :** The requirements are illustrated with the help of numbered sentences in natural language. Every sentence must express one requirement.
- **Structured natural language :** Natural language is used to write the requirements in a standard format or template. Every field provides information regarding an aspect of the requirement.
- **Design description languages :** Programming language is used by this approach having more abstract features to specify the requirements by the way of defining an operational model of the system. Nowadays this approach is not frequently used even when it can be useful for interface specifications.
- **Graphical notations :** Graphical models which hare mainly supplemented by text annotations, are referred to define the important functional requirements for the system; UML use case as well as sequence diagrams are commonly used.
- **Mathematical specifications :** These notations are in general based on mathematical notions like finite-state machines or sets. Even if these definite specifications can minimize the ambiguity in a requirements document, most customers are not able to recognize a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract.

Syllabus Topic : Structured and Tabular SRS for an Insulin Pump Case Study

2.7.5 Structured and Tabular SRS for an Insulin Pump Case Study

UQ. 2.7.5 Explain structured SRS with case study of Insulin Pump. **SPPU - Aug. 17, 4 Marks**

A structured specification of a requirement for an insulin pump

Function	Compute insulin dose: Safe sugar level.
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1).
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose - the dose in insulin to be delivered
Destination	Main control loop.
Action	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requirements	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Post-condition	r0 is replaced by r1 then r1 is replaced by r2.
Side effects	None.

Tabular specification of computation for an insulin pump

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2 – r1) < (r1 – r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ((r2 – r1) ≥ (r1 – r0))	CompDose = round ((r2 – r1)/4) If rounded result = 0 then CompDose = MinimumDose

2.7.6 SRS Document for Online Student Feedback System

LQ. 2.7.6 Build an SRS document for online student feedback system. **(5 Marks)**

Table of Contents

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Overview
2. General Description
   - 2.1 User Manual
3. Functional Requirements
   - 3.1 Description
4. Interface Requirements
   - 4.1 GUI
   - 4.2 Hardware Interface
   - 4.3 Software Interface
5. Performance Requirements
6. Design Constraints
7. Other non-functional Attributes
   - 7.1 Security
   - 7.2 Reliability
   - 7.3 Availability
   - 7.4 Maintainability
   - 7.5 Reusability
8. Operational Scenarios
9. Preliminary Schedule

1. Introduction

1.1 Purpose

This document gives detailed functional and non-functional requirements for online student feedback system. The purpose of this document is that the requirements mentioned in it should be utilized by software developers to implement the system.

1.2 Scope

This system allows the students to provide quick feedback which is provided by collage staff. The feedback report is generated and which is checked by HOD's. He can view grade and grade obtained to the lecturers.

1.3 Overview

This system provides an easy solution to collage staff and students for maintaining feedback related to collage staff and infrastructure, facility.

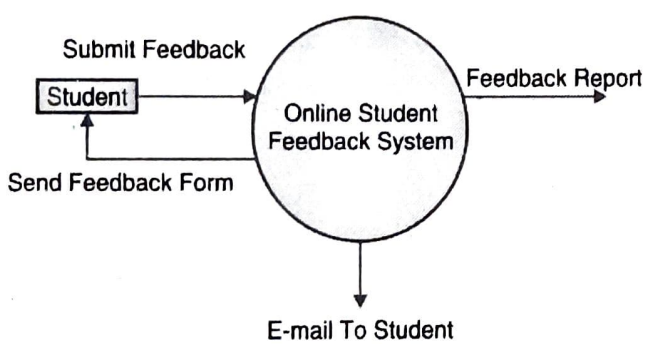


Fig. 2.7.3 : Online student feedback system

2. General Description

- This online student feedback system replaces the traditional, manual feedback system by which lots of paper work will be reduced. The teachers are able to provide feedback regarding facility and students are able to provide feedback easily. This is primary feature of this system.
- Another feature is that feedback form can be provided to student and staff by emailing for filling it.

2.1 User Manual

- The system should provide Help option in which how to operate system should be explained. Also hard copy of this document should be given to user in booklet form.

3. Functional Requirement

3.1 Description

- For identity of staff, system should display staff photograph along with their names for that corresponding subject and skills.
- Statistical report accordingly subject or skill should display individual's report whenever required.

4. Interface Requirement

4.1 GUI

Student

Student can give the feedback about the lecturers on the scale of ten. Student can give feedback about the lecturers based on interaction of lecturer in the class rooms with students and facility provided by collage like infrastructure etc.

Staff

The feedback given by students can be viewed by the staff and improve their performance in teaching and other aspects.

Head of Department

The feedback report can be checked by HOD. He can view overall grades and view the grades obtained by the lecturers and give report to principal and he can give counseling to the collage staff.

Principal

Finally, report was referred by principal who can give suggestions to the lecturers to improve their teaching.

4.2 Hardware Interface

The system should be embedded in all desktops.

4.3 Software Interface

(i) Online Student Feedback System

(ii) The feedback database transmitted to database server.

(iii) Report generator.

5. Performance Requirements

- The system should work concurrently on multiple processors between the collage hours.
- The report should be generated immediately within one hour.

6. Design Constraints

The system should be designed within 6 months.

7. Other non-functional attributes

7.1 Security

The student and staff should be provided password to log on to the system.

7.2 Reliability

Reliability should be guaranteed due to wired connectivity.

7.3 Availability

The system should be available during college hours.

7.4 Maintainability

There should be facility to add and delete feedback form for different purpose.

7.5 Reusability

The same system will be used in each semester.

8. Operational Scenarios

- There should be student database and teacher database. The student database should contain name and feedback information.
- The teacher database should contain name, subject, skills, and other details.

9. Preliminary Schedule

- The system should be designed within 6 months.

2.7.7 SRS for Hospital Management System

LQ. 2.7.7 Develop a software requirement specification (SRS) for developing a software for Hospital Management System.

Create an SRS that contains following:

1. Objective and scope
2. Product Perspective
3. Functional Requirement
4. Non Functional Requirement **(5 Marks)**

Title

- System Requirement Specification Document for Hospital Management System.

Objective

- To get with preparing requirement document, which will be used to capture and document all the requirements at the start of project. In the assignment we mainly focus on functional requirements.

1. Introduction

1.1 Purpose

The main purpose of our system is to make hospital task easy and is to develop software that replaces the manual hospital system into automated hospital management system. This document serves as the unambiguous guide for the developers of this software system.

1.2 Document Conventions

- HMS - Hospital Management System
- GUI - Graphical User Interface
- PHID - Patient Hospital Identification Number

1.3 Scope of the Project

- The purpose of this specification is to document requirements for a system to manage the hospital.
- The specification identifies what such a system is required to do.
- The Hospital Management System will manage a waiting list of patients requiring different treatments.
- The availability of beds will be determined and if beds are available the next appropriate patient on the list will be notified.
- Nurses will be allocated to wards depending on ward sizes, what type of nursing is needed, operating schedules, etc.
- The current manual method of managing patients, nurses, and beds is time consuming and error prone. It is also difficult to manage the large paper flow involved in this process.
- The Hospital Management System will allow hospital administrative staff to access relevant information efficiently and effectively.
- The goal of HMS is to manage nurses, patients, beds, and patients' medical information in an cost-effective manner.
- All of these sub-systems (managing nurses, beds, patient medical information) need to be designed and implemented so that HMS can run effectively.

2. Overall Description

2.1 Product Perspective

The HMS is designed to help the hospital administrator to handle patient, nurse and bed information. The current design goal is to build an internal system to achieve the functionality outlined in this specification.

2.2 Product Functions

- The HMS will allow the user to manage information about patients, nurses, and beds. Patient management will include the checking-in and checking-out of patients to and from the hospital.
- The HMS will also support the automatic backup and protection of data.

2.3 Operating Environment

Following are the requirements for running the software successfully :

- Processor – Pentium III or Higher.
- Ram – 512 MB or Higher.
- Disk Space – 10 GB or Higher.
- OS – Windows XP or Above.

2.4 Design and Implementation Constraint

- GUI only in English.
- Login and password is used for identification of user and there is no facility for guest.

2.5 Assumption and Dependencies

- It is assumed that one hundred compatible computers will be available before the system is installed and tested.
- It is assumed that Hospital will have enough trained staff to take care of the system.

3. External Interface Requirements

3.1 User Interface

Input from the user will be via keyboard and mouse. The user will navigate through the software by clicking on icons and links. The icons will give appropriate responses to the given input.

3.2 Hardware Interface

These are the minimum hardware interfaces :

- **Processor :** Pentium III or Higher.
- **Ram :** 512 MB or Higher.
- **Disk Space :** 10 GB or Higher.

3.3 Software Interface

These are the minimum software interfaces :

- **Technologies :** C# .Net 2.0
- **Database :** SQL server (standard edition).
- **Operating system :** Windows XP or above.

4. System Features

4.1 System Features

- **Work Scheduling :** Assigning nurses to doctors and doctors to patients.
- **Admissions :** Admitting patients, assigning the patients to appropriate wards.
- **Patient Care :** Monitoring patients while they are in the hospital.
- **Surgery Management :** Planning and organizing the work that surgeons and nurses perform in the operating rooms.
- **Ward Management :** Planning and coordinating the management of wards and rooms.
- **Waiting list :** Monitoring to see if there are any patients waiting for available beds, assigning them to doctors and beds once these become available.

5. Other Non-functional Requirements

5.1 Performance Requirements

The performance of our software is at its best when the following regularly are done :

- Password Management
- Regular Database Archiving
- Virus Protection

5.2 Safety Requirements

Humans are error-prone, but the negative effects of common errors should be limited. e.g. users should realize that a given command will delete data, and be asked to confirm their intent or have the option to undo.

5.3 Security Requirements

- Each member is required to enter an individual Username and password when accessing the software. Administrators have the option of increasing the level of password security their members use.
- The data in the database is secured through multiple layers of Protection.
- One of those security layers involves member passwords. For maximum Security of your software, each member must protect their password.

5.4 Software Quality Attributes

The Quality of the system is maintained in such a way that it can be very user-friendly. The software quality attributes are assumed as follows :

- Accurate and hence reliable.
- Secured.
- Fast Speed.
- Compatibility.

Syllabus Topic : Requirements Elicitation and Analysis : Process

2.8 Requirements Elicitation and Analysis : Process

UQ. 2.8.1 Why requirement elicitation is difficult ?

SPPU - Aug. 17, 1 Mark

UQ. 2.8.2 Give general process models of the requirement elicitation and analysis process.

SPPU - Dec. 17, 3 Marks

UQ. 2.8.3 Explain the tasks done during elicitation.

SPPU - May 18, 3 Marks

UQ. 2.8.4 Why requirement elicitation is difficult? What are the problems in requirement elicitation?

SPPU - Dec. 18, 5 Marks

- In requirements engineering, requirements elicitation is the practice of researching and discovering the requirements of a system from users, customers, and other stakeholders. The practice is also sometimes referred to as "requirement gathering".
- The term elicitation is used in books and research to raise the fact that good requirements cannot just be collected from the customer, as would be indicated by the name requirements gathering.
- Requirements elicitation is non-trivial because you can never be sure you get all requirements from the user and customer by just asking them what the system should do OR NOT do (for Safety and Reliability).
- Requirements elicitation practices include interviews, questionnaires, user observation, workshops, brainstorming, use cases, role playing and prototyping.
- Before requirements can be analyzed, modeled, or specified they must be gathered through an elicitation process. Requirements elicitation is a part of the requirements engineering process, usually followed by analysis and specification of the requirements.
- There are several critical tasks have to be done in requirement elicitation, hence it is considered as difficult. There several problems in requirement elicitation while performing those tasks.
- The requirements elicitation and analysis has 4 main processes.
- We typically start by gathering the requirements, this could be done through a general discussion or interviews with the stakeholders, also it may involve some graphical notation.
- Then we have to organize the related requirements into sub components and prioritize them, and finally, refine them by removing any ambiguous requirements that may raise from some conflicts.
- Here are the four main process of requirements elicitation and analysis.

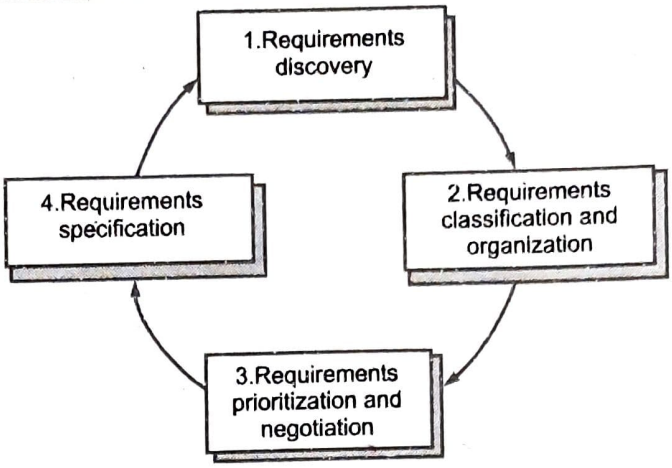


Fig. 2.8.1 : Process of requirements elicitation and analysis

- It shows that it's an iterative process with a feedback from each activity to another. The process cycle starts with requirements discovery and ends with the requirements document. The cycle ends when the requirements document is complete.

1. Requirements Discovery

- It's the process of interacting with, and gathering the requirements from, the stakeholders about the required system and the existing system (if exist).
- It can be done using some techniques, like interviews, scenarios, prototypes, etc, which help the stockholders to understand what the system will be like.
- There are different ways to identify customer requirement / requirement elicitation :

UQ. 2.8.5 State and explain different methods of requirement elicitation.

SPPU - Aug. 17, 4 Marks

(i) Interviews

- Interviews are important way for gathering requirements. Organization may take various kinds of interviews such as :
- Structured or closed interviews in which information to be collected from customer is decided in advance.

- Non-structured or open interviews in which details needs to be collected from customer are decided in advance.
- Oral interviews.
- Written interviews.
- Face-to-face interviews which are held among two people across the table.
- Groups of persons are participating in group interviews. They support to cover any missing requirement as number of people participates in this process.

(ii) Surveys

- Organization may perform surveys of different stakeholders by questioning them about their requirements and expectation from the proposed software system.
- The advantage of this technique is that, collecting requirements is economically beneficial because it collects requirements from a large number of persons at same time.
- Surveys are less effective method of data discovery.
- Frequently, a survey's main finding is that other questions should have been asked instead. Surveys are most useful for capturing clear factual information.

(iii) Questionnaires

- Questionnaires are a document way which contains in-built set of objectives that is based on questions and their options.
- This document is given to all stakeholders to give answer of those questions which are gathered and compiled.
- Output of this mechanism is, if an answer for some question is not given in the questionnaire, the issue may be remaining unattended.
- The advantage of this technique is that, collecting requirements economically is beneficial because it collects requirements from huge amount of persons at same time.
- Questionnaires are less effective method of data discovery.

(iv) Task analysis

- In this technique, team of software developers and engineers may identify the functional specifications for which the new system is needed to develop.
- If the customer has some software to do the particular operations then it is analyzed by this team to find out requirements for proposed system.

(v) Domain Analysis

- Each software put into some domain category.
- The experienced persons in that domain can be a great support to study general and specific requirements.

(vi) Brainstorming

- Brainstorming is an informal debate held between different stakeholders and all their suggestions are documented for further requirement analysis.

(vii) Prototyping

- Prototyping is a technique in which, we create user interface without including detail functionality for user to interpret features of desired software product.
- It supports providing details about requirements.
- If the client does not know its own requirements, in such case the developer develop a prototype based on requirements provided at initial stage.
- The prototype is displayed to the client and the feedback is collected from client.
- The client feedback is used as an input for requirement gathering.

(viii) Observation

- Team of experienced persons visits the organization or workplace of client.
- They observe existing system's work.

- They observe the flow of control at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.

2. Requirements Classification and Organization

- It's very important to organize the overall structure of the system. Putting related requirements together, and decomposing the system into sub components of related requirements. Then, we define the relationship between these components.
- What we do here will help us in the decision of identifying the most suitable architectural design patterns.

3. Requirements Prioritization and Negotiation

- Eliciting and understanding the requirements is not an easy process.
- One of the reasons is the conflicts that may arise as a result of having different stakeholders involved. Why? because it's hard to satisfy all parties, if it's not impossible.
- This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiations until you reach a situation where some of the stakeholders can compromise.
- Prioritizing your requirements will help you later to focus on the essentials and core features of the system, so you can meet the user expectations. It can be achieved by giving every piece of function a priority level. So, functions with higher priorities need higher attention and focus.

4. Requirements Specification

- The requirements are then documented.
- It's the process of writing down the user and system requirements into a document. The requirements should be clear, easy to understand, complete and consistent.
- In practice, this is difficult to achieve as stakeholders interpret the requirements in different ways and there are often inherent conflicts and inconsistencies in the requirements.
- The processes in requirements engineering are interleaved, and it's done iteratively. First iteration you specify the user requirements, then, you specify a more detailed system requirements.

Syllabus Topic : Requirements Validation

2.8.1 Requirements Validation

- It is a process of ensuring the specified requirements meet the customer needs. It's concerned with finding problems with the requirements.
- These problems can lead to extensive rework when these are discovered in the later stages, or after the system is in service.
- The cost of fixing a requirements problem by making a system change is usually much greater than repairing design or code errors. Because a change to the requirements usually means the design and implementation must also be changed, and re-tested.
- During the requirements validation process, different types of checks should be carried out on the requirements. These checks include :
  - **Validity checks :** The functions proposed by stakeholders should be aligned with what the system needs to perform.
  - **Consistency checks :** Requirements in the document shouldn't conflict or encounter different description of the same function
  - **Completeness checks :** The document should include all the requirements and constrains.
  - **Realism checks :** Ensure the requirements can actually be implemented using the knowledge of existing technology, the budget, schedule, etc.

- **Verifiability :** Requirements should be written so that they can be tested. This means you should be able to write a set of tests that demonstrate that the system meets the specified requirements.

- There are some techniques that can use to validate the requirements, and we may use one or more of them together, depending on the needs :

1. **Requirements Reviews :** The requirements are analyzed systematically by a team of reviewers who check for errors and inconsistencies.
2. **Prototyping :** In this approach to validation, an executable model of the system in question is demonstrated to end-users and customers. They can experiment with this model to see if it meets their real needs.
3. **Test-case generation :** Requirements should be testable. If the tests for the requirements are devised as part of the validation process, this often reveals requirements problems. If a test is difficult or impossible to design, this usually means that the requirements will be difficult to implement and should be reconsidered. Developing tests from the user requirements before any code is written is an integral part of extreme programming.

Syllabus Topic : Requirements Management

2.8.2 Requirements Management

UQ. 2.8.6 Explain the tasks done during requirement management. **SPPU - May 18, 3 Marks**

- The purpose of requirements management is to ensure that an organization documents, verifies, and meets the needs and expectations of its customers and internal or external stakeholders. Requirements management begins with the analysis and elicitation of the objectives and constraints of the organization.
- Requirements management further includes supporting planning for requirements, integrating requirements and the organization for working with them (attributes for requirements), as well as relationships with other information delivering against requirements, and changes for these.
- The traceability thus established is used in managing requirements to report back fulfilment of company and stakeholder interests in terms of compliance, completeness, coverage, and consistency.
- Traceabilities also support change management as part of requirements management in understanding the impacts of changes through requirements or other related elements (e.g., functional impacts through relations to functional architecture), and facilitating introducing these changes.
- Requirements management involves communication between the project team members and stakeholders, and adjustment to requirements changes throughout the course of the project.
- To prevent one class of requirements from overriding another, constant communication among members of the development team is critical. For example, in software development for internal applications, the business has such strong needs that it may ignore user requirements, or believe that in creating use cases, the user requirements are being taken care of.

2.8.2(A) Requirements Traceability

- Requirements traceability is concerned with documenting the life of a requirement. It should be possible to trace back to the origin of each requirement and every change made to the requirement should therefore be documented in order to achieve traceability. Even the use of the requirement after the implemented features have been deployed and used should be traceable.
- Requirements come from different sources, like the business person ordering the product, the marketing manager and the actual user. These people all have different requirements for the product. Using

requirements traceability, an implemented feature can be traced back to the person or group that wanted it during the requirements elicitation. This can, for example, be used during the development process to prioritize the requirement, determining how valuable the requirement is to a specific user. It can also be used after the deployment when user studies show that a feature is not used, to see why it was required in the first place.

2.8.2(B) Requirements Activities

- At each stage in a development process, there are key requirements management activities and methods. To illustrate, consider a standard five-phase development process with Investigation, Feasibility, Design, Construction and Test, and Release stages.

1. Investigation

- In Investigation, the first three classes of requirements are gathered from the users, from the business and from the development team. In each area, similar questions are asked; what are the goals, what are the constraints, what are the current tools or processes in place, and so on. Only when these requirements are well understood can functional requirements be developed.
- In the common case, requirements cannot be fully defined at the beginning of the project. Some requirements will change, either because they simply weren't extracted, or because internal or external forces at work affect the project in mid-cycle.
- The deliverable from the Investigation stage is a requirements document that has been approved by all members of the team. Later, in the thick of development, this document will be critical in preventing scope creep or unnecessary changes. As the system develops, each new feature opens a world of new possibilities, so the requirements specification anchors the team to the original vision and permits a controlled discussion of scope change.
- While many organizations still use only documents to manage requirements, others manage their requirements baselines using software tools. These tools allow requirements to be managed in a database, and usually have functions to automate traceability (e.g., by allowing electronic links to be created between parent and child requirements, or between test cases and requirements), electronic baseline creation, version control, and change management. Usually such tools contain an export function that allows a specification document to be created by exporting the requirements data into a standard document application.

2. Feasibility

UQ. 2.8.7 What is mean by feasibility study ?

SPPU - Dec. 17, 2 Marks

- In the Feasibility stage, costs of the requirements are determined. For user requirements, the current cost of work is compared to the future projected costs once the new system is in place.
- Questions such as these are asked : "What are data entry errors costing us now?" Or "What is the cost of scrap due to operator error with the current interface?" Actually, the need for the new tool is often recognized as these questions come to the attention of financial people in the organization.
- Business costs would include, "What department has the budget for this?" "What is the expected rate of return on the new product in the marketplace?" "What's the internal rate of return in reducing costs of training and support if we make a new, easier-to-use system?"
- Technical costs are related to software development costs and hardware costs. "Do we have the right people to create the tool?" "Do we need new equipment to support expanded software roles?" This last question is an important type. The team must inquire into whether the newest automated tools will add sufficient processing power to shift some of the burden from the user to the system in order to save people time.

- The question also points out a fundamental point about requirements management. A human and a tool form a system, and this realization is especially important if the tool is a computer or a new application on a computer.
- The human mind excels in parallel processing and interpretation of trends with insufficient data. The CPU excels in serial processing and accurate mathematical computation. The overarching goal of the requirements management effort for a software project would thus be to make sure the work being automated gets assigned to the proper processor.
- For instance, "Don't make the human remember where she is in the interface. Make the interface report the human's location in the system at all times." Or "Don't make the human enter the same data in two screens. Make the system store the data and fill in the second screen as needed."
- The deliverable from the Feasibility stage is the budget and schedule for the project.

3. Design

- Assuming that costs are accurately determined and benefits to be gained are sufficiently large, the project can proceed to the Design stage. In Design, the main requirements management activity is comparing the results of the design against the requirements document to make sure that work is staying in scope.
- In the construction and testing stage, the main activity of requirements management is to make sure that work and cost stay within schedule and budget, and that the emerging tool does in fact meet requirements. A main tool used in this stage is prototype construction and iterative testing.
- For a software application, the user interface can be created on paper and tested with potential users while the framework of the software is being built. Results of these tests are recorded in a user interface design guide and handed off to the design team when they are ready to develop the interface. This saves their time and makes their jobs much easier.

☞ Requirements change management

- Hardly would any software development project be completed without some changes being asked of the project.
- The changes can stem from changes in the environment in which the finished product is considered to be used, business changes, regulation changes, errors in the original definition of requirements, limitations in technology, changes in the security environment and so on.
- The activities of requirements change management include receiving the change requests from the stakeholders, recording the received change requests, analyzing and determining the desirability and process of implementation, implementation of the change request, quality assurance for the implementation and closing the change request. Then the data of change requests be compiled, analyzed and appropriate metrics are derived and dovetailed into the organizational knowledge repository.

4. Release

- Requirements management does not end with product release. From that point on, the data coming in about the application's acceptability is gathered and fed into the Investigation phase of the next generation or release. Thus the process begins again.

5. Tooling

- Acquiring a tool to support requirements management is no trivial matter and it needs to be undertaken as part of a broader process improvement initiative.
- It has long been a perception that a tool, once acquired and installed on a project, can address all of its requirements management-related needs. However, the purchase or development of a tool to support requirements management can be a costly decision.
- Organizations may get burdened with expensive support contracts, disproportionate effort can get

misdirected towards learning to use the tool and configuring it to address particular needs, and inappropriate use that can lead to erroneous decisions.

- Organizations should follow an incremental process to make decisions about tools to support their particular needs from within the wider context of their development process and tooling. The tools are presented in Requirements traceability.

2.9 Requirement Model

LQ. 2.9.1 Explain requirement model. (5 Marks)

- Requirement modeling is implemented after the requirements and constraints have been collected and further analysed.
- Requirement modeling is considered as vital activity to make sure the consistency as well as completeness of the requirements.
- There are different options to model functional, quality attributes and constraints. Selection of suitable approach is depending upon the type of system and the organizational standards.

There are many ways requirements can be modelled; some of the most common requirement models which are as follows :

(a) The Domain Model

- The Domain Model captures all the respective things (Domain Concepts) the product or system "deals with", i.e. represents and manipulates from a business perspective.
- The information captured for a Domain Concept is a description and often properties and their relationship to each other.
- The purpose of the Domain Model is to establish a common understanding of the static aspects of the business domain among all the stakeholders.
- The Domain Model supports the creation of quality requirements by establishing a common vocabulary across all stakeholders reducing ambiguity and redundancy.
- As a result, the following guidelines should be considered :
  - The contents should be written in the business language of the domain and avoid implementation specific or technical aspects (such as operations or actions, which some presentations do not rule out): Use good business names!
  - Stakeholders should be comfortable with the chosen representation of the Domain Model in order to be able to provide feedback. As a result, multiple views on the domain model may be necessary for different stakeholders.
  - The domain model is developed in iterations where necessary details are enriched over time.
- An obvious way to identify Domain Concepts is to identify nouns and phrases in textual descriptions of a domain. The information about the Domain Concepts can come from existing documentation, industry standard documents for a specific domain, and output from the requirements elicitation.
- The Domain Model can be represented in different ways. The representation is dependent on the methodology and the formality employed in the project as well as the experience of the stakeholders exposed to the Domain Model. It can stretch from a Word document, an Excel table and diagrammatic representations to a fully detailed model representation using a UML class diagram.
- The Domain Driven Design approach is an extension of the Domain Model to produce an implementation which is linked to an evolving model of the domain, it is appropriate for a highly complex domain and produces a highly maintainable system, and the implementation can be done by Domain Specific Language.

(b) The Use Case Model

- A Use Case Model describes the proposed functionality of a new system.
- A Use Case represents a discrete unit of interaction between a user, called an Actor, (human or machine) and the system.
- This interaction is a single unit of meaningful work, such as Create Account or View Account Details.
- Each Use Case describes the functionality to be built in the proposed system, which can include another Use Case's functionality or extend another Use Case with its own behavior.

Some other models used to capture functionality include :

(c) Life Cycles

- These are descriptions of the valid states in each Domain Concept's "life", along with the relationships between those states (e.g. legal transitions).
- It is important to note that such Life Cycles are related to a single Domain Concept (i.e. not to the system as a whole, one or more Use Cases, or the interaction of the system with the outside world).
- Life Cycles are often referred to as State Machines or State Charts.

(d) Business Process models

- The use of Business Process Modelling (BPM) to represent the functionality of the enterprise is an alternative approach for developing software as fundamental concept. It t is to develop the model in a tool which can be then used to produce an executable model.
- Each element has KPIs (Key performance indicators) attached to it so that it can be frequently used as part of a performance improvement process.

(e) User Stories

- In agile projects user stories are used to capture requirements, these are short descriptions of how a particular user wants to interact with a system, in the form "As I want to, so that".
- The user story includes a set of acceptance criteria which describe the boundary of the story and defines when it is done.

(f) Traceability Matrix

- To fulfill the need for traceability a traceability matrix can be produced.
- This can be as simple as a table which cross references the requirements to the software component that fulfils it, however this can become very hard to maintain, so it is more common to use a requirements management tool when this level of formality is required.

Syllabus Topic : Case Studies : The Information System Case study - Mental health care patient management system (MHC-PMS)

2.10 Case Studies : The Information System

Case study - Mental health care patient management system (MHC-PMS)

Please refer section 1.18 of Unit 1..

...Chapter Ends

□□□

Design Engineering

University Prescribed Syllabus

Design Process and quality, Design Concepts, The design Model, Pattern-based Software Design.

Architectural Design : Design Decisions, Views, Patterns, Application Architectures,

Modeling Component level Design : Component, Designing class based components, conducting component -level design,

User Interface Design : The golden rules, Interface Design steps and Analysis, Design Evaluation,

Case Study : Web App Interface Design.

Syllabus Topic : Design Process and Quality

3.1 Design Process and Quality

LQ. 3.1.1 What is Design ? (3 Marks)

- Once the requirements document regarding the software to be developed is presented, the phase of software design gets started.
- The requirement specification activity is considered purely related with the problem domain whereas design is considered as the initial phase of transforming the problem into a solution.
- In the design phase, all the relevant entities such as the customer, business requirements and technical considerations collaborate to formulate a product or a system.
- In design process, there are several elements such as set of principles, concepts and practices, which help a software engineer to model the system or product which is to be built.
- The design model is assessed for quality and reviewed before generation of code and execution of tests.
- The design model gives detailed information regarding software data structures, architecture, interfaces and components which are necessary to employ the system.

Basic of Software Design

- Software design is considered as a phase in software engineering which develops a blueprint that will be a base for constructing the software system.
- **IEEE** defines software design as 'both a process of defining the architecture, components, interfaces, and other characteristics of a system or component and the result of that process.'
- In the design phase, important and strategic decisions are made to get the expected functionality and quality of the system.
- These decisions are considered to successfully develop the software and handle its maintenance in a way that the quality of the end product will be improved.
- Software design encompasses the set of principles, concepts, and practices that lead to the development of a high-quality system or product.
- Design concepts must be understood before the design practices are applied.
- Design practice itself leads to the creation of various representations of the software.
- Design Practices serve as a guide for the construction activity that follows.
- The design model provides details about software data structures, architecture, interfaces, and components that are necessary to implement the system.
- Mitch Kapor, the creator of Lotus 1-2-3, presented a "software design manifesto" in Dr. Dobbs Journal.
- He said :! Good software design should exhibit :

A. Firmness : A program should not have any bugs that inhibit its function.

B. Commodity : A program should be suitable for the purposes for which it was intended.

C. Delight : The experience of using the program should be pleasurable one.

Syllabus Topic : Design Model

3.1.1 Design Model

Software design model consists of 4 designs

1. Data/class design
2. Architectural design
3. Interface design
4. Component design

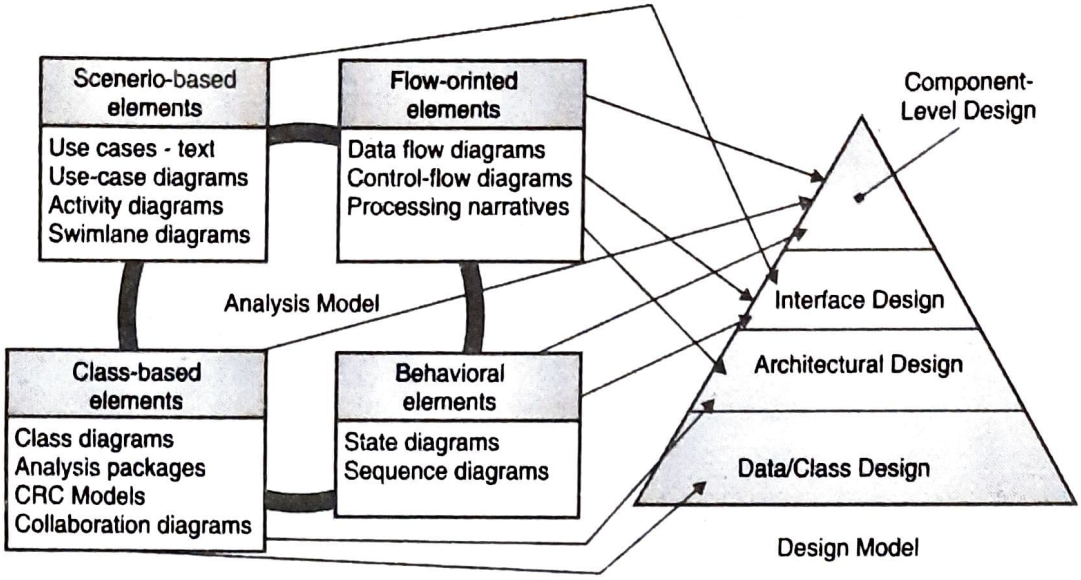


Fig. 3.1.1 : Translating Requirement Model into Design Model

Design Guidelines

1. A design should exhibit an architecture that :
   (a) Has been created using recognizable architectural styles or patterns,
   (b) Is composed of components that exhibit good design characteristics and
   (c) Can be implemented in an evolutionary fashion!
2. A design should be modular; that is, the software should be logically partitioned into elements or subsystems.
3. A design should contain distinct representations of data, architecture, interfaces, and components.
4. A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
5. A design should lead to components that exhibit independent functional characteristics.
6. A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
7. A design should be derived using a repeatable method that is driven from information obtained during software requirements analysis.
8. A design should be represented using a notation that effectively communicates its meaning.

3.1.2 Qualities of Good Design

LQ. 3.1.2 Explain qualities of good software design. **(5 Marks)**

There are number of qualities of good design :

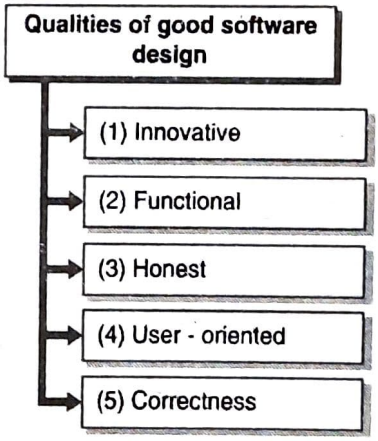


Fig. 3.1.2 : Qualities of good software design

(1) Innovative

- Innovative design can be either completely new design or redesign of existing product.
- New design gives unseen value to market where as redesign improves the quality of an existing product.

(2) Functional

Good design fulfils all its intended functions to solve user's problem. It focuses on usefulness of a product by optimizing its functionality.

(3) Honest

A good design is honest. An honest design expresses the functions and values it offers. It never attempts to modify Organiser's and User's view with promises it can't keep.

(4) User - oriented

Good design is developed based on its use and intended to improve solution to problem for its user.

User-oriented design gives intellectual as well as material value to system which in turn achieves user's satisfaction.

(5) Correctness

Correctness is an important quality of good design. A good design should correctly achieve all required functionalities as per SRS document.

3.2 Design Principles

LQ. 3.2.1 Explain design principles. (5 Marks)

Following are important design principles :

1. The design process should not suffer from 'tunnel vision'.
2. The design should be traceable to the analysis model.
3. The design should not reinvent the wheel.
4. The design should "minimize the intellectual distance" between the software and the problem as it exists in the real world.
5. The design should exhibit uniformity and integration.
6. The design should be structured to accommodate change.
7. The design should be structured to degrade gently, even when unusual data, events, or operating conditions are encountered.
8. Design is not coding, coding is not design.
9. The design should be assessed for quality as it is being created, not after the creation.
10. The design should be reviewed to minimize conceptual (semantic) errors.

Syllabus Topic : Design Concepts

3.3 Design Concepts

UQ. 3.3.1 Explain different design concepts.

SPPU - Aug. 17, 3 Marks

UQ. 3.3.2 Define Following Design concepts :

i) Patterns

ii) Information Hiding

iii) Architecture

iv) Refinement

SPPU - Aug. 17, 4 Marks

- All of the software processes are characterized by basic concepts in conjunction with some practices or methods.
- Methods represent the way by which the concepts are applied. An Old technology is replaced by new technology, several modifications occur in the methods which are used to apply the concepts for the process of developing the software.
- However, the fundamental concepts which underline the software design process always remain the same. These concepts are :

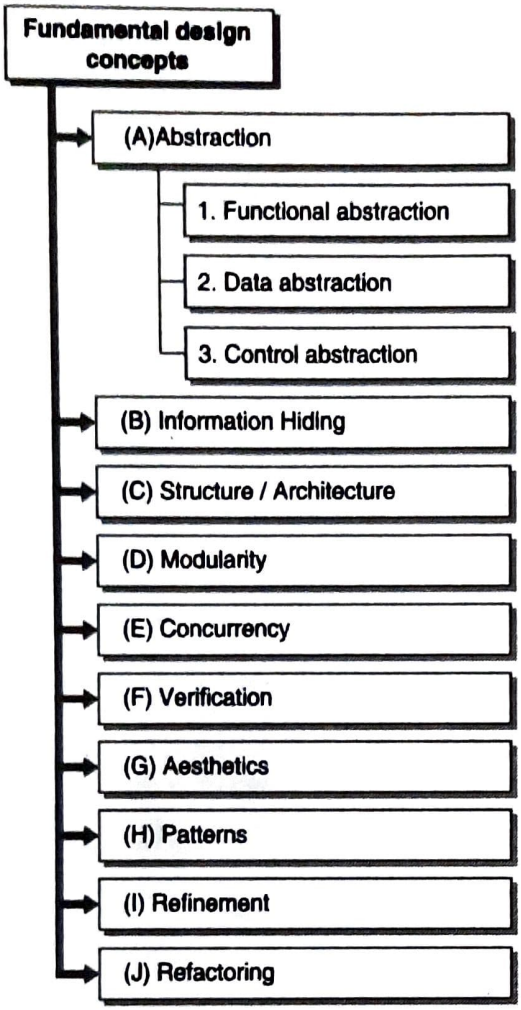


Fig. 3.3.1 : Fundamental design concepts

(A) Abstraction

LQ. 3.3.3 Explain design concept - Abstraction. (5 Marks)

- Abstraction refers to a process by which software designers consider components at an abstract level, at the same time as ignoring the implementation details of the components.
- **IEEE** defines abstraction as 'a view of a problem that extracts the essential information relevant to a particular purpose and ignores the remainder of the information.'
- There are two ways to use the concept of abstraction; as a process and as an entity.
- As a **process,** abstraction refers to a mechanism of hiding irrelevant extra details and representing just the important essential features of an element so that one can concentrate on essential things at a time.
- As an **entity,** abstraction refers to a model or view of an item.
- In the software process all the steps are accomplished via several levels of abstraction.
- At the highest level, one can get the outline of the solution to the respective problem while at the lower levels, he/she will get the solution to the problem in detail.
- For example, in the requirements analysis phase, language is used to provide a solution to the problem and as one proceeds during the software process, the abstraction level reduces and source code of the software is generated at the lowest level.
- In general three abstraction mechanisms used in software design are functional abstraction, data abstraction and control abstraction.
- These mechanisms help to control the complexity of the design process by starting from the abstract design model and ending at concrete design model in a systematic manner.

1. **Functional abstraction :** This type of abstraction includes the use of parameterized subprograms. Functional abstraction can be created as a set of subprograms known as 'groups'. In these sets, routines are exist which may be visible or invisible. The use of visible routines can be done within the containing groups and also within other groups, while invisible routines are hidden from other groups and are allowed to be used in the containing group only.
2. **Data abstraction :** This type of abstraction includes describing data which specifies a data object. For example, the data object *flower* encompasses a set of attributes (color, fragrance) which describe the flower object clearly. In this abstraction mechanism, representation as well as manipulation details are ignored.

3. **Control abstraction :** This type of abstraction states the desired effect, without conducting the exact mechanism of control. For example, in programming language like C, *if* and *while* statements are abstractions of machine code implementations containing conditional instructions.

(B) Information Hiding

- It is important to specify and design modules in such a manner that the data structures as well as processing details of one module should not be accessible to any other modules.
- Only required information is transferred between the modules. The way of hiding unnecessary details is referred to as **information hiding.**
- **IEEE** defines information hiding as the technique of encapsulating software design decisions in modules in such a way that the module's interfaces reveal as little as possible about the module's inner workings; thus each module is a 'black box' to other modules in the system.
- Information hiding is an important aspect when there is need of modifications during the testing and maintenance phase.

☞ Advantages of information hiding

1. Results in low coupling.
2. Put emphasis on communication by controlled interfaces.
3. Reduces the possibility of adverse effects.
4. Controls the impacts of changes in one component to others.
5. Results in higher quality software.

(C) Structure / Architecture

UQ. 3.3.4 What is architecture ?
SPPU - Aug. 17, 1 Mark

- Software architecture refers to the structure of the system, which consists of several components regarding a program/ system, the attributes (properties) of those components and the relationship among them.
- The software architecture helps the software engineers in the process of analyzing the software design proficiently.
- Additionally it also helps the software engineers in decision-making as well as handling risks. Following tasks are implemented by the software architecture :
  - Give an insight to all the relevant stakeholders which helps them to communicate with each other.
  - Spot the early design decisions having big impact on the software engineering activities such as coding and testing which are followed after the design phase.
  - Generates intellectual models regarding the way by which the system is organized into components and the way by which these components communicate with each other.

(D) Modularity

LQ. 3.3.5 Explain design concepts : Modularity.
(5 Marks)

- Modularity can be achieved by the process of dividing the software into components which are uniquely named and addressable. These components are also known as **modules.**
- A complex system (huge application) is partitioned into a set of discrete modules in such a manner that one should be able to develop those modules independent of each other.
- After development is completed, the modules are integrated to form an entire project.
- One has to remember that as the number of modules a system is divided into increases, it becomes easy to handle the software but it also increases the effort required to integrate the modules.

- The process of modularizing a design helps to plan the development in a more efficient way, accommodate changes without difficulty, conduct testing and debugging effectively as well as efficiently, and carry out the maintenance work without adversely affecting the functionality of the software.

Example,

- The ubiquitous television set is an example of a system made up of a number of modules - speakers, projection tube, channel buttons, volume buttons, etc.
- Each module has its own defined functionality but when they are put together synergistically, the complete functionalities of a television are realized.

(E) Concurrency

- It is important to utilize the resources efficiently as much as possible. For this purpose multiple tasks must be executed concurrently.
- This aspect makes concurrency one of the important concepts of software design.
- Every system should be designed in such a manner that it should facilitate multiple processes to execute concurrently.
- For example, if the currently executing process is waiting for some resource, the system must be able to execute any other process in the mean time.

(F) Verification

- Verification is described as a mechanism of confirmation by examining and giving evidence that there is no mismatch in design output and the design input specifications.
- Design input can be any physical as well as performance requirement which is used as the basis for the purpose of designing.
- Design output is considered as the result of all the design phases' efforts. The final design output is a basis for device master record.

(G) Aesthetics

- Aesthetics is the philosophical study of beauty and taste.
- Software engineering is a largely creative design discipline, where new designs (both graphical design models as well as code) are created from scratch.
- Compared to art, there are many targets (e.g. desired functionality) and constraints (e.g. desired quality attributes) that the software engineer must adhere to, but these still leave almost infinitely many ways to create a model that fulfills all those requirements.
- In addition, for essentially the same model, there are many ways to represent it. Since models and code must also serve to communicate ideas to other people (or to the same people months or years later), the chosen structure and representation are key attributes for the success of the product.
- It's a well-accepted hypothesis that the **aesthetics** of the artifacts produced during software development are an indication of 'the quality' : it is very common to comment on designs as being 'beautiful" or "elegant".
- Such a qualification is used to indicate the apparent quality of the design. Apparently, we make an intuitive connection between aesthetics and quality.

(H) Patterns

- We use **patterns** to identify solutions to design problems that are recurring and can be solved reliably.
- A pattern must be guaranteed to work so that it may be reused many times over, but it also must be relevant to the current project at the same time.
- If the said pattern does not fit into the overall design function of the current project, it might be possible to reuse it as a guide to help create a new pattern that would be more fitting to the situation.
- There are three main patterns :
  - **Architectural** - High-level pattern type that can be defined as the overall formation and organization of the software system itself.

- **Design** - Medium-level pattern type that is used by the developers to solve problems in the design stage of development. Can affect how objects or components interact with one another.
- **Idioms** - Low-level pattern type, often known as coding patterns, they are used as a workaround means of setting up and defining how components will be interacting with the software itself without being dependent on the programming language. There are many different programming languages all with different syntax rules, making this a requirement to function on a variety of platforms.

(I) Refinement

- It is the process of elaboration. Refinement is a top-down design approach.
- A hierarchy is developed by decomposing a macroscopic statement of function in a step-wise fashion until programming language statements are reached.
- In each step, one or several instructions of a given program are decomposed into more detailed instructions. Abstraction and Refinement are complementary concepts.
- Abstraction is concerned with hiding lower levels of detail; it moves from lower to higher levels. Refinement is the movement from higher levels of detail to lower levels. Both concepts are necessary in developing software.

(J) Refactoring

UQ. 3.3.6 What do you mean by refactoring ?

SPPU - Dec. 18, 2 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

UQ. 3.3.7 Give the importance of refactoring in improving quality of service.

SPPU - Dec. 18, 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

3.4 Effective Modular Design

LQ. 3.4.1 Write note on effective modular design.

(5 Marks)

- Modularity can be defined as dividing the software into distinctively named and addressable components, which are also called as modules.

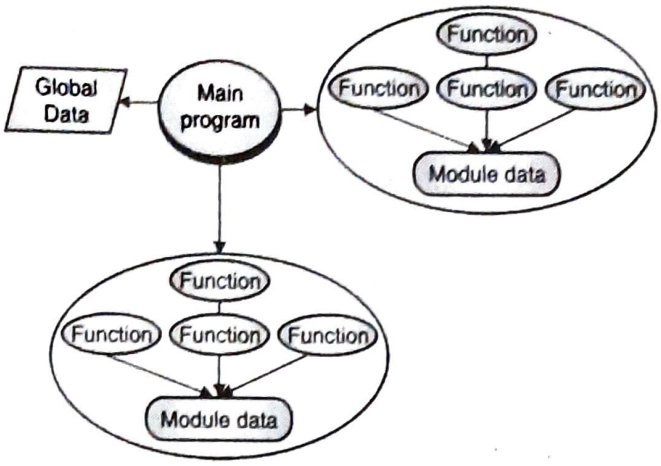


Fig. 3.4.1 : Modularity

- A complex system (large program) is divided into a set of distinct modules in such a way that each module can be developed independent of other modules.
- After developing the individual modules, all these modules are incorporated together to fulfill the software requirements.
- As the modules are developed separately if the number of modules is very huge then it requires more efforts for incorporating them.
- Modularizing a design assists to plan the development in a more efficient manner, put up changes easily, carries testing and debugging effectively and efficiently, and conducts maintenance work without harmfully affecting the functioning of the software.

3.4.1 Advantages of Modularization

1. Maintaining smaller components is very easy.
2. According to the functional aspects the program can be divided.
3. As per the requirement, the level of abstraction can be carried out in the program.
4. Components with high cohesion can be re-used again.
5. Concurrent execution can be made possible.
6. Security can be achieved.

- Modularity has become an accepted approach in all engineering disciplines. A modular design reduces complexity, facilitates change (a critical aspect of software maintainability), and results in easier implementation by encouraging parallel development of different parts of a system.

3.4.2 Functional Independence

LQ. 3.4.2 Write note on function independence. **(5 Marks)**

- The concept of functional independence is considered as a direct outcome of the modularity and the concepts of abstraction as well as information hiding.
- Functional independence is implemented by the process of developing modules with "single-minded" function and an "aversion" with extreme communication with other modules.
- In other words, we need to design software in order that each module is concerned with a particular sub-function of requirements and posses only one interface when seen from different parts of the program structure.
- One good question is that why independence is important. Software having efficient modularity, that is, independent modules, is simple to develop since function may be compartmentalized and simplification of interfaces is done (consider the process of ramifications when a team conducts development).
- It is simple to maintain independent modules (and test) since there is limitation in secondary effects which are generated by design or code modification. Also there is reduction in error propagation, and possible to reuse modules.
- As a result functional independence is a key to good design, and in turn the respective design is the key to good quality of the software.
- There are two qualitative criteria to measure independence : cohesion and coupling.
- Cohesion is used to measure the relative functional strength of a module where as coupling is used to measure the relative interdependence in between modules.

3.4.3 Cohesion

UQ. 3.4.3 What do you mean by the term cohesion in the context of software design? How is this concept useful in arriving at a good design of software? **SPPU - Aug. 17, 2 Marks**

- Cohesion define the degree to which components of system are related to each other i.e. it measures the strength of relationship between two components of system.
- Good system design must have high cohesion between the components of system.

3.4.3(A) Different Types of Cohesion

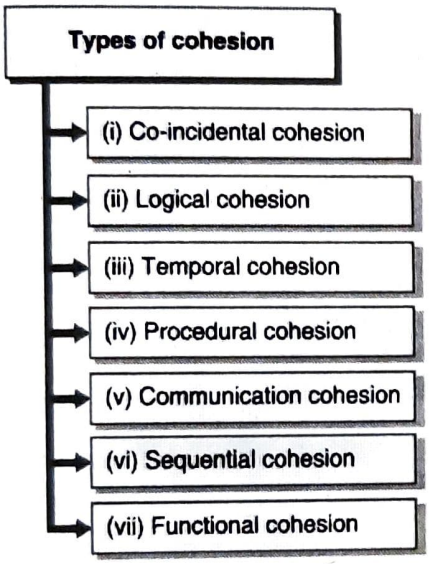


Fig. 3.4.2 : Types of cohesion

(i) Co-incidental cohesion

- An unplanned cohesion which results into decomposition of a program into smaller components for the modularization of system is called co-incidental cohesion.
- Typically these types of cohesions are never accepted.

(ii) Logical cohesion

- When logically classified elements are combined into a single module then it is called as logical cohesion.
- Here all elements of modules contribute to same system operation.

(iii) Temporal cohesion

- Temporal cohesion occurs when component of a system performs more than one function and these functions must occur within same time span.
- It is generally found in initiation and termination activities.

(iv) Procedural cohesion

- When components of system are related to each other only by sequence then there exists procedural cohesion.
- Loop in programming language is good example of procedural cohesion.

(v) Communication cohesion

- When elements of module perform different functions but each function accepts same input and generates same output then that module is said to be communicational cohesive.
- It is sometimes acceptable if no other alternative is easily found.

(vi) Sequential cohesion

- A module is said to be sequentially cohesive if its functions are related in a way such that output of one function acts as an input data to other function.
- This type of cohesion is easily maintainable.

(vii) Functional cohesion

- If elements of module are grouped together since they contribute to a single function then such a module is functionally cohesive module.
- All elements of such a module are necessary for successful execution of function.

3.4.3(B) Advantages of Cohesion

(i) High cohesion among system components results in Better program design.

(ii) High cohesion components can be easily reused.

(iii) High cohesion components are more reliable.

3.4.3(C) Disadvantages of Cohesion

(i) Low cohesion components are difficult to maintain.

(ii) Low cohesion components cannot be reused.

(iii) Low cohesion components are difficult to understand.

(iv) Low cohesion components are less reliable.

3.4.4 Coupling

UQ. 3.4.4 What do you mean by the term coupling in the context of software design? How is this concept useful in arriving at a good design of software ? **SPPU - Aug. 17, 2 Marks**

- Coupling is an indication of strength of interconnection between various components of a system.
- Highly coupled components are more dependent on each other whereas low - coupled components are less dependent or almost independent on each other.
- Good design always have low coupling between components of system.

Example,

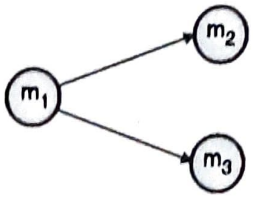


Fig. 3.4.3 : Coupling

- Here module m_1 is dependent on modules m_2 and m_3 for its functionality i.e. if any one of m_2 and m_3 fails m_1 will also fail.
- But m_2 and m_3 are free modules they do not depend on any other component for their functions. Thus success or failure of any other module does not affect modules m_2 and m_3.
- If modules are highly coupled with each other then change in any one of them force change in other modules or components.
- One of the solutions to reduce coupling between system components is a functional design.
- In general, in system development coupling is always constructed with the cohesion.

3.4.4(A) Types of Coupling

LQ. 3.4.5 Explain different types of coupling. **(5 Marks)**

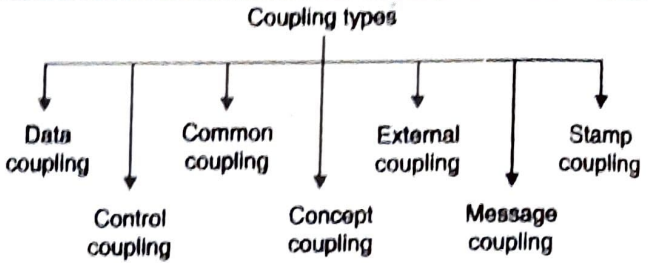


Fig. 3.4.4 : Coupling types

(i) Data coupling

- Data coupling occurs between two components of system when they pass data to each other by parameters using argument list and each element in argument list is used.
- A component becomes difficult to maintain if too many parameters are passed.

(ii) Control coupling

- When data are passed between two components and that affects internal logic of a component then there exists control coupling between those two components.
- It passes the control flags between system components.

(iii) Common coupling

- Common coupling is also called as global coupling since in these coupling components of system communicates with each other by using global area.
- Common coupling becomes difficult to track when data in global area is changed.

(iv) Content coupling

- When one component refers to the internals of the other component then those two components are said to be content coupled.

- As content coupling is worst type of coupling it should not be used in designing.

(v) External coupling

- If two components of system share an externally imposed data format, communication protocol or device interface then they are said to be externally coupled.
- External coupling refers to communication between system components and external tools or devices.

(vi) Message coupling

- Message coupling occurs between two components of system when those components communicate with each other via message passing.
- In this coupling components are independent of each other; thus it is low type coupling.

(vii) Stamp coupling

- Stamp coupling occurs between two components of system when data between them are passed by arguments using a data structure containing elements which may or may not be used.
- Stamp coupling is also low type coupling.

3.4.4(B) Advantages of Coupling

(i) Low coupling components do not force ripple effect to other components.

(ii) Low coupled components can be reused.

(iii) With low coupling among components, system can built faster.

3.4.4(C) Disadvantages of Coupling

(i) High coupling components are difficult to understand.

(ii) High coupling components forces change in other components if one component changes.

(iii) High coupling slows down the development process.

3.4.4(D) Advantages of High Cohesion and Low Coupling

LQ. 3.4.6 What are benefits of high cohesion and low coupling ? **(4 Marks)**

Benefits of high cohesion

1. **Readability :** (Closely) Related functions are contained in a single module.
2. **Maintainability :** Debugging tends to be contained in a single module.
3. **Reusability :** Classes that have concentrated functionalities are not polluted with useless functions.

Benefits of low coupling

1. **Maintainability :** Changes are confined in a single module.
2. **Testability :** Modules involved in unit testing can be limited to a minimum.
3. **Readability :** Classes that need to be analyzed are kept at a minimum.

3.4.4(E) Differences between Coupling and Cohesion

LQ. 3.4.7 Differentiate between cohesion and coupling. **(5 Marks)**

Parameter	Cohesion	Coupling
Concept	Cohesion indicates relationship with in the module.	Coupling indicates the relationship between two or more different modules.
Represents	Cohesion represents relative functional strength of a module.	Coupling represents relative interdependence among the modules.
Degree	It is a degree up to which module focuses on systems' single function/component.	It is a degree to which one module depends on another module.
High or Low	High cohesion is good for system design.	Low coupling is good for system design.

Syllabus Topic : Pattern Based Software Design

3.5 Pattern Based Software Design

- Following are important characteristics of an effective design pattern :

1. **It solves a problem :** Patterns capture solutions, not just abstractv principles or strategies.
2. **It is a proven concept :** Patterns capture solutions with a trackv record, not theories or speculation.
3. **The solution isn't obvious :** Many problem-solving techniques (such as software design paradigms or methods) try to derive solutions from first principles. The best patterns generate a solution to a problem indirectly a necessary approach for the most difficult problems of design.
4. **It describes a relationship :** Patterns don't just describe modules, but describe deeper system structures and mechanisms. The pattern has a significant human component (minimize humanv intervention). All software serves human comfort or quality of life; the best patterns explicitly appeal to aesthetics and utility.

3.5.1 Kinds of Patterns

There are several types of design patterns as follows :

1. Architectural Design
2. Component Level Design
3. Class Based Design
4. User Interface Design
5. Scenario Based Patterns
6. Flow Based Patterns
7. Behavioral Based Patterns
8. System Level Design

Syllabus Topic : Architectural Design

3.6 Architectural Design

LQ. 3.6.1 Explain architectural design. **(5 Marks)**

- Requirements of the software should be transformed into an architecture that describes the software's top-level structure and recognizes its components.
- This can be achieved by using the architectural design which acts as an initial 'blueprint' from which software can be developed.
- IEEE described architectural design as a process of defining a set of hardware and software components and their interfaces to set up the framework for the development of a computer system. This framework is established by observing the software requirements document and designing a model for providing implementation details.
- These details are used to state the components of the system together with their inputs, outputs, functions, and the interaction between them.
- Architectural design can be represented with the help of following models :

1. **Structural model :** Demonstrate architecture as an ordered collection of program components.
2. **Dynamic model :** This model illustrates the behavioural aspect of the software architecture and shows how the structure or system configuration changes as the function changes because of the change in the external environment.
3. **Process model :** This model concentrates on the design of the business or technical process, which must be implemented in the system.
4. **Functional model :** This model signifies the functional hierarchy of a system.
5. **Framework model :** This model ties to recognize repeatable architectural design patterns which are found in same types of applications. This will results in increase in the level of abstraction.

3.6.1 Functions of Architectural Design

1. It describe an abstraction level at which the designers can state the functional and performance behavior of the system.
2. It estimates all top-level designs.
3. It acts as a guideline for improving the system by illustrating those features of the system that can be changed easily while maintaining the system integrity.
4. It develops and documents top-level design for the external and internal interfaces.
5. It develops initial versions of user documentation.

3.6.2 Architectural Design Document

- The result of architectural design process is Architectural Design Document (ADD).
- This document contains a number of graphical representations that encompasses software models along with related descriptive text.
- The software models consist of static model, interface model, relationship model, and dynamic process model which illustrates how the system is arranged into a process during run-time.
- Architectural design document provides the developers a solution to the problem defined in the Software Requirements Specification (SRS).
- In addition to ADD, other outputs of the architectural design are given below :
  1. A range of reports including audit report, progress report, and configuration status accounts report.
  2. Different plans for detailed design phase, which consist of following :
     (i) Software verification and validation plan,
     (ii) Software configuration management plan,
     (iii) Software quality assurance plan,
     (iv) Software project management plan.

3.6.3 Architectural Context Diagram

UQ. 3.6.2 Explain the architecture context diagram.
SPPU - Aug. 17, 2 Marks

- At the architectural design level, an ACD (architectural context diagram) is used by software architect for the purpose of modeling the manner in which software communicates with entities which are mainly external to its boundaries.
- Fig. 3.6.1, illustrates the generic structure of the architectural context diagram.

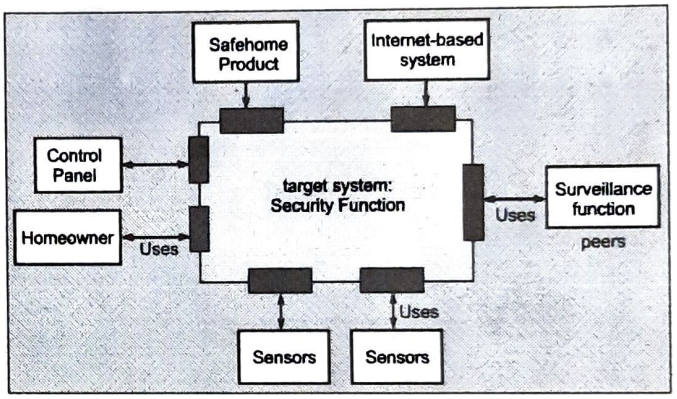


Fig. 3.6.1 : Architectural Context Diagram (ACD)

In Fig. 3.6.1, we can observe that the respective systems which interoperate with the specified target system (the particular system for which there is development of an architectural design) are represented as :

- **Super ordinate systems :** Those are the systems which refers the target system as part of some higher-level processing scheme.
- **Subordinate systems :** Those are the systems which are used by the target system and provide data or processing which are important to complete target system functionality.
- **Peer-level systems :** Those are the systems which communicate on a peer-to- peer basis (i.e., information is either generated or utilized by the peers and the target system.
- **Actors entities (people, devices) :** That communicate with the target system by generating or utilizing information.

- All of the respective external entities interact with the target system with the help of an interface (the small shaded rectangles).

Syllabus Topic : Design Decisions

3.6.4 Design Decisions

LQ. 3.6.3 Explain the design decision in software design. **(5 Marks)**

- Architectural design is considered as a creative process in which a system organization is designed which will fulfill the functional and as well as non-functional requirements of the respective system.
- Since Architectural design is a creative process, the activities included in the process are based on the type of system being developed, the background as well as work experience of the system architect, and the particular requirements of the respective system.
- Hence it is important to consider architectural design as a sequence of decisions to be made instead of a series of activities.
- In the architectural design process, system architects have responsibility to make a several structural decisions which profoundly impact the system and its development process.
- As per their knowledge as well as experience, they need to consider the following fundamental questions regarding the system :
  1. Is there availability of generic application architecture which can work as a template for the system which is being designed?
  2. What will be the way of distribution of system across a number of cores or processors?
  3. What will be architectural patterns or styles might be used?
  4. What fundamental approach will be referred to structure the system?
  5. What will be the way by which the structural components in the system will be decomposed into subcomponents?
  6. What strategy will be used to control the operation of the components in the system?
  7. What architectural organization is best for delivering the non-functional requirements of the system?
  8. How will the architectural design be evaluated?
  9. How should the architecture of the system be documented?
- Even if every software system is considered as unique, systems present in the same application domain usually have equivalent architectures which reflect the fundamental concepts of the domain.
- For example, application product lines are considered as applications which are built around a core architecture having variants which fulfill particular customer requirements.
- While designing system architecture, it is important to decide what the system and broader application classes have in common, and make a decision regarding how much knowledge from these application architectures can be reused.
- For embedded systems and the particular systems which have been designed for personal computers, there is in general just a single processor and there is no need to design a distributed architecture for such system. However, nowadays most of the large systems are distributed where the system software is distributed across several computers.
- The selection of distribution architecture is considered as a key decision which impacts the performance as well as reliability of the respective system.
- The architecture of a software system may be depending upon any specific architectural pattern or style. An architectural pattern is a considered as thorough description of a system organization, for

example a client-server organization or a layered architecture.

- As a consequence of the strong relationship among non-functional requirements and software architecture, the specific architectural style and structure selected for a particular system must be based on the non-functional system requirements :

1. **Performance :** If performance is considered as an important requirement, the architecture must be designed to localize significant activities within a small number of components, having deployment of all these components on the same computer instead of distributed across the network.

   It indicates the using a few comparatively large components instead of small, fine-grain components, which decreases the number of component communications. Run-time system organizations can be considered which lets the system to be replicated and further executed on separate processors.

2. **Security :** If security is an important requirement, there must be use of layered structure for the architecture, having the most important assets confined in the innermost layers, with application of a high level of security validation to these layers.

3. **Safety :** If safety is an important requirement, the design of architecture must be in such a way that safety-regarding operations are all located in either a single component or in a small number of components. This decreases the costs and issues of safety validation and provides related protection systems which can safely shut down the system in the situation of failure.

4. **Availability :** If availability is an important requirement, the design of architecture must be in such a way that it can include redundant components so that it will be easy to replace and update components without need of stopping the system.

5. **Maintainability :** If maintainability is an important requirement, the system architecture should be designed with the help of fine-grain, self-contained components which may readily be updated. The data producers should be separated from consumers and shared data structures should be avoided.

Syllabus Topic : Views

3.6.5 Views / Architectural Views

LQ. 3.6.4 Write note on Architectural Views. (4 Marks)

- 4 + 1 is a view model designed by Philippe Kruchten for "describing the architecture of software-intensive systems, based on the use of multiple, concurrent views".
- The views are used to describe the system from the viewpoint of different stakeholders, such as end-users, developers, system engineer, and project managers. The four views of the model are logical, development, process and physical view. In addition selected use cases or scenarios are used to illustrate the architecture serving as the 'plus one' view.
- Hence the model contains 4+1 views :

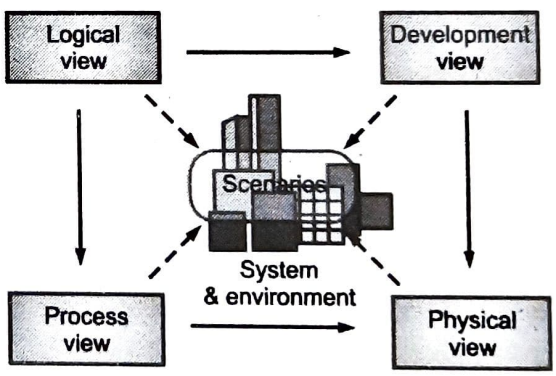


Fig. 3.6.2 : Architectural Views

- **Logical view :** The logical view is concerned with the functionality that the system provides to end-users. UML diagrams used to represent the logical view include, class diagrams, and state diagrams.
- **Process view :** The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system. The process view

addresses concurrency, distribution, integrators, performance, and scalability, etc. UML diagrams to represent process view include the activity diagram.

- **Development view :** The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view. It uses the UML Component diagram to describe system components. UML Diagrams used to represent the development view include the Package diagram.
- **Physical view :** The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer as well as the physical connections between these components. This view is also known as the deployment view. UML diagrams used to represent the physical view include the deployment diagram.
- **Scenarios :** The description of an architecture is illustrated using a small set of use cases, or scenarios, which become a fifth view. The scenarios describe sequences of interactions between objects and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. This view is also known as the use case view.
- The 4 + 1 view model is generic and is not restricted to any notation, tool or design method.

Syllabus Topic : Patterns

3.6.6 Architectural Patterns / Styles

LQ. 3.6.5 Explain different Architectural styles. (5 Marks)

- Architectural Patterns or styles describe a set of systems which are internally linked with each other and share structural and semantic properties.
- In short, the purpose of using architectural styles is to set up a structure for all the components present in a system.
- If an existing architecture is to be re-engineered, then burden of an architectural style results in basic changes in the structure of the system.
- By applying certain restriction on the design space, we can make different style-specific analysis from an architectural style.
- A computer-based system (software is part of this system) demonstrates one of the several available architectural styles.
- Every architectural style defines a system type that includes the following :

1. Computational components like clients, server, filter, and database to execute the required system function.
2. A collection of connectors like procedure call, events broadcast, database protocols, and pipes to offers communication between the computational components.
3. Constraints to describe integration of components to form a system.
4. A semantic model, which allow the software designer so as to recognize the characteristics regarding the system as a whole by the way of studying the characteristics of its components.

- Following are the common architectural styles :

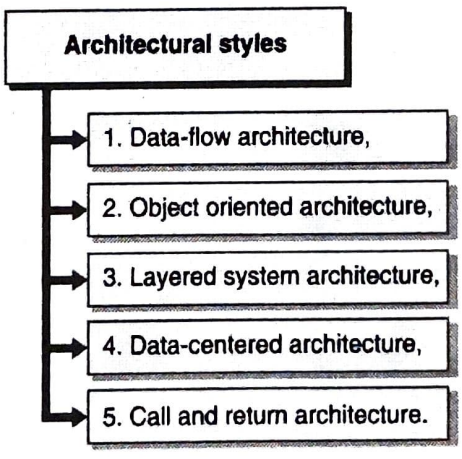
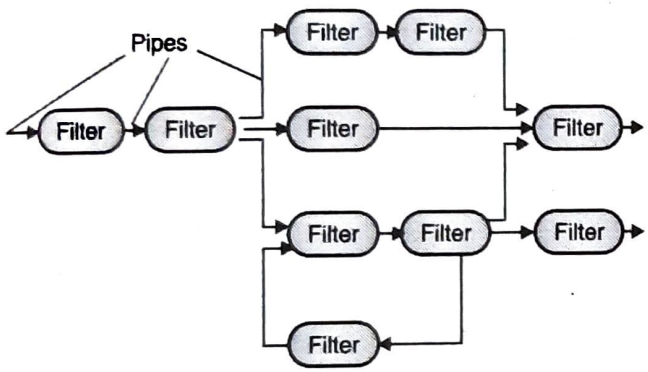


Fig. 3.6.3 : Architectural Patterns

1. Data-flow Architecture

- The purpose of data-flow architecture is to accept some inputs and transforms it into the required outputs by applying a sequence of transformations.
- Each component called as filter, transforms the data and sends this transformed data to other filters for additional processing with the help of the connector called as pipe.
- Every filter operates as an independent entity, that is, it is not focused on the filter which is generating or consuming the data.
- A pipe is a unidirectional channel which transfers the data received on one end to the other end.
- It does not change the data in anyway; it just delivers the data to the filter on the receiver end.



(a) Pipes and filters

(b) Batch sequential

Fig. 3.6.4 : Data flow architecture

- Most of the times, the data-flow architecture reproduce a batch sequential system.
- In this system, a batch of data is accepted as input and then a sequence of sequential filters are applied to transform this data.
- Example of this architecture is UNIX shell programs.
- In these programs, UNIX processes act as filters and the file system by which UNIX processes communicates act as pipes.

Advantages of data-flow architecture

1. It supports maintainability, reusability and modifiability.
2. Concurrent execution is supported by this style.

Disadvantages of data-flow architecture

1. It frequently degenerates to batch sequential system.
2. The application which needs more user interactions is not supported by data-flow architecture.
3. It is not easy to coordinate two different but related streams.

2. Object-oriented Architecture

- In this architectural style, the components of a system wrap data and operations into single unit, which are used to change the data.
- In this style, components are represented as *objects* and they communicate with each other through methods (connectors).

Characteristics of object-oriented architecture

1. Objects preserve the integrity of the system.
2. An object is not aware of other objects' representation.

Advantages of object-oriented architecture

1. It allows designers to divide a problem into a group of independent objects.
2. The implementation detail of objects is known to other object so that they can be changed without affecting other objects.

3. Layered System Architecture

UQ. 3.6.6 Explain layered system architecture with neat diagram.
SPPU - Dec. 17, Dec. 18, 5 Marks

- In layered architecture, every layer is responsible for performing a well-defined set of operations.
- These layers are organized in a hierarchical manner, where every layer is built upon one below another.

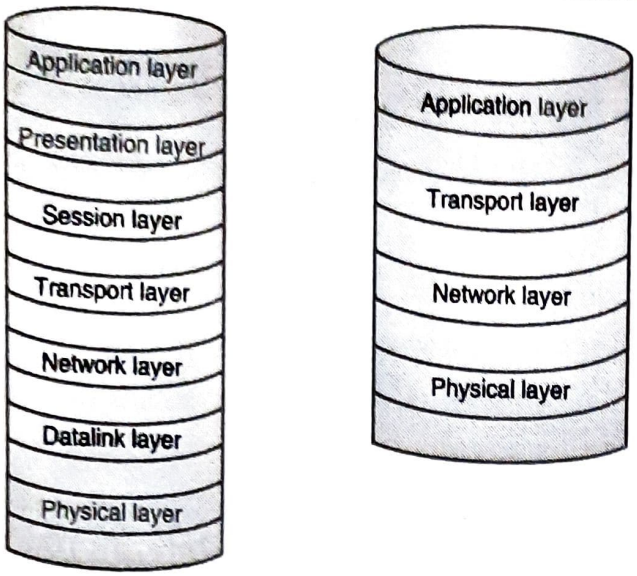


Fig. 3.6.5 : OSI and Internet Protocol Suite

- Each layer offers a set of services to the layer which is located above it and acts as a client to the layer which is located below it.
- The interaction among layers is provided through protocols that describe a set of rules to be followed throughout the interaction.
- One common example of this architectural style is OSI-ISO (Open Systems Interconnection-International Organization for Standardization) communication system.

4. Data-centered Architecture

UQ. 3.6.7 Explain detail Data-centered Architectural Style. **SPPU - May 18, 5 Marks**

- A data-centered architecture has two unique components :
  1. Central data structure
  2. Collection of client software
- A central data structure is also known as data store.
- The data store such as database or a file demonstrates the current state of the data where as the client software performs various operations such as add, delete, update, etc. on the data which is stored in the data store.
- Sometimes the data store permits the client software to access the data independent of any changes or the actions of other client software.
- In this architectural style, new components related to clients can be added and existing components can be changed easily which does not require change in other clients.
- This is because client components work without depending upon any other components.
- A variation of this architectural style is blackboard system in which the data store is transformed into a blackboard that informs the client software when the data is modified.
- Additionally, the information can be transferred between the clients with the help of blackboard components.

Advantages of the data-centered architecture

1. Data repository is not dependant of the clients.
2. Clients work independent of each other.
3. Adding new clients can be easy.
4. Modification can be very easy.
5. It accomplishes data integration in component-based development with the help of blackboard.

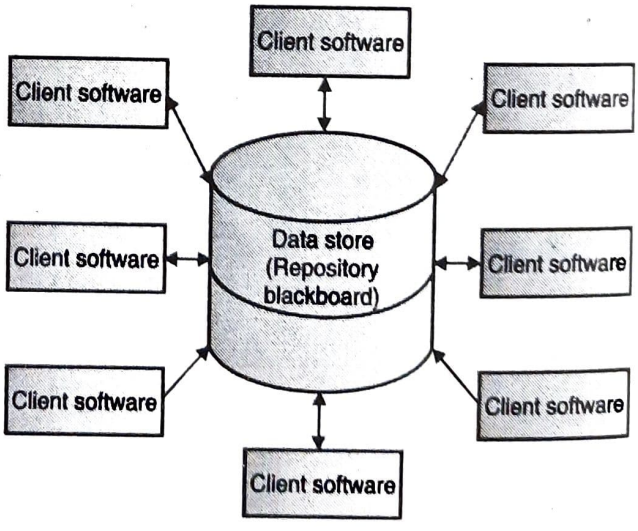


Fig. 3.6.6 : Data-centered architecture

5. Call and Return Architecture

- A call and return architecture allows software designers to get such a program structure where modification is easy.
- This architecture style contains following two sub-styles.

A. Main program/subprogram architecture

In this style, function is divided into a controlled hierarchy where the main program contains and executes a number of program components, where as the contained components may in turn executes other components.

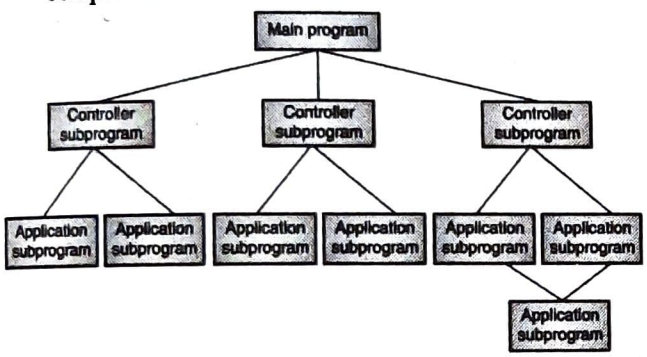


Fig. 3.6.7 : Main program/subprogram architecture

B. Remote procedure call architecture

In this style, components of the main program or subprogram architecture are distributed over a network across number of computers which can be accessed remotely.

LQ. 3.6.8 Explain architectural design for e-commerce System. (5 Marks)

Description

- Any member can register and view available products
- Only registered member can purchase multiple products regardless of quantity.
- ContactUs page is available to contact Admin for queries.
- There are three roles available : Visitor, User and Admin.
- Visitor can view available products.
- User can view and purchase products.
- An Admin has some extra privilege including all privileges of visitor and user.
- Admin can add products, edit product information and remove product.
- Admin can add user, edit user information and can remove user.
- Admin can ship order to user based on order placed by sending confirmation mail.

Using the code

- Attach the database in your "SQL Server Management Studio Express".
- Run the application on Microsoft Visual Studio as web site.
- Locate the database.

MasterPage details

- OnlineShopping Master Page (Similar MasterPage for Visitor, User and Admin)

Web Pages details

- Home Page
- AboutUs Page
- Clothing Page
- OrderUs Page
- ContactUs Page
- Admin Page
- Login Page
- Register Page
- Track

Project Detail

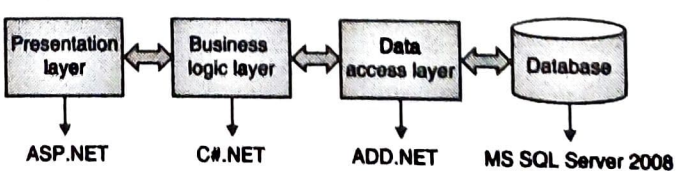


Fig. 3.6.8 : Online shopping - Three Layer Architecture

System requirements

Use-case diagram

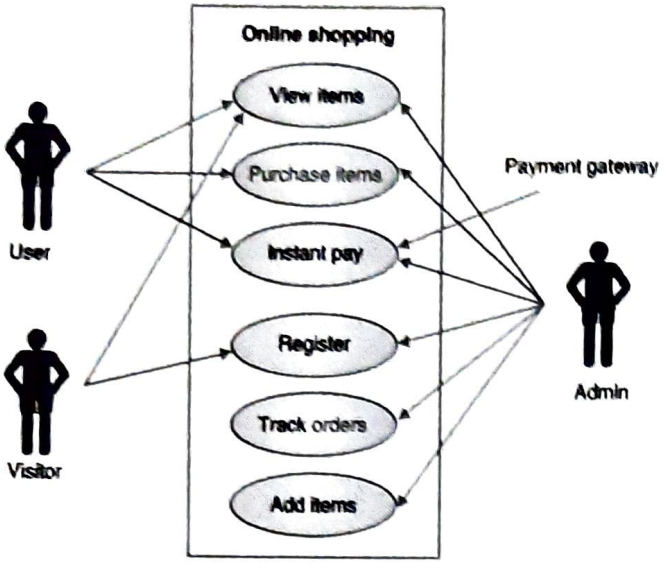


Fig. 3.6.9

Syllabus Topic : Application Architectures

3.6.7 Architectural Patterns / Styles

LQ. 3.6.9 Write note on Architectural Patterns. **(4 Marks)**

- Application systems are designed to meet an organisational need.
- As businesses have much in common, their application systems also tend to have a common architecture that reflects the application requirements.
- A generic architecture is configured and adapted to create a system that meets specific requirements.
- Use of application architectures.
- As a starting point for architectural design.
- As a design checklist.
- As a way of organising the work of the development team.
- As a means of assessing components for reuse.
- As a vocabulary for talking about application types.

Application types

- **Data processing applications :** Data driven applications that process data in batches without explicit user intervention during the processing.
- **Transaction processing applications :** Data-centred applications that process user requests and update information in a system database.
- **Event processing systems :** Applications where system actions depend on interpreting events from the system's environment.
- **Language processing systems :** Applications where the users' intentions are specified in a formal language that is processed and interpreted by the system.

Syllabus Topic : Modeling Component Level Design

3.7 Modeling Component Level Design

LQ. 3.7.1 Write note on Component Level Design. **(5 Marks)**

- The main function of component level design is to define data structures, algorithms, interface characteristics as well as communication mechanisms for all the present software components which are identified in the phase of architectural design.
- The phase of component level design is expected when the data, architectural, and interface designs are completed.
- The software is represented by the component-level design in such a manner which lets the designer to review it for correctness as well as consistency prior to its built.
- The work product generated is considered as a design for all of the software components. It is represented with the help of graphical, tabular, or text-based notation.

- Design walkthroughs are carried out to observe accuracy of the data transformation or control transformation which has been allocated to all of the components in the earlier design steps.

Syllabus Topic : Component

3.7.1 Component

- A **component** is a modular, deployable, replaceable part of a system which encapsulates implementation and exposes a set of interfaces.
- As per object-oriented view a component contains a set of collaborating classes.
- As per traditional view, a component (or module) resides in the software and serves one of three roles as follows :
  - Control components coordinate invocation of all other problem domain components
  - Problem domain components implement the functionality needed to the customer
  - Infrastructure components implement the functionality required to support the processing needed in a domain application.
- Process-Related view focus on building systems out of existing components which are selected from a catalog of reusable components as a way of populating the architecture.

Object-Oriented Component Design

- Put emphasis on describing the domain oriented analysis classes as well as the definition of infrastructure classes.
- Comprehensive information of class attributes, operations, and interfaces is necessary before beginning construction activities.

Syllabus Topic : Designing Class Based Components

3.7.2 Designing Class Based Components

Principles for Designing Class-based Components

UQ. 3.7.2 Explain object oriented view of component level design with suitable example. SPPU - May 18, 6 Marks

- ***Open-Closed Principle (OCP)*** : Extensions should be allowed in the class but not modification.
- **Liskov Substitution Principle (LSP)** : It should be possible to substitute subclasses for their base classes.
- **Dependency Inversion Principle (DIP)** : Depend on abstractions, do not depend on concretions.
- **Interface Segregation Principle (ISP)** : Multiple client oriented interfaces are considered as better compared to one general purpose interface.
- **Release Reuse Equivalency Principle (REP)** : The granule of reuse is the granule of release.
- **Common Closure Principle (CCP)** : Classes that change together belong together.
- **Common Reuse Principle (CRP)** : Classes that can't be used together should not be grouped together.

Component-Level Design Guidelines

UQ. 3.7.3 Explain guidelines for component level design. SPPU - Dec. 17, 5 Marks

1. Components

- Establish naming conventions in the phase of architectural modeling.
- Names of Architectural components should be meaningful for the stakeholders.
- Names of Infrastructure components must have implementation specific meanings.

- The nature of components can be identified with the help of UML stereotypes.

2. Interfaces

- Use of lollipop representation is considered as better as compared to formal UML box and arrow notation.
- For the purpose of consistency, the flow of interfaces should be from the left-hand side of the component box.
- Display only those interfaces which are relevant to the component under construction.

3. Dependencies

- To improve readability it should be better to have the model dependencies from left to right and inheritance from bottom (child classes) to top (parent classes).
- Interfaces must be used to represent component interdependencies rather than component to component dependencies.

Syllabus Topic : Conducting Component Level Design

3.7.3 Conducting Component Level Design

LQ. 3.7.4 Write short note on component Level Design. (5 Marks)

1. Identify all of the respective design classes which are corresponding to the problem domain.
2. Identify all of the respective design classes which are corresponding to the infrastructure domain.
3. Elaborate all of the design classes which are not acquired as reusable components.
   (a) Specify details regarding the message when there is collaboration of classes or components.
   (b) Identify suitable interfaces for all the components.
   (c) Elaborate attributes and define data types and data structures which are necessary for their implementation.
   (d) Describe processing flow of all the operations thoroughly.
4. Make the Identification of persistent data sources such as databases and files and recognize the classes which are necessary to manage them.
5. Develop and elaborate behavioral representations for all the classes or components.
6. Elaborate deployment diagrams for the purpose of providing additional implementation details.
7. Factor all of the component-level diagram representations and consider alternatives.

☞ Object Constraint Language (OCL)

- Complements UML by permitting a developer to use formal grammar as well as syntax for the purpose of constructing unambiguous statements regarding design model element.
- Parts of OCL language statements :
  1. A context which indicates the restricted situation in which the statement is considered as valid.
  2. A property which represents few characteristics of the context.
  3. An operation which manipulates or qualifies a property.
  4. Keywords which are used for the purpose of specifying conditional expressions.

☞ Conventional or Structured Component Design

- Each block of code has a single entry at the top.
- Each block of code has a single exit at the bottom.
- Only three control structures are required : sequence, condition (if-then-else), and repetition (looping).
- Reduces program complexity by enhancing readability, testability, and maintainability.

☞ Design Notation

- **Flowcharts :** Arrows for flow of control, diamonds for decisions, rectangles for processes.
- **Decision table :** Subsets of system conditions and actions are associated with each other to define the rules for processing inputs and events.
- **Program Design Language (PDL) :** Structured English or pseudocode used to describe processing details.

☞ Program Design Language Characteristics

- Predetermined syntax with keywords provided for the purpose of representing all structured constructs, data declarations as well as module definitions.
- Free syntax of natural language for the purpose of describing processing features.
- Data declaration facilities for simple as well complex data structures.
- Subprogram definition as well as invocation facilities.

☞ Design Notation Assessment Criteria

- **Modularity :** Notation provide support for development of modular software.
- **Overall simplicity :** Easy to learn, easy to use, easy to write.
- **Ease of editing :** Easy to make changes in design representation whenever required.
- **Machine readability :** Notation can be input directly into a computer-based development system.
- **Maintainability :** Maintenance of the configuration in general engage maintenance of the procedural design representation.
- **Structure enforcement :** Enforces the use of structured programming constructs.
- **Automatic processing :** Allows the designer to verify the correctness and quality of the design.
- **Data representation :** Ability to represent local as well as global data directly.
- **Logic verification :** Automatic logic verification improves testing competence.
- Easily converted to program source code (makes code generation quicker).

Syllabus Topic : User Interface Design

3.8 User Interface Design

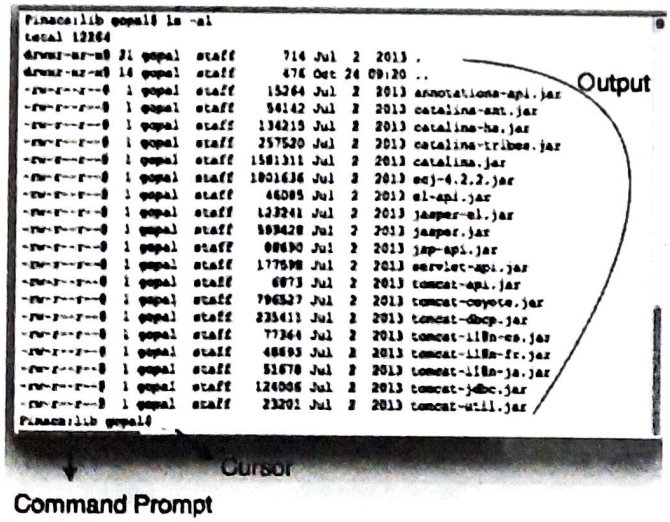
LQ. 3.8.1 Write short note on User Interface Design. (5 Marks)

- User interface is considered as a front-end application view which is interacted by the user so as to use the software.
- User is able to manipulate as well as control the software and hardware with the help of user interface.
- Nowaday, user interface appears in each and every place where there is existence of digital technology, right from desktops, smart phones, cars, music players, airplanes, ships etc.
- User interface is an integral component of software and is designed in such a manner that it should be able to provide the user insight of the software.
- UI offers a fundamental platform for the communication of user and computer.
- The form of UI can be graphical, text-based, audio-video based. It is usually depends upon the underlying platform (hardware and software combination).
- The popularity of software depends upon following aspects :
  - o Attractive
  - o Simple to use
  - o Responsive in short time
  - o Clear to understand
  - o Consistent on all interfacing screens
- UI is broadly divided into two categories :
  1. Command Line Interface
  2. Graphical User Interface

3.8.1 Command Line Interface (CLI)

- CLI was considered as a great tool of communication with computers in previous days.
- Now-a-days also CLI is the first choice of number of technical users as well as programmers. CLI is considered as minimum interface which software can provide to its users.
- A command prompt is provided by the CLI which is a place where the user writes instructions and feeds to the system.
- Here there is need for the user to remember the syntax of command and its use. In previous days CLI were not programmed to handle the user errors effectively.
- A command is considered as a text-based reference to set of instructions, which should be executed by the respective system.
- There are several methods such as macros, scripts which make it simple for the user to operate.
- Very less amount of computer resource is used by the CLI in comparison with GUI.

CLI Elements



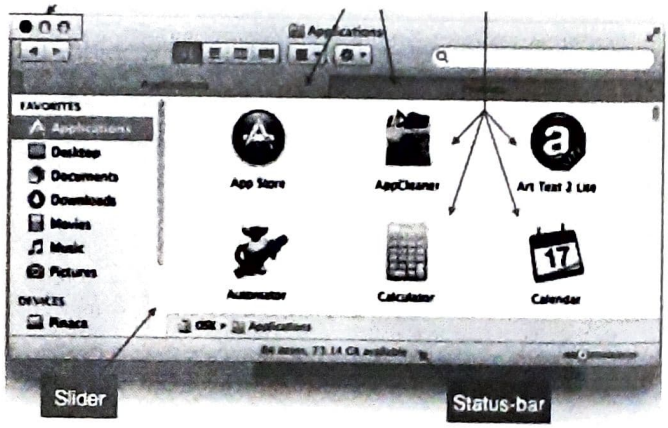
- There are following elements in a text-based command line interface :
- **Command Prompt :** It is text-based notifier which usually displays the context where the user is working. Software system generates it.
- **Cursor :** It is a small horizontal line or a vertical bar with the height of line. It is used to represent position of character at the time of typing. Cursor is usefully found in blinking state. It moves when user writes or deletes text.
- **Command :** A command is nothing but an executable instruction. There may be one or more parameters to it. Output of command is displayed inline on the screen. After displaying output, command prompt is set to the next line.

3.8.2 Graphical User Interface (GUI)

- Graphical User Interface offers graphical means to the use for interacting with the system. GUI can be combination of both hardware and software. Using GUI, user is able to interpret the software.
- In general GUI is considered as more resource consuming as compared to CLI. With the help of latest advanced technology, the programmers and designers are able to create complex GUI designs which perform with better efficiency, accuracy and speed.

3.8.2(A) GUI Elements

- GUI provides several components for the purpose of interacting with software or hardware.
- All of these graphical components give ways to interact with the system.
- Following are some important elements of GUI :



1. Window

- It is the area in which contents of application are displayed.
- To represent file structure, the Window contents can be displayed in the form of icons or lists.
- In an exploring window, the navigation is very easy for user.
- Windows provide facility to minimize, resize or maximize. It is possible to move them anywhere on the screen. A window may contain another window (child window) of the same application inside it.

2. Tabs

- When an application executes more than one instances of itself, then those instances appear on the screen as separate windows.

3. Tabbed Document Interface

- Provides the facility to open more than one document in the same window. This interface is used to view preference panel in application. Now-a-days this feature is used by almost all web-browsers.

4. Menu

- Menu is considered as an array of standard commands, combined together and located at a prominent place (mostly top) inside the application window. It is also possible to program the menu to appear or hide on mouse clicks.

5. Icon

- An icon is small picture which is used to represent an application. These icons are clicked or double clicked to open the application window.

6. Cursor

- In GUI cursors are used to represent interacting devices like mouse, touch pad, digital pen, etc.
- In almost real-time, the instructions given by hardware are followed by the cursor. Cursors which are also called as pointers are used to select menus, windows and other application features.

3.8.2(B) Application Specific GUI Components

A GUI of an application contains following elements :

1. Application Window

- Most of the application windows use the constructs provided by underlying OS but some have their own customized windows to contain the contents of application.

2. Dialogue Box

- Dialog box is a child window which has message for the user and request for some action to be taken. For Example : A dialog box is displayed by an application to get confirmation from user to delete a file.

3. Text-Box

- Used to accept input from user in text format.

4. Buttons

- Used to submit inputs to the application.

5. Radio-button

- Displays list of options for selection. Only single radio button can be selected among all offered.

6. Check-box

- Its functionality is same as of the radio button but it allows multi selection.

7. List-box

- Provides list of available items using single control. Multiple items can be selected.

Syllabus Topic : Interface Design Steps and Analysis

3.8.2(C) Interface Design Steps and Analysis

- Several activities are carried out for designing user interface. The process of GUI design and implementation is same as of the SDLC (Software Development Life Cycle).

- For GUI implementation there are number of models available such as Waterfall, Iterative or Spiral Model.
- Following GUI specific steps should be fulfilled by the model used for GUI design and development.

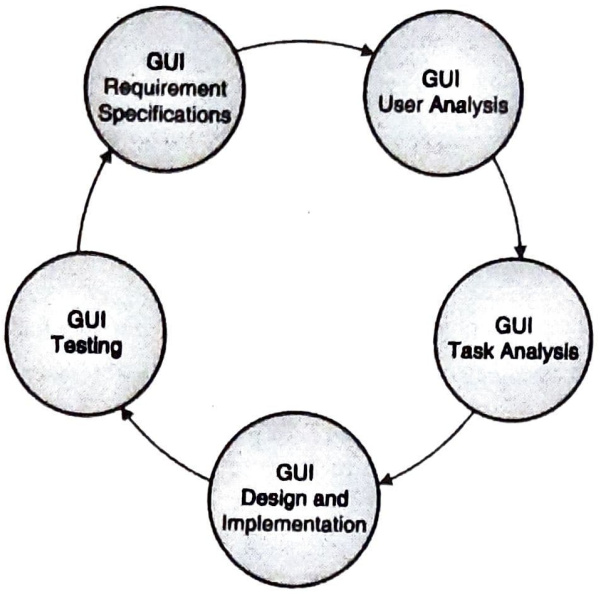


Fig. 3.8.1 : GUI Steps

- **GUI Requirement Gathering :** The designers need the list of list of all functional as well as non-functional requirements regarding GUI. These requirements are accepted from users and their existing software solution.
- **User Analysis :** The designer do the study regarding who will actually use the software GUI. The target audience is an important aspect since the design details change according to the knowledge as well as competency level of the end user.
- If user looks to be techno savvy then advanced and complex GUI can be developed.
- For a new user, more informative data is included regarding how-to of software.
- **Task Analysis :** Designers have an important responsibility to analyze the task which is to be done by the respective software solution.
- Here in case of GUI, it is not important how it will be done. It is possible to represent the task in hierarchical manner considering one major task at the top and dividing it further into smaller sub-tasks.
- For GUI presentation, goals are provided by the tasks.
- In the software the flow of GUI contents is decided by the flow of information among sub-tasks.
- **GUI Design and implementation :** Designers after getting complete information regarding requirements, tasks and user environment, design the respective GUI and implement the design into code and integrate the GUI with currently working or any dummy software in the background. Then self-testing is implemented by the developers.
- **Testing :** There are several ways to implement the GUI testing. Organization has options such as in-house inspection, straight involvement of end users and prior release of beta version. Important aspects verified in testing are usability, compatibility, user acceptance etc.

3.8.2(D) GUI Implementation Tools

- There are number of tools available for designers to create entire GUI on a single mouse click. Out of them some tools can be easily integrated into the software environment.
- A huge set of GUI controls is provided by GUI implementation tools.
- The code is modified by the designers for software customization.
- There are diverse segments of GUI tools in accordance with their use and platform.

Example

- Mobile GUI, Computer GUI, Touch-Screen GUI etc.
- Following are some tools which are used to build GUI :
  - FLUID
  - AppInventor (Android)
  - LucidChart
  - Wavemaker
  - Visual Studio

3.8.2(E) Principles of user Interface Design

UQ. 3.8.2 Explain the user interface design principles. **SPPU - Dec. 17, 5 Marks**

- The principles of user interface design are intended to improve the quality of user interface design :

1. **The structure principle :** Design should organize the user interface purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with overall user interface architecture.
2. **The simplicity principle :** The design should make simple, common tasks easy, communicating clearly and simply in the user's own language, and providing good shortcuts that are meaningfully related to longer procedures.
3. **The visibility principle :** The design should make all needed options and materials for a given task visible without distracting the user with extraneous or redundant information. Good designs don't overwhelm users with alternatives or confuse with unneeded information.
4. **The feedback principle :** The design should keep users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.
5. **The tolerance principle :** The design should be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing, while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions reasonable.
6. **The reuse principle :** The design should reuse internal and external components and behaviors, maintaining consistency with purpose rather than merely arbitrary consistency, thus reducing the need for users to rethink and remember.

Syllabus Topic : The Golden Rules

3.8.2(F) The Golden Rules

UQ. 3.8.3 Enlist the golden rules of User Interface Design. **SPPU - Aug. 17, 4 Marks**

- Designers need to solve problems every day, and finding the right solution involves in-depth research and carefully planned testing.
- The following rules are known as the golden rules for GUI design :

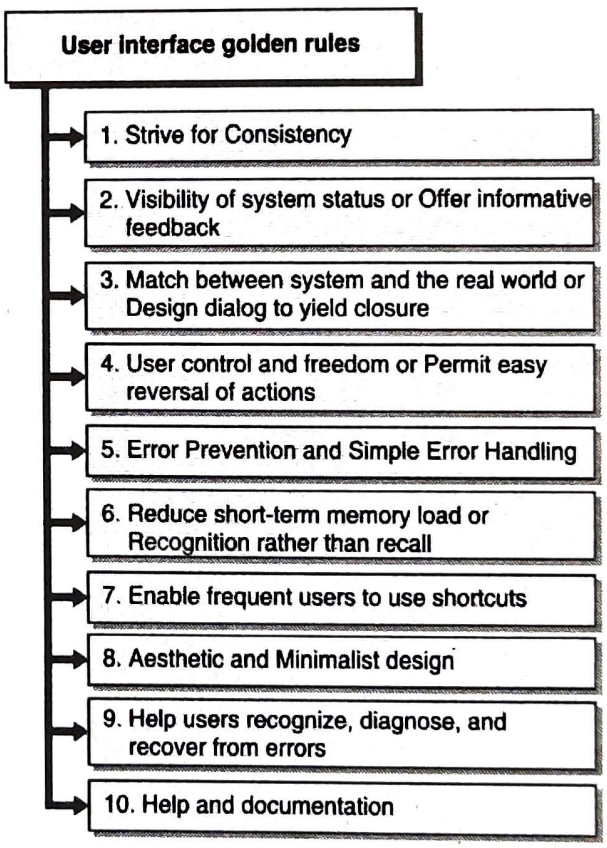


Fig. 3.8.2 : User interface golden rules

1. Strive for Consistency

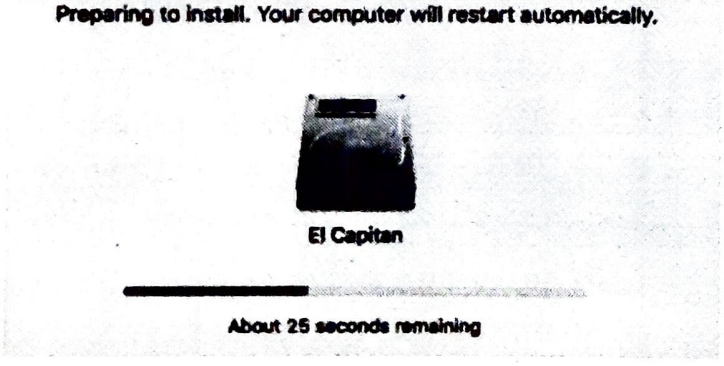
- Users should not have to wonder whether different words, situations, or actions mean the same thing. Do not confuse your user keep *words and actions* consistent.

- Use "The Principle of Least Surprise".
- **Example :** Car Climate Control

- In other words, use all elements across your application consistently. For example a certain style of button should always do the same thing, or navigation should function logically, going deeper in hierarchy.
- **Consistency of :**
  - Workflow / Processes
  - Functionality
  - Appearance
  - Terminology

2. Visibility of system status or Offer informative feedback

- The system should always keep users informed about what is going on. Through appropriate feedback in a reasonable time. Don't keep the users guessing tell the user what's happening.
- **Example :** OS Installation status



- The user wants to be in control, and trust that the system behaves as expected. It could be even said that users don't like surprises.
- For frequent and minor actions, the response can be modest, while for infrequent and major actions, the response should be more substantial.
- **Feedback**
  - Relevant
  - Fits importance and urgency
  - Comprehensible and meaningful
  - Within appropriate context (time and place)

3. Match between system and the real world or Design dialog to yield closure

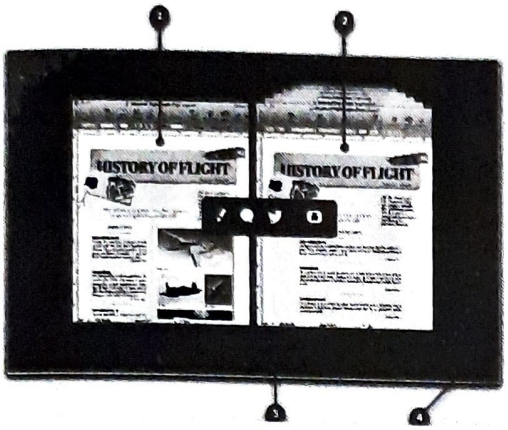
- Again, the less the users have to guess the better. The system should speak the users' language (use words, phrases and concepts familiar to the user), rather than special system terms.
- **Example :** Payment proceeding (by Ramakrishna).
- Sequence of actions should be organized into groups with a beginning, middle and end. When a process is finished, remember to display a notification message.
- Let the user know that he has done all that's needed.

Design

- Grouping of actions.
- Explicit completion of an action.
- Well-defined options for the next step.

4. User control and freedom or Permit easy reversal of actions

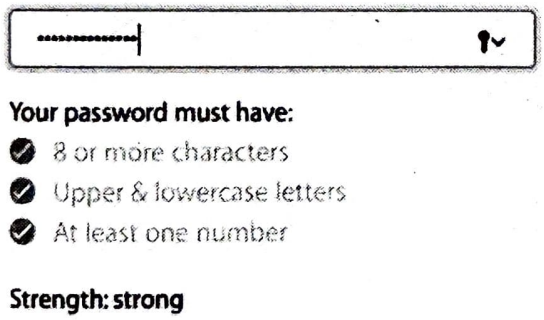
- Shneiderman puts it nicely "This feature relieves anxiety, since the user knows that errors can be undone; it thus encourages exploration of unfamiliar options."
- **Example :** Document History



- In applications, this refers to the undo and redo functionality. Clearly mark an "*emergency exit*" to leave the unwanted state without having to go through an extended dialogue.
- **Reversal of actions**
  - No interference with workflow
  - More freedom for the user
  - Single-action undo and action history

5. Error Prevention and Simple Error Handling

- Users hate errors, and even more so hate the feeling that they themselves have done something wrong. Either *eliminate* error-prone conditions or *check for them* and notify users about that before they commit to the action.
- **Example :** Password Enter.



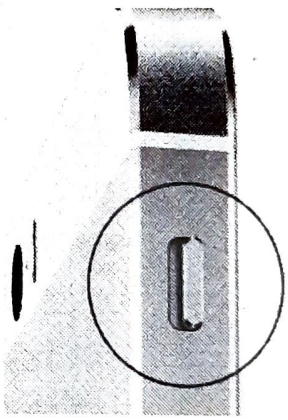
- As much as possible, design the system so the user cannot make a serious error.
- If an error is made, the system should be able to detect the error and offer a simple, comprehensive mechanism for handling the error.

Error Prevention

- Error prevention over error correction
- Automatic detection of errors
- Clear error notifications
- Hints for solving the problem

6. Reduce short-term memory load or Recognition rather than recall

- As Nielsen says, recognizing something is easier than remembering it. Minimize the user's memory load by making objects, actions, and options *available*.
- The user should not have to remember information from one part of the dialogue to another. Instructions should be visible.
- **Example :** Ring/Silent switch.



- Use iconography and other visual aids such as themed coloring and consistent placement of items to help the returning users find the functionalities. Reduce memory load
  - App has a clear structure
  - "Recognition over recall"
  - Implicit help
  - Visual aids

7. Enable frequent users to use shortcuts

- Allow users to tailor (manipulate and personalize) frequent actions.
- Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.

- **Example :** Click Commands and Shortcuts.

New Window	⌘N
New Private Window	⇧⌘N
New Tab	⌘T
Open File...	⌘O
Open Location...	⌘L

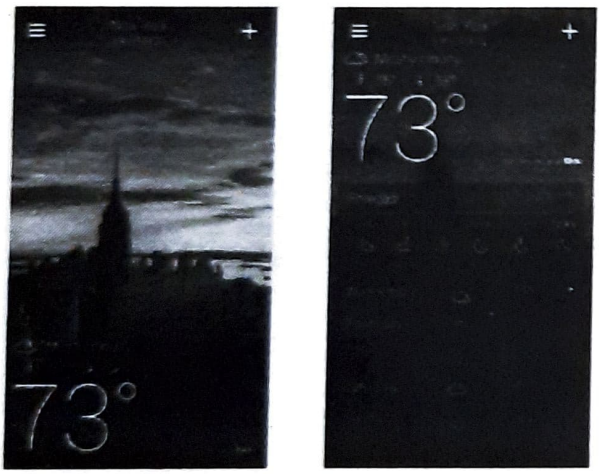
Shortcuts

- Keyboard shortcuts
- "Power User" features
- Action automation

8. Aesthetic and Minimalist design

- Minimalist doesn't mean limited. All information should be *valuable* and *relevant*.

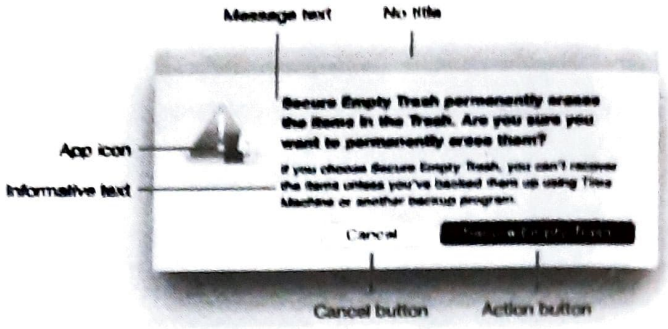
Example : Simple Interface



- Simplify interfaces by removing unnecessary elements or content that does not support user tasks.

9. Help users recognize, diagnose, and recover from errors

- Error messages should be expressed in *plain language* (don't use system language to describe what the system is doing), precisely *indicate* the problem, and constructively *suggest* a solution.
- **Example :** Plain Message



- Tell the user clearly and plainly what's happening both on the background and when they perform an action.

10. Help and documentation

- Even though it is better if the interface can be used without documentation, it is necessary to provide help and documentation.
- Any help information should be *easy to search*, *focused* on the user's task, list of *concrete steps*, and *not too large*.

3.8.3 User Interface Design Issues

UQ. 3.8.4 Explain the user interface design issues.

SPPU - May 18, Dec. 18, 4 Marks

- There are several issues regarding user interface design :

1. Response time

- *System response time* is considered as the basic complaint for several interactive applications. Usually, system response time is measured from the particular point at which the any control action is performed by the end user (such as return key or mouse click) until the software responds with expected output or action.
- There are two essential characteristics of system response time : length and variability.
- When the *length* of system response is too long, it results in the user frustration and stress. On the other hand, a very short response time can also be problematic if the user is being paced by the interface. A fast response may force the user to make hurry which may leads to happen some mistakes.

- *Variability* refers to the deviation from average response time, and in several ways, it is considered as the most significant response time characteristic. When the variability is low, it enables the user to set an interaction rhythm, even when response time is comparatively long.

2. Help facilities

- Nowadays all of the users which use interactive, computer-based system needs help all the time. In some situations, a simple question addressed to a knowledgeable and experienced colleague can solve the issue. Whereas sometimes there is necessity of detailed research in a multivolume set of "user manuals".
- In several cases, however, latest software offers on-line help facilities which help user to get a question answered or resolve a problem without need of leaving the respective interface.
- There are basically two different types of help facilities present: integrated and add-on.
- An *integrated help facility* is incorporated into the respective software from the initial stage. It is often considered as context sensitive, which enables the user to choose from the topics which are related to the actions which are currently being performed. Apparently, this decreases the time needed for the user to get help and increases the "friendliness" of the interface.
- An *add-on help facility* is added to the respective software after the system has been completely developed. In number of ways, it is actually an on-line user's manual having limited query capability. The user may need to search through a list of several topics to get proper guidance, often doing several false starts and getting much irrelevant information.

3. Error Handling

- Error messages or warnings are always considered as "bad news" received by users of interactive systems when something has gone wrong. Sometimes, error messages or warnings pass on only useless or misleading information and frustrate the user.
- Yet, the error message provides no real indication of what is wrong or where to look to get additional information. An error message presented in this manner does nothing to assuage user anxiety or to help correct the problem.

4. Menu and command Labeling

- Some years ago, the typed command was the most common mode of communication between user and system software and was in general used for all the types of applications. Nowadays, the use of window-oriented, point and pick interfaces has decreases dependence on typed commands, but number of power-users continue to prefer a command-oriented mode of interaction.
- A number of design issues arise when typed commands are provided as a mode of interaction :
  - Will every menu option have a corresponding command?
  - What form will commands take? Options include a control sequence (e.g., alt-P), function keys, or a typed word.
  - How difficult will it be to learn and remember the commands? What can be done if a command is forgotten?
  - Can commands be customized or abbreviated by the user?

Syllabus Topic : Design Evaluation

3.8.4 Design Evaluation

LQ. 3.8.5 Write short note on Design Evaluation. **(5 Marks)**

- After generation of an operational user interface prototype, it is important to evaluate it for the purpose of determining whether it fulfills the requirements of the user.

- Evaluation can span a formality spectrum which ranges from an informal "test drive," where a end-user gives spontaneous feedback to a formally designed study which takes reference of statistical methods for the purpose of evaluating questionnaires completed number of end-users.
- The Fig. 3.8.3 illustrates the form of user interface evaluation cycle. After completion of the design model, a first-level prototype is generated.
- User evaluates the prototype, and give the designer direct comments regarding the efficacy of the interface. Also, if there is use of formal evaluation techniques such as questionnaires, rating sheets, then it is possible for the designer to extract information from these data.

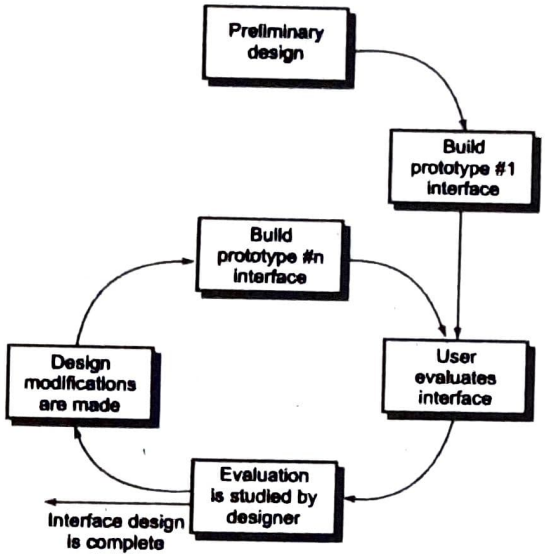


Fig. 3.8.3 : The interface design evaluation cycle

- Modifications in design are done based on user input and the next level prototype is created.
- There is continuation of the evaluation cycle until there is no need of any further modifications to the interface design.
- The prototyping approach is considered as effective, but is it possible to evaluate the quality of a user interface before a prototype is built? If possible issues can be uncovered and corrected in early stage, the number of loops through the evaluation cycle will be lessen which leads to decrease in development time.
- If there is generation of a design model of the interface, it is possible to apply number of evaluation criteria during early design reviews :

1. The length and complexity of the written specification of the system and its interface provide an indication of the amount of learning required by users of the system.
2. The number of user tasks specified and the average number of actions per task provide an indication of interaction time and the overall efficiency of the system.
3. The number of actions, tasks, and system states indicated by the design model imply the memory load on users of the system.
4. Interface style, help facilities, and error handling protocol provide a general indication of the complexity of the interface and the degree to which it will be accepted by the user.

- After building the first prototype, it is possible for the designer to collect a variety of qualitative as well as quantitative data which will help in evaluating the interface.
- To gather qualitative data, questionnaires can be distributed to users of the prototype.
- Questions can be all (1) simple yes/no response, (2) numeric response, (3) scaled (subjective) response, or (4) percentage (subjective) response.

Examples are

1. Were the icons self-explanatory? If not, which icons were unclear ?
2. Were the actions easy to remember and to invoke ?
3. How many different actions did you use ?
4. How easy was it to learn basic system operations (scale 1 to 5) ?
5. Compared to other interfaces you've used, how would this rate-top 1%, top 10%, top 25%, top 50%, bottom 50%?

Syllabus Topic : Case Study - Web App Interface Design

3.9 Case Study : Web App Interface Design

UQ. 3.9.1 Enlist and explain the Webapp design principles in detail.

SPPU - Aug. 17, 3 Marks

- Design for WebApps included several types of technical as well as non-technical activities.
- The look and feel of content present in the application is developed as an integral part of graphic design; the aesthetic layout of the respective user interface is generated as part of interface design, whereas the technical structure of the WebApp is created as part of architectural and navigational design.
- In each instance, it is necessary that a design model must be generated prior to beginning of construction, but a good Web engineer always able to recognize that the respective design will evolve as more is understood regarding stakeholder requirements as the WebApp is built.

3.9.1 WebApp Design Principles

- **Anticipation** - A WebApp should be designed in such as manner that it should be able to anticipates the use's next move.
- **Communication** - The interface should be designed in such as manner that it should be able to communicate the status of any activity which has been initiated by the user
- **Consistency** - The color, shape, layout of navigation controls, menus, icons must be consistent throughout the application.
- **Controlled autonomy** - The interface should be designed in such as manner that it should be able to facilitate user movement throughout the WebApp, but it must be implemented in such a manner which enforces navigation conventions that have been set for the application.
- **Efficiency** - The design of the WebApp and its interface should optimize the user's work efficiency, not the efficiency of the Web engineer who designs and builds it or the client-server environment that executes it.
- **Focus** - The WebApp interface along with its content has responsibility to stay focused on the user task(s) at hand.
- **Latency reduction** - The WebApp should be able to implement multi-tasking in a manner that allows the user proceed with work as if the operation has been completed.
- **Learnability** - A WebApp should be designed in such as manner that it should be able to minimize learning time, and once learned, to lessen the amount of relearning needed when the WebApp is revisited.
- **Maintain work product integrity** - A work product such as online form filled by user should be automatically saved so that it will not be lost in case of any error.
- **Readability** - Persons of all ages should be able to read the information presented through the interface.
- **Track state** - When suitable, the user state regarding her interaction must be tracked as well as stored so that a user can logoff and come back later to continue from where she left off.

...Chapter Ends

□□□