Note Junction

Best Note Provider

Note By: Roshan BiSt

UNIT-1
Introduction

Software is set of computer programs and associated documentation. Software engineering is strategy for producing quality software. It is the establishment and use of sound engineering principles in order to obtain economically reliable software and works efficiently on real machines. It is an engineering discipline which is concerned with all aspects of software production.

⊛. Software Types:

i) Generic: These are the systems used for developing a general purpose software. From designing and marketing perspective, this kind of development is very difficult. Large number of users may be using this kind of software. Development team controls the process of generic software development. Word-editing software is it's example.

ii) Custom: (also called bespoke) These are the systems used for developing a software product as per the needs of particular customer. This development does not require marketing, because it is developed for appropriate group of users. Customer determines the process of software development in this type of product. Control system for electronic device is it's example.

⊛. Attributes of Good Software: (OR characteristics of software)

The essential attributes of good software are as follows:-

i) Maintainability: Software must evolve to meet changing need of customer. It is called maintainability. So, software should be written in such a way that, it may evolve changing needs of customer.

ii) Dependability: Dependability of software is a property of software that reflects it's trustworthiness. It is the degree of confidence a user has that the system will operate as they expect and the system will not fail in normal use.

iii) Portability: It refers to the ease with which software developers can transfer software from one platform to another.

iv) Efficiency: Software should not make wasteful use of system resources such as memory, processor cycle. So, efficiency means, responsiveness, processing time, memory utilization.

v) Usability: Software must be easily usable, it should not be too complicated to use. It should have adequate documentation and appropriate interface.

⊛. Advantages of software engineering:

→ Improved quality.
→ Improved reliability
→ Improved productivity
→ Improved requirement specification
→ Improved cost and schedule estimates.
→ Well defined process.

⊛. Importance of software engineering (Why software engineering?):

→ The economics of all developed nations are dependent on software.
→ More and more systems are software controlled.
→ Software engineering is concerned with theories, methods and tools for professional software development.
→ Software engineering expenditure represents a significant fraction of GNP in all developed countries.

⊛. Fundamental Software Engineering Activities:

There are many different kinds of software processes, but each and every one involve following four fundamental activities:

i) Software Specification:- Software specification is the process of understanding and defining what services are required from the system and identifying the constraints on the system operation and development. It is a critical stage, because any error in this stage will lead to later problems in software design and implementation.

ii) Development: It includes software design and implementation. It is the process of converting the system specifications into an executable system.

iii) Validation: It is concerned with building the right system. It is intended to show that a system confirms to its specifications and meets the user expectations.

iv) Evolution: Evolution is the time to time maintenance of the system to meet changing needs of customer with time. So, software should be written in such a way that, it may evolve changing needs of customer.

⊛. Differences between software engineering and computer science:

Software Engineering	Computer Science
i) Software Engineering is the study of how software systems are built.	i) Computer science is the study of how computers perform theoretical and mathematical tasks
ii) It involves the study and application of software only.	ii) It involves the study and application of software and hardware both.
iii) It is the structural process of checking, verifying, finding the errors and bugs according to the need of software and then provide a solution for removing that bug.	iii) It is not a structural process as everything is to be done in a process and requires proper study before executing
iv) It involves some areas of study which are software development, Software testing and Quality assurance.	iv) It involves areas of study which are networking, artificial intelligence, database systems etc.
v) Software Engineering majorly defines architecture and structural properties.	v) Computer science involves the study of both principles and the use of computers.

⊛. Differentiate between software engineering and system engineering:

Software engineering	System engineering
i) Software engineering is an engineering discipline that is concerned with all aspects of software production.	i) System engineering is a field of engineering and engineering management that focus on how to design and manage complex system over their life cycle.
ii) Software engineering highly focuses on implementing quality software.	ii) System engineering highly focuses on the users and domains.
iii) Software engineering includes in computer science or computer based engineering background.	iii) System engineering may ~~cover~~ cover a broader area, entire system development.
iv) Software engineering focus solely on software components.	iv) System engineering focus on hardware engineering
v) Software engineering is newly developed discipline.	v) System engineering is an older discipline

⊛. Challenges of software engineering:

i) **Heterogenity challenge:** Heterogenity means diversity or variety. There are different types of computer and with different kinds of support systems. The heterogenity challenge is the challenge of developing techniques to build software which is flexible to support by most of the systems.

ii) **The legacy challenge:** The legacy challenge is the challenge of maintaining and updating this software in such a way that excessive costs are avoided and essential business services continue to be delivered.

iii) **The delivery challenge:** Software engineering techniques are time-consuming, to achieve better software quality. Most of businesses nowdays want software systems quickly. This shortening delivery time of system for large and complex systems without compromising system quality is called delivery challenge.

iv) Trust challenge: A software is twisted with all aspects of our lives, it is essential that we can trust that software, so the trust challenge is to develop techniques that demonstrate that software can be trusted by its users.

v) Risk challenge: In safety-critical areas such as space, aviation, nuclear power plants, etc. the cost of software failure can be massive because lives are at risk. Dealing with the increased complexity of software need for new applications.

⊛. Cost of software engineering:

The distribution of costs across the different activities in the software process depends on the process used and the type of software that is being developed. For example, real-time software usually requires more extensive validation and testing than web-based systems. So, roughly 60 percent of costs are development costs and 40 percent are testing costs. For customer software evolution costs often exceeds development costs. Distribution of costs depends on the development model that is used.

⊛. Professional software development:

Lots of people write programs. People in business write spreadsheet programs to simplify their jobs, scientists and engineers write programs to process their experimental data, some people write programs for their own interest and enjoyment. However, most software development is a professional activity where software is developed for business purposes. This developed software is maintained and changed throughout its life.

Software engineering is intended to support professional software development, rather than individual programming. A professionally developed software system is often more than a single program, a system may consist of several seperate programs and configuration files that are used to set up these programs. It may include system documentation, which describes structure of the system, user documentation, which explains how to use the system and web sites for users to download recent product information.

⊛. Software engineering diversity:

There are no universal software engineering methods that are suitable for all systems and all companies. Rather, a diverse set of software engineering methods and tools has evolved over the past 50 years. The most significant factor in determining which software engineering methods and techniques are most important is the type of application that is being developed.

We should make as effective use as possible for existing resources. This means that, where appropriate, we should reuse software that has already been developed rather than write new software.

⊛. Internet software engineering:

Rather than local system, the internet is now a platform for running applications. Internet service allows application functionality to be accessed over the internet. With the help of internet, instead of writing software and deploying it on users PC, the software can be developed on web server that can be accessed through browsers. This made it much cheaper to change and upgrade the software as there was no need to install the software on every PC.

⊛. Software engineering ethics:

Following are some software engineering ethics:

i) Confidentiability: We should normally respect the confidentiality of our employees or clients irrespective of whether a formal confidentiability agreement has been signed.

ii) Competence: We should never misrepresent our skills and the level of competency. We should never accept any work which is out of our competency.

iii) Intellectual property rights: We should be aware of local laws governing the use of intellectual property such as patents and copyright.

iv) Computer misuse: We should not use our technical skills to misuse other people's computers. Computer misuse ranges from simple (like game playing on an employer's machine) to extremely serious (dissemination of viruses).

UNIT-2

Software Processes

A software process is a set of related activities that leads to the production of software system. There are many different types of software systems, and there is no universal software engineering method that is applicable to all of them. Consequently, there is no universally applicable software process. The process used in different companies depends on the type of software being developed, the requirements of the software customer, and the skills of the people writing the software. However, although there are many software processes, they all must include, in some form, the four fundamental software engineering activities:

- → Software specification
- → Software development ← these points already discussed in unit 1 in short.
- → Software validation
- → Software evolution.

⊛. Software Process Models:

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective. It represents the order in which the activities of software development will be undertaken. The general process models are as follows:

1) Waterfall model: The waterfall model is a sequential approach, where each fundamental activity of a process represented as a seperate phase, arranged in linear order. Each phase is carried out completely before proceeding to the next. The process is strictly sequential - no backing up or repeating phases. This process is strictly documented and predefined with features expected to every phase of SDLC model.

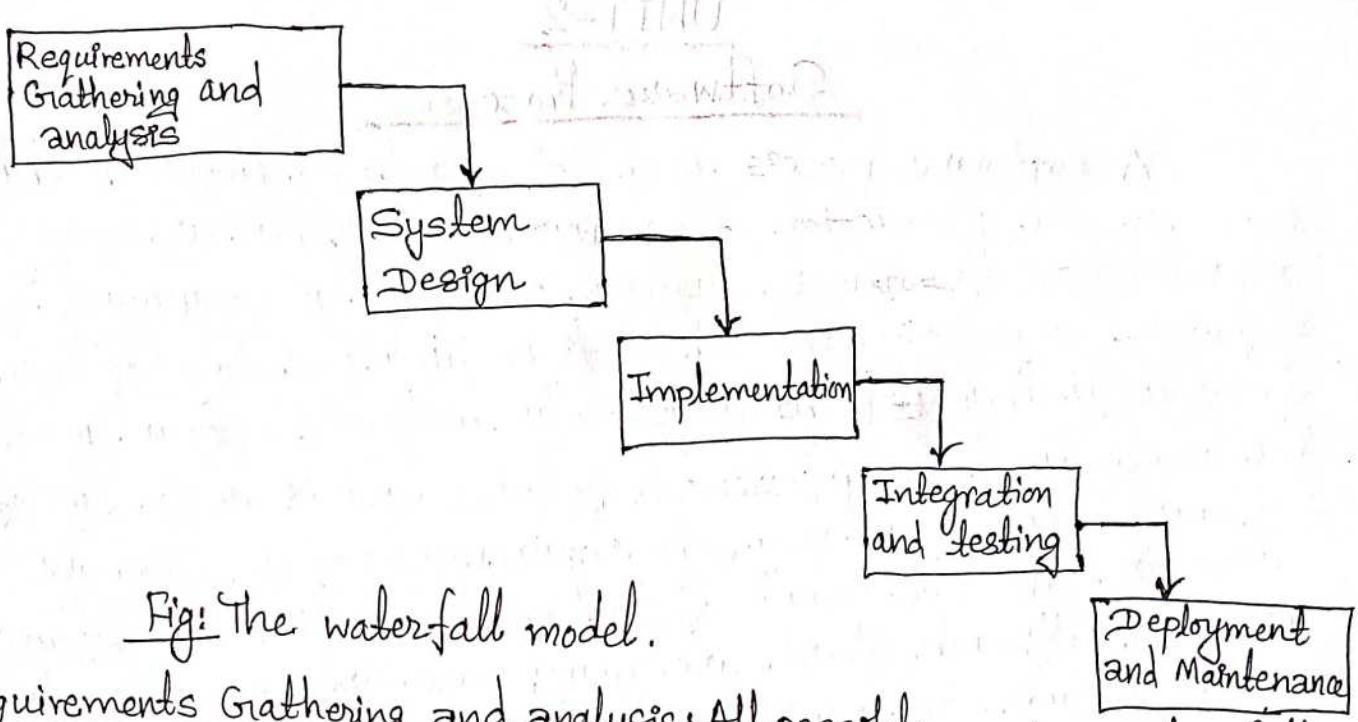


Fig: The waterfall model.

Requirements Gathering and analysis: All possible requirements of the system to be developed are captured in this phase. Then analysis is done and documented in a requirement specification document.

System Design: The requirement specifications from first phase are studied in this phase and design for system is prepared based on requirements gathered and analysed.

Implementation: Now designed system is developed by programmers in this phase. The system is first developed in small programs called units, which are integrated and tested in next phase.

Integration and Testing: All the units developed in the implementation phase are integrated into a system after testing of each unit.

Deployment and Maintenance: Once, the functional and non-functional testing is done; the product is deployed in the customer environment or released in the market. Maintenance is done to fix issues and to enhance the product to some better versions.

Merits of waterfall model:

↳ This model is simple to implement also the number of resources that are required for it is minimal.
↳ The requirements are simple and explicitly declared; they remain unchanged during the entire project development.
↳ The start and end points for each phase is fixed, which makes it easy to cover progress.
↳ Easy to manage due to rigidity of the model.

Demerits of waterfall model:

↳ No working software is produced until late during the life cycle.
↳ This model cannot accept changes in requirements during development.
↳ It becomes tough to go back to the phase.
↳ Since the testing is done at larger stage, risk reduction strategy is difficult to prepare.

2) Incremental development model:

In an incremental model, customers identify, in outline, the services to be provided by the system. They identify which of the services are most important and which are least important to them. A number of delivery increments are then defined, with each increment providing a sub-set of the system functionality. The allocation of services to increments depends on the service priority with the highest priority services delivered first.

Once increment is completed and delivered, customers can put it into services they can experiment with the system that helps to clarify their requirements for later increments and for later versions of the current increment. As new increments are completed, they are integrated with existing increments so that the system functionality improves each delivered increment.

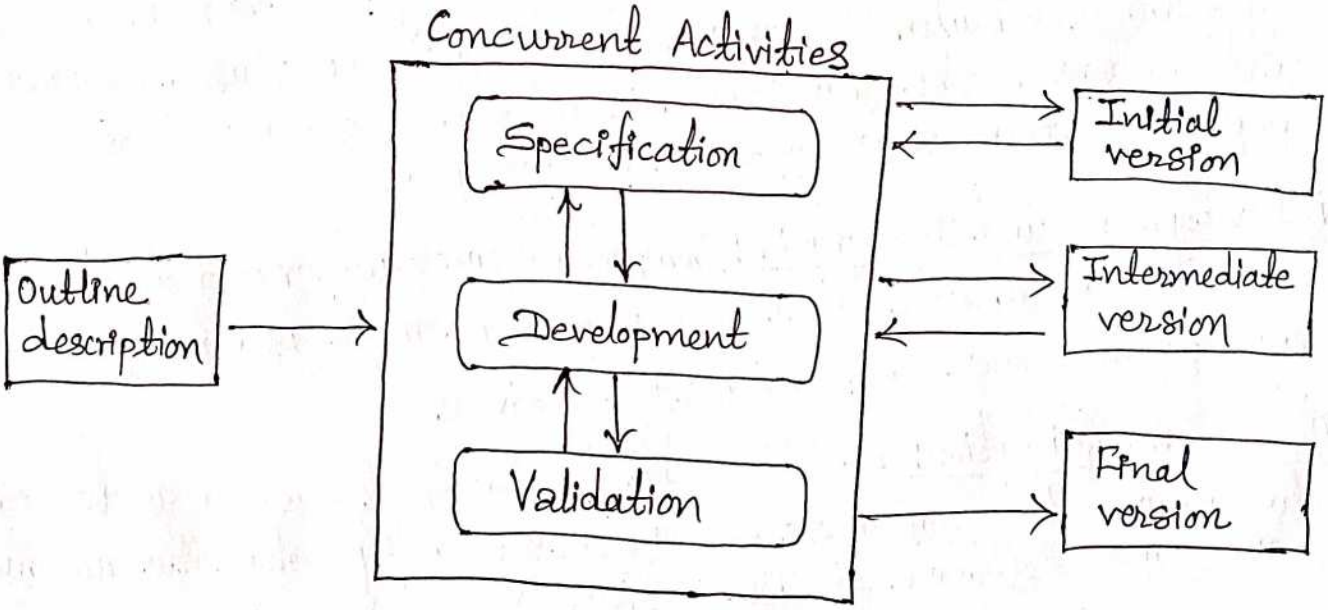


Fig: Incremental development model.

3) Integration and Configuration model:

OR Reuse-oriented development model.

In the majority of software projects, there is some software reuse. This often happens when people working on the project know of or search for code that is similar to what is required. They look for these, modify them as needed, and integrate them with the new code that they have developed.

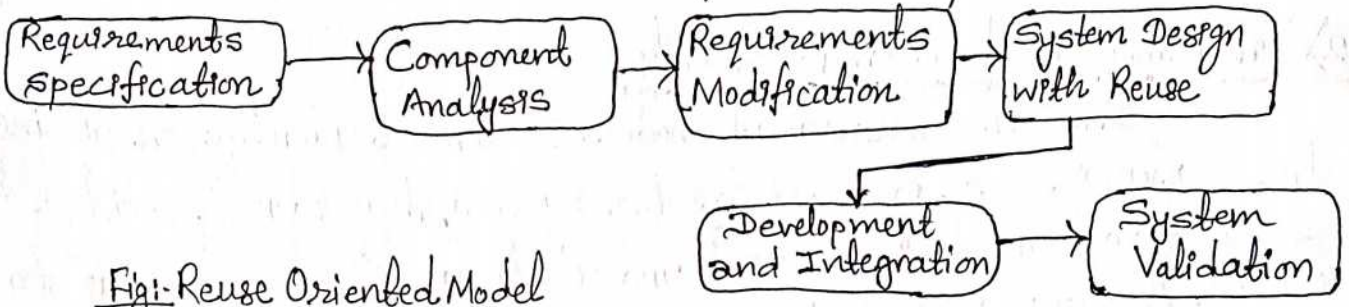


Fig:- Reuse Oriented Model

Stages:

i) **Requirement specification:** All possible requirements of the system to be developed are captured in this phase.

ii) **Component Analysis:** Based on requirement specification, a search is made for components that can implement the given specification. Usually there is no exact match, and the components that may be used only provide some of the functionality required.

iii) **Requirements Modification:** Requirements are modified according to available components, to reflect the services of available components.

iv) **System design with reuse:** During this stage the design of system is build. Designer must consider the reused component and organize the framework. If reused component is not available then new software is developed.

v) **Development and Integration:** Components are integrated to develop new software. Integration in this ~~part is~~ model is part of development rather than seperate activity.

vi) **System validation:** In this step, the developed system is tested to ensure that software system does exactly what the customer wants and software is defect free.

⊛. Coping with change:

Change is sure to happen in all large software projects. The system requirements change as business respond to external pressures, competition, and changed management priorities. Change adds to the costs of software development because it usually means that work that has been completed has to be redone. Following are the two ways of coping with change and changing system requirements:

1) Prototyping:

In prototyping instead of spending a lot of time producing very detailed specifications, the developers find out only outline of system. The complete system is not developed at once. Instead, a quick prototype is created which contains some portions of the system, and the prototype is refined and extended iteratively until the final specifications. Prototyping approach is used when requirements are not clear or well-understood.

A software prototype can be used in a software development process to help anticipate changes that may be required:

(expect or predict)

(hint or suggestion)

→ In the requirement engineering process, a prototype can help with the elicitation and validation of system requirements.

→ In the system design process, a prototype can be used to explore software solutions and in the development of a user interface for the system.

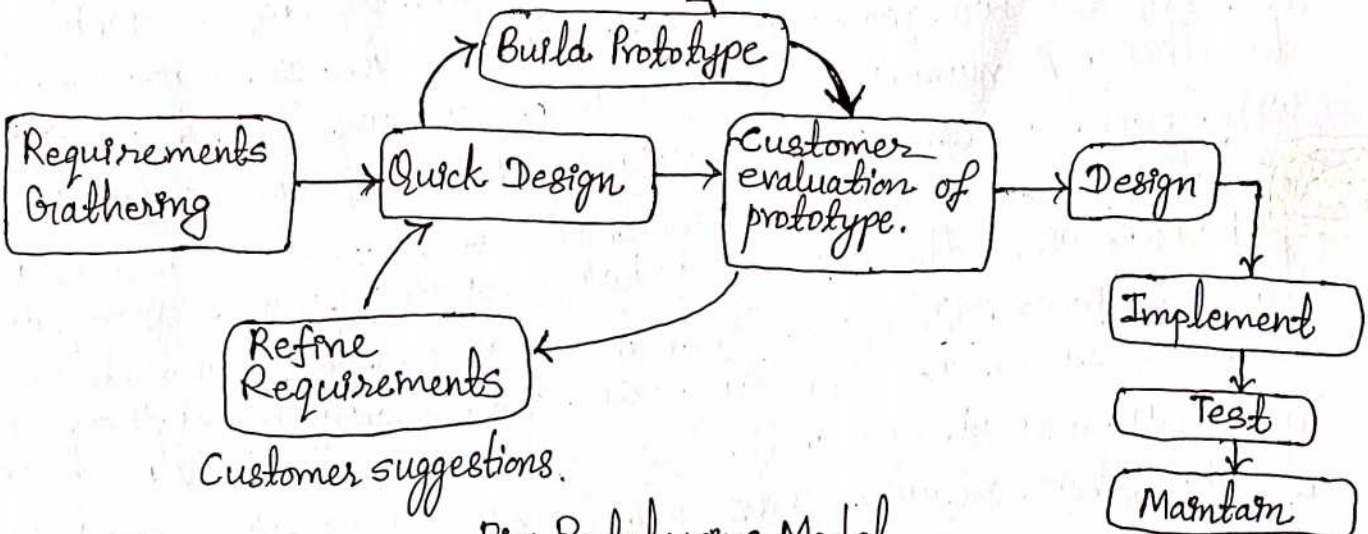


Fig: Prototyping Model

Q. Discuss evolutionary prototyping and throw-away prototyping in the software process.

Solution:

Evolutionary prototyping: In this method, the prototype developed initially is incrementally refined on the basis of customer feedback till it finally gets accepted. It helps us to save time as well as effort. That's because developing a prototype from scratch for every interaction of the process can sometimes be very frustrating. This model is helpful for a project which uses a new technology that is not well understood. It is also used for complex project where every functionality must be checked once. It is helpful when the requirement is not stable or not understood clearly at the initial stage.

Throw-away prototyping: Throwaway is based on the preliminary requirement. It is quickly developed to show how the requirement will look visually. The customers feedback helps drives changes to the requirement, and the prototype is again created until the requirement is baselined. In this method, a developed prototype will be discarded and will not be a part of the ultimately accepted prototype. This technique is useful for exploring ideas and getting instant feedback for customer requirements.

2) Incremental Delivery: (2nd way of coping with change 1st way is prototyping.)

Incremental delivery is an approach to software development where some of the developed increments are delivered to the customer and deployed for use in their working environment. In an incremental delivery process, customers define which of the services are most important and which are least important to them. A number of delivery increments are then defined, with each increment providing a subset of system functionality. The allocation of services to increments ~~are t~~ depends on service priority, with the highest priority services implemented and delivered first. Once an increment is completed and delivered, it is installed in the customers normal working environment. They can experiment with the system, and this helps them clarify their requirements for later system increments.

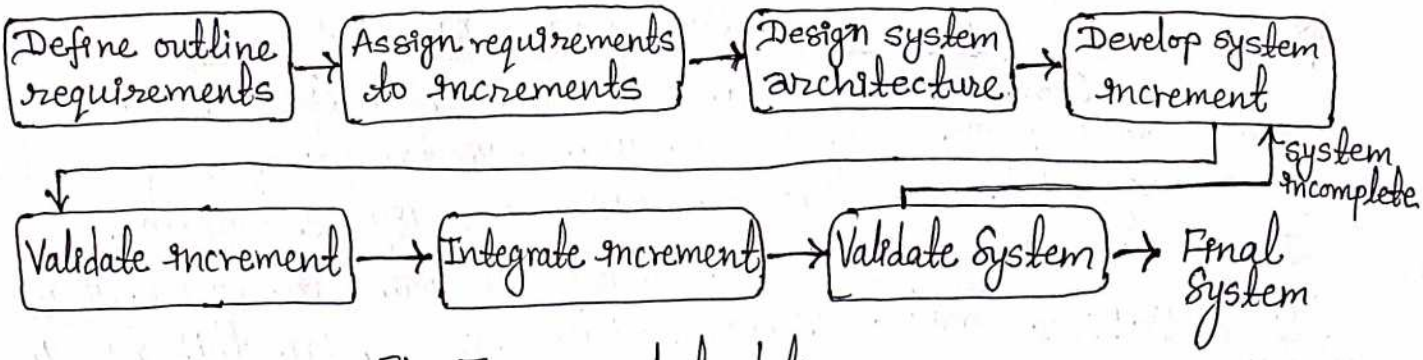


Fig: Incremental delivery

⊛. Process Improvement:

Process improvement means understanding existing processes and changing these processes to increase product quality and/or reduce cost and development time. Two quite different approaches to process improvement and change are used:

i) Process maturity approach: It has focused on improving process and project management and introducing good software engineering practice into an organization. The level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development process.

ii) Agile approach: It has focused on iterative development and the reduction of overheads in the software process. The primary characteristic of agile methods are rapid delivery of functionality and responsiveness to changing customer requirements.

⊛. Differentiate between V-shape model and spiral model:

V-Shape model	Spiral model.
i) In V-model testing activities start with the first stage.	i) In spiral model testing is done at the end of engineering phase.
ii) Guarantee of success through V-model is high.	ii) Guarantee of success through Spiral model is low.
iii) It is not iterative	iii) It is iterative.
iv) Cost of V-model is less expensive than spiral model.	iv) Cost of Spiral model is very expensive.
v) In this model development and testing are not concurrent.	v) In this model development and testing are concurrent.

⊛. Component Based Software Engineering (CBSE):

Component-based software engineering is a procedure that focuses on the design and development of computer based systems with the help of existing reusable software components. It is also known as reuse oriented model. Integration and Configuration model is reuse oriented model. In this rather than developing system from scratch, we search for existing reusable components and design the desired system in terms of those components.

Advantages:

→ Provides reusability of components.
→ Software development risk is reduced.
→ Software development process becomes easier and faster.
→ Reduces cost of development of system.
→ Easier to add functionalities to the system.

UNIT-3

Agile Software Development

Agile Development: It is a software development method based on iterative and incremental development in which requirement and solutions evolve through collaboration between self organizing, cross functional teams. It was mainly intended for helping developers build a project which can adopt to transforming requests quickly. So, it was developed to make easy and rapid project achievement. It is also known as rapid software development or agile methods.

Agile Principles:

1. **Customer Involvement:** In agile development, customers are closely involved in the development team throughout the development process, to evaluate newly developed increaments and provide the feedback.
2. **Incremental Delivery:** Software is developed in increments with the customer specifying the requirements to be included in each increment.
3. **People not Process:** the skills of development team should be recognized and team members should be left to develop in their own ways of working.
4. **Embrace change:** Agile development process expects the system requirement to change as much as possible.
5. **Maintain Simplicity:** In agile development development team actively works to eliminate complexity from the system whereever possible.

Advantages of Agile Method:

→ Customer satisfaction by rapid, continious delivery of useful software.
→ Face-to-face conversation is the best form of communication.
→ Regular adaptation to changing circumstances.
→ Even late changes in requirements are welcomed.

Disadvantages of Agile Method:

→ Due to lack of formal documents, it creates different confusions.
→ Due to absence of proper documentation, maintenance of the developed project can become a problem.
→ The project can easily get taken off track if the customer is not clear what final outcome they want.
→ Only senior programmers are capable of taking decisions required during agile development process.

When to use Agile Method:

→ When new changes are needed to be implemented. Agile method provides facility such that new changes can be implemented at very little cost because of the frequency of new increments that are produced.

→ To implement a new feature the developers need to lose only the work of a few days, or even only hours, to roll back and implement it.

→ When very limited planning is required to get started with the project.

→ When changes can be discussed and features can be newly affected or removed based on feedback.

⊛ Plan-Driven vs. Agile Development:

Plan-Driven Development	Agile Development
i) In plan-driven development all of the process activities are planned in advance and progress is measured against this plan.	i) In agile development, planning is incremental and it is easier to change the plan and software to reflect changing customer requirements.
ii) Iteration occurs within activities in plan-driven development.	ii) Iteration occurs across activities in agile development.
iii) We use this approach when we have a very detailed specification and design.	iii) We use this approach when very limited planning is required to get started with project.
iv) Sometimes it requires lower skill levels than agile based approach.	iv) Sometimes it requires higher skill levels than plan-driven approach.
v) It is more suitable for large systems.	v) It is more suitable for smaller systems.

⊛ Agile Development techniques: (less imp)

The most significant approach to changing software development culture was the development of Extreme Programming (XP).

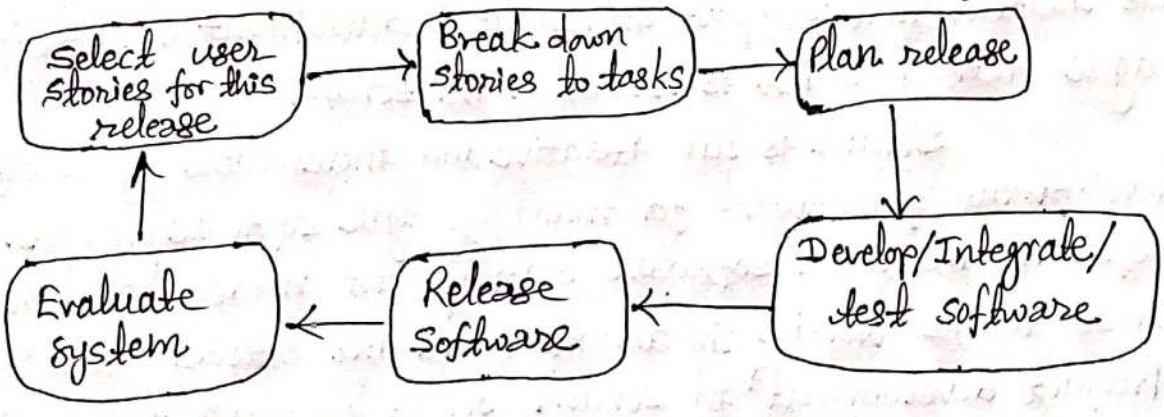


Fig: The XP release cycle.

Extreme Programming Practices:

Principle or practice	Description
Collective ownership	The pairs of developers work on all areas of the system. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated to the whole system.
Incremental planning	Requirements are recorded on "story cards".
On-site customer	The customer should be available full time for the use of the XP team.
Pair programming	Developers work in pairs, checking eachothers work and providing the support.
Simple design	Enough design is carried out to meet current requirements.

Introduction to Agile Project Management:

Software project manager have to manage the software project so that the software is delivered on time and with the estimated budget for software. To accomplish this, manager have to follow the plan based approach and should have a stable view of everything that has to be developed. But in agile methods requirement are developed incrementally, and delivered in short time interval. To manage agile project, different approach to project management is adapted that support incremental development and strength of agile method which is realized in scrum.

Scrum is an iterative and incremental agile software development framework for managing agile software projects. It refers to the agile software development model based on multiple small teams working in an intensive and interdependent manner. Product development in scrum occurs in small pieces, with each piece building up on previously created pieces.

UNIT-4

Requirements Engineering

⊛. User Requirement:

User requirements are the statements in natural language plus diagram of what services that the system is expected to provide and the constraints under which it must operate. These are often referred to as user needs, describe what the user does with the system. User requirements are generally documented in user requirement document using narrative text.

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 the information they provide may also be incomplete, inaccurate and self-conflicting. This is why user requirements are generally considered seperate from system requirements.

⊛ System Requirement:

System requirements are more detailed descriptions of the software system's functions, services and operational constraints. The system requirements document should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.

Readers of user requirement are Client Manager, System and user, Contractor manager, and System Architects. But readers of system requirement are System and user, Client engineer, Software developers, and System Architect.

⊛. Functional Requirement: Functional requirements are the statement of the services that the system must provide. Functional requirements define the basic system behaviour. They are what the

system does or must not do, and can be thought in terms of how the system responds to inputs. Functional requirements are features that allow the system to function as it was intended. If the functional requirements are not met, the system will not work.

Functional requirements are the main things that the user expects from the software. For e.g, if the application is banking application, that application should be able to create new account, update account, delete a account etc. The functional requirement for the system should be both complete and consistent. Completeness means that all services required by the user should be defined and consistency means that requirement should not have contradictory meaning.

Examples of Functional Requirements:

→ The software automatically validates customers against the Contact Management System.
→ The Sales system should allow users to record customers sales.
→ Only Managerial level employees have the right to view revenue data.
→ The software system should be integrated with banking API.

Advantages of functional Requirement:

→ It helps us to check whether the application is providing all the functionalities that were mentioned by user.
→ It helps us to define the functionality of a system or it's sub-system.
→ Errors caught in the functional requirement gathering stage are the cheapest to fix.
→ Support user goals, ~~but~~ tasks or activities for easy project management.

⊛ Non-Functional Requirements:

Non-functional requirement are requirements that are not directly concerned with specific functions delivered by the system but they are concerned with developing system properties, such as reliability, response time, security, safety etc. Non-functional requirements specify how the system do it. Non-functional requirements do not affect the basic functionality of the system. Even if the non-functional requirements are not met, the system will still perform its basic purpose.

Examples of Non-Functional Requirements:

→ Users must change the initially assigned login password immediately after the first successful login.
→ Employees never allowed updating their salary information. Such attempt should be reported to the security administrator.
→ A website should be capable enough to handle millions of users without affecting its performance.
→ The software should be portable. So moving from one OS to other OS does not create any problem.

Advantages of Non-Functional Requirement:

→ The non-functional requirements ensure the software system follow legal and compliance rules.
→ They ensure the reliability, availability, and performance of the software system.
→ They ensure good user experience and ease of operating the software.
→ They help in formulating security policy of the software system.

Types of Non-Functional Requirements:

1). **Product Requirement:** The requirements which specify the product behaviour are called product requirements. The product's requirements include all functions, features and behaviours that the product must possess, so that it will be efficient, ease to use, safe, low cost etc.

2). **Organization Requirement:** The requirements which are derived from politics and procedures in the customer's and developers organization are called organizational requirement. It includes the process standard used, implementation such as programming language, design methods etc.

3) **External Requirement:** The requirements that are derived from the factors external to the system and it's development process. It ensures that whether the system operates with law or not, whether it will be acceptable to it's users and general public or not.

⊛ Differences between Functional and Non-Functional Requirements:

Functional Requirements	Non-functional Requirements
i) A functional requirement defines a system or it's component.	i) A non-functional requirement defines the quality attribute of a software system.
ii) It specifies "What should the software system do?"	ii) It specifies "How should the software system fulfill the functional requirements?"
iii) It is specified by user.	iii) It is specified by technical peoples.
iv) It is mandatory	iv) It is not mandatory.
v) It helps us to verify the functionality of software.	v) It helps to verify the performance of the software.
vi) Usually easy to define	vi) Usually more difficult to define.

⊛ Requirement Engineering Process:

It is a process that involves all of the activities required to create and maintain a system requirement document. The purpose of requirements engineering is to make the problem that is being stated clear and complete, and to ensure that the solution is correct, reasonable, and effective. The process used for requirement engineering varies widely depending on the application domain, the people involved and the organization developing the requirement. However, following are common to all processes:

1) Feasibility Study: Information such as resource availability, cost estimation for software development, benefits of the software to the organization after it is developed and cost to be incurred on its maintenance are considered during the feasibility study.

10 marks में आएगा types लिख देना

Technical Feasibility: It includes the hardware and software required to complete user requirements in the software within allocated time and budget.

Economic Feasibility: Economic feasibility determines whether the required software is capable of generating financial gains for an organization.

Operational Feasibility: It involves the extent to which the required software performs a series of steps to solve user requirements.

2) Requirement Elicitation and Analysis:

Elicitation is gathering of all the system requirements from stakeholders. Requirement Elicitation is the practice of researching and discovering the requirements of a system from users, customers, and other stakeholders. Following are four main activities of requirements elicitation and analysis:

Requirements Discovery: Requirement discovery is the process of interacting with, and gathering the requirements from, the stakeholders about the required system and the existing system.

Requirement classification and Organization: It is the process of putting related requirements together, and decomposing the system into sub components of related requirements.

Requirement Prioritization and Negotiation: This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiations until we reach a situation where some of the stakeholders can compromise.

Requirement specification: In this step, requirements are checked to discover whether they are complete, consistent or not and they are documented and input in to the next round of spiral.

3) Requirement Specification: The management of the organization studies feasibility report and suggests the modifications in the requirement if any. Knowing the constraints on available resources and modified requirements specified by organization, final specification of the system to be developed is drawn up by the system analyst.

4) Requirement Validation: Requirement validation is the process of checking the requirements for validity, consistency, completeness, realism and verifiability. It is concerned with showing that the requirement actually define the system which the customer wants. Requirement validation process consists of different types of checks like validity check, completeness check, consistency check etc.

5) Requirement Change Management: Requirement management is the process of managing changes to a system requirement. It is the process of understanding and controlling changes to system requirement. It is essential because we need to decide if the benefits of implementing new requirements are justified by the costs of implementation.

Q. What are the different types of requirement elicitation techniques? Explain in brief.

Ans: Requirement elicitation techniques are:

1) Document analysis: Document analysis is one of the most important technique in understanding the current process. Documents like user manuals, software vendor manuals, process documents about the current system can provide the inputs for the new system requirements.

2) Observation: The elicitation technique helps in collecting requirements by observing users or stakeholders. This can provide information about the existing process, inputs and outputs.

3) Interview: An interview is a systematic approach to elicit information from a person or group of people. In this case, the business analyst acts as an interviewer.

4) Questionaries: Questionaries are useful when there is a lot of information to be gathered from a larger group of stakeholders. This enables the business team to gather requirements from stakeholders remotely.

5) Prototyping: Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product. It helps in giving better idea of requirements.

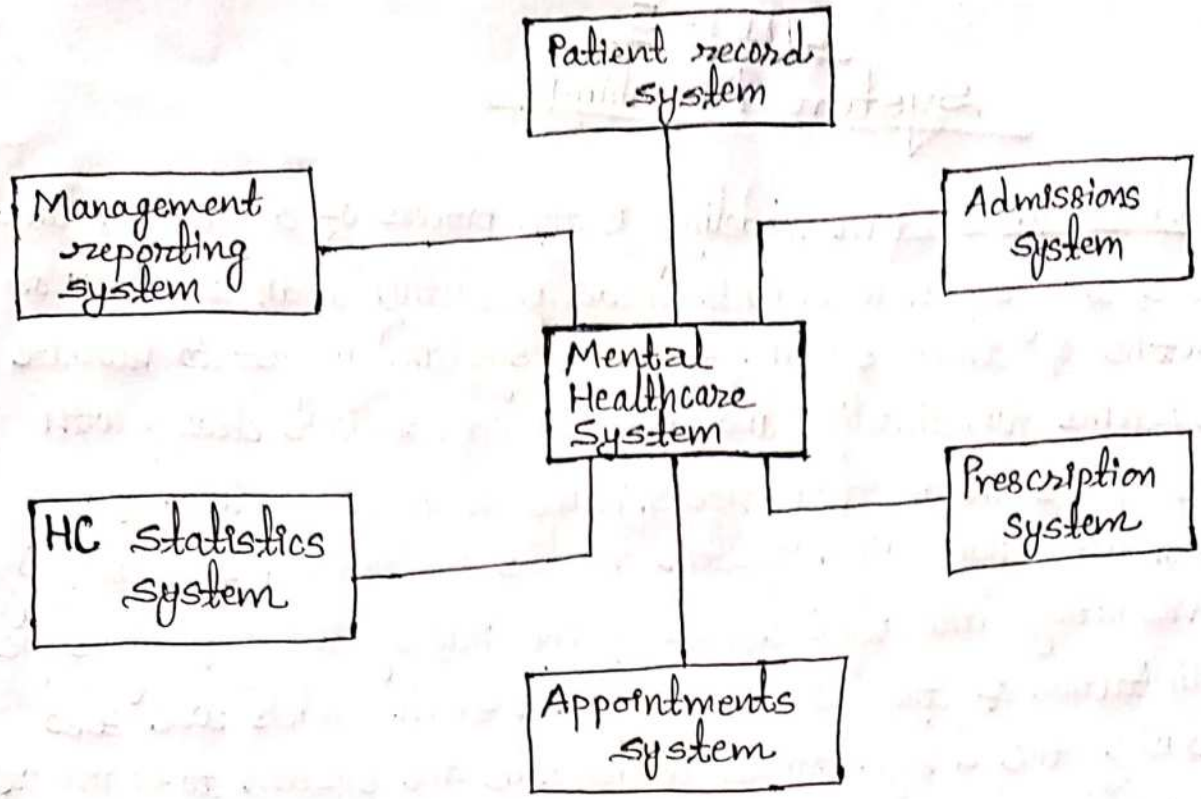
UNIT-5
System Modeling

⊛ Introduction: System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. It helps system analyst to validate the system's functionality and helps to communicate clearly with the customers about their needs. Hence, it is mainly used for requirement engineering. System models mainly help in identifying and validating the requirements of the new system by finding scope and limitation of the existing system. System models are also used during the design process to describe the system to engineers implementing the system. There are different types of models to represent the system from different perspectives: Contextual model, Interaction model, Structural model, Behavioral model.

⊛. Context Models:

Context models represents the operational environment of the system, they represent what lies outside the system boundaries. The environment of the system contains social and organizational concerns which directly or indirectly effects the position of system boundaries. Hence, architectural models are used to show the system and its relationship with other systems. Context models should be developed early in the process to reduce the system costs and the time needed for understanding the system requirements and design.

Figure below is the context model for mental healthcare system that shows the mental healthcare system and the other systems in its environment.



⊛ Interaction models:

Modeling user interaction is important as it helps to identify user requirements. Modeling system-to-system interaction highlights the communication problems that may arise. Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance. Use case diagrams and sequence diagrams are used for interaction modeling.

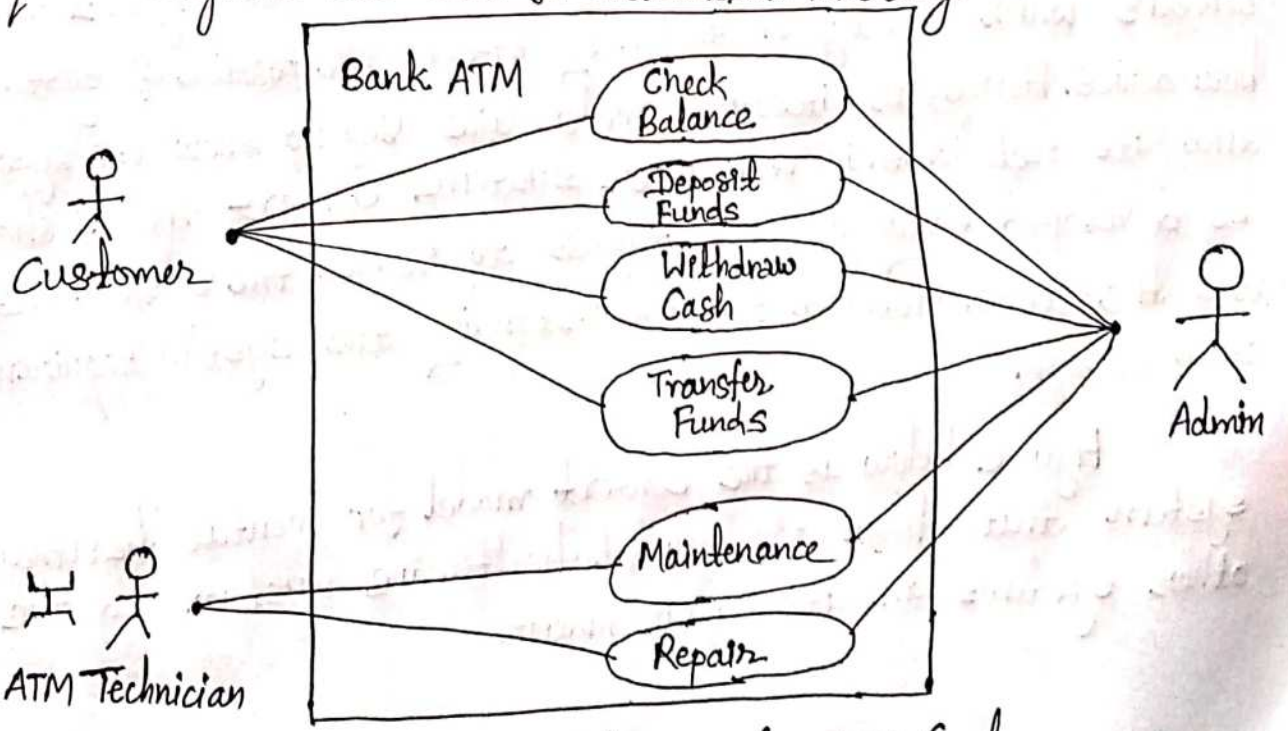


Fig: Use Case diagram for ATM System

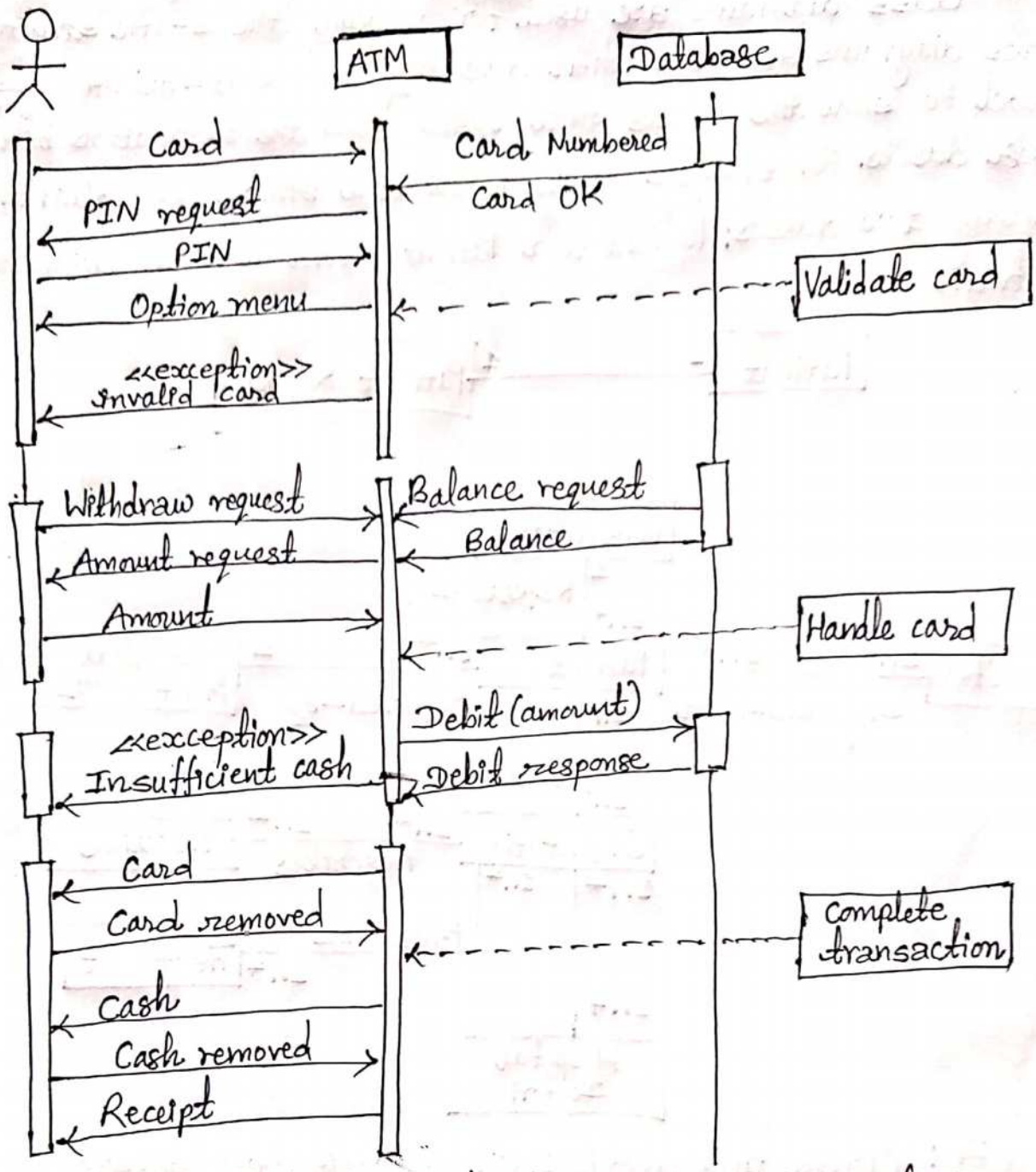
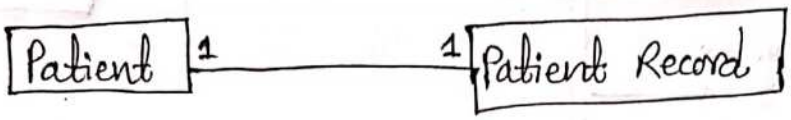


Fig: Sequence diagram of ATM withdrawl

⊛ Structural Models:

Structural models show the organization of a system in terms of the components that make up the system and their relationships. Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing. We create structural models of a system when we are discussing and designing the system architecture.

Class diagrams are used for modeling the static structure. Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes. For example, Figure below is a simple class diagram showing two classes: Patient and Patient Record with an association between them.



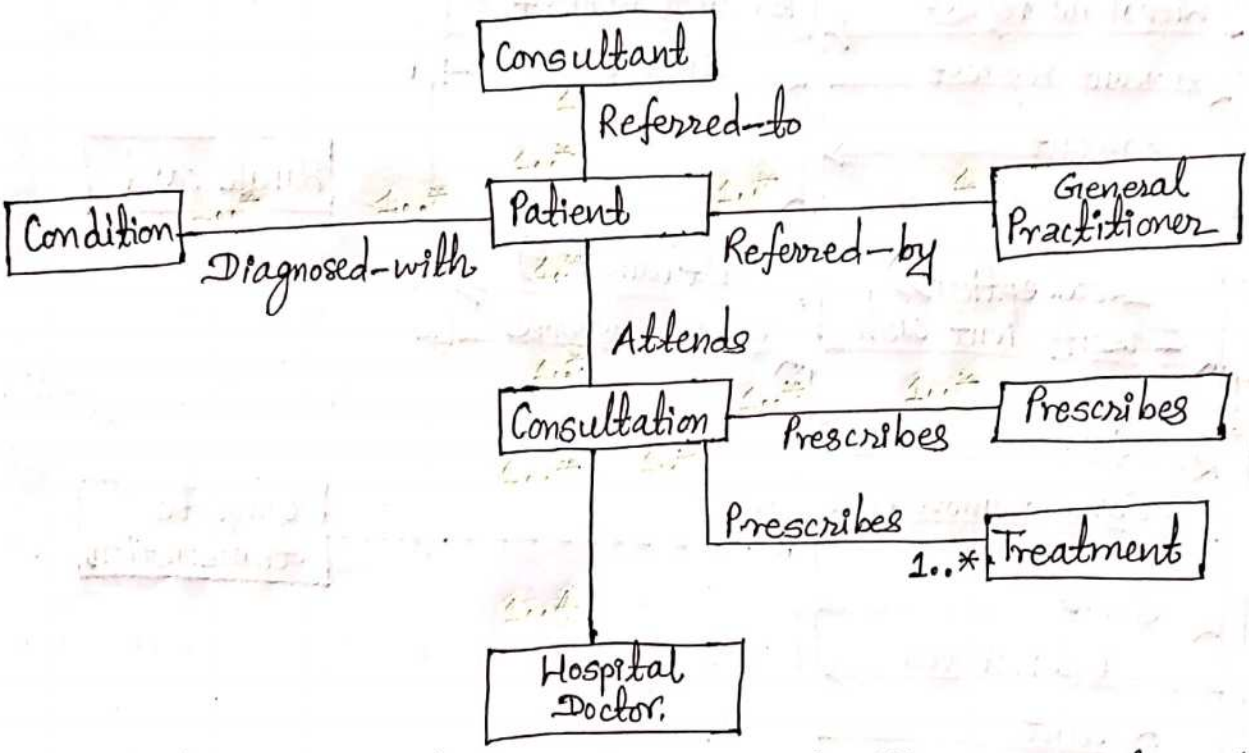
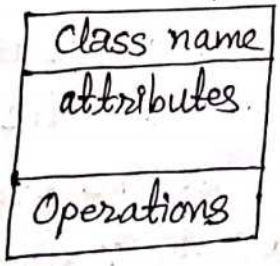


Fig: Classes and associations in a healthcare system.

To define the classes in more detail, we add information about their attributes and operations.



⊛Behavioural models:

Behavioural models are models of the dynamic behaviour of a system as it is executing. They show what happens or what is supposed to happen when a system responds to a stimulus from its environment. We can think of these stimuli as being of two types:

1) Data: Some data arrives that has to be processed by the system.
2) Events: Some event happens that triggers system processing.

Data-driven Modeling: Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing. Data-driven models show the sequence of actions involved in processing input data and generating an associated output.

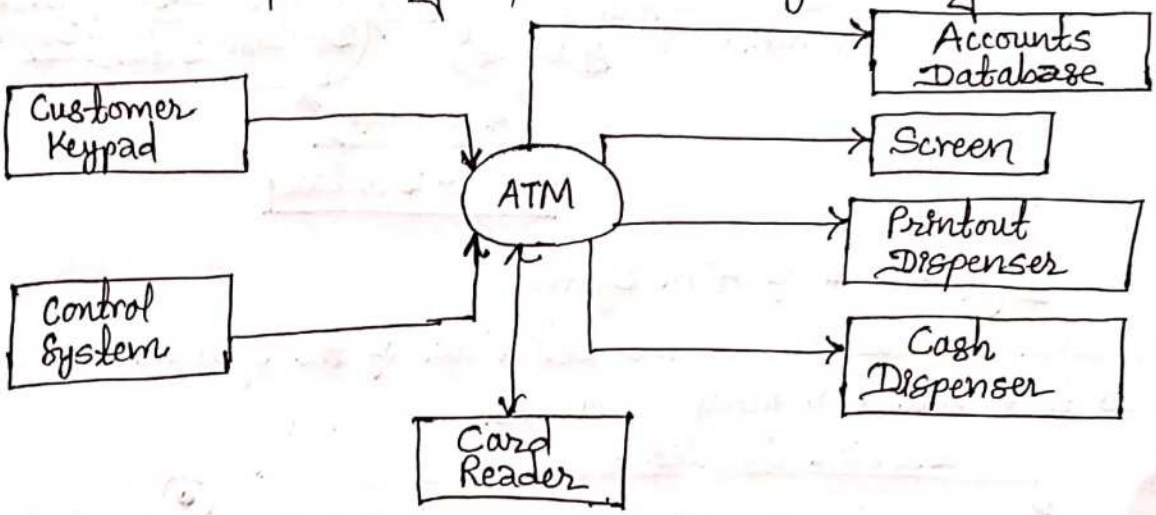


Fig: DFD 0 of ATM System

Event-driven Modeling: Real-time systems are often event-driven, with minimal data processing. For example, a landline phone switching system responds to events such as 'receiver off hook' by generating a dial tone. Event-driven modeling shows how a system responds to external and internal events. It is based on the assumption that a system has a finite number of states and that events may cause a transition from one state to another.

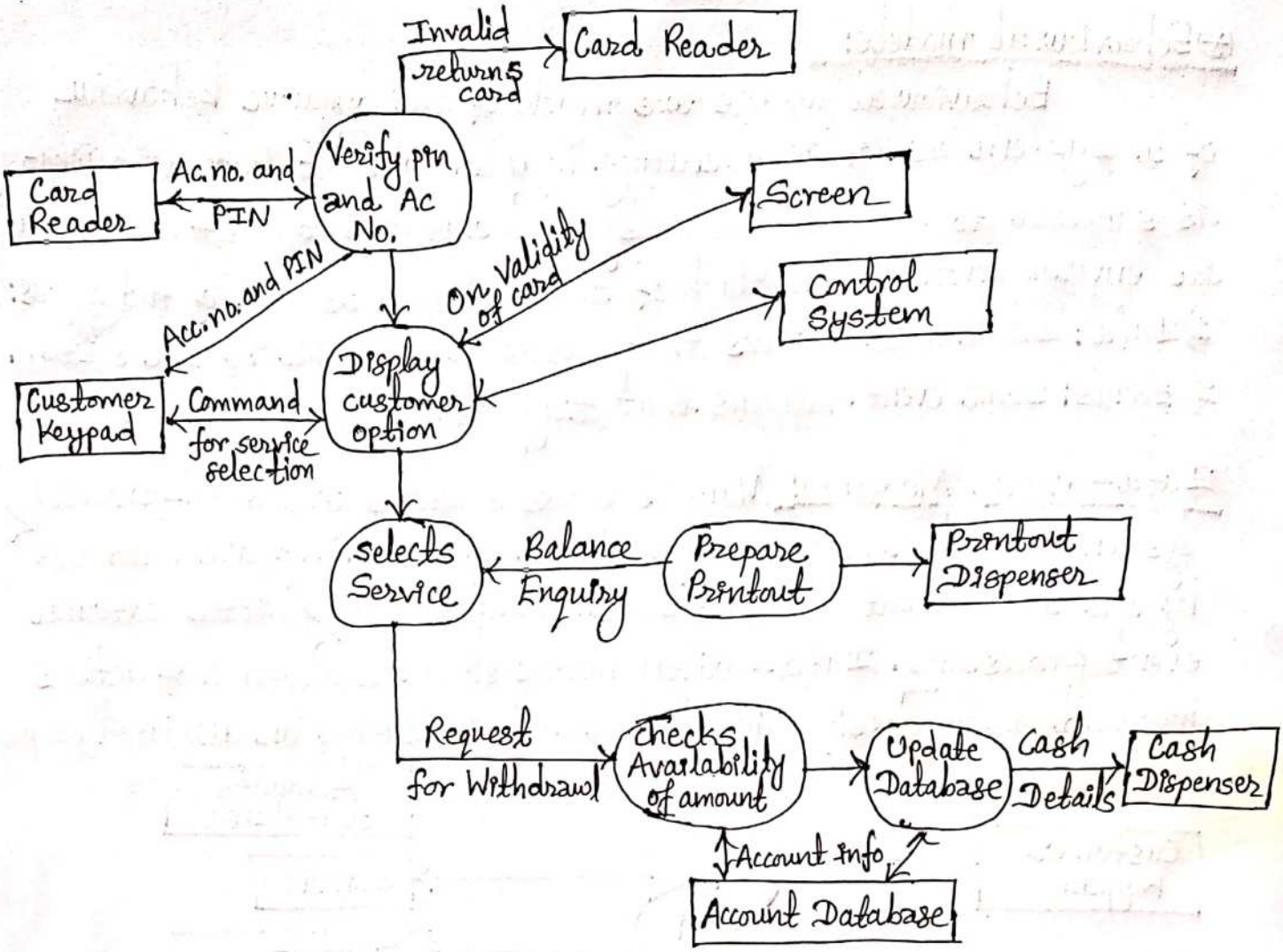


Fig: DFD 1 of ATM System

State machine models model the behaviour of the system in response to external and internal events.

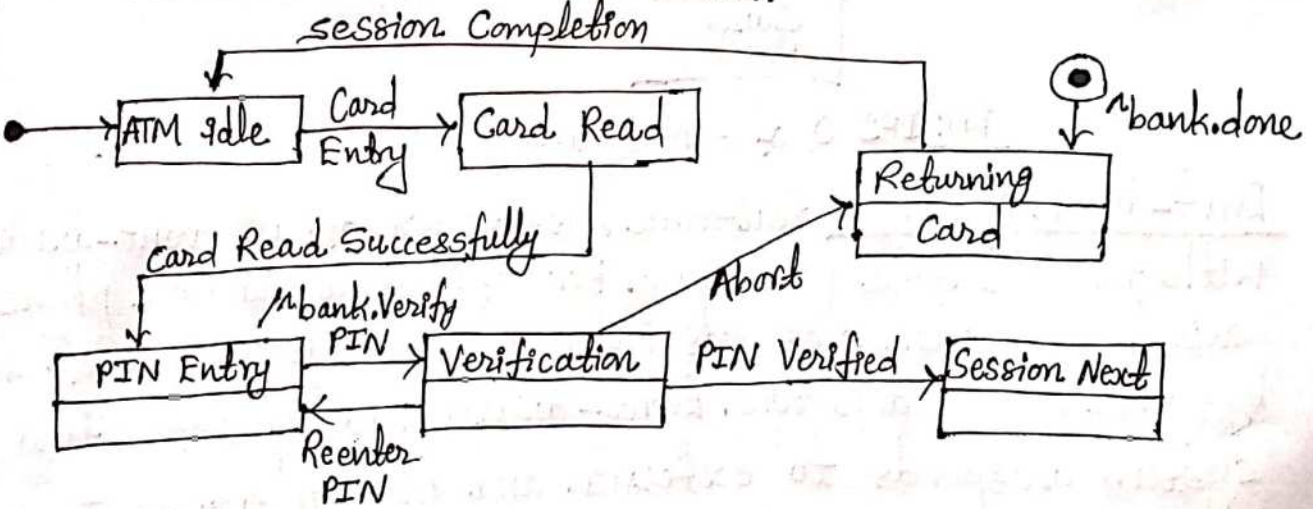


Fig: State diagram of ATM System.

⊛ Model-driven architecture:

Model-driven architecture is an approach to software design and implementation that focuses on developing models. It uses UML models to describe a system. Here, models at different levels of abstraction are created. The Model-driven architecture (MDA) recommends that three types of abstract system model should be produced:

1). A computation independent model (CIM): CIMs models the important domain abstractions used in a system and so are sometimes called domain models. We can develop several different CIMs reflecting different views of system.

2) A platform-independent model (PIM): PIMs model the operation of the system without reference to its implementation. A PIM is usually described using UML models that show the static system structure and how it responds to external and internal events.

3) Platform-specific models (PSM): PSMs are transformations of the platform-independent model with a seperate PSM for each application platform. In principle, there may be layers of PSM, with each layer adding some platform-specific detail.

Pros: Allows systems to be considered at higher levels of abstraction. Generating code automatically means that it is cheaper to adapt systems to new platforms.

Cons: Models for abstraction and not necessarily right for implementation. Savings from generating code may be outweighted by the costs of developing translators for new platforms.

Q. Define software. Discuss system modeling with suitable example.

Ans: Software is an organized collection of computer programs and associated documentation. A software system consists of a number of several programs, configuration files which are used to set of these programs and system documentations which describe the structure of system and user documentation which explains how to use software.

System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. It is about representing a system using some kind of graphical notation, such as Unified Modeling Language (UML).

Models can explain the system from different prespectives:

External: where we model the context or environment of system.

Interaction: where we model the interactions between a system and its environment, or between the components of system.

Structural: where we model the organization of a system or the structure of the data processed by the system.

Behavioural: where we model the dynamic behaviour of the system and how it responds to events.

Five types of UML diagrams that are the most useful for system modeling:

Activity diagrams: which show activities involved in a process or data processing.

Use case diagrams: which show the interactions between the system and its environment.

Sequence diagrams: which show interactions between actors and the system and between system components.

Class diagrams: which show the object classes in the system and associations between these classes.

State diagrams: which show how the system reacts to internal and external events.

UNIT-6

Architectural Design

Introduction: Architectural design is concerned with understanding how a software system should be organized and designing the overall structure of that system. It identifies the main structural components in a system and the relationships between them. Architectural design is the first stage in the software design process. It is the link between design and requirements engineering.

Architectural design decisions:

During the architectural design process, system architects have to make a number of structural decisions. Based on their knowledge and experience, they have to consider the fundamental questions as:

→ Is there a generic application architecture that can be used?
→ What approach will be used to structure the system?
→ What architectural styles are appropriate?
→ How will the system be distributed?
→ How should the architecture be documented?
→ How will the architectural design be evaluated?

The architectural design of the system affects the performance, dependability, maintainability etc. of the system. The particular architectural design and ~~chosen~~ structure chosen for an application depends on the non-functional system requirements: Maintainability, Performance, Availability, Security, and Safety.

⊛ Architectural Views:

A view is a representation of an entire system from the perspective of a related set of concerns. It describe the system from the viewpoint of different stakeholders such as end-users, developers, project managers and testers. It provides four essential views:

1) Physical view: It shows the system how hardware and software components are distributed across the processors in the system. This view is useful for systems engineers planning a system deployment.

2) Logical view: It shows the key abstractions in the system as objects or object classes. It will be possible to relate the system requirements to entities in this view.

3) Process view: It shows how the system at runtime is composed of interacting processes. This view is useful for making judgements about non-functional system characteristics such as performance and availability.

4) Development view: It shows how the software is decomposed for development. This view is useful for software managers and programmers.

⊛ Architectural Patterns:

Architectural Patterns are a ways of presenting, sharing, and reusing knowledge about software systems that has been adopted in a number of areas of software engineering. Architectural pattern is a stylized, abstract description of good practice, which has been tried and tested in different systems and environments. So, a architectural pattern should describe a system organization that has been successful in previous systems. It should include information on when it is appropriate and when not to use that pattern, and details on the patterns strengths and weaknesses.

There are many generic patterns that can be used in software development. Some examples of patterns that are widely used and that capture good architectural design principles are as follows:

1) Model-view-controller pattern:

Model-view-controller pattern in short is called as MVC pattern. MVC pattern separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other.

Model component: It manages the system data and associated operations on that data.

View component: It defines and manages how the data is presented to the user.

Controller component: It manages user interaction (e.g, key press, mouse click etc.) and passes these interactions to the View and the model.

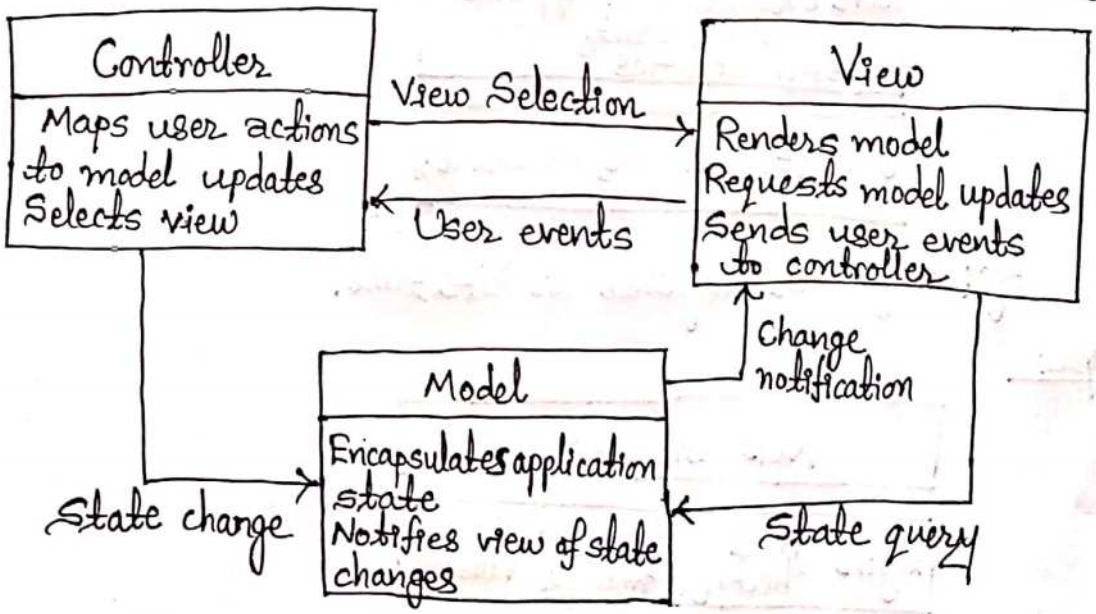


Fig: The organization of model view controller.

Advantages:

→ It allows data to change independently of its representation and vice-versa.

→ Support Presentation of the same data in different ways with changes made in one representation shown in all of them.

Disadvantages:

→ It can involve additional code and code complexity when the data model and interactions are simple.

2) Layered Architecture pattern:

In layerd architecture pattern, the system functionality is organized into separate layers, and each layer only relies on the facilities and services offered by the layer immediately below it. It organizes the system into layers, with related functionality associated with each layer. A layer provides services to the layer above it, so the lowest level layers represent core services that are likely to be used throughout the system.

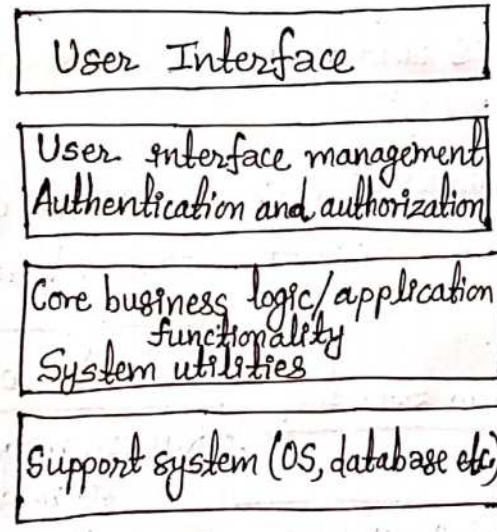


Fig: A generic layered architecture.

Example:

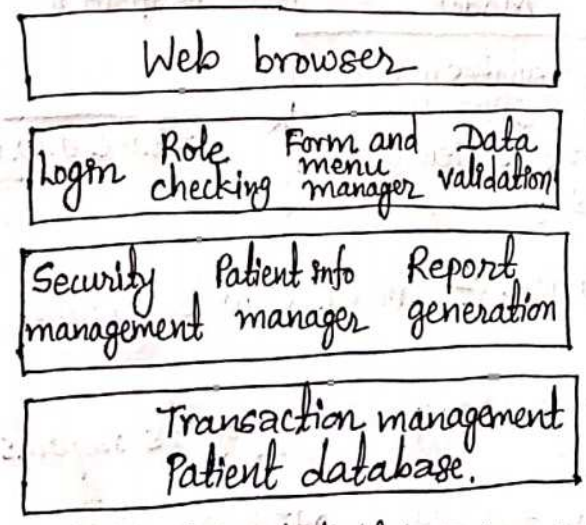


Fig: the layered architecture of a health care system.

3) Repository architecture pattern:

The repository pattern, describes how a set of interacting components can share data. All the data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.

This model is suited to applications in which data is generated by one component and used by another. Examples of this type of system include command and control systems, management information systems, CAD systems, and interactive development environments for software.

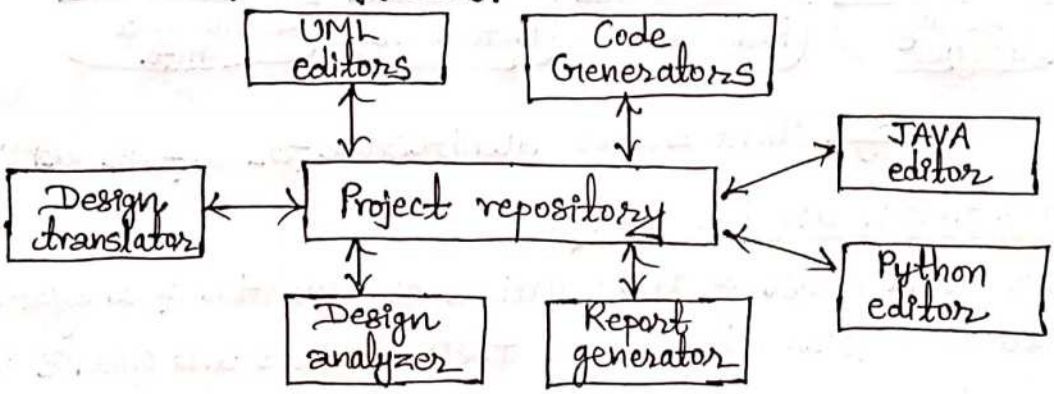


Fig: a repository architecture for IDE

Advantage: Components can be independent, they do not need to know of the existence of other components. Changes made by one component can be propagated to all components.

Disadvantage: The repository is a single point of failure so problems in the repository affect the whole system.

4) Client-server architecture pattern:

It is a system that follows the Client-server pattern is organized as a set of services and associated servers, and clients that access and use the services. The major components of this model are:

Clients: that call on the services offered by servers. There will normally be several instances of a client program executing concurrently on different computers.

Servers: that offer services to other components. Examples of servers include print servers that offer printing services, file servers that offer file management services etc.

Network: that allows clients to access these services. Client-server systems are usually implemented as distributed systems, connected using Internet protocols.

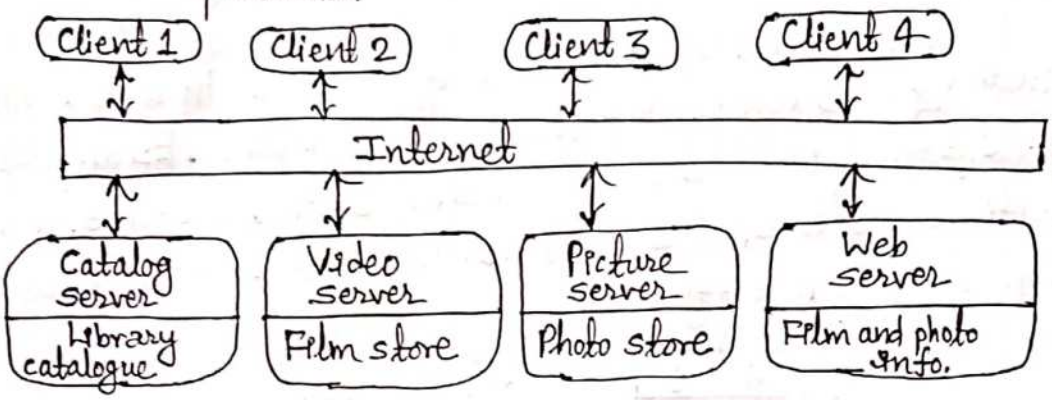


Fig: client server architecture for a film library.

5) Pipe and filter architecture:

This is a model of the runtime organization of a system where functional transformations process their inputs and produce outputs. The processing of the data in a system is organized so that each processing component is discrete and carries out one type of data transformation. The data flows from one component to another for processing.

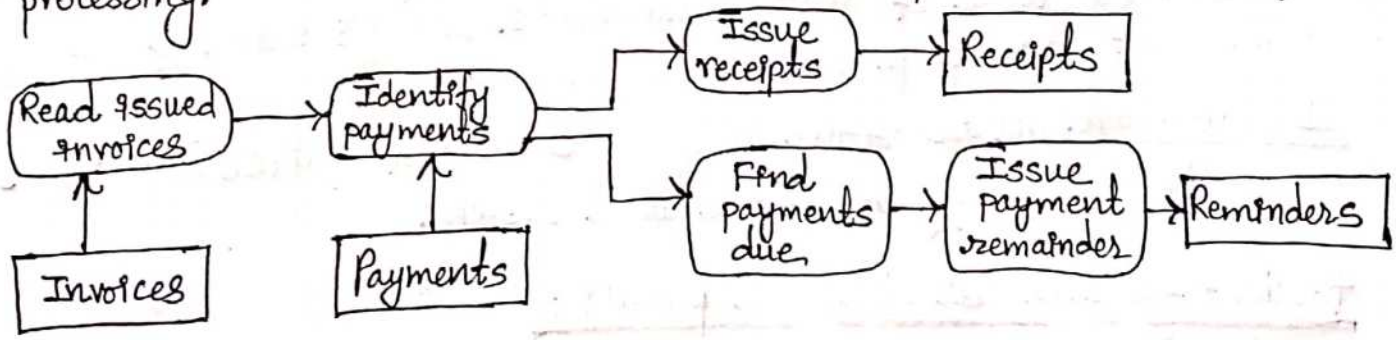


Fig: An example of the pipe and filter architecture.

⊛Application Architectures:

Application architecture describes the patterns and techniques used to design and build an application. Application Architecture is the process of defining the framework of an organization's application solutions against business requirements. It helps to ensure that applications are scalable and reliable. Application architecture encapsulates the principal characteristics of a class of system and designers can use of models of application architecture in number of ways:

→ While developing any software system if the designer is unfamiliar with the type of application that he/she is developing then initial design can be made using generic application architecture.

→ We can use application architecture as means of judging components for reuse.

→ Application architecture can be used to compare the applications of same type.

Q. Why modular decomposition is used in architectural design?

Ans: Modular decomposition is a process of decomposing subsystems into modules. After decomposition of the system into subsystems, subsystems must be decomposed into modules. Two modular decomposition models are used for:

Object model: where the system is decomposed into interacting objects.

Data-flow model: where the system is decomposed into functional modules which transform inputs to outputs. Also known as pipeline model.

⊛ What are the activities of architectural design process?

Ans: Architectural design process includes following activities:

System structuring: The system is decomposed into major sub-systems and communication mechanisms are identified.

Control modelling: A model of the control relationships between the sub-systems is established.

Modular decomposition: The identified sub-systems are decomposed into lower-level modules (components, objects etc.)

UNIT-7
Design and Implementation

Introduction: Software design is the stage at which the conceptual/logical model is converted to physical model. It is the process by which an agent creates a specification of a software outlook intended to accomplish goals, using a set of components.

Implementation is the process of realizing the design as a program. It is a systematically structured approach to transform the design model into a working product.

Object Oriented Design using the UML:

Object oriented design is a means of designing software so that the functional components in the design represent objects with their own private states and operations rather than functions. This process involves designing the object classes and the relationship between these classes. Software that are developed using object oriented design are easier to change than the systems developed using functional approaches. There are a variety of different object-oriented design processes that depend on the organization using the process. Common activities in these processes include:

→ Define the context and modes of use of the system;
→ Design the system architecture;
→ Identify the principal system objectives;
→ Develop design models;
→ Specify object interfaces.

System context and interactions:

Understanding the relationships between the software that is being designed and its external environment is essential for deciding how to provide the required system functionality and how to structure the system to communicate with its environment. Understanding of the context also lets us establish the boundaries of the system. Setting the system boundaries helps us to decide what features are implemented in the system being designed and what features are in other associated systems.

A system context model is a structural model that ~~depends~~ demonstrates the other systems in the environment of the system being developed. An interaction model is a dynamic model that shows how the system interacts with its environment as it is used.

Use case Model: A use case model is a graphical representation of the interactions among the elements of a system and its external entities called actors. It consists of actors use cases and their relationships. The diagram is used to model system/sub-system of an application.

Purposes of use case diagram:

- Used together with requirement of a system.
- Used to get an outside view of system.
- Identify external and internal factors influencing the system.
- Focuses on functional requirement.

Symbols

1) 

→ External entity that interacts with the system. It can be human or machine. e.g. paypal, e-sewa etc.

2) 
[Use case] or action
→ It is an action that the user performs within the system.

3) [straight line] → Used to show interaction between the actor and user.

4) [Boundry] → System boundry used to represent the system scope.

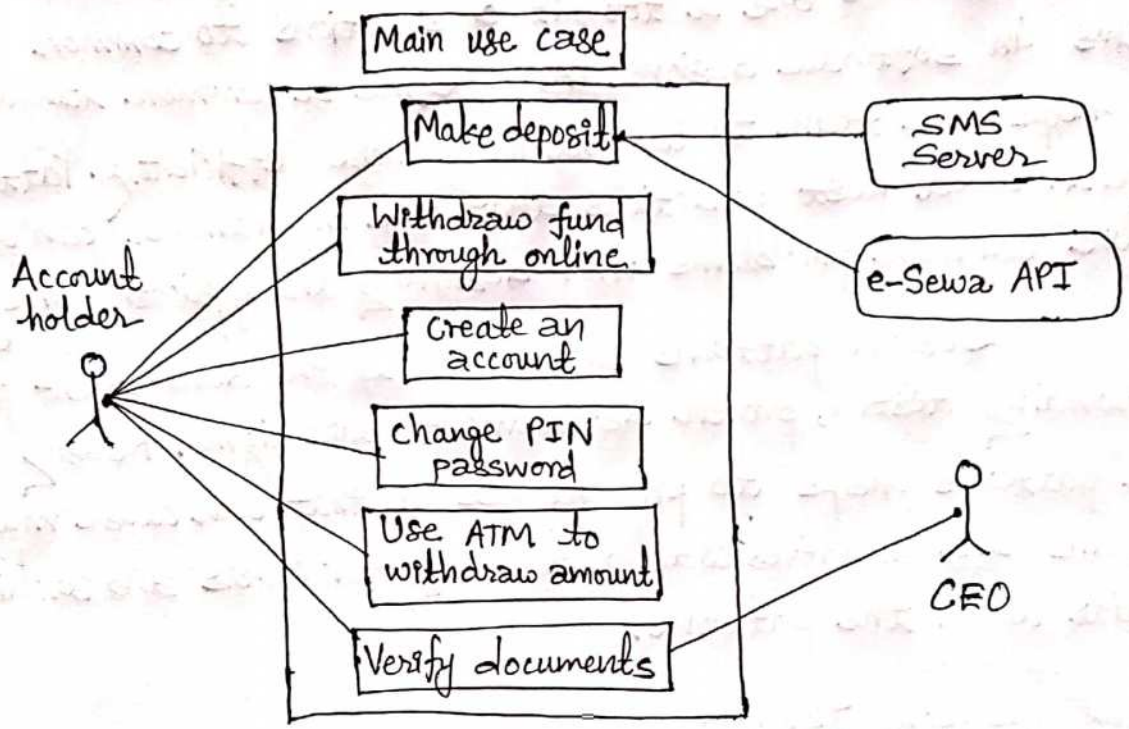


Fig: Bank ATM system use case diagram.

Architectural design: Architectural design is the process of decomposing the system into subsystems and establishing the relationship between subsystems. For this any of the architectural pattern is chosen from repository architecture, client server architecture or layered architecture.

Object class identification: Identifying object class is often a difficult part of object oriented design. There is no 'magic formula' for object identification. It relies on the skill, experience and domain knowledge of system designers. Object identification is an iterative process.

⊛ Design Patterns:

Design patterns are typical solutions to common problems in software design. Each pattern is like a blueprint that we can customize to solve a particular design problem in our code. Patterns are a toolkit of solutions to common problems in software designs. They define a common language that helps our team to communicate more efficiently. Patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.

Design patterns can speed up the development process by providing tested, proven development paradigms. Reusing design patterns helps to prevent issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.

⊛ Implementation Issues:

Some aspect of implementation issues are:

1) **Reuse:** Nowdays, modern software are constructed by reusing existing components or systems. Software reuse is possible at different levels.

Object level: At this level, we directly reuse objects from a library rather than writing the code ourselves.

Component level: Components are collection of objects and object classes that we reuse in application systems. We can reuse the component by adding some code of our own.

System level: At this level, we reuse entire application systems.

2) **Configuration Management:** Configuration management is the name given to the general process of managing a changing software system. During the development process, we have to keep track of

the many different versions of each software component in a configuration management system. The aim of configuration management is to support the system integration process so that all developers can access the project code and documents in a controlled way, find out what changes have been made, and link components to create a system.

3) Host-target Development: Generally, the software is developed in one computer and excuted on the same computer as the software development environment. Host target development is the methodology in which software is developed in one computer but runs on a seperate computer.

⑧. Open Source Development:

Open source development refers to something that is free and that anyone can inspect, modify, share etc. Open source software is usually a free software product, where developers have access to the source code. They can enhance the program's performance, add some features, and fix errors. Open source development is an approach to software development in which the source code of a software system is published and volunteers are invited to participate in the development process. Examples of open source product are Linux, apache web server, MYSQL, Java etc. A fundamental principle of open-source development is that source code should be freely available; this does not mean that anyone can do as they wish with that code. Developers involved should follow some terms and conditions and under some boundry conditions.

UNIT-8
Software Testing

⊛ Introduction:

Software testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use. When we test software, we execute a program using artificial data. We check the results of the test run for errors, anomalies or information about the programs non-functional attributes.

Program testing goals:

↳ To demonstrate to the developer and the customer that the software meets its requirements.

↳ To discover situations in which the behaviour of the software is incorrect.

⊛ Verification and Validation testing:

Verification specifies that "Are we building the system right". The software should conform to its specification. Validation specifies that "Are we building the right system". The software should do what the user really requires. Verification and validation is the name given to checking and analyzing processes that ensure that software confirms to its specification and meets the need of customer who are paying for that software.

V & V are independent procedure that is used together for checking that a product service or system meets requirement specifications and that it fulfills its intended purpose.

Differences between Verification and Validation:

Verification	Validation
i) It is the process of evaluating product of development phase to find out whether they meet the requirement and design specification.	i) It is the process of evaluating software at the end to determine whether it meets the customer expections and requirements, or works or not as it was intended.
ii) Activities involved are: review, meetings and inspections.	ii) Activities involved are; black box testing, white box testing.
iii) Verification is carried out by QA team.	iii) Validation is carried out by testing team.
iv) Execution of code does not come under verification.	iv) Execution of code comes under validation.
v) It is basically manual checking of documents like: requirement specification, design specification etc.	v) It is basically checking of developed software by executing source code.
vi) It specifies "Are we building the system right"	vi) It specifies "Are we building the right system"

⊛ Software Inspection:

Software inspection is a control technique for ensuring that the documentation produced during a given phase remains consistent with the documentation of the previous phases and respects reestablished rules and standards. These involve people examining the source representation with the aim of discovering anomalies and defects. The aim of inspection is to locate faults and process should be driven by fault check list.

Inspection Roles:

i) **Author:** An author is the person who created the work product being inspected, responsible for producing documents, fixing defects.

ii) **Moderator:** Leader of the inspection plans the inspection and coordinates it.

iii) **Chief Moderator:** Responsible for inspection process improvement, checklist updating etc.

iv) **Reader:** Reader is the person reading through the documents one item at a time presents the code or documents at inspection meeting.

v) **Inspector:** Person that examines the work product to identify possible defects, find errors etc.

Inspection Process:

i) **Planning:** The inspection is planned by moderator which involves selecting an inspection team, organizing a meeting room and materials.

ii) **Overview:** In this step, the software and documents to be inspected are presented to the inspection team.

iii) **Individual Prepration:** In this step, each inspection team member studies the specification and the program and looks for defects.

iv) **Inspection Meeting:** During this meeting, the readers read through the work product part by part and inspectors point out defects for every part.

v) **Rework:** The author makes changes to work product according to the action plan from the inspection meeting in this step.

vi) **Follow up:** In this step, the changes made by the author are checked to make sure that everything correct. The moderator should decide whether reinspection of the code and document is required. If not the software is then approved by moderator for release.

⊛ Software Testing Process:

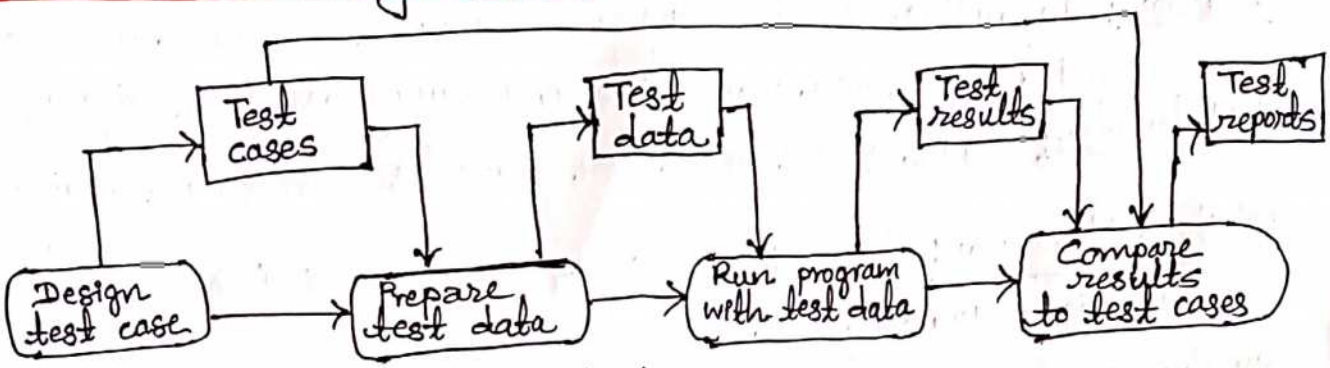


Fig: Software testing process.

Software testing process starts with prepration of test cases. Test cases are specification of the inputs to the test and the expected output from the system together with what is being tested. From the design of test cases we get test case list. After that we prepare the test data. Test data are the inputs which have been designed to test the system. Test data can be generated automatically or manually. From this step we get test data set. Then we run the program with test data for all test cases and generate the result. Finally we compare the test result to test cases and generate the test report.

Types of software testing / Software Testing Levels:

i) **Unit Testing:** It focuses on the smallest unit of software design. In this, we test an individual unit or group of interrelated units. It is often done by the programmer by using sample input and observing its corresponding outputs.

ii) **Integration Testing:** The objective is to take unit tested components and build a program structure, then the components are integrated to produce output.

iii) **System Testing:** A complete and integrated software is tested to verify that it meets specified requirements.

iv) Acceptance testing: Acceptance testing is done to verify if system meets the customer specified requirements. It has two types:

Alpha testing: It is a testing performed to identify bugs before releasing product to real users. It is typically done by QA people.

Beta testing: In this testing version is released for a limited number of users for testing in a real-time environment.

v) Regression Testing: Every time a new module is added leads to changes in the program. This testing makes sure that the whole component works properly even after adding components to system.

Testing Methods: Mainly there are two testing methods black-box testing and white-box testing.

Black box testing	White box testing
i) The internal workings of an application are not required to be known.	i) Tester has full knowledge of the internal working of the application.
ii) Performed by end users, testers and developers.	ii) Normally done by testers and developers.
iii) This is least time consuming and exhaustive.	iii) This is most exhaustive and time consuming.
iv) Not suited to algorithm testing.	iv) Suited for algorithm testing.
v) It is performed by testing team.	v) It is performed by developers themselves.
vi) Basis of test case is requirement specification.	vi) Basis of test case is detail design of the system.

⊛ Development Testing:

It is a method of applying testing practices consistently throughout the software development life cycle process. The testing ensures the detection of bugs or errors at the right time which further ensures delay of any kind of risk in terms of time and cost. It aims to establish a framework to verify whether the requirements of a given project are met in accordance with the rules of the mission to be accomplished. This testing is performed by the software developers or other engineers during the construction phase of the software development life cycle.

⊛. Test-driven development:

Test-driven development (TDD) is an approach to program development in which we inter-leave testing and code development. Tests are written before code and 'passing' the tests is the critical driver of development. We develop code incrementally, along with a test for that increment. We don't move on to the next increment until the code that we have developed passes its test.

⊛. Release testing:

Release testing refers to coding practices and test strategies that give teams confidence that a software release candidate is ready for users. Release testing aims to find and eliminate errors and bugs from a software releases so that it can be released to users. The primary goal of the release testing process is to convince the customer of the system that it is good enough for use.

(xx). User testing:

User testing is a stage in the testing process in which users or customers provide input and advice on system testing. User testing is essential even when comprehensive system and release testing have been carried out. The reason for this is that influences from user's working environment have a major effect on the reliability, performance, usability and robustness of a system. These cannot be replicated in a testing environment.

Q. What is software quality assurance? Explain with example. [Model set]

Ans: Software Quality Assurance (SQA) is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedures as well as standards for the project and implemented correctly. It is a process which works parallel to development of software. It focuses on improving the process of development of software so that problems can be prevented before they become a major issue.

Example: Library Management System:

To build this software system, first of all we have to create an software quality assurance (SQA) management plan that includes how SQA will be carried out in our project. Then, we setup the checkpoints or making schedule by dividing work so that particular work can be finished on time. Then we apply software engineering techniques (specification development testing) and then executing it for formal review to assure its quality. In this way we can assure SQA in our project.

UNIT-9

Software Evolution

⊛ Software Evolution Process:

Software development process does not end when system is delivered but continue through the lifetime of system. After a system has been deployed it needs to be changed time to time due to factors like: requirement changes, Environment changes, Errors or security breaches, improvements to system etc. This time to time process of developing, maintaining, and updating software for various reasons is referred to as software evolution.

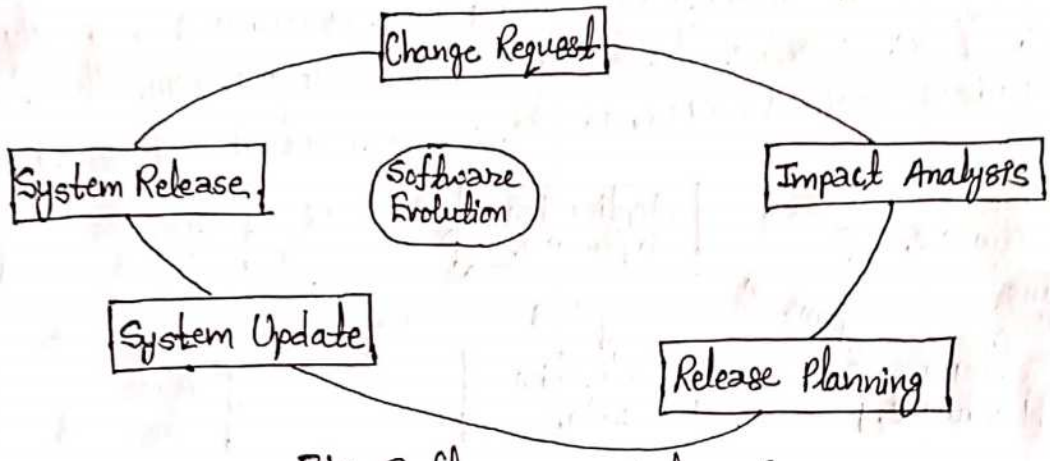


Fig: Software Evolution Process

Software evolution processes vary depending on type of software being maintained, the development process used in an organization, and the skills of people involved. However evolution process includes the fundamental activities of change request, impact analysis, release planning, system update, and system release.

Evolution process starts from the change request process. After which the impact of these changes are analyzed to see how much of the system is affected by the change and much it might cost to implement the change. If the proposed changes are accepted a new release of the system is planned. Then the changes are implemented to the next version of system and the system is released to customers.

⊛. Legacy Systems:

A legacy system is a computer system, programming language, software application, process or other technology that is outdated or that can no longer receive support and maintenance but is essential for organizations or companies and cannot be replaced or updated for different reasons.

Legacy systems are socio technical computer based systems that have been developed in past using older technology. They include application software together with business process, support software, system hardware etc.

For example: In a bank, banking management system was one of their earliest systems. Organization policies and procedures may rely on this system. If we replace the banking management system there could be a serious business risk, if the replacement did not work properly.

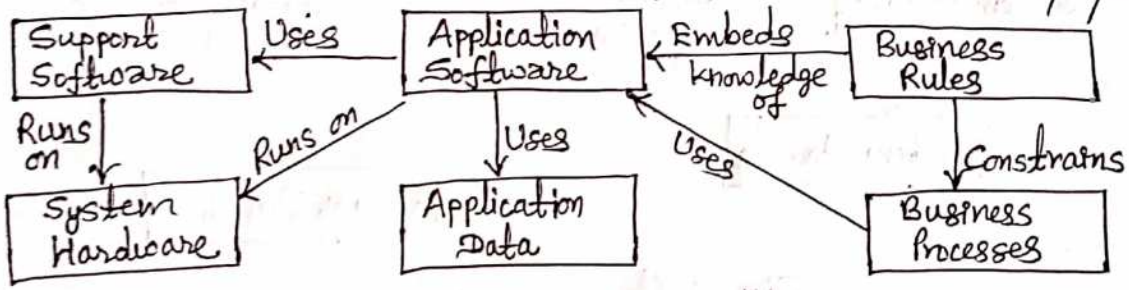


Fig: Legacy System Components

Strategies for evolving legacy systems:

i) Scrap the system completely: When the system is not making an effective contribution to business processes then the system will be scrapped.

ii) Reengineer the system to improve it's maintainability: This method is valid when the system quality has been degraded by change and where a new change to the system is still being proposed.

iii) Leave the system uncharged and continues with regular maintenance: This option is choosen when the system is still required but is fairly stable and the system users make relatively few change requests.

iv) Replace all or Part of the system with a new system:

This option should be chosen when factors, such as new hardware, mean that the old system cannot continue in operation or where off-the shelf systems would allow the new system to be developed at a reasonable cost.

⑧. Software Maintenance:

Software maintenance is an activity which includes optimization, error correction, and deletion of discarded features and enhancement of existing features. Maintenance does not normally involve major changes to the system's architecture. Changes are implemented by modifying existing components and adding new components to the system. There are four major activities that occur during maintenance:

i) Obtaining maintenance requests: In this step a formal process is established where users can submit system change requests.

ii) Transforming Request into Changes: Once a request is received, analysis must be performed to identify scope of the request. It must be determined how the request will affect the current system and how long such a project will take.

iii) Designing changes: A change request can be transformed into a formal design change, which can then be fed into maintenance phase.

iv) Implementing changes: Once the change design is approved, proposed changes are implemented in respective components of the system.

Types of Maintenance:

i) Corrective maintenance: Corrective maintenance deals with the repair of faults or defects found in day-day system functions. It refers to changes made to repair defects in the design, coding, or implementation of the system.

ii) Adaptive maintenance: Adaptive maintenance is the implementation of changes in a part of system, which has been affected by a change that occured in some other part of the system.

iii) Perfective maintenance: Perfective maintenance involves making functional enhancements to the system to increase systems performance.

iv) Preventive maintenance: Preventive maintenance involves performing activities to prevent the occurance of errors.

⊛. The Cost of Maintenance:

The cost of maintenance represent a large proportion of the budget of most organization that use the software system. For some organizations, as much as 60 to 80 percent of their information systems budget is allocated to maintenance activities. These huge maintenance costs are due to the fact that many organizations have accumulated more and more older legacy systems that require more and more maintenance.

Factors influencing Maintenance Cost:

i) Latent defects: This is the number of unknown errors existing in the system after it is installed.

ii) No. of customers: Greater the number of customers in the system, greater the maintenance costs.

iii) Quality of system documentation: Higher the quality of system documentation, lower the maintenance costs.

iv) Tools: Tools that can automatically produce system documentation can also lower maintenance costs.

v) Well-structured programs: Well-designed system is easier to understand and fix. So, more well-structured programs in system leads to lower maintenance costs.

UNIT-10

Software Management

⊛Software Project Management:

Software project management is an art and science of planning and leading software projects. It is a sub-discipline of project management in which software projects are planned, implemented, monitored and controlled. It is an essential part of software engineering. Good project management cannot guarantee project success however bad management usually results in project failure.

Software Project Manager:

A software project manager is a person who undertakes the responsibility of planning, executing and controlling the software project. Project managers are responsible for planning and scheduling project development. Software project management is differing from other type of project management due to following reasons:

i) The product is intangible in nature and it can be realized only it cannot be seen or touched.

ii) There are no standard software processes

iii) Large software projects are often One-off projects.

⊛. Project Management Activities:

1) Project Planning: Project planning is an organized and integrated management process, which focuses on activities required for successful completion of the project. Project planning involves breaking down the work into parts, assign these to project team members, allocate resources, and prepare solutions to problems.

Project planning sections:

i) **Introduction:** It describes objectives of the projects and sets out the constraints like time, budget etc.

ii) **Project organization:** It describes the people involved and their role in team.

iii) **Risk analysis:** It describes possible project risks.

iv) **Hardware and software resource requirement:** It specifies the hardware and software required to carry out the development.

v) **Work breakdown:** The project is breakdown into activities and identifies milestones and deliverables associated with each activity.

vi) **Project schedule:** It shows the dependencies between activities.

vii) **Monitoring and reporting mechanism:** This defines the management report that should be produced.

Types of project plan:

i) **Quality plan:** It describes the quality procedures and standards that will be used in a project.

ii) **Validation plan:** It describes the approach, resources and schedule used for system validation.

iii) **Configuration management plan:** It describes the configuration management procedures and structures to be used.

iv) **Maintenance plan:** It predicts the maintenance requirements of the system, maintenance costs and effort required.

v) **Staff development plan:** It describes how the skill and experience of the project team members will be developed.

2) Project Scheduling:

Project scheduling is the process of separating the total work involved in a project into separate activities and judging the time required to complete these activities. It is a detailed plan showing dates when each activity should start and finish, also it includes when and how much resource will be required. Creating a project schedule includes four stages:

- → Constructing an ideal activity plan.
- → Risk analysis
- → Resource allocation
- → Schedule production.

Project schedules are usually represented as a set of charts showing the work breakdown, activities dependencies and staff allocations. Bar charts (for example Gantt chart) and activity networks are graphical notations that are used to illustrate the project schedule. Bar charts show who is responsible for each activity and when the activity is scheduled to begin and end. Activity networks show the dependencies between the different activities.

3) Risk Management:

Risk management is process of identifying risks that might affect the project schedule or quality of project being developed and taking action to avoid these risks. Risk management involves identifying and assessing major project risks to establish the probability that they will occur and the consequences for the project if that risk does arise.

There are mainly three categories of risks which are likely to occur in software projects:

1) Project Risk: Project risks influences the schedule of software project and that if occur, delay the development process.
Example: staff turnover, hardware unavailability etc.

ii) Technical risks: Technical risks involves implementation, testing, and maintenance issue. Most technical risks appear due to the development teams insufficient knowledge about the project.

iii) Business risk: It affects the organization business that is developing the software. Business risks could be quite dangerous for the long-term sustainability of the business. Example: Technology change, Product completion etc.

Risk Management Process:

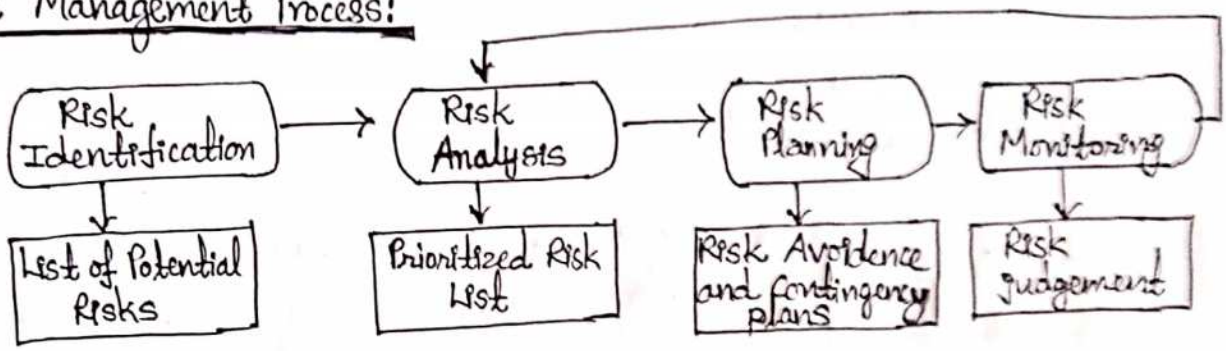


Fig: Risk management process.

The risk management process is an iterative process which continues throughout the project. Once an initial set of plans are drawn up, the situation is monitored.

i) Risk identification: It is the first stage of risk management. It is concerned with discovering possible risk to the project. It also involves prepration of risk list.

ii) Risk Analysis: In this phase all identified risks are considered and made a judgement about problems causing risk in the project. Identify probability of occurrence of risk and the impact of the risk. The probability of risks may be judged as very low (< 10%), low (10-25%), moderate (26-50%), high (51-75%), and very high (>75%). Risks are tabulated in order according to the seriousness of the risk.

iii) Risk planning: It is the process of consideration of the key risks that have been identified and formulation of strategies to manage the risk. The strategies to manage risk are:

→ Avoidance Strategies (that tells reduction of arising risks).
→ Minimization Strategies (that tells impact of risk will be reduced).
→ Contingency Plans (Prepared for worst and have strategy to deal with it).

iv) Risk monitoring: Risk monitoring means regularly judging each of the identified risk to decide whether or not the risk is becoming more or less portable and whether the effect of risk have changed.

4) People Management:

The process of planning, organizing, directing, and controlling human resources is called people management. It is also called human resource management. People Management ensures that organizational, individual, and societal needs are satisfied. Human resource management includes all activities used to attract and retain employees.

5) Proposal Writing: Project manager have to write the project proposal to win a contract from the customer. Proposal writing is a skill that acquire through practice and experience. Proposal should include the objective of the project and how it will be carried out. It also includes cost and schedule estimates.

⊛. Milestones and Deliverable:

Project milestones are the predictable outcome of an activity where some formal report of progress are addressed. Milestones are recognizable end points of software process activity, they may be short report of what has been completed and presented to the management. Milestones should represent the end of a distinct, logical stage in the project.

A deliverable is a project report (result) that is delivered to the customer at the end of major project phase. Deliverables are usually milestones but milestones need not to be deliverables. A deliverable could be a report, a document, a server upgrade of an overall project. A deliverable may be composed of multiple smaller deliverables.

⊛ Software Pricing:

Software pricing must take into account broader organizational, economic, political and business consideration. The major factors that affect the software pricing are as follows:

→ Development organization quotes a low price if want to move into a new segment of software market. Accepting low profit in one project may give the opportunity of more profit later.

→ If an organization has no idea about cost estimation then it may increase its price by some contingency

→ If the customer is not clear about their requirement then there is chance of changing requirement at that time organization may lower its price.

→ If the financial health of the organization is not good then they may lower their price to gain contract and establish themselves in business.

⊛. Estimation Techniques:

The software estimation process includes estimating the size of software product, effort required, and overall cost of the project. The approaches to cost estimation can be tackled using either top-down or a bottom up approach.

1) **Top-down Approach:** Top-down estimating method is also called Marco Model. Using this method, an overall cost estimation for the project is derived from the global properties of the software project,

and then the project is partitioned into various low-level components. This method is more applicable to early cost estimation when only global properties are known. This approach starts at system level.

Advantages:

↳ It will not miss the cost of system-level functions.

↳ It requires minimal project detail, and it is usually faster, easier to implement.

Disadvantages:

↳ It often does not identify difficult low-level problems.

↳ It provides no detailed bases for justifying estimates.

2) Bottom-up Approach:

Using this method, the cost of each software components is estimated and then combines the results to arrive at an estimated cost of overall project. It aims at constructing the estimate of a system from the knowledge gained about small software components and their interactions. The leading method using this approach is COCOMO's detailed model.

Advantages:

↳ It allows software group to handle an estimate in an almost traditional fashion.

↳ It is more stable because the estimation errors in the various components have a chance to balance out.

Disadvantages:

↳ It tends to be more time-consuming

↳ It may not be accurate because the necessary information may not be available in the early phase.

⊛ COCOMO Model:

The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model. This model predicts or estimates the effort required for the project, total project cost and scheduled time for the project. This model depends on the number of lines of code for software product development.

In COCOMO, projects are categoried into three types:

i) **Organic:** A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past. Example of this type of projects are simple business systems, simple inventory management systems, and data processing systems.

ii) **Semi-detached:** In this type the development consists of a mixture of experienced and inexperienced staff. Example of Semi-detached system includes developing a new operating system, a database management system, and complex inventory management system.

iii) **Embedded:** A software project with requiring the highest level of complexity, creativity, and experience requirement fall under this category. For example: ATM, Air Traffic control.

Formulas for estimating the effort based on the code size:

Organic: Effort = 2.4 (KLOC) 1.05 PM

Semi-detached: Effort = 3.0 (KLOC) 1.12 PM

Embedded: Effort = 3.6 (KLOC) 1.20 PM

Formulas for estimating the development time based on the effort:

Organic: Tdev = 2.5 (Effort) 0.38 Months

Semi-detached: Tdev = 2.5 (Effort) 0.35 Months

Embedded: Tdev = 2.5 (Effort) 0.32 Months

(LOC means lines of code) (K means thousand)

Example: A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

Solution:

The semi-detached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence,

$$\text{Effort} = 3.0(200)^{1.12} = 1133.12 \text{ PM}$$ (indicates person months)

$$T_{dev} = 2.5(1133.12)^{0.35} = 29.3 \text{ Months}$$

$$\text{Average staff size} = \frac{\text{Effort}}{T_{dev}} = \frac{1133.12}{29.3} = 38.67 \text{ Mans}$$

$$\text{Productivity of software} = \frac{KLOC}{\text{Effort}} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

$$= 176 \text{ LOC/PM}$$

⊛. COCOMO model types:

There are three types of COCOMO model:

1. Basic COCOMO: It is one type of static model to estimate software development effort quickly and roughly. It mainly deals with the number of lines of code and the level of estimation accuracy is less as we don't consider the all parameter belongs to the project. The estimated effort and scheduled time for the project are given by the relation:

$$\text{Effort}(E) = a*(KLOC)^b \text{ PM}$$

$$\text{Scheduled Time } (D) = c*(E)^d \text{ Months (M)}$$

where,

E = Total effort required for the project in Person-Months (PM)

D = Total time required for project development in Months (M).

KLOC = the size of the code for the project in Kilo lines of code.

a, b, c, d = The constant parameters for a software project.

ii) Intermediate COCOMO:

The intermediate model estimates software development effort in terms of size of the program and other related cost driver parameters (product parameter, hardware parameter, resource parameter) of the project. The estimated effort and scheduled time are given by the relationship:

$$\text{Effort (E)} = a^* (\text{KLOC})^b * \text{EAF PM}$$

$$\text{Scheduled Time(D)} = c^* (E)^d \text{ Months}$$

where, EAF = Effort Adjustment Factor, which is calculated by multiplying the parameter values of different cost driver parameters. For ideal, the value is 1.

iii) **Detailed COCOMO:** It is the advanced model that estimates the software development effort like Intermediate COCOMO in each stage of software development life cycle.

⊛. Introduction to Quality Management and Configuration Management:

Software Quality Management ensures that the required level of quality is achieved by submitting improvements to the product development process. Software quality assurance aims to develop culture within the team and it is seen as everyone's responsibility. Software quality management following activities:

i) **Quality Assurance:** QA aims at developing organizational procedures and standards for quality at organizational level.

ii) **Quality Planning:** Select applicable procedures and standards for a particular project and modify as required to develop a quality plan.

iii) **Quality control:** Ensure that best practices and standards are followed by the software development team to produce quality products.

Configuration Management is concerned with the policies, processes and tools for managing changing software systems. It is essential because it is easy to lose track of what changes and component versions have been incorporated into each system version. Configuration management activities are as follows:

→ Change management
→ Version management
→ System building
→ Release management.

Included at last. It is part of Unit-2

Q. What are the drawbacks of software reuse? Explain. [Model Set]

Ans:- Software elements of a product take time to create and reuse of them saves development time. But still it got some drawbacks:

i) Maintenance cost increase: Sometimes the old created components may fail doing their functions resulting high maintenance cost.

ii) Software tools require longer support: As the software tools that were used previously may not be in use today so we have to maintain those tools.

iii) Software tools may become obsolete: Software tools may get outdated after sometime, so we need to keep them up-to-date.

iv) Long Selection Time: It takes time to select which component can be used for long term.

v) Component Failure: One component may not work, then interconnected components will also not work, resulting failure of system.

Q. Differences between alpha testing and beta testing. [May be important, we can find solution everywhere read this in addition].

If my notes really helped you,then you can support me on esewa for my hardwork.

Esewa ID: 9806470952