Chapter::: Software Engineering Paradigms.

- What is software engineering? [NU: 2009, 2011, 2013, 2014,
  &Give the IEEE definition of software engineering. [NU: 2010, 2012,
- What do you mean by software quality? [NU: 2013,
- Classify the qualities of software.
- List the characteristics of software quality. [NU: 2013,
- State and explain some qualities that are used to assess software. [NU: 2014,
  & State and briefly discuss four qualities that are used to assess software. [NU: 2010,
- Explain the attribute of a good software product. [NU: 2012,
- What are the objectives of software engineering? [NU: 2011,
- Mention the key challenges facing software engineering. [NU: 2010, 2013,
- What is meant by software quality assurance? [NU: 2010,
- Discuss about the professional and ethical responsibilities of a software engineer. [NU: 2013,
- Define agility and agile team. [NU: 2014,
- Describe the principle of agile method. [NU: 2009,
  Or, Write down the principles of agile process method. [NU: 2012, 2013,
- What do you know about software crisis? [NU: 2010,
- Write short note on legacy software. [NU: 2013,
- Write short note on CMM. [NU: 2013,
  &What is CMM? [NU: 2011,
  & State and explain process maturity levels in SEIs CMM. [NU: 2014,
  Or, Describe the five levels defined in the SEI process maturity model. [NU: 2011,
- Distinguish between ISO 9000 and SEI CMM. [NU: 2010,
- What are the difference between software engineering and system engineering? [NU: 2009, 2011,
- Distinguish between software engineering and computer science. [NU: 2011, 2012,
  &Is there any difference between software engineering and computer science? Justify your answer.[NU: 2014,
- Write short note on software prototyping. [NU: 2014,
  Or, Describe prototyping process model. [NU: 2009,
  Or, Give a brief description of software prototyping and discuss the various prototyping techniques in a nutshell. [NU: 2011,
- Distinguish between evolutionary prototyping and throw-away prototyping. [NU: 2012, 2014,
- What is software engineering process? [NU: 2009,
- Distinguish between software process and software process model. [NU: 2010,
- Discuss the generic view of software engineering. [NU: 2010,
  Or, State and explain the generic activities followed in all the software process. [NU: 2011,
- What are the umbrella activities of software engineering? [NU: 2012, 2014,
- What is meant by component-based software engineering? Explain. [NU: 2010,
- Briefly describe each step of Software Development Life Cycle (SDLC). [NU: 2009,
- Explain waterfall model with merits and demerits. [NU: 2010, 2012,
- Explain incremental software process model with its merits and demerits. [NU: 2013,
- Write short note on RAD software process model. [NU: 2014,
- Write the applications of the evolutionary development model. [NU: 2010,
- Write short note on spiral model. [NU: 2012, 2014,
- Discuss Rational Unified Process (RUP) model with merits. [NU: 2011, 2014,
- Write short note on V & V software process model. [NU: 2013,

Question: What is software engineering? [NU: 2009, 2011, 2013, 2014

& Give the IEEE definition of software engineering. [NU: 2010, 2012,

Answer: Software Engineering : Software engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

- Fritz Bauer.

The IEEE has developed a more comprehensive definition when it states:

Software engineering:

(i) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software

(ii) The study of approaches as in (1).

Software engineering is a layered technology. It consists of a set of three key elements-

- Methods
- Tools
- Procedure

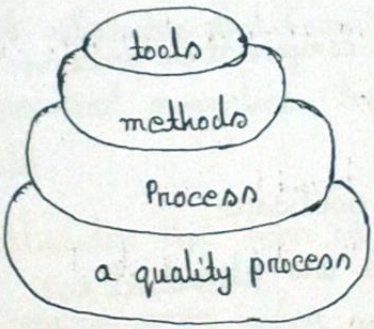


Fig: Software engineering layers.

Method: Method is a board array of task that includes-

- ✓ Project planning and estimation.
- ✓ System and software requirements analysis.
- ✓ Design of structure.
- ✓ Program, architecture and algorithm procedure.
- ✓ coding
- ✓ Testing
- ✓ Maintenance.

Tools: Software engineering tools provide automated or semi-automated support for methods. When tools are integrated so that information created by one tool can be used by another called Computer Aided Software Engineering (CASE).

Procedure: Software engineering procedure holds the methods and tools together and they enable rational and timely development of computer software.

Question: What do you mean by software quality? [NU: 2013,

Answer: Software quality: Software quality means conformance to explicity state functional and performance requirement, explicitly documented development standards, implicit characteristics expected for professionally developed software.

IEEE Definition of software quality:

- The degree to which a system component, or process meets specified requirements
- The degree to which a system component or process meets customer or user needs or expectations.

Question: Classify the qualities of software.

Answer: The qualities of software are classified as:

- ✓ Qualities of design
- ✓ Qualities of conformance

Qualities of design: Qualities of design refers to the characteristics that designers specify for an item. The grade of materials, tolerance and performance specification all contributes the quality of design. As higher graded materials are used and higher tolerance and greater levels of performance specied, the design quality of a product increases, if the product is manufactured according to specification. In software development, quality of design encompasses requirements, specifications and the design of the system

Quality of conformance : It is the degree to which the design specification are followed during manufacturing. Again the greater the degree of conformance, the higher the level of quality of conformance. In software development it focuses primarily on implementation follows the design and the resulting system meets its requirements and performance goals, conformance quality is high.

Question : List the characteristics of software quality. [NU: 2013,

Answer: Software Quality may be evaluted by the following characteristics:

- Functionality.
- Reliability.
- Usability
- Efficiency
- Maintainability.
- Portability.

Software quality characteristics:

Functionality: A set of attributes that bear on the existence of a set of function and their specified properties. The functions are those that satisfy stated or implied needs.

Reliability: A set of attributes that bear on the capability of software to maintain its level of performance under stated condition for a stated period of time.

Usability: A set of attributes that bear on the effort needed for use, and on the individual assessment of such use by a stated or implied set of users.

Efficiency: A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.

Maintainability: A set of attributes that bear on the effort needed to make specified modified modifications.

Portability: A set of attributes that bear on the ability of software to be transferred from an environment to another.

Quality sub-characteristics:

✓ Functionality:
- Suitability.
- Accuracy
- Interoperability.
- Compliance
- Security

✓ Reliability:
- Maturity
- Fault - tolerance
- Recoverability.

✓ Usability:
- Understandability
- Learnability.
- Operability.

✓ Efficiency
- Time Behavior
- Resource behavior.

Question: State and briefly discuss four qualities that are used to access software. [NU: 2010, 2014,

Answer: " Follow previous question Ans."

Question: Explain the attribute of good software product. [NU: 2012,

Answer: The four important attributes of software products are -

Maintainability: Software should be written in such a way that it may evolve to meet the changing needs of customers. This is a critical attribute software change is an inevitable consequence of a changing business environment.

Dependability: It has a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure.

Efficiency: Software should not make wasteful use of system resources such as memory and process cycles. Efficiency therefore includes responsiveness, processing time, memory utilization etc.

Usability: Software must be usable, without undue effort, by the type of user for whom it is designed. This means that it should have an appropriate user interface and adequate documentation.

Question: What are the objectives of software engineering? [NU: 2011,

Answer: Objectives of software engineering:

- Maintainability - the ease with which changes in a functional unit can be performed in order to meet prescribed requirements.
- Correctness - the extent to which software meets its specified requirements.
- Reuseability - the extent to which a module can be used in multiple applications.
- Testability - the extent to which software facilitates both the establishment of test criteria and the evaluation of the software with respect to those criteria.
- Reliability - an attribute of software quality. The extent to which a

program can be expected to perform its intended function over an arbitrary time period.

- Portability - the ease with which software can be transferred from one computer system or environment to another.
- Adaptability - the ease with which software allows differing system constraints and user needs to be satisfied by making changes to the software.

Question: Mention the key challenges facing software engineering. ENU: 2010, 2013, 1, b

Answer: The key challenges facing software engineering:

- The heterogeneity challenge: Increasingly, systems are required to operate as distributed systems across networks that include different types of computers and with different kinds of support systems. It is often necessary to integrate new software with older legacy systems written in different programming languages.
- The delivery challenge: Many traditional software engineering techniques are time-consuming. The time they take is required to achieve software quality. The delivery challenge is the challenge of shortening delivery times for large and complex systems without compromising system quality.
- The trust challenge: As software is intertwined with all aspects of our lives, it is essential that we can trust that software. The trust challenge is to develop techniques that demonstrate that software can be trusted by its users.

Question: What do you mean by software quality assurance? ENU: 2010,

Answer: SQA: Software quality assurance is a set of auditing and reporting function that assess effectiveness and completeness of quality control activities.

SQA is a process that ensures that developed software meets and complies with defined or standardized quality specifications.

Question: Discuss about the professional and ethical responsibilities of a software engineer. [NU: 2013]

Answer: I/we should always uphold normal standards of honesty and integrity. I/we should not use our/my skills and ability to behave in a dishonesty way or in a way that will bring disrepute to the software engineering profession. Theare are areas where standards of acceptable behaviour are not bounded by laws but by the more tenuous notion of professional responsibility. Some of these are:

- Confidently: We/I should normally respect the confidentiality of our employers or clients irrespective of whether a formal confidentiality agreement has been signed.
- Competence: We/I should not misrepresent our level of competence. We/I should not knowingly accept work that is outside your competence.
- Intellectual property rights: I/we should be aware of local laws governing the use of intellectual property of employers and clients is protected.
- Computer misuse: We/I should not use our technical skills to misuse other people's computers. Computer misuse range from relatively trivial to extremely serious.
- Rights: Every intellectual property possess some rights which may come under patents or copyrights. It must not be violated while developing any product.

Thus following some ethical standards in the software world is very much necessary to repute the profession.

Question: Define agility and agile team. [NU: 2014]

Answer: Agility has become today's buzzword when describing a modern software process. Every-one is agile. An agile team is a nimble team able to appropriately respond to changes. change is what software development is very much about. changes in the software being built, changes to the team members, changes because of new technology, changes of all kinds that may have an impact on the product they build or the project that creates the product. An agile team recognizes that software is developed by individuals working in teams and that the skills of these people, their ability to collaborate is at the cover for the success of the project.

Question: Write down the principles of agile process method. [NU: 2009, 2012, 2013]

Answer: The principles of agile process method:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile process harness change for the customer's competitive advantage.
- Deliver working software frequently from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversion.
- Working software is the primary measure of progress.
- Agile process promote sustainable development
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity - the art of maximizing the amount of work not-done is essential.
- The best architectures, requirements and design emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Question: What do you about software crisis? [NU: 2010]

Answer: Software crisis means:-

- Many software projects failed
- Many software projects late, over budget, providing unreliable software
- Many software projects produced software which did not satisfy the requirements of the customer.
- Complexities of software projects increased as hardware capability increased.
- Larger software system is more difficult and expensive to maintain.
- Demand of new software increased faster than ability to generate new software.

Question: Write short note on legecy software. [NU: 2013]

Answer: Lagecy software systems... were developed decades ago and have been continually modified to meet changes in business requirements and computing platforms. The proliferation of such systems is causing headaches for large organizations who find them costly to maintain and risky to evolve.

There is one additional characteristic that can be present in legacy software - poor quality. Lagacy system sometimes have inextensible design, convoluted code, poor or nonexistent documentation.

lagecy system often evolve for one or more of the following reasons:

✓ The software must be adapted to meet the needs of new computing environments or technology.

✓ The software must be enhanced to implement new business requirements.

✓ The software must be extended to make it interoperable with more modern systems or databases.

✓ The software must be re-architected to make it viable within a network environment.

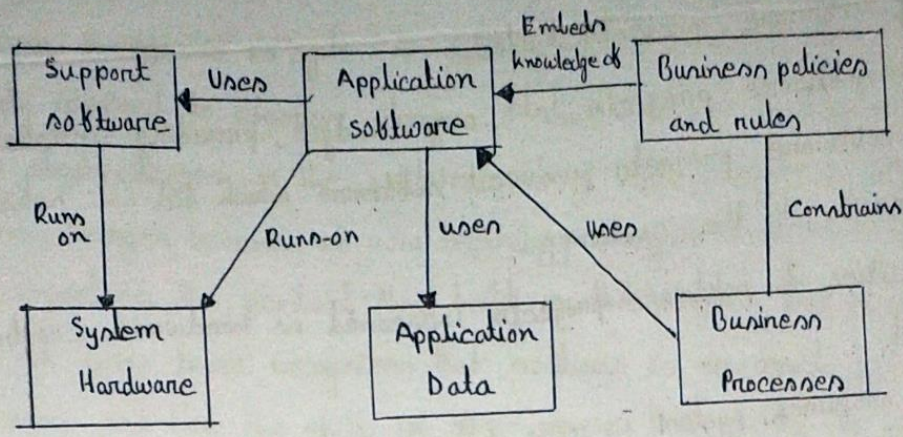


Fig: Legacy system components.

The different logical parts of a legacy system and their relationships

✓ System Hardware: Legacy systems have been written for mainframe hardware which is no longer available, which is expensive to maintain and which may not be compatible with current organisational IT purchasing policies.

✓ Support Software: The legacy system may rely on a range of different support software from the operating system and utilities provided by the hardware manufacture through to the compilers used for system development. Again, these may be obsolete and no longer supported by their original provides.

✓ Application Software: The application system which provides the business services is usually composed of a number of separate programs which have been developed at different times. Sometime the term legacy system means this application software rather than the entire system.

✓ Application Data: These are the data which are processed by the application system. In many legacy systems, an immense volume of data has accumulated over the lifetime of the system. This data may be inconsistant and may be duplicated in different files.

✓ Business processes: These are processes which are used in the business to achieve some business objective. An example of a business process in an insurance company would be issuing an insurance policy, in a manufacturing company, a business process would be accepting an order for products and setting up the associated manufacturing process.

✓ Busines policies and rules: These are definitions of how the business should be carried out and constraints on the business. Use of the legacy application system may be embedded in these policies and rules.

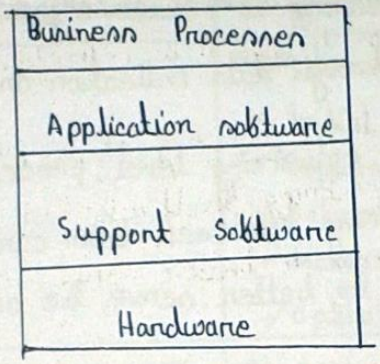


Figure: Layered model of a legacy system.

4, d

Question: Write short note on CMM [NU: 2011, 2013]
& State and explain process maturity levels in SEIs CMM. [2011, 2014]

Answer: CMM: The Software Engineering Institute (SEI) capability Maturity Model (CMM) specifies an increasing series of levels of a software development organization. The higher the level, the better the software development process hence reaching each level is an expensive and time-consuming process.

State and explain process maturity levels in SEIs CMM:

Level ONE : Initial: The software process is characterized as inconsistent, and occasionally even chaotic. Defined process and standard practices that exist are abandoned during a crisis. Success of the organization majorly depends on an individual effort, talent and heroics. The heros eventually move on to

other organizations taking their wealth of knowledge on lessons learnt with them.

Level two: Repeatable: At the repeatable level basic project management techniques are established and successes could be repeated, because the requisite processes would have been made established, defined and documented.

Level three: Defined: At the defined level, an organization has developed its own standard software process through greater attention to documentation, standardization, and integration.

Level four: Managed: At the managed level, an organization monitors and controls its own processes through data collection and analysis.

Level five: Optimizing: At the optimizing level, processes are constantly being improved through monitoring feedback from current processes and introducing innovative processes to better serve the organization's particular needs.

4.d

Question: Distinguish between ISO 9000 and SEI CMM. [NU: 2010]

Answer: Difference between ISO 9000 and SEI CMM:

ISO 9000	CMM
It applies to any type of industry.	CMM is specially developed for software industry
ISO 9000 addresses corporate business process	CMM focuses on the software engineering activities.
ISO 9000 specifies minimum requirement	CMM gets into technical aspect of software engineering.
ISO 9000 restricts itself to what is required	It suggests how to fullfill the requirement
ISO 9000 provides pass or fail criteria	It provides grade for process maturity
ISO 9000 has no levels.	CMM has 5 levels: ✓ Initial ✓ Repeatable ✓ Defined ✓ Managed ✓ Optimization
ISO 9000 does not specifies sequence of steps required to establish the quality system.	It reconnects the mechanism for step by step progress through its successive maturity levels.
Certain process elements that are in ISO are not include in CMM like: • Contract management. • Purchases and customer supplied components	Similarly other process in CMM are not include in ISO 9000 • Project tracking. • Process and technology change management
• Personal issue management. • Packaging, delivery and installation management.	• Intergroup coordinating to meet customer's requirement. • Organization level process focus, process development and integrated management.

Question: What are the difference between software engineering and computer science ? [NU: 2009, 2011,

Answer: The difference between software engineering and computer science:

Software Engineering	Computer Science
The software engineering is concerned with the practical problems of producing software.	The computer science deals with the theories and methods used by the computers and software systems.
Some knowledge of computer science is necessary for the software engineers to develop the software	Elegent theories cannot be completely applicable to the software engineering when software solution to complex or real world problem has to be developed

Question: What are the difference software engineering and system engineering ? [NU: 2011, 2012, 2014, 1, a

Answer: The difference between software engineering and system engineering:

System Engineering	Software Engineering
It deals with aspects of computer system development	Software engineering is a part of system engineering.
In system engineering the overall objective of the system must be defined. In the system engineering the role of hardware, software, people, database, procedures and other system elements must be identified.	The software engineering is concerned with practical problems of producing software.

Question: Write short note on software prototyping. [NU: 2009, 2014,

Answer: Software Prototyping: Prototype is a working model of software with some limited functionality. The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation. Prototyping is used to allow the users evaluate developer proposals and try them out before implementation. It also help understand the requirements which are user specific and may not have been considered by the developer during product design.

Software Prototyping types:

- Throwaway Prototyping: Throwaway prototyping is also called as rapid or close ended prototyping. This type of prototyping uses very little efforts with minimum requirements analysis to build a prototype.
- Evolutionary Prototyping: Evolutionary prototyping also called as bread-board prototyping is based building actual functional prototypes with minimal functionality in the beginning. Using evolutionary prototyping only well understood requirements are added as and when they are understood.
- Incremental Prototyping: Incremental prototyping refers to building multiple functional prototypes of the various sub systems and then integrating all the available prototypes to form a complete system.
- Extreme Prototyping: Extreme prototyping is used in the web development domain. It consists of three sequential phases. First, a base prototype with all the existing pages is presented in the html format. Then the data processing is simulated using a prototype services layer. Finally the services are implemented and integrated to the final prototype. This process is called extreme prototyping

used to draw attention to the second phase of the process, where a full functional UI is developed with very regard to the actual services.

Question: Distinguish between evolutionary prototyping and throw-away prototyping. [NU: 2012, 2014]

Answer: Difference between evolutionary prototyping and throw-away prototyping:

Evolutionary Prototype	Throw-away Prototype
The principal objective of evolutionary model is to deliver the working system to the end-user.	The principle objective of throw away prototype is to validate or derive the system requirements.
The process of development starts with well understood requirements	The process of development starts with poorly understood requirements
It must be developed for the systems where the specifications cannot be developed in advance	The throw away prototype is developed to reduce the requirement risks.

Question: What is software engineering process? [NU: 2009,

Answer: Software engineering Process: A software engineering process is the model chosen for managing the creation of software from initial customer inception to the relase of the finished product. The chosen process usally involves techniques such as,

- Analysis,
- Design,
- Coding,
- Testing and
- Maintenance.

Question: Distinguish between software process and software process model. [NU: 2010,

Answer: Software process: A software process is the set of activities and associated results that produce a software product. There are four fundamental process activities that are common to all software processes. They are:

- ✓ Software Specification
- ✓ Software Development
- ✓ Software validation
- ✓ Software evolution.

Software process model: A software process model is a simplified description of a software process that one view of that process. Process models may include activities that are part of the software process, software products and the roles of people involved in software engineering. The types of software process model:

- ✓ A waterfall model.
- ✓ A dataflow or activity model.
- ✓ A role/action model.

Question: State and explain the generic activities followed in all the software process. [NCE: 2010, 2011]

Answer: A process framework establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity. The process framework encompasses a set of umbrella activities that are applicable across the entire software process.

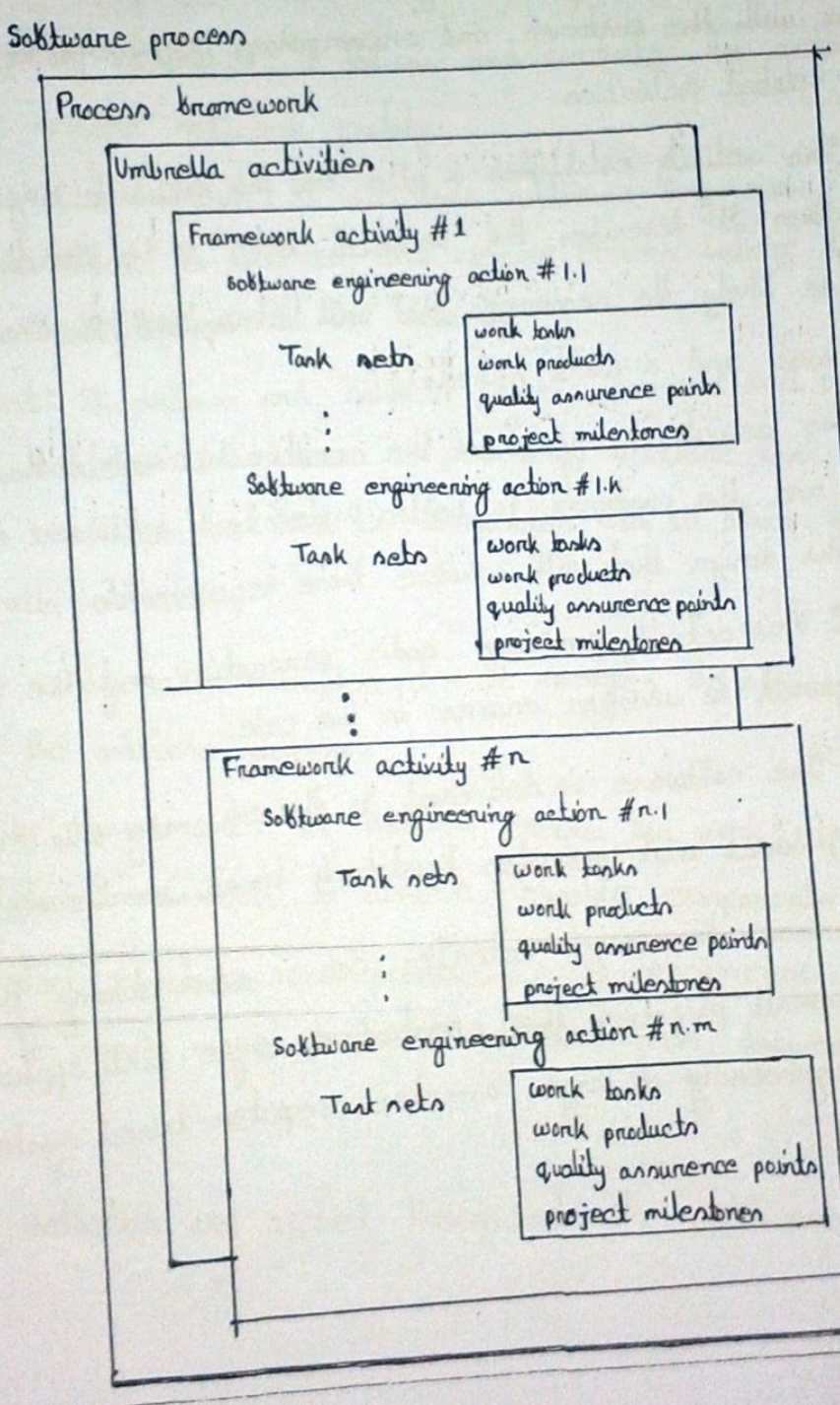


Figure: Generic activities.

From figure - each framework activity is populated by a set of software engineering actions - a collection of related tasks that produces a major software engineering product. Each action is populated with individual work tasks that accomplish some part of the work implied by the action:

The following generic process framework is applicable to the vast majority of software projects:

- **Communication:** This framework activity involves heavy communication and collaboration with the customer and encompasses requirements gathering and other related activities.
- **Planning:** This activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.
- **Modeling:** This activity encompasses the creation of models that allow the developer and the customer to better understand software requirements and the design that will achieve those requirements.
- **Construction:** This activity combines code generation and the testing that is required to uncover errors in the code.
- **Development:** The software is delivered to the customer who evaluates the delivered product and provides feed-back based on the evaluation.

These five generic framework activities can be used during the development of small programs, the creation of large web applications and for the engineering of large, complex computer-based systems.

Question: What are the umbrella activities of software engineering? [NU: 2010,201

Answer: The framework described in the generic view of software engineering is complemented by a number of umbrella activities. Typical activities in this category include:

- Software project tracking and control: It allows the software team to assess progress against plan and take necessary action to maintain schedule.
- Risk management: It assesses risks that may effect that may effect the outcome of the project on the quality of the product.
- Software quality assurance: It defines and conducts the activities required to ensure software quality.
- Formal technical reviews: It assesses software engineering work products in effort to uncover and remove errors before they are propagated to the next action or activity.
- Measurement: It defines and collects process, project and product measures that assist the team in delivering software that meets customers needs, can be used in conjunction with all other framework and umbrella activities.
- Software configuration management - It manages the effect of change throughout the software process.
- Reusability management: It defines criteria for work product reuse and establishes mechanisms to achieve reusable components
- Work product preparation and production: It encompases the activities required to create work products such as models, documents, logs, forms and lists.

Umbrella activities are applied throughout the software process.

Question: What is meant by component based software engineering? ENU: 2010,

Answer: CBSE: Component based software engineering emerged in the late 1990s as a reuse-based approach to software systems development. Its creation was motivated by designers frustration that object-oriented development had not led to extensive reuse, as originally suggested.

CBSE is the process of defining, implementing and integrating or composing loosely coupled independent components into systems. It has become as an important software development approach because software systems are becoming larger and more complex and customers are demanding more dependable software that is developed more quickly. The only way that we can with complexity and deliver better software more quickly is to reuse rather that re-implement software components.

The essential of component-based software engineering are:

✓ Independent components that are completely specified by their interfaces.

✓ Component standards that facilitate the integration of components.

✓ Middleware that provides software support for component integration.

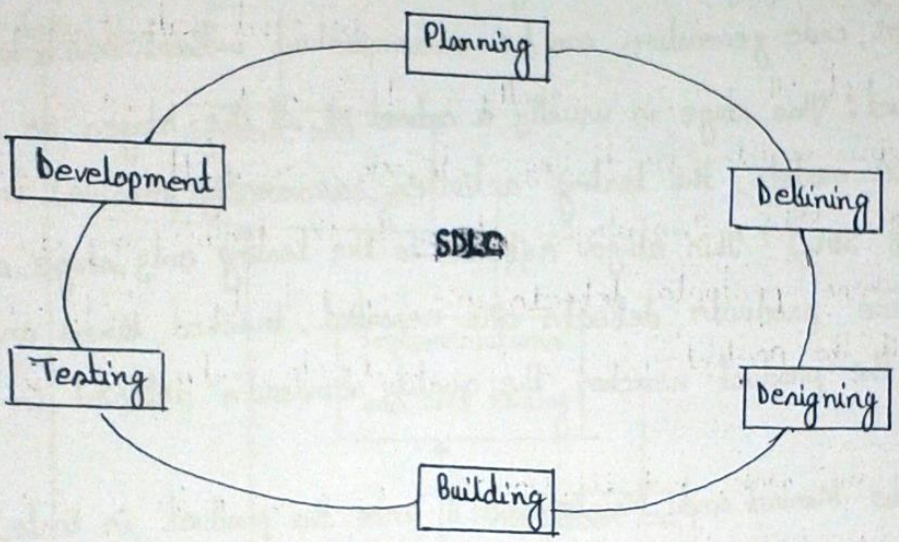
✓ A development process that is geared to component-based software engineering.

The CBSE process is characterized in a manner that not only identifies candidate components but also qualifies each component's interface, adapts components to remove architectural mismatches, assembles components into a selected architectural style, and updates components as requirements for the system change.

Question: Briefly describe each step of Software Development Life Cycle (SDLC). [NU: 2009]

Answer: SDLC: SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



Step of Software Development life Cycle:

- Planning and requirement analysis: Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage.

- Defining Requirement: The requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysis. Software Requirement Specification document which consists of all the product requirements to be designed and developed during the project life cycle.

- Designing the product architecture: SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.
- Building or Developing the product: In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during the stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.
- Testing the product: This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. This stage refers to the testing only stage of the product where products defects are reported, tracked, fixed and retested, untill the product reaches the quality standards defined in the SRS.
- Deployment in the Market and Maintenance: Once the product is tested and ready to be deployed it is released formally in the a app market. Sometime product deployment happens in stages as per the organizations, business strategy. The product may first be released in a limited segment and tested in the real business environment.

Question: Explain waterfall model with merits and demerits. [NU: 2010, 2012

Answer: The waterfall model, sometimes called the classic life cycle suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction and development, culminating is on-going support of the completed software. The waterfall model is the oldest paradigm for software engineering.

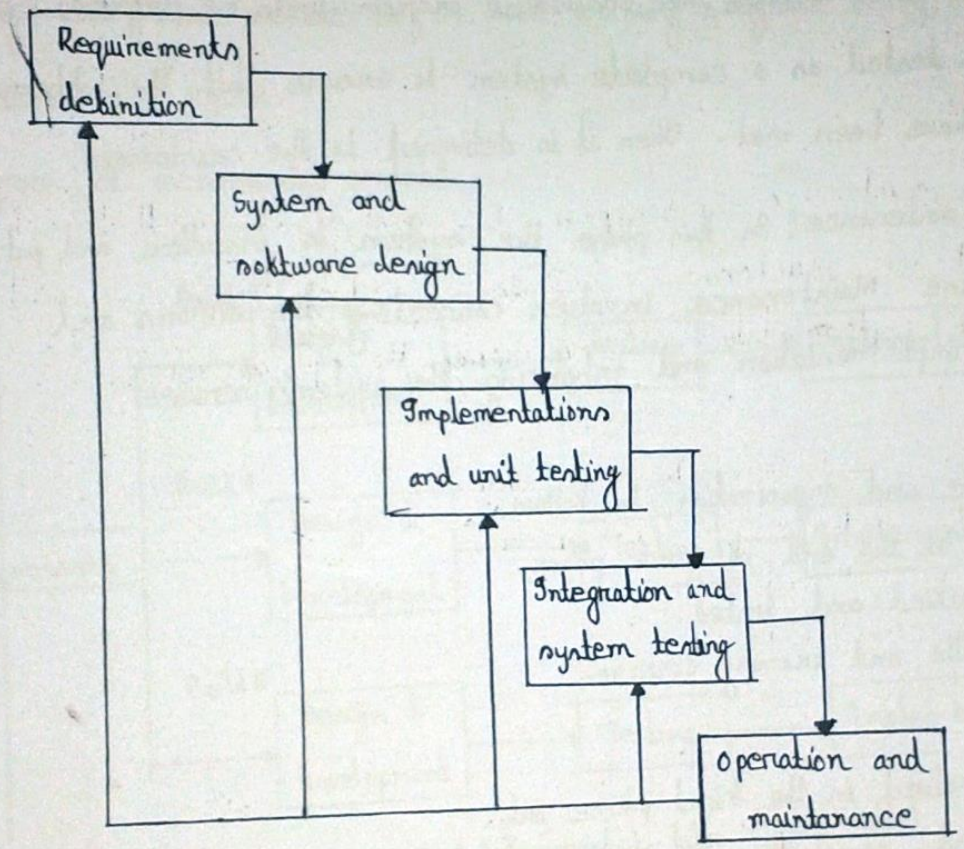


Fig: Software life cycle.

The following lists the process step and corresponding activities for each phase:

Requirements analysis and design: The system's service, constrains and goals are established by consultation with system user and then defined which is understandable by users and development staffs.

System and software design: The system design process partitions the hardware and software requirements and establishes overall system architecture. Software design involves representing the software system functions.

Implementation and unit testing: During this stage, the software design is realized as a set of program or program units. Unit testing involves verifying that each unit meets its specification.

Integration and system testing: The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. Then it is delivered to the customer.

Operation and maintenance: In this phase the system is installed and put into practical use. Maintenance involves correcting the errors and improving the implementation and enhancing the system's service.

Merits:

- Clear structure and organization to follow.
- Documentation at the end of each phase.
- Each step verified and tested.
- Established skills and training courses.

Demerits:

- Customer involved in the first phase only.
- Sequential and complete execution of phases often not desirable.
- Process difficult to control.
- The product becomes available very late in the process.

Question: Explain incremental software process model with its merits and demerits. [NU: 2013]

Answer: Incremental model the whole requirement is divided into various builds. Multiple development cycles take place here, making the life cycle a "multi-waterfall" cycle. Cycles are divided up into smaller, more easily managed modules. Each module process through the requirements design, implementation and testing phases. A working version of software is produced during the first module, so we have working software early on during the software life cycle.

Diagram of incremental model:

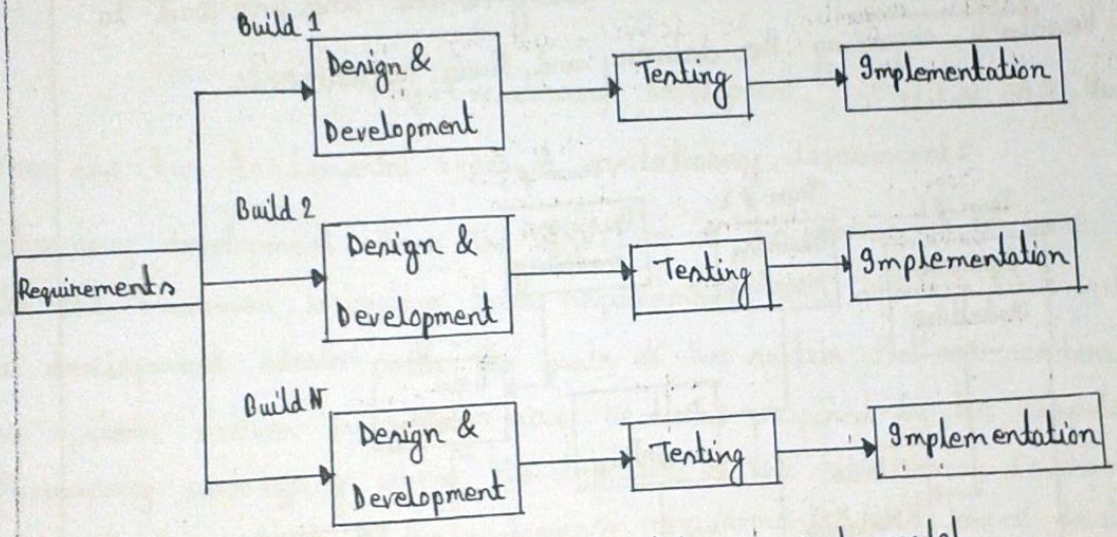


Fig: Incremental life cycle model.

Merits of Incremental model:

- Generates working software quickly and early during the software life cycle.
- This model is more flexible - less costly to change scope and requirements
- It is easier to test debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it'd iteration.

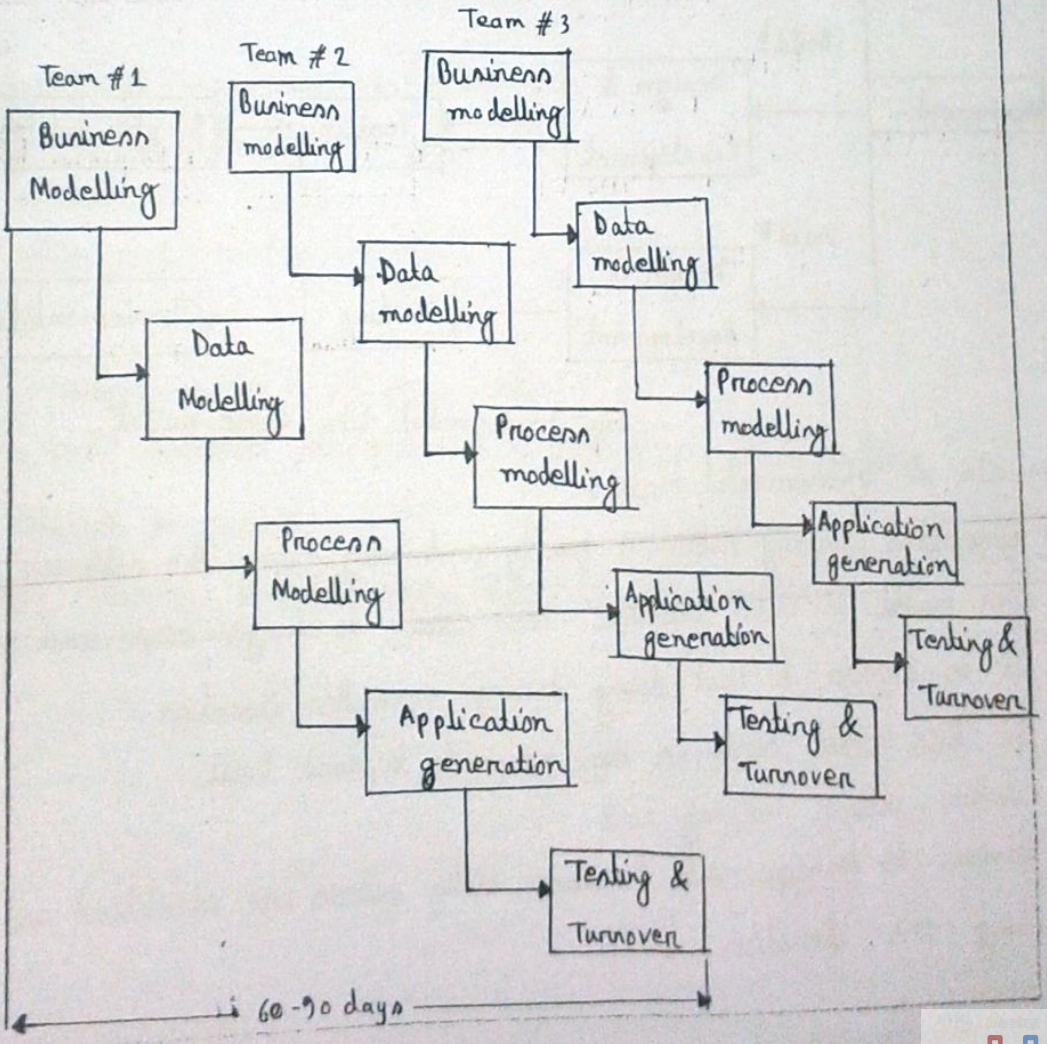
Demerits of incremental model:

- Needs good planning a design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

Question: Write short note on RAD software process model. [NU: 2014,

Answer: RAD model is Rapid Application Development model. It is a type of incremental model. In RAD model the components or functions are developed in parallel as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype. This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirments.

Diagram of RAD Model:



Question: Write the applications of the evolutionary model. [NU: 2018,

Answer: Evolutionary development is based on the idea of developing an initial implementation, exposing this to user comment and refining it through many versions until an adequate system has been developed.

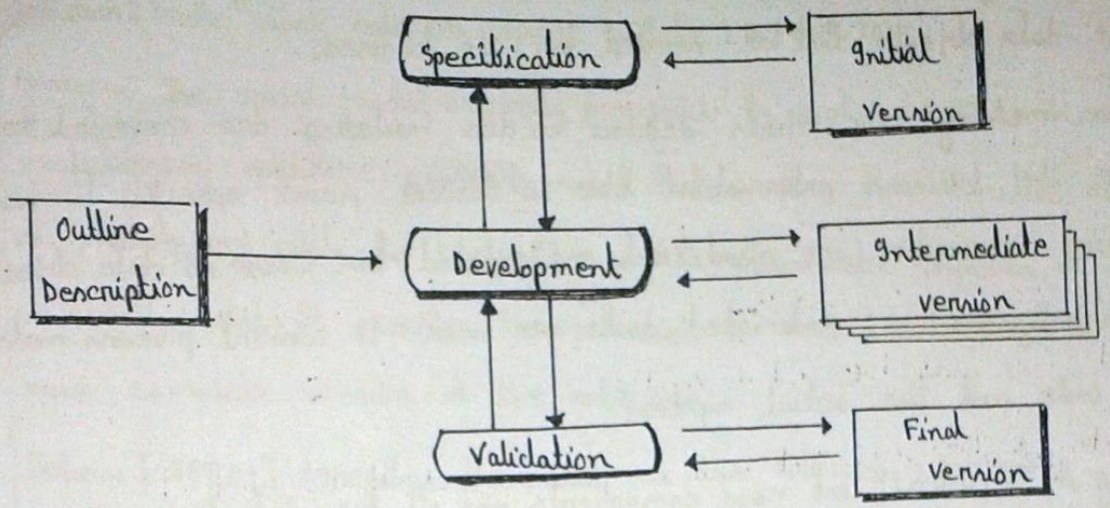


Fig: Evolutionary Development.

There are two fundamental types of evolutionary development:

- Exploratory development where the objective of the process is to work with the customer to explore their requirements and deliver a final system. The development starts with the parts of the system that are understood. The system evolves by adding new features proposed by the customer.
- Throwaway prototyping where the objective of the evolutionary development process is to understand the customer's requirements and hence develop a better requirements definition for the system. The prototype concentrates on experimenting with the customer requirements that are poorly understood.

Advantage of Evolutionary Development –

✓ Specification can be developed incrementally.

✓ As users develop a better understanding of their problem, this can be reflected in the software system.

The phases in the rapid application development (RAD) model are:

Business modelling: The information flow is identified between various business functions.

Data modeling: Information gathered from business modeling is used to define data objects that are needed for the business.

Process modeling: Data objects defined in data modeling are converted to achive the business information flow to achieve some specific business objective. Description are identified and created for CRUD of data objects.

Application generation: Automated tools are used to convert process models into code and the actual system.

Testing & Turnover: Test new components and all the interfaces.

Merits of the RAD model:

- ✓ Reduced development time.
- ✓ Increases reusability of components.
- ✓ Quick initial reviews occur.
- ✓ Encourages customer feedback.
- ✓ Integration from very beginning solves a lot of integration issues.

Demerits of the RAD model:

- ✓ Depends on strong team and individual performances for identifying business requirements.
- ✓ Only system that can be modularized can be built using RAD
- ✓ Requires highly skilled developers/designers.
- ✓ High dependency on modeling skills.
- ✓ Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

Disadvantages of Evolutionary Development:

- ✓ The process is not visible.
- ✓ System are often poorly structured.
- ✓ Special tools and technique may be required.

Question: Write short note on spiral model. [NU: 2012, 2014,

Answer: The spiral model originally proposed by Boehm (BOE88), is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systemmatic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complete version of the software.

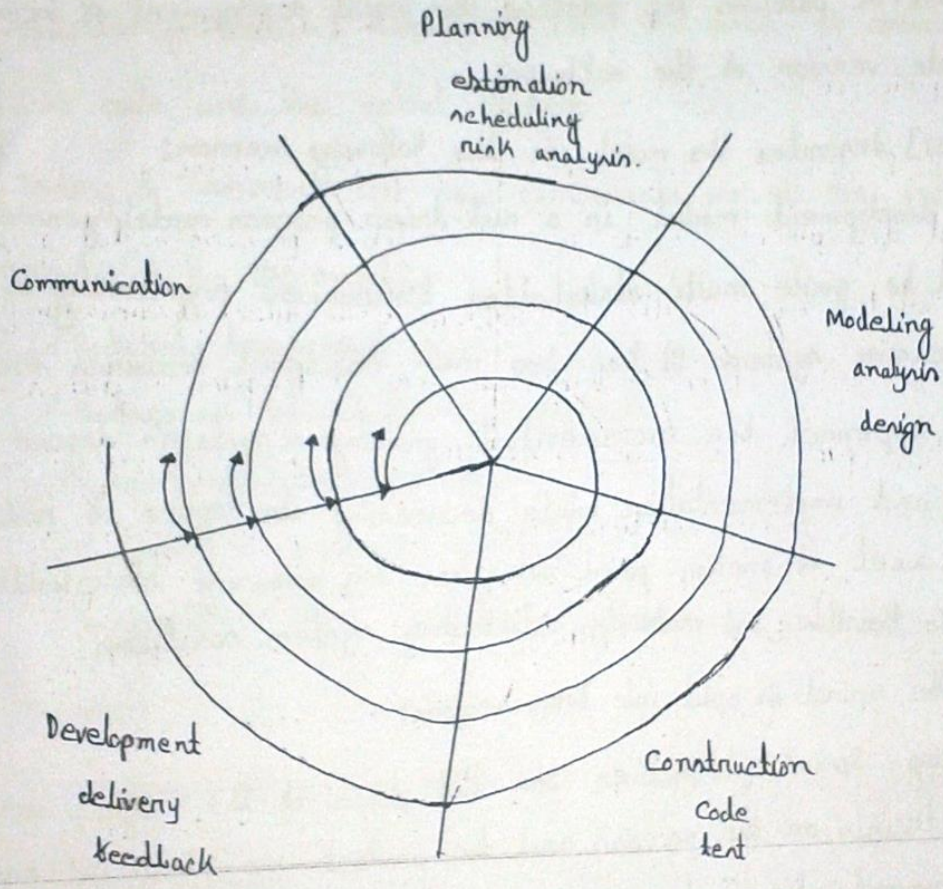
Bohem [BOE01] describes the model in the following manner:

The spiral development model is a risk-driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems. It has two main distinguish features. One is a cycle approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk. The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.

Each loop in the spiral is split into four sectors:

- Objective setting - Specific objectives for that phase of the project are defined. Constraints on the process and the product are identified and a detailed management plan is drawn up. Project risks are identified. Alternative strategies, depending on these risks, may be planned.
- Risk assessment and reduction - For each of the identified project risks, a detailed analysis is carried out. Steps are taken to reduce the risk. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.

- Development and validation - After risk evaluation a development model for the system is chosen. For example, if user interface risks are dominant, an appropriate development model might be evolutionary prototyping. If safety risks are the main consideration, development based on formal transformation may be the most appropriate and so on.
- Planning - The project is reviewed and a decision made whether to continue with a further loop of the spiral. If it is decided to continue, plans are drawn up for the next phase of the project



Merits of spiral model:

✓ It is a realistic approach to the development of large scale systems and software.

✓ Uses prototyping as a risk reduction mechanism.

✓ Enables the developer to apply the prototyping approach at any stage of the evolution.

✓ It uses systematic stepwise approach, but incorporates it into iterative frame work.

✓ Demands a direct consideration of technical risks at all stages.

✓ Should reduce risks before they become problematic.

Demerits of spiral model -

✓ Difficult to convince customers that the evolutionary approch is controllable.

✓ If a major risk is not uncovered and manged, problem will undoulte-dly occur.

✓ It is relatively new and has not been used widely.

Question: Discuss Rational unified Process (RUP) model with merits. TNU: 2011; 2014

Answer: RUP has a series of phases and milestones that blow into each other, phases consist of:

- Inception: Projects scope, estimated contri, risk, business case, environment and architecture are identified.
- Elaboration: Where requirements are specified in detail, architecture is validated, the project environment is further defined and the project team is configured.
- Construction: Where the software is built and tested and supporting documentation is produced.
- Transition, where the software in system tested, user tested, reworked and deployed.

Advantages of RUP:

- Process details are expressed in general terms, allowing local customization.
- Heavy emphasis on documentation (UML)
- Can embrace incremental releases.
- Evolutionary approach can lead to clean implementations.

Disadvantages of RUP:

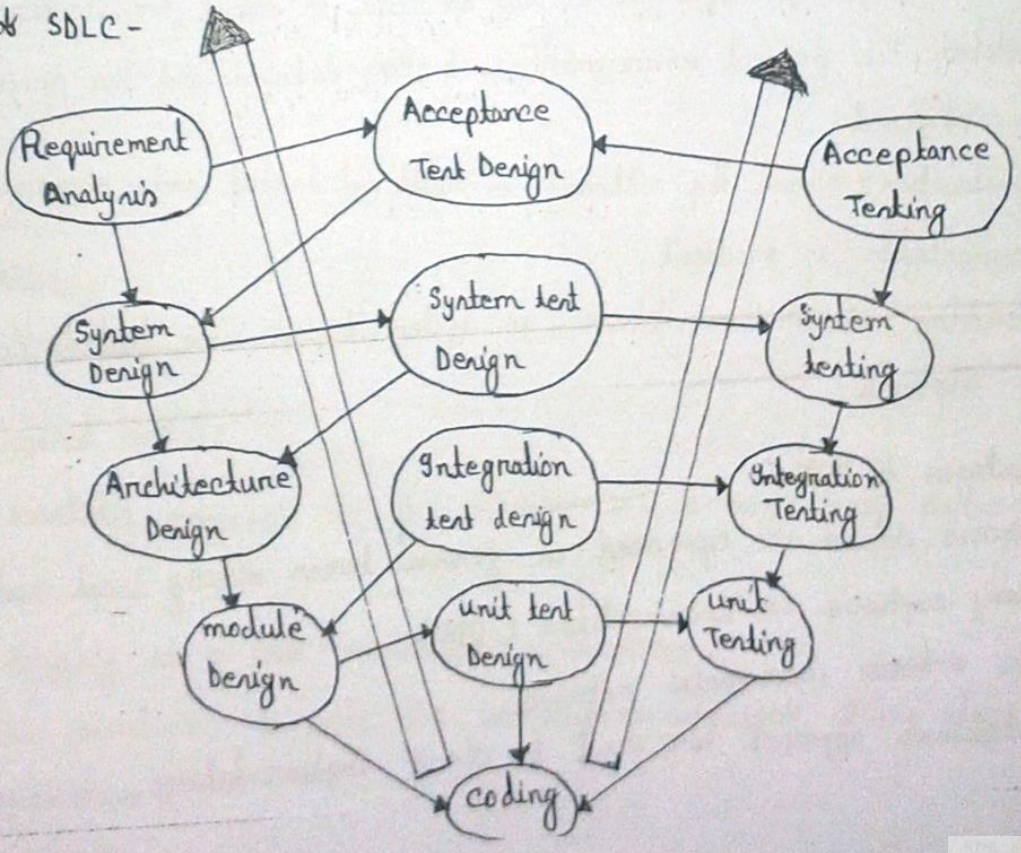
- Process details are expressed in general terms, providing minimal guidance and requiring local customization.
- Complex.
- Heavy documentation can be expensive.

Question: Write short note on V & V software process model. [NU: 2013,

Answer: The V-model is SDLC model where execution of process happens in a sequential manner in V-shape. It is also known as verification and validation model.

V-model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase. This is a highly disciplined model and phase starts only after completion of the previous phase.

V-model design: The below figure illustrates the different phases in V-model of SDLC-



Validation Phases: Following are the validation phases in V-model-

- Unit testing: Unit tests designed in the module design phase are executed on the code during this validation phase.
- Integration Testing: Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.
- System testing: System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems.
- Acceptance testing: Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment.

V-model application:

V-model application is almost same as waterfall model, as both the models are of sequential type. Following are the suitable scenarios to use V-model-

- ✓ Requirements are well defined, clearly documented and fixed.
- ✓ Product definition is stable.
- ✓ Technology is not dynamic and is well understood by the project team.
- ✓ There are no ambiguous or undefined requirements.
- ✓ The project is short.

Advantage of V-model:

- ✓ It is very easy to understand and apply.
- ✓ The simplicity of this model also makes it easier to manage.

Disadvantage of V-model:

- ✓ The model is not flexible to change
- ✓ It becomes very expensive to make the change.

Verification phases:-

Following are the verification phases in V-model:

Business requirement analysis: This is the first phase in the development cycle where the product requirement are understood from the customer perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement.

System Design: Once we have the clear and detailed product requirements, its time to design the complete system. System design would comprise of understanding and detailing the computer hardware and communication setup for the product under development.

Architecture design: Architectural specifications are understood and designed in this phase. Usually more that one technical approach is proposed are based on the technical and financial feasibility the final decision is taken. This is also referred to as High Level Design (HLD).

Module design: In this phase the detailed internal design for all the system modules is specified, referred to as Low Level Design (LLD). Unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage.

Coding phase:

The actual coding of the system module designed in the designer phase is taken up in the coding phase. The best suitable programming language is decided based on the system and architectural requirements. The coding is performed based on the coding guidelines and standards.

Chapter::: Requirements Analysis Fundamentals

- Define requirement engineering (RE).[NU: 2011,
  &What are the different types of user requirement process?[NU: 2012,
- What are the functional and non-functional requirements of software engineering? Explain.[NU: 2010,
  &List some non-functional requirements of software and describe them. [NU: 2014,
  &Explain the functional and non-functional requirements of software engineering process. [NU: 2013,
- What do you know about Software Requirement Specification (SRS)?[NU: 2011,
  &Write short note on Software Requirement Specification (SRS).[NU: 2012,
- Distinguish between requirement definition and requirement specification. [NU: 2013,
- Define the terms stack holder and use case. [NU: 2011, 2013
- "All stack holder should be involved in requirements elicitation and analysis" - justify.[NU: 2010,
  &Brief the process activities of requirements elicitation and analysis with figure. [NU: 2014,
- What are the factor consider in the case of requirement validation? [NU: 2012,
- Draw a use case diagram that depicts the structural requirements of a Library Management System.[NU: 2009,

1.d

Question: Define requirement engineering (RE). [NU: 2011,
& What are the different types of user requirement process? [NU: 2012,

Answer: (Requirement Engineering: Requirement engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, specifying the solution unambiguously, validating the specification and managing the requirements as they are transformed into an operational system.)

The requirements engineering process is accomplished through the execution of seven distinct functions:

- ✓ Inception
- ✓ Elicitation
- ✓ Elaboration
- ✓ Negotiation
- ✓ Specification

☑ Inception: In inception, software engineers ask a set of context free question. The intent is to establish a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired and the effectiveness of preliminary communication and collaboration between the customer and the developer.

☑ Elicitation: It certainly seems simple enough - ask the customer, the users, and the others what the objectives for the system or product are, what is to be accomplished, how the system or product fits into the needs of the business, and finally, how the system or product is to be used on a day-to-day basis. But it isn't simple - it's very hard.

A number of problem that help us understand why requirements elicitation is difficult:

✓ Problems of scope: The boundary of the system is ill-defined or the customers/users specify unnecessary technical detail that may confuse, rather than clarify, overall system objectives.

✓ Problems of understanding: The users are not completely sure of what is needed, have a poor understanding of the capabilities and limitations of their computing environment, don't have a full understanding of the problem domain.

✓ Problems of volatility: The requirements change over time.

⊡ Elaboration: Elaboration is an analysis modeling action that is composed of a number of modeling and requirements tasks. Elaboration is driven by the creation and refinement of users scenarios that describe how the end-user with interact with the system.

⊡ Negotiation: The requirement engineer must reconcile these conflicts through a process of negotiation. Customers, users and other stakeholders are asked to rank requirements and then discuss conflicts in priority. Risks associated with each requirement are identified and analyzed.

⊡ Specification: The term specification means different people. A specification can be a written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype, or any combination of these. The specification is the final work product by the requirement engineer.

⊡ Validation: The work products produced as a consequence of requirements engineering are assessed for quality during a validation step. Requirements validation examines the specification to ensure that all software requirements have been stated unambiguously, that inconsistencies, omissions, and errors have been detected and corrected and that the work products conform to the standards established for the process, the project and the product.

Question: Explain the functional and non-functional requirements of software engineering process. ENU: 2010, 2013, 2014, 2, C

Answer: Software system requirements are often classified as:

- ✓ Functional Requirements
- ✓ Non-functional Requirements.

☐ Functional Requirement: These are statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do.

Functional requirement for a software system may be expressed in a number of ways,

- The user shall be able to search either all of the initial set of data-bases or select a subset from it.
- The system shall provide appropriate viewers for the user to read documents in the document store.
- Every order shall be allocated a unique identifier (ORDER_ID), which the user shall be able to copy to the account's permanent storage area.

☐ Non-Functional Requirement: These are constraints on the services or function offered by the system. They include timing constraints, constraints on the development process and standards. Non-functional requirements often apply to the system as a whole. They do not usually just apply to individual system features or services.

The types of non-functional requirements are:

- Product requirements - These requirements specify product behaviour. Examples include performance requirements on how fast the system

must execute and how much memory it requires, reliability requirements that set out the acceptable failure rate; portability requirement, and usability requirements.

- Organisational requirements - These requirements are derived from policies and procedure in the customers and developer's organization. Examples include process standards that must be used, implementation requirements such as the programming language or design method used and delivery requirements that specify when the product and its documentation are to be delivered.
- External requirements - This broad heading covers all requirements that are derived from factors external to the system and its development process. These may include interoperability requirements that define how the system interacts with systems in other orgnisations. Ethical requirements are requirements placed on a system to ensure that it will be acceptable to its users and the general public.

Question: Define the terms stack holder and use case. [NU: 2011, 2013,

Answer: Stakeholder: Stakeholder is anyone in the organization who has a direct business in the system or product to be built. Stakeholder is anyone who stake in successful outcome of project such as:

- Business Managers
- End-users
- Software Engineer
- Support People.

Question: While short note on Software Requirement Specification (SRS)? [ENU: 2011/2012]

Answer: SRS: An SRS is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies at a particular point in time (usually) prior to any actual design or development work. It's a two-way insurance policy that both the client and the organization understand the other's requirements from that perspective at a given point in time.

An SRS contains functional and nonfunctional requirements only, it doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be.

A well-designed, well written SRS accomplishes four major goals:

- It provides feedback to the customer. An SRS is the customer's assurance that the development organization understands the issues or problems to be solved and the software behavior necessary to address those problems. Therefore, the SRS should be written in natural language, in an unambiguous manner that may also include charts, tables, data flow diagrams, decision tables and so on.
- It decomposes the problem into component parts. The simple act of writing down software requirements in a well-designed format organizes information, places borders around the problem, solidifies ideas, and helps break down the problem into its component parts in an orderly fashion.
- It serves as an input to the design specification. As mentioned previously, the SRS serve as the parent document to subsequent documents, such as the software design specification and statement of work.
- It serves as a product validation check. The SRS also serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.

Qualities of SRS:

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance
- Verifiable
- Modifiable
- Traceable

Question: Distinguish between requirement definition and requirement specification. [NU: 2013]

Answer: Difference between requirement definition and requirement specification:

Requirement	Specification
• A requirement can be any need or expectation for software development.	• A specification is defined as a document that states requirements.
• Requirement reflect the stated or implied needs of the customer, and may be market-based, contractual or statutory as well as an organization's internal requirements.	• It may refer to or include drawing, patterns or other relevant documents and usually indicates the means and the criteria where by conformity with the requirement can be checked.
• There can be many different kinds of requirement (e.g design, functional, implementation, interface, performance or physical requirements).	• There are many different kinds of written specification, e.g. system requirements specification, software requirements specification, software design specification, software test specification, software integration specification etc.

1d

Question: "All stock holder should be involved in requirements elicitation and analysis" - Justify. TNU: 2013, 2014,

Answer: Requirements elicitation and analysis may involve a variety of people in an organisation. The term stakeholder is used to refer to any person or group who will be affected by the system, directly or indirectly. Stakeholders include end-users who interact with the system and everyone else in an organisation that may be affected by its installation. Eliciting and understanding stakeholder requirements is difficult for several reasons:

✓ Stakeholders often don't know what they want from the computer system except in the most general term. They may find it difficult to articulate what they want the system to do or make unrealistic demands because they are unware of the cost of their requests.

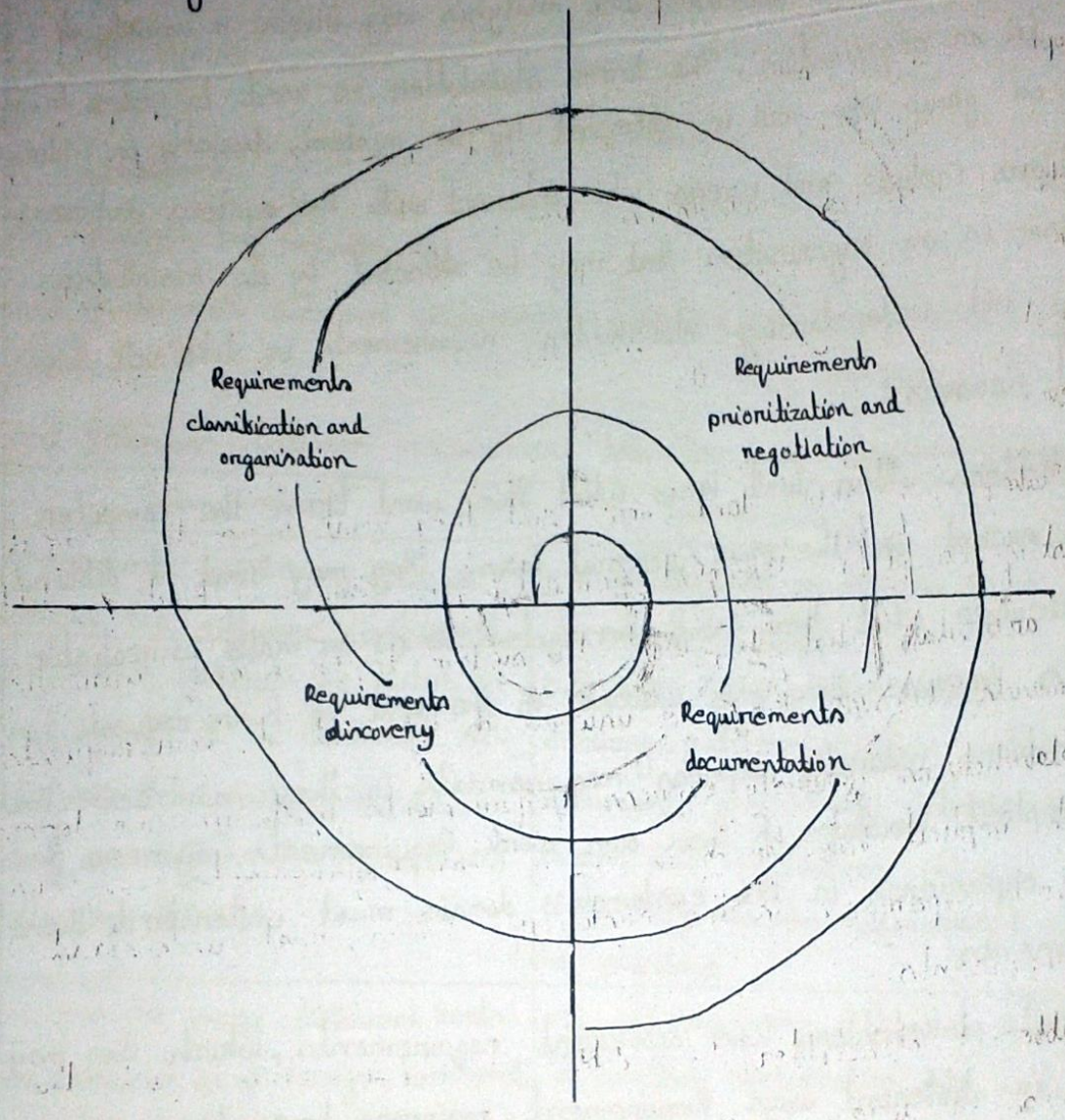
✓ Stakeholders naturally express requirements in their own terms and with implicit knowledge of their own work. Requirements engineers, without experience in the customer's domain, must understand these requirements.

✓ Different stakeholders have different requirements, which they may express in different ways. Requirement engineers have to consider all potential sources of requirement and discover commonalities and conflict.

✓ Political factors may influence the requirements of the system.

✓ The economic and business environment in which the analysis taken place is dynamic. It inevitably changes during the analysis process. Hence the importance of particular requirement may change.

A very general process model of the elicitation and analysis process is shown in figure-



The process activities are:

- Requirements discovery - This is the process of interacting with stakeholders in the system to collect their requirements.
- Requirements classification and organisation - This activity takes the unstructured collection of requirements, groups related requirements and organises them into coherent clusters.
- Requirements prioritisation and negotiation - Inevitably, where multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritising requirements, and finding and resolving

requirement conflicts through negotiations.

⊡ Requirement documentation: The requirement are documented and input into the next round of the spiral. Formal or informal requirements documents may be produced.

Question: What are the factor considered in the case of requirement validation? [NU: 2012,

Answer: Requirement validation is concerned with showing that the requirements actually define the system that the customer wants. Requirements validation overlaps analysis in that it is concerned with finding problems with the requirements. Requirements validation is important because errors in a requirements document can lead to extensive rework costs when they are discovered during development or after the system is in service.

During the requirements validation process, checks should be carried out on the requirements in the requirement document. These checks include:

⊡ Validity checks: A system is needed to perform certain functions. Systems have diverse stakeholders with distinct needs and any set of requirements is inevitably a compromise across the stakeholder community.

⊡ Consistency checks- Requirements in the document should not conflict. That is there should be no contradictory constraints or descriptions of the same system function.

⊡ Completeness checks: The requirements document should include requirements, which define all functions and constraints intended by the system user.

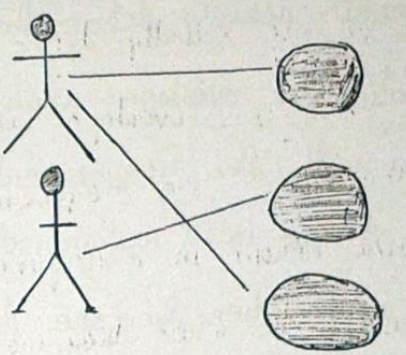
⊡ Realism checks: Using knowledge of existing technology, the requirements should be checked to ensure that they could actually be implemented.

☐ Verifiability: To reduce the potential for dispute customer and contractor, system requirements should always be written so that they are verifiable.

Question: Draw a use case diagram that depicts the structural requirements of a library management system. [NU: 2009]

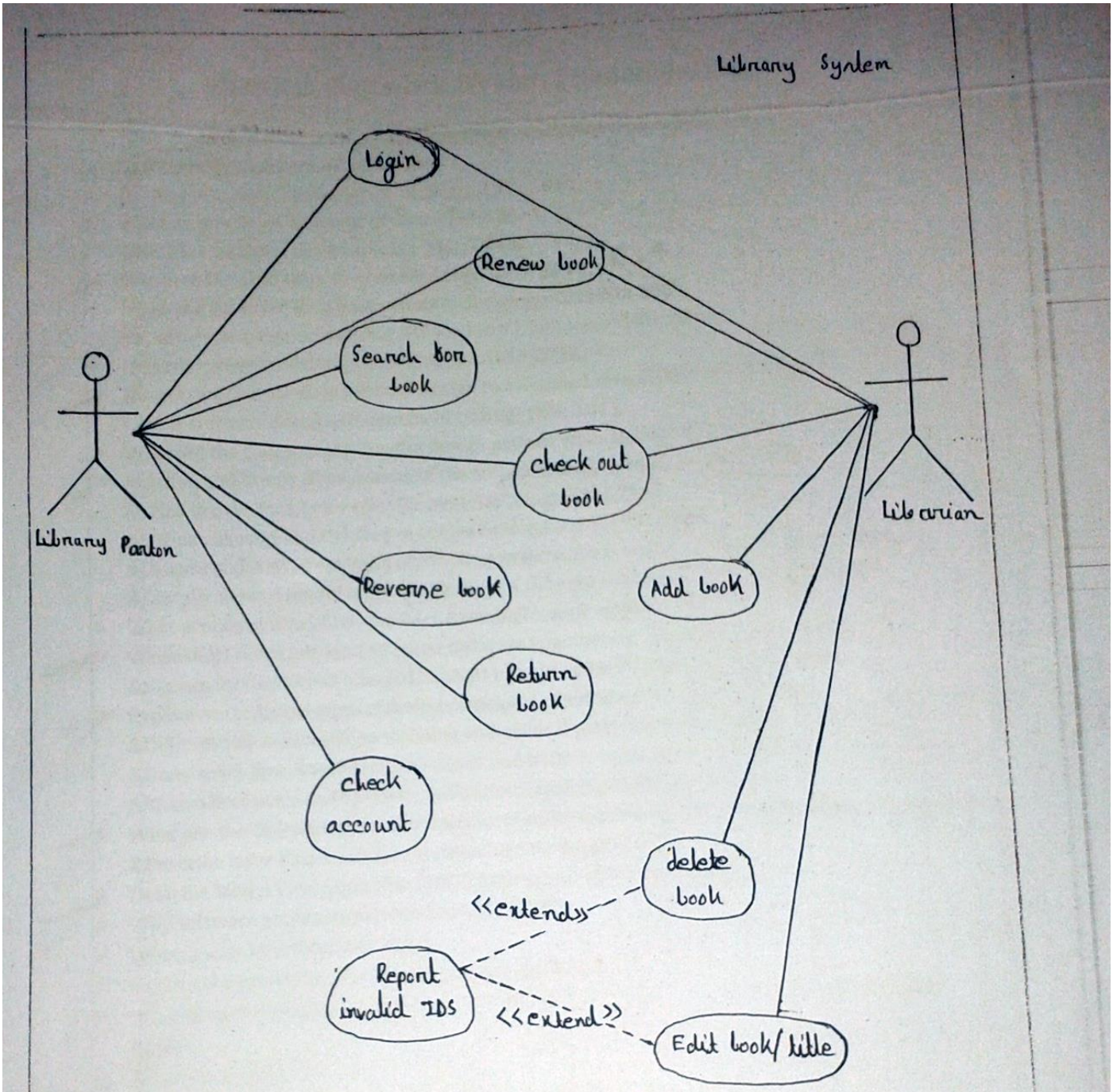
Answer: Use case diagram for library management system (UML):

A use case diagram is ideal for representing user interactions with a system. It can also help we identify and define the requirement of a system. To better understand use cases, try out the template below.



Use case diagram for library management system:

Even in this age of high-powered computers, an old-fashioned library has its place. To find information, a patron and librarian must work together to narrow search parameters and identify relevant resources. In, UML, the process of checking out a book can be represent actors and other essential entities. To create a use case diagram,



Use case diagrams are simple to make and read. The unified modeling language, or UML provides a useful modeling tool with which to begin our diagram. Use it to represent the goal of our actors and systems, or to express the whole scope of the system. We can choose exactly who can see or edit the diagram and we even offer text and video chat to streamline communication. For inspiration, check out the chart above - use case diagram for library management system (UML) - or choose another customizable template

Chapter:::Software Design Fundamentals.

- State the definition of software architecture and software design.[NU: 2010,
- Explain design process with diagram.
  Or, Describe logical construction of design.[NU: 2011,
- What do you mean by design process? [NU: 2011,
  Or, Define software design process. [NU: 2014,
- What are the objectives of software design?[NU: 2013, 2, a
- Write the guideline for a good software design process.[NU: 2009,
  Or, What are the guidelines that will lead to a good design?[NU: 2012,
- Describe procedural design with diagram. [NU: 2011,
- How do we transform an informal design to a detailed design?[NU: 2013, 2, a
- How is software design different from coding? [NU: 2013,
- What are the concepts of following design pattern when coding? [NU: 2009,
- Mention and briefly discuss some of the software design principles.[NU: 2010,
  &What are the factors for effective modular design?[NU: 2014,
  &Define cohesive and coupling in context of modularity.[NU: 2009,
  &Discuss about different types of coupling in the context of s/w design.[NU: 2012, 2014,
  &Discuss about different types of cohesion in the context of software design.[NU: 2013,
- What problems arise if two modules have high coupling?[NU: 2013,
- Write short notes onObject oriented software engineering. [NU: 2009, 2012, 2013,
  &Explain various object oriented concept used in software engineering.[NU: 2014,
- Explain the different types of design principles of software.[NU: 2012, 2014,
  &Differentiate horizontal partitioning and vertical partitioning.[NU: 2012,
  &Draw work flow diagram for top-down and bottom-up design.[NU: 2011,
  &Write short notes onTop-down and bottom-up design.[NU: 2009, 2012,
- What are the different types of architectural styles exist for s/w design?[NU: 2012,
  &Describe layer based software architecture model.[NU: 2009,
- Describe Model View Controller (MVC) architecture.[NU: 2009, 2012,
- Why software architecture is needed? [NU: 2009,
- What is user interface?[NU: 2012,
- Explain the principles of user interface design.[NU: 2012,
- What factors should be considered when designing interface for human software user?[NU: 2009,
- Write short notes on:
  - ✓ Data Flow Diagram (DFD).[NU: 2011, 2012,
  - ✓ Real-time design.[NU: 2011,
  - ✓ The design implementation of large multi-modules programs systems.[NU: 2009,

Question: State the definition of software architecture and design. ENU: 2010

Answer: Software Architecture: Architecture serves as a blueprint for a system. It provides an abstraction to manage the system complexity and establish a communication and co-ordination mechanism among components. It defines a structured solution to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security. These decision comprise of:

- Selection of structural elements and their interfaces by which the system is composed.
- Behavior as specified in collaborations among those elements.
- Composition of these structural and behavioral elements into large subsystem.
- Architectural decisions align with business objectives.
- Architectural styles guide the organization.

Software design: Software design provides a design plan that describes the element of a system, how they fit and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows-

✓ To negotiate system requirements and to set expectations with customer, marketing, and management personal.

✓ Act as a blueprint during the development process.

✓ Guide the implementation tasks, including detailed design, coding, integration and testing.

Question: Explain design process with diagram. [NU: 2011,
as a logical construction of design

Answer: Design process: The design process involves developing several models of the system at different levels of abstraction. As a design is decomposed, errors and omissions in earlier stages are discovered. These feed back to allow earlier design models to be improved. Figure shows the model of design process:

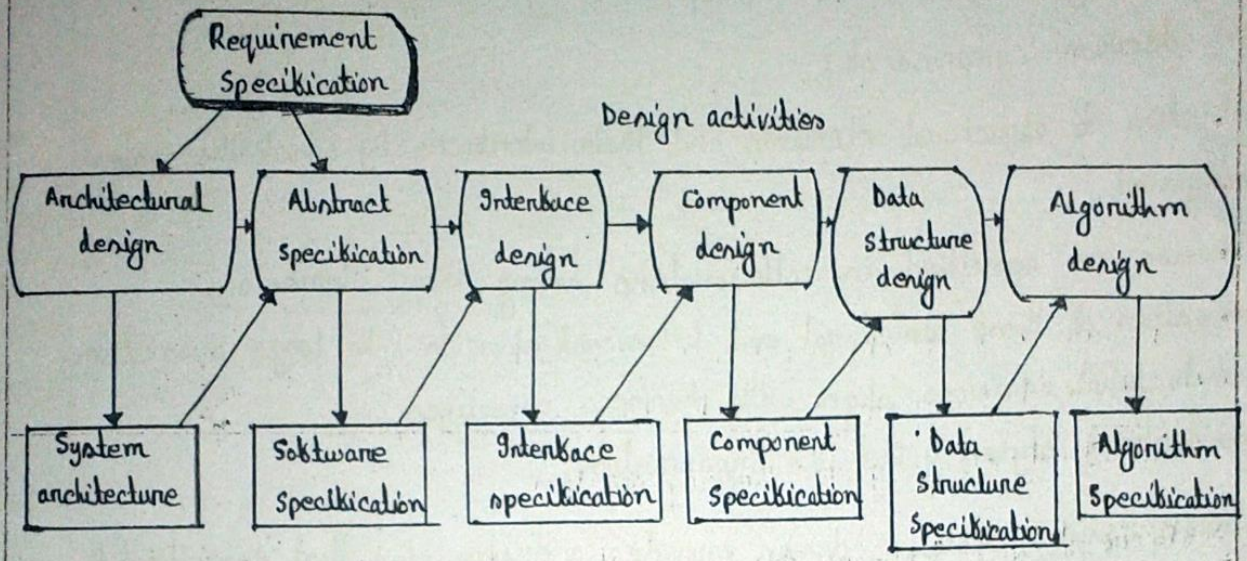


Fig: A general model of design process.

The specific design process activities are:

Architectural design: The sub-systems making up the system and their relationships are identified and documented.

Abstract specification: For each sub-system, its interface with other sub-system is designed and documented. This interface specification must be unambiguous as it allows the sub-system to be used without knowledge of the sub-system operation.

Component design: Services are allocated to different components and the interfaces of these components are designed.

Data structure design: The data structures used in the system implementation are designed in detail and specified.

Algorithm design: The algorithms used to provide services are designed in detail and specified.

Question: Define software design process [NU: 2011, 2014]

Answer: Software design Process: Software design is an iterative process through which requirements are translated into a 'blueprint' for constructing the software. The blueprint depicts a holistic view of software. The design is represented at a high level of abstraction a level that can be directly traced to the specific system objective and more detailed data, functional and behavioral requirements.

Question: What are the objectives of software design? [NU: 2013]

Answer: The objectives of software design:

- The design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.
- The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.
- The design should provide a complete picture of the software addressing the data, function, and behavioral domains from an implementation perspective.

Each of these characteristics is actually a goal of the design process.

Question: Write the guideline for a good software design process. [NU: 2009, 2012,

Answer: The quality of a design representation, we must establish technical criteria for good design. For a good software design process, we present the following guidelines:

- A design should exhibit an architecture that
  - ✓ has been created using recognizable architectural styles or patterns.
  - ✓ is composed of components that exhibit good design characteristics.
  - ✓ can be implemented in an evolutionary fashion, thereby facilitating implementation and testing.
- A design should be modular, that is the software should be logically partitioned into elements or subsystems.
- A design should contain distinct representations of data, architecture, interfaces and components.
- A design should lead a data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
- A design should lead to components that exhibit independent functional characteristics.
- A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
- A design should be derived using a repeatable method is driven by information obtained during software requirements analysis.
- A design should be represented using a notation that effectively communicates its meaning.

Question: Describe procedural design with diagram. ENU: 2022

Answer: A design methodology combines a systematic set of rules for creating a program design with diagramming tools needed to represent it. Procedural design is best used to model programs that have an obvious flow of data from input to output. It represents the architecture of a program as a set of interfacing processes that pass data from one to another.

Design Tools:

The two major diagramming tools used in procedural design are data flow diagrams and structure charts.

Data flow diagram: A data flow diagram is a tool to help for discover and document program's major processes. The following table shows the symbols used and what each represents.

Data Flow Diagram Symbols.

Name	Gane & Sarson Symbol	Yourdon Symbol	Description
Process	# Process Goal	# process Goal	A major task that the program must perform.
Data Flow	Name →	Name →	Data that flow into and out of each process
Data Store	ID Description	Name	An internal data structure that holds data during processing
External Entity	Name	Name	Device or humans which input data and to which data is output

The DFD is a conceptual model - it doesn't represent the computer program, it represents what the program must accomplish. By showing the input and output of each major task, it shows how data must have through and be transformed by the program

Structure charts:

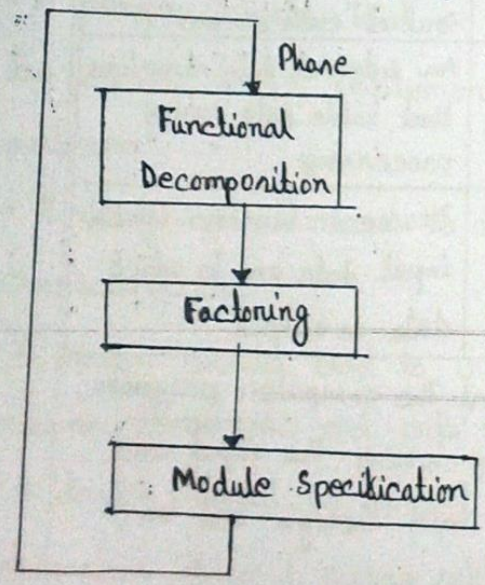
A structure chart is a tool to help we derive and document the program's architecture. It is similar to an organization chart

Structure Chart Symbols	
Symbol	Description
Component	A major component within the program.
	connects a parent component to one of its children
Name →	Data that is passed between components

When a component is divided into separate pieces, it called the parent and its piece are called its children. The structure chart shows the hierarchy between a parent and its children.

Design Methodology:

Here are the basic steps you follow to create a procedural design.



Functional Decomposition: In computer programming, decomposition is the process of dividing a large entity into more manageable pieces. For a procedural design, this means dividing tasks into sequences of smaller tasks which is functional decomposition.

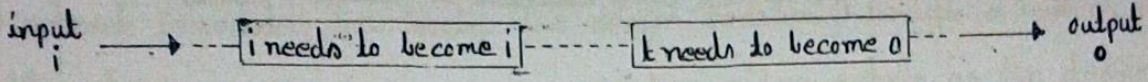
One technique for doing this, called data flow analysis, involves

1) identifying a major data flow.
2) following it from input to output
3) determining where it undergoes a major transformation

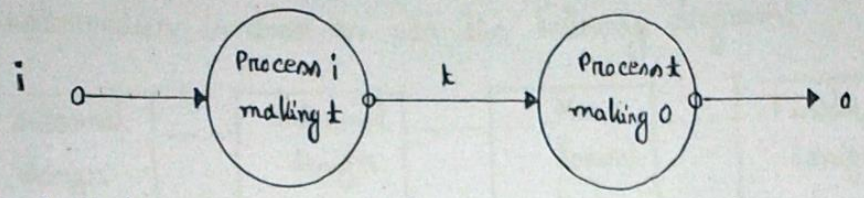
4) dividing the processing at that point.

To illustrate, give the following program requirements:

Program requirements



Data flow analysis yields:



Factoring: Factoring is the second phase of procedural design in which we create a structure chart that shows that program components need to be implemented. Do this two passes, First, arrange our DFD hierarchically. Second, identify exactly which conceptual processes are to be implemented as physical components in the program.

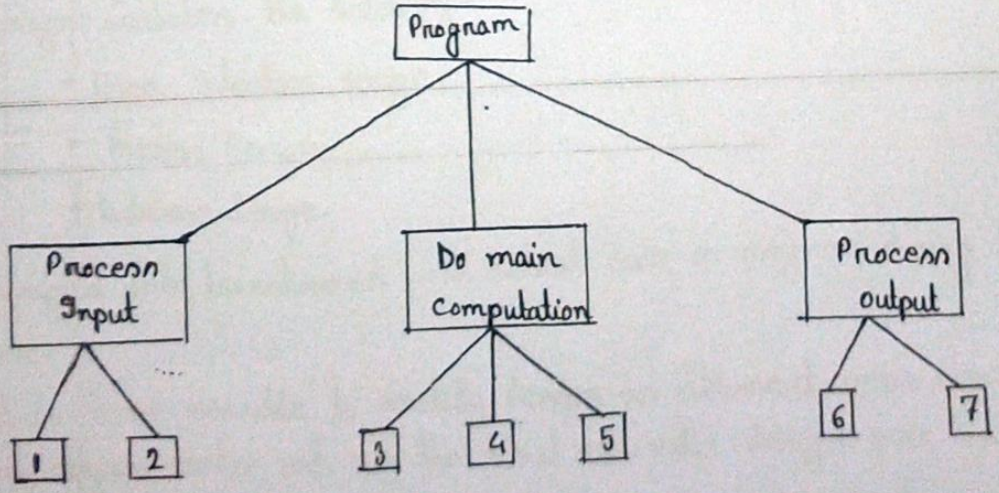
To arrange our DFD hierarchically, cut it into three partitions:

a) processes that prepare input for the main computation.

b) processes that perform the main computation.

c) processes that prepare the output.

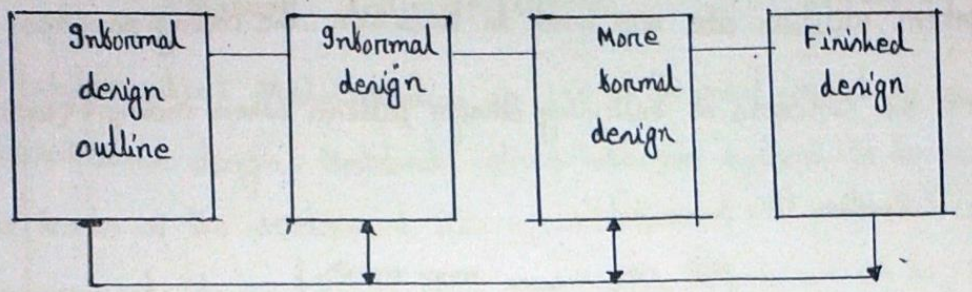
Organize the processes under one or more "managing" processes that control the flow of the computation. The above DFD becomes:



Module Specification: Module specification is the act of documenting our program design by fully describing each of its modules. Module is a general term that can refer to any manner of computer program components, including a single method, a single class, a single object or a collection of related methods.

Question: How do we transform an informal design to a detailed design? ENU: 2013,

Answer: Non-formal methods of specification can lead to problem during coding, particularly if the coder is a different person from the designer that is often the case. Software designers do not arrive at a finished design document immediately but develop the design iteratively through a number of different phases. The design process involves adding details as the design is developed with constant backtracking to correct earlier, less formal, designs. The transformation is done as per the following diagram:



Question: How is software design different from coding? ENU: 2013,

Answer: Points of difference between software design and coding can be laid down as under:

Design:

✓ Design is most crucial and time-consuming activity.

✓ Screen of the system depends on the correct design specifications which is a key activity of the process.

✓ Software design is based on the findings collected in the initial investigation phase.

✓ Design includes the following-

- User interface design.
- Process Design.
- Database design.

✓ Designs are transformed into actual code or program during the implementation phase.

✓ It is more feasible to rectify design as different users may have conflicting user requirements and only the final and valid design goes for next phase.

coding:

✓ Involves conversion of detailed design specification laid out by designers into actual code, files or database.

✓ Less time consuming then the design phase and performed by programmers or coders.

✓ More concerned with technical aspect of the software rather than its functional aspect.

✓ Different software such as programming language front-end tools, database management system, utilities etc are used to facilitate the coding process.

Question: What are the concepts of following design pattern when coding? [NU: 2009]

Answer:

✗ Question টা Incomplete

তাই কোন Answer পাওয়া যায় নাই।

Question: Mention and briefly discuss some of the software design principles

Answer: The software design principles:

- Modularization
- Abstraction
- Encapsulation
- Coupling and cohesion
- Separation of interface and implementation
- Sufficiency and completeness.

> Ques: What factor for effective modular design? [NU: 2010,
>
> Answer:
>
> এখানে Ans হবে মূলত কয়েকটি design process, যেগুলো তুমি Question এর Ans হবে।

Modularization: Modularization is one of the most important design principles in software design. Modularity allows software systems to be manageable at all levels of the development like cycle. That is, the work products of the requirements, design, construction and testing efforts can all be modularized to efficiently carry out the operations. In the design phase, modularization is the principle that drives the continuous decomposition of the software system until fine-grained components are created. Modularization plays a key role during all design activities, including software architecture and detailed and construction design; when applied effectively, it provides a roadmap for software development starting from coarse-grained components that are further modularized into fine-grained components directly related to code.

Abstraction: While the principle of modularization specifies what needs to be done, the principle of abstraction provides the guidance as to how it should be done. Abstraction is the principle that deals with creating conceptual entities required to facilities problem solving by focusing on essential characteristic of entities in their active content - while deferring unnecessary details. Abstraction is used to facilitate problem-solving by defining details to later stages. The principle of abstraction can be

classified as:

- ✓ Procedural abstraction
- ✓ Data abstraction.

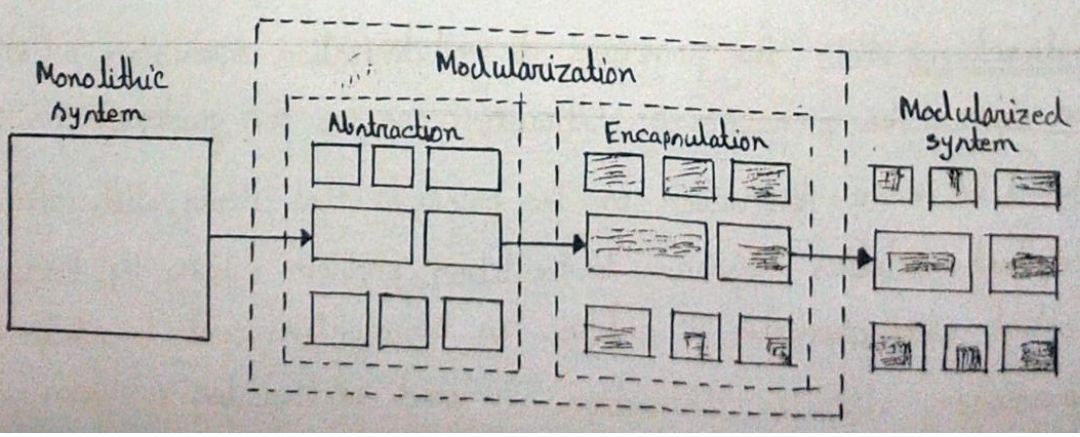
Procedural abstraction: Procedural abstraction is a specific type of abstraction that simplifies behavioral operations containing a sequence of steps of other procedural abstraction.

Example: Consider a client-server application in which the client sends data to the server through the Internet. In this case, the send procedural abstraction can be used to denote a series of operations.

Data abstraction: Data abstraction is used to simplify the structural composition of data objects.

Example: The Message data abstraction can be used to represent various messages with different attributes, such as the message's ID, content, and format.

- Encapsulation: Encapsulation is the principle that deals with providing access to the services of conceptual entities by exposing only the information that is essential to carry out such services while hiding details of how the services are carried out. The relationship among modularization, abstraction and encapsulation is presented in figure:



Question: Discuss about different types a coupling in the context of S/W design. [NU: 2012, 2014]

13 Coupling: The module's connections to other module are called coupling. That is, module coupling refers to the number of connections between a "calling" & a "called" module & the complexity of these connections. There must be at least one connection between a module & a calling module.

Example: In fig (a); - O, P & Q are couples. Module A calls module B passing O downward and receiving Q back.

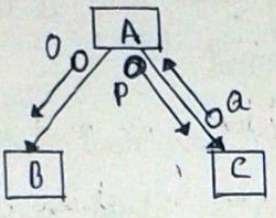


Fig: (a)

Myers has classified seven level of coupling:

- ✓ No direct coupling (best)
- ✓ Data coupling.
- ✓ Stamp coupling
- ✓ Control coupling
- ✓ External coupling
- ✓ Common coupling
- ✓ Content coupling (worst)

No direct coupling: No-direct coupling occurs when two modules do not in any way relate to each other but each independently relates only to the main program.

Data coupling: Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes

data structure as parameter, then the receiving module should use all its components.

Stam coupling: When multiple modules share common data structure and work on different part of it, it is called stamp coupling.

Control coupling: Two modules are called control-coupled if one of them decides the function of the other module or change its flow of execution.

External coupling: External coupling is possible in some languages where we can control the variable coupling and limit it to those variables which are formally declared as external.

Common coupling: When multiple modules have read and write access to some global data, it is called common or global coupling.

Content coupling: When a module can directly access or modify or refer to the content of another module, it is called content level coupling.

⊡ Cohension: Cohension means strength within module. That is, degree of relationships between elements within module. Module that perform only one tasks are said to be more cohesive (& less error prone) than modules that perform multiple tasks.

There are seven types of cohension, namely:

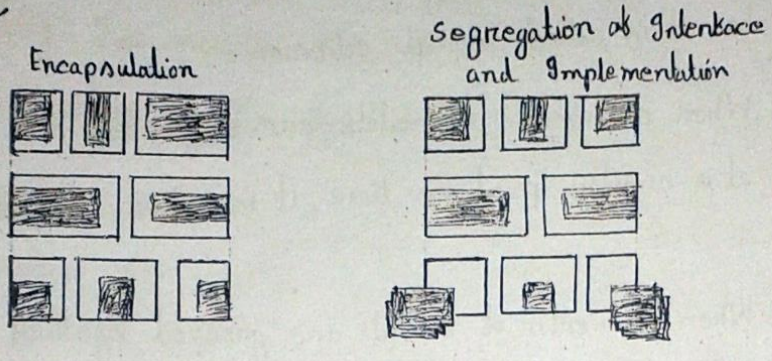
- ✓ Con-incidial cohension.
- ✓ Logical cohension.
- ✓ Temporal cohension
- ✓ Procedural cohension
- ✓ Communicational cohension
- ✓ Sequential cohension
- ✓ Functional Cohension

- Co-incidental cohesion: It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.
- Logical cohesion: When logically categorized elements are put together into a module, it is called logical cohesion.
- Temporal cohesion: When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.
- Procedural cohesion: When elements of module are grouped together, which are expected sequentially in order to perform a task, it is called procedural cohesion.
- Communicational cohesion: When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.
- Sequential cohesion: When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.
- Functional cohesion: It is considered to be the highest degree of cohesion and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

Question: Define cohesive and coupling in context of modularity. [NU: 2009,

Question: Discuss about different types of cohesion in the context of software design. [NU: 2013,

□ Separation of Interface and Implementation: The principle of separation of interface and implementation deals with creating modules in such way that a stable interface is identified and separated from its implementation. This design principle should not be confused with encapsulation. During encapsulation,



interfaces are created to provide public access to service provided by the design unit while hiding unnecessary details, which include implelation. While encapsulation dictates hiding the details of implementation, the principle of separation dictates their separation, so that different implementation of the same interface can be swapped to provide modified or new behavior.

□ Completeness and sufficiency: The principles of completeness and sufficiency deal with efficient module creation. Completeness is a characteristic that measures how well design units provide the requid services to achieve their intent. On the other hand, sufficiency measures how well design units are at providing only the services that are sufficient for achieving their intent. Consider the same communication class, which can include services for logging statistics, visualization of network activity, or any other capability applicable to the communication task.

Question: What problems arise if two modules have high coupling? [NU: 2013,

Answer: Coupling means the interconnection of different modules with each other or we can say, it tells about the interrelationship of different modules of a system. A system with high coupling means there are strong interconnections between its modules. If two modules are involved in high coupling, it means their interdependence will be very high. Any changes applied to one module will affect the functionality of the other module. Greater the degree of change, greater will be its effect on the other. As the dependence is higher, such change will affect modules in a negative manner and in-turn, the maintainability of the project is reduced. This will further reduce the reusability factor of individual modules and hence lead to unsophisticated software. So, it is always desirable to have inter-connection and interdependence between modules.

Question: Explain various object oriented concept used in software engineering. [NU: 2009, 2012, 2013, 2014,

Answer: Various concepts of object oriented concept used in software engineering:

✓ Object: An object is something which is capable of being seen, touched or sensed. Each object has certain distinctions or attributes which enable we to recognize and classify it. Each object has certain actions or methods associated with it.

✓ Class: A class encapsulates data and procedural abstractions required to describe the content and behavior of some real world entity. A class is a generalized description that describes a collection of similar objects.

✓ Encapsulation: An object is said to be encapsulate (hide) data and program. This means that the user cannot see the inside of the object but can use the objects by calling the program part of the object. This hiding of details of one object from another object, which uses it, is known as encapsulation.

✓ Inheritance: Inheritance is defined as the property of objects by which instances of a class can have access to data and programs contained in a previously defined class, without those definitions being restated. Classes

are linked together in a hierarchy. They from a tree, whose root is the class of "objects". Each class will have a super class and possibly subclasses. A classes can inherit methods from its supper class and in turn can pass methods on to its subclasses. A subclass must have all the properties of the parent class, and other properties as well.

✓ Polymorphism: Polymorphism includes the ability to use the same message to objects to different classes and have them behave differently. Thus we could define the message "+" for both the addition of numbers and the concatenation (joining) of characters. Polymorphism provides the ability to use the same word to invoke different methods, according to similarity of meaning.

✓ Object/class associations: object/classes interact with each other. Multiplicity defines how many instances of one object/class can be associated with one instance of another object/class.

✓ Messages: The interaction or communication between the different objects and classes is done by passing message. The object, which requires communicating with another object, will send a request message to the latter. A message can be sent between two objects only if they have an association between them.

Question: Explain the different types of design principles of software. [NU: 2012, 2014]

Answer: Three types of design principles are as follow:-

- Problem partitioning.
- Abstraction
- Top-down and Bottom-up design. [NU: 2012]

☐ Problem partitioning: When solving a small problem, the entire problem can be tackled at once. For solving larger problems, the basic principle is the time-tested principle of "divide and conquer". This principle suggests dividing into smaller pieces, so that each piece can be conquered separately.

Problem partitioning can be divided into two categories:

(i) Horizontal partitioning.

(ii) Vertical partitioning.

✓ Horizontal partitioning: Horizontal partitioning define separate branches of modular hierarchy for each major program function. The simplest approach to horizontal partitionary defines three partitions: input, data transformation and output. Partitioning their architecture horizontally provides a number of distinct benefits:

- Software that is easier to test.
- Software that is easier to maintain.
- Software that is easier to extend.
- Propagation of fewer side effects.

Conversely, horizontal partitioning often causes more data to be passed across module interfaces and can complicate the overall control of program flow.

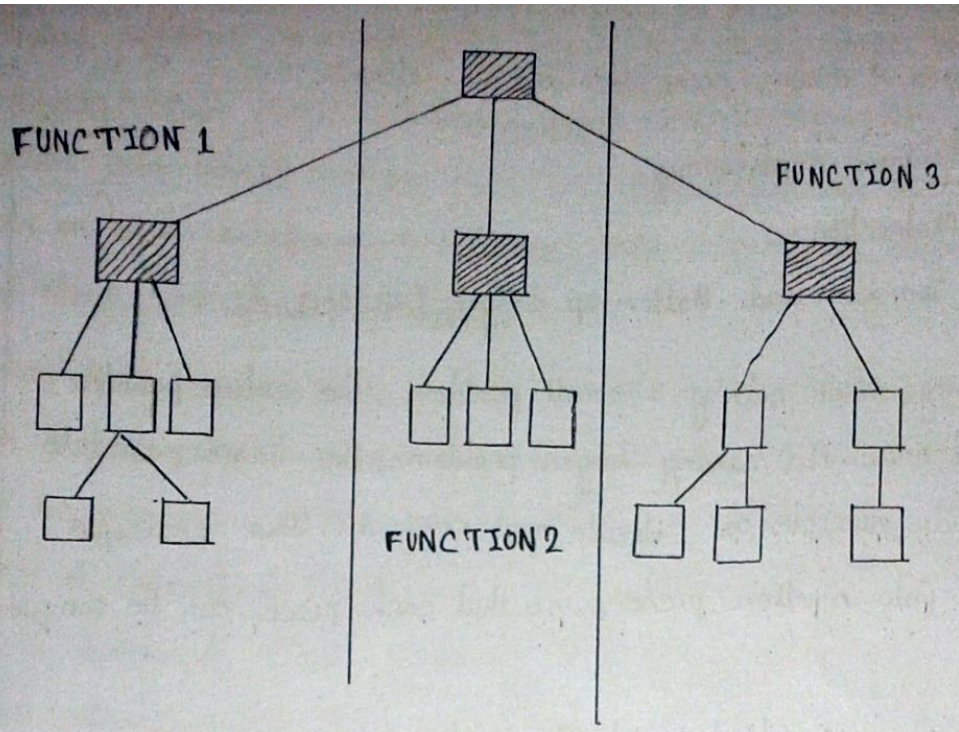


Figure: Horizontal Partitioning.

✓ Vertical partitioning: Vertical partitioning, often called factoring, suggests that control and work should be distributed from top-down in the program structure. Top-level modules should perform control functions and do actual processing work. Modules that reside low in the structure should be the workers, performing all input, compilation and output tasks.

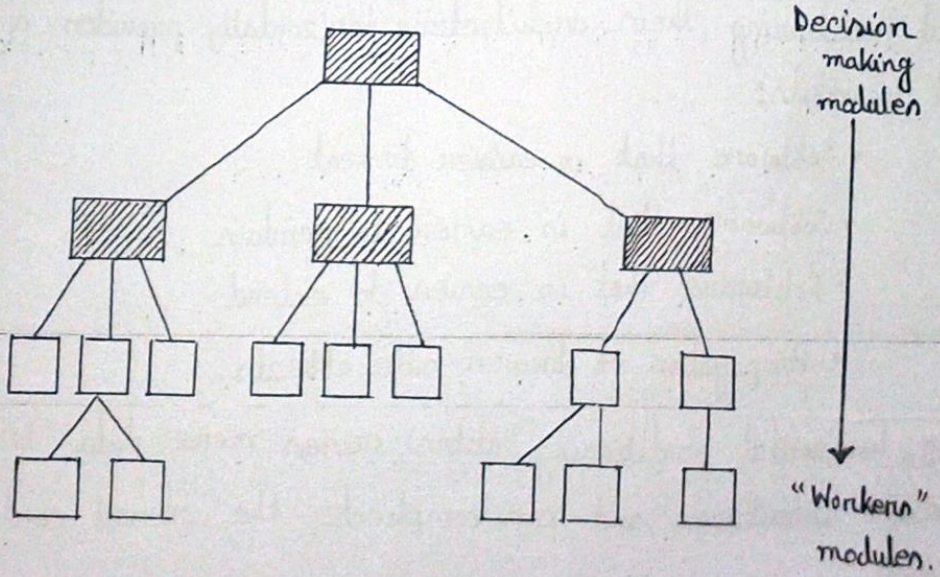


Figure: Vertical partitioning.

Question: Differentiate horizontal and vertical partitioning. [NU:2012]

Abstraction: An abstraction of a component describe the external behavior of that component without bothering with the internal details that produce the behavior. Abstraction is an indispensable part of the design process and it is essential for problem partitioning. Partitioning essentially is the exercise in determining the components of a system.

Abstraction is used for existing components as well as components that are being designed. Abstraction of existing components plays an important role in the maintenance phase.

There are two common abstraction mechanism for software systems: functional abstraction and data abstraction. In functional abstraction, a module is specified by the function it performs. When the problem is being partitioned, the overall transformation function for the system is partitioned into smaller functions that comprise the system function. The second unit for abstraction is data abstraction. There are certain operations required from a data object, depending on the object and the environment in which it is used.

Top-down and Bottom-up Design: [NU: 2011, 2009, 2012,

Top-down Design: Top-down design is basically a decomposition process, which focuses on the flow of control or on the control structure of the program. The first step is to study the overall aspects of the task & break it into a number of modules. Then break this independent module, which are small enough to divide.

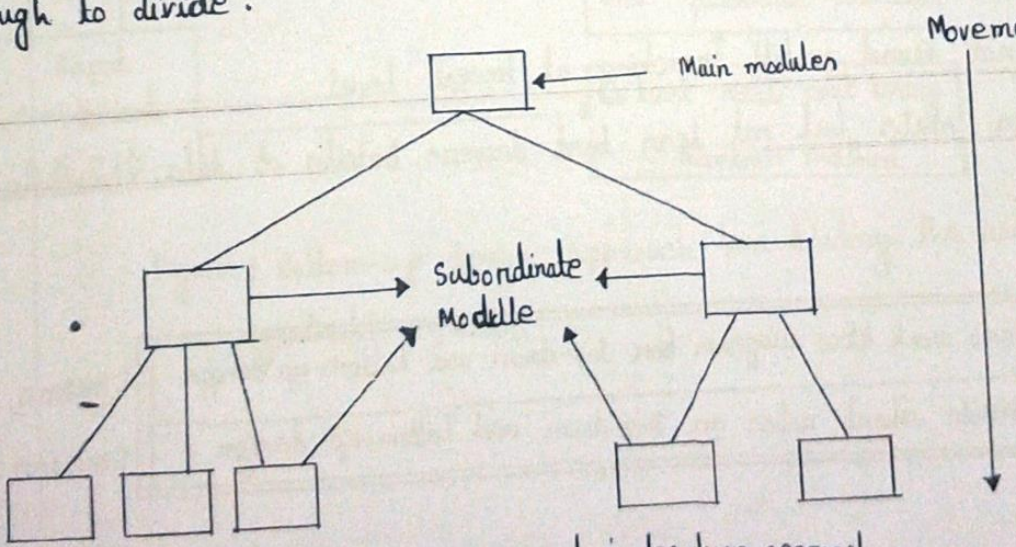


Figure: Movement in top-down approach.

One important feature of top-down design is that at each level the details of the design at lower levels are hidden. Only the necessary data and control, which must be passed back and forth over the interface are defined.

Block diagram:

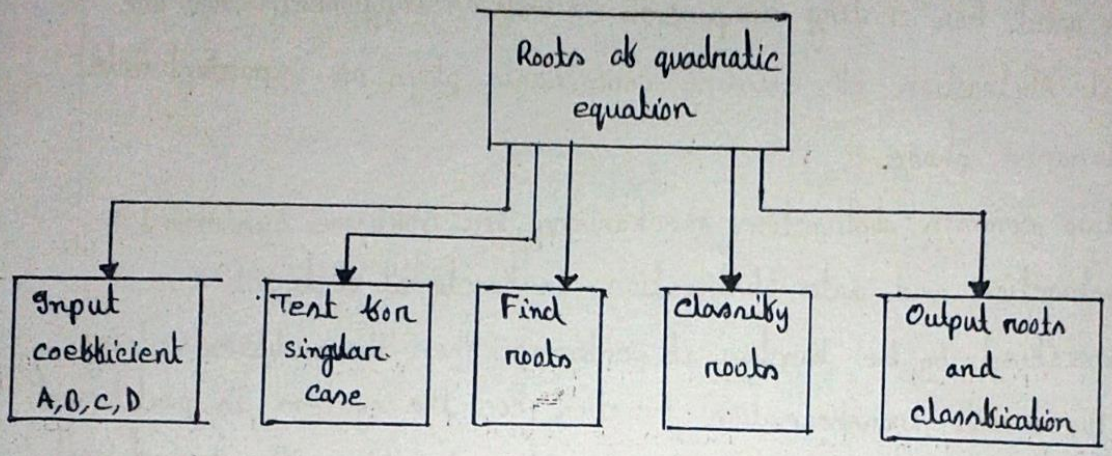


Figure: Top-down design approach for finding the roots of a quadratic equation.

Advantages:

- Additional features easy to include because they generally represent added modules o redesign of a new modules.
- Usually no serious surprise occur to interrupt the integration progresses because phase, because the interfaces already designed.

Disadvantages:

- Requires well-defined specification & definition of environment from outset
- Error are not found untill reaching at lowest level.
- Requires tests stubs but not less test drivers. Details of data structures may be delayed.

Question: Draw work flow diagram for top-down and bottom-up design.	[NU: 2011,
Question: Write short notes on Top-down and bottom-up design.	[NU: 2009, 2012,

Bottom-up design: In a bottom-up design one first visualizes a typical system design and decides by experience, intuition or perhaps simple, quick analysis which part or parts of the design are the most difficult or limiting ones.

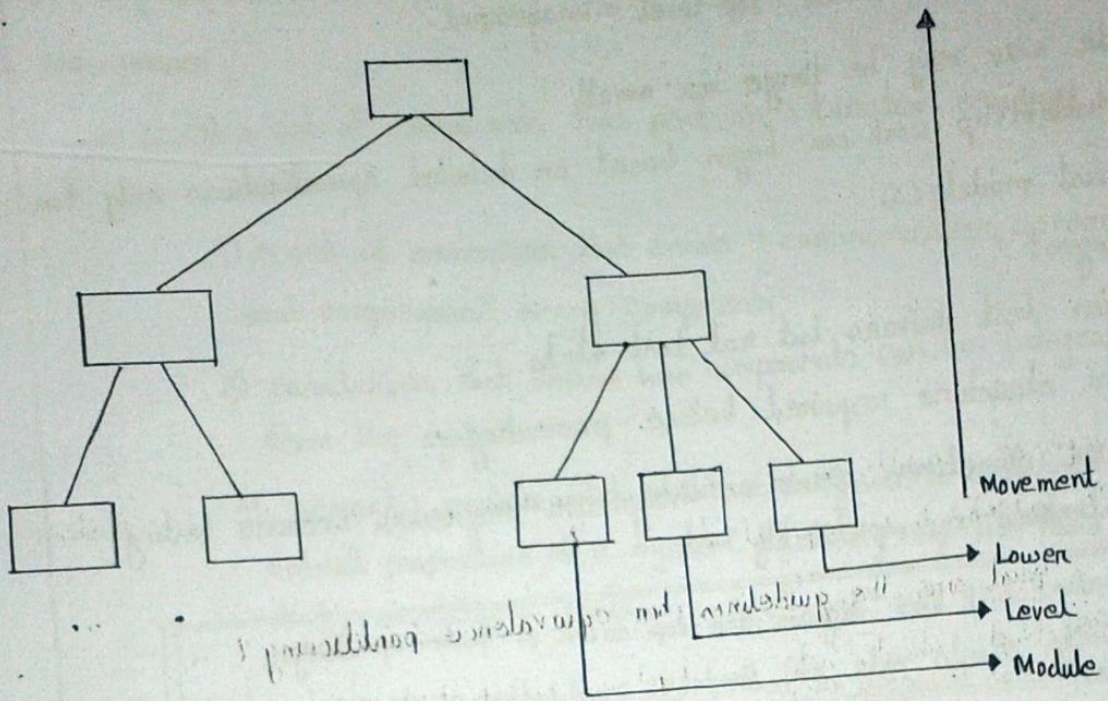


Figure: Movement in bottom-up approach.

Block diagram:

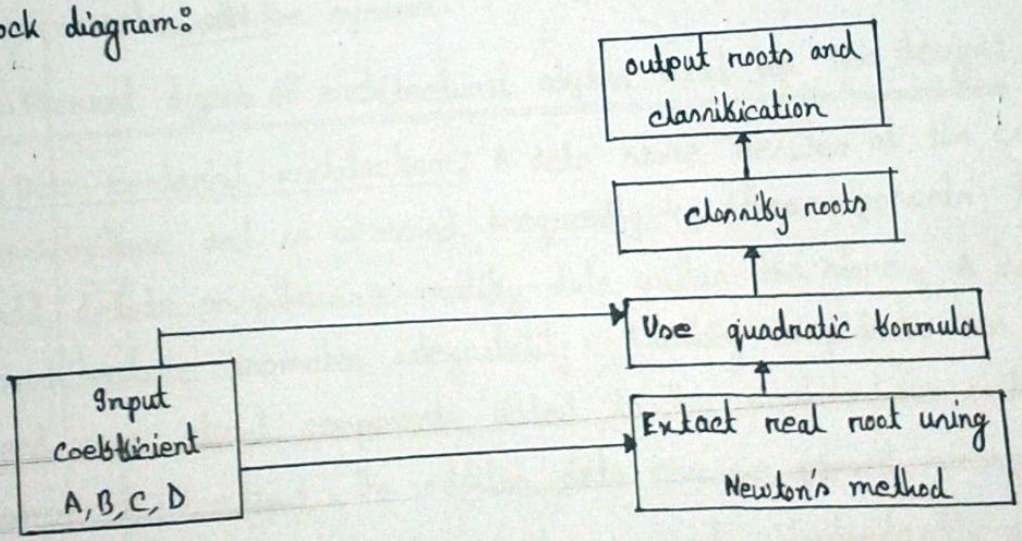


Figure: Bottom-up design approach for finding the roots of a quadratic equation.

Advantages:

- More tolerant of early error since they only represent redesign & recoding of a single module test.
- It is easy to convince top-level management.
- Module size may be large or small
- But bottom up work can begin based on detailed specifications only for the crucial module(s).

Disadvantages:

- Requires test drivers but not test stubs.
- All data structure required before proceeding.
- Surprises sometimes occur as integration progresses because testing reveals interface incompatibility.

Question: Draw work flow diagram for top-down & bottom up design.
Question: Write shorts note on topdown and bottom up design.

Khairul

Question: What are the different types of architectural styles exist for s/w design? [NU: 2012]

Answer: The software that is built for computer-based systems also exhibits one of many architectural styles. Each style describes a system category that encompasses,

1) a set of components that perform a function required by a system.
2) a set of connectors that enable "communication, coordination, and cooperation" among components.
3) constraints that define how components can be integrated to form the system.
4) semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

An architectural style is a transformation that is imposed on the design of an entire system. The intent is to establish a structure for all components of the system.

Different types of architectural styles exist for s/w design:

- Data-centered architecture: A data store resides at the center of this architecture and is accessed frequently by other components that update, add, delete or otherwise modify data within the store. A data-centered architecture promotes integrability. Existing components can be changed and new client components added to the architecture without concern about other client. In addition, data can be passed among clients using the blackboard mechanism. Client components independently execute processes.

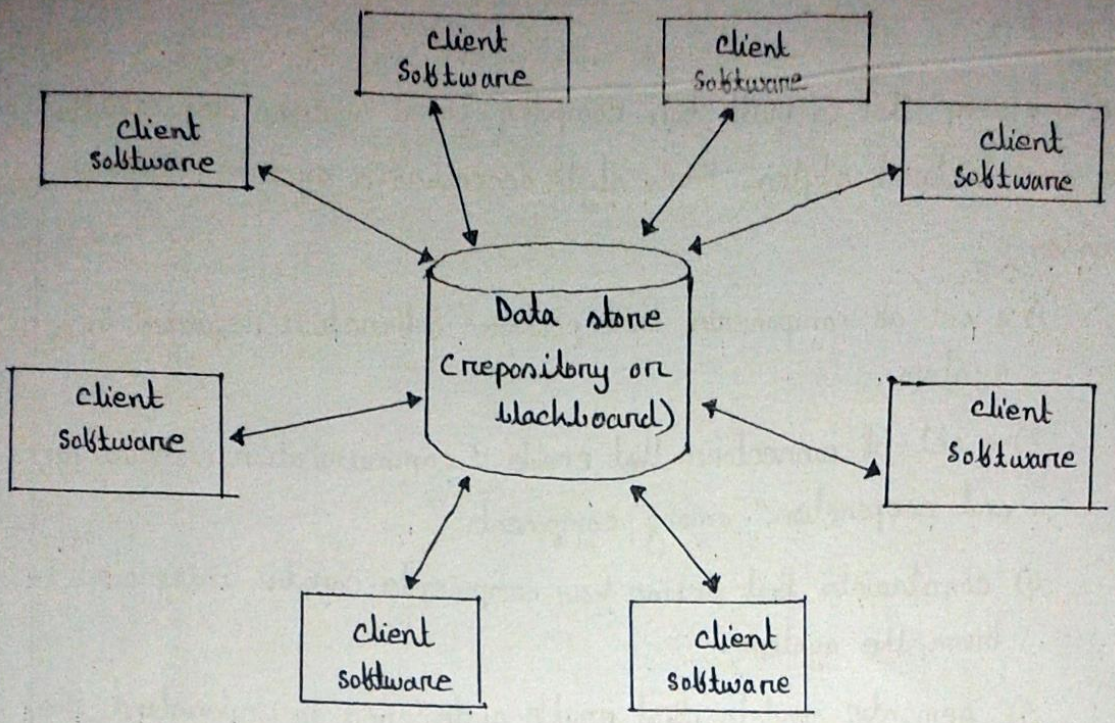


Fig: Data-centered architecture.

- Data-flow architecture: This architecture is applied when input data are to transformed through a series of computational or manipulative components into output data.

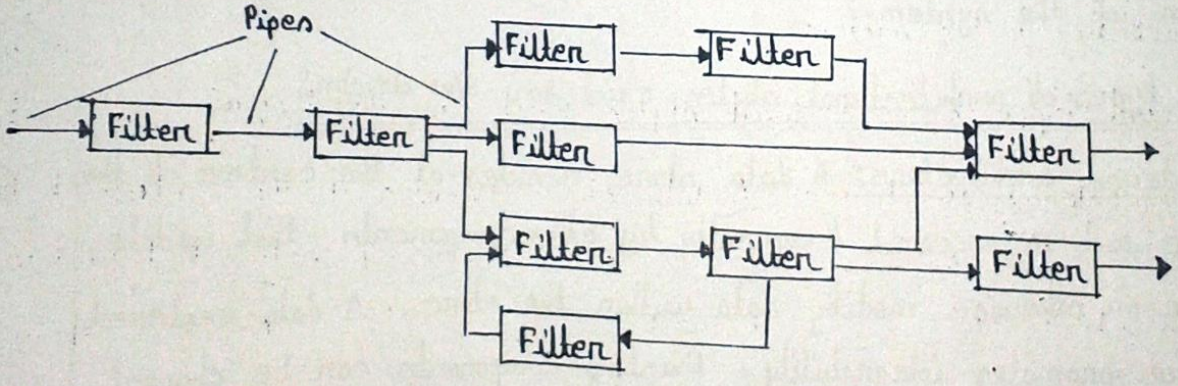


Fig: Pipes and filters.

A pipe and filters structure has a set of components, called filters, connected by pipes that transmit data from one component to the next. Each filter works independently of those components upstream and downstream, is designed to expect data input of a certain form, and produces data output of a specified form.

⊡ Call and return architecture: This architectural style enables a software designer to achieve a program structure that is relatively easy to modify and scale. Two substyles exist within this category:-

✓ Main program / subprogram architecture: This classic program structure decomposes function into a control hierarchy where a "main" program invokes a number a program components, which in turn may invoke still other components.

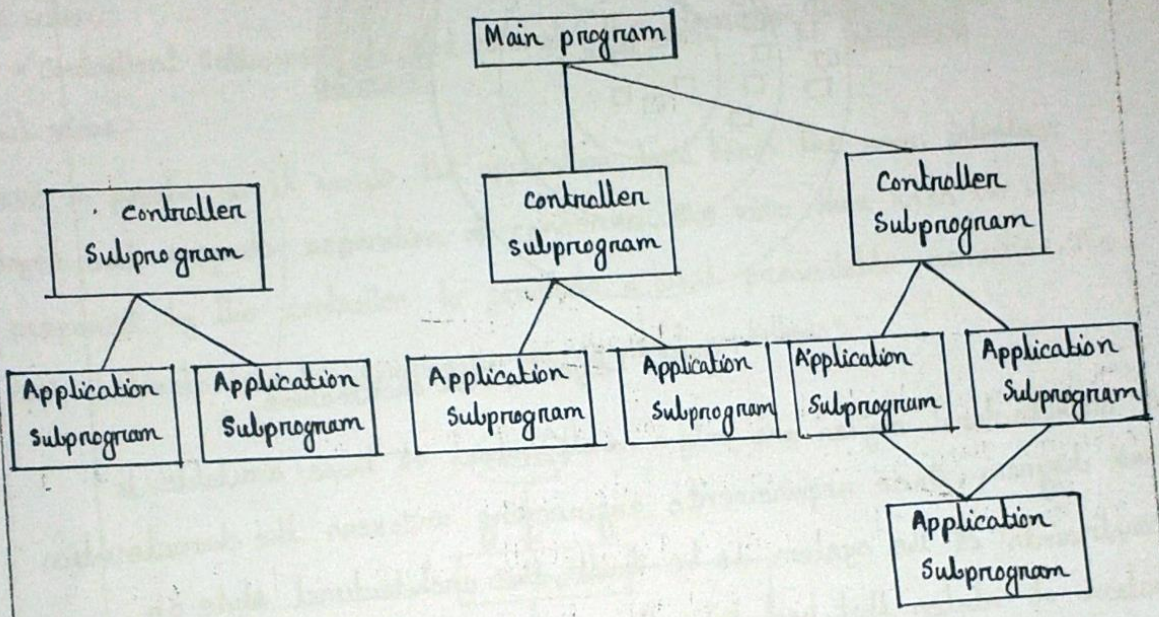


Figure: Main program

✓ Remote procedure call architecture: The components of a main program/ subprogram architecture are distributed across multiple computers on a network.

⊡ Object-oriented architecture: The components of a system encapsulate data and the operations that must be applied to manipulate the data. communication and coordination between components is accomplished via message passing.

⊡ Layered based architecture: The basic structure of a layered architecture in figure. A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set. At the outer layer, components service user interface operations. At the inner layer, components

perform operating system interfacing. Intermediate layers provides utility services and application software functions.

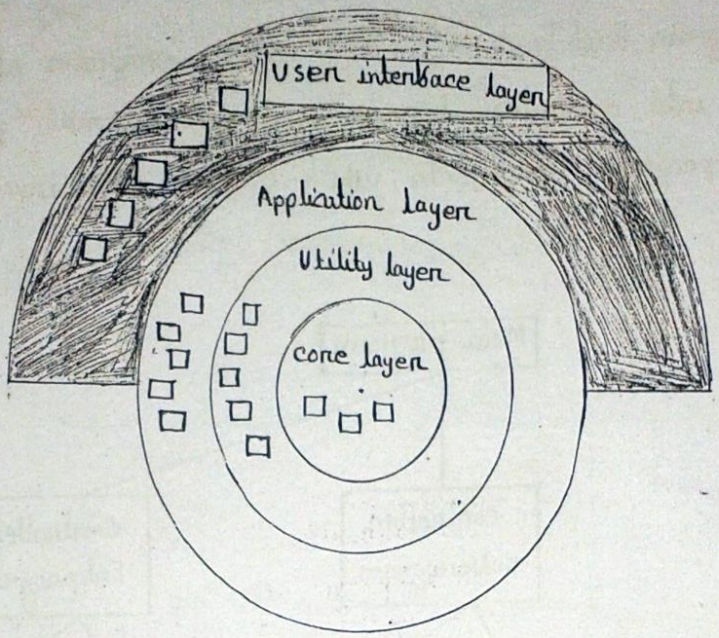


Figure: Layered-base architecture

These architectural styles are only a small subset of those available to software diagram. Once requirements engineering uncovers the characteristics and constraints of the system to be built, the architectural style or combination of styles that best fits those characteristics and constraints can be chosen. In many cases, more than one style might be appropriate and alternatives might be designed and evaluated.

question: Describe layer based software architecture model. [NU: 2009]

Khairul

Question: Describe Model view controller (MVC) architecture. [NU: 2009, 2012,

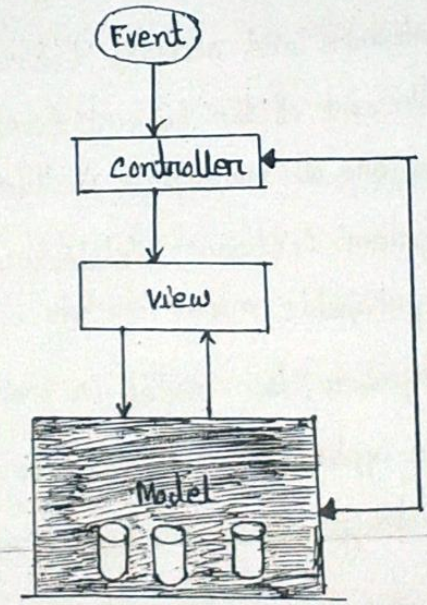
Answer: Model view controller or MVC as it is popularly called, is a software design pattern for developing web applications. A model view controller pattern is made up of the following three parts:

✓ Model: The lowest level of the pattern which is responsible for maintaining data.

✓ View: This is responsible for displaying all or a portion of the data to the user.

✓ Controller: Software code that controls the interaction between the model and view.

MVC is popular as it isolate the application logic from the user interface layer and supports separation of concerns. The view then uses the data prepared by the controller to generate a final presentable response. The MVC abstraction can be graphically represented as follows:



The model: The model is responsible for managing the data of the application. It respond to the request from the view and it also responds to instructions from the controller to update itself.

The view: A presentation of data in a particular format, triggered by a controller's decision to present the data. They are script based templating systems JSP, ASP, PHP and very easy to integrate with AJAX technology.

The controller: The controller is responsible for responding to user input and performs interactions on the data model objects. The controller receive the input, it validates the input and then performs the business operation that modifies the state of the data model.

question: Why software architecture is needed? [NU: 2019]

Answer: Software architecture is needed, fundamentally there are three reasons why software architecture is important to large, complex, software intensive systems:

(1) Communication among stakeholders. Software architecture represents a common high-level abstraction of a system that most if not all of the system's stakeholders can use as a basis for creating mutual understand.

(2) Early design decisions. The software architecture of a system is the earliest artifact that enables the priorities among competing concerns to be analyzed and it is the artifact that manifests the concerns as system qualities. The trade-offs between performance and security, between maintainability and reliability, and between the cost of the current development effort and the cost of future developments are all manifested in the architecture.

(3) Transable abstraction of a system. Software architecture constitutes a relatively small, intellectually graspable model for how a system is structured and how its components work together, this model is transferable across systems; in particular, it can be applied to other systems exhibiting similar requirements and can promote large-scale reuse.

Question: What is user interface? [NU: 2012,

Answer: User Interface: A user interface, also called a "UI" or simply an "interface" is the means in which a person controls a software application or hardware device. A good user interface provides a "user friendly" experience, allowing the user to interact with the software or hardware in a natural and intuitive way.

The user interface (UI) is everything designed into an information device with which a human being may interact- including display screen, keyboard, mouse, light pen, the appearance of a desktop, illuminated characters, help message and how an application program or a web site invites interaction and responds do it.

Question: Explain the principles of user interface design. [NU: 2012,

Answer: The general principles of the user interface design as follows:

☑ Aesthetically pleasing:

✓ A design is aesthetically pleasing if it is attractive to the eye. It draws attention subliminally, conveying a message clearly & quickly.

✓ Visual appeal is provided by following the presentation & graphic design principles which include meaningful contrast btween screen elements, creating spatial groupings, aligning screen element, providing three-dimensional representation & using color & graphics effectively.

☑ Clarity:

✓ User interface must be clear in visual appearance, concept & wording.

✓ Visual elements should be understandable & related to real world concepts and functions.

✓ Interface-words and text should be simple, unambiguous and free of computer jargon.

☑ Compatibility: Compatibility needs to be provided as -

• User Compatibility: "Know the user" is the fundamental principle in interface design as no users are alike & they think, feel & behave differently compared

to the developer.

✓ Task and Job compatibility: The structure and flow of functions should permit easy transition between task. The user must never be forced to navigate between applications on many screens to complete routine daily tasks.

✓ Product compatibility: Compatibility across products must always be considered in relation to improving interfaces, making new systems compatible with existing systems will take advantage of what users already know & reduce the necessity for new learning.

Comprehensibility:

✓ The steps to complete a task should be obvious. System should be understandable and flowing in meaningful order.

✓ A user should know what to look at, what to do, when to do it, where to do it, why to do it & how to do it.

Configurability:

✓ A default configuration as well as easy personalization & customization though configuration and reconfiguration should be provided.

✓ Customization enhances sense of control, encourages an active role in understanding and allows personal preference and difference in experience levels leading to high user satisfaction.

Consistency:

✓ Consistency is important because it can reduce requirements for human learning by allowing skills learned in one situation to be transferred to another like it.

✓ Any new system must impose some learning requirements on its users but avoid unnecessary activity.

Control:

✓ The user must control the interaction & never be interrupted for errors.

✓ Actions should result from explicit user requests & be performed quickly.

Directness:

✓ Tasks should be performed directly & alternative should be visible reducing the user's mental workload.

✓ Tasks are performed by directly selecting an object then selecting an action performed and then seeing the action being performed.

Efficiency:

✓ Transition between various systems controls should flow easily & freely.

✓ Navigation paths should be as short as possible.

✓ Eye movement through a screen should be obvious & sequential.

Familiarity:

✓ Build into the interface concepts, terminology, workflows & spatial arrangements already familiar to the user.

✓ Familiar concepts enable people to get started & become productive quickly.

Flexibility:

✓ Flexibility is the system's ability to respond to individual difference in people.

✓ Permitting system customization.

Forgiveness:

✓ People will make mistakes, a system should be able to tolerate those that are common & unavoidable.

✓ A forgiving system keeps people out of trouble.

Predictability:

✓ All actions should lead to results the user expects. Current operations should provide clues as to what will come next.

✓ Design consistency enhances predictability.

Recovery:

✓ A person should be able to retract any action by issuing an undo command.

✓ The goal is stability on returning easily to the right track when a wrong track has been taken.

✓ Recovery should be obvious, automatic, easy & natural to perform.

Responsiveness:

✓ A user must be responded quickly.

✓ Substantial or more informative feedback is most important for the casual or new system user.

✓ All requests must be acknowledged in some way.

Simplicity:

✓ Simplicity can be achieved by progressive disclosure, provide defaults, minimize screen alignment points, make common action points, make common actions simpler & provide uniformity & consistency.

Transparency:

✓ Permit the user to focus on the task or job without concerning the mechanics of the interface.

✓ Working and reminders of working inside the computer should be invisible to the user.

Trade-offs:

✓ Final design will be based on a series of trade-offs balancing often-conflicting design principle.

✓ People's requirements always take precedence over technical requirements.

Question: What factors should be considered when designing interface for human software user?

Answer: The factors that should be considered in designing interface of a software are:

Strive for consistancy: Consistent sequence of actions should be required in similar situations. Identical terminology should be used in prompts means and help screens.

Enable frequent users to use short-cuts: The users desire to reduce the number of interactions increases with the frequency of use.

Offer informative feedback: For every operator action, there should be some system feedback for frequent and minor actions, the response must be modest, while for infrequent and major actions, the response must be more substantial.

⊡ Design dialogs to yield closure: Sequences of actions should be organized into groups with a beginning, middle and end. The informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of relief the signal to drop contingency plans and options from their minds.

⊡ Offer simple error handling: As much as possible design the system so the user will not make a serious error. If an error is made, the system should be able to detect it and offer simple comprehensible mechanism for handling the error.

⊡ Permit easy reversal of actions: This feature relieves anxiety since the user knows that errors can be undone.

⊡ Support internal locus of control: Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions.

Question: Write shorts notes on:

a) Data flow Design. (DFD) [NU: 2011, 2012,

b) Real-time design.

c) The design implementation of large multi-modules programs systems.

Answer:

a) Data flow Diagram: Data flow diagram is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows through the system.

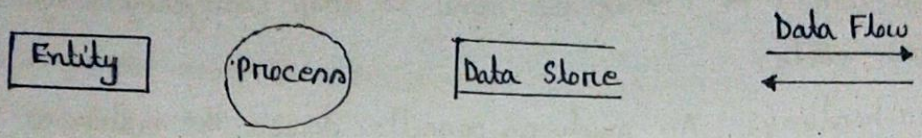
Types of DFD: Data flow diagrams are either Logical or Physical.

✓ Logical DFD: This type of DFD concentrates on the system process and flow of data in the system.

✓ Physical DFD: This type of DFD shows how the data flow is actually implemented in the system. It is more specific and close to the implementation.

DFD Components:

DFD can represent source, destination, storage and flow of data using the following set of components -



- Entities: Entities are source and destination of information data. Entities are represented by a rectangles with their respective names.
- Process: Activities and action taken on the data are represented by Circle or Round-edged rectangles.
- Data storage: There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.
- Data flow: Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as it source towards head of the arrow as destination.

Levels of DFD:

- Level 0 - Highest abstraction level DFD is known as level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known as context level DFDs.
- Level 1 - The level 0 DFD is broken down into more specific Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and source of information.
- Level 2 - At this level, DFD shows how data flows inside the modules mentioned in level 1.

Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achived.