Software Engineering.

UNIT - I :-

Introduction to Software Engineering : The evolving role of software, changing nature of Software, Legacy s/w, Software myths.

A Generic View of Process :-

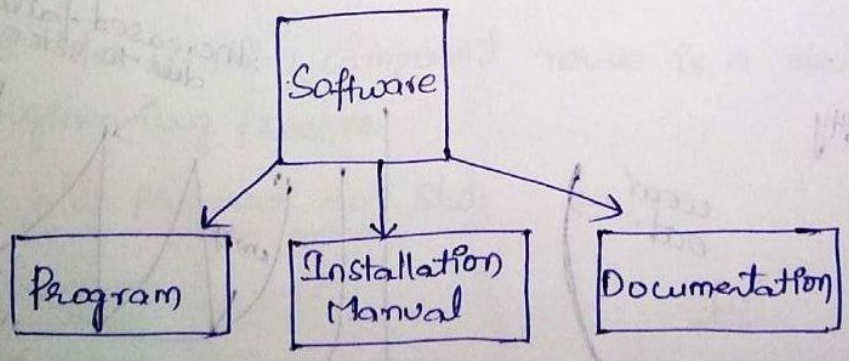
Software Engineering - A Layered technology, A process framework, The capability Maturity Model Integration (CMMI), Process patterns, Process Assessment, personal and team process models.

Process Models :-

The waterfall model, Incremental process models, Evolutionary Process models, specialized process models, The unified process.

Software :-

* Software is a broad term as it includes program, data structure, and documentation which are generated during different stages of life cycle of Software Development and instation files which are used in a unified manner to perform a particular computing task.



* computer software is the product that software professionals build and then support over the long term.

Scanned by CamScanner

[Q1] Charaiteristics of Software :-

[1] Software is developed or engineered ; it is not manufacitured in classical sense :

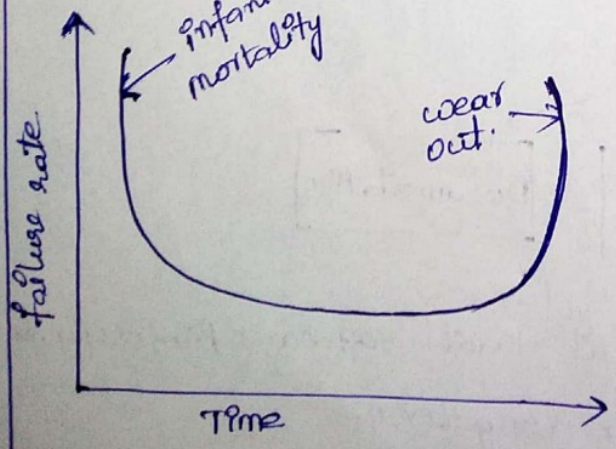
* Although some similarities exists blw software development and hardware manufacturing, but few activities are fundamentally different.

* In both activities high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems than software.

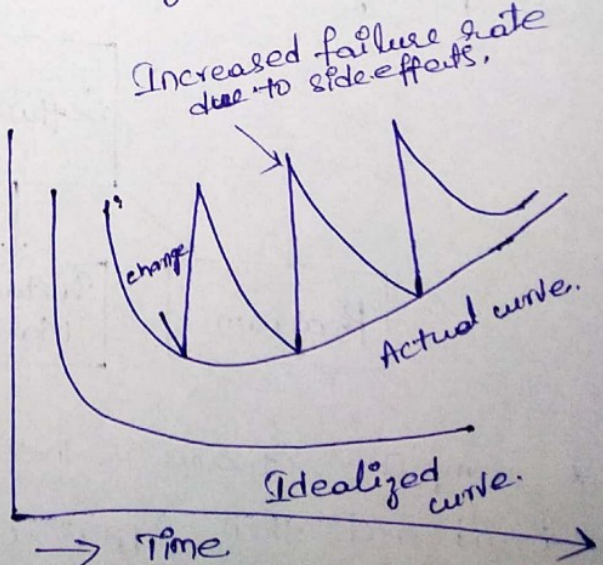
[2] Software doesn't ~~wearout~~ "wear out"

* H/w components suffer from the growing effects of dust, vibration, abuse, temparature extremes, and many other environmetal maladies. Stated simply the hardware begins to wear out.

* S/w is not susceptable to the environment maladies that cause hardware to "wear out."
In theory, therefore, the failure rate curve for software should take the form of the "idealized curve."



H/w failure curve.



S/w failure curve.

Scanned by CamScanner

* when a hardware component wears out, it is replaced by a spare part.
* there are no software spare parts.
* Every software failure indicates an error in design or in the process through which design was translated into machine executable code.
* therefore, the s/w maintenance tasks are considerably complex than h/w maintenance
* however, the implication is clear — s/w doesn't wear out. But it does deteriorate.

[3] Although the industry is moving toward component-based construction, most s/w continues to be custom built.

* A s/w component should be designed and implemented so that it can be reused in many different programs.
* Modern reusable components encapsulate both data and the processing that is applied to the data, enabling the s/w engineer to create new applications from reusable parts.
* In the h/w world, component reuse is a natural part of the engineering process.

Difference b/w Program and s/w:

Program	s/w
→ Small in size	→ Large in size.
→ Author himself is user-soul	→ Large number.
→ Single developer	→ Team developer.
→ Adopt development	→ Systematic development
→ Lack proper interface	→ well defined interface
→ Lack proper documentation	→ well documented.

Scanned by CamScanner

[Q2] Evolving role of Software?

Nowadays, software plays a major role with dual activity. It is a Product and a Vehicle for delivering a product.

* As a product, it delivers the computing potential embodied by computer hardware or a network of computers that are accessible by local hardware.
* whether the product or software resides within a mobile phone or operates inside a mainframe computer.
* Software is an information transformer likely to produce, manage, acquire, modify, display or transmit the information.

Dual role of S/w

* A product
  → Information transformer – Producing, managing and displaying.
* → Vehicle for delivering a Product.
  → control of computer (operating system), the communication of information (networks) and the creation of other programs.

The Software :-

* Provides good product with useful information.
* transforms the data so that it can be more useful in a local context
* manages business information to enhance competitiveness
* Provides a gateway to world wide networks like Internet

Scanned by CamScanner

The role of computer software has undergone significant changes over a time span of little more than 50 years.

* The software has seen many changes since its inception. After all, it has evolved over the period of time against all odds and adverse circumstances.

* Computer industry has also progressed at a break-neck speed through the computer revolution, and recently, the network revolution triggered and/or accelerated by the explosive spread of the internet and most recently the web.

* Computer industry has been delivering exponential improvement in price performance, but the problems with software have not been decreasing.

* Software still come late, exceed budget and are full of residual faults. As per the latest IBM report,

" 31% of the projects get cancelled before they are completed, 53% over-run their cost estimates by an average of 189% and for every 100 projects, there are 94 restarts."

=

Definition of Software Engineering.

* Software Engineering is defined as systematic, disciplined and quantifiable approach for the development, operation and maintenance of software.

—x—

Scanned by CamScanner

[Q3] The changing Nature of Software?

* The Software going to develop in different areas depends on the user. so, the developers of s/w, would need to know different categories of softwares before they are going to develop the project.
* There are seven broad categories of computer software Present for different users depending on the requirements they requested.
* The 7 broad categories of s/w are.

① System Software:—

* System s/w is a collection of programs written to service other programs.
* The systems s/w is characterized by
  → heavy interaction with computer h/w.
  → heavy usage by multiple users.
  → concurrent operation that requires scheduling resarce sharing and sophisticated process management.

  Eg: operating system, compiler,

② Application Software:—

* Application s/w consists of standalone programs that solve a specific business need.
* Application faillitates business operations or management/technical decision making.
* It is used to control business functions in real-time.

  Eg: Ms-office, Adobe reader, photoshop.

③ Engineering & Scientific s/w:—

* MS software faulitates engineering funtions & tasks.

Scanned by CamScanner

Eg: CAD – computer aided design application which is used to help an engineer in drawing and test the designed information.

④ Embedded Slw:-

* this Software resides within a product or System and is used to implement the control features and functions for the end-user and for the system itself.
* Embedded Slw can perform limited functions or significant function and control capability.
* while designing phase of the manufacturing Slw can be embedded inside, once the product is manufactured, the Slw can't be changed.

Eg: washing machines, Gps devices, Calculator.

⑤ Web Applications:-

* This kind of Slw is developed for users over the internet.
* Web apps are evolving into sophisticated computing environments that not only provide standalone features, computing functions and content to the end user, but also are integrated with corporate databases and business applications.

⑥ Artificial Intelligence Slw:-

* Artificial Intelligence Slw makes use of non numerical algorithms to solve complex problems that are not ~~amea~~ amenable to computation or straight forward analysis.

Eg: Robotics, expert systems, pattern recognition, artificial neural nlws, theorem proving, & game playing.

Scanned by CamScanner

[Q4] What is Legacy Software?

* Older programs are often referred to as legacy software.
* Legacy software systems were developed decades ago and have been continually modified to meet changes in business requirements and computing platforms.
* Many legacy systems remain supportive to core business functions and are indispensable to the business.
* Legacy software is characterized by longevity and business criticality.

Changes are made to legacy software:-

* Legacy systems must undergo some significant change. The changes needed are.
  1) The s/w must be adopted to meet the needs of new computing environments or technology.
  2) The s/w must be enhanced to implement new business requirements.
  3) The s/w must be extended to make it interoperable.
  4) The s/w must be re-architectured to make it viable within a n/w environment.

The quality of legacy s/w:-

Legacy systems sometimes have inextensible designs, convoluted code, poor or non existent documentation, test cases, and results that were never archived.

Scanned by CamScanner

[Q5] Discuss about Software Myths?

* In Software engineering, myths are the wrong beliefs or misbeliefs about ~~and~~ software and the process used to built it.
* myths are the misleading attitudes that have caused serious problems
* Software myths Propage false beliefs and confusions in the minds of Managers, users and developers.

3 types of myths.

1) Management myths.
2) customer myths.
3) Developer or Practitioner myths.

[1] Management myths:-

(a) The members of the organization believe that, they can acquire all the information they want. Standards, procedures, and principles are present to develop the project.

Reality:-

* Standards are often incomplete, inadaptable and outdated.
* Developers are often unaware of all the established standards
* Developers rarely follow all the known standards.

[b] If the project is behind schedule, increasing the number of programmers can reduce the time gap.

Reality:

* Adding more manpower to the project, which is already behind schedule, further delays the project.
* New workers take longer to learn about the project as compared to those already working on the project.

[c] If the project is outsourced to a third party, the management can relax and let the other firm develop s/w for them.

* outsourcing s/w to a third party doesn't help the organization, which is ~~incompleted~~ incompetent in managing and controlling the s/w project internally.

Scanned by CamScanner

[2] customer Myths :—

* A customer who requests computer s/w may be a person, a technical group, the marketing/sales department, or an outside company that has requested s/w under contract.

* The common customer myths are

[a] Brief requirements stated in the initial process is enough to start development; detailed requirements can be added later

Reality :—

* Adding requirements at a later stage often repeating the entire development process.

[b] s/w is flexible

* Starting development with incomplete and ambiguous requirements often lead to s/w failure. Instead, a complete and formal description of requirements is essential before starting development.

[b] Software is flexible; hense s/w requirement changes can be added during any phase of the development process.

Reality :* Incorporating change requests earlier in the development process costs lesser than those that occurs at later stages.

* s/w may require redesigning and extra resources.

[3] Developer or practitioner's myths :—

common developer myths are

[a] s/w development is considered complete when the code is delivered.

Reality :—

* 50% to 70% of all the efforts are expended after the s/w is delivered to the user.

[b] The success of a s/w project depends on the quality of the product produced.

Reality :— * The quality of programs is not the only factor that makes the project successful. Instead the documentation and s/w configuration also plays crucial role.

Scanned by CamScanner

[c] Software engineering requires unnecessary documentation, which slows down the project.

Reality:-

* Sw engineering is about creating quality at every level of the Sw project. Proper documentation enhances quality which results in reducing the amount of rework.

[d] The only product that is delivered after the completion of a project is the working programs.

Reality:-

* The deliverables of a successful project includes not only the working program ~~that~~ also but also the documentation to guide the users for using the Sw.

[Q6] Explain the Generic View of Process

* The road map to build high quality Sw product is ~~Sw~~ Process. Software process.
* Software process is adopted to meet the need of software engineers and the managers as undertake the development of software product.
* Different types of projects requires different Sw processes.

Software Engineering Layers:-

* Sw Engineering is a layered technology.

Definition:-

> Software Engineering is the Process of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

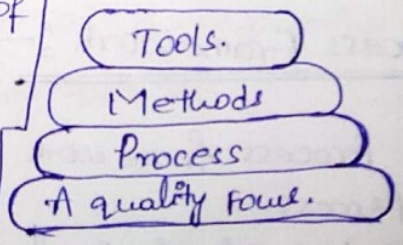


Fig: Sw Engineering Layers.

* Software Engineering is divided into 4 Layers.

Scanned by CamScanner

(1) The quality Focus:—

* An engineering approach must rest on quality.
* The "bed rock" that supports S/w Engineering is a quality focus.

(2) Process:—

* The foundation of Software Engineering is the Process Layer.
* It purely based on technology.
* Timely development of computer s/w
* Forms the base for management control of S/w Projects.

(3) Methods:—

* Methods provide technical aspects for building s/w.
* Methods encompass a broad array of tasks include communication, requirements analysis, design modeling, Program construction, testing and support.

(4) Tools:—

* Tools provide automated or semiautomated support for the Process and the methods.
* When tools are integrated so that information created by one tool can be used by another.
* A system for the support of s/w development is called computer aided s/w.

Process Framework:—

* Process framework is the foundation for a complete s/w process.
* It has set of small number of framework activities that are applicable to all s/w projects, regardless of their size or complexity.
* A Process framework in turn has set of umbrella activities

Scanned by CamScanner

that are applicable across the entire software process.

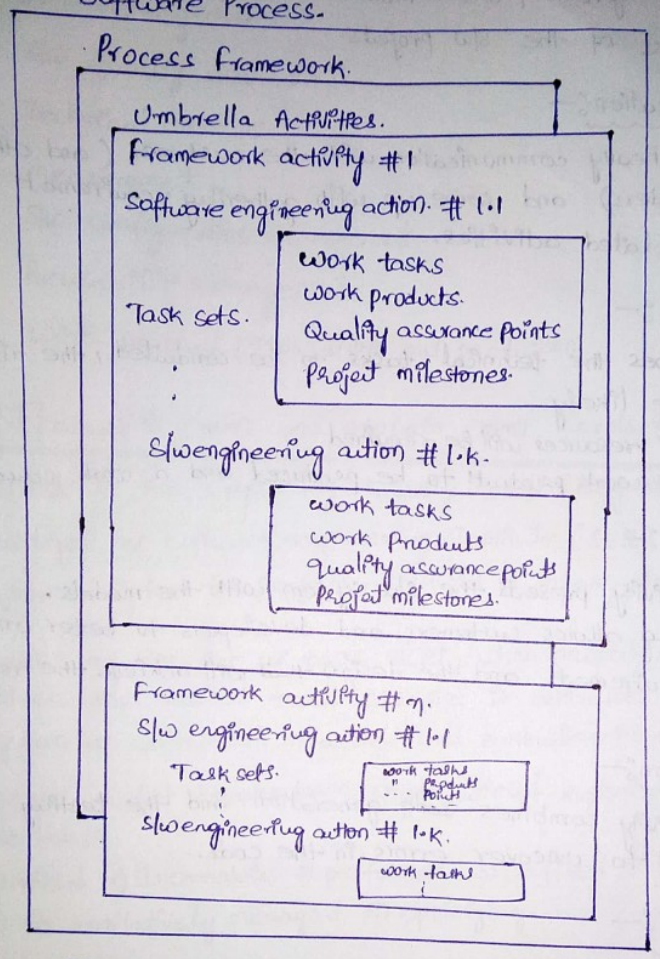


Fig:

* Each framework activity consists of set of S/w engineering actions
  -> A collection of related tasks (called task sets)
     work tasks
     work Products
     Quality assurance points
     Project milestones.

* Task sets will vary depending on the characteristics of a Project

Scanned by CamScanner

A generic Process Framework :-

A generic process framework is the basis and applicable to the most of the s/w projects.

(i) Communication :-

* Involves heavy communication with the customer (and other stakeholders) and come up with gathering requirements and other related activities.

(ii) planning :-

* Describes the technical tasks to be conduited, the risks that are likely
  - → The resources will be required
  - → The work products to be produced and a work schedule.

(iii) Modeling :-

* This activity presents the s/w system with the models.
* Modeling allows customers and developers to better understand s/w requirements and the design that will achieve the required system.

(iv) Construction :-

* This activity combines code generation and the testing that is required to uncover errors in the code.

(v) Deployment :-

* Deliver the project to the customer who evaluates the delivered Product and provides feedback.

S/w engineering Umbrella Activities :-

* S/w engineering Procen framework activities are followed by a number of umbrella activities.
* Umbrella activities are applied throughout a s/w project and help s/w team manage, control progress, quality, change & risk.

Scanned by CamScanner

activities are

* S/w project tracking and control
* Risk management
* S/w quality assurance
* Technical reviews
* measurement
* S/w configuration management
* Reusability management
* work product preparation and production.

[Q7] What is CMMI and explain CMMI Levels.?

* CMMI is the Capability Maturity model Integration which was developed by software engineering Institute (SEI) as a process improvement tool for projects, divisions or organizations.

* process as the set of tasks that, when properly performed produces the desired result otherwise it addresses the problems they can be controlled, measured and ~~controlled~~ improved.

* CMMI model breaksdaon organizational maturity into 5 levels.
five levels.
1) ~~Initial~~ 1) Incomplete 2) performed 3) Defined
4) Quantitatively managed 5) optimizing.

Level 0:- [Incomplete]

* process is not performed or does not achieve all goals defined for

Level 1:- [performed]

* work tasks required to produce required work products are being conducted

Level 2:- [Managed]

* people doing work have accen to adequate resources to get job done, stakeholders are actively involved, work tasks and Products are monitored, reviewed, and evaluated for conformance

Scanned by CamScanner

to process description.

Level 3: [Defined]

* Management and engineering process documented, standardized and integrated into organization-wide software process.

Level 4: [Quantitively Managed]

* Slw process and products are quantitatively understood and controlled using detailed measures.

Level 5:- [optimizing]

* continuous process improvement is enabled by quantitive feedback from the process and testing innovative ideas.

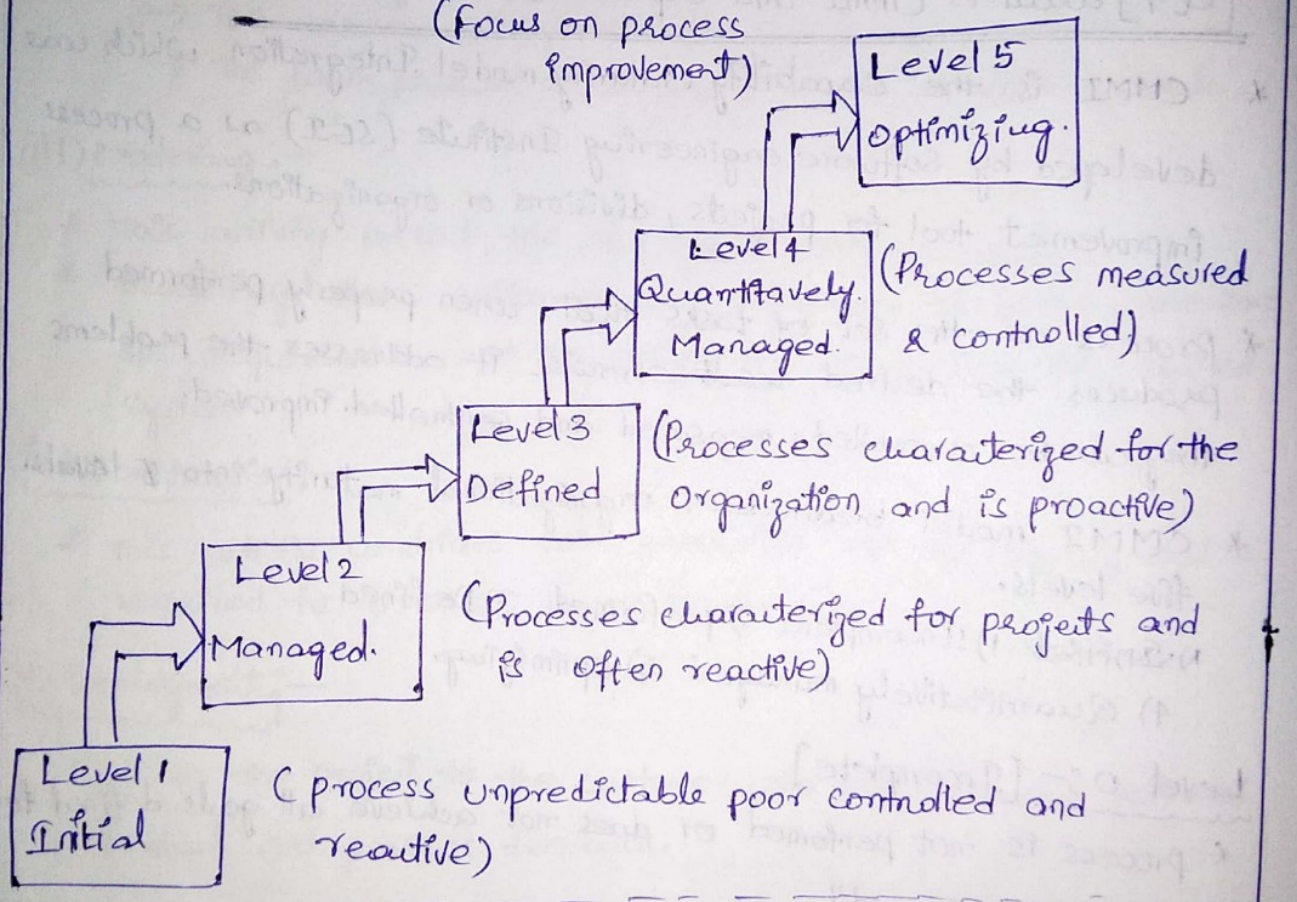
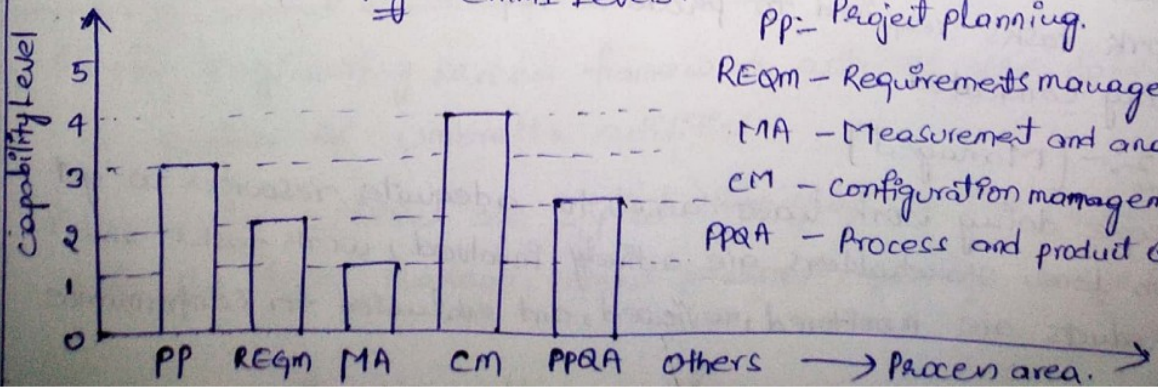


Fig: CmmI Levels.

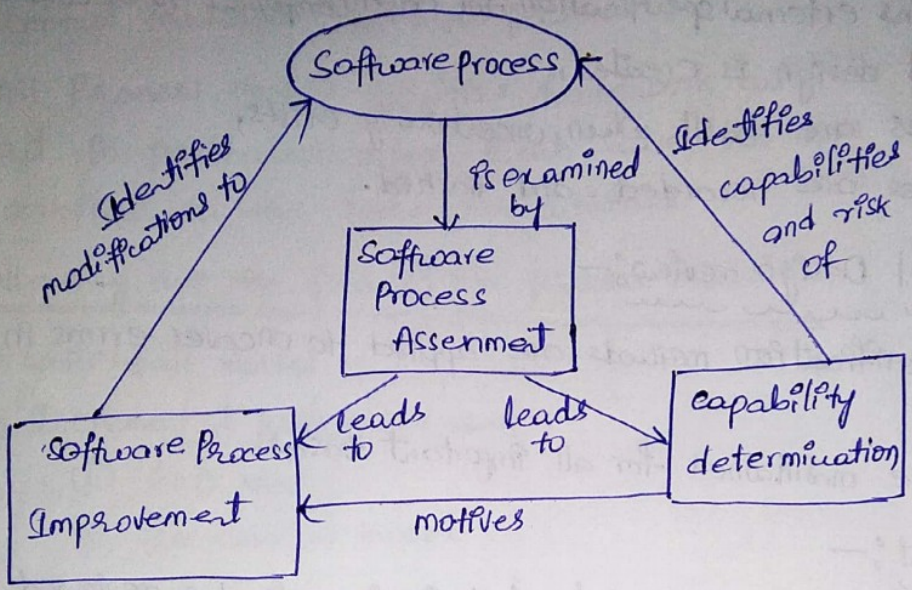


PP:- Project planning.
REQm - Requirements management
MA - Measurement and analysis.
CM - configuration management
PPQA - Process and product QA.

Scanned by CamScanner

Process Assessment:-

* The process should be assessed to ensure that it meets a set of basic process criteria that have essential for a successful software Engineering.
* This is used by industry professionals.



[Q] Explain personal and Team Process models.?

* Psp Personal software procen (PSP) and Team software process (TSP), both are largely iterative or evolutionary in the approach to software development.
* psp and TSp are interesting but are not pivotal to an understanding of process issues.

Personal software Process (psp):-

* The personal software process (PSP) model is good from the perspective that an individual software engineer can use it to improve his or her personal productivity and work product quality.
* PSP process model defines five activities
  1) planning 2) high-level design 3) high-level design review
  4) development 5) post mortem.

Scanned by CamScanner

* It stresses the need to identify errors early and to understand the types of errors.

1) Planning:- It isolates requirements and based these project schedule is created.

2) High-Level Design:-
* It contains external specification for each component to be constructed, component design is created, prototypes are built when uncertainty exists, all issues are recorded and tracked.

3) High-Level Design review:-
* Formal verification methods are applied to uncover errors in the design.
* metrics are maintained for all important tasks.

(4) Development:-
* ~~the component~~ component level design is refined & reviewed
* code is generated, reviewed, compiled and tested

(5) Postmortem:-
* using the measures and metrics collected, the effectiveness of the process is determined.

Team Software Process:-
* The goal of TSP is to build a "self-directed" project team that organizes itself to produce high quality s/w.
* Each project is "launched" using a "script" that defines the tasks to be accomplished.
* Teams are self-directed
* Measurement is encouraged
* measures are analyzed with the intent of improving the team process.

=

Scanned by CamScanner

Process models.

Prescriptive Process models:-

* we call the models as prescriptive because they Prescribe a set of process elements framework activities, s/w engineering actions, tasks, work products, quality assurance, and change control mechanism for each product.
* All Process models specifies framework activities to be carried out in particular order. Each have it's own emphasis workflow to the framework activities.

Following are the prescriptive process models.

1) waterfall model
2) Incremental process models.
   (i) RAD model
   (ii) Incremental model.
3) Evolutionary process model
   (i) prototype model.
   (ii) spiral model.
   (iii) concurrent process model.
4) Specialized process model.
   (i) component-Based development
   (ii) Formal methods model.
   (iii) Aspect-oriented s/w development
5) unified process.

=

Scanned by CamScanner

[Q] Explain about waterfall process model ?

* waterfall model sometimes called the classic life cycle, suggest a systematic, sequential approach to sw development that begins with customer specification of requirements and progresses through planning, modeling, construction and deployment, ~~contin~~
* There are times when the requirements of a problem are reasonably well understood,
* when work flows from communication through deployment in a reasonably linear fashion.
* when well-defined adoptions or enhancements to an existing system must be made.
* It may also occur in a limited number of new development efforts, but only when requirements are well-defined and reasonably stable.

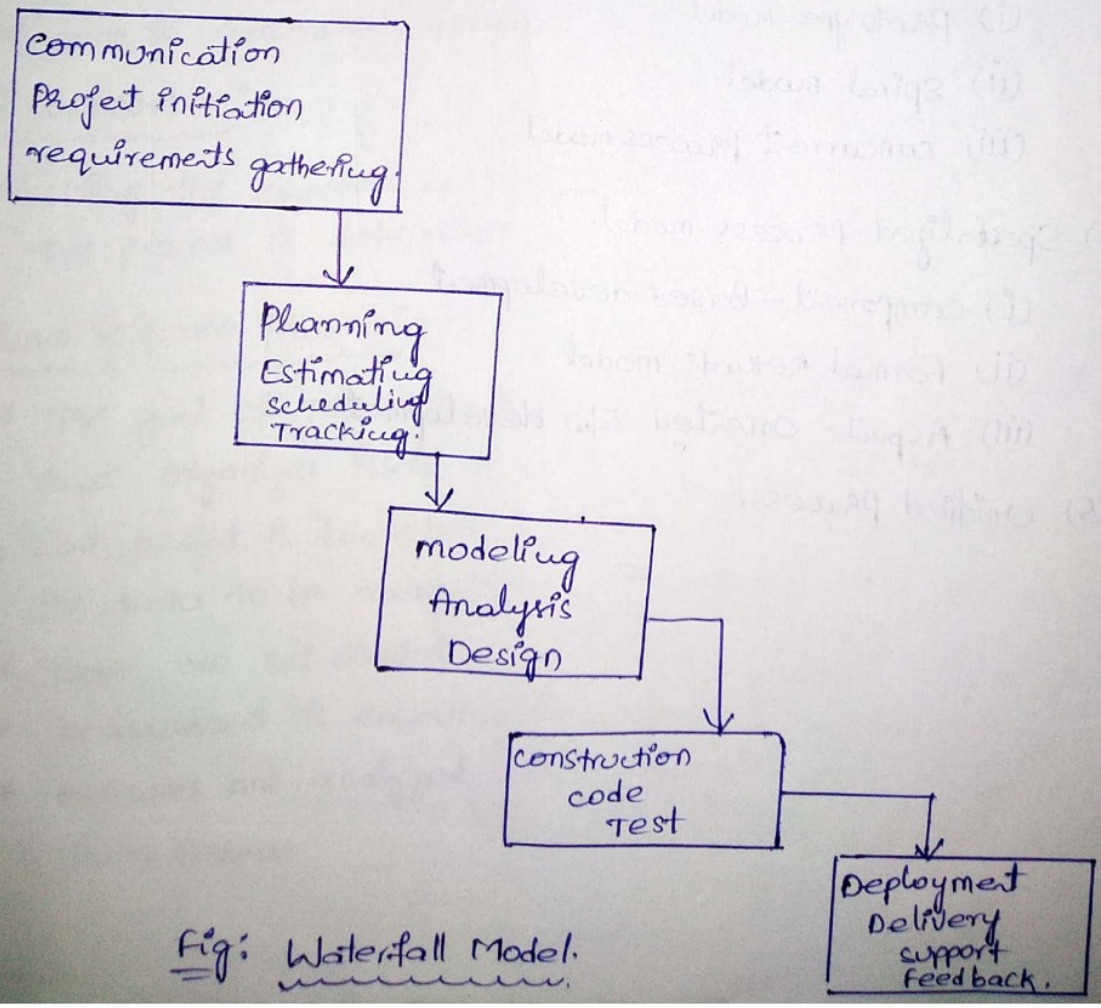


Fig: Waterfall Model.

Scanned by CamScanner

* It is a sequential model, The phases in this model are communication, planning, modeling, construction, and deployment.
* ~~The Input~~ In waterfall model, upon completion of one stage only next stage will start working.

Limitations :-

* The nature of the requirements will not change during development ; during evolution.
* The model implies that you should attempt to complete a given stage before moving on to the next stage.
* Does not account for the fact that requirements constantly change.
* It also means that customers cannot use anything until the entire system is complete.
* The model implies that once the product is finished, everything else is maintenance.
* Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.

=

[2] Incremental Process models.

(i) Incremental model :-

* Incremental model in SW engineering is a one which combines the elements of waterfall model in an iterative manner.
* It delivers a series of releases called increments which provide progressively more functionality for the client as each increment is delivered.
* In the incremental model of SW engineering, the waterfall model is repeatedly applied in each increment.

Scanned by CamScanner

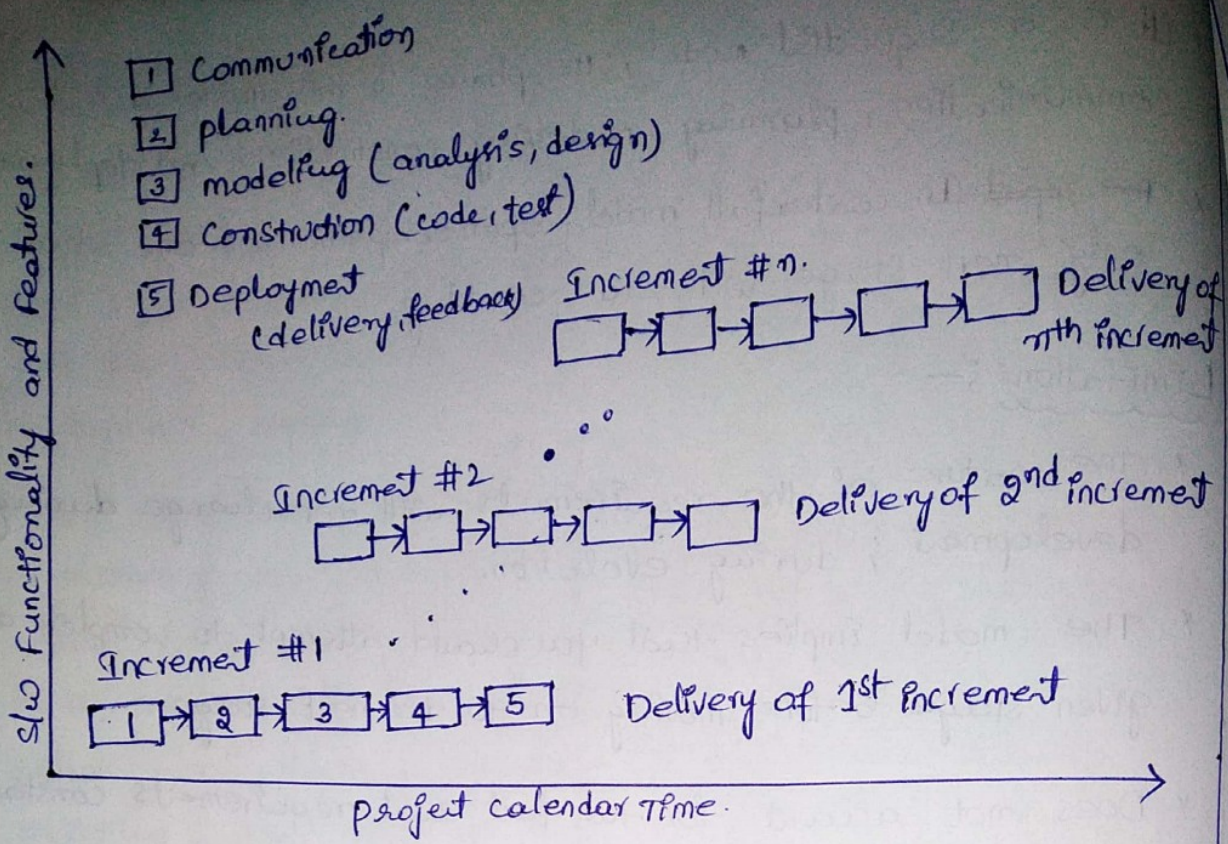


Fig: Incremental model.

As from the diagram, you can see there are 5 phases (tasks) which are carried out in each increment.

* The first increment is often a core product where the necessary requirements are addressed, and the extra features are added in the next increments.
* The core product is used and evaluated by the client.
* once the customer assesses the core product, there is plan development for the next increment
* Thus in every increment, the needs of the client are kept in mind, and more features and functions are added, and the core product is updated.
* This process continues until the complete product is Produced.
* one of the benefits of the incremental process model is that it can be planned to manage technical risks.

Scanned by CamScanner

Advantages of Incremental model :-

* Initial product delivery is faster.
* Lower initial delivery cost.
* Risk of changing requirement is reduced.
* workload is ~~les~~ less.

Disadvantages :-

* Each phase of an iteration is rigid and do not overlap each other.
* It requires a good planning designing.
* Rectifying a problem in one unit requires correction in all the units and consumes lot of time.

(ii) The RAD Model :-

* Rapid Application Development (RAD) is an incremental software process model that emphasizes a short development cycle.
* RAD is a "high-speed" adaption of the waterfall model, in which rapid development is achieved by using a component based construction approach.
* If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a fully functional system within a short period of time.

Drawbacks

(1) For large, but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.

2) If a system cannot properly be modularized, building the components necessary for RAD will be problematic.

Scanned by CamScanner

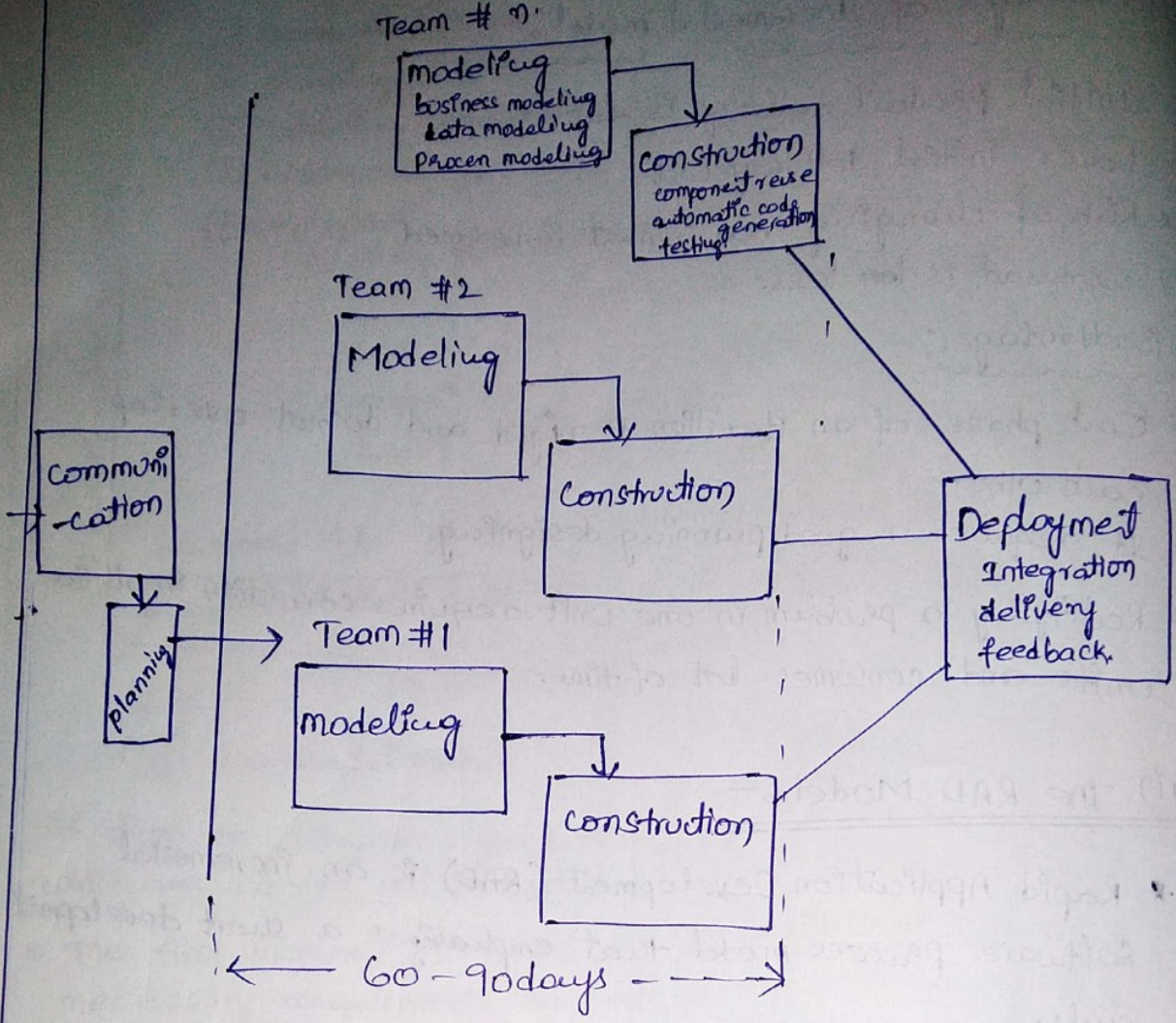


Fig: The RAD model

III. Evolutionary process models.

* Evolutionary models are iterative type models. They allow to develop more complete versions of the software.
* Following are the evolutionary process models.
  1. prototyping model
  2. The spiral model.
  3. Concurrent development model.

(i) The prototyping model:-

* The basic idea here is that instead of freezing the requirements before a design or coding can proceed,

Scanned by CamScanner

a throwaway prototype is built to understand the requirements.

* By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system.

* The prototyping paradigm begins with communication. The s/w engineer and customer meet and define the overall objectives for the s/w, identify whatever requirements are known.

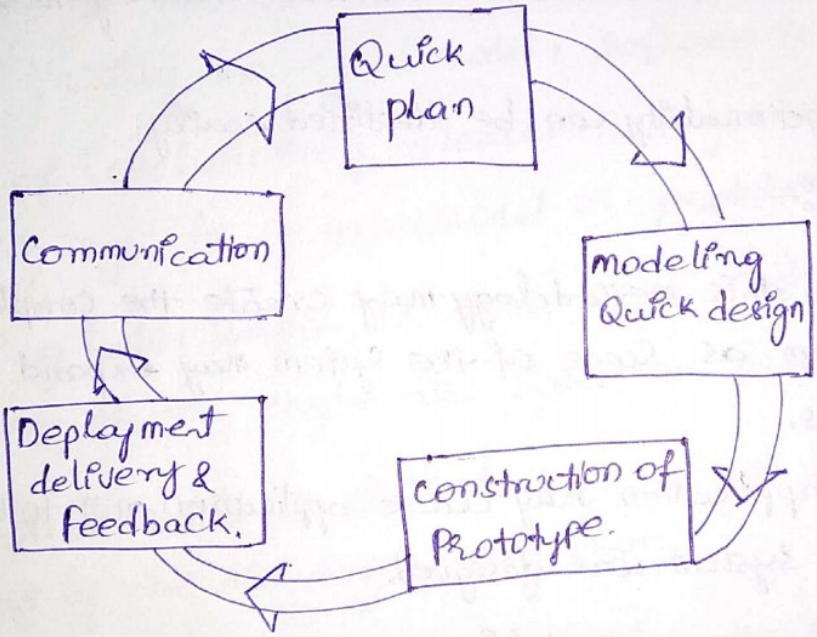


Fig: prototyping model

* A prototyping iteration is planned ~~quick~~ quickly and modeling occurs.
* The quick design focuses on a representation of those aspects of the s/w that will be visible to the customer/ end user.
* (eg: human interface layout or output display formats)
* The quick design leads to the construction of a prototype.

Scanned by CamScanner

* The prototype is deployed and then evaluated by the customer/user.
* Feedback is used to refine the requirements for the s/w. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.

Advantages:-

* users are actively participated in the development.
* Errors can be detected much earlier.
* Quicker user feedback is available leading to better solutions
* missing functionality can be identified easily.

Disadvantages:-

* Practically, this methodology may create the complexity of the system as scope of the system may expand beyond original plans.
* Incomplete application may cause application not to be used as the full system was designed.

When to use Prototype model:-

* prototype model should be used when the desired system needs to have a lot of interaction with the end users.
* Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model.

Scanned by CamScanner

(2) The Spiral Model :—

* The spiral model is evolutionary process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.

It has two distinguishing features

(a) A cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk.

(b) A set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory solutions.

Using the spiral model, software is developed in a series of evolutionary releases. During early stages, the release might be a paper model or prototype.

* During later ~~stages~~ iterations, increasingly more complex versions of the engineered system are produced.

* A spiral model is divided into set of framework activities divided by the s/w engineering team.

* As this evolutionary process begins, the software team performs activities that are implied by circuit around the spiral in clockwise direction, beginning at the center. Risk is considered as ~~revolution~~ each revolution is made.

Scanned by CamScanner

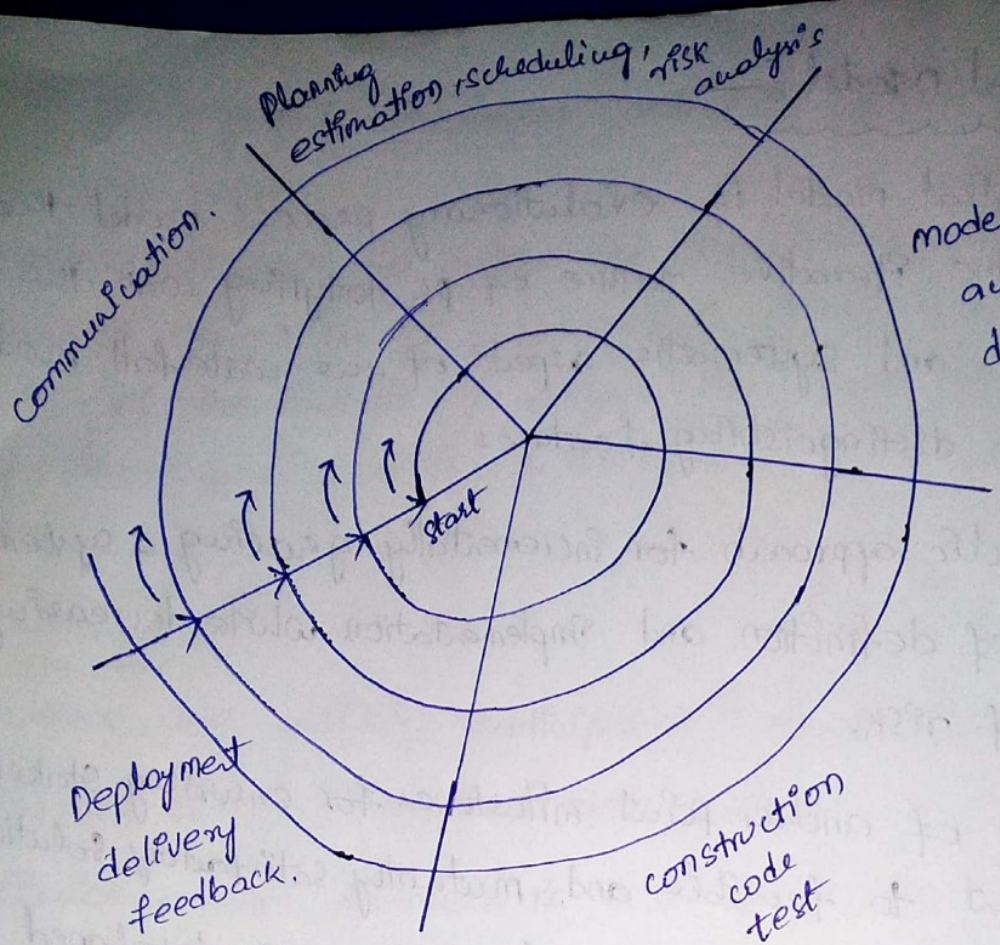


Fig: A typical spiral model.

The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the s/w.

* Each pass through the planning region results in adjustments to the project plan.
* cost and schedule are adjusted based on feedback derived from the customer after delivery.
* Unlike other process models that end when software is delivered, the spiral model can be adapted to apply throughout the life of the software.

Scanned by CamScanner

(3) The CONCURRENT DEVELOPMENT MODEL.

* The concurrent develop model, Sometimes called concurrent engineering can be represented schematically as a series of framework activities, s/w engineering actions of tasks, and their associated states.
* The concurrent model is more appropriate for system engineering profets where different engineering teams are involved.

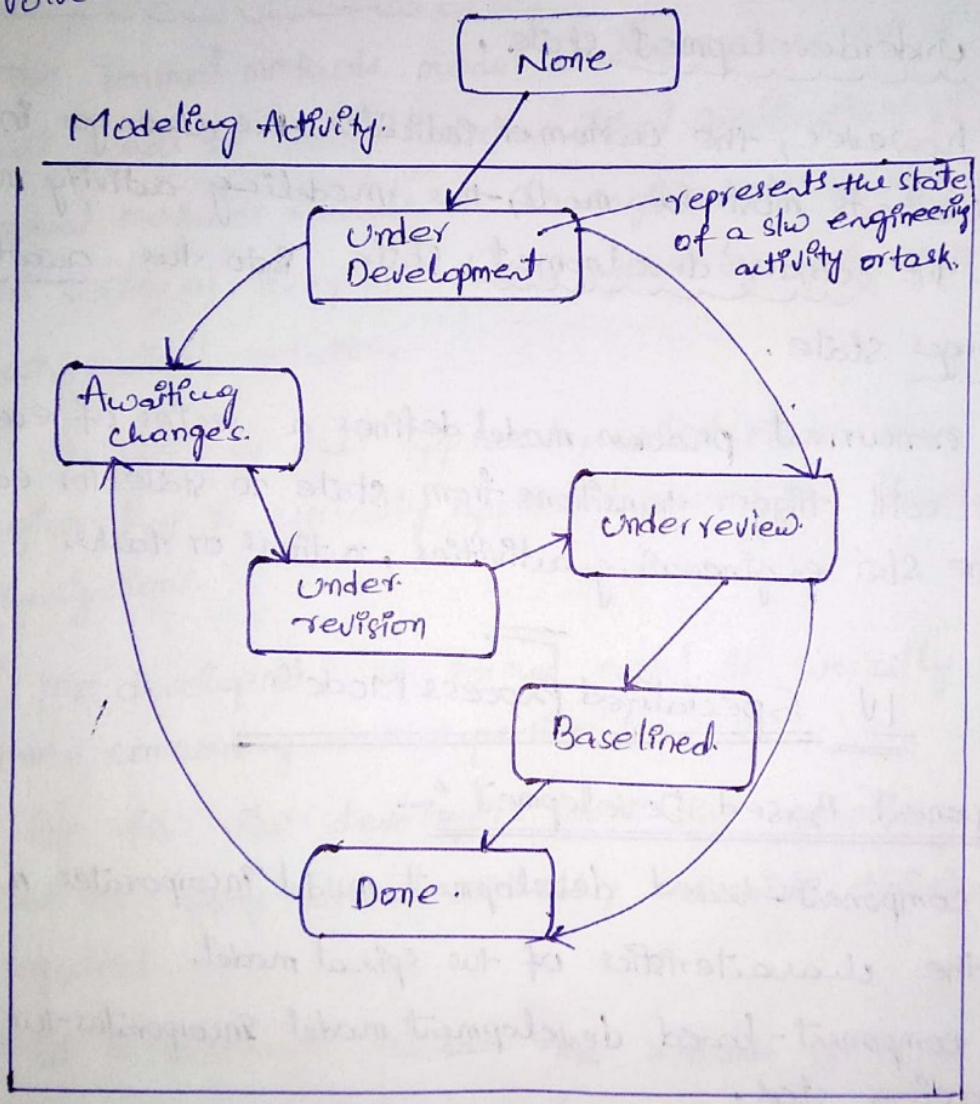


Fig: one Element of the concurrent process model.

* Figure above provides a schematic representation of one software engineering task within the modeling activity for the concurrent process model.

Scanned by CamScanner

* The activity - modeling may be in any one of the states noted at any given time.
* All activities exist concurrently but reside in different states. For example, early in the project the communication activity has completed its first iteration and exists in the awaiting changes state.
* The modeling activity which existed in the none state while initial communication was completed now makes a transition into underdevelopment state.
* If, however, the customer indicates the changes in requirements must be made, the modeling activity moves from the under development state into the awaiting changes state.
* The concurrent process model defines a series of events that will trigger transitions from state to state for each of the s/w engineering activities, actions or tasks.

IV Specialized Process Models.

(a) Component Based Development :-

* The component-based development model incorporates many of the characteristics of the spiral model.
* The component-based development model incorporates the following steps.
  (i) Available component-based products are researched and evaluated for the application domain.
  (ii) Component integration issues are considered
  (iii) S/w architecture is designed to accomodate the components.

Scanned by CamScanner

(iv) Components are integrated into the architecture;

(v) Comprehensive testing is conducted to ensure proper functionality.

The component-based development model leads to s/w reuse, and reusability Provides s/w engineers with a number of measurable benefits.

=

(b) The Formal Methods model :—

* The formal methods model encompasses a set of activities that leads to formal mathematical specifications of s/w.
* Formal methods enable a s/w engineer to specify, develop and verify a computer-based system by applying a rigorous, mathematical notation.
* A variation of this approach, called clean-room s/w engineering is currently applied by some s/w development organizations.

(i) The development of formal model is currently quite time-consuming and expensive

(ii) B/c few s/w developers have the necessary background to apply formal methods, extensive training is required.

(iii) It is difficult to use the methods as a communication mechanism for technically unsophisticated customers.

=

Scanned by CamScanner

V. UNIFIED PROCESS MODEL

Unified Process is incremental and iterative process model extentensively used for object oriented software development.

* In this model, requirements or features had to be identified before hand. And then over several iterative development the sw is developed and each time it is deployed at the client side, client feedback is obtained.
* Unified process development occurs over 4 phases.
  1) Inception
  2) Elaboration
  3) Construction
  4) Transition.
* Each of these above phases, consists of many increments.

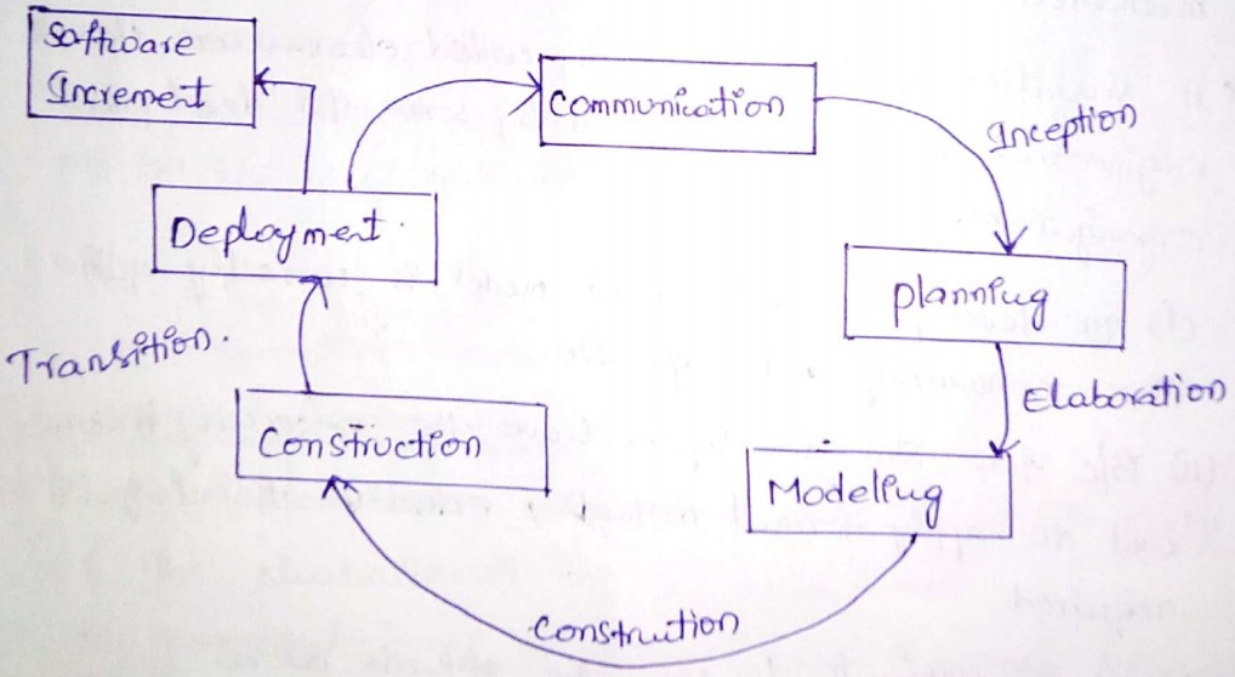


Fig: Unified process for one increment.

* In the inception phase, the initial use case model is proposed; That is the features, the business case, risk list, project plan, planning documents, and any prototype.

Scanned by CamScanner

* In the elaboration phase, models are made like analysis model, preliminary model, manual use case model and so on.
* In the construction phase, the s/w is developed, the test plan is made, several manuals like user manual and installation manuals are prepared. And finally the developed software increment is deployed at the customer site; The beta test reports and the user feedback is obtained.

Unified process work products.

Inception	Elaboration	construction	Transition.
* Vision daument * Initial use case model * Initial business case * initial risk list * project plan * prototype(s)	* use-case model * Requirements Analysis model * Preliminary model * revised risk list * preliminary manual	* Design model * s/w components * Test plan * Test procedure * test cases * user manual * Installation manual	* s/w increment * Beta test reports * user feedback.

[Q] SEI-CMMI LEVELS.

* CMMI is capability maturity model Integration which is proposed by Software Engineering Institution (SEI)
* CMMI helps organizations to improve the quality of the s/w they developed
* Adopting to SEI-CMMI model had significant benefits to the business.
* many commercial organizations adopt CMMI framework for their own improvement initiatives.

Scanned by CamScanner

* The different levels of SEI-CMMI have ~~been~~ been designed so that it is easy for an organization to slowly build its quality system starting from scratch.

* SEI-CMMI classifies 5 maturity levels.

  Level 1 : Initial.

  Level 2 : Repeatable.

  Level 3 : Defined

  ~~Lel~~
  Level 4 : managed

  Level 5 : optimising.

Level 1 :- [Initial]

* S/w organization at this level is characterized by ad hoc activities.
* Very few or no processes are defined and followed.
* And no formal project management practices are followed, as a result time pressures at the end of the delivery time, as a result it leads to low quality products.

Level 2 : [Repeatable]

* At this level, the basic project management practices such as tracking cost and schedules are established.
* configuration management tools are used on items identified for configuration control.
* Size and cost estimation techniques such as function point analysis, cocomo models are used.
* Repeatable level occurs for organizations, when a company produces family of products. Since the product are very similar, the success story on development of one product can repeated for another.

Scanned by CamScanner

Level 3 :- [Defined].

* At this level, the process for both management and development activities are defined and documented.
* At this level, the organization builds up the capabilities of its employees through periodic training programs. Also review techniques are emphasized and documented.

Level 4 :- [Managed]

* At this level, the focus is on software metrics. Both Process and product metrics are collected
* quantitative quality goals are set for the products and at the time of completion of development it was checked whether the quantitative quality goals for the product are met.

Level 5 :- [optimising]

* At this stage process and product metrics are collected. and process measurement data are analyzed for continuous Process Improvement.
* At CMM Level 5, an organization would identify the best Software Engineering practices and innovations (which may be tools, methods, processes)
* Level 5 organization usually have a department whose responsibility is adopting to latest tools and techniques, technologies and propagate them world wide.
* Level 5 Organizations use configuration management techniques to manage process change, since process changes occurs continuously

Scanned by CamScanner

UNIT – II

Software Requirements :- Functional and non functional requirem, User requirements, System requirements, Interface specification, the Software requirements doumet.

Requirements Engineering Process :- feasibility study, Requirements elicitation and analysis, Requirements validation, Requirements management.

System models : context models, Behavioural models, Data models, Objet models, Structured models.

(I) Functional and Non functional requirements :

(i) Functional requirements :

* The functional requirements for a system describe what the system should do.
* hao system react to particular inputs, hao system should behave in particular situations. (Some cases it may also say what system should not do)

(ii) Non-functional requirements :

* Non-functional requirements are the constraints on services offered by the system (timing constraint, development procen, constraints, standard constraints).
* These requirements apply on the system as a whole rather than individual system.

1) Functional requirements :-

* Example of mental health care patiet management system (MHC-PMS) maintains information about patients receiving treatment for mental health problems

1. User shall able to search the appointments list for all clinics.
2. System shall generate each day, for each clinic, a list of Patients who are expeited to attend appointments that day.

3). Each staff using the system should uniquely identified by his/her eight digit employee number.

* Above all functional user requirements defines specific facilities
* system developer to interpret ambiguous requirement in a way that simplifies its implementation.

Eg: user shall search the appointment (1st requirement)

patients with mental health problem are sometimes confused, ie. they may have appointment in one clinic but actually go to a different clinic. (bcoz if appointment is there they are recorded as attended irrespective of clinic)

* so system developer implement in another way first choose a clinic and then carry out search.
* so requirements should be complete (all services required by user should be defined) and consistent. (requirements should not have contradictory definitions).

Problems :-

Practically for large complex systems it is impossible to achieve requirements consistency and completements.

reason :- 1) easy to make mistakes & omission when writing specification.

2) Many stakeholders in a large system, each of them have different inconsistent needs.

Stakeholder (a person or role that is affected by the system in some way)

Solution :- inconsistente requirements are included in the specification (problem only arise through deep analysis or after system is delivered to the customer)

2) Non-functional requirements

* Non-functional requirements, not directly concerned with the specific services delivered by the system.
* They may relate to emergent system properties such as reliability, response time, store occupancy, security, safety, interoperability etc.
* They define constraints on system implementation (I/O device capabilities, interface with other system)
* Non-functional requirements such as performance, security or availability usually specify constraint characteristics of a system as a whole.
* failing to meet non functional requirements means whole system is unusable.

Ex: 1) aircraft system if doesn't meet reliability requirements will not be certified as safe for operation.

2) Embedded control system fail to meet its ~~operation~~ performance requirements the control functions will not operate correctly.

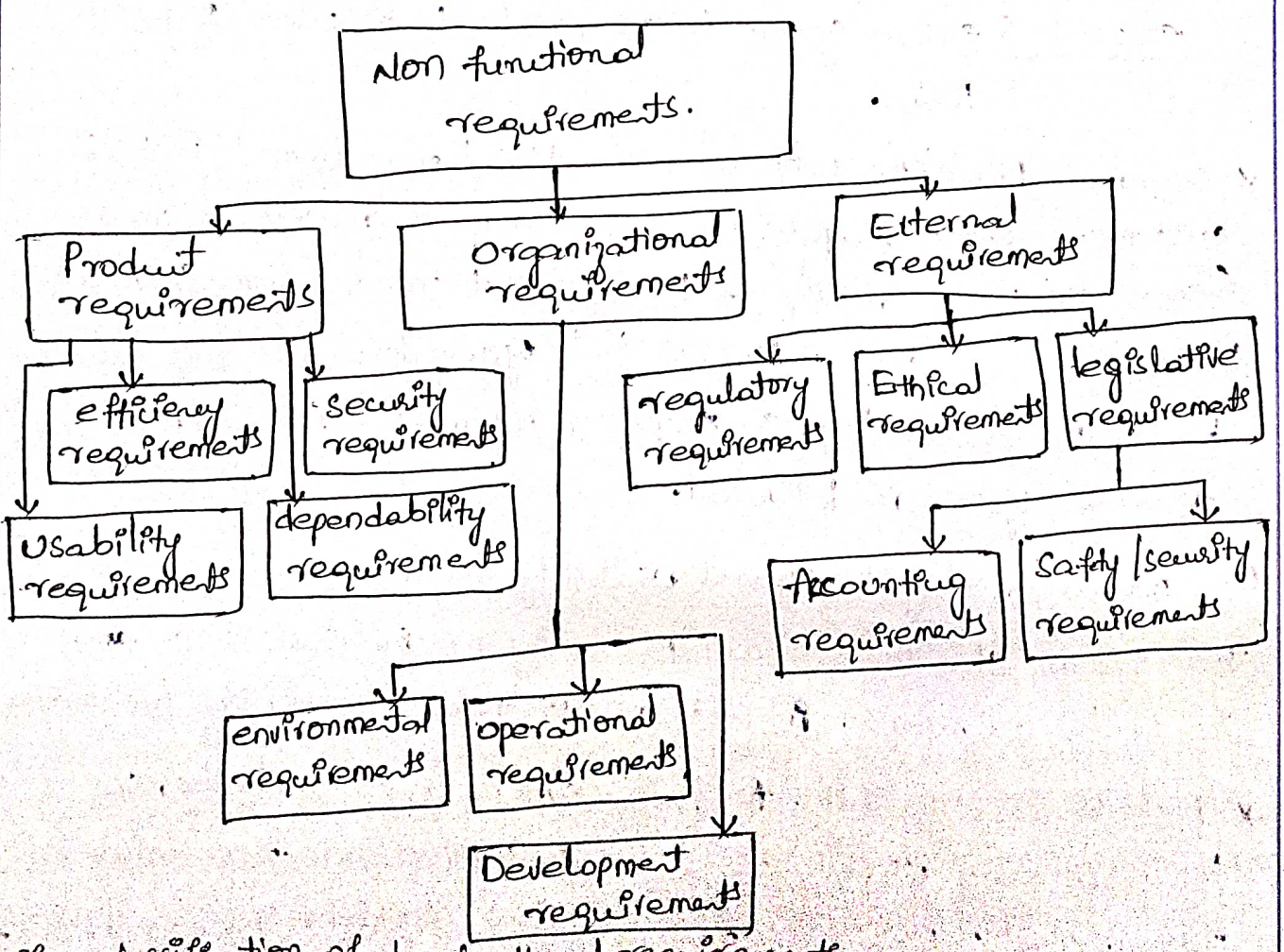


Fig: Classification of Non-functional requirements

Product requirements :- Specify the behaviour of the software
Ex: performance requirements (how fast system must execute, how much memory it requires), reliability requirements (acceptable failure rate), security requirements & usability req.

Organizational requirements : these requirements are system requ derived from policies and procedures in customer & developer organization.

Ex: operational : define how the system will be used
development : Process standard to be used
environmental : specify operating environment of the system.

External requirements : covers all requirements that are derived from factors external to the system and its development process.

Ex: regulatory : what must be done for the system to be approved for a use.
legislative : System operates within the law, it is give assurance
ethical : ensure system is acceptable to its users & general public.

* common problems with non-functional requirements is that customers often propose these requirements as general goals (ease of use, ability of the system to recover from failure)
* goals are of good intensions but system developer face problems bcoz. subsequent dispute, scope of interpretation is seen by him after the system is delivered.

Ex: manager might express usability requirements.
The system should be easy to use by the medical staff and should be organized in such a way that user errors are minimized
* but it is impossible to verify system goal objectively, so by converting usable requirements to testable non-functional requirement. we can atleast include software instrumentation.

to count errors made by users when testing the system. Medical staff shall be able to use all the system functions after proper training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use.

* So, if possible write non-functional requirements, it can be quantitatively measured and objectively tested.

* Metrics to specify non functional requirements.

Speed	-	Processed transactions event response time Screen refresh time
Size	-	1M bytes Number of Rom chips
Easeofuse	-	Training time Number of help frames.
Reliability	-	mean time to failure Probability of unavailability Availability.
Robustness	-	Time to restart after failure Percentage of events causing failure.
Portability	-	percentage of target dependent statements. No. of target systems

II) *** The Software Requirements document (SRS Document)

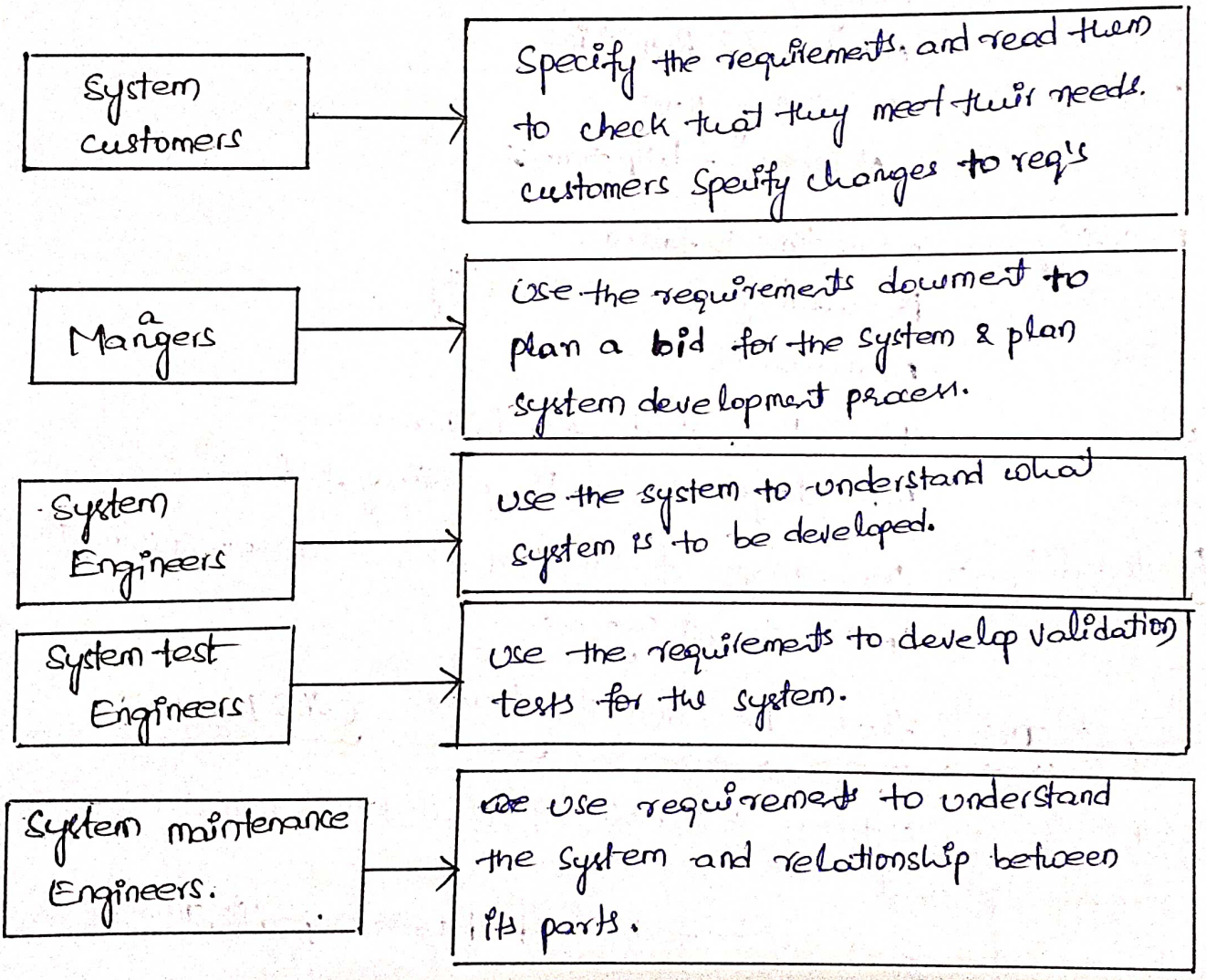
* SRS is an official statement of what the system developers should implement
* It includes user requirements and detailed specification of the system requirements
* If there are large number of requirements, the detailed system requirements may be presented in separate document.

* requirements doument is essetial when outside contractor is developing software system. But agile development says requirements change rapidly. So requirements doument become outdated as soon as it is written, causing waste of effort.

* Instead of that prepare doument using extreme programming collect requirements incrementally and write them as user stories, the user then prioritizes requirements for implementation.

Ex: Business systems.

* requirements doument has a diverse set of uses, ranging from seniors management to the engineers responsible for developing the software.



* Different types of users means the requirements doument has to be a compromise between communicating the requirements to customers, defining requirements in detail for developers & testers including possible system evolution.

* Level of detail in requirements daument depends on the type of system that is being developed and development process used.
* Critical systems needs detail daument becoz safety & security is analyzed.
* System developed by separate company than system specifications need to be detail.
* If Individual Iterative process is used then requirements daument can be less detail and any ambiguity can be resolved during development process.

(i) IEEE Standard for requirements daument.

Chapter	Description.
Preface	– Define expected readership of daument, describe history, include logical basis for creation of new version & summary of changes made in each version.
Introduction	– Describe need for the system, describe systems functionalities & explain how they work with other systems, how systems fits in overall business; objectives of the organization.
Glossary	– Define technical terms used, do not make any assumptions about the experience.
User requirements Definition	– * Describes services provided by users * Non functional req * Description use natural language, diagrams or other notations are used for representing non functional requirements. * Product and process standards must be specified.
System architecture	– * High level overview of system architecture * Distribution of functions across system modules * Architectural components that are reused are highlited

System requirements Specification – functional & non functional in detail further detail add to non functional interface to other system may be defined.

System models – set out one or more system models showing relationship b/w system components & system & its environment.

System Evolution – Anticipated changes due to H/w evolution, changing user needs.

Appendices – H/w & Database descriptions related to the application which is being developed.
Ex: optimal configuration of the system, Logical organization of data ~~based~~ used by the system & relationship b/w data.

Index – Normal alphabet index, index of diagrams index of functions etc.

Domain Requirements:-

* derived from the application domain of the system rather than from the specific needs of system users.
* They include specialized domain terminology or reference to domain concepts.
* Set out how the configuration is carried out becoz these requirements are specialized, so software engineers often find it difficult to understand how they relate to other system requirements.
* These requirements are important becoz they often reflect fundamentals of application domain, if these req's are not satisfied then it is impossible to make the system work satisfactorily.

E1: There shall be a standard user interface to all databases that shall be based on Z39.50 standard.

=

(III) User Requirements:-

* These requirements derive the functional and non functional req's so that they are understandable by system users without detailed technical knowledge.
* Should specify external behavior of the system and avoid system design characteristics.
* write the req's in simple language with simple tables & forms and intuitive diagrams.

Problems with natural language:-

1. Lack of clarity:- It is sometimes difficult to use language in a precise and unambiguous way without making the document difficult to read.

2. Requirements confusion:- functional & non functional req's goals and design information may not be clearly distinguished.

3. Requirements Amalgamation:- several req's may be expressed together as a single requirement.

E1: LIBsys shall provide financial accounting system that maintains records of all payments made by users of the system, system managers may configure the system so that regular users may receive discounted rates.

This example includes both conceptual & detailed information. i.e, it should be an accounting system as an inherent part of LIBsys also includes detail that accounting system should support documents of regular LIBsys users.

* To minimize misunderstandings when writing user req's following simple guidelines can be followed.

1. Invent a standard format and ensure that all req's definition use the same format like
   → Initial requirement in bold face
   → state of logic basis with each user requirements
   → reference to more detail system requirement specification

2. Use language consistently:- always distinguish between mandatory (written as shall; ie system must support) and desirable requirements. (written with should not essential)

3) Use text highlighting (Bold, italic) to pick out key parts of the requirements

4) Avoid use of computer jargon (non technical terms)

=

(IV) SYSTEM REQUIREMENTS.

* System requirements are used as starting point for the system design.
* They describe external behavior of the system and its operational constraint
* Not concerned with how the system should be designed or implemented
** If a complex software system is developed, it is impossible to Provide all design & information.

Reasons:-

1)* You design an initial architecture of the system to help structure the requirements specification
System req's are ~~designed~~ organized accordingly to different subsystems that make up the system.

2) * Most cases system must interoperate with other existing systems

3) Use of specific architecture to satisfy non functional req's may be necessary (Because external regulatory needs to certify that the system is safe).

(i) Natural language specification:—

* Natural language is used to write system & user reqts. for system requirements natural language specification is can be confusing and hard to understand.

① Natural language understanding relies on the specification readers & writers using same words for same concept. which leads to misunderstandings becoz of ambiguity of natural lang

② Language is over flexible (you can say anything in different ways it is upto reader to findout whether requirements are same or different)

③ No easy way to modularize natural language requirements, it may be difficult to find all related requirements.

— Became of these problems natural language specification is prone to misunderstandings, and if any requirements discovered in the later stages leads to expensive to resolve.

=

Notations for requirements specifications:—

*Alternatives to natural language specification for specifying reqts are

1) Structured Natural language: This approach depends on designing standard form or templates to express the reqts specification.

2) Design description languages:— This approach uses a language like a programming lang but with more abstrait features to specify reqts by defining an operational model of the system.

3) Graphical Notations:— A graphical language, telt annotations is used to define the functional reqts. Usecase & sequence diagrams are commonly used

(4) Mathematical specifications:— Notations based on mathematical concepts such as finite state machine or sets.

(1) Structured Language Specifications

* Is a way of writing system req's where there is less freedom of all requirements written in a standard way.

Advantages:-

1) Expressiveness & understandability of natural language but ensure degree of uniformity.
2) limit the terminology and use templates to specify system req's
3) Incorporate control constructs derived from programming languages and graphical highlighting to partition the specification.

Ex:- Insulin pump system

Function : Insulin dose

Description : Dose of insulin if it is in safe zone 3 and ≠ units

Inputs:- current reading (r_2), previous two reading (r_0, r_1)

Source : current reading from sensor

Outputs:- Dose of insulin delivered.

Destination:- Maintain control loop

Action:- Dose is zero if sugar level is stable or falling or level is increasing but rate of increase is decreasing level is increasing but rate of increase is increasing then dose compute by dividing diff blw current sugar level and previous level by 4 and rounding the result, if result : zero the dose is set to minimum dose to deliver.

Requires: Two previous readings to compute sugar level rate of change

Precondition:- insulin reservior contains at least min allowed single dose.

Post condition:- r_0 is replaced r_1 then r_1 is replaced by r_2.

Side effects : None.

When standard form is used for specifying functional req's the following information should be included.

1. Description of the function or entity being specified
2. Description of i/p's and where these come from
3. " " o/p's and where they go to
4. Indication of what other entities are used
5. Description of action to be taken
6. If function is used pre-condition setting out what must be true before function called and post-condition specifying what is true after the function is called.
7. Description of side effects if any.

(V) Interface Specification:—

* All software systems must operate with existing systems that have already implemented & installed in an environment.
* If new & existing system should work together. Interface of existing system must be specified.

Three types of interfaces:—

1. Procedural interface:— where existing programs or such systems offer a range of services that are accessed by calling interface procedures (API's)
2. Data Structures:— that are passed from one sub system to another graphicals, data models are best notation.
3. Representation of data:— that have been established for an existing sub system (common in embedded, real time system) best way to describe these is probably to use a diagram of the structure with annotations explaining the functions of each group.

Ex: procedural interface definition defined in java

```
Interface PrintServer {
    // defines an abstract printerServer
    // requires interface printer, interface PrintDoc
    // Provides initialize, print, displayPrintQueue, cancelPrintJob,
                                                    switchPrinter.
```

```
Void initialize(printer P)
Void print (Printer P1, PrintDoc d);
Void displayPrintQueue (Printer P);
Void cancelPrintJob (Printer P1, PrintDoc d);
Void switchPrinter (Printer P1, printer P2, PrintDoc d);
} // printServer.
```

=.

(VI) ## Requirements Engineering Process.

Requirements Engineering Process includes 4 high-level activities.

1) Focus on assessing if the system is useful to the business (feasibility study)
2) Discovering requirements (elicitation and analysis)
3) Converting these requirements into some standard form (specification) and
4) checking that the requirements actually define the system that the customer wants (Validation)

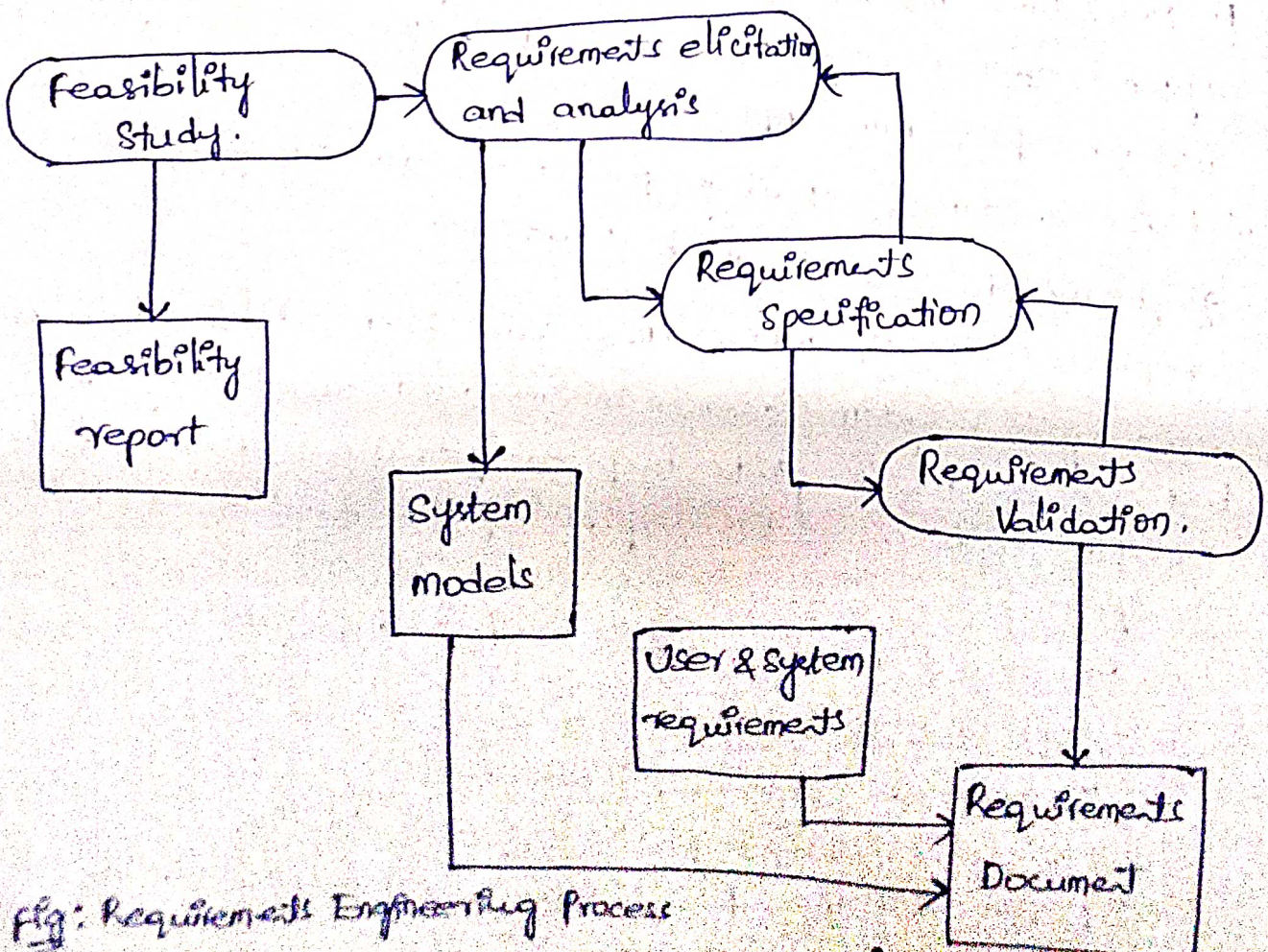
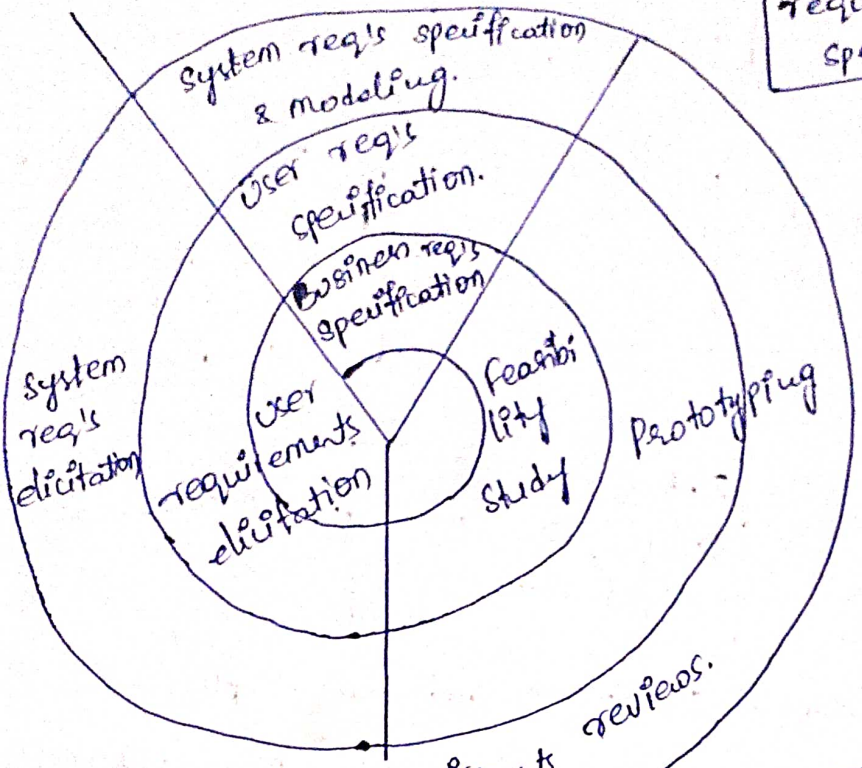


Fig: Requirements Engineering Process

Spiral model of requirements engineering process.



* people consider that, requirements engineering process is a way of specifying requirements using structured analysis method such as analyzing the system, and developing a set of graphical system models, such as use case model, structured method specificati of requirements belongs to only one part in requirements engineering. Still there is much more to be added for requirements engineering.

One of such activity is Requirement elicitation.

* It is a human centered activity.

(VII) *** Imp. **Requirements Elicitation and Analysis**

* After initial feasibility study, the next stage of requirements engineering process is requirements elicitation & analysis
* In this activity, SW engineers work with customers & system end-users to find out about the application domain,
* what services the system should provide, the required performance of the system, HW constraints and so on.

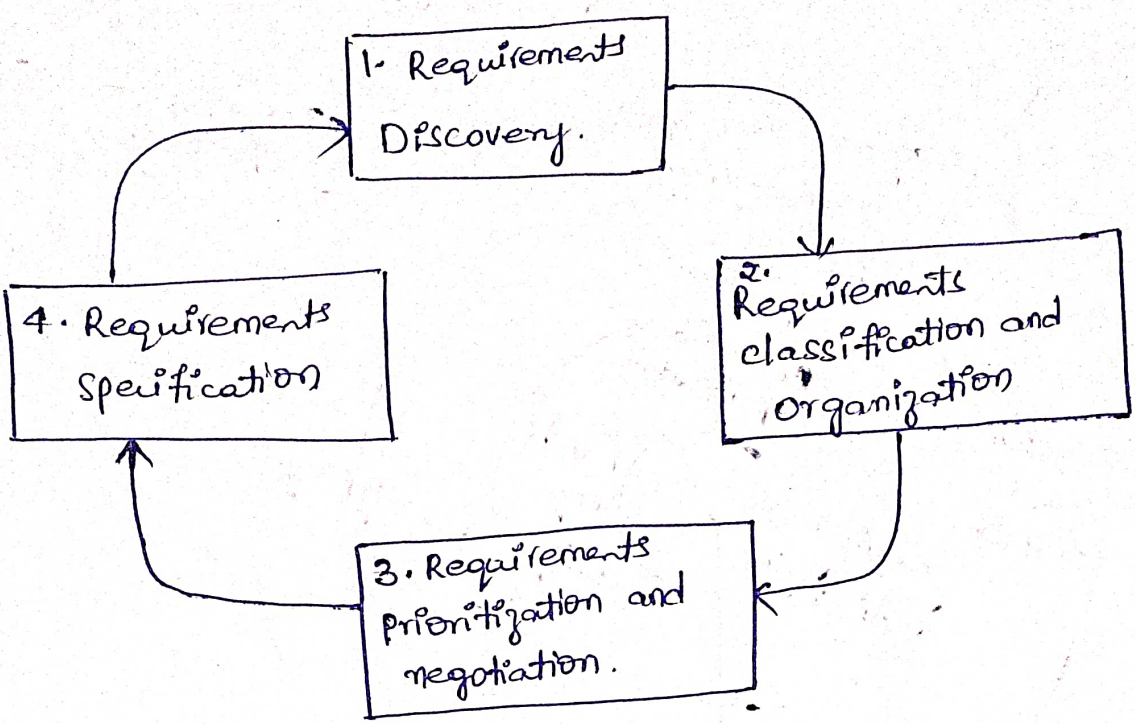


Fig: Requirements Elicitation and analysis process.

* Requirements elicitation and analysis may involve a variety of different kinds of people in an organization.
* A system stakeholder is anyone who should have some direct or indirect influence on the system requirements.
* An end user - who will interact with the system, and anyone else in the organization who will be affected by it.
* Other system stakeholders might be engineers who are developing or maintaining other related system, business managers, domain experts and trade union representatives.

The process activities are :-

1. Requirements Discovery :-

* This is the process of interacting with stakeholders of the system to discover their requirements.
* Domain req from stakeholders and documentation are also discovered during this activity.
* There are several techniques that can be used for requirements discovery.

2) Requirements classification and organization:—

* This activity takes the unstructured collection of requirements, groups related requirements, and organizes them into coherent clusters.

③ Requirements prioritization and negotiation:—

* When multiple stake holders are involved, requirements will conflict
* This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation.
* Usually stakeholders have to meet to resolve differences and agree on compromise requirements.

④ Requirements specification:—

* In this activity requirements are doumented and are provided as input to the next spiral.
* Formal or informal requirements doument may be produced.

→ Requirements elicitation and analysis is an iterative process with continual feedback from each activity to other activities.

* Process cycle starts with requirements discovery and ends with the requirements documentation. In each iteration some kind of improvement will take place by the analyst.

Understanding stakeholders requirements is difficult for several reasons:—

1. Stakeholders often they don't know what they want from a computer system, they ~~dont~~ may find difficult to articulate what they want the system to do, They may make unrealistic demands, because they don't know what is feasible what is not feasible.
2. Stakeholders in a system naturally express requirements in their own terms and with implicit knowledge of their own work. Requirements engineers without experience in the customer's domain, may not understand the requirements.

* Stakeholders are the people related to the system. They may be end-users, managers, external stakeholders, They all should certify the acceptability of the system.

Example: Stakeholders for a Bank ATM

1. current bank customers who receives services from the system
2. Representatives from other Banks
3. Managers of the bank branches (obtain management infor from system)
4. Counter staff (who involved in the day-to-day running of the system)
5. Database administration (integrate system with bank customer db)
6. Bank security managers (ensure system will not pose a security hazard)
7. Bank's marketing department (intrested to use the system for marketing)
8. H/w & S/w maintenance engineers (upgrading of h/w & s/w)
9. National banking regulators (system confirms banking regulations)

In addition to stakeholders, requirements may also come from the application domain and from other systems that interact with the system being developing.

* Requirement sources (stakeholders, domain, systems) can all be represented as system Viewpoints

(1) Viewpoints:- Viewpoints are a way of structuring requirements to represent the perspectives of different stakeholders. Stakehold may be classified under different viewpoints.

Types of Viewpoints:-

1) Interactor viewpoints
2) Indirect viewpoints
3) Domain viewpoints.

1. Interactor Viewpoints :— people or other systems that interact directly with the system.

* These viewpoints provide detailed system requirements covering the system features and interfaces.
* In an ATM, the customer's and the account database are interactor viewpoints.

2. Indirect Viewpoints :— stakeholders who do not use the system themselves but who influence the requirements.

* These viewpoints are more likely to provide high-level organizational requirements and constraints.
* In an ATM, management and security staff are indirect viewpoints

3. Domain Viewpoints :—

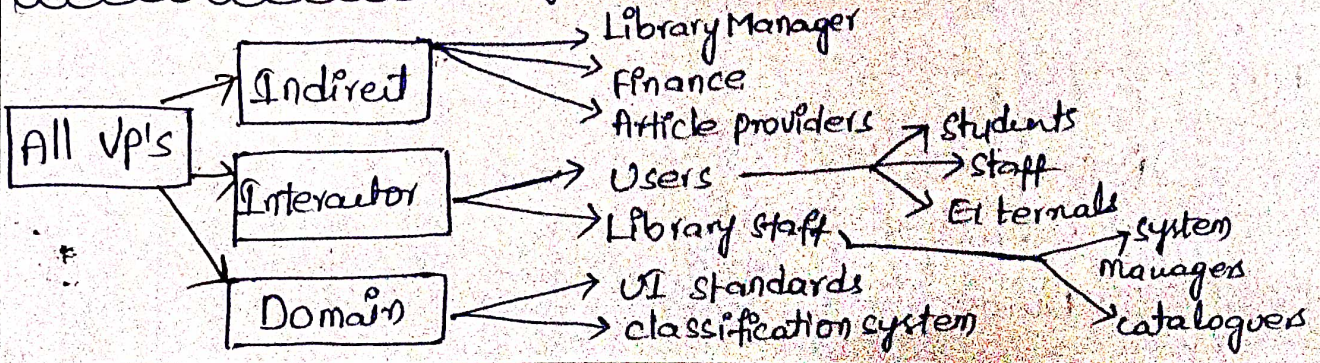
* Domain characteristics and constraints that ~~left~~ influence the requirements.
* These normally provide domain constraints that apply to the system.
* In an ATM, an example would be standards for inter-bank communications.

Viewpoints identification :—

Identify viewpoints using:

→ providers and receivers of the system services;
→ Systems that interact directly with the system being specified;
→ Regulations and standards.
→ Sources of business and non-functional requirements.
→ Engineers who have to develop and maintain the system.
→ Marketing and other business viewpoints.

LIBSYS viewpoint hierarchy :



(2) Interviewing :- Formal or informal interviews with system stakeholders are part of most req's engineering processes.

* Req's engineering team puts questions to the stakeholders about the system, that they currently use and the system to be developed.
* Requirements are derived from the answers to these questions.

Interviews may be of two types :-

1. closed interviews, where the stakeholders answers a predefined set of questions.
2. Open interviews, where there is no predefined agenda. The req's engineering team explore a range of issues with system stakeholders and hence develop a better understanding of the system

Interviews in practice will be like

1) Mix of closed and open-ended interviewing.
2) Interviews are good to conduct for better understanding of stakeholders views about the system. working with current system, issues with the current system.
3) Interviews are not good for ~~what they need from~~ understanding system domain requirements.

* Domain requirements completely related to technical information about the system. so, if interviews happened with stakeholders regarding domain req's, req's engineers cannot understand specific domain terminology.

Effective Interviewers must have two characteristics

1. Interviewers will not be open-minded, willing to listen to stakeholders and should not have pre-conceived ideas about req's
2. They should prompt the interviewee with a question or a proposal and should not simply expect themselves to respond to a

question such as 'what do you want"

(3) Scenarios :-

* Scenarios are real life examples real time examples than to abstract description. They can understand and analyze a scenario of how they might interact with a s/w system.
* requirements engineer gather information for this discussion to formulate the actual system requirements.
* Each scenario covers one or more possible interactions. Several scenarios have developed each of which provides different types of information at different levels of detail about the system.

Scenarios include :-

→ what the system and users except when the scenarios starts
* Description of normal flow of events
* Description of the starting situation
* Description of what can go wrong.
* Information about other concurrent activities.
* Scenarios may be written as text, implemented by diagrams, screen shots and so on.

(IX) Ethnography :-

* Software systems do not exist in isolation (separate) they are used in a social and organizational context and software system requirements may be derived or constrained by the context
* Satisfying these social and organizational requirements is often critical for the success of the system.

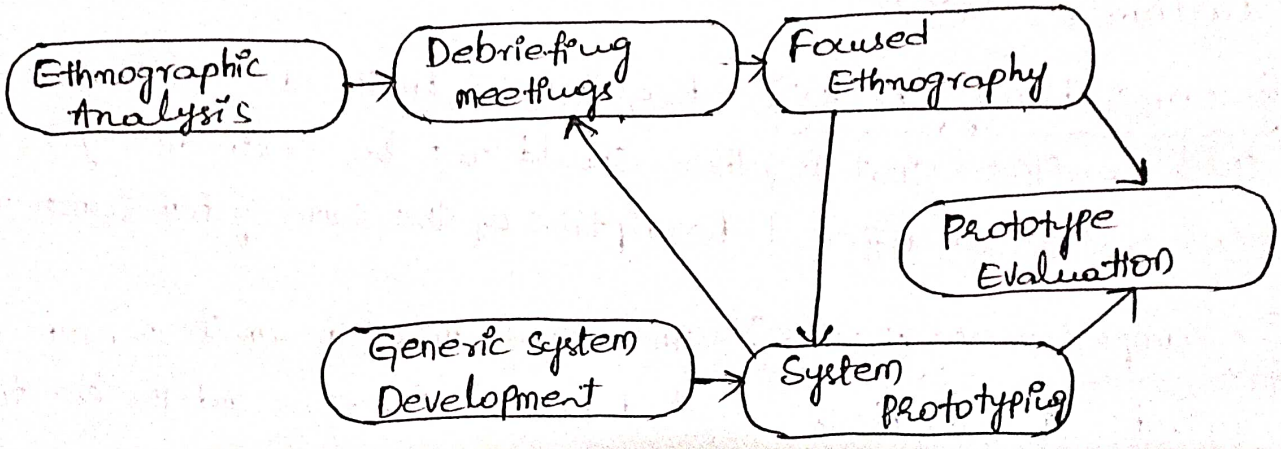
reason: Many s/w systems are delivered but never used becoz they do not take proper account of the importance of these requirements.

* Ethnography is an observational technique that can be used to understand social and organizational requirements.
* Value of ethnography is that it helps analysts discover implicit (internal) system requirements that reflect actual rather than formal processes in which people are involved.

Ethnography is effective at discovering two types of requirements.

* Requirements that are derived from the way in which people actually work rather than the way in process definitions say they ought to work.
* Requirements that are derived from cooperation & awareness of other people's activities. (Ex: air traffic controllers may use awareness of other controllers)

Ethnography and prototyping for requirement analysis.



* Ethnography can be combined with prototyping. The Ethnography informs the development of the prototype so that fewer prototype refinement cycles are required. Furthermore, the prototyping focuses the ethnography by identifying problems and questions that can then be discussed with the ethnographer.

=

(8X) Requirements Validation :- Requirements validation is the process of checking that requirements actually define the system that the customer really wants

* Requirements validation is important because errors in a requirements document can lead to extensive rework costs when these problems are discovered during development or after the system is in service.
* During the requirements validation process, different type of checks should be carried out on the requirements in the requirements document.
* These checks include :

1. Validity checks :- A user may think that a system is needed to perform certain functions. It checks "Does the system provide the services or functions which really best support the customer's need?

2. Consistency checks :- Requirements in the document should not conflict. That is, there should not be contradictory constraints or different descriptions of the same system function.

3. Completeness checks :- The requirements document should include requirements that define all functions and the constraints intended by the system user.

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

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

There are a number of requirements validation techniques available.

1. Requirements reviews :- The requirements are analyzed systematically by a team of reviewers who check for errors and inconsistencies.

2. Prototyping :- In this approach, an executable model of the system in question is demonstrated to end-users and customers.

3. Test-case generation :- Requirements should be testable. If the tests for the requirements are devised as part of the validation process, then often reveals requirements problems.

=

(XI) Requirements Management

The requirements for large software systems are always changing. The reason for this is, during the software process, the stakeholder's understanding of the problem is constantly changing. The system requirements must

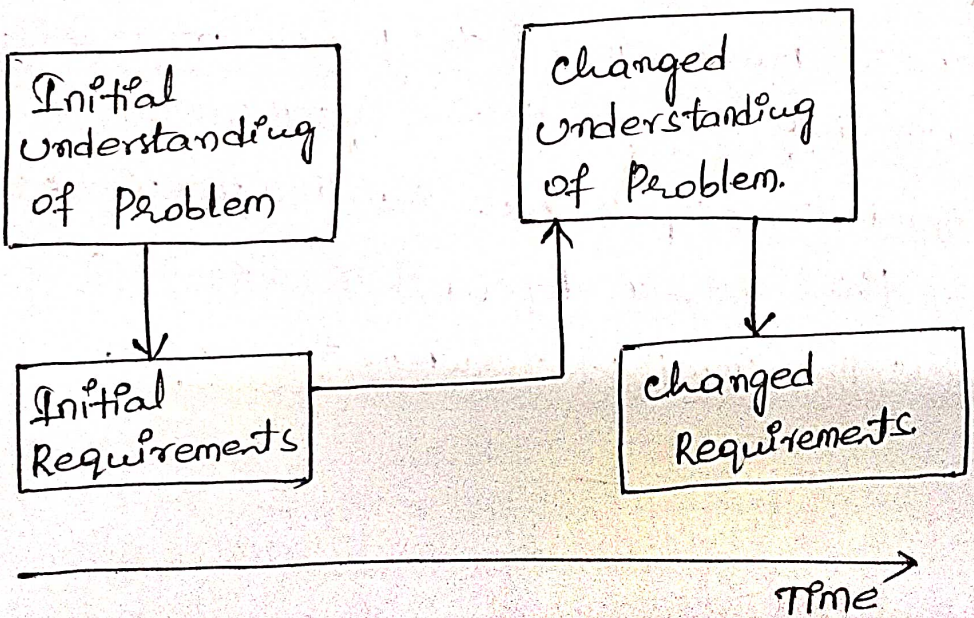


Fig: Requirements evolution.

=

then also evolve to reflect this changed problem view.

There are several reasons why change is inevitable:

1. The business and technical environment of the system always changes after installation. New hlw may be introduced, it may be necessary to interface the system with other systems, business priorities may change, and new legislation and regulations may be introduced that the system must include.

2. The people who pay for a system and the users of that system are rarely the same people. System customers impose requirements because of organizational & budget constraints. These may conflict with end-user requirements after delivery, so new features may have to be added for user support.

3. Large systems usually have a diverse user community. Many users having different requirements and priorities that may be conflicting. The final system requirements discovered would be that the balance of support given to different users has to be changed.

→ Requirements management is the process of understanding and controlling changes to the system requirements. There is a need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirement changes.

(i) Requirements change Management :-

Requirements change management should be applied to all proposed changes to a system's requirements after the requirements doument has been approved.

* change management is essential because you need to decide if the benefits of implementing new requirements are justified by the costs of implementation.

* There are three principal stages to a change management Procen.

1) problem analysis and change specification : The process starts with an identified requirements problem or, Sometimes, with a specific change proposal. During this stage, the problem or the change proposal is analyzed to check that it is valid.

2. change analysis and costing :- The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. The cost of making the change is estimated both in terms of modifications to the requirements doument and if appropriate, to the system design and implementation. once the analysis is completed, a decision is made whether or not to proceed with the requirements change.

3. change implementation : Requirements document should be organized in such a way that it allows to make chages to it without extensive rewriting and reorganization.

* As with programs changeability in douments is achieved by minimizing external references and making the doument section as modular as possible.

=

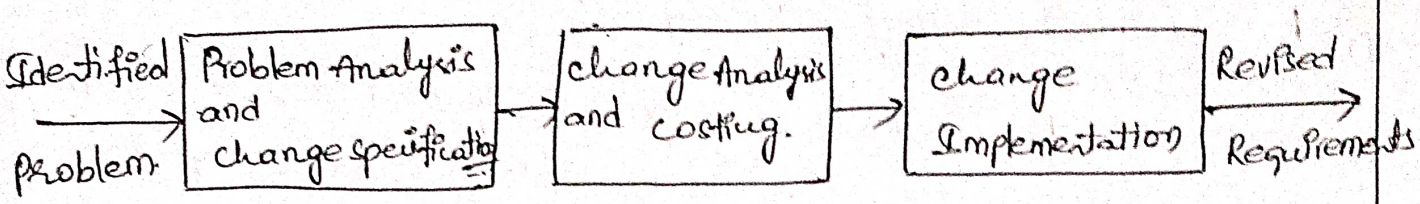


Fig: Req's change Management

(ii) Requirements management planning :-

planning is an essential first stage in requirements management process. The planning stage establishes the level of requirements management detail that is required.

* During the requirements management stage, you have to decide on:

1. Requirements identification :- Each requirement must be uniquely identified so that it can be cross-referenced with other requirements and used in traceability assessments.

2. A change management process :- This is the set of activities that assess the impact and cost changes.

3. Traceability policies :- These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.

4. Tool support Requirements Management involves the Processing of large amounts of information about the requirements. Tools that may be used range from specialist requirements management systems to spreadsheets and simple database system

UNIT - III

7 Design Engineering

7.1 : Design Process and Design Quality

Explain the term - Software design
[JNTU : Part A, Marks 2]

: Software design is model of software which lates the requirements into finished software duct in which the details about software data uctures, architecture, interfaces and components are necessary to implement the system are given.

What do you mean by design quality ?
[JNTU : Part A, Dec.-16, Marks 3]

: • Design quality is the level of effectiveness of design function in determining a product's tional requirements that can be converted into shed product.

e goal of any software design is to produce high ality software.

order to evaluate quality of software there uld be some predefined rules or criteria that need to be used to assess the software product. Such criteria serve as characteristics for good design.

Q.3 Elaborate on software design engineering in detail.
[JNTU : Part B, Dec.-10, ECE, Marks 16]

Ans. : • During software engineering process, the first task is to identify and analyze the requirements. Based on this analysis the software design is developed. This design serves as a basis for code generation and testing. Hence software design is an intermediate process which translates the analysis model into design model.

- The four types of elements of **analysis model** are scenario based elements, class based elements, behavioral elements, and flow oriented elements.
- During the scenario based analysis various models such as use case diagrams, activity diagrams or swimlane diagrams are created. During the class based analysis the class diagrams are created. During flow oriented analysis the Data flow

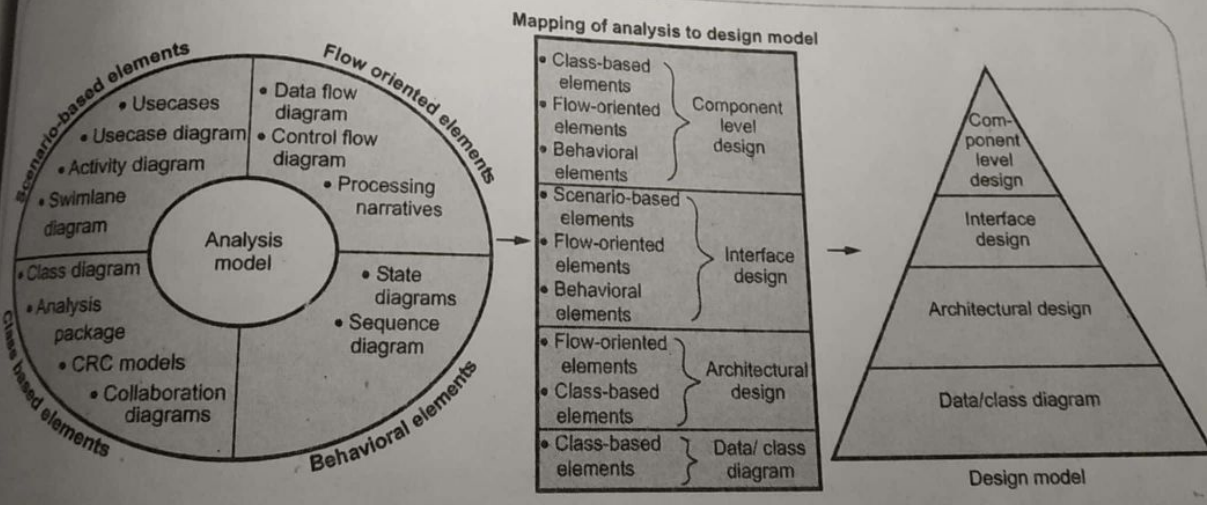


Fig. Q.3.1 Translating analysis model into the design model

diagrams are created. Behavioral analysis can be done using state diagrams and sequence diagrams.

- The **class based elements** of analysis model are used to create **class diagrams**.
- The **flow oriented elements** and **class based elements** are used to create **architectural design**.
- The **scenario based elements, flow oriented elements and behavioral elements** are used to create **Interface Design**. The interface design describes how software communicates with systems.
- The **class based elements, flow oriented elements and behavioral elements** are used to create **component level design**. This design transforms the structural elements of software architecture into procedural description of software module. (Refer Fig. Q.3.1).

Q.4 State and explain various activities in software design. And also discuss the importance of design phase. **[JNTU : Part B, Dec.14, Marks 7]**

Ans. : Activities in Software Design - Refer Q.3.

Importance of software design phase :

- Software design is very important for assessing the quality of software. Because design is the only way that we can accurately translate the user requirements into the finished software product.
- Without software design, it is difficult to test the software product. Not only this, but for making a small change in the software - it is the software design which helps us to make the necessary changes without disturbing other part of the software.

Q.5 Describe the design process in software development. **[JNTU : Part A, Feb.-10,Marks 3]**

Ans. : • Software design is an iterative process using which the user requirements can be translated into the software product.

- At the initial stage of the software is represented as an **abstract view**.
- During the sub-sequent iterations data, functional and behavioral requirements are traced in detail. That means at each iteration the refinement is made to obtain lower level details of the software product.

Q.6 What are the characteristics and criteria for design ? **[JNTU : Part A, Feb.-10, Marks 3]**

Ans. : 1.The good design should **implement all the requirements** specified by the customer. Even if there are some implicit requirements of the software product then those requirements should also be implemented by the software design.

2. The design should be **simple** enough so that the software can be understood easily by the developers, testers and customers.
3. The design should be comprehensive. That means it should provide complete picture of the software.

7.2 : Design Concepts

Q.7 Explain about various design concepts in detail.
[JNTU : Part B, Dec.-12, Marks 7,May-13, Marks 5]

Ans. : The software design concept provides a framework for implementing the right software.

1. **Abstraction :** Software design occurs at different levels of abstraction. At each stage of software design process levels of abstractions should be applied to refine the software solution. At the higher level of abstraction, the solution should be stated in broad terms and in the lower level more detailed description of the solution is given.
2. **Modularity :** The software is divided into separately named and addressable components that called as modules. Monolithic software is hard to grasp for the software engineer, hence it has now become a trend to divide the software into number of products.
3. **Architecture :** Architecture means representation of overall structure of an integrated system. In architecture various components interact and the data of the structure is used by various components.
4. **Refinement :** Refinement is actually a process of elaboration.The process of program refinement is analogous to the process of refinement and partitioning that is used during requirements analysis.

2.	Content coupling is also known as **pathological coupling**.	Common coupling is also known as **global coupling**.
3.	This type of coupling **does not require any global data or global variable** for communication.	This type of coupling is based on the communication using **global data areas**.
4.	The **major drawback** of content coupling is that - if we want to reuse one component we need to import all the components that are coupled with the component being reused.	The **major drawback** of common coupling is that - modules in the common coupling are tightly coupled. A fault in one module using global data may show up in another module because global data may be updated by any module at any time. This greatly affects the reusability of modules.
5.	The access to internal structure is of main concern in this kind of communication.	The access to global data is of main concern in this kind of communication.
6.	In this coupling, component directly modifies another's data.	In common coupling, it is difficult to determine all the components that affect a data element.
7.	**Example** - Part of Program that searches for the entry of particular employee. when the module does not find the entry for the employee, it directly adds employee modifying the contents of data structure containing employee data.	**Example** - A process control component that maintains the status of operations in a global data store. This control block gets data from multiple sources or supplies data to multiple sinks. Each source process writes the status of its operation directly to the global data store.

Q.13 Define and explain about coupling and cohesion. Also differentiate between them.
[JNTU : Part B, June-14, Marks 7]

Ans. :

Sr. No.	Coupling	Cohesion
1.	Coupling represents how the **modules are connected** with other modules or with the outside world.	In cohesion, the cohesive module performs **only one thing.**
2.	With coupling **interface complexity** is decided.	With cohesion, data **hiding** can be done.
3.	The goal of coupling is to achieve **lowest coupling.**	The goal of cohesion is to achieve **high cohesion.**
4.	Various **types** of couplings are - Data coupling, Control coupling, Common coupling and Content coupling.	Various **types** of cohesion are - Coincidental cohesion, Logical cohesion, Temporal cohesion, Procedural cohesion and Communicational cohesion.

Q.14 Define and explain about different types of cohesion
[JNTU : Part B, Dec.-16, Marks 5]

Ans. : • With the help of cohesion the **information hiding** can be done.

- A cohesive module performs only **"one task"** in software procedure with little interaction with other modules. In other words cohesive module performs only one thing.
- Different types of cohesion are :

1. **Coincidentally cohesive** - The modules in which the set of tasks are related with each other loosely then such modules are called coincidentally cohesive.
2. **Logically cohesive** - A module that performs the tasks that are logically related with each other is called logically cohesive.
3. **Temporal cohesion** - The module in which the tasks need to be executed in some specific time span is called temporal cohesive.
4. **Procedural cohesion** - When processing elements of a module are related with one another and must be executed in some specific order then such module is called procedural cohesive.

5. **Pattern :** The design pattern can be defined as - It is a named sugget(something valuable) of insight which conveys the essence of proven solution to a recurring problem within a certain context.
6. **Information Hiding :** The term information hiding means the modules are designed in such a way that information contained in one module cannot be accessible to the other module.
7. **Functional Independence :** By using functional independence functions may be compartmentalized and interfaces are simplified. Independent modules are easier to maintain with reduced error propagation.
8. **Refactoring :** Refactoring is necessary for simplifying the design without changing the function or behavior. Fowler has defined refactoring as "the process of changing a software system in such a way that the external behavior of the design do not get changed, however the internal structure gets improved".

Q.8 Justify how modular design is an effective design method ?
[JNTU : Part B,Feb.-10, Dec.-10,Marks 8]

Ans. : Meyer defines five criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system :

1. **Modular decomposability** : A design method provides a systematic mechanism for decomposing the problem into sub-problems. This reduces the complexity of the problem and the modularity can be achieved.
2. **Modular composability** : A design method enables existing design components to be assembled into a new system.
3. **Modular understandability** : A module can be understood as a standalone unit. It will be easier to build and easier to change.
4. **Modular continuity** : Small changes to the system requirements result in changes to individual modules, rather than system-wide changes.
5. **Modular protection** : An aberrant condition occurs within a module and its effects are constrained within the module.

Q.9 Define coupling and cohesion.
[JNTU : Part B, Dec.-10, Marks B, Dec.-17, Marks 2]

Ans. : Coupling : Coupling represents how the module can be **connected with other module or with the outside world.**

Cohesion : Cohesion is a measure that represents that a **module performs only one task** in software procedure with little interaction with other modules.

Q.10 What is data abstraction ? Give an example.
[JNTU : Part A, Marks 3]

Ans. : In **data abstraction** the collection of data objects is represented. For example for the procedure search the data abstraction will be record. The record consists of various attributes such as record ID, name, address and designation.

Q.11 Differentiate between procedural and data abstractions.
[JNTU : Part B, April-11, Marks 8]

Ans. : The procedural abstraction gives the named sequence of instructions in the specific function. That means the functionality of procedure is mentioned by its implementation details are hidden. For example : Search the record is a procedural abstraction in which implementation details are hidden (i.e. Enter the name, compare each name of the record against the entered one, if a match is found then declare success. Otherwise declare 'name not found')

In data abstraction the collection of data objects is represented. For example for the procedure search the data abstraction will be record. The record consists of various attributes such as record ID, name, address and designation.

Q.12 Differentiate between content and common coupling. **[JNTU : Part B, Dec.-11, Marks 7]**

Ans. :

Sr. No.	Content Coupling	Common Coupling
1.	Content coupling occurs between two modules when one module refers to the **internals** of the other module.	Common coupling occurs when modules communicate using **global data areas.**

5. **Communicational cohesion** - When the processing elements of a module share the data then such module is communicational cohesive.

• The goal is to achieve high cohesion for modules in the system.

Q.15 Define and explain about different types of coupling [JNTU : Part B, Marks 5]

Ans. : Various types of coupling are :

i) **Data coupling** - The data coupling is possible by parameter passing or data interaction.

ii) **Control coupling** - The modules share related control data in control coupling.

iii) **Common coupling** - In common coupling common data or a global data is shared among the modules.

iv) **Content coupling** - Content coupling occurs when one module makes use of data or control information maintained in another module.

Q.16 Discuss the advantages and disadvantages of modularization [JNTU : Part B, May-08, Marks 7]

Ans. : Advantages :

(1) Due to modularization algorithm can be **understood** easily.

(2) Many programmers can **work on different modules** at the same time which **saves the time.**

(3) **Testing** of **each module** can be done thoroughly.

(4) Modules can be **reused** for similar kind of projects.

(5) **Library programs** can be inserted in separate module.

Disadvantages :

(1) **Documentation** of each module need to be maintained.

(2) It can create problems when modules are **linked together** because each link need to be tested properly.

Q.17 Discuss about refactoring in detail [JNTU : Part B, Dec.-11, Marks 7]

OR What is refactoring? Why is it done? [JNTU : Part B,May 08, Marks 7]

Ans. : • Refactoring is necessary for simplifying the design without changing the function or behaviour.

• **Fowler** has **defined refactoring** as "The process of changing a software system in such a way that the external behavior of the design do not get changed, however the internal structure gets improved".

Benefits of refactoring are -

i) The **redundancy** can be achieved.

ii) **Inefficient algorithms** can be eliminated or can be replaced by efficient one.

iii) Poorly constructed or **inaccurate data structures** can be removed or replaced.

iv) Other **design failures** can be rectified.

The decision of refactoring particular component is taken by the designer of the software system.

Q.18 Discuss the statement, "Abstraction and refinement are complementary concepts. [JNTU : Section B, June-14, Marks 8]

Ans. : Abstraction and refinement are complementary concepts. The major difference is that - in the abstraction low-level details are suppressed.

Refinement helps the designer to elaborate low-level details.

Q.19 "Modularity is the single attribute of the software that allows a program to be intellectually manageable" - How this is true ? [JNTU : Section A, Marks 2]

Ans. : This is quoted by Glenford Mayers. Consider that there is a large program composed of single module. Such a program cannot be grasped by reader. The control paths, span of reference, number of variables and overall complexity of such a program is beyond understanding. On the other hand if the program is divided into certain number of modules then overall complexity of the program gets reduced. The error detection and handling can be done effectively. Also changes made during testing and maintenance become manageable. Hence it is true that "Modularity is the single attribute of the software that allows a program to be intellectually manageable"

Q.20 Differentiate between : abstraction and modularization. [JNTU : Section A, Marks 2]

Ans. : • Abstraction is a software design concept which is applied to refine software solution. In the process of abstraction only necessary information is abstracted form requirement analysis to create the software solution in broad terms. While moving through different levels of abstraction the procedural and data abstraction are prepared. During abstraction the procedural and data objects are prepared.

• Modularity is a design concept in which requirements are divided into separately named and addresable components called as modules. Modularity in design reduces complexity and helps in easier implementation.

Q.21 Discuss the relationship between the concept of information hiding as an attribute of effective modularity and the concept of module independence ? [JNTU : Section B, Feb.-10, CSE/IT/CS & SE, Marks 8]

Ans. : Information hiding can be related to both coupling and cohesion concepts.

For reducing the coupling only those modules are linked together that are absolutely needed. This reduces the coupling between modules. Similarly information is isolated so that the functionalities can be isolated. This improves the cohesion of individual modules.

Information hiding implies that effective modularity can be achieved by defining a set of independent modules that communicate with one another only that information necessary to achieve software function. Using abstractions the procedural entities are defined. This is helpful for testing and maintenance.

The concept of module independence helps in achieving modularity, abstraction and information hiding. Independent modules are easier to maintain and test. Reusability of such modules is also possible.

Q.22 What do you understand by the term Design classes ? [JNTU : Part A, Marks 2]

Ans. : Design classes are defined as the classes that describe some elements of the problem domain, in which the problem is viewed from user's point of view.

Q.23 What are the goals of design classes ? [JNTU : Part A, Marks 2]

Ans. : The **goal** of design classes is :

1. To refine the analysis classes by providing the **detail design**. Using detailed design further implementation is carried out.
2. To create new set of classes for implementing the core requirements of the software.

Q.24 What type of design classes does the designer create ? Explain about the "well formed" design class. [JNTU : Part B, May-09, Marks 8]

Ans. : 1. User Interface Class : The user interface class defines all the interactions with the system. The user interface classes is basically a **visual representation**. It is also called Human Computer Interface (HCI)

2. Business Domain Class : Business domain classes are the refinement of analysis classes. These classes identify the **attributes and services** that are needed to implement some elements of **business domain.**

3. Process Class : Process class is used business domain. It define processes required by business domain.

4. Persistent Class : These classes represent the data store such as databases which will be retained as it is even after the execution of the software.

5. System Class : These classes are responsible for software management and control functions that are used for system operation.

Each class must be **well formed design class.**

Q.25 Write any three desirable characteristics of design classes [JNTU : Part A, Marks 3]

Ans. : • Complete and Efficient

A design class must be properly **encapsulated** with corresponding attributes and methods. Design class must contain all those methods that are sufficient to achieve the intent of the class.

(2) Interface Elements
(3) Data Design Elements
(4) Component Level design Elements

Design Model :

- The **process dimension** denotes the progress of design model that occurs due to various software tasks that get executed as the part of software process.
- The **abstract dimension** represents in how much detail the analysis model is transformed into design model.
- In Fig. Q.26.1 the **dashed line** shows the boundary between analysis and design model.
- In both the analysis and design models the same UML diagrams are used but in analysis model the these diagrams are abstract and in design model these diagrams are refined and elaborated. Moreover in design model the implementation specific details are provided.
- Along the horizontal axis various elements such as architecture element, interface element, component level elements and deployment level elements are given. It is not necessary that these elements have to be developed in sequential manner. First of all the preliminary architecture design occurs then interface design and component level design occur in parallel. The deployment level design ends up after the completions of complete design model.

Q.27 Explain the concept of data design element.

GTU : Part B, Marks 5]

Ans. : • The data design represents the **high level of abstraction**.

- This data represented at data design level is **refined gradually** for implementing the computer based system.
- The data has great impact on the architecture of software systems. Hence structure of data is very important factor in software design.
- Data appears in the form of **data structures and algorithms at the program component level.**

- At the **application level** it appears as the **database** and at the **business level** it appears as **data warehouse and data mining.**
- Thus data plays an important role in software design.

Q.28 Explain the architectural Design Element

GTU : Part B, Marks 5]

Ans. : • The architectural design gives the layout to overall view of the software.

- Architectural model can be built using following sources -
- **Data flow models** or **class diagrams.**
- Information obtained from application domain
- Architectural **patterns and styles.**
- The **architectural style** is a pattern for creating system architecture for given problem. Commonly used architectural styles are -

(1) Data centered architecture
(2) Data flow architecture
(3) Call and return architecture
(4) Object oriented architecture
(5) Layered architecture.

- The **architectural pattern** is basically an approach for handling behavioral characteristics of software systems.

Q.29 Explain the interface level design elements

GTU : Part B, Marks 5]

Ans. : Interface Design represents the **detailed design** of the software system. In interface design **how information flows** from one component to other component of the system is depicted. Typically there are **three types of interfaces -**

User interface : By this interface user interacts with the system. For example - GUI

External interface : This is the interface of the system components with the external entities. For example - Networking.

Internal Interface : This is an interface which represents the inter component communication of the system. For example - Two classes can communicate with each other by operations or by passing messages

Fig. Q.29.1 Interface design elements

- **Primitiveness**

Methods associated with one particular class must perform unique service and the class should not provide another way to implement the same service. There should not be conflicting methods.

- **High Cohesion**

A cohesive designed class must posses small, focused set of responsibilities and these responsibilities must be associated with all the attributes of that class.

- **Low Coupling**

Design classes must be collaborated with manageable number of classes. If one design class is collaborated with all other design classes then the implementation, testing and maintenance of the system becomes complicated. The law of Demeter suggests that the one particular design class should send messages to the methods of neighbouring design classes only.

7.3 : The Design Model

Q.26 What are the major elements of the design model ? Explain the abstraction dimensions and process dimensions of the analysis and design model.

GTU : Part B, May-09, Feb.-10, Marks 16, GTU May-13, Marks 5]

Ans. : The major element of design model are -

(1) Architecture Elements

Fig. Q.26.1 Dimension of design model

Q.30 Explain component level design elements.
GE(JNTU : Part B, Marks 5)

Ans. : The component level design is more detailed design of the software system along with the specifications. The component level design elements describe the internal details of the component. In component level design all the local data objects, required data structures and algorithmic details and procedural details are exposed.

Fig. Q.30.1 represents that component order makes use of another component form.

Fig. Q.30.1 Components

The **order** is dependent upon the component form. These two objects can be interfaced with each other.

Q.31 Explain deployment level design elements
GE(JNTU : Part B, Marks 6)

Ans. : The deployment level design elements indicate how software functions and software subsystems are assigned to the physical computing environment of the software product. For example web browsers may work in mobile phones or they may run on client PC or can execute on server machines. Refer Fig. Q.31.1.

7.4 : Pattern Based Software Design

Q.32 Explain about pattern based software design in detail
GE(JNTU : Part B, Dec.-16, Marks 5)

Ans. : • In software designing process, certain design patterns can be created and used to solve the problems.

• Sometimes instead of creating new design patterns the existing pattern that satisfies the design needs can be reused.

• Mature engineering branches like mechanical and electrical engineering make use of design patterns. For example - to solve some particular problem in electrical engineering the integrated circuit design created. This IC design may very based on

Fig. Q.31.1 Deployment diagram

nature of the problem. Thus design can be viewed as an activity in which the same design pattern can be reused with little or no change in the configuration.

- The attributes and operations of the pattern can be inherited.
- The design pattern can be used using following design pattern template.

Template for Design Pattern	
Name :	The name of the design pattern is given initially. The name must be short and expressive
Purpose :	The purpose of the design pattern is given in this section.
Known-as :	If any synonym for the pattern is existing then it should be mentioned.
Motivation :	It denotes nature of the problem along with supporting example.
Applicability :	It describes the design situation in which the pattern is applicable.
Structure :	It describes the classes that are required to implement the problem
Participants :	It includes the responsibilities of the classes that are used in the pattern.
Collaborations :	It describes all the collaborating classes that share some responsibilities.
Consequences :	It describe various design forces of the pattern.
Cross References :	If any related design patterns exist then those are described under this section.

Q.33 Explain various types of patterns in Design.

Ans. : Various types of design patterns are -

- **Architectural Pattern**

Architectural design pattern are those design patterns that depict the overall structure of software. These patterns also indicate the relationship among the components and define the rules for inter-relationship between the elements of the software architecture.

- **Design Pattern**

The design patterns are used to represent the design of specific elements, relationships among the components and component to component communication.

- **Idioms**

These are also called as **coding patterns**. These patterns are language specific and are used to implement an algorithmic elements, interface protocols or methods used for component component communication.

END...4

- **Internal data** : It is the own workspace.
- The two modules can be connected to each other by a connector as shown below.

Fig. Q.3.1 Connector

- The module uses data and flags. The data is processed by different modules. The flag is used as a control signal. It can be set or reset.

For example if we have to modules one for getting the employee detail(calling) and another module is for finding the employee name(called). Then the caller module will sent the data as employee's ID and using that ID the called module will find the name of employee. If the employee ID is valid then that message will be given by the called module to a caller module. The use of data and flag can be as shown below. (Refer Fig. Q.3.2)

- The iterations and decisions on a structured chart is as shown below. (Refer Fig. Q.3.3.)

Fig. Q.3.2 Use of flag

Fig. Q.3.3 Interaction

UNIT - III

8 Creating an Architectural Design

8.1 : Software Architecture

Q.1 What is software architecture ? Why is it important ? **[SPPU : Part A, Marks 3]**

Ans. : Software Architecture is a structure of systems which consists of various components, externally visible **properties** of these components and the **inter-relationship** among these components.

Importance of software architecture

There are three reasons why the software architecture is so important ?

1. Software architecture gives the representation of the computer based system that is to be built. Using this system model even the **stakeholders** can take **active part** in the software development process. This helps in clear specification/understanding of requirements.
2. Same **early design decisions** can be taken using software architecture and hence system performance and operations remain under control.
3. The software architecture gives a clear cut idea about the computer based system which is to be built.

Q.2 Differentiate between Software architecture and software design process.

Ans. :

Sr. No.	Software architecture	Software design process
1.	**Software architecture** deals with high level concepts that do not include any details as how they will be implemented.	**Software design process** deals with high level concepts and is applied to the concrete details so that software can be implemented.
2.	It deals with the issues such as what kind of data storage is present, how do modules interact with each other, what recovery system is in place and so on.	It basically deals with the individual components, what are the functionalities or modules, what kind of tasks these modules should perform and so on.
3.	Software architecture represents the **bigger** picture which deals with the choice of software framework, languages, scope, goals and high level methodologies.	Software design represents the **smaller** picture which deals with what are different modules of each subsystems and how will they interact with each other.
4.	For building the software architecture **architectural patterns** are applied.	For building the software design the **design patterns** are applied.

Q.3 What is software architecture ? Explain about structure chart with an example. **[SPPU : Part B, Feb.-10, Dec.-10, Marks 8]**

Ans. : Software Architecture - Refer Q.1.

Structure chart -

- The structure chart is a principle tool of structured design.
- The basic element in the structured chart is module. Module is defined as a collection of program statement with four attributes.
- **Input and output** : What the module gets from the invoker is called input and what the receiver gets from the module is called output.
- **Function** : The function processes the input and produces the output.
- **Mechanics** : The code or the logic by which the function is carried out.

Q.6 Write and explain the steps used in data design. **[JNTU : Part B, Marks 5]**

Ans. : 1. *Apply systematic analysis on data*

Represent data objects, relationships among them and data flow along with the contents.

2. *Identify data structures and related operations*

For the design of efficient data structures all the operations that will be performed on it should be considered.

3. *Establish data dictionary*

The data dictionary explicitly represents various data objects, relationships among them and the constraints on the elements of data structures.

4. *Defer the low-level design decisions until late in the design process*

Major structural attributes are designed first to establish an architecture of data. And then low-level design attributes are established.

5. *Use information hiding in the design of data structures*

The use of information hiding helps in improving quality of software design. It also helps in separating the logical and physical views.

6. *Apply a library of useful data structures and operations*

The data structures can be designed for reusability. A use of library of data structure templates (called as abstract data types) reduces the specification and design efforts for data.

7. *Use a software design and programming language to support data specification and abstraction*

The implementation of data structures can be done by effective software design and by choosing suitable programming language.

8.3 : Architectural Styles and Patterns

Q.7 What is architectural style ? **[JNTU : Part A, May-07, Marks 02]**

Ans. : The architectural model or style is a pattern for creating the system architecture for given problem. However, most of the systems are heterogeneous and do not follow single architectural style.

Q.8 List out some commonly used architectural styles. **[JNTU : Part A, Marks 3]**

Ans. : The commonly used architectural styles are,

1. Data centered architectures.
2. Data flow architectures.
3. Call and return architectures.
4. Object oriented architectures.
5. Layered architectures.

Q.9 Explain about architectural pattern in detail. **[JNTU : Part B, April-11, Marks 8]**

Ans. : The architectural pattern is basically an approach for handling **behavioural characteristics** of software systems. Following are the architectural pattern domains.

1. Concurrency : Concurrency means handling multiple tasks in parallel. For example in operating system, **multiple tasks** are executed in parallel. Hence concurrency is a pattern which represents that the system components can interact with each other in parallel. The benefit of this pattern is that system **efficiency** can be achieved.

2. Persistence : Continuity in the data can be maintained by the persistence pattern. In other words the data used in earlier execution can be made available further by storing it in files or in databases. These files/databases can be modified in the software system as per the need. In **object oriented** system the values of all attributes various operations that are to be executed are persistent for further use. Thus broadly there are two patterns. **i) Database management pattern ii) Application level pattern.**

3. Distribution : Distribution pattern refers to the way in which the system components communicate with each other in distributed systems. There are two major problems that occur in distribution pattern

- The nature of interconnection of the components.
- The nature of communication.

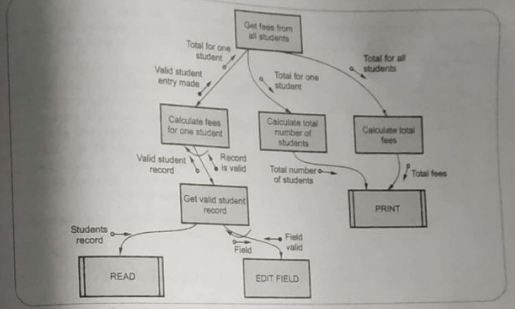


Fig. Q.3.4 Structure chart

- The example of a structure chart is as given below. (Refer Fig. Q.3.4)

Q.4 Discuss why software architecture play important role during development and discuss various architectural terminology **[JNTU : Part B, Dec.-10, 12, Mark 7]**

Ans. : Software Architecture Importance : Refer Q.1.

A computer-based system (software is part of this system) exhibits one of the many available architectural styles. Every architectural style describes a system category that includes the following -

(1) **Components :** Computational components such as clients, server, filter, and database to execute the desired system function.

(2) **Connectors :** A set of connectors such as procedure call, events broadcast, database protocols, and pipes to provide communication among the computational components.

(3) **Constraints :** Constraints to define integration of components to form a system.

(4) **Semantic model :** A semantic model, which enable the software designer to identify the characteristics of the system as a whole by studying the characteristics of its components.

8.2 : Data Design

Q.5 What is data design ? Explain with an example. **[JNTU : Part B, May-09, Dec.-14, Marks 7]**

Ans. :

- Data design is basically the model of data that is represented at the high level of abstraction.
- The data design is then progressively refined to create implementation specific representations.
- Various elements of data design are,
  - **Data object** - The data objects are identified and relationship among various data objects can be represented using entity relationship diagrams or data dictionaries.
  - **Databases** - Using software design model, the data models are translated into data structures and databases at the application level.
  - **Data warehouses** - At the business level useful information is identified from various databases and the data warehouses are created. For extracting or navigating the useful business information stored in the huge data warehouse then data mining techniques are applied.

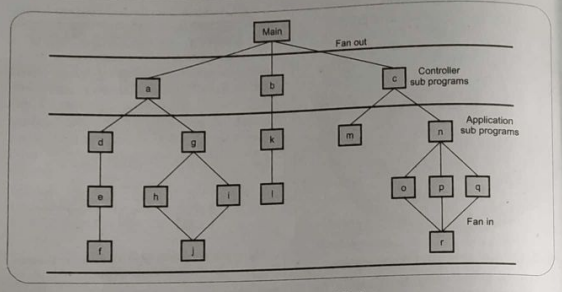


Fig. Q.11.1 Call and return architecture

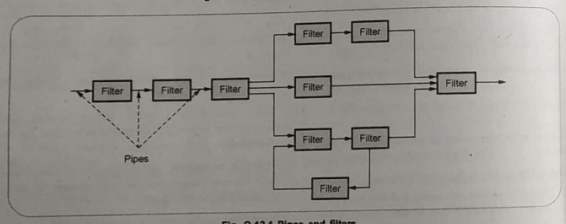


Fig. Q.12.1 Pipes and filters

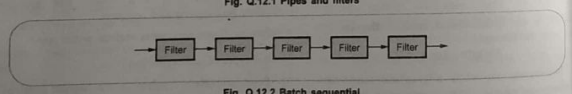


Fig. Q.12.2 Batch sequential

If the data flow degenerates into a single line of transforms, it is termed as batch sequential.

In this pattern the transformation is applied on the batch of data. (Refer Fig. Q.12.1.)

Q.13 Explain layered architecture in detail.
[JNTU : Part B, Marks 5]

Ans. : The layered architecture is composed of different layers. Each layer is intended to perform specific operations so machine instruction set can be generated. Various components in each layer perform specific operations.

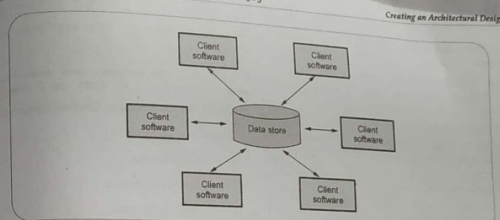


Fig. Q.10.1 Data centered architecture

These problems can be solved by other pattern called **broker pattern.** The broker pattern lies between server and client components so that the client server communication can be established properly. When client want some service from server, it first sends message to broker. The broker then conveys this message to server and completes the connection. Typical example is CORBA. The CORBA is a distributed architecture in which broker pattern is used.

Q.10 Explain data centered architectural style.
[JNTU : Part B, Marks 5]

Ans. : In this architecture the data store lies at the center of the architecture and other components frequently access it by performing add, delete and modify operations. The client software requests for the data to central repository. Sometime the client software accesses the data from the central repository without any change in data or without any change in actions of software actions.

Data centered architecture posses the property of interchangeability. Interchangeability means any component from the architecture can be replaced by a new component without affecting the working of other components.

In data centered architecture the data can be passed among the components.

In data centered architecture,

Components are : Database elements such as tables, queries.

Communication are : By relationships.

Constraints are : Client software has to request central data store for information. Refer Fig. Q.10.1.

Q.11 Explain Call and Return Architecture style in detail.
[JNTU : Part B, Marks 5]

Ans. : The program structure can be easily modified or scaled. The program structure is organized into modules within the program. In this architecture how modules call each other. The program structure decomposes the function into control hierarchy where a main program invokes number of program components.

In this architecture the hierarchical control for call and return is represented. (Refer Fig. Q.11.1 on next page.)

Q.12 What data Flow architecture style ? Explain.
[JNTU : Part B, Marks 5]

Ans. : In this architecture series of transformations are applied to produce the output data. The set of components called **filters** are connected by **pipes** to transform the data from one component to another. These filters work independently without a bothering about the working of neighbouring filter.

3) **Control unit or controller :** Controllers are the entities that are useful for controlling behaviour of the system. For example, in temperature control system. When the temperature exceeds beyond some threshold value alarming and dis-alarming system is required. Such a system acts as a controller or control unit.

4) **Indicator or output :** It represents the generic output functionalities. For example, monitoring system of any computer based system acts as an indicator.

- In software engineering archetype is a number of **major components** that are used to describe the system which we want to build.

8.5 : Assessing Alternative Architectural Designs

Q.19 List out two alternative approaches used for architecture design. **[JNTU : Part A, Marks 2]**

Ans. : There are two alternative approaches used for architecture designs -

1. The first approach is to apply iterative method to access design trade-offs.
2. The pseudo quantitative technique for accessing design quality.

Q.20 Discuss various criteria for accessing architectural design. **[JNTU : Part B, April-11, Marks 8]**

Ans. : Two criteria for accessing architectural Design are -

1. **Architecture Trade-off Analysis Method :** This method has an iterative evaluation process.

Various design analysis activities that are conducted are -

1. ***Collect Scenarios :*** Use cases are developed for representing the system from user's point of view. Hence a set of use cases help in collecting the scenarios.
2. ***Elicit requirements, constraints and environment description :*** This information is obtained during requirement analysis. Customer, user and stakeholders must specify the requirements

3. ***Describe architectural style chosen for representing scenarios and requirements :*** The architectural style should describe various views about the system. Those views are -
   i) **Model view :** Using model view the working of components and the information hiding among them can be described.
   ii) **Process view :** This view helps to analyse the system performance
   iii) **Data flow view :** Using this analysis, the functional requirements analysis can be made.
4. ***Evaluate quality attributes individually :*** Various quality attributes that can be evaluated are - reliability, performance, security, flexibility, maintainability, testability, portability, interoperatibility and reusability.
5. ***Identify the sensitivity of quality attribute :*** For checking the sensitivity of these attributes, small changes are made in the architecture and observe the effects of these changes on the quality attributes. If any attribute gets affected significantly by these changes then that attribute is considered as a sensitive point.
6. ***Criticize quality attribute :*** By making small changes in the architecture, sensitive points are obtained. Now the architecture trade-off points can be obtained by criticizing these quality factors.

2. Pseudo Quantitative Technique :

- In this approach, particular architecture should meet a predefined goodness criteria. This criteria is called **design dimension**. This criteria consists of various quality attributes such as reliability, security, performance, maintainability, flexibility, testability, reusability and interoperability.
- First model is **spectrum analysis**. This model assesses various architectural designs using the **goodness** spectrum. Then particular software architecture is proposed. This architecture is assessed by assigning some **score** to each of its design dimension. All these scores are summed up to obtain the total score of the entire design. The best case, worst cases are considered to obtain the best case and worst case score of the system. From

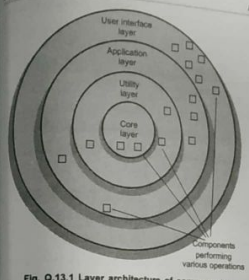


Fig. Q.13.1 Layer architecture of components

- The outer layer is responsible for performing the user interface operations while the components in the inner layer perform operating system interfaces.
- The components in intermediate layer perform utility services and application software functions.

Q.14 Discuss various categories of architectural style. **[JNTU : Part B, May-07, Marks 05]**

Ans. : Refer Q.7, Q.8, Q.9, Q.10, Q.11, Q.12 and Q.13.

Q.15 Give a brief taxonomy of architectural styles and patterns. **[JNTU : Part B, May-13, Marks 5]**

Ans. : Refer Q.8 and Q.9, Q.10 and Q.11.

8.4 : Architectural Design

Q.16 What is an architectural context diagram ? Explain with an example. **[JNTU : Part B, Dec.-10, Marks 8]**

Ans. : The context model is a graphical model in which the environment of the system is defined by showing the external entities that interact with the software system.

Q.17 Give the architecture context diagram for room temperature monitor system. **[JNTU : Part B, May-13]**

Ans. :

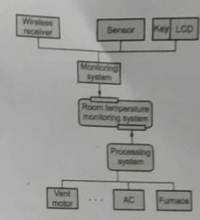


Fig. Q.17.1 Context diagram for room temperature monitoring system

Q.18 What is an archetype ? How would you define it ? **[JNTU : Part B, Dec.-10, Marks 8]**

Ans. : • Defining archetype is a **basic step** in architectural design, more precisely in functionality based design.

- Archetype is a **core abstraction** using which the system can be structured.
- Using archetypes, a **small set** of entities that describe the major part of system behaviour can be described.
- Typically archetypes are the stable elements and they do not change even though system undergo through various changes.
- Identifying archetypes is a critical task and it requires well experienced architect.
- Following figure represents various archetypes.

Various types of archetypes are

1) **Point or node :** It refers to the highest level of abstraction in which there is a cohesive collection of input and output functionalities.
2) **Detector :** Detectors capture the core functionalities of the system. For example, in temperature control system sensors for temperature is a sensor.

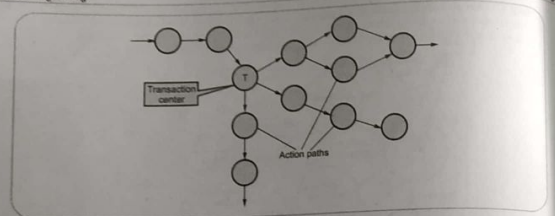


Fig. Q.21.2 Transaction flow

Q.22 Explain the process of mapping dataflow into software architecture. [JNTU : Part B, Dec.-16, Marks 5]

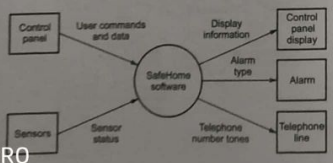
Or Discuss with an example, how to map data flow into a software architecture. [JNTU : Part B, Dec.-13, Marks 7]

Ans. : The steps for transform mapping are as follows -

1. Review the fundamental system model.
2. Review and refine DFD for the software.
3. Determine whether the DFD has transform or transaction flow characteristics
4. Isolate the transform center by specifying incoming and outgoing flow boundaries.
5. Perform "First Level Factoring".
6. Perform "Second Level Factoring".
7. Refine the "First-cut" program structure using design heuristics for improved software quality.

Example - Mapping dataflow into software architecture for SafeHome Software

Step 1 : Review the fundamental system model.



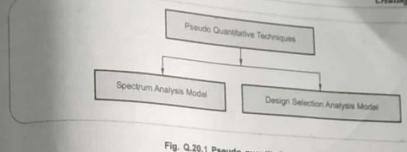


Fig. Q.20.1 Pseudo quantitative techniques

these values spectrum index is calculated using following equation

$$I_S = [(S - S_W)/(S_b - S_W)] \times 100$$

where

S is the total score

S_W is the worst case score

S_b is the best case score

This spectrum index indicates the spectrum within which the proposed system architecture can be built.

- **Design selection analysis** is another model in which a set of **design dimensions** are defined. The proposed architecture is assessed to determine the number of design dimensions that achieve when compared with best case system. The design selection index d can be calculated as -

$$d = (N_s/N_a) \times 100$$

where

N_s is number of design dimensions achieved by the proposed architecture

N_a sis the total number of design dimensions.

If the index value is higher then the proposed architecture is more close to the ideal system.

8.6 : Mapping Data Flow into Software Architecture

Q.21 What is transform flow and transaction flow ?

Ans. : Transform flow : A transform flow is a sequence of paths which forms transition in which input data are transformed into output data. (Refer Fig. Q.21.1.)

Transaction flow :-A transaction flow represents the information flow in which single data item triggers the overall information flow along the multiple paths. This triggering data item is called **transaction**. (Refer Fig. Q.21.2 on next page.)

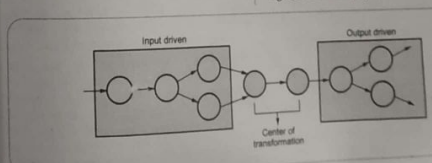
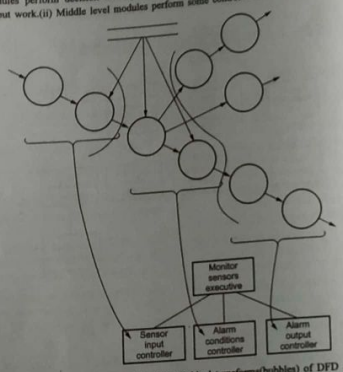
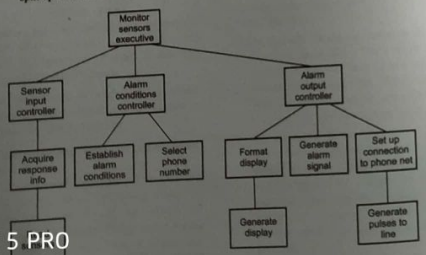


Fig. Q.21.1 Transform flow

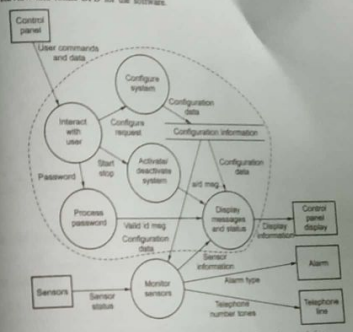
Step 5 : Perform "First Level Factoring" - (i) The factoring results in a program structure in which top level modules perform decision making and low level modules perform most input, computational and output work.(ii) Middle level modules perform some control and do moderate amounts of work.



Step 6 : Perform "Second Level Factoring" - (i) Individual transforms(bubbles) of DFD are now mapped into appropriate modules within the program structure. (ii) In the next step, modules can be combined or split apart to achieve a good level of cohesion and coupling within the program structure.

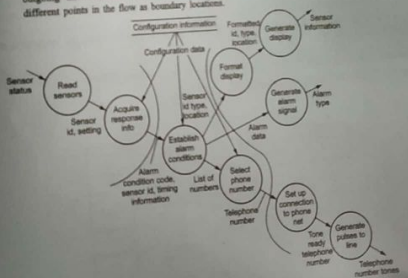


Step 2 : Review and refine DFD for the software.



Step 3 : Determine whether the DFD has transform or transaction flow characteristics

Step 4 : Isolate the transform center by specifying incoming and outgoing flow boundaries. Incoming and outgoing flow boundaries are open to interpretation and different designers may select slightly different points in the flow as boundary locations.



UNIT - III

9 Modeling Component Level Design

9.1 : Designing Class Based Components

Q.1 What is component ? JNTU : Part A, Marks 2]

Ans. : • Component is nothing but a model for computer software.

- The OMG Unified Modeling Language Specification defines the concept of component more formally and it is -
  "Component is a modular, deployable and replaceable part of system that encapsulates the implementation and exposes the set of interfaces".
- Components are the part of software architecture and hence they are important factors in achieving the objectives and requirements of system to be built.

Q.2 Discuss - Designing class based components.
JNTU : Part B, Dec.-14, Marks 7]

Ans. : • Component is represented as a part of architectural model. It collects the information about the system as a part of analysis model.

- If **object oriented software engineering approach** is adopted then the component level design will emphasize on elaboration of analysis classes and refinement of **infrastructure classes.**
- The detailed description of attributes, operations and interfaces of these infrastructure classes is required during the system building.

Q.3 Explain the principles used for class based component level design.

Ans. : 1. The Open-closed Principle : This principle states that - A module or a component should be open for extension and should be closed for modification. A designer should design a component in such a manner that some functionalities can [illegible] if required but in doing so there should not be any change in the internal design of the component itself.

2. The Liskov Substitution Principle : This principle states that - subclasses should be substitutable for their base classes. A component that contains the base class and if that component works properly then the same component should work properly even if the derived class of that base class is used by the component as a substitute.

3. Dependency Inversion Principle : The components should be dependant on the other abstract components and not on the concrete components.

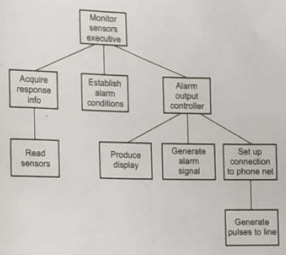
4. The Interface Segregation Principle : Many client specific interfaces are better than one general purpose interface. According to this principle, designer should create specialized interfaces to each major category. So that the clients can access the interfaces based on the category.

5. The Common Closure Principle : If a modification is applied to the classes that are packaged together, then the changes must occur within only those classes that get affected due to the modification.

6. The Release Reuse Equivalency Principle : This principle states that - The granule of reuse is the granule of release. That means, while releasing a newer version from the older version, all those reusable classes must be grouped together into a package that can be managed and controlled.

7. The Common Reuse Principle : This principle states that -the classes that aren't reused together should not be grouped together. If we need the services offered by a class, we must import the package containing the necessary classes. when we import a package, we also may utilize the services offered by any public class within the package.

Step 7 : Refine the "First-cut" program structure using design heuristics for improved software quality - Modules can be exploded or imploded to produce sensible factoring, good cohesion, minimal coupling and most importantly, a structure that can be implemented.



END...

9.4 : Designing Conventional Components

Q.8 Discuss - Designing conventional components.
[JNTU : Part B, Dec.-14, Marks 8]

Ans. : • **Component level design** is also called as **procedural design**. After data, architectural and interface design the component level design occurs.

• The **goal** of component level design is to translate design model into operational software.

• Graphical, tabular or text based notations are used to create a set of **structured programming constructs**. These structured programming constructs translate each component into **procedural design model**. Hence the work product of component level design is procedural design model.

• There are **three constructs** of structured programming

1. Sequence 2. Condition 3. Repetition

• **Sequence** denotes a linear processing of the statements. **Condition** provides the facility to test the logical conditions (For example if-then-else conditions). **Repetition** is used to denote the looping.

Programming constructs can be represented as follows -

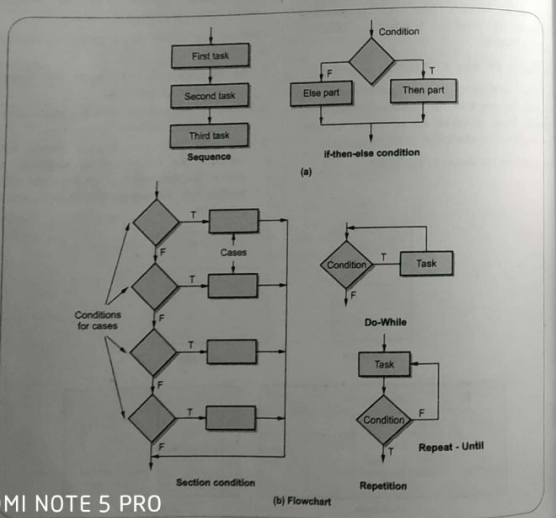


Fig. Q.8.1

9.2 : Conducting Component Level Design

Q.4 Define component level design.
[JNTU : Part A, May-13, Marks 2]

Ans. : Component level design is a design of the architecture system using various components that collaborate and communicate with each other.

Q.5 Explain the steps used for conducting component level design.

Ans. : Following is a set of steps that are applied for component level design -

1. Identify all the design classes from the problem domain.
2. Identify all the design classes from the infrastructure domain.
3. Detail out all the design classes which are not the reusable components.
   a. When components communicate or collaborate then represent their communication using messages.
   b. Identify interfaces for each component.
   c. Define the attributes of the classes by specifying the data types and deciding the data structures.
   d. Specify all the operations by describing the processing flow within each operation.
4. Specify the databases and files and identify the classes that are involved in handling them
5. Describe the behavioural representation for the class.
6. Design the deployment diagram fro providing additional implementation.
7. Factor the component level design.
8. Find out alternative design solutions as well.

9.3 : Object Constraint Language

Q.6 Explain about object constraint language.
[JNTU : Part B, May-13, Marks 5]

Ans. : • The design model is represented by the Unified Modeling Languages(UML). But the graphical representations are not enough.

• There are situations in which we want to describe the constraints of the objects. Of course, these constraints can be represented by natural languages but this representation can be ambiguous and inconsistent.

• To represent unambiguous and consistent way of representing the constraints of the objects a formal language is introduced. This language is called object **Constraint language** (OCL).

• This language makes use of UML along with the formal grammar and syntax to construct various design elements.

• There are four parts of OCL -

1. **Context** : It defines the situation for the design element.
2. **Property** : It represents the characteristic of the element.
3. **Operation** : Various arithmetical, logical or relational operations can be carried out of the design elements.
4. **Keywords** : There are some reserved words in this language which are associated with the some meanings. For instance : if, else, not, or, implies.

• For example : Following is a part of OCL for **gaming software** for defining the **move.**

```
context Game::move(position : Integer)
pre validPosition: position >= 0 and position
                                < board.size
pre WinnerStatus: winner().>isEmpty()
post changedCurrPlayer:
currPlayer = players->at
((players->indexOf(currPlayer@pre) + 1).mod
(players->size()))
```

The **pre** and **post** are for specifying the pre and post conditions. These are called the invariants. Before performing some operation all the preconditions must get satisfied and after performing the operation post condition must be satisfied.

Q.7 Explain clearly the component level design and object constraint language.
[JNTU : Part B, Dec.-13, Marks 15]

Ans. : Refer Q.4, Q.5 and Q.6.

These programming constructs can be nested one. That means one programming construct can be within another construct.

Q.9 Enlist the advantages of structured programming constructs. **[JNTU : Part A, Marks 3]**

Ans. : Following are the advantages of using structured programming constructs -

1. The structured programming constructs **reduces** program **complexity**.
2. It enhances **readability**, **testability** and **maintainability** of the procedure.
3. The structured programming constructs are the **logical chunks** that allow a reader to recognize procedural element from each programming module. This ultimately enhances the readability of the program.

Q.10 What is NS chart ? **[JNTU : Part B, Mark 5]**

Ans. : The developers Nassi and Shneiderman developed the notations for structured programming constructs. These notations are known as NS chart. These diagrams are also called as box diagrams.

These notations are shown in Fig. Q.10.1.

Following are the **characteristics of the NS chart** -

1. The scope of the programming constructs such as repetition, if-then-else is well defined.
2. Arbitrary transfer of the control is not possible using this notation.
3. Recursion can be represented conveniently
4. The scope of local and global data can be defined systematically.

Q.11 Explain the tabular design notations in detail. **[JNTU : Part B, Marks 5]**

Ans. : Programming constructs can be represented by tabular design notation. This an alternative representation to the graphical notations.

There are four sections in the tabular representation method -

1. The first section is at the upper left corner. It consists of list of conditions.
2. The second section is the lower left hand corner. It consists of list of actions.
3. The right hand portion is a matrix which represents the combination of conditions and actions. The combination of condition and actions together form **processing rules** for that particular procedure.

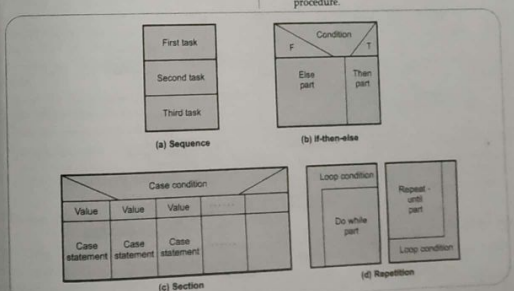


Fig. Q.10.1 NS - chart notations

Following steps are applied to develop a decision table -

- List all the actions associated with particular procedure.
- List all the conditions associated with particular procedure.
- Associate each condition with each corresponding action. Any impossible permutation must be eliminated. All possible permutations of conditions and corresponding actions must be represented in the table.
- Create the processing rule indicating that particular action exists on particular condition.

For example -

Conditions	1	2	3	4	5		n
Condition 1	✔	✔			✔		
Condition 2		✔	✔				
Condition 3	✔			✔	✔		
Actions							
Action 1	✔		✔	✔			
Action 2		✔	✔	✔			
Action 3	✔				✔		

Q.12 Write short note on - Program Design Language

Ans. : • A program design language is also called pseudo code or structured English.

- This is similar to English but is used as a generic reference for design language.
- The PDL is not compiled.
- It is used to translate the design into the programming language.
- A basic PDL syntax should posses the programming constructs for interface description, procedure definition, data declaration, condition constructs, repetition constructs and I/O constructs.
- **For example :** Following is a PDL which demonstrates the procedure for searching the name John from the table. IF the name is found then index of the table is returned

```
Search(table,number of items)
 Set count to Zero
 Read first item from table
 DO FOR count is <=number of items
  IF table.name="John"
   RETURN table.index
  ELSE
   count=cout+1;
   Read Next item from table
  ENDIF
 ENDFOR
END
```

END...

Following are the principles suggested by Mandel to reduce the memory load of the user.

1. **Do not force the user to have short term memory :** The user interface should be such that the user need not have to remember past actions and results. And this can be achieved by providing proper visual interface.
2. **Establish meaningful defaults :** Meaningful default options should be available to the user. For example in the Microsoft word the user can set the default font size as per his choice.
3. **Use intuitive shortcuts :** For quick handling of the system such short cuts are required in the user interface. For example control+s key is for saving the file.
4. **The visual layout of the interface should be realistic :** When certain aspect/feature of the system needs to be highlighted then use of proper visual layout helps a casual user to handle the system with ease.
5. **Disclose the information gradually :** The information should be presented to the user in a systematic manner.

Make User Interface Consistent :

A set of design principles that help make the interface consistent :

(1) Allow the user to put the current task into a meaningful context : The user should be able to determine where he has come from and what alternatives exist for a transition to a new task.

(2) Maintain consistency across a family of applications : For instance "MS Office Suite"

(3) If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so : Once a particular interactive sequence has become a de facto standard (Alt-S to save file), the user expects this in every application he/she encounters.

10.2 : User Interface Analysis and Design

Q.6 Enlist the user interface analysis and design steps. [JNTU : Part A, Marks 3]

Ans. : User interface analysis and design can be done with the help of following steps.

- Create different **models** for system functions.
- In order to perform these functions identify the human-computer interface tasks.
- Prepare all **interface designs** by solving various design issues.
- Apply modern tools and techniques to prototype the design.
- **Implement** design model.
- **Evaluate** the design from end user to bring quality in it.

Q.7 What are interface design model ? Explain. [JNTU : Part B, May-13, Marks 5]

Ans. : Typically there are four types of models that can be prepared in interface analysis and design. Let us understand each of them

- **User model -** To design any user interface it is a must to understand the user who is using the system. Hence in this model the profile of user is prepared by considering age, sex, educational, economical and cultural background. The software engineer prepares user model. Globally there are three kinds of users

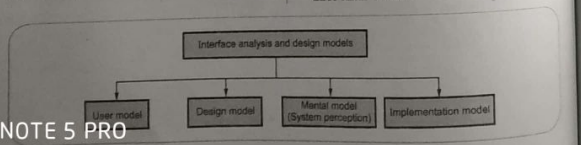


Fig. Q.7.1 Analysis and design model

UNIT - III

10 Performing User Interface Design

10.1 : Golden Rules

Q.1 Explain any three different ways by which one can learn what the user wants from the user interface [JNTU : Part A, Marks 3]

Ans. : Following are the ways by which one can learn **what the user wants** from the user interface -

- **User interviews** : This is the most effective technique in which some representatives from software team meet the end user to better understand the user needs, motivations and many other issues. Meetings are conducted for this purpose.
- **Sales input :** Sales people interact with the users regularly and collect the information. Based on this information the users are categorized in particular groups and thereby their requirements are better understood.
- **Marketing input :** Market analysis is made in order to understand the usage of software.
- **Support input** : Support staff keeps a regular interaction with the user interaction for knowing the certain things like the likings and dislikes of the users, which features are easy to use and so on.

Q.2 What is the necessity of good user interface ? [JNTU : Part A, Dec.-17, Marks 2]

Ans. : The purpose of user interface design is to have effective communication medium between the computerized system and the user. This kind of interface design is necessary because of many reasons such as : Software is difficult to use, the use of software forces the user to make mistakes due to lack of understandings about the system.

Q.3 Briefly explain about the golden rules for user interface design. [JNTU : Part B, May-13,16, Marks 5, Dec.-17, Marks 2]

Ans. : Thao Mandel has proposed three golden rules for user interface design -

1. **Place the user in control :** Various design principles for this rule are
   i) Define the interaction modes in such a way that user will be restricted from doing the unnecessary actions.
   ii) The interaction should be flexible.
   iii) Provide the facility of 'undo' or 'interruption' in user interaction.
   iv) Hide technical details from the user.
2. **Reduce the user's memory load :** This can be done using various rules like
   i) Do not force the user to have short term memory.
   ii) Establish meaningful defaults.
   iii) Use intuitive shortcuts.
   iv) The visual layout of the interface should be realistic.
3. **Make the interface consistent :** For this some guiding principles can be
   i) Allow user to direct the current task into meaningful manner.
   ii) If certain standards are maintained in previous model of application do not change it until and unless it is necessary.

Q.4 What are the design principles that allow user to maintain control ? [JNTU : Part B, Dec.-10, Marks 8]

Ans. : Refer Q.3(1)

Q.5 How to reduce user's memory load and how to make user interface consistent ? [JNTU : Part B, Dec.10, June-14, Marks 8]

Ans. : Reduce User's Memory Load : If the user interface is good then user has to remember very less.

User	Description
Novice	The user with very little knowledge of the computer who simply knows semantics (simply the wants) of the system and does not know the implementation of such system.
Knowledgeable and intermittent	The user having knowledge about the semantics of the system as well as having little knowledge of syntactic of the system.
Knowledgeable and frequent	The user with good semantic as well as syntactic knowledge of the system.

- **Design model** - It consists of data, architectural, interface and procedural representation of the software. While preparing this model the requirement specification must be used properly to set the system constraints. Software engineer prepares the design model of the interface.
- **Mental model (system perception)** - The user model is the representation of **what user thinks** about the system ? Basically any user interface design is heavily dependant upon the description obtained from the user about his wants and needs. If the user is knowledgeable then more complete description of the system can be obtained than that of novice user.
- **Implementation model** - The implementation model generates the **look and feel** of interface. This model describes the system's semantic and syntax. It is very necessary to match the implementation model with the user's mental model then only user can feel comfortable with the developed system.

Q.8 Explain the user interface analysis and design process along with spiral diagram.

Ans. : The user interface analysis and design process can be implemented using iterative spiral model. It consists of four framework activities.

As shown in the above figure each of these tasks can be performed more than once. At each pass around the system more requirements can be elaborated and detailed design can be performed.

Environment analysis and modelling - In this phase three major factors are focused i.e. user, task and environment. First of all the user profile is analysed to understand the user and to elicit the requirements, then the tasks that are required to carry out desired functionality are identified and analysed. The analysis is made on user environment which involves the physical work environment. Finally analysis model is created for the interface. This model serves as a basis for the design of user interface.

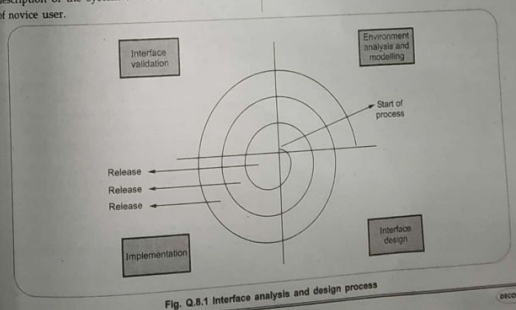


Fig. Q.8.1 Interface analysis and design process

Interface design - The interface design is a phase in which all the interface objects and corresponding actions of each task are defined.

Implementation - The implementation phase involves creation of prototype using which the interface scenarios can be evaluated. To accomplish the construction of user interface some automated tools can be used

Validation - The goal of validation is to validate the interface for its correct performance. During validation it is tested whether all the user requirements get satisfied or not. The purpose of validation is also to check whether the interface is easy to learn and easy to use.

10.3 : Interface Analysis

Q.9 What is the use of interface analysis ? Explain. **[JNTU : Part A, Dec.-16, Marks 2]**

Ans. : The interface analysis need to be performed for following reasons -

1. For understanding people or user who actually interact with the system.
2. For knowing the tasks that are performed by the end user for interacting the system.
3. The contents of the system will be displayed to the user in systematic manner.
4. The environment in which the task will be conducted can be known to the user.

Q.10 How to determine the format and aesthetic of content displayed as part of user interface ? **[JNTU : Part B, Dec.-10, Marks 10]**

Ans. : For determining the format and the aesthetics of the contents following questions can be asked or answered.

a. Is it possible for the interface to display various types of data in a consistent manner on geographic location ?

b. Is it possible to customize the screen location with respect to the contents ?

c. The information is partitioned properly for the sake of understanding ?

d. Is the summary of large amount of data available directly ?

e. Will the graphical output be displayed properly so that it will fit to the display device ?

f. Is there a use of sophisticated color combination ?

g. Are the errors and warning messages presented to the user in interactive manner ?

- Answers to these questions will help in gathering the requirements for the presentation of the contents.

Q.11 Discuss the importance of analysis of the work environment in user interface design. **[JNTU : Part B, Dec.-10, Marks 6]**

Ans. : • The analysis of work environment is very important.

- In some applications the user interface for the computer based systems is placed in very user friendly environment. In such a situation, there must be interactive presentation of the information, proper lighting, good display height, easy keyboard access, proper sitting arrangement and so on. In such work environment using the application becomes an enjoyable activity.
- But there are some workplaces in which there may be insufficient light, lot of noise and distractions, unavailability of keyboard or mouse interfacing and so on. Using the application in such environment becomes difficult and frustrating.
- In addition to these physical factors, considerations of work place culture is extremely important. These factors are raised by following questions -
  - Will more than one person access the same information before providing the input ?
  - Will the system interaction be measured in terms of some measure. For instance : Time required for processing of data.
  - Will the system provide some kind of support to the user for handling it ?
- These issues must be solved before the completion of interface design phase.

(ii) Response Time :

- System response time is the primary complaint for many interactive applications.
- The system response time is measured from the point at which the user performs some control action.
- There are two important characteristics of response time - (i) Length and ii) Variability.
- If system response is too long, there will be user frustrations and stress. The variability means the deviation from average response time and it is an important response time characteristic.

(iii) Error Handling : Errors and warning cause the frustrations to the user. Sometimes the error messages are in very vague form. For example : Application

ABC has encountered error 1033. Now such type of error messages do not direct the user about what went wrong. Following are the characteristics for presenting errors.

1. The error message should be in a language which user can understand. For example : Remote server not responding.
2. There should be some useful message along with error in order to recover from the error. For example : Press enter to exit
3. There should be some audible or visual cue along with the error message. For example: beep sound or highlighting back ground of the text.
4. There should not be any negative impact of error on the user. For example : File XYZ.SYS has been deleted such error will cause frustration on the user.
5. The wording of the error should be carefully used and it should not blame user. (because nobody likes to get criticized!!)

Thus error messages should be so effective that they should bring qualitative interaction between the user and the system.

- There are situations in which the user interface is developed for localised purpose and for local language. If the same interface is required in other side of the country then existing interface is used with more or less modifications.
- The real challenge is to develop globalized software. That means the user interfaces should be designed to accommodate a generic core functionality. This functionality can be used by any user belonging to any country.
- Localized user interface is useful to only localized market.
- There are many internationalization guidelines that are available for software developers.
- These guidelines address the design and implementation issues.
- The **Unicode standard** has been developed to handle the challenges of managing the natural languages with lots of characters and symbols.

Q.15 State why response time variability of user interface can be an issue. Illustrate with example.
[JNTU : Part B, June-14, Marks 7]

Ans. : • Response time variability means deviation from average response time. It is an important characteristic of the system.

- Low variability enables the user to establish an interaction rhythm, even if response time is relatively long.
- For example 1 second responds to command and this response can vary 0.1 to 2.5 seconds
- When variability is significant, the user will always feel that something unusual has occurred behind the scene.

10.5 : Design Evaluation

Q.16 Explain the design evaluation.
[JNTU : Part B, Marks 5]

Ans. : • The **implementation** phase of user interface includes **creation of prototype** that enables user to evaluate the usage scenarios.

- This evaluation is necessary to determine whether the created interface satisfies user demands or not. After evaluating the usage scenario the user immediately submits his **comments** to the designer.

Q.12 List various questions that will help the interface designer better understand the users of a system.
[JNTU : Part B, April-11, Marks 8]

Ans. : Following is a set of questions that will help the **interface designer better understand the users** of a system -

1. Are user trained professionals, technicians, or worker ?
2. Are the users capable of learning from written material ? OR they require any classroom training ?
3. What is the formal education of the user ?
4. Are users aware of using keyboard ?
5. What is the age group of user ?
6. Will user be represented dominantly by one gender ?
7. Does the user work in office hour or do they work until the job is done ?
8. What kind of compensation will be given to the user for the work they perform ?
9. Will the user make use of the software frequently or occasionally ?
10. What is the primary spoken language among the users ?
11. What will happen if the user makes a mistake in handling the system ?
12. Are the expert users addressed by the system.
13. Do users want to know the technology used behind the interface ?

10.4 : Interface Design Steps

Q.13 State and explain various interface design activities.
[JNTU : Part B, Dec.-12, Marks 8]

OR State and explain the generic tasks that always performed in user interface design.
[JNTU : Part B, Dec.-10, Marks 8]

Ans. : Following are the commonly used interface design steps -

1. During the interface analysis step **define** interface **objects** and corresponding **actions** or operations.
2. Define the major **events** in the interface. These events depict the user actions. Finally **model** these events.
3. **Analyse** how the **interface** will look like from user's point of view.
4. **Identify** how the **user understands** the interface with the information provided along with it.

While designing the interface software engineer communicates the user and according to his thinking about the interface, software engineer draws the sketches. Get it approved from the user and then work on defining objects and corresponding actions.

Q.14 State and explain clearly the various user interface design issues.
[JNTU : Part B, April-11, Marks 16]

OR How following factors are addressed in interface design ? i) Help facilities ii) Response time iii) Error handling iv) Internationalization.
[JNTU : Part B, Feb.-10, Marks 16]

Ans. : Various user interface design issues are as follows -

(i) Help Facilities : This is the most essential criteria for any user interface this makes the system more interactive. The help can be **on-line help** or it can be in the form of **user manual**. Even some software provide the help in the **assistance** form i.e. the question may be asked by the user and the answer given by the system will serve as a help to the user.

Number of design issues that must be addressed when a help facility is considered are as given below -

(a) Help can be available for all system functions or at all times during system interaction.
(b) User can request for help using Help menu, a special function key or using HELP command.
(c) Help can be represented in a separate window, or reference to printed document.
(d) User can return to normal interaction from help menu using the return button.
(e) Help can arranged in flat structure or in a layered hierarchy of information.

For Mid Term Exam

Fill in the Blanks

1. ___ is a process of translating analysis model into design model.
2. A _____ is a named collection of data that describes a data object. **JNTU : August-16**
3. In object-oriented design, the modules in the design represent ____
4. _____ is the process of changing a software system. **JNTU : August-16**
5. ____ and ___ are two qualitative criteria used for functional independence.
6. The weakest coupling is ____
7. The dashed line in design model represents the ___ between analysis and design
8. The type of cohesion in which task within a module need to be executed in some specific time span is called ___
9. The goal of good design is ____ cohesion and ___ coupling
10. Archetype describes the pattern which is used in designing the ___ system.
11. UML is mainly used for ___ design.
12. The Mean Time To Failure(MTTF) is a metric that is widely used to measure the ____.
13. The modularity criteria proposed by Meyer are - modular decomposability modular composability, understandability, ___ and ___.
14. Various elements of Data design are ___, ____ and ____.
15. A data flow diagram is mapped into program structure using transform mapping and/or transaction mapping.
16. ____ type of mapping is applied to any information flow that exhibits distinct boundaries between incoming and outgoing data.
17. The constructs fundamental to structured programming are sequence, ____ and repetition.
18. PDL stands for _____.

Multiple Choice Questions

Q.1 In ___ coupling the global variables are used.

a. Stamp b. data
c. common d. content

Q.2 Which of the following is a tool in design phase ?

a. Abstraction
b. Refinement
c. Information hiding
d. All of the above

Q.3 Information hiding is to hide information from user ____.

a. that is relevant to him,
b. that is irrelevant to him
c. that can be maliciously handled by him
d. none of these

Q.4 Design phase includes _____.

a. data, architectural, and procedural design only
b. architectural, procedural and interface design only
c. data, architectural and interface design only
d. data, architectural interface and procedural design only

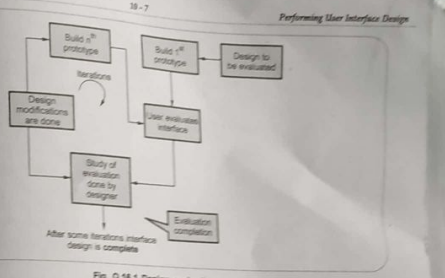


Fig. Q.16.1 Design evaluation

- The designer then makes corresponding modifications in the interface and then again it is submitted to further evaluation.
- This design evaluation of user interface is an **iterative process**. It is clearly illustrated by following Fig. Q.16.1.
- After creation of **first prototype** it is evaluated by the user.
- The evaluation made by the user is then submitted to the designer who then makes necessary changes in the interface and builds the next prototype.
- This process is repeatedly performed until the user gets satisfied completely and no further modifications are necessary in the interface.
- Thus prototype approach proves to be effective for design evaluation.

END...

Testing Strategies

⇒ Software Testing Introduction:-

→ s/w testing is a critical element of s/w quality assurance and represents the ultimate review of specification, design & coding.

→ The purpose s/w testing is to ensure whether the s/w functions appears to be working according to specifications and performance requirements.

→ The objective of testing is a process of executing a program with the intend of finding an error.

→ A good test case is one that has high probability of finding an undiscoved error.

→ All tests should be traceble to customer requirements.

→ Generally, testing is a process that requires more effort than any other s/w engineering activity.

⇒

→ Various testing phases are

1. Test planning: The test plan or test script is prepared. These are generated from requirements analysis document and program code.

2. Test case Design:- The goal of test case design is to create a set of tests that are effective in testing.

3. Test execution: The test data is derived through various test cases in order to obtain the test results.

4. Data collection: The test results are collected and verified.

5. Effective evaluation: All the above test activities are performed on the s/w model and the maximum no. of errors are uncovered.

→ The testing strategy provides a process that provides describes for the developer, quality analysts and the customer the steps conducted as part of testing.

* → The strategic approach for sw testing can be -

1. Just before starting the testing process the "formal technical reviews" must be conducted. This will eliminate many errors before the actual testing process.
2. At the beginning, various components of the system are tested, then gradually each interface is tested & thus the entire computer based system is tested.
3. Different testing techniques can be applied at different point of time.
4. The developer of the sw conducts testing. For the large projects the Independent Test Groups (ITG) also assist the developers.
5. Testing & debugging are different activities that must be carried out in sw testing.
6. Debugging also lies within any testing strategy.

There are two levels specified in the testing strategy and those are "low level" and "high level".

→ The low level tests that are necessary to verify small source code segment has been correctly implemented.

→ The high level tests should be conducted that are validate major system functions against customer requirements.

→ Difference between verification & validation:-

Verification	validation.
* Verification refers to set of activities that ensure s/w correctly implements the specific function.	* validation refers to the set of activities that ensure that the s/w that has been built is traceble to customer requirements.
* According to Boehm verification says " Are we building the product right"?	* According Boehm the validation says "Are we building the right product?
* After a valid and complete specification the verification starts.	* validation begins as soon as project starts.
* The verification is conducted to ensure whether s/w meets spe-fication or not.	* validation is conducted to show that the user requirements are getting satisfied.

→ s/w testing is only one element of SQA.

→ Verification and validation involve large no. of s/w quality assurance activities such as-

1. Formal technical reviews
2. Quality & configuration measurements
3. performance monitoring
4. Feasibility study
5. Documentation review
6. Database review
7. Algorithmic analysis
8. Development testing
9. Installation testing.

⇒ Testing strategies

→ We begin by "testing-in-the-small" & move toward "testing-in-the large".

→ Various testing strategies for conventional s/w are

1. unit testing
2. Integration testing
3. validation testing
4. system testing.

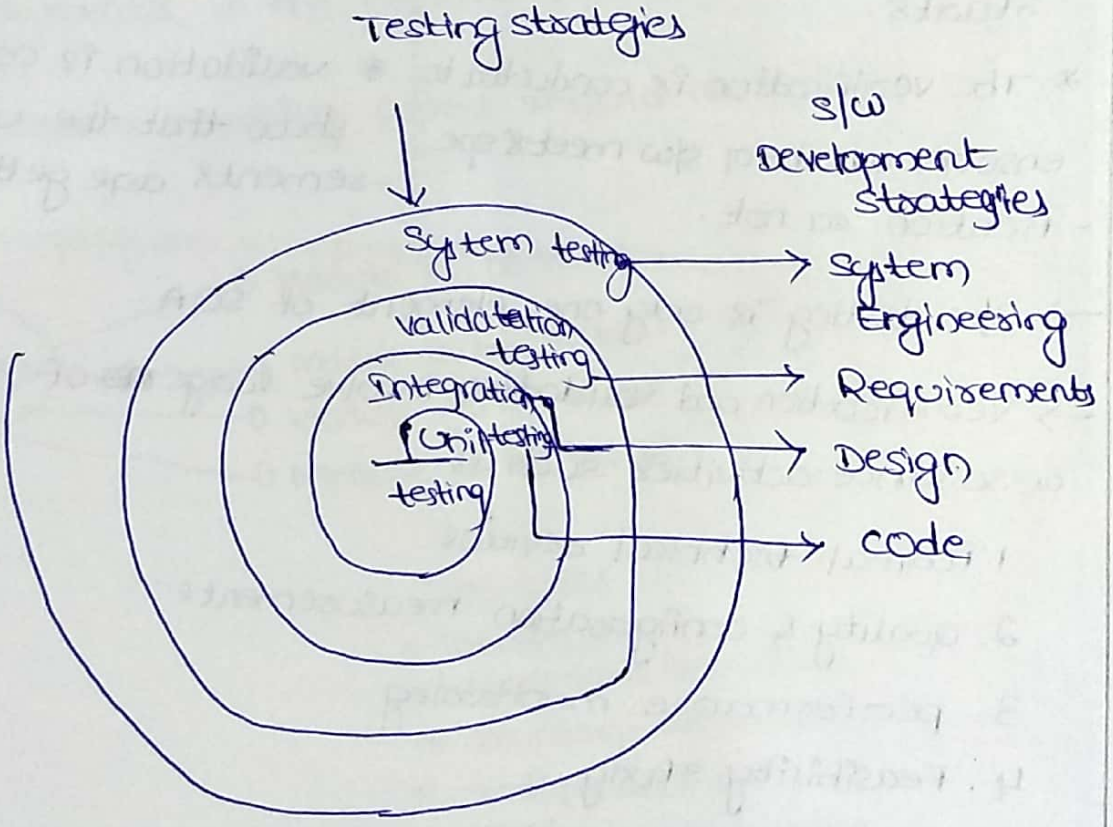


fig. testing strategy.

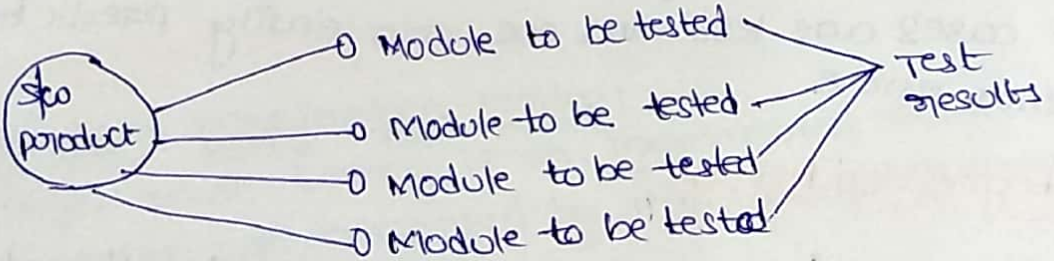
⇒ unit testing :-

→ In unit testing the individual components are tested independently to ensure that their quality.

→ The focus is to uncover the errors in design & implementation.

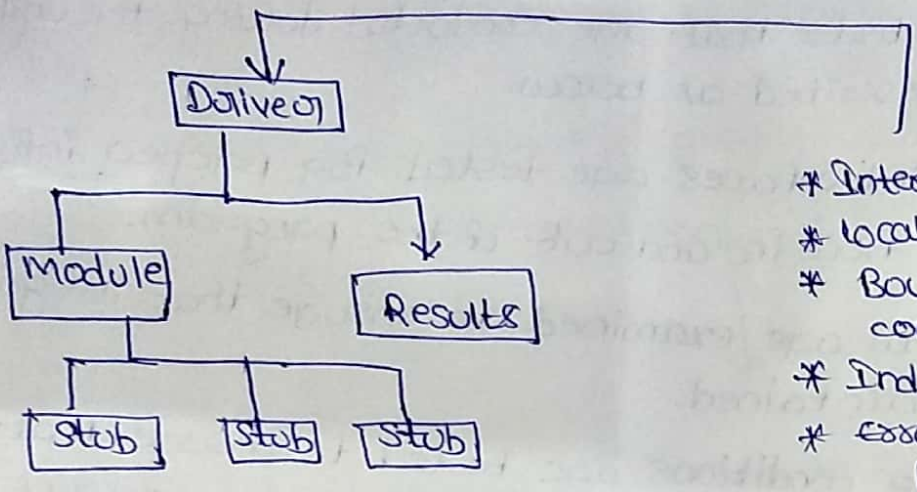
→ The various tests that are conducted during the unit test are described as below.

* Module interfaces are tested for proper info -rmation flow in and out of the program.
* Local data are examined to ensure that integri -ty is maintained.
* Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
* All the basis paths are tested for ensuring that all statements in the module have been executed only once.
* All error handling paths should be tested.



Test cases
* interfaces
* Local data structures
* Boundary conditions
* Independent path
* Error handling paths.

→ Drivers and stub s/w need to be developed to test incomplete s/w. The "driver" is a program that accepts the test data & prints the relevant results. And the "stub" is a subprogram that uses the module interfaces and performs the minimal data manipulation if required.



* Interfaces
* local data structure
* Boundary conditions
* Independent paths
* error handling paths.

fig unit testing environment

→ The unit testing is simplified when a component with high cohesion is designed. In such a design the no. of test cases are less and one can easily predict or uncover errors

⇒ Integration Testing:-

→ A group of dependent components are tested together to ensure their quality of their integration unit.

→ The objective is to take unit tested components and build a program structure that has been dictated by s/w design.

→ The focus of integration testing is to uncover errors in

* Design and construction of s/w architecture.
* Integrated functions or operations at subsystem level.
* Interfaces and interactions b/w them.
* Resource integration.

→ The integration testing can be carried out using 2 approaches.

1. Non-Incremental Integration → Bigbang
2. Incremental Integration → Top-down testing
   → Bottom-up integration
   → Regression testing
   → Smoke testing.

Non-Incremental Integration

*** → The Non-incremental integration is given by the "bigbang" approach. All components are combined in advance.

→ the entire program is tested as a whole. And chos usually results.

→ A set of errors is tested as a whole. correction is diffi -cult because isolation of causes is complicated by the size of the entire program. once these errors are corre -cted new once appear. this process continues infinitely.

Incremental Integration:-

1) Top-down Integration-testing:

→ Top-down testing is an incremental approach in which modules are integrated by moving down through the control structure.

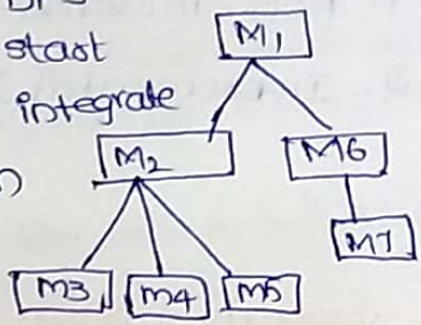
→ Module subordinate to the main control module are inco -rporated into the system in either DFS or BFS

→ In top-down Integration process can be performed us -ng following steps:

1. the main control module is used as a test driver & the stubs are substituted for all modules directly subordinate to the main control module.
2. subordinate stubs are replaced one at a time with actual modules using either DFS or BFS.
3. Tests are conducted as each module is integrated.
4. on completion of each set of tests, another stub is replaced with the real module.

→ Regression testing is conducted to prevent the introduction of new errors.

Eg: In top-down integration if DFS approach is adopted then we will start integration from M1 then we will integrate M2, then M3, M4, M5, M6 and then M7.



If BFS approach is adopted then we will integrate module M1 first then M2, M6, then we will integrate module M3, M4, M5 and finally M7.

M1

⇒ Bottom-Up-Integration Testing:

→ In bottom-up integration the modules at the lowest levels are integrated at first, then integration is done by moving upward through the control structure.

→ The bottom-up integration process can be carried out using following steps.

1. Low-level modules are combined into clusters that perform a specific sw subfunction.
2. A driver program is written to co-ordinate test-case i/p & o/p.
3. The whole cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure.

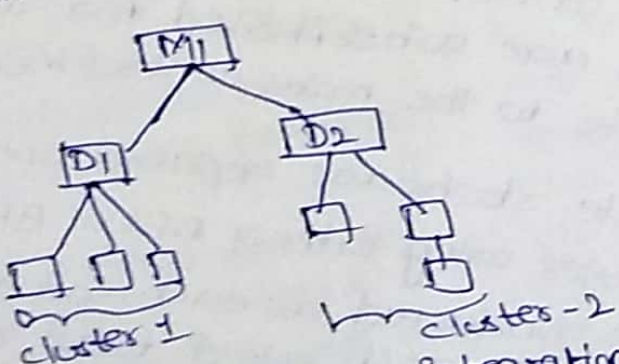


fig: Bottom-up integration testing

⇒ Following activities need to be carried out in smoke testing.

1) software components already translated into code are integrated into a "build." The "build" can be data files, libraries, reusable modules or program components.
2) A series of tests are designed to expose errors from build so that the "build" performs its functioning correctly.
3) The "build" is integrated with the other builds and the entire product is smoke tested daily.

⇒ Validation Testing:-

→ The integrated s/w is tested based on requirements to ensure that the desired product is obtained.

→ In validation testing the main focus is to uncover errors in

* System i/p or o/p
* System functions & information
* user interfaces
* system behaviour & performance

⇒ Acceptance Testing:-

The acceptance testing is a kind of testing conducted to ensure that the s/w works correctly in the user work environment. The acceptance testing can be conducted over a period of weeks or months.

Difference b/w Alpha & Beta testing:-

Alpha testing	Beta Testing
1. This testing is done by a developer or by a customer under the supervision of developer in company premises.	1. This testing is done by the customer without any interface of developer and is done at customer's place.
2. Sometime full product is not tested using alpha testing & only core functionalities are tested.	2. The complete product is tested under this testing. such product is usually given as free trail version.

→ First components are collected together to form cluster1 and cluster2. Then each cluster is tested using a driver program.

→ The clusters subordinate the driver module. After testing the driver is removed and clusters are directly interfaced to the modules

⇒ Regression Testing:-

* Regression testing is used to check for defects propaga -ted to other modules by changes made to existing program. Thus regression testing is used to reduce the side effects of the changes.
* there are three different classes of test cases involved in regression testing.
  * Representative sample of existing testcases is used to exercise all s/w functions.
  * Additional test cases focusing s/w functions likely to be affected by the change.
  * tests cases that focus on the changed s/w compo -nents.

→ After product had been deployed, regression testing could be necessary because after a change has been made to the product an error that can be discovered and it should be corrected.

→ similarly for for deployed product addition of new features may be requested and implemented.

⇒ 4. smoke testing:-

→ the smoke testing is a kind of integration testing technique used for the time caitical projects wherein the project need to be assessed on frequent basis.

Black-Box testing:-

→ Black-box testing is defined as a testing technique in which functionality of the Application under test(AUT) is testing.

→ In black-box-testing the application is tested without looking at the internal code structure, implementation details and knowledge of internal paths of s/w.

→ This type of testing is based on entirely on s/w requirements and specifications.

→ In black-box testing we just focus on inputs & outputs of the s/w system without bothering about internal knowledge of the s/w program.

Eg:- o/s like windows, website like google.

→ under black-box testing, you can test these applications by just focusing on the i/p's & o/p's without knowing their internal code implementation.

How to do black-box testing:

1. Initially, the requirements & specifications of the system are examined.
2. Tester chooses valid i/p's (positive test scenario) to check whether system processes them correctly. Also, some invalid i/p's (negative test scenario) are chosen to verify that the system is able to detect them.

→ Tester determines expected o/p's for all those i/p's.
→ s/w tester constructs test cases with the selected i/p's.
→ Test cases are executed.
→ s/w tester compares the actual o/p's with the expected o/p's
→ Defects if any are fixed and re-tested.

Types of black-box testing:-

1) Functional testing: This type of testing is related to functional requirements of a system.

2) Non-functional testing: This type of testing is not related to testing of specific functionality. but non-functional requirements such as performance, scalability etc.

3) Regression testing: Regression testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

Black-box testing Techniques:-

i) Equivalence class testing:-
→ Equivalence partitioning is also known as equivalence class partitioning.
→ In equivalence partitioning, inputs to the s/w or system are divided into groups.
→ Each & every condition of particular partition works as same as other. If a condition in a partition valid, other conditions are valid too. If a condition in partition in invalid, other conditions are invalid too.
→ It helps to reduce the total no. of testcases from infinite to finite. The selected testcases from these groups ensure coverage all possible scenarios.

Guidelines for equivalence partitioning.

* If i/p condition specifies a range, one valid and two invalid equivalence age defined
* If an i/p condition specifies a specific value, one valid & two invalid equivalence classes are defined.
* If an i/p condition specifies a member of a set, one valid & one invalid equivalence class is defined.
* If an i/p condition is boolean, one valid & one invalid equivalence class is defined.

Eg: We have to test a field which accepts age 18-56.

Age [Enter age] * Accepts value 18 to 56

Equivalence partitioning		
Invalid	valid	Invalid
<=17	18-56	>=57

Valid i/p : 18-56

Invalid i/p : <=17 & >=57

Valid class: 18-56 = pick any one i/p test data from 18-56.

Invalid class1: <=17 pick any one i/p test data less than or equivalence to 17

Invalid class 2 : >=57 pick any one i/p test-data greater than or equal to 57

2) Boundary value Analysis:-

→ Boundary value analysis is done to check boundary conditi-ons.

→ In this testing technique in which the elements at the edge of the domain are selected & tested.

→ using boundary value analysis, instead of focusing on i/p conditions only, the test cases from o/p domain are also derived.

⇒ Test cases for BVA accepting numbers b/w 1 & 1000 using BVA.

1) Test cases with test data exactly as the i/p boundaries of i/p domain i.e values 1 & 1000 in our case.

2) Test data with values just below the extreme edges of i/p domains i.e values 0 & 999.

3) Test data with values above the extreme edges i/p domain i.e values 2 & 1000|

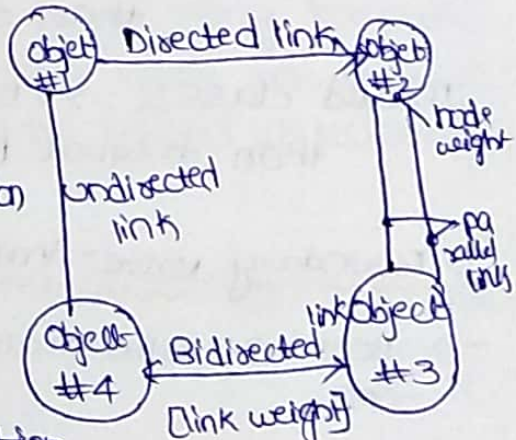
→ Boundary value analysis is often called as a part stress and negative testing.

3) Graph-based testing:-

→ In the graph based testing, a graph of objects present in the system is created.

→ The graph is basically a collection of nodes & links. Each node represents the object that is participating in the s/w system and links represents the relationship among these objects.

→ The node weight represents the properties of object & link weight represents the properties or characteristics of the relationship of the objects.



→ After creating the graph, important objects & their relationships are tested.

⇒ White-box testing :-

→ White-box testing is defined as the testing of a s/w solution's internal structure, design, and coding.

→ In this type of testing, coding is visible to the tester.

→ It focuses primarily on verifying the flow of i/p's & o/p's through the application, improving design & usability

→ White-box testing is also known as clear box testing, open-box testing, structural testing, glass-box testing, Transparent-box testing, code-based testing.

→ White-box testing involves the testing of the s/w code for the following:

* Internal security holes.
* Broken or poorly structured paths in the coding processes.
* The flow of specific inputs through the code.
* Expected output.
* The functionality of conditional loops.
* Testing of each statement, object, & function on an individual basis.

Testing techniques:-

i) Basis path testing:

→ In this method the procedural design using basis set of execution path is tested. This basis set ensures that every execution path will be tested at least once.

⇒ Flowgraph Notation:-

→ path testing is a structural testing strategy. This method is intended to exercise every independent execution path of a program atleast once.

→ Following are the steps that are carried out while performing path testing.

step1: Design the flowgraph for the program or a component.

step2: calculate the cyclomatic complexity.

step3: select a basis set of path.

step4: Generate testcases for these paths.

Eg:-

Step1: Design the flowgraph for the program or a component.

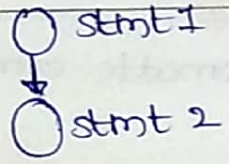
flowgraph is a graphical representation of logical control flow of the program. such a graph consists of circle called a flowgraph node which basically represents one or more procedural statements. and arrows called as edges or links. which basically represent control flow. In this flow graph the areas bounded by nodes and edges are called regions.

→ Eg: program for searching a number using binary search method. Draw a flow graph for the same.

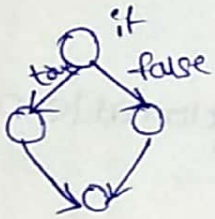
```
Void search (int key, int n, int a[])
{
    int mid;
1   int bottom = 0;
2   int top = n-1;
3   while (bottom ≤ top)
    {
4       mid = (top + bottom) / 2;
5       if (a[mid] == key)
        {
6           printf ("Element is present");
```

```
7 return ;
  } // end of IF
  else
  {
8 if (a[mid] < key)
9 bottom = mid + 1;
  else
10 top = mid - 1;
  } // end of else
  } // end of while
11 } // end of search
```

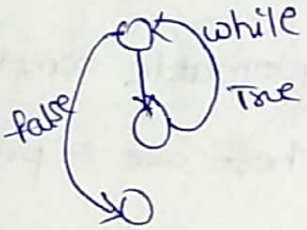
sequence



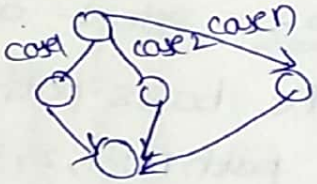
if-else



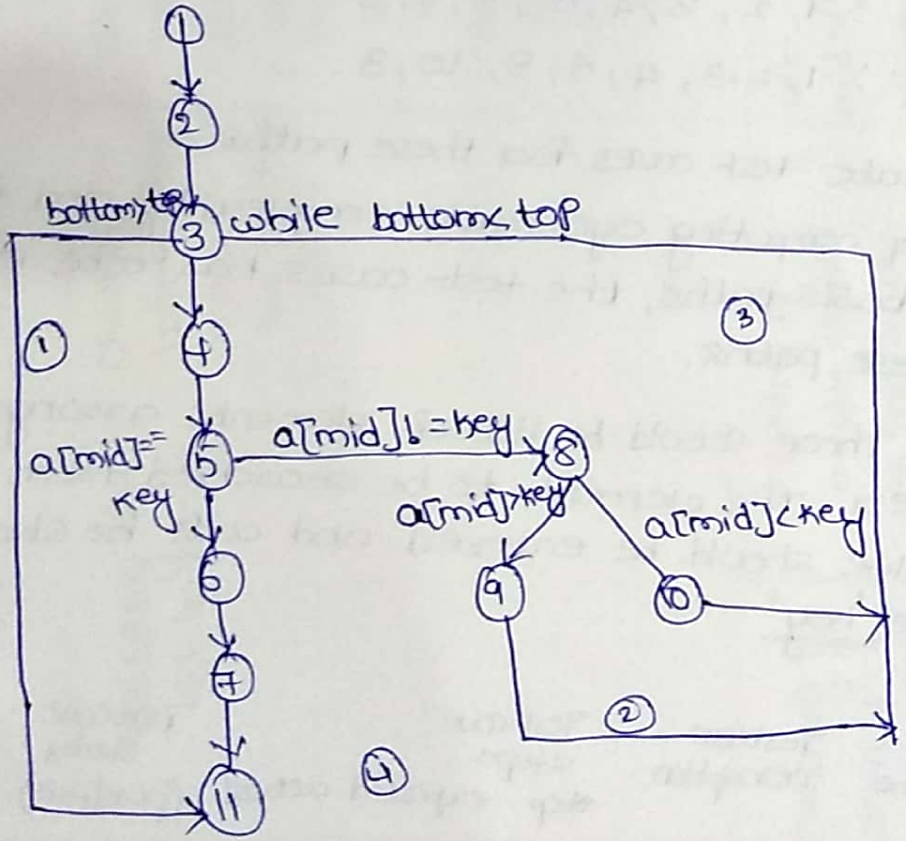
while



case



→ flow graph will be for binary search



Step 2: calculate the cyclomatic complexity.

→ The cyclomatic complexity can be computed by 3 ways

1) cyclomatic complexity = Total no. of regions in the flo -w graph = 4

2) cyclomatic complexity = E - N + 2
= 13 - 11 + 2
= 2 + 2
= 4

3) cyclomatic complexity = P + 1 = 3 + 1 = 4
There are 3 predicate nodes 3, 5, 8.

Step 3:- select a basis set of path.

The basis paths are

path 1 : 1, 2, 3, 4, 5, 6, 7, 11
path 2 : 1, 2, 3, 11
path 3 : 1, 2, 3, 4, 5, 8, 9, 3
path 4 : 1, 2, 3, 4, 5, 8, 10, 3 - -

Step 4: Generate test cases for these paths.

After computing cyclomatic complexity and finding independent basis paths, the test cases has to be execu -ted for these paths.

precondition: There should be list of elements arranged in ascending order. The element to be searched from the list, its value should be entered and will be stored in variable "key".

Test Case id	Test case name	Testcase Description	Test-case steps		Testcase status (pass/fail)	test priority
			step	expected	actual	

Defect severity.

Test case id	Test case name	Test case Description	Test steps		Test case status	Test stats (P/F)	TP	DS
			step	E A				
1.	validating the list boundary	checking the bottom & top values for the list of elements	set bottom=0 top = n-1 check if bottom<= top by while loop. This condition defines the length of the list from which the key searched.	Initially bottom≤ top will be true. But during iterations list's length will be reduced & if entire list gets scanned at one point (bottom≥ top) will be reached then return to main.	Design			
2.	checking list element with key	checking if middle element of array is equal to key value	set mid = (top+bottom)/2 Then compare if a[mid] is equal to key.	If a[mid]=key value then print message "element is present" & return to main	Design			

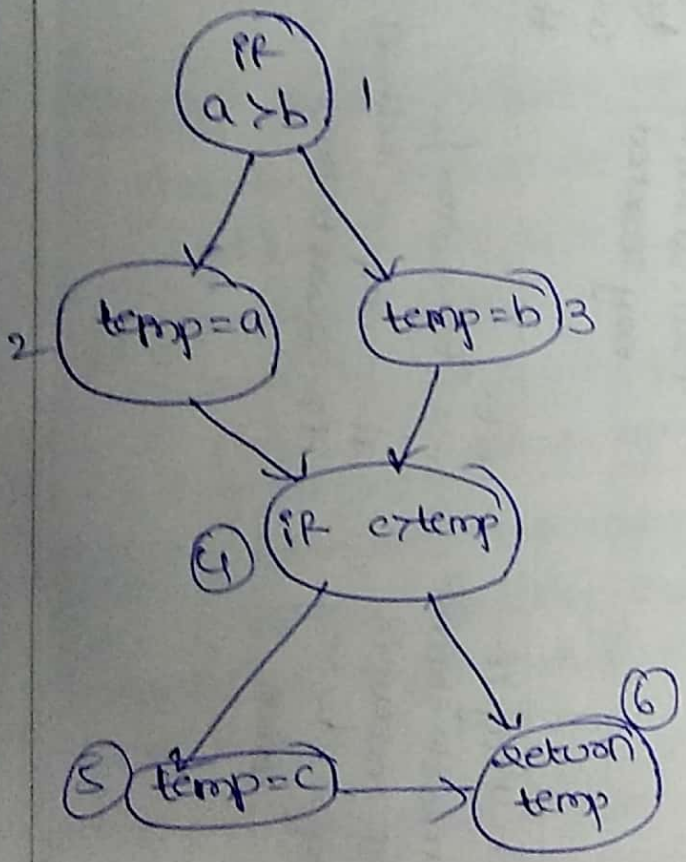
2) Graph Matrices:-

Graph matrix is a square matrix whose size is equal to no. of nodes of the flow graph.

→ For computing cyclomatic complexity following steps are adopted.

Eg: step 1: create a graph matrix. Mark the corresponding entry as 1 if node A is connected to node B.

step 2: count total no. of 1's from each row & subtract 1 from each corresponding each row.



step 3 Add cyclomatic complexity

	1	2	3	4	5	6	
1		1	1				2-1=1
2				1			1-1=0
3				1			1-1=0
4					1	1	2-1=1
5						1	1-1=0
6							3

step

⇒ System testing:-

→ The system test is a series of tests conducted to fully the computer based system.

→ Various types of system tests are

1. Recovery testing
2. security "
3. Stress "
4. performance "

→ The main focus of such testing is to test -

* system functions & performance.
* system reliability & Recoverability
* system installation
* system behavior in the special conditions
* system user operations.
* H/w & S/w integration & collaboration.
* Integration of external S/w & the system

⇒ 1) Recovery testing :-

→ Recovery testing is intended to check the system's ability to recover from failures.

→ In this type of testing the S/w is forced to fail and then it is verified whether the system recovers properly or not

→ For automated recovery then reinitialization, checkpoi -nt mechanisms, data recovery are restart are verified.

⇒ 2) security testing :-

→ security testing verifies that system protection mech -anism prevent improper penetration or data alterna -tion.

→ It also verifies that protection mechanisms built into the system prevent intrusion such as unauthorized internal or external access.

→ system design goals is to make the penetration attempt more costly than the value of the information that will be obtained

⇒ 3. stress testing:-

→ Determines breakpoints of a system to establish maximum service level.

→ In stress testing the system is executed in a manner that demands resources in abnormal quantity, frequency or volume.

→ A variation of stress testing is a technique called sensitivity testing.

→ The sensitive testing is testing in which it is tried to uncover data from a large class of valid data that may cause improper processing.

⇒ 4. performance testing:-

→ performance testing evaluates run-time performance of s/w, especially real time s/w.

→ In performance testing resource utilization such as CPU load, throughput, response time, memory usage can be measured.

→ For big systems involving many users connecting to servers & performance testing very difficult.

→ Beta testing useful for performance testing.

⇒ control structure testing:

* The structural testing is sometime called as white-box testing.
* objective of structural testing is to exercise all program statements.

→ The conditional testing is used to test the logical conditio -ns in the program.

→ The condition can be boolean condition or relational Ex -pression.

→ The condition is incorrect, in following situations.

1) Boolean operator is incorrect, missing or extra.
2) Boolean variable is incorrect.
3) Boolean paranthesis may be missing.
4) Error in relational operator.
5) Error in arithmatic expression.

→ The condition testing, focuses on each testing condition in the program.

→ The branch testing is condition testing strategy in which for a compound condition each & every true or false branches are tested.

→ The domain testing is a testing strategy in which rela -tional expression can be tested using three or four tests.

⇒ Loop Testing:-

Loop testing is a white-box testing testing techniq -ue. which is used to test the loop constructs.

⇒ Basically there are 4 types of loops.

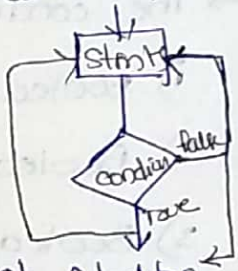
i) simple loop
ii) Nested loop
iii) concanterated loops
iv) unstructured loops.

1) Simple loops:-

→ The tests can be performed for n no. of classes.

i) n=0 that means skip the loop completely
ii) n=1 that means one pass through the loop is tested.
iii) n=2 that means two passes through the loop is tested.
iv) n=m that means testing is done when there are m passes where m<n.
v) perform the testing when no. of passes are n-1, n, n+1.

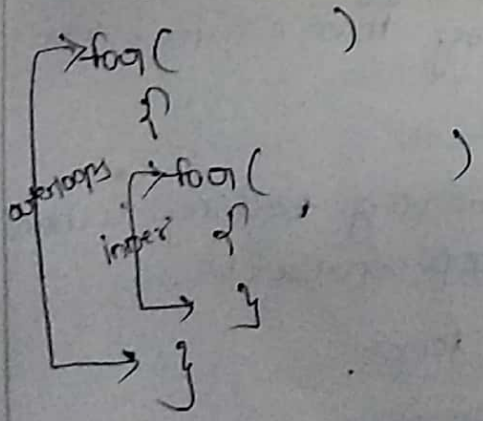
Eg foo ( )
{
y = .
}

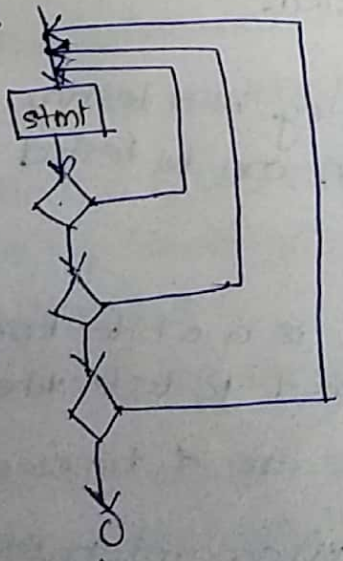


2) Nested loops:-

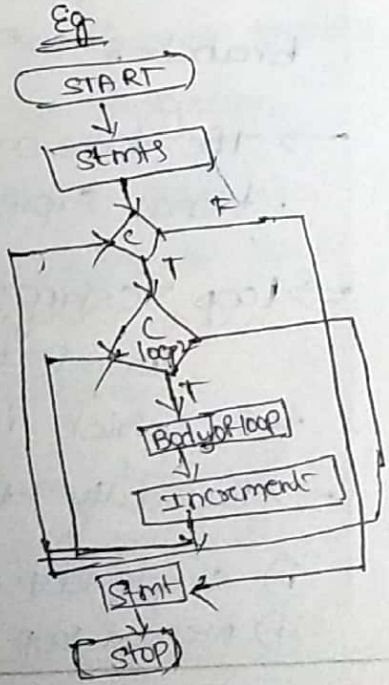
→ The nested loop can tested as follows.

1) Testing begins from the innermost loop first. At the same time set all the other loops to minimum values.
2) The simple loop test for innermost loop is done.
3) conduct the loop testing for the next loop by keeping the outer loops at the minimum values and other nested loops at some specified value.
4) This testing process is continued until all the loops have been tested.





Eg



3) Concanterated loops:-

The concanted loops can be tested is the same manner as simple loop tests.

4) unstructured loops:-

The testing can't be effectively conducted for a unstructu -red loops. Hence these types of loops needs to be redesign -ed.

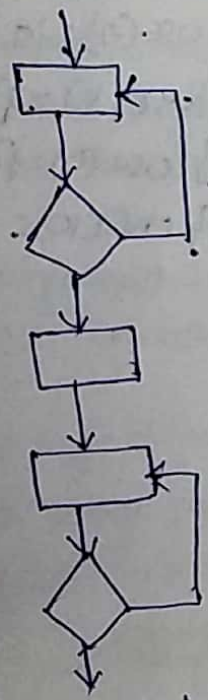


fig: concanted loops.

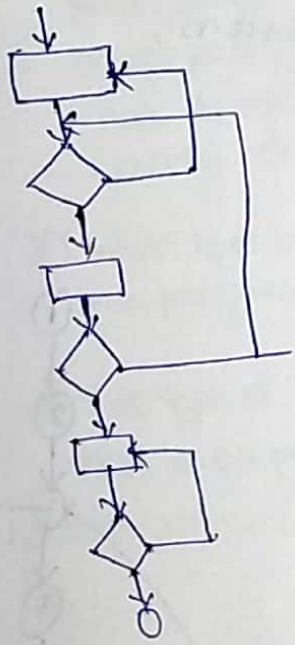


fig: unstructured loops.

⇒ Dataflow testing:-

→ The testing based on dataflow mechanism performs testing on definitions and uses of variables in the program.

→ In this method definition & use chain is required. the DU chain is obtained by identifying the def & use pairs from the program structure.

→ This testing is also called DU testing strategy.

* set DEF(n) contains variables that are defined at node n.
* set USE(n) contains variables that are read or used in at node n.

Eg:

```
1 S: =0;
2 a: =0;
3 while (a<b) {
4    a: =a+2;
5    b:= b-4;
6    if (a+b<20)
7    S: = S+a+b;
8    else
9       S: = S+a-b;
10 }
```

⟹

DU chain will be

DEF(1) = {S} USE(1) = {ϕ}

DEF(2) = {a} USE(2) = {ϕ}

DEF(3) = {ϕ} USE(3) = {a,b}

DEF(4) = {a} USE(4) = {a}

DEF(5) = {b} USE(5) = {b}

DEF(6) = {ϕ} USE(6) = {a,b}

DEF(7) = {S} USE(7) = {S,a,b}

DEF(8) = {S} USE(8) = {S,a,b}

DEF(9) = {ϕ} USE(9) = {ϕ}

DEF(10) = {ϕ} USE(10) = {ϕ}

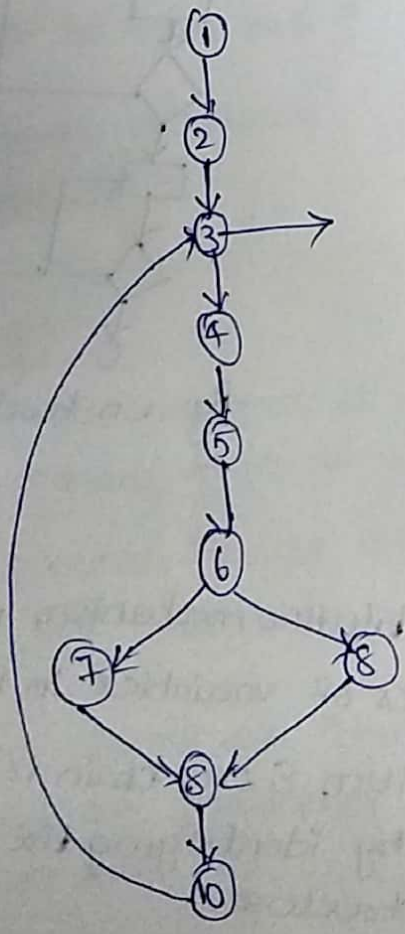


fig flowgraph

⇒ comparison b/w black box & white-box testing:-

Blackbox testing	Whitebox testing
1) Black-box testing is called behavioural testing.	1) White box testing is called glass box testing.
2) Black box testing examines some fundamental aspect of the system with little regard for internal logical structure of the soft.	2) In white box testing the procedural details, all the logical paths, all the internal data structures are closely examined.
3) During black-box testing the program cannot be tested 100%.	3) White box testing lead to test the program thoroughly.
4) This type of testing is suitable for large projects.	4) This type of testing is suitable for small projects.

⇒ list of debugging:-

→ Debugging is a process of removal of a defect. It occurs as a consequence of successful testing.

→ Debugging process starts with execution of test cases.

→ The actual test results are compared with the expected results.

→ The suspected causes are identified & additional tests or regression tests are performed to make the system to work as per requirement.

common approaches in debugging:

1) Brute-force method: The memory dumps and run-time traces are examined and program with write statements is loaded to obtain clues on error causes.

→ In this method " Let computer find the error" approach is used.

→ This is the best efficient method of debugging.

2) Back-tracking method :- This method is applicable to small program. In this method, the source code is examined by looking backwards from symptom to potential causes of errors.

3) cause elimination method :- this method uses binary partitioning to reduce the no. of locations where errors can exist.

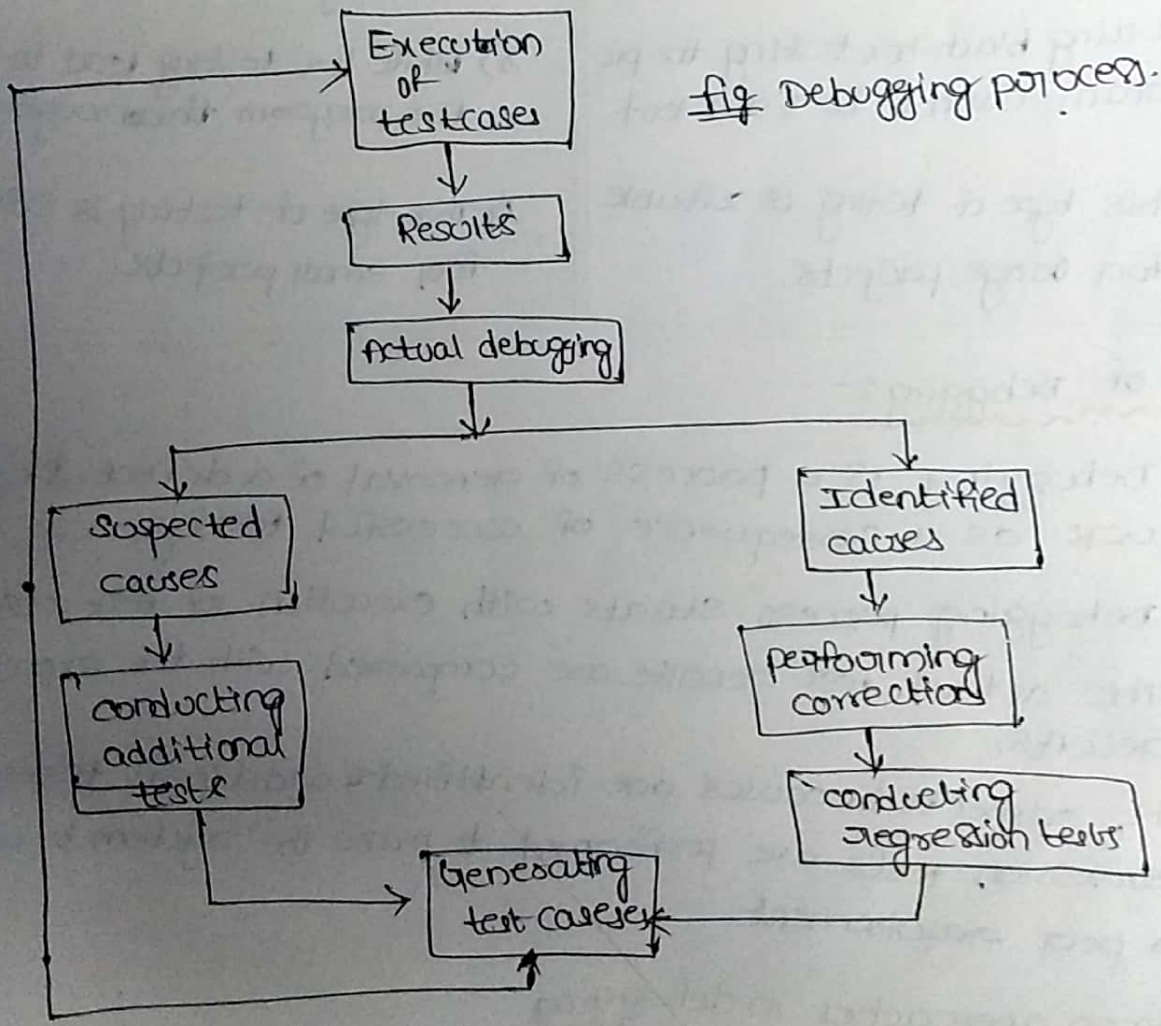


fig Debugging process.

→ Testing Vs Debugging :-

Testing	Debugging
1) Testing is a process in which the bug is identified	1) Debugging is the process in which the bug or error is corrected by programmer.

2) In testing process, it is identified where the bug occurs

3) Testing starts with the execution results from the testcases.

2) In debugging rootcause of error is identified.

3) Debugging starts only after the testing process.

UNIT-IV (Part-I)

Product metrics

* High quality software is an important goal in slw development.
* Slw quality is conformance to explicitly stated functional and performance requirements, explicitly documented development standards and implicit characteristics that are expected to slw development.

Mc Call's Quality factors:—

The factors that affect slw quality can be categorized into two broad categories.

1) factors that can be directly measured
   (eg: defects uncovered during testing)
2) factors that can be measured indirectly
   (eg: usability or maintainability)

* Mc Call's Richards and coalters propose a category of factors that affect slw quality.
  1) operational characteristics
  2) Ability to undergo change
  3) ability to undergo new environment.

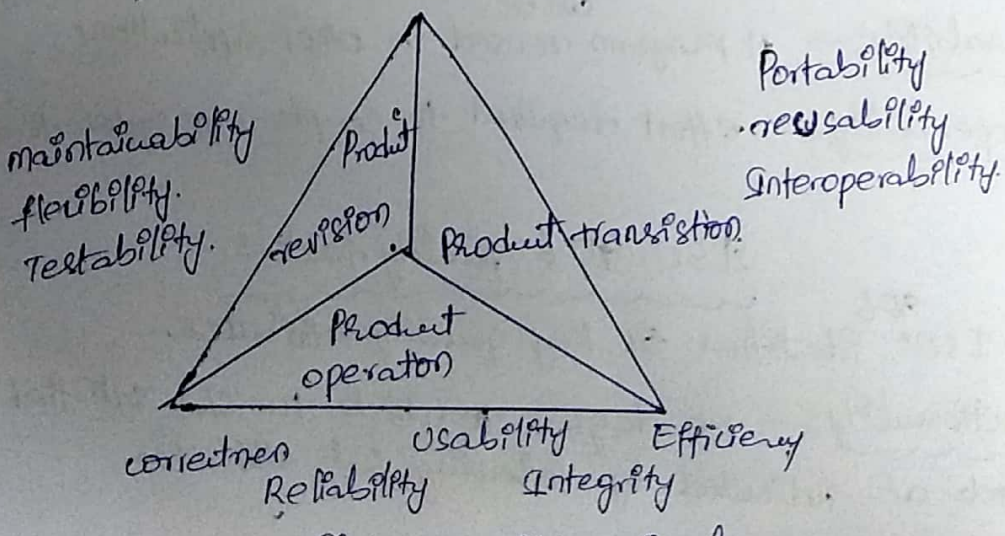


Fig: McCall's triangle.

* McCall's quality factors are as valid today as they were in 1970's. So it is clear that factors that affect slw quality do not change with time.

(i) Correctness :- the extent to which program satisfies its specification and fulfils the customer's objectives

(ii) Reliability :- the extent to which a program can be expected to perform its intended function with required precision.

(iii) Efficiency :- the amount of computing resources and code needed by a program

(iv) Integrity :- the extent to which slw or data controlled from unauthorized accen.

(v) Usability :- the effort required to learn, operate, Prepare input. and interpret output of a program.

(vi) Maintainability :- effort required to locate and fix an error in a program.

(vi) flexibility :- the effort required to modify an operational program

(vii) Testability :- the effort required to test and ensure correctness of the program.

(viii) Portability :- the ability of the slw to work properly even if the environment get changed

(ix) Reusability :- A program can be reused in other applications

(x) Interoperability :- effort required to couple one system to another.

ISO 9126 quality factors

The ISO 9126 identifies six key quality attributes.

1) Functionality :- the degree to which the slw satisfied stated needs as indicated by following sub attributes.

(i) sustainability (ii) accuracy (iii) interoperability
(iv) compliance (v) security.

(2) Reliability ← maturity
fault tolerance
recoverability.

3) usability ← understandability
learnability
operability.

(4) Efficiency :— the degree to which the s/w makes optimal use of system resources as indicated by the following.
(i) time behavior (ii) resource behavior.

(5) maintainability :— the ease with which repair may be made to the s/w as indicated by the following.
→ Analyzability
→ changeability
→ stability
→ Testability.

(6) portability :— the ease with which the s/w can be transposed from one environment to another
→ adaptability
→ Instability
→ conformance
→ replaceability.

A framework for Product metrics.

A set of basic principles for the measurement of Product metrics for S/w.

Measures, metrics and Indicators.

Measure:- It is a quantitative indication of the extent amount, dimension, or size of the some attribute of a Product or process.

Metric:- It is the degree to which a system, component, or Process possess a given attribute. The s/w metrics relate several measures.

for eg:- Average no. of errors found per review.

Indicators:- Indicators mean combination of metrics that Provides insight into the s/w process, project or Product.

Purpose of Product metrics:-

→ Aid in the evaluation of analysis and design models.
→ Provide an indication of the complexity of Procedural designs and source code.
→ facilitate the design of more effective testing techniques
→ Assess the stability of a s/w product

Measurement principles or activities of a measurement process

In the measurement process, first we formulate, collect, analysis, Interpretation, & feedback.

1. formulation:-

* The appropriate s/w measures and metrics should be considered for the representation of the s/w.

2. Collection :- The mechanisms used collect the results or data obtained from the formulated metrics.

3. Analysis :- The analysis should be made on the computation of metrics and application of mathematical tool.

4) Interpretation :- The evaluation metrics provides the insight for the s/w quality.

5) feedback :- Interpretations obtained from product metrics must be submitted to the s/w team for review and feedback.

Goal oriented s/w measurement

* Goal / question oriented s/w measurement (GQM) is a technique for identifying meaningful metrics for any part of s/w process.
* For applying this technique following are the requirements.

① The explicit measurement goals must be established which is based on process activity or process characteristics

② Prepare set of questionnaire which will help to find out measurement goals

③ ~~Finded~~ identify well formulated metrics that will help to answer the prepared set of questions.

Goal Definition Template :-

Analyze {name of activity to be measured}
Purpose of {object or purpose}
with respect to {activity or attribute that is to be considered}
from the viewpoint of {stakeholders performing measurement}
context of {environment in which the measurement takesplace}.

Attributes of effective S/w metrics

Effective S/w metrics should have following attributes

(1) simple and computable:—

The derivation of metric should be easy to compute and should not be a time consuming activity.

(2) empirically and intuitively persuasive:—

* It should be immediate and can be derived based on observations

(3) Consistent and objective:—

* The metric should produce unambigous results anybody should get the same result by using metrics when same set of information is used.

(4) consistent in its use of units and dimensions:-

* the mathematical units and dimensions used for the metric should be consistent. And these should not be intermining of units.

(5) programming language independent:-

* The metric should be based on analysis model, design model and program structure. It should be independent of programming languages, syntax, or semantic of any programming language.

Metrics for Analysis Model :-

* Metrics for the analysis model are useful in estimating the project. In order to determine the metrics in analysis model "size" of the s/w is used as a measure.

Function point model :-

* The function point model is based on functionality of the delivered application.
* These are generally independent of the programming lang used.
* This method is developed by Albrecht in 1979.
* Using historical data, function points can be used to
  → estimate the cost or effort required to design, code and test the s/w
  → predict the number of errors encountered during testing.
  → Forecast the number of projected source code lines in the implemented system.

How to calculate function point :-

The data for following information domain characteristics are collected to calculate function point.

1. Number of user inputs : Each user input that provides distinct data to the s/w is counted.
2. Number of user outputs : Each user output that provides information to the user is counted. (reports, o/p screens, error messages etc)
3. Number of user inquiries :-
   An inquiry is that an on-line input that results in the generation of some immediate s/w response

in the form of an on-line output. (eg: google search)

4. No. of files:- Each logical master file (i.e. large database or separate file) is counted.

5. Number of external interfaces:-

All machine readable interfaces (eg: data files on storage media) that are used to transmit information to another system are counted.

Function point computation:-

The process involved in function point computation is

1. Identify / collect the information domain values
2. Complete the table as shown below to get the count total.
3. * Associate a weighting factor (ie. Complexity Value) with each count based on criteria established by the s/w development organization.

3. Evaluate and sum up the adjustment factors.
"F_i" refers to 14 value adjustment factors, with each ranging in value from 0 (not important) to 5 (absolutely essential)

4. Compute the number of function points (FP)

$$Fp = \text{count total} * [0.65 + 0.01 * \text{Sum}(F_i)]$$

* cost total can be computed with the help of below table.

Information Domain Value	count	weighting factor simple	Average	complex	
1. External inputs (EIs)	☐ x	3	4	6	=
2. External outputs (EOs)	☐ x	4	5	7	=
3. External inquiries (EQs)	☐ x	3	4	6	=
4. Internal Logical files (ILFs)	☐ x	7	10	15	=
5. External interface files (EIFs)	☐ x	5	7	10	=
		cost total →			☐

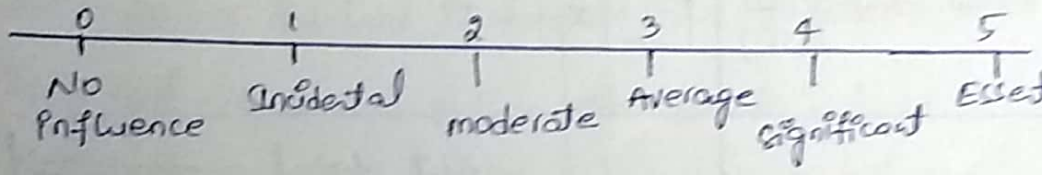
* The weighting factors should be determined by observations or by experiments.

* Now the sw complexity can be computed by answering following questions.

1. Does the system need reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance of the system critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations.

8. Are the master files updated on-line?
9. Are the Inputs, output, files or inquiries complex?
10. Is the internal processing complex?
11. Is the code which is designed being reusable?
12. Are conversation and installation included in the design?
13. Is the system designed for multiple installations in different Organizations?
14. Is the application designed to facilitate change and ease of use by the user?

* Rate each of the above factors according to the following scale:



* once function point is calculated then we can compute various measures as follows.

→ Productivity = FP / person-month.

→ Quality = Number of faults / FP

→ Cost = $ / FP

→ Documentation = pages of documentation / FP.

Function point Example:—

Domain value	count		weighting factor simple	average	complex	
External Inputs	3	x	3	4	6	= 9
External outputs	2	x	4	5	7	= 8
External inquiries	2	x	3	4	6	= 6
Internal Logical files	1	x	7	10	15	= 7
External interface files	4	x	5	7	10	= 20
	count total = 50					50

$FP = count\ total * [0.65 + 0.01 * sum(F_i)]$

$FP = 50 * [0.65 + (0.01 * 46)]$

$FP = 55.5$ (rounded up to 56)

=

Imp * **Metrics for specification quality**

To assess the quality of analysis model and corresponding requirements Davis and his colleagues has suggested some characteristics. These characteristics are.

* Completeness * correctness
* Understandability * verifiability
* Internal and external consistency
* Achievability
* Concision * Traceability
* modifiability * precision
* Reusability.

The total requirements in the specification can be specified as n_r

where n_r

~~where~~ $n_r = n_f + n_{nf}$

where

n_r = Total number of requirements

n_f = Total number of functional requirements

n_{nf} = Total number of non functional requirements.

* Davis has suggested the metric for Specificity of req as

$Q_1 = n_{ui} / n_r$

where n_{ui} is the number of requirements that have unique interpretation and n_r is the total no. of requirements

* Completeness of functional requirements is given by.

$$Q_2 = n_u / [n_i * n_s]$$

where

n_u = Number of unique functional requirements

n_i = Number of input given in the specification

n_s = Number of states specified.

Thus Q_2 gives the percentage of necessary functions but here it does not consider the non functional requirements.

* The overall metric ~~for~~ (even by considering the non functional requirements) for completeness can be obtained by validating the requirements.

$$Q_3 = n_c / [n_c + n_{nv}]$$

where

n_c is number of requirements that are validated as correct

n_{nv} is the number of requirements that are not been validated.

Metrics for Design model :-

* metrics for design model focus on determining the measurement of design quality. These metric guides the SW design activity as design evolves.

* Design model metrics ~~to~~ considers three aspects

1) Architectural design
2) Object oriented design
3) User interface design.

Architectural Design metrics :-

While determining the architectural design primarily characteristics of program architecture are considered. It does not focus on inner working of the system.

Metrics by Card and Glass :-

Two scientists Card and Glass has suggested three design complexity measures as

1) Structural complexity :-

Structural complexity depends upon the fan-out for modules. It can be defined as

$$S(K) = f^2_{out}(K)$$

where fout represents fan-out for module k.
(fan out means number of modules that are subordinating module k).

2) Data Complexity :-

Data complexity is the complexity within the interface of internal module for some module k it can be defined as

$$D(K) = \frac{tot\_Var(K)}{[f_{out}(K)+1]}.$$

where tot_var is the total number of input and output variables going to and coming out of the module.

3) System complexity :-

System complexity is the combination of structural and data complexity. It can be denoted as

$$Sy(K) = S(K) + D(K)$$

* when structural, data and system complexity get increased the overall architectural complexity also get increased.

Metrics for Object-oriented Design:-

Whitmire has suggested nine measurable characteristics of object oriented design and those are.

1) Size :- It can be measured using following factors.

2) Complexity:- It is a measure representing the characteristics that how the classes are interrelated with each other.

3) Coupling:- It is a measure stating the collaborations between classes or number of messages that can be passed between objects.

4) completeness :- It is the measure representing all the requirements of the design component

5) cohesion:- It is the degree by which the set of properties that are working together to solve particular property.

6) primitiveness:- the degree by which the operations are simple. In other words, the measure by which number of operations are independent upon other.

These Product metrics for object-oriented design is applicable to design as well as analysis model.

CK Metrics Suite

CK have Proposed six class-based design metrics for object oriented systems.

1. weighted methods per class (wmc):-

* Assume that n methods of complexity $c_1, c_2, \ldots c_n$ are defined for a class C
* The specific complexity metric that is choosen (eg., cyclomatic complexity) should be normalized so that normal complexity for a method takes on a value of 1.0

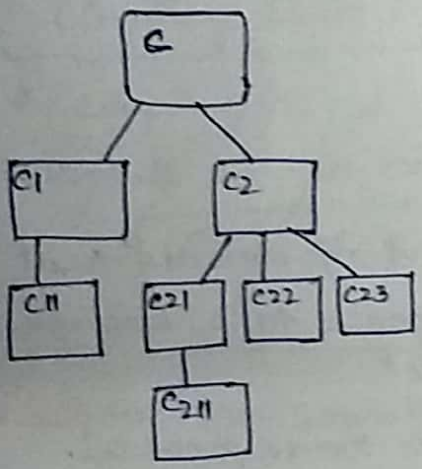
$$WMC = \Sigma c^i$$

for i = 1 to n. The number of methods and their complexity are reasonable indicators of the amount of effort required to implement and test a class.

* So. If no. of methods are increased, complexity of class also increased. Therefore, limiting potential reuse.

2. Depth of the Inheritance (DIT)

* This metric is "the maximum length from the node to the root of the tree".



* Refering to figure, the value of DIT for the class ~~hier~~ hierarchy shown is 4.
* As DIT grows, it is likely that lower-level classes will inherit many methods. This leads to potential ~~difficulties~~ difficulties when attempting to predict the behavior of a class.
* A Deep class hierarchy is also leads to greater design complexity.
* on the positive side, large DIT value imply that many methods may be reused.

3. Number of children (NOC) :-

* The Subclasses that are immediately sub ordinate to a class in the class hierarchy are termed as its children.
* refering to previous figure, class C2 has three children - Subclasses C21, C22 and C23.
* As the number of children grows, reuse increases, the abstraction represented by the Parent class can be diluted.

* As Noc increases, the amount of testing will also increase.

4. Coupling between object classes (CBO):— * CBO is no. of collaborations b/w the classes
* The CRC model may be used to determine the value for CBO
* CBO is the number of collaborations listed for a class on its CRC Index card.
* As CBO increases, it is likely that the ~~reuse~~ reusability of a class will decrease.
* If values of CBO is high, then modification get complicated.
* Therefore, CBO values for each class should be kept as low as is reasonable.

5. Response for a class (RFC)

* Response for a class is "a set of methods that can potentially be executed in response to a message received by an object of that class"
* RFC is the number of methods in the response set.
* As RFC increases, the effort required for testing also increases because the test sequence grows. as well as overall design complexity of the class increases.

Lorenz and Kidd metrics suite.

Lorenz and Kidd have proposed the conceptual division of class based metric into four distinct categories.

1) size 2) Inheritance 3) Internal 4) external.

* Size-oriented metrics for the OO class focus on counts of attributes and operations for an individual class.
* Inheritance-based metrics focus on the manner in which operations are reused through the class hierarchy.
* metrics for class internals look at cohesion and code oriented issues, and external metrics examine coupling and reuse.

Imp

The MOOD Metrics Suite.

MOOD metrics suite is proposed by Harrison, Counsell and Nithi for object oriented design. It includes two metrics MIF and CF.

1. Method inheritance factor (MIF):-

$$MIF = \frac{\sum Mi(Ci)}{\sum Ma(Ci)}$$

* The degree to which the class architecture of an OO system makes use of inheritance for both methods. (operations) and attributes is defined.
* Value of MIF indicates impact of inheritance on the OOS/w.

* where Ci is a class within the architecture
Mi(Ci) is number of methods inherited in Ci
Md(Ci) number of method declared in Ci

2. Coupling factor (CF):-

* Coupling is an indication of the connections between elements of the OO design.

$$CF = \sum_i \sum_j is\_client(C_i, C_j) / (T_c^2 - T_c)$$

i and j varies from 1 to total number of classes in the architecture Tc

* where the summation occur over i=1 to Tc and j=1 to Tc.
* Function (is_client) = 1, if and only if a relationship exist between the client class, Cc, and the server class, Cs, and Cc ≠ Cs. = 0 otherwise.
* As CF increases the complexity of object oriented design get increased.

Operation oriented metrics

* operation oriented metrics reside within a class.

(1) Average operation size:- (OSavg)

* Lines of code (Loc) could be used as an indicator for operation size.
* operation has some roles and responsibilities related to Product

* As the number of messages sent by a single operation increases, it is likely that responsibilities have not been well-allocated within a class.

2. Operation complexity (OC):-

* operations should be limited to a specific responsibility, the designer should strive to keep OC as low as possible.

3. Average number of parameters per operation (NPavg):-

* The larger the number of operation parameters, the more complex the collaborations blw objects.
* In general, NPavg should be kept as low as possible.

Metrics for source code.

Halstead has proposed in "software science" some software science metrics. These metrics are based on

* common sense
* Information theory
* psychology.

* In the proposed metrics the used measures are

n_1 = The number of distinct operators in the program.

n_2 = The number of distinct operands in the program.

The paired block such as {---}, or begin .. end or repeat... until are treated as single operator. The program length N can be defined as.

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

The program volume can be defined as

$$V = N \log_2 (n_1 + n_2)$$

The program volume is heavily dependent upon the program length

Let N_1 = Total count for all the operations in the program.

N_2 = Total count for all the operands in the program

The program volume ratio L can be defined as $L = 2/n_1 \times n_2/N_2$

Metrics for Testing.

Halstead's metrics for estimating the testing efforts are as given below.

The Halstead effort can be defined as

$$e = V/PL$$

where V is the program volume and PL is the Program Level.

The program level can be compated as

$$P_L = 1/[(n_1/2) \times (N_2/n_2)]$$

* The percentage of overall testing effort =

$$\frac{\text{testing effort of specific module}}{\text{testing efforts of all the module.}}$$

Metrics for maintenance

* The stability of sw product is given by an IEEE standard which suggest a metrics software maturity index (SMI) for that matter.
* It is given as follows.

$$SMI = (M - (A + C + D))/M$$

where,

M = Number of modules in current version

A = Number of added modules in current version

C = Number of changed modules in current version

D = Number of deleted modules in current version compared to previous version.

When SMI reaches to the value 1.0 the product becomes more and more stabilized

Syllabus

Metrics for Process and products :- Software measurement, metrics for s/w quality.

Product metrics outline

Product metrics

(I) Software Quality
- (i) mccall's Quality factors
- (ii) ISO 9126 Quality factors.

(II) A Framework for Product metrics
- (i) measures, metrics and Indicators
- (ii) The challenge of product metrics
- (iii) measurement principles
  - formulation
  - collection
  - Analysis
  - Interpreted
  - Feedback.
- (iv) Goal oriented s/w measurement
- (v) The attributes of effective s/w metrics.

(III) metrics for analysis model.
- FP (function point) model.
- metrics for specification quality

(IV) metrics for design model.
- → Architectural design metrics
  - metrics by card & Glan
    - Structural complexity
    - Data complexity
    - System complexity.
- → metrics for object-oriented design.
  - size
  - complexity
  - coupling
  - completeness
  - cohesion.
- → CLASS-Oriented metrics
  - → CK metrics suite.
    - WMC
    - DIT
    - NOC
    - CBO
    - RFC
  - → Lorenz and kidd metrics suite
  - → MOOD metrics suite
    - MIF
    - CF
  - (metrics for object oriented design)
  - → operation Oriented metrics
    - OSAvg
    - OC
    - NPavg.

(V) metrics for source code.

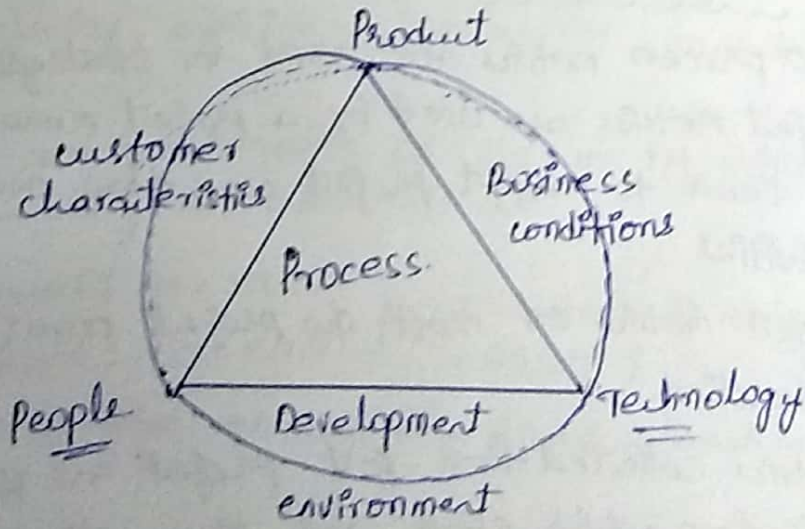
(VI) metrics for testing.

Process metrics and Software Process Improvement

* Process metrics are collected across all projects and over long periods of time. their intent is to provide a set of process indicators that lead to long-term sw process improvement.

* Project metrics enable a sw project manager to
  1) assess the status of an ongoing project.
  2) track potential risk
  3) uncover problem areas before they go "critical"
  4) Adjust work flow or tasks, and
  5) evaluate the project team's ability to control quality of sw work products.

* In making improvements to any sw system, there are three basic quality factors to consider: product, people and technology.



* Fig: Determinants for sw quality and organizational effectiveness.

* Process at the center connecting 3 factors that have a profound influence on sw quality and organizational.

Performance.

* The skill and motivation of people has been shown to be the single most influential factor in quality and Performance.
* The complexity of the Product can have a substantial impact on quality and team performance.
* The technology that populate the process also has an impact
* Praess triangle within the circle, specifies the environmental conditions such as
  → customer characteristics (communication and collaboration between user and developer)
  → Business conditions (organizational policies, Business rules)
  → Development Environment (use of new technologies, use of automated tools)

Project metrics

* s/w process metrics are used for strategic purposes.
* project metrics are used by a project manager and a s/w team to adapt project work flow and technical activities
* Project metrics on most s/w projects occurs during estimation.
* metrics collected from past projects are used as a basis from which effort and time estimates are made for current s/w work
* As a project proceeds, measures of effort and calendar time expended are compared to original estimates.

Intent of project metrics is

1) used to minimize the development schedule
2) used to assen product quality.

Software measurement

* measurement of slw can be classified into two categories.

1) Direct measures 2) Indirect measures.

Direct measures:-

* Direct measure of the slw Process includes cost and effort applied.
* Direct measures of the product includes lines of code, (LOC) produced, execution speed, memory size and defects reported over some set period of time.

Indirect measure:-

* Indirect measures of the product include functionality, complexity, efficiency, reliability, maintainability etc.
* The quality and functionality of slw or its efficiency or maintainability are more difficult.
* Team A found: 342 errors
  Team B found: 184 errors } which team is more efficient?
* It is depends on size or complexity (i.e functionality) of the projects.

Size Oriented metrics:-

* Size-oriented slw metrics are derived by normalizing quality and/or productivity measures by considering the size of the slw that was been produced.

* S/w organization can maintains simple records as shaonin fig.
* The table lists each S/w development project that has been completed over the past few years and corresponding measures for that project.

Project	LOC	Effort	$(cost)	Doc. (Pgs)	Errors	defects	people
ABC	10,000	20	170	400	100	12	4
PQR	20,000	60	300	1000	129	32	6
XYZ	35,000	65	522	1290	280	87	7

Table : Size measure.

* A simple set of size measure that can be developed as
  - → Size = thousand Lines of code (kLOC)
  - → Quality = No. of faults/KLOC
  - → Effort = person/month
  - → cost = $/LOC
  - → pages of documentation/kLOC

* Size-oriented metrics are widely used but there is a slight debate about validity and applicability.
* Size-oriented metrics are programming language dependent
* It is difficult to estimate LOC in the early stages of development.

Function oriented metrics

* It use a measure of functionality delivered by the application as a normalization value.
* Since 'functionality' cannot be measured directly, it must be derived indirectly using other direct measures.
* Function Point (FP) is widely used as function oriented metrics
* FP is based on characteristic of s/w information domain.
* FP is programming language independent.

Relationship between LOC and FP metrics

* Relationship between LOC and FP depends upon
  - → The programming language that is used to implement the s/w
  - → The quality of the design.
* FP and LOC have been found to be relatively accurate predicators of s/w development effort and cost.
  - → However, a historical baseline of information must first be established
* LOC and FP can be used to estimate object-oriented s/w projects.
  - → However, they do not provide enough granularity for the schedule and effort adjustments required in the iterations of an incremental process.

Programming language	Average	median	Low	high
Ada	154	-	104	205
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
COBOL	77	77	14	400
Java	55	53	9	214
VB	47	42	16	158

* The table above provides a rough estimate of the average LOC to one FP in various programming languages.

=

Object oriented metrics:-

1) No. of scenario scripts (i.e., use cases)

* This number is directly related to the size of an application and to the number of test cases required to test the program.

2) Number of ~~class~~ key classes (the highly independent components)

* Key classes are defined early in object-oriented analysis and are central to the problem domain.

* This number indicates the amount of effort required to develop the S/W.

* It also indicates the potential amount of reuse to be applied during development.

3) No. of support classes:-

→ Support classes are required to implement the system but are not immediately related to the problem domain. (e.g., user interface, database, computation)

4) Average number of support classes per key class :-

* Key classes are identified early in a project (eg., at requirements analysis)
* Estimation of the number of support classes can be made from the number of key classes.
* GUI applications have between two and ~~three~~ times more support classes as key classes.

5) Number of subsystems :-

* A subsystem is an aggregation of classes that support a function that is visible to the end user of a system.

Metrics for S/w quality

* The goal of S/w engineering is to produce a high-quality system, application, or product within timeframe that satisfies the market need.
* To achieve this goal, S/w engineers must apply effective methods with modern tools within the context of a mature S/w process.

Measuring quality :-

There are many measures of software quality.

correctness,
Maintainability,
Integrity,
Usability

} useful indicators for the project team.

1. Correctness :-

* Correctness is a degree to which the s/w produces the ~~desig~~ desired functionality. The correctness can be measured as

$$\text{correctness} = \frac{\text{No. of Defects}}{\text{per KLOC}}$$

where defect means lack of conformance to requirements. Such defects are generally reported by the user of the Program.

2) Maintainability :-

* This describes the ease with which a program can be corrected if an error is found, adapted if the environment changes, or enhanced if the customer has changed requirements.

* Mean time to change (MTTC) :- The time to analyze, design, implement, test and distribute a change to all users.

3) Integrity :-

* S/w Integrity has become increasingly important in the age of hackers and firewalls.

* This attribute measures a system's ability to withstand attacks to its security.

* Attacks can be made on all three components of s/w.
  - → programs
  - → Data
  - → Documents.

* To measure Integrity, two additional attributes must be defined.
  - → Threat
  - → Security.

Measuring Defect Removal Efficiency (DRE):-

While developing the s/w project many work products such as SRS, design document, source code are being created.

* Along with these work products many errors may get generated. project manager has to identify all these errors to bring quality software.
* Error tracking is a process of assessing the status of the s/w project.
* The s/w team performs the formal technical reviews to test the s/w developed. In this review various errors are identified and corrected. Any errors that remain uncovered and are found in later tasks are called defects.
* The defect removal efficiency can be defined as

$$DRE = E/(E+D)$$

where DRE represents Defect removal efficiency.
E is the error.
and D is defect.

* The DRE represents the effectiveness of quality assurance activities. The DRE also helps the project manager to assess the progress of s/w project as it gets developed through it scheduled work task.
* During errors tracking activity following metrics are computed

1. Errors per requirements specification page: denoted by Ereq
2. Errors per Component - design level : denoted by Edesign
3. Error per component - code level ; ~~def~~ denoted by Ecode.
4. DRE - requirement analysis.

DRE - architectural design

DRE - component level design.

DRE - coding.

Project manager calculates current values for.

Ereq, Edesign, and Ecode.

* These values are then compared with past projects. If the current result differs more than 20% from the average, then there may be cause for concern and investigation needs to be made in this regard.

* These error tracking metrics can also be used for better target review and testing resources.

UNIT-V
Risk Management (Part-1)

Introduction:-

→ "The risk denotes the uncertainty that may occur in the choices due to past actions and risk is something which causes heavy losses"

"The risk is nothing but it is uncertain event that may occ or may not occur in the future. That may affect +ve or -ve impact on the product."

Risk Management:-

→ Risk management is a methodology or a mechanism carried out throughout the development process to identify, manage and control risks evolved before & during software process.

→ Various activities carried out for risk management

i) Risk identification
ii) Risk projection
iii) Risk refinement
iv) RMMM.

→ Charette proposed a risk is categorized into known risks & unknown risk

→ known risks are software risks that are actually facts known to the team actually as well as to the entire project.

Eg:- Not having enough number of developers can delay the project delivery.

Unknown risks :- Those kind of risks about which the organization has no idea. These risks can't guess earlier.

⟹ Reactive & proactive risk strategy :-

→ Reactive and proactive risk strategies are the approaches used for managing the risk.

Reactive risk strategy :-

* Reactive risk management is a risk management strategy in when risk project gets into trouble then only corrective action is taken.
* When such risks can not be managed and new risks come up one after other, the slw team flies into action in an attempt to correct problems rapidly. These activities are called "firefighting activities"
* Resources are utilized to manage such risks. And if still the risks do not get managed then project is in danger.
* In this strategy no preventive action is taken about the risks. they are handled only on their occurances
* This is an older approach of risk management.

Proactive risk strategy :-

* proactive risk management strategy begins before the technical activity by considering the probable risks.
* In this strategy potential risks are identified first. then their probability & and impact is analyzed.
* such risks are the specified according to their priorities (i.e high priority risk should be managed first)

* Finally the slw team prepares a plan for managing these risks.
* The objective of this strategy is to avoid risks. But it is not possible to avoid all the risks, hence team prepares the risk management plan in such a manner that risk controlling can done efficiently.

⇒ Software Risks:-

1) The risk may or may not happen. It shows the uncertainty of the risks.
2) When risk occur, unconsequences or losses will occur.

Types of risk:-

1) project risk:- project risks arise in the slw develop-ment process then they basically effect budget, schedule, staffing, resources, and requirements. When project risks become severe then the total cost of the project gets increased.

2) Technical risk: These risks affect quality & timeliness of the project. If technical risks become reality the potential design implementation, interface, verification & maintenance problems gets created. Technical risks occur when problem becomes harder to solve.

3) Business risk:- When feasibility of slw product is in suspect then business risk occur.

→ Business risk can be further categorized as.

i) Market risk; When quality slw product is built but it the is customer for this product then it is called market risk.

ii) Strategic risk:- When a product is built and if it is not following the company's business policies then such a produ-ct brings strategic risk.

iii) Sales risk: When a product is built but how to sell it is not clear then such a situation brings sales risk.

iv) Management risk: When a senior management or staff responsible people leaves the organization then management risk occurs.

v) Budget risk: Losing the overall budget of the project is called budget risk.

⟹ Risk identification:-

→ Risk identification can be defined as the efforts taken to specify threats to the project plan. Risk identification can be done by identifying known & predictable risks.

→ Risk identification based on two approaches.

1) Generic risk identification: It includes potential threat identification to s/w project.

2) product-specific risk identification: It includes product specific threat identification by understanding people, technology & working environment in which the product gets built.

→ Generally Risk identification is done by project manager.

→ The project manager follows some risk item check list to identify risks in product.

1) product size: The risk items based on overall size of the s/w product is identified.

2) Business impact:- Risks items related to the market place or management can be predicted.

3) customer-characteristics: Risks associated with customer-developed communication identified.

4) Development Environment: The risk associated with the technology and tool being used for developing the product.

5) staff size & experience: once the technology and tool re-lated risks items are identified. it is essential to iden-tify the risks associated with sufficient, highly experi-enced and skilled staff who will do the development.

⇒ creating risk components & drivers list.

→ The set of risk components and drivers list is prepar-ed along with their probability of occurence. then their impact on the project can be analyzed.

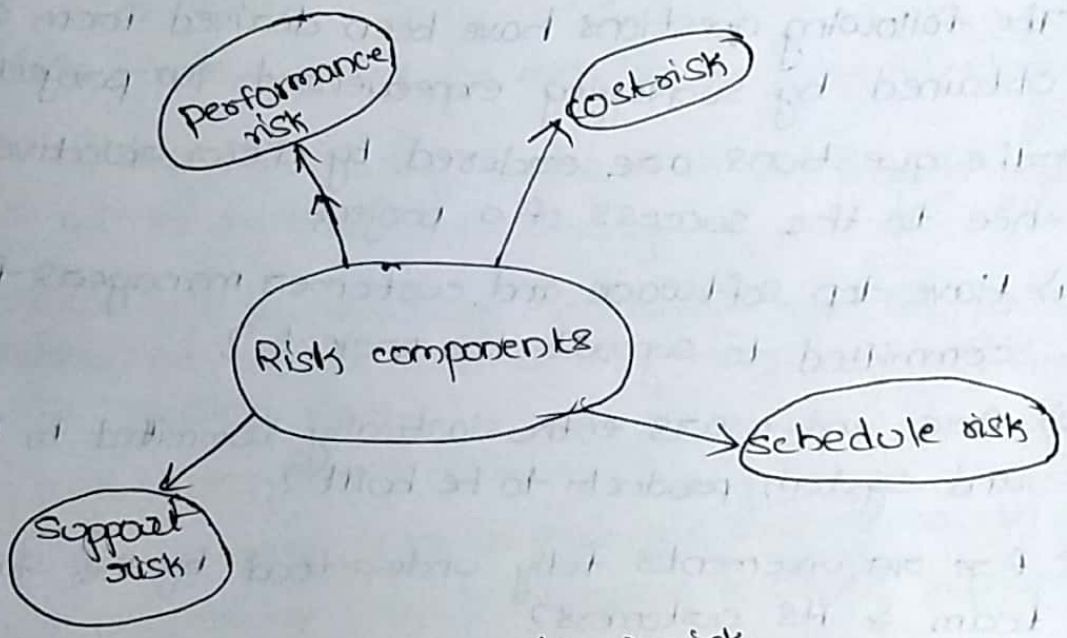


fig components of risk.

1) performance risk:- It is the degree of uncertainty that the product will satisfy the requirements.

2) cost risk:- It is the degree of uncertainty that the project will maintain the budget.

3) support risk: It is the degree of uncertainty that the resu project being developed will easy to correct, modify or adapt.

4) Schedule risk: It the degree of uncertainty that the s/w project will maintain the schedule and the project will be delivered in time.

→ The components and risk drivers that are used to analyze the impact of the risk.

→ The risk drivers are. 1) Negligible - 4
2) Marginal - 3
3) critical - 2
4) catastrophic - 1

Assessing overall risk project:-

→ The following questions have been derived from risk data obtained by surveying experienced s/w project managers.

→ The questions are ordered by their relative importance to the success of a project.

1) Have top software and customer managers formally committed to support the project?
2) Are end-users enthusiastically committed to the project and system/product to be built?
3) Are requirements fully understood by the s/w engineering team & its customers?
4) Have customer been involved fully in the definition of requirements?
5) Do end-users have realistics expectations?
6) Is the project scope stable?
7) Are project requirements stable?
8) Does the software engineering team have the right mix of skill?

⇒Risk projection :-

→ The risk projection is also called as risk estimation.

→ There are two ways by which risk can be rated
1) probability that the risk is real
2) consequences of problems associated with the risk.

→ The project planner, technical staff, project manager performs following steps for risk projection.
i) Enlist the consequences of the risk.
ii) Establish a scale that indicates the probability of risk being real.
iii) Estimate the impact of the risk on the project & product
iv) Note the overall accuracy of the risk projection so that there will be no misunderstandings.

→ These steps help to prioritize the risks. once the risks are prioritized then it becomes easy to allocate the resources for handling them.

→ Buiding risk table :-

1) Building the risk table is the simplest and most commonly used technique adopted by project managers in order to project the risk.

Risk table

Risk	category	probability	Impact	RMMM
Is the skilled people available	staff	60%	1	
Is that teamsize sufficient	staff	62%	2	
Have the staff received sufficient training	staff	30%	2	

will technology meet the Expectations	Technology	30%	2.
How much amount of reused s/w is required?	Project size	60%	3

→ While building the risk table the project team first of all enlists all probable risks with the help of risk item checklist.

→ Each risk is then categorized.

a) projectsize
b) Technology
c) customer
d) staff
e) Business
f) Developing environment

→ probability of occurences of each risk is then estimated by each team member individually.

→ The impact of each risks is assessed. While calculating the im-pact of each risk, each using the cost drives.

2) After building the this table then sorted by probability & impact. The high probability & high impact risks will be at the top of the table. And low probability & low impact risk will be at the bottom of the table. This arrangement of the tabled "first-order prioritization".

3) Then the project manager goes through this first-order prioritized risk table and draws horizontal line at some point in the table. This line is called cut-off line.

4) The risk table below the cut-off line is again sorted & a "second-order prioritization" is applied on this table.

5) The risk table above the cut-off line is having the risks with high probability & high impact.

6) All the risks that lie above the cut-off line should be managed.

⇒ Assessing risks impacts:-

While assessing the risks impact three factors are considered

* Nature of risk
* scope of risk
* Timing at which risk occurs

Nature of risks: It denotes the type or kind of risk. for eg if s/w requirement is poorly understood, the s/w processes gets poorly designed and ultimately it will create a problem in unit testing.

Scope of risk:- It describes the severity of the risk.

Timing of risk:- It determines at which phase of s/w development lifecycle the risk will occur and how long it will persist.

Risk Exposure:-

Risk Exposure = probability of occurence of risk * cost

Eg:- consider a s/w project with 77% of risk probability in which 15 components are developed from the scratch. Each component have on an avg 500 LOC & each LOC have an average cost of $10. Then the risk exposure can be calculated as

first calculate cost

cost = N.o. of components * LOC * cost of each LOC
= 15 * 500 * 10
= $75000

Risk Exposure = probability of occurence of risk * cost
= 77/100 * 75000
= $57750.

⇒ Risk Refinement :-

Risk refinement is a process of specifying the risk in more detail. The risk refinement can be represented using CTC format suggested by D.P. Gluch.

→ The CTC is condition-Transition-consequence. The condition is stated first and then based on this condition sub conditions can be derived.

→ Then determine the effects of these sub conditions in order to refine the risk.

→ This refinement helps in exposing the underlying risks.

→ This approach makes easier for the project manager to analyze the risk in greater detail.

⇒ Risk Mitigation, Monitoring and Management (RMMM) :-

→ There are three issues in strategy for handling the risk is

1) Risk avoidance
2) Risk Monitoring
3) Risk Management

1) Risk Mitigation :- Risk mitigation means preventing the risks to occur.

→ Following are the steps to be taken for mitigating the risks.

1) communicate the concerned people to find of probable risk.
2) Find out and eliminate all those causes that can create risk before the project starts.
3) Develop a policy in an organization which will help to continue the project even though some staff leaves the organization
4) Everybody in the project team should be acquainted with the current development activity.

5) Maintain the corresponding documents in timely manner, this documentation should be strictly as per the standards set by the organization.

6) conduct timely reviews in order to speed up the works.

7) For conducting every critical activity during s/w develop-ment; provide additional staff if required.

Risk Monitoring :-

→ The project manager must be monitored the following steps

1) The approach or behaviour of the team members as project pressure

2) The degree in which the team perform with the spirit of "team work"

3) The type of co-operation among the team members.

4) The types of problems that are occuring.

5) Availability of jobs within and outside the organization.

6) Conduct timely reviews in order to speed up the works.

7) For conducting every critical activity during software development, provide the additional staff if required.

⇒ Risk Mo

8) Availability of jobs within and outside the organization.

→ The objective of risk monitoring is.

1) To check whether the predicted risks really occur or not.

2) To ensure the steps defined to avoid risk are applied properly or not.

3) To gather the information which can be useful for analy-zing the risk.

⇒ Risk Management :-

project manager performs this tasks when risks become reality. If project manager is successful in applying the project

mitigation effectively then it becomes very much easy to manage risk.

Ex: consider a scenario that may people are leaving the organization then if sufficient additional staff is avai -lable, if current development activity is known to every -body in the team, if latest and systematic documenta -tion is available then any 'new comer' can easily under -stand current development activity. This will ultimately help in continuing the works without any interval.

⇒ RMMM plan:-

The RMMM plan is a document in which all the risk analysis activities are described. sometimes project managed includes this document as a part of overall project plan. sometimes specific RMMM plan is not created, howe -ver each risk can be described individually using risk information sheet.

→ Typical template for RMMM plan or Risk Informationsheet.

→ The risk information sheet can be maintained by data -base systems. After documenting the risk using ei -ther RMMM plan or Risk information sheet the risks mitigation, monitoring & analysis activities are stopped.

Risks Information sheet

Project Name <enter name of the project for which risks can be identified>

Risk id <#>	Date <date at which risk is identified>	probability <risk probability>	Impact <low/ medium/ high>
Origin <the person who has identified the risk>		Assigned to <who is responsible for mitigating the risk>	
Description <Description of risks identified>			
Refinement/ context <associated information for risk refinement>			
Mitigation/ Monitoring <enter the mitigation/ monitoring steps taken>			
Trigger/ contigency plan <if risk mitigation fails then the plan for handling the risk>			
Status <Running the status that provides a history of what is being done for the risk and changes in the risk. Include the date the status entry was made>			
Approval <name & signature of person approving closure>		closing date <date>	