UNIT-1

Introduction:

Software :-

It is a set of program containing instructions that provides desired functionality.

Engineering is the process of designing and building something that serves a particular purpose and finds a cost-effective solution to problems.

Software Engineering

→ Software engineering is the process of designing, developing, testing and maintaining software.

→ It is a systematic and disciplined approach to software developement that aims to create high-quality, reliable and maintainable software.

→ Software engineering includes a variety of techniques, tools and methodologies, including requirement analysis, design, testing and maintenance.

* Generally the software engineering to be built of two methods.

Arts and science methodology :

→ The software engg. to be used arts methodology because the developer 1st collects all the required ingrediant and information to which is previously developed.

→ The software engg. to be used science methodology because the developer to be used some new technique which is to be satisfied the user needs and also saves echonomical money for further reference.

→ As the software engg. to be develope very complex applications and that complex application to be executed by using one type of machine i.e. Turing machine. This turing machine was developed by Alan Turing. So that the alan Turing is the father of software engineering.

Evolution of software engineering

→ As the computer to be developed different types on the use of technology phase by phase, always the software engg. to be execute the software by using 4 methods.

→ (i) First the developer to be designing the abstract view of the particular software, which satisfied the user needs.

(ii) Then the developer must to be check the hardware and software product are available or not. Then also check all the hardware fiasibility satisfied or not.

(iii) The most objective of the hardware engg. to be check how the software to be devloped in a special time period and consumes less money for creating the s/w.

(iv) Lastly the ~~hardware~~ software engineer to be check the required software to be satisfied the user needs or not. That means at the time of impl. tion the software to be work properly or not.

So, the s/w engeneer to developed 5 tech. to create an ensure application.

(i) Explanatory s/w tech.
(ii) control structure
(iii) Data structure
(iv) Network oriented
(v) OOP

<u>Explanatory software technology</u>

-> It means in this case the developer can't create any specific systematic order the execution of program flow.

-> In this case all the computer to be execute very slow as compair to todays, because it used <u>vacuum tube</u> technology and all the source code to be rewritten in low level language i.e Assembly language.

-> Every program consist of 100 source code lines. So that the explanatory technology to be execute very small program and required very much time for rectifing error. So that it is very <u>sophisticated technology.</u>

(II) Control Structure :

→ The control structure tech. to be used for regulating all the disadvantages which occur in explanatory tech.

→ In this generation the control structure use Semiconductors technique for fast processing and gives the appropriate result.

→ In this case all software to be rewritten by using 100 source code line.

Defn It provides all the objective to be executed in a proper sequence by using a method i.e goto. The goto technique can't provide any security because it follows some explanatory methods so at that time a scientist Bohm's Jackni developed another three methods for security mechanism i.e selection (if, if-else, nested if-else), decision (switch), iteration looping (while, do-while, for, while-wend, do until, d-exit)

→ The control structure technique can't provides all the schematic representation of the particular software. That means the blue print of the particular software.

(III) Data Structure :-

→ It is the collection of all the logical information relivant information of a particular data, that data may be represented linear or non-linearly.

→ Generally the data structure to be developed by scientist when the Krichof project to invented

by Jackson.

→ The data structure technique first designed all the blue print of the particular software, which is the user needs and then the co-ordinating to be created.

→ The designing of the software to built of by using two mechanism.

(i) Entity relationship mechanism
(ii) Dataflow design (DfD)

→ The developed a technique which represent the schematic structure and the relationship between the data i.e named as E.R mechanism.

→ The data flow digram to be invented by J. henery and which specifies all the description of the data details in dynamically

→ But at a time DfD to be creating relationship between two data object.

Object oriented technique:

→ In object oriented techique to be used very large scale integration methods, which provides all authentication mechanism when the developer developing any client server architecture.

→ By using an object the developer provides security, data abstraction and encapsulation method, polymorc-phism, data binding, class and message passing to end user.

The object technique to be used online and offline application creation, but when it embeded with large application the WAN network can't be easily connected.

Aspect oriented technical / client server technique :

→ This technique used for authentication data flow in a sequential foremat for client to server.

→ This technique provides 2 methods for data flow server to client.

1. Post method () → uploading
2. get method () → accessing downloading

→ The JVM, always support the client server architecture for dynamic data flow.

Software laws :

S - law / static law
P - law / practical law
E - law / embeded law
Enlarge / joining

→ ~~1. S - law / static law~~ The scienties 'lehman' Provides 3 mechanism which is approve by international standard organization on year 1957, when the software evalued.

S - law / static law

→ The static law always check which s/w only provides the specification and solution at the time of execution.

ex :- Calculator

2. P- law / Pratical law :-

→ The law always checks, which s/w provides all aspects and details features about the s/w and dynamic solution at the time of execution.

ex :- All science pratical game.

E-law / Embeded law :-

→ This law always to be used when any realtime implification s/w to be developed, because this law always update the s/w the day by day activity.

→ on other worlds the embeded law always checks when the s/w developer to adding same new from existing s/w.

ex :- Social application s/w like facebook.

Software devlopement / s/w paradism :-

→ The paradism refers to creating a technique or initiation method fore program flow in a significant manner at the time of execution.

→ The software paradism always execute by using three perception technique.

(i) s/w design

(ii) s/w program

(iii) s/w Implementation

Software design :-

-> The s/w designing always stores the abstract view of the object flow digram in execution folder.

-> It also checkout which programming language to be embeded for creating s/w.

Software Programming :-

-> It refers to check all the programming code for detecting the error at the time of execution.

-> So it always check the program flow at the time of execution into 3 format.

(I) first it checks all the hardware and system software to satisfy all the program coding or not i.e called α - Testing.

(II) When the α - testing performed, then the developer checks this software gives appropriate objective or not i.e called β - testing.

(III) when the s/w execution to be completed then the end user check it to be satisfy all needs or not i.e called acceptance.

Software Implementation :-

-> The s/w implementation always gives how much time and space for required the program execution and also it provides the cost of the software.

Software engineering project :-

Project :- series of task that needed to completed in order to reach a specific outcome.

→ set of input & outputs requires to achieve a particular goal.

Software project :-

→ Programs are develop by individuals for there Personal use.

→ Therefore small in size & have limited functionality.

→ The author of program is usually the soul of a Program & himself maintains the user program.

→ Therefore usually lack of good user interface & Proper documentation.

Ex :- A Program developed by a student is part of his class assignments are programs and not software products. On the other hand software Products have multiple users, therefore have good user interface proper users manuals & good documentation support.

→ Not consist of programs but also associated of documentation.

Types of software devlopement project :-

→ A S/W devlopement company is tipically structured into a large no. of devlopement team.

→ Each team either devlop some S/W product or handles some out-sourced projects.

(i) S/w product devlopement project
(ii) out sourced project.

(i) S/w product devlopement project

→ A software product can either be generic or a custom develop product.

→ Generic products one used by a large & diverse range of customers.

ex/ Generic products are operating system such as microsoft windows, DBMS such as ORACLE, My SQL & banking s/w product such as financial.

→ A company developing a generic product, first determines what may be useful to a large cross-section of users.

→ On the other hand a customized s/w product is developed according to the specification drawn up by one or atmost a few customers.

(ii) Outsourced Project :-

→ Indian software Industries have excellent in executing outsourced s/w projects & have made a name for themselves all over the world.

→ The indian companies have slowly started to focus on product developement as well (which was the front they were lacking in till date).

→ The type of development worked being handled by a company can have an impact on its profitability.

ex

company that has developed a generic s/w product usually gets an interrupted stream of revinue for several years on the other hand outsourced project fetch (access) a one time revenue to the developing company.

Exploratory style of software developement :-

→ The type of development worked being handled by a company can have an impact on its profitability.

ex

- company that has developed a generic s/w product usually gets an interrupted stream of revinue for several years on the other hand outsourced project fetch (access) a one time revenue to the developing company.

Exploratory style of software developement :-

→ Exploratory program developement style refers to an informal developement style or builds and fix the style in which the programmers uses his own institution to develope a program rather than making user systematic body of knowldge which is catagorized under s/w engeneering.

→ The style of developement gives complete freedom to program to choose activities which they like to develop software.

→ This style doesn't offer any rules to start developing any s/w.

→ The following block digram will clears some fact relating to this model.

(Fig-1)

→ The block of initial briefing by the customer i.e brief introduction of the problem by the customer after the briefing, control goes to initial coding that is as soon as the developer or programmer known about the problem he starts coding to coding to develop a working program without considering any kind of requirement analysis the program will be tested "bug" found and there getting fixed by programers.

→ The cycle continous untill satisfactory code is not obtain after finding satisfactory code development gets completed.

⊛ So the s/w developement style is used for the development of small program.

→ In an exploratory developement scinario, the effort and time required to develop professional

s/w increases with an increase in program size.

(fig-2 Increase in development time and effort with problem size)

→ i) Thick line represent exploratory style used to develop a program are program size increases required effort and time increases. It would take too long and cost too much.

ii) Thin solid line is used to represent a case when development is carried out using s/w engg. principles It becomes possible to solve a problem with effort and time i.e almost linear in program size.

(iii) The dotted line is used to represent in case when development is carried out by an automated machine an increase in effort and time with size would be even closer to a linear increase with size.

Emergence of software Engineering :-

-> There are mainly 6 stages of emergence of s/w engineering

(i) Early computer programming
(ii) High level language programming
(iii) control flow based design
(iv) Data structure oriented design
(v) Data flow oriented design
(vi) Object oriented design

(i) Early computer programming

-> Programs were being written in assembly language programs were limited to about a few hundred lines of assembly code.

-> Every programmer develope his own style of writting programs.

(ii) High level language programming

-> computers become faster with the introduction of the semiconductor technology in the early 1960's

-> faster semiconductor transitors replaced the pre-valent vaccume tube based circuit in a computer.

-> It became possible to solve larger and complex problem

-> At this time high level language such as fortan, Cobol are introduced.

→ These reduced the effort required to develope software products and help programmers to write larger program.

→ Programmers were still using the exploratory style of s/w development typically programs were limited to size of around a few thousand of lines of source code.

(iii) Control flow based design :-

→ The size of complexity kept on increasing, the exploratory programming style proved to be insufficient.

→ Programmers found it increasingly difficult not only to write cost effective and correct programs.

→ But also to understand & maintain programs written by others to pay particular attention to the design of programs control flow structure.

→ A program control flow structure indicates the sequence in which the programs instruction are executed.

→ To help develope programs having good control flow structure the flow chating technique was developed. This technique is being used to represent and design algorithms.

(iv) Data structure oriented design :-

→ Data structure oriented design became more powerful with the integrated circuit in the early 1970's

→ These could now be used to solve more complex problems.

→ Software engineering were task to develope layer and more complecated s/w products which obtain required writting in excess of several tens of thousand of lines of source code.

→ The control flow based program developement technique couldn't be used satisfactorily any more to write those programs, and more effecitive program develope technique were needed.

→ While developing a program, it was much more important to pay attention to the design of the important data structure design technique based on this principle are called data structure oriented design technique.

for example :- Data structure oriented technique is design to the JSP (Jackson's structured programming). This technique developed by "michel Jaction" in 1975

→ JSP methodology a program data structure first design using notation for sequence, selection & iteration.

(V) Data flow oriented design :-

→ As computer became still faster and more powerful with the introduction of very large scale integrated circuits (VLSI) and some new architectural concept more complex and sophiscated s/w products were needed to solve further challenging problems.

→ Therefore s/w engg. looked out for more effective technology for designing software products as soon data flow oriented technique were purposed.

→ The function also called as processes and the data items that are exchanged between the different function are represented in digram known as data flow digram (DFD).

(vi) Object oriented design :-

→ Object oriented design technique such as employees, payrole register etc. Relevant to a problem are 1st identifyied then the relationship among the objects such as composition, reference and inheritance are determine.

→ Object - oriented technique have gain wide spread asceptance because of their simplicity, the slope of code and design reuse promise of lower developement, lower developement cost, more robust code and easier maintaince.

→ Each object essentailly act as a data hiding is also known as data abstraction.

Computer system engineering :-

→ Computer system engineers are employed in industries, govt., education and consulting and are concerned with the design, developement and operation hardware and software components of a computer system.

(fig-3)

Software Developement life cycle :- (SDLC)

→ The SDLC model generally to be used for graphically depicts, the different phases through which a software evolves.

→ It is usually represented by a textual description of the different activities that need to be carried out during each phase.

→ A good SDLC decides clearly define the different phases in life cycle, so that there is no redundancy, consistancy to be occure at the time of software creation.

→ Every software developer to creating the software, it follows several phases.

(fig-4 SDLC)

i) feasibility study :-

→ The main focus of the feasibility study is to be determine wheather it would be financially and technically feasible to develop the software that means

→ 1st the developer to check what the information to be collected and which language to be coding that to be supported on operating system or not.

→ Always the feasibility study provides some constraints condition at the time of s/w creation. If the s/w to be satisfied all the constraints then the s/w to be ready for creation.

(ii) Requirement analysis and specification :-

→ The aim of the requirement analysis and specification to understand the exact ingradients or raw documents for creating the s/w.

→ The phase consist of two distinct activities namely requirement gathering and analysis and there other one is requirement specification.

a) Requirement gathering and analysis :-

→ The goal of the requirement gathering activity is to collect all relivant information regarding the software as user needs. But it collects all over the information in random way with the help of different engg. analysis.

b) Requirement specification :

→ After the requirement gathering and analysis activities are complete, only the optimization information which is required for s/w creation and also the s/w specification checks all the optimization information which is not an ambiguous.

→ It is otherwise called as good s/w requirement specification (SRS).

(iii) Design

→ The goal of the designing phase is to transform the requirement specified in the SRS document into a structure which is very suitable for program creation and also to easily understandable the end user.

→ There are 2 methodology to be used for designing the SRS document.

→ one for procedural oriented programming (POP) language s/w creation and the another one is object oriented programming (OOP) language.

→ The technique DFD (data flow digram) to be used for traditional application, which is close from top to bottom i.e POP.

→ Another one technique i.e ER (entity relationship) to be used for multiple entity connection on the which is flow from top to bottom and bottom to up i.e OOP.

(iv) Coding :

→ After designing phase the engg. to be coding the designing by using any language, that language may be POP and OOP.

→ As the coding to be very understandable and easily to implement in programming system. It to be devided into 3 parts. i.e

(i) Unit coding
(ii) Integration coding
(iii) System coding.

Unit coding:-

→ 1st the iniciated designed the software in indivisual format.

→ After that it collects all the indivisual code and then united into single format i.e called ~~Integ~~ Integration coding.

→ finally at the end period the integration coding to verify the interface among different units are working satisfictory, then ~~it~~ alocated as system coding.

(V) Testing :

→ This SDLC phase is very important, because after creating all the programming code, then the initiator to be checkout this slw gives designed output or not as the user needs.

→ Generally the testing phase to be tested in 3 format.

(I) α - Testing

α - testing is the system testing performed by the development team.

(II) β - Testing

This is the system testing performed by friendly set of customers which is nearer the development team.

(iii) Acceptance :-

After the s/w has been delivered, the customer perform system testing to determine wheather to accept the delivered s/w or to reject it.

(Vi) Maintenance :-

→ The total effort spend on maintenance of typical s/w form starting to ending.

→ The maintenance to be perform in between system develop s/w goes to 3 categorised.

(i) Corrective :-

→ This type of maintenance is carried out to correct errors that were not discovered during the product development phase as the user needs.

(ii) Perfective maintenance :-

This type of maintenance is carried out to improve the performance of the system or to enhance the functionalities of the system based on customers request.

(iii) Adaptive maintenance :-

It is usually required for porting / transfer the S/w to work in a new environment.

(vii) Implementation :-

→ It is the last phase of SDLC, it to be checkout end user feed back after using the particular S/w.

SDLC model / S/w processing model :-

→ The S/w development models are the various process or methodology that are being selected for the development of the project depending on the project aims and goals.

→ There are many development life cycle models that have been developed in order archieve different required objective.

① waterfall model :- (1970)

→ It is the simplest model of S/w development paradism

→ It says the all the phases of SDLC will function one after another in linear manner. That means when the first phase is finished

accurately, then only the second phase will start and so on.

→ This model assumes that everything is carried out and taken place perfectly as planned in the previous stage. and there is no need to thing about the first issue that may arrised in the next phase.

→ This model doesn't work smoothly if there are some issues left at the previous step.

→ This model is best suited when developers already have designed and developed similar s/w in the past.

→ This type of s/w development model basically used for the project which is small and there are no uncertain requirement.

Advantages :-

→ This model is simple and easy to understand and use.

→ It is easy to manage due to rigidity of the model, each phase has specific objective provids and a review process.

→ At a time the s/w to be completely design and the phases do not overlap.

Dis advantages :-

→ once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.

-> High amount of risk and uncertainty

-> Generally this technique can't be used for new thought application and oops technique.

② Iterative waterfall model :-

-> An iterative life cycle model doesn't attemped to start with a full specification of requirements instead, developments begins by specifying and implementing just part of the s/w, which can then be reviewed inorderes to identify future requirements.

-> This process is then repeated producing a new version of the s/w for each cycle of the model.

ex -> suppose we created a fully product s/w which specify all electronics materials.

Suppose we create a calculator s/w which perform some mathmatical operation like add, sub, mul, div. so by using Iterative model mechanism, 1st the developer creating 1st function un above four operation. After completing the 1st operation starting from feasibility study to Implementation then again review these steps for 2nd operation and so on.

Advantages :-

→ In iterative model we can only create a highlevel design of the application like any game application or any business oriented s/w to be develop.

→ In iterative model we are building and improving the product step by step, because it provieds feed back option.

→ In iterative model the new s/w which is developed accurately & also releable.

→ Iterative model also used OOP applications & POP applications.

Disadvantages :-

→ It consumes more time for s/w development.

→ each phase of an iteration is rigid with no overlaps.

→ costly system architecture or design issues may arrise because not all requirement are gathered of front for the entire life cycle.

→ Generally this model to be used, when the developer collecting some relivant data from existing s/w and also add some new requirement. data for s/w creation.

→ No intermediate delivery
→ No changes (rigidity)

③ Prototyping model :-

→ It is the extension of the iterative waterfall model.
→ This model suggest building a working prototype of the system, before development of the actual s/w.
→ A prototype is a toy and crud implement of a system. It has limited functional capabilities, low reliability, low existance period or inefficient performance as compared to the actual s/w.
→ A prototype can be build very quickly by using several shortcut methods. The shortcut usually involve developing inefficient inaccurate or dummy function.

→ After creating the proto s/w, then the developer testing this s/w an provids for implementation to customer.

→ After customer prevolution of prototype s/w then the developer checks refine requirment incorporating customer suggestion Then it again design the prototype s/w until the customer objective ~~fulfil~~ fullfilment.

customer not clear with idea
Throw away model
good for technical & requirement risk
Increase in cost of development

Advantages :-

→ The prototyping models to be used for new s/w development.

→ By using this models, the developer developing any graphical user interface s/w very convienty.

→ The prototyping design models provides very usere lable programming code, so that every customer very easly to understand.

→ It provides user interface mechanism which is very optimized.

→ It provides always very efficient and accurate objective of the user requirement. so this s/w models is otherwise known as get it right model.

④ Incremental model :-

→ The incremental models to be derived from the technique of iterative waterfall model and prototyping models.

→ In incremental model, the requirement phase (1) breakendown the s/w into several models.

→ In this case each model first creating a dummy s/w containing some basic flow features, then deliver to the customer. if any revalution feedback to be provided customer, then the developer rebuilt the dummy s/w until the complete modul to be produce and all modules to be executed on similar way.

→ When all the modules to be initiated then all are to combining into single one by sharing is feedback.

ex:-

→ The incremental model always split into several modules after the collecting requirement data gathering & specification.

→ Starting with the module-1 in each suless increment, the next module is constructed using an iterative waterfall model of development and implemented at the customer side.

Advantages :-

-> It provides new features on new module to be added for existing s/w.

-> It also reduce the error objective form of the final product and leading to greater reliability of the s/w.

-> This models also to be used for large organisation manipulating s/w.

Disadvantages :-

-> It can't used the pre-existing s/w method or code information.

⑤ Evolutionary Model :- / V-model :-

→ The main objective of this model by its conveyed name.

→ In this case all requirements, plan, Estimates and solution evolve over the iterations, ratherthan fully defined and frozen in a measure up-front specification effort before the development iteration begin.

→ first the developer collect all rough specification data for developing the s/w. then the developer analysis it and identify the core part which is essential for s/w creation and separated other non-core part.

→ Now by using core-data, the developer devided the main objective into different section. by using incremental model method.

→ After completing incremental model the developer implements fully complete s/w by using iterative waterfal technique.

Advantages :-

→ The main advantage of this model is well suited to use in object oriented s/w development project.

⑥ Spiral model :- / Bohem's spiral model.

→ The spiral model is similar to the incremental model, with more emphasis placed on risk analysis.

→ The spiral model has four phases : Planning, risk analysis, Engineering and evaluation.

→ A software project repeatedly passes through these phases in iterations.

→ The four phases for risk analysis are given below.

(i) Planning phase :-

→ Requirements are gathered during the planning phase.

→ Requirements like BRS (Business requirement specification and SRS (System requirement specification) that all risk are analysis on this phase.

(ii) Risk analysis :-

→ AT the time of user implementation a process is undertaken to identify risk and alternet solution. So a prototype is produced at the end of the risk analysis phase.

(3) Engineering phase :-

→ In this phase software is developed, along with testing at the end of the phase. If any corrupted occur in between s/w and hardware this risk to be calculated in this phase.

(4) Evalution phase :-

→ This phase allows the customer to Evalute the output of the project and also the developer calculate to its investment structures.

fig :- spiral model.

Advantages :-

→ High amount of risk analysis hence, avoidance of risk is enhanced.

→ Good for large and mission critical project

→ Strong approval and documentation control because this model to be standardies by ISO

⑦ Algile s/w :-

→ It is a time boned iterative approach s/w delivery that builds s/w incrementally from the start of the program instead of fully creation of the s/w.

→ It works by breaking project down into little bit of user functionalities called user stories, Prioritizing, then contineously delevering them into a shorted time period. Then time period may be Algile s/w 1 weak or 7 weak.

Agile s/w development life cycle model :-

→ Agile SDLC model is the combination of iterative and incremental process model with focus. on process adaptability and customer satisfication by rapidly data flow on end user site.

Step-1

→ Before creating any s/w by using some technique which is performed all the task in a shorter time interval like URL (uniform resource location)

Step-2

→ So the agile SDLC models provides different stages for creating a satisperform s/w for end user.

(i) **Planning :-**

→ first the programmer to be checkout all the organisation and enduser background information.

→ After collecting all the information to be designing into different phases like technical phases, professional phases.

(ii) **Requirement analysis :-**

→ The objective of this phase to collect all the relivant ingradiant information to be accepted by team member scientist and other have neglated.

(iii) **Design :-**

→ To creating a blueprint of the particular s/w, first the developer collecting all the user need information and then check it which information are ingradiant to be required first. After that it creating the priority of particular ingradiant to be design by using dataflow methods.

(iv) **Coding :-**

→ Now to implement all the priority ingradiant design digram into the format of coding by using any language like C, C++ etc.

⑤ Unit testing :-

→ After compliting the code then the developer to implement and check it gives the desire output or not.

(vi) Acceptance testing :-

→ After unit testing the indivisual s/w to be delivered to end user.

⑧ RAD (Rapid application development) model :

→ RAD is a linear sequential software development process model that emphasizes a concise development cycle using an element based construction approach

→ If the requirement are well understood and described ~~, and well understood and described,~~ and the project scope is a constraint.

→ The RAD process enables a development team to create a fully functional system within a concise time period.

→ RAD is a concept that products can be developed faster and of higher quality through :

* Gathering requirement using workshop or focus groups
* Prototyping and early, reiterative user testing or designs.

* The reuse of software components.
* A rigidly paced schedule that refers design improvements to the next product version.
* Less formality in reviews and other team communication.

The various Phases of RAD

1. Business modelling :-

The information flow among business functions is defined by answering questions like what data drives the business process, what data is generated, who generates it, where does the information go, who process it and so on.

2. Data modeling :

The data collected from business modelling is refined into a set of data objects (entities) that are needed to support the business. The attributes (character of each entity) are identified, and the relation betn these data objects is defined.

3. Process modeling :

→ The information object defined in the data modelling phase are transformed to achieve the data flow necessary to implement a business function.

→ Process descriptions are created for adding, modifing, deleting or retriving a data object.

4. Application Generation :

Automated tools are used to facilitate construction of the software, even they use the 4th GL technique.

5. Testing & Turnover :

Many of the programming components have already been tested since RAD emphasis reuse. This reduced the overall testing time. But the new part must be tested and all interfaces must be tested and all interfaces must be fully exercised.

→ When the system should need to create the project that modularizes in a short span time (2-3 months).

→ When the requirements are well-known.

→ When the technical risk is limited.

→ When there's a necessity to make a system, which modularized in 2-3 monts of period.

→ It should be used only if the budget allows the use of automatic code generating tools.

Advantages of RAD model

→ This model is flexible for change.

→ In this model, changes are adoptable.

→ Each phase in RAD brings highest priority functionality to the customer.

→ It reduced development time.

→ It increases the reusability of features.

Disadvantage of RAD model :-

→ It required highly skilled designers.

→ All application is not compatible with RAD.

→ for smaller projects, we cannot use the RAD model.

→ On the high technical risk, it's not suitable.

→ Required user involvement.

Verification	Validation
-> we check whether we are developing the right product or not.	-> we check whether the developed product is right.
-> verification is also know as static testing.	-> validation is also known as dynamic testing.
-> It is done by the developer.	-> It is done by the tester.
-> verification includes different methods like inspections, reviews and walkthroughs.	-> validation include testing like functional testing, system testing, integration, and user acceptance testing.
-> verification is done before the validation.	-> After verification validation testing takes place

Responsibilities of a software Project Manager

A software project manager is the most important person inside a team who takes the overall responsibilities to manage the software projects and plays an important role in the successful completion of the projects.

A project manager has to face many difficult situations to accomplish these works. In fact, the job responsibilities of a project manager range from invisible activities like building up team morale to highly visible customer presentations.

Most of the managers take responsibility for writing the project proposal, project cost estimation, scheduling, project staffing, software process tailoring, project monitoring and control, software configuration management, risk management, managerial report writing and presentation and interfacing with clients. The task of a project manager are classified into two major types:

1. Project planning
2. Project monitoring and control

1. **Project Estimation:** Project Size Estimation is the most important parameter based on which all other estimations like cost, duration and effort are made.
   - **Cost Estimation:** Total expenses to develop the software product is estimated.
   - **Time Estimation:** The total time required to complete the project.
   - **Effort Estimation:** The effort needed to complete the project is estimated.
2. **Scheduling:** After the completion of the estimation of all the project parameters, scheduling for manpower and other resources is done.
3. **Staffing:** Team structure and staffing plans are made.
4. **Risk Management:** The project manager should identify the unanticipated risks that may occur during project development risk, analyze the damage that might cause these risks, and take a risk reduction plan to cope with these risks.
5. **Miscellaneous plans:** This includes making several other plans such as quality assurance plans, configuration management plans, etc.
   - **Lead the team:** The project manager must be a good leader who makes a team of different members of various skills and can complete their individual tasks.
   - **Motivate the team-member:** One of the key roles of a software project manager is to encourage team members to work properly for the successful completion of the project.

- **Tracking the progress:** The project manager should keep an eye on the progress of the project. A project manager must track whether the project is going as per plan or not. If any problem arises, then take the necessary action to solve the problem. Moreover, check whether the product is developed by maintaining correct coding standards or not.
- **Liaison:** The project manager is the link between the development team and the customer. Project manager analysis the customer requirements and convey it to the development team and keep telling the progress of the project to the customer. Moreover, the project manager checks whether the project is fulfilling the customer's requirements or not.
- **Monitoring and reviewing:** Project monitoring is a continuous process that lasts the whole time a product is being developed, during which the project manager compares actual progress and cost reports with anticipated reports as soon as possible. While most firms have a formal system in place to track progress, qualified project managers may still gain a good understanding of the project's development by simply talking with participants.
- **Documenting project report:** The project manager prepares the documentation of the project for future purposes. The reports contain detailed features of the product and various techniques. These reports help to maintain and enhance the quality of the project in the future.
- **Reporting:** Reporting project status to the customer and his or her organization is the responsibility of the project manager. Additionally, they could be required to prepare brief, well-organized pieces that summarise key details from in-depth studies.

- Knowledge of project estimation techniques
- Good decision-making abilities at the right time
- Previous experience managing a similar types of projects
- Good communication skills to meet the customer satisfaction
- A project manager must encourage all the team members to successfully develop the product
- He must know the various type of risks that may occur and the solution to these problems

Project Planning

Once a project is found to be possible, computer code project managers undertake project designing. Project designing is undertaken and completed even before any development activity starts. Project designing consists of subsequent essential activities:

Estimating the subsequent attributes of the project:

- **Project size:**
  What's going to be downside quality in terms of the trouble and time needed to develop the product?

- **Cost:**
  What proportion is it reaching to value to develop the project?
- **Duration:**
  However long is it reaching to want complete development?
- **Effort:**
  What proportion effort would be required?

The effectiveness of the following designing activities relies on the accuracy of those estimations.

- planning force and alternative resources
- workers organization and staffing plans
- Risk identification, analysis, and abatement designing
- Miscellaneous arranges like quality assurance plan, configuration, management arrange, etc.

Precedence ordering among project planning activities:
The different project connected estimates done by a project manager have already been mentioned. The below diagram shows the order during which vital project coming up with activities is also undertaken. It may be simply discovered that size estimation is that the 1st activity. It's conjointly the foremost basic parameter supported that all alternative coming up with activities square measure dispensed, alternative estimations like the estimation of effort, cost, resource, and project length also are vital elements of the project coming up with.

Precedence ordering among planning activities

Project Size Estimation

project size estimation is the most fundamental parameter. If this is estimated accurately then all other parameters like effort, duration, cost, etc can be determined easily.

At present two *techniques* that are used to estimate project size are:

1. **Lines of code or LOC**
2. **Function point**

Lines of code

Lines of code or **LOC** is the most popular and used metrics to estimate size.

LOC determination is simple as well. LOC measures the project size in terms of number of lines of statements or instructions written in the source code.

In LOC count, comments and headers are ignored.

Shortcomings of LOC

- Estimating LOC by analyzing the problem specification is difficult. Estimation of accurate LOC is only possible once the complete code has been developed. As project planning needs to be done before the development work begins so this metrics is of little use for project managers.
- Two different source files having same number of lines may not require same effort. The file having complex logic would require more effort than one with simple logic. Based on LOC proper estimation may not be possible.
- LOC is the numerical measurement of problem size. This metrics will vary to a large extent from programmer to programmer. An experienced professional may write same logic in less number of lines than a novice programmer

Advantages:

- Universally accepted and is used in many models like COCOMO.
- Estimation is closer to the developer's perspective.
- Both people throughout the world utilize and accept it.
- At project completion, LOC is easily quantified.
- It has a specific connection to the result.
- Simple to use.

Disadvantages:

- Different programming languages contain a different number of lines.
- No proper industry standard exists for this technique.
- It is difficult to estimate the size using this technique in the early stages of the project.
- When platforms and languages are different, LOC cannot be used to normalize.

.

---

2. Function Point Analysis: In this method, the number and type of functions supported by the software are utilized to find FPC(function point count). The steps in function point analysis are:

- Count the number of functions of each proposed type.
- Compute the Unadjusted Function Points(UFP).
- Find the Total Degree of Influence(TDI).
- Compute Value Adjustment Factor(VAF).
- Find the Function Point Count(FPC).

The explanation of the above points is given below:

- **Count the number of functions of each proposed type:** Find the number of functions belonging to the following types:
    - External Inputs: Functions related to data entering the system.
    - External outputs: Functions related to data exiting the system.

    - External Inquiries: They lead to data retrieval from the system but don't change the system.
    - Internal Files: Logical files maintained within the system. Log files are not included here.
    - External interface Files: These are logical files for other applications which are used by our system.
- **Compute the Unadjusted Function Points(UFP):** Categorise each of the five function types like simple, average, or complex based on their complexity. Multiply the count of each function type with its weighting factor and find the weighted sum. The weighting factors for each type based on their complexity are as follows:

Function type	Simple	Average	Complex
External Inputs	3	4	6
External Output	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10

- **Find the Function Point Count:** Use the following formula to calculate FPC

```
FPC = UFP * VAF
```

Advantages:
- It can be easily used in the early stages of project planning.
- It is independent of the programming language.
- It can be used to compare different projects even if they use different technologies(database, language, etc).

Disadvantages:
- It is not good for real-time systems and embedded systems.
- Many cost estimation models like COCOMO use LOC and hence FPC must be converted to LOC.

Project Estimation Techniques

Project size estimation is a crucial aspect of software engineering, as it helps in planning and allocating resources for the project. Here are some of the popular project size estimation techniques used in software engineering:

Software Project parameters can be estimated using:

1. Empirical estimation techniques
2. Heuristic estimation techniques
3. Analytical estimation techniques

Empirical estimation techniques

Empirical estimation techniques is done on the basis of educated guess of the project parameters.

This technique becomes more helpful of the estimator has already had experience doing similar project in past.

Therefore making suitable educated guess, having prior experience and common sense is good element for this technique.

Popularly used *empircal project estimation techniques are*:

1. **Expert judgement technique**
   Expert judgment is one of the most widely used estimation techniques. In this approach, an expert makes an educated guess of the problem size after analyzing the problem thoroughly. Usually, the expert estimates the cost of the different components (i.e. modules or subsystems) of the system and then combines them to arrive at the overall estimate. However, this technique is subject to human errors and individual bias. Also, it is possible that the expert may overlook some factors inadvertently. Further, an expert making an estimate may not have experience and knowledge of all aspects of a project. For example, he may be conversant with the database and user interface parts but may not be very knowledgeable about the computer communication part. A more refined form of expert judgment is the estimation made by group of experts. Estimation by a group of experts minimizes factors such as individual oversight, lack of familiarity with a particular aspect of a project, personal bias, and the desire to win contract through overly optimistic estimates.

   2.Delphi estimation techniques
   Delphi cost estimation approach tries to overcome some of the shortcomings of the expert judgment approach. Delphi estimation is carried out by a team comprising of a group of experts and a coordinator. In this approach, the coordinator provides each estimator with a copy of the software requirements specification (SRS) document and a form for recording his cost estimate. Estimators complete their individual estimates anonymously and submit to the coordinator. In their estimates, the estimators mention any unusual characteristic of the product which has influenced his estimation. The coordinator prepares and distributes the summary of the responses of all the estimators, and includes any unusual rationale noted by any of the estimators. Based on this summary, the estimators re-estimate. This process is iterated for several rounds. However, no discussion among the estimators is allowed during the entire estimation process. The idea behind this is that if any discussion is allowed among the estimators, then many estimators may easily get influenced by the rationale of an estimator who may be more experienced or senior. After the completion of several iterations of estimations, the coordinator takes the responsibility of compiling the results and preparing the final estimate.

Heuristic techniques

Heuristic estimation techniques works on assumption that the relationships among various project parameters can be represented in the form suitable mathematical expressions.

Once the basic parameters are known, the other parameters can be determined by substituting value of basic parameter in mathematical expression.

It can be widely divided into two classes

- Single variable model
- Multivariable model

Popularly used *heuristic estimation techniques are*:

1. COCOMO

Analytical estimation techniques

Analytical estimation techniques derive the required results starting with certain basic assumptions regarding the project.

Popularly used *analytical estimation techniques are*:

1. Halstead software science

Next →← Prev

COCOMO Model

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

The necessary steps in this model are:

1. Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).
2. Determine a set of 15 multiplying factors from various attributes of the project.
3. Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort E_i in person-months the equation used is of the type is shown below

$$E_i=a*(KDLOC)b$$

PauseNext

Unmute

Current TimeÂ 0:25

/

DurationÂ 18:10

Loaded: 7.70%

Â

Fullscreen

The value of the constant a and b are depends on the project type.

In COCOMO, projects are categorized into three types:

1. Organic
2. Semidetached
3. Embedded

1.Organic: A development project can be treated of the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects. **Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.**

2. Semidetached: A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. **Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.**

3. Embedded: A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. **For Example:** ATM, Air Traffic control.

For three product categories, Bohem provides a different set of expression to predict effort (in a unit of person month)and development time from the size of estimation in KLOC(Kilo Line of code) efforts estimation takes into account the productivity loss due to holidays, weekly off, coffee breaks, etc.

According to Boehm, software cost estimation should be done through three stages:

1. Basic Model
2. Intermediate Model
3. Detailed Model

1. Basic COCOMO Model: The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

Effort=a_1*(KLOC) a_2 PM

Tdev=b_1*(efforts)b_2 Months

Where

KLOC is the estimated size of the software product indicate in Kilo Lines of Code,

a_1,a_2,b_1,b_2 are constants for each group of software products,

Tdev is the estimated time to develop the software, expressed in months,

Effort is the total effort required to develop the software product, expressed in **person months (PMs)**.

Estimation of development effort

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic: Effort = 2.4(KLOC) 1.05 PM

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

Embedded: Effort = 3.6(KLOC) 1.20 PM

Estimation of development time

For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic: Tdev = 2.5(Effort) 0.38 Months

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

Embedded: Tdev = 2.5(Effort) 0.32 Months

Some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig shows a plot of estimated effort versus product size. From fig, we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.

Effort versus product size

The development time versus the product size in KLOC is plotted in fig. From fig it can be observed that the development time is a sub linear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. Further, from fig, it can be observed that the development time is roughly the same for all three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type.

Development time versus size

From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.

It is important to note that the effort and the duration estimations obtained using the COCOMO model are called a nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

Example1: Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

Solution: The basic COCOMO equation takes the form:

$Effort = a_1 * (KLOC)^{a_2}$ PM

$Tdev = b_1 * (efforts)^{b_2}$ Months

Estimated Size of project= 400 KLOC

(i)Organic Mode

$E = 2.4 * (400)^{1.05} = 1295.31$ PM

$D = 2.5 * (1295.31)^{0.38} = 38.07$ PM

(ii)Semidetached Mode

$E = 3.0 * (400)^{1.12} = 2462.79$ PM

$D = 2.5 * (2462.79)^{0.35} = 38.45$ PM

(iii) Embedded Mode

$E = 3.6 * (400)^{1.20} = 4772.81$ PM

$D = 2.5 * (4772.8)^{0.32} = 38$ PM

2. Intermediate Model: The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

Classification of Cost Drivers and their attributes:

(i) Product attributes -

- Required software reliability extent
- Size of the application database
- The complexity of the product

Hardware attributes -

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

Personnel attributes -

- Analyst capability

- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

Project attributes -

- Use of software tools
- Application of software engineering methods
- Required development schedule

Intermediate COCOMO equation:

$$E=a_i\ (KLOC)\quad b_i*EAF$$

$$D=c_i\ (E)d_i$$

3. Detailed COCOMO Model: Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost driver?s effect on each method of the software engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed cocomo, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System structure
3. Complete structure
4. Module code and test
5. Integration and test
6. Cost Constructive model

The effort is determined as a function of program estimate, and a set of cost drivers are given according to every phase of the software lifecycle.

Halstead's Software Metrics

Halstead's Software metrics are a set of measures proposed by Maurice Halstead to evaluate the complexity of a software program. These metrics are based on the number of distinct operators and operands in the program and are used to estimate the effort required to develop and maintain the program.

Field of Halstead Metrics

- **Program length (N):** This is the total number of operator and operand occurrences in the program.
- **Vocabulary size (n):** This is the total number of distinct operators and operands in the program.
- **Program volume (V):** This is the product of program length (N) and the logarithm of vocabulary size (n), i.e., V = N*log2(n).
- **Program level (L):** This is the ratio of the number of operator occurrences to the number of operand occurrences in the program, i.e., L = n1/n2, where n1 is the number of operator occurrences and n2 is the number of operand occurrences.
- **Program difficulty (D):** This is the ratio of the number of unique operators to the total number of operators in the program, i.e., D = (n1/2) * (N2/n2).
- **Program effort (E):** This is the product of program volume (V) and program difficulty (D), i.e., E = V*D.
- **Time to implement (T):** This is the estimated time required to implement the program, based on the program effort (E) and a constant value that depends on the programming language and development environment.

Halstead's software metrics can be used to estimate the size, complexity, and effort required to develop and maintain a software program. However, they have some limitations, such as the assumption that all operators and operands are equally important, and the assumption that the same set of metrics can be used for different programming languages and development environments. Overall, Halstead's software metrics can be a useful tool for software developers and project managers to estimate the effort required to develop and maintain software programs.

```
n1 = Number of distinct operators.
n2 = Number of distinct operands.
N1 = Total number of occurrences of operators.
N2 = Total number of occurrences of operands.
```

Halstead Metrics

Halstead metrics are:

- **Halstead Program Length:** The total number of operator occurrences and the total number of operand occurrences.
  N = N1 + N2
  And estimated program length is, $N^{\wedge} = n1\log_2 n1 + n2\log_2 n2$
  The following alternate expressions have been published to estimate program length:
  - $N_J = \log_2(n1!) + \log_2(n2!)$
  - $N_B = n1 * \log_2 n2 + n2 * \log_2 n1$
  - $N_C = n1 * sqrt(n1) + n2 * sqrt(n2)$

    - $N_S = (n * \log_2 n) / 2$
- **Halstead Vocabulary:** The total number of unique operators and unique operand occurrences.
  n = n1 + n2
- **Program Volume:** Proportional to program size, represents the size, in bits, of space necessary for storing the program. This parameter is dependent on specific algorithm implementation. The properties V, N, and the number of lines in the code are shown to be linearly connected and equally valid for measuring relative program size.
  $V = Size * (\log_2 vocabulary) = N * \log_2(n)$
  The unit of measurement of volume is the common unit for size “bits”. It is the actual size of a program if a uniform binary encoding for the vocabulary is used. And error = Volume / 3000
- **Potential Minimum Volume:** The potential minimum volume V* is defined as the volume of the most succinct program in which a problem can be coded.
  $V^* = (2 + n2^*) * \log_2(2 + n2^*)$
  Here, n2* is the count of unique input and output parameters
- **Program Level:** To rank the programming languages, the level of abstraction provided by the programming language, Program Level (L) is considered. The higher the level of a language, the less effort it takes to develop a program using that language.
  L = V* / V
  The value of L ranges between zero and one, with L=1 representing a program written at the highest possible level (i.e., with minimum size).
  And estimated program level is $L^\wedge = 2 * (n2) / (n1)(N2)$
- **Program Difficulty:** This parameter shows how difficult to handle the program is.
  D = (n1 / 2) * (N2 / n2)
  D = 1 / L
  As the volume of the implementation of a program increases, the program level decreases and the difficulty increases. Thus, programming practices such as redundant usage of operands, or the failure to use higher-level control constructs will tend to increase the volume as well as the difficulty.
- **Programming Effort:** Measures the amount of mental activity needed to translate the existing algorithm into implementation in the specified program language.
  E = V / L = D * V = Difficulty * Volume
- **Language Level:** Shows the algorithm implementation program language level. The same algorithm demands additional effort if it is written in a low-level program language. For example, it is easier to program in Pascal than in Assembler.
  L’ = V / D / D
  lambda = $L * V^* = L_2 * V$
- **Intelligence Content:** Determines the amount of intelligence presented (stated) in the program This parameter provides a measurement of

program complexity, independently of the programming language in which it was implemented.
I = V / D

- **Programming Time:** Shows time (in minutes) needed to translate the existing algorithm into implementation in the specified program language.
T = E / (f * S)
The concept of the processing rate of the human brain, developed by psychologist John Stroud, is also used. Stoud defined a moment as the time required by the human brain requires to carry out the most elementary decision. The Stoud number S is therefore Stoud's moments per second with:
5 <= S <= 20. Halstead uses 18. The value of S has been empirically developed from psychological reasoning, and its recommended value for programming applications is 18.
Stroud number S = 18 moments / second
seconds-to-minutes factor f = 60

Counting Rules for C Language

1. Comments are not considered.
2. The identifier and function declarations are not considered
3. All the variables and constants are considered operands.
4. Global variables used in different modules of the same program are counted as multiple occurrences of the same variable.
5. Local variables with the same name in different functions are counted as unique operands.
6. Functions calls are considered operators.
7. All looping statements e.g., do {…} while ( ), while ( ) {…}, for ( ) {…}, all control statements e.g., if ( ) {…}, if ( ) {…} else {…}, etc. are considered as operators.
8. In control construct switch ( ) {case:…}, switch as well as all the case statements are considered as operators.
9. The reserve words like return, default, continue, break, size, etc., are considered operators.
10. All the brackets, commas, and terminators are considered operators.
11. GOTO is counted as an operator and the label is counted as an operand.
12. The unary and binary occurrences of "+" and "-" are dealt with separately. Similarly "*" (multiplication operator) is dealt with separately.
13. In the array variables such as "array-name [index]" "array-name" and "index" are considered as operands and [ ] is considered as operator.
14. In the structure variables such as "struct-name, member-name" or "struct-name -> member-name", struct-name, and member-name are taken as operands, and '.', '->' are taken as operators. Some names of member elements in different structure variables are counted as unique operands.
15. All the hash directives are ignored.

Example – List out the operators and operands and also calculate the values of software science measures like

```
int sort (int x[ ], int n)

{
    int i, j, save, im1;
    /*This function sorts array x in ascending order */
    If (n< 2) return 1;
    for (i=2; i< =n; i++)
    {
        im1=i-1;
        for (j=1; j< =im1; j++)
            if (x[i] < x[j])
            {
                Save = x[i];
                x[i] = x[j];
                x[j] = save;
            }
    }
    return 0;
}
```

Explanation

Operators	Occurrences	Operands	Occurrences
int	4	sort	1
()	5	x	7
,	4	n	3
[]	7	i	8
if	2	j	7
<	2	save	3
;	11	im1	3
for	2	2	2
=	6	1	3

Operators	Occurrences	Operands	Occurrences
–	1	0	1
<=	2	–	–
++	2	–	–
return	2	–	–
{}	3	–	–
n1=14	N1=53	n2=10	N2=38

```
Therefore,
N = 91
n = 24
V = 417.23 bits
N^ = 86.51
n2* = 3 (x:array holding integer
to be sorted. This is used both
as input and output)
V* = 11.6
L = 0.027
D = 37.03
L^ = 0.038
T = 610 seconds
```

Advantages of Halstead Metrics

- It is simple to calculate.
- It measures the overall quality of the programs.
- It predicts the rate of error.
- It predicts maintenance effort.
- It does not require a full analysis of the programming structure.
- It is useful in scheduling and reporting projects.
- It can be used for any programming language.
- Easy to use: The metrics are simple and easy to understand and can be calculated quickly using automated tools.
- Quantitative measure: The metrics provide a quantitative measure of the complexity and effort required to develop and maintain a software program, which can be useful for project planning and estimation.
- Language independent: The metrics can be used for different programming languages and development environments.
- Standardization: The metrics provide a standardized way to compare and evaluate different software programs.

Disadvantages of Halstead Metrics

- It depends on the complete code.
- It has no use as a predictive estimating model.
- Limited scope: The metrics focus only on the complexity and effort required to develop and maintain a software program, and do not take into account other important factors such as reliability, maintainability, and usability.
- Limited applicability: The metrics may not be applicable to all types of software programs, such as those with a high degree of interactivity or real-time requirements.
- Limited accuracy: The metrics are based on a number of assumptions and simplifications, which may limit their accuracy in certain situations.

Staffing level estimation

Once the effort required to develop a software has been determined, it is necessary to determine the staffing requirement for the project. Putnam first studied the problem of what should be a proper staffing pattern for software projects. He extended the work of Norden who had earlier investigated the staffing pattern of research and development (R&D) type of projects. In order to appreciate the staffing pattern of software projects, Norden's and Putnam's results must beunderstood.

Norden's Work

Norden studied the staffing patterns of several R & D projects. He found that the staffing pattern can be approximated by the Rayleigh distribution curve. Norden represented the Rayleigh curve by the following equation: $E = K/t^2d * t * e-t^2 / 2 t^2 d$ Where E is the effort required at time t. E is an indication of the number of engineers (or the staffing level) at any particular time during the duration of the project, K is the area under the curve, and td is the time at which the curve attains its maximum value. It must be remembered that the results of Norden are applicable to general R & D projects and were not meant to model the staffing pattern of software development projects.

The Rayleigh manpower loading Curve

Putnam's Work

Putnam studied the problem of staffing of software projects and found that the software development has characteristics very similar to other R & D projects studied by Norden and that the Rayleigh-Norden curve can be used to relate the number of delivered lines of code to the effort and the time required to develop the project. By analyzing a large number of army projects,

Putnam derived the following expression: L = Ck K 1/3td 4/3

The various terms of this expression are as follows:

• K is the total effort expended (in PM) in the product development and L is the product size in KLOC.

• td corresponds to the time of system and integration testing. Therefore, td can be approximately considered as the time required to develop the software.

• Ck is the state of technology constant and reflects constraints that impede the progress of the programmer. Typical values of Ck = 2 for poor development environment (no methodology, poor documentation, and review, etc.), Ck = 8 for good software

development environment (software engineering principles are adhered to), Ck = 11 for an excellent environment (in addition to following software engineering principles, automated tools and techniques are used). The exact value of Ck for a specific project can be computed from the historical data of the organization developing it. Putnam suggested that optimal staff build-up on a project should follow the Rayleigh curve. Only a small number of engineers are needed at the beginning of a project to carry out planning and specification tasks. As the project progresses and more detailed work is required, the number of engineers reaches a peak. After implementation and unit testing, the number of project staff falls. However, the staff build-up should not be carried out in large installments. The team size should either be increased or decreased slowly whenever required to match the Rayleigh-Norden curve. Experience shows that a very rapid build up of project staff any time during the project development correlates with schedule slippage. It should be clear that a constant level of manpower through out the project duration would lead to wastage of effort and increase the time and effort required to develop the product. If a constant number of engineers are used over all the phases of a project, some phases would be overstaffed and the other phases would be understaffed causing inefficient use of manpower, leading to schedule slippage and increase in cost. Effect of schedule change on cost By analyzing a large number of army projects, Putnam derived the following expression: L = CkK 1/3td 4/3 Where, K is the total effort expended (in PM) in the product development and L is the product size in KLOC, td corresponds to the time of system and integration testing and Ck is the state of technology constant and reflects constraints that impede the progress of the programmer Now by using the above expression it is obtained that, K = L 3 /C k 3 td 4

software project scheduling?

Software Project Scheduling is a significant project planning step.

It becomes so important as this decide the time-frame for each task that would be undertaken. It judges the whole plan and points down what would possibly be needed at which time of software project development.

As part of *project scheduling activities in software engineering*, the *project manager* does following:

1. Points out the various tasks that would be done during project.
2. Divide the large works into smaller activities or module.
3. Find out the dependency among various activities of the software project.
4. Determine the probable time that would be needed to complete each tasks.
5. Allocate the resources to each activities.
6. Plan when to start and when to end each activities.
7. Determine the critical path of the project.

A critical path is the chain of important activities that would help in determining the duration of the project.

So from the above, we can see that determining necessary activities to complete the project is the first step. If the project manager has good experience then this step would be effectively completed. Next the task are broken down into several module following **Work Breakdown Structure**.

Next dependencies among various activities are determined. Dependencies among various activities are represented in the form of **activity networks**.

After finding dependencies:

- *resource allocation* is done using **Gantt chart**.
- After resource alloaction, **PERT chart** is prepared for *project monitoring and control activity*.

The point at which each activity completes is termed as ***milestone****.*

Software Project Team Organization

There are many ways to organize the project team. Some important ways are as follows :

1. Hierarchical team organization
2. Chief-programmer team organization
3. Matrix team, organization
4. Egoless team organization
5. Democratic team organization

Hierarchical team organization :

In this, the people of organization at different levels following a tree structure. People at bottom level generally possess most detailed knowledge about the system. People at higher levels have broader appreciation of the whole project.

Benefits of hierarchical team organization :

- It limits the number of communication paths and stills allows for the needed communication.
- It can be expanded over multiple levels.
- It is well suited for the development of the hierarchical software products.
- Large software projects may have several levels.

Limitations of hierarchical team organization :

- As information has to be travel up the levels, it may get distorted.
- Levels in the hierarchy often judges people socially and financially.
- Most technical competent programmers tend to be promoted to the management positions which may result in loss of good programmer and also bad manager.

Chief-programmer team organization :

This team organization is composed of a small team consisting the following team members :

- **The Chief programmer :** It is the person who is actively involved in the planning, specification and design process and ideally in the implementation process as well.
- **The project assistant :** It is the closest technical co-worker of the chief programmer.
- **The project secretary :** It relieves the chief programmer and all other programmers of administration tools.
- **Specialists :** These people select the implementation language, implement individual system components and employ software tools and carry out tasks.

Advantages of Chief-programmer team organization :

- Centralized decision-making
- Reduced communication paths
- Small teams are more productive than large teams
- The chief programmer is directly involved in system development and can exercise the better control function.

Disadvantages of Chief-programmer team organization :

- Project survival depends on one person only.
- Can cause the psychological problems as the “chief programmer” is like the “king” who takes all the credit and other members are resentful.
- Team organization is limited to only small team and small team cannot handle every project.
- Effectiveness of team is very sensitive to Chief programmer’s technical and managerial activities.

Matrix Team Organization :

In matrix team organization, people are divided into specialist groups. Each group has a manager. Example of Metric team organization is as follows :

Egoless Team Organization :

Egoless programming is a state of mind in which programmer are supposed to separate themselves from their product. In this team organization goals are set and decisions are made by group consensus. Here group, ‘leadership’ rotates based on tasks to be performed and differing abilities of members.

In this organization work products are discussed openly and all freely examined all team members. There is a major risk which such organization, if teams are composed of inexperienced or incompetent members.

Democratic Team Organization :
It is quite similar to the egoless team organization, but one member is the team leader with some responsibilities :

- Coordination
- Final decisions, when consensus cannot be reached.

Advantages of Democratic Team Organization :

- Each member can contribute to decisions.
- Members can learn from each other.
- Improved job satisfaction.

Disadvantages of Democratic Team Organization :

- Communication overhead increased.
- Need for compatibility of members.
- Less individual responsibility and authority.

Risk Management

A software project can be concerned with a large variety of risks. In order to be adept to systematically identify the significant risks which might affect a software project, it is essential to classify risks into different classes. The project manager can then check which risks from each class are relevant to the project.

There are three main classifications of risks which can affect a software project:

1. Project risks
2. Technical risks
3. Business risks

1. Project risks: Project risks concern differ forms of budgetary, schedule, personnel, resource, and customer-related problems. A vital project risk is schedule slippage. Since the software is intangible, it is very tough to monitor and control a software project. It is very tough to control something which cannot be identified. For any manufacturing program, such as the manufacturing of cars, the plan executive can recognize the product taking shape.

2. Technical risks: Technical risks concern potential method, implementation, interfacing, testing, and maintenance issue. It also consists of an ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence. Most technical risks appear due to the development team's insufficient knowledge about the project.

3. Business risks: This type of risks contain risks of building an excellent product that no one need, losing budgetary or personnel commitments, etc.

Other risk categories

1. **1. Known risks:** Those risks that can be uncovered after careful assessment of the project program, the business and technical environment in which the plan is being developed, and more reliable data sources (e.g., unrealistic delivery date)
2. **2. Predictable risks:** Those risks that are hypothesized from previous project experience (e.g., past turnover)
3. **3. Unpredictable risks:** Those risks that can and do occur, but are extremely tough to identify in advance.

Principle of Risk Management

1. **Global Perspective:** In this, we review the bigger system description, design, and implementation. We look at the chance and the impact the risk is going to have.
2. **Take a forward-looking view:** Consider the threat which may appear in the future and create future plans for directing the next events.
3. **Open Communication:** This is to allow the free flow of communications between the client and the team members so that they have certainty about the risks.
4. **Integrated management:** In this method risk management is made an integral part of project management.
5. **Continuous process:** In this phase, the risks are tracked continuously throughout the risk management paradigm.

Software Configuration Management

When we develop software, the product (software) undergoes many changes in their maintenance phase; we need to handle these changes effectively.

Several individuals (programs) works together to achieve these common goals. This individual produces several work product (SC Items) e.g., Intermediate version of modules or test data used during debugging, parts of the final product.

The elements that comprise all information produced as a part of the software process are collectively called a software configuration.

As software development progresses, the number of Software Configuration elements (SCI's) grow rapidly.

These are handled and controlled by SCM. This is where we require software configuration management.

A configuration of the product refers not only to the product's constituent but also to a particular version of the component.

Therefore, SCM is the discipline which

- Identify change
- Monitor and control change
- Ensure the proper implementation of change made to the item.
- Auditing and reporting on the change made.

Configuration Management (CM) is a technic of identifying, organizing, and controlling modification to software being built by a programming team.

What Is Requirement Analysis?

Let us understand the Requirement Analysis with the help of an example.

Customer/End-user expectation:

Customer/End-user receive:

As you can analyse from the above images that there is a mismatch in the end product to customer expectations. This could be due to myriad reasons, viz. incorrect implementation of customer requirements, faulty design, a wrong understanding of customer requirements by programmers and quality team, etc.

However, as you can see, be it any reason for wrong product delivery, the primary reason goes to requirement. Hence, if correct requirement understanding, capturing, implementation, and testing were done, it would have helped to mitigate the erroneous and incomplete product delivery to the customer/end-user.

A good product delivery requires correct requirement gathering, the efficient examination of requirements that are gathered, and finally clear requirement documentation, as the precondition. This entire process is also called **Requirement Analysis in Software Development Life Cycle (SDLC)**

Requirement Analysis Stages/Steps

As you can see, Requirement Analysis is the first activity in SDLC followed by Functional Specification and so on. Requirement analysis is a vital step in SDLC as it resonates with acceptance testing that is critical for product acceptance by customers.

In this tutorial, we will explain how requirement analysis is done in SDLC. We will also see the various steps involved, outcomes, challenges, and corrective measures in requirement analysis.

Requirement analysis starts with:

1. **Requirement gathering** is also called elicitation.
2. This is followed by **analysing** the collected requirements to understand the correctness and feasibility of converting these requirements into a possible product.
3. And finally, **documenting** the requirements collected.

#1) Requirement Gathering

To make sure that all the steps mentioned above are appropriately executed, clear, concise, and correct requirements must be gathered from the customer. The customer should be able to define their requirements properly and the business analyst should be able to collect them in the same way the customer is intending it to convey them.

Many a time it is not possible that requirement gathering is done efficiently by business analysts from the customer. This might be due to dependency on many people related to the expected end product, tools, environment, etc. Thus, it is always a good idea to involve all the stakeholders who could influence or could be influenced by the end product.

The possible stakeholder's group could be Software Quality engineers (both QC and QA), any third party vendor who could provide support in the project, end-user for whom the product is intended, software programmers, and another team within the organization who might provide a module or software platform, software libraries, etc. for product development.

Example: In an organization, they develop an ADAS product (surround-view camera system for a prestigious OEM) that needs **Autosar stack** and **Bootloader** binaries that are received from another supplier.
Involving various stakeholders in the requirement gathering stage helps in understanding various dependencies on each other and any possible future conflict could be averted.

Sometimes, it is a good idea to create a prototype model of the intended product planned, and show it to the customer. This is an excellent way to convey to customers what product are they expecting and how it may look later. This helps the customer in visualizing the product that they are expecting and helps them to come up with clear requirements.

This prototype creation depends on two types of product:

1. A similar product that customers intended, exists within the organization.
2. New product to be developed.

(i) In the first case, it is easier to show the customer what the end product would look like and how it would be developed. In automotive ADAS, it is possible to show customers another product that is already on the market and was developed within the organization.

For Example, A surround-view camera system developed for an OEM (GM, Volkswagen, BMW, etc.) can be showcased to another OEM.
Please note, it is not wise to show product/proto product to the customer which is under development as it may violate the Non-Disclosure Agreement signed with another customer for whom that product is being developed. It may also result in an unnecessary legal tussle.
Requirement Gathering Example: In the ADAS surround-view camera system, one possible user story could be: "As a user, I should be able to see what's there in the rear-view of my car".
There could be many **"why,"** questions asked on each user story, which will help the customer in providing more detailed information on the requirement.
In the above user story, if a customer says "As a user, I should be able to see what's there in the rear-view of my car", asking the question **"why"** could give "As a user, I should be able to see what's there in the rear-view of my car, *so that I can park my car safely*".
Asking the question **"why"** also helps in creating objective and atomic requirements from humongous natural language statements given by the customer. This can easily be implemented later in the code.
Another way of gathering the requirement is in the form of **use cases**. A use case is a step-by-step approach for achieving a certain result. This does not tell how the software will work on user input rather it says, what is expected of user inputs.
Example:

#2) Analysing Gathered Requirements

Post requirement gathering, analysis of requirement starts. At this stage, various stakeholders sit and do a brainstorming session. They analyse the requirements gathered and look for the feasibility to implement them. They discuss it with each other and any ambiguity is sorted out.

This step is important in the requirement analysis process due to the following main reasons:
(i) Customer may provide some requirements which could be impossible to implement due to various dependencies.
Example: Customers may ask to surround-view the camera system with a rear-view camera feature that will help in **parking** the car. The customer may also ask for the **Trailer** hitch feature which also uses the rear camera to work.

If the customer states a requirement that the rear-view camera for **parking** assistance should work at all times without an exception, then the **Trailer** feature would never work and vice versa.
(ii) A business analyst might have understood the requirement from the **customer** differently than how a **programmer** would have interpreted it.
Since programmers think as technical experts, it is always possible that customer requirements are incorrectly converted to functional specifications which will later be incorrectly made to architecture and design documents and subsequently to code. The impact is exponential and so should be checked.

Software Requirement Specifications

The production of the requirements stage of the software development process is **Software Requirements Specifications (SRS)** (also called a **requirements document**). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analysed. **SRS** is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be written by a developer of the system. The two methods create entirely various situations and establish different purposes for the document altogether. The first case, SRS, is used to define the needs and expectation of the users. The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.

Characteristics of good SRS

Following are the features of a good SRS document:

1. Correctness: User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.

2. Completeness: The SRS is complete if, and only if, it includes the following elements:

(1). All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.

(2). Definition of their responses of the software to all realizable classes of input data in all available categories of situations.

(3). Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

3. Consistency: The SRS is consistent if, and only if, no subset of individual requirements described in its conflict. There are three types of possible conflict in the SRS:

(1). The specified characteristics of real-world objects may conflict. For example,

(a) The format of an output report may be described in one requirement as tabular but in another as textual.

(b) One condition may state that all lights shall be green while another states that all lights shall be blue.

(2). There may be a reasonable or temporal conflict between the two specified actions. For example,

(a) One requirement may determine that the program will add two inputs, and another may determine that the program will multiply them.

(b) One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

(3). Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement's and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

4. Unambiguousness: SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

5. Ranking for importance and stability: The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional.

6. Modifiability: SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

7. Verifiability: SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.

Properties of a good SRS document

The essential properties of a good SRS document are the following:

Concise: The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

Structured: It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

Black-box view: It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behaviour of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behaviour of the system. For this reason, the SRS report is also known as the black-box specification of a system.

Conceptual integrity: It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

Verifiable: All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.

Formal System Specification Axiomatic Specification

Formal Methods

Formal specification is part of a more general collection of techniques that are known as 'formal methods'

"Formal Methods" These are all based on the mathematical representation and analysis of software

Formal methods include

- Formal specification
- Specification analysis and proof
- Transformational development
- Program verification

Formal Methods in reality

When software was first developed

a. Is was done using assembly language

b. No OO, no high level languages
c. limited understanding of software testing

Modern software development

a. Many ways to make high quality software
b. Mostly formal methods not used
c. The most acceptable techniques are approaches like programming by contract (e.g. Eiffel)

Acceptance of Formal Methods

- Formal methods have not become mainstream software development techniques as was once predicted.
- Other software engineering techniques have been successful at increasing system quality. Hence the need for formal methods has been reduced.
- Market changes have made time-to-market rather than software with a low error count the key factor. Formal methods do not reduce time to market.
- The scope of formal methods is limited. They are not well suited to specifying and analysing user interfaces and user interaction for example.
- Formal methods are hard to scale up to large systems

Use of Formal Methods

- Their principal benefits are in reducing the number of errors in systems so their main area of applicability is critical systems:
  - Air traffic control information systems,
  - Railway signalling systems
  - Spacecraft systems
  - Medical control systems
- In this area, the use of formal methods is most likely to be cost-effective.

Axiomatic Specification:

In the **Axiomatic Specification** of a system, first-order logic used to write the pre-and post-conditions to specify the operations of the system in the form of axioms. The pre-conditions capture the conditions that must be satisfied before an operation can successfully be invoked.

In essence, the pre-conditions capture the requirements on the input parameters of a function. The post-conditions are the conditions that must be satisfied when a function post-conditions are essentially constraints on the results produced for the function execution to be considered successful.

Algebraic Specification:

In the **Algebraic Specification** technique, an object class or type is specified in terms of relationships existing between the operations defined on that type. It first brought into prominence by **Guttag** (1980-1985) in the specification of abstract data types. Various notations of algebraic specifications have evolved, including those based on OBJ and Larch languages.
Essentially, algebraic specifications define a system as a heterogeneous algebra. A heterogeneous algebra is a collection of different sets on which several operations are

defined. Traditional algebras are homogeneous. A homogeneous algebra consists of a single set and several operations defined in this set such as – { **+, -, *, /** }.

EXECUTABLE SPECIFICATION AND 4GL

When the specification of a system is expressed formally or is described by using a programming language, then it becomes possible to directly execute the specification without having to design and write code for implementation. However, executable specifications are usually slow and inefficient, 4GLs (4th Generation Languages) are examples of executable specification languages. 4GLs are successful because there is a lot of large granularity commonality across data processing applications which have been identified and mapped to program code. 4GLs get their power from software reuse, where the common abstractions have been identified and parameterized. Careful experiments have shown that rewriting 4GL programs in 3GLs results in up to 50 per cent lower memory usage and also the program execution time can reduce up to ten folds.

Software Design Process

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language.
The software design process can be divided into the following two levels of phases of design:

- Preliminary (or high-level) design
- Detailed

The meaning and scope of these two stages can vary considerably from one design methodology to another. However, for the traditional function-oriented design approach, it is possible to define the objectives of the high-level design as follows:

Through high-level design, a problem is decomposed into a set of modules. The control relationships among the modules are identified, and also the interfaces among various modules are identified

The outcome of high-level design is called the program structure or the software architecture. High-level design is a crucial step in the overall design of a software. When the high-level design is complete, the problem should have been decomposed into many small functionally independent modules that are cohesive, have low coupling among themselves, and are arranged in a hierarchy. Many different types of notations have been used to represent a high-level design. A notation that is widely being used for procedural development is a tree-like diagram called the structure chart. Another popular design representation technique called UML that is being used to document object-oriented design, involves developing several types of diagrams to document the object-oriented design of a systems. Though other notations such as Jackson diagram

[1975] or Warnier-Orr [1977, 1981] diagram are available to document a software design, we confine our attention in this text to structure charts and UML diagrams only.

Once the high-level design is complete, detailed design is undertaken. During detailed design each module is examined carefully to design its data structures and the algorithms. The outcome of the detailed design stage is usually documented in the form of a module specification (MSPEC) document. After the high-level design is complete, the problem would have been decomposed into small modules, and the data structures and algorithms to be used described using MSPEC and can be easily grasped by programmers for initiating coding. In this text, we do not discuss MSPECs and confine our attention to high-level design

Characteristics of a good software design

For good quality software to be produced, the software design must also be of good quality. Now, the matter of concern is how the quality of good software design is measured? This is done by observing certain factors in software design. These factors are:

1. Correctness
2. Understandability
3. Efficiency
4. Maintainability

Now, let us define each of them in detail,

1) Correctness

First of all, the design of any software is evaluated for its correctness. The evaluators check the software for every kind of input and action and observe the results that the software will produce according to the proposed design. If the results are correct for every input, the design is accepted and is considered that the software produced according to this design will function correctly.

2) Understandability

The software design should be understandable so that the developers do not find any difficulty to understand it. Good software design should be self- explanatory. This is because there are hundreds and thousands of developers that develop different modules of the software, and it would be very time consuming to explain each design to each developer. So, if the design is easy and self- explanatory, it would be easy for the developers to implement it and build the same software that is represented in the design.

3) Efficiency

The software design must be efficient. The efficiency of the software can be estimated from the design phase itself, because if the design is describing software

that is not efficient and useful, then the developed software would also stand on the same level of efficiency. Hence, for efficient and good quality software to be developed, care must be taken in the designing phase itself.

4) Maintainability

The software design must be in such a way that modifications can be easily made in it. This is because every software needs time to time modifications and maintenance. So, the design of the software must also be able to bear such changes. It should not be the case that after making some modifications the other features of the software start misbehaving. Any change made in the software design must not affect the other available features, and if the features are getting affected, then they must be handled properly.

Coupling and Cohesion

Module Coupling

In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other. **Uncoupled modules** have no interdependence at all within them.

The various types of coupling techniques are shown in fig:

A good design is the one that has low coupling. Coupling is measured by the number of relations between the modules. That is, the coupling increases as the number of calls between modules increase or the amount of shared data is large. Thus, it can be said that a design with high coupling will have more errors.

1. **No Direct Coupling:** There is no direct coupling between M1 and M2.

In this case, modules are subordinates to different modules. Therefore, no direct coupling.

2. Data Coupling: When data of one module is passed to another module, this is called data coupling.

3. Stamp Coupling: Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc. When the module passes non-global data structure or entire structure to another module, they are said to be stamp coupled. For example, passing structure variable in C or object in C++ language to a module.

4. Control Coupling: Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.

5. External Coupling: External Coupling arises when two modules share an externally imposed data format, communication protocols, or device interface. This is related to communication to external tools and devices.

6. Common Coupling: Two modules are common coupled if they share information through some global data items.

7. Content Coupling: Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

Module Cohesion

In computer programming, cohesion defines to the degree to which the elements of a module belong together. Thus, cohesion measures the strength of relationships between pieces of functionality within a given module. For example, in highly cohesive systems, functionality is strongly related.

Cohesion is an **ordinal** type of measurement and is generally described as "high cohesion" or "low cohesion."

1. **Functional Cohesion:** Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.
2. **Sequential Cohesion:** A module is said to possess sequential cohesion if the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.
3. **Communicational Cohesion:** A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.
4. **Procedural Cohesion:** A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.
5. **Temporal Cohesion:** When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.
6. **Logical Cohesion:** A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.
7. **Coincidental Cohesion:** A module is said to have coincidental cohesion if it performs a set of tasks that are associated with each other very loosely, if at all.

Differentiate between Coupling and Cohesion

Coupling	Cohesion
Coupling is also called Inter-Module Binding.	Cohesion is also called Intra-Module Binding.

Coupling shows the relationships between modules.	Cohesion shows the relationship within the module.
Coupling shows the relative **independence** between the modules.	Cohesion shows the module's relative **functional** strength.
While creating, you should aim for low coupling, i.e., dependency among modules should be less.	While creating you should aim for high cohesion, i.e., a cohesive component/ module focuses on a single function (i.e., single-mindedness) with little interaction with other modules of the system.
In coupling, modules are linked to the other modules.	In cohesion, the module focuses on a single thing.

Layered Technology

Software engineering is fully a layered technology; to develop software we need to go from one layer to another. All the layers are connected and each layer demands the fulfilment of the previous layer.

Fig: The diagram shows the layers of software development

Layered technology is divided into four parts:

1. A quality focus: It defines the continuous process improvement principles of software. It provides integrity that means providing security to the software so that data can be accessed by only an authorized person, no outsider can access the data. It also focuses on maintainability and usability.
2. Process: It is the foundation or base layer of software engineering. It is key that binds all the layers together which enables the development of software before the deadline or on time. Process defines a framework that must be established for the effective delivery of software engineering technology. The software process covers all the activities, actions, and tasks required to be carried out for software development.

Process activities are listed below: -

- **Communication:** It is the first and foremost thing for the development of software. Communication is necessary to know the actual demand of the client.
- **Planning:** It basically means drawing a map for reduced the complication of development.
- **Modelling:** In this process, a model is created according to the client for better understanding.
- **Construction:** It includes the coding and testing of the problem.
- **Deployment: -** It includes the delivery of software to the client for evaluation and feedback.
-

3. Method: During the process of software development the answers to all “how-to-do” questions are given by method. It has the information of all the tasks which includes communication, requirement analysis, design modelling, program construction, testing, and support.

4. Tools: Software engineering tools provide a self-operating system for processes and methods. Tools are integrated which means information created by one tool can be used by another.

System Design Strategy

A good system design is to organize the program modules in such a way that are easy to develop and change. Structured design techniques help developers to deal with the size and complexity of programs. Analysts create instructions for the developers about how code should be written and how pieces of code should fit together to form a program.

Importance:

1. If any pre-existing code needs to be understood, organized, and pieced together.
2. It is common for the project team to have to write some code and produce original programs that support the application logic of the system.

There are many strategies or techniques for performing system design. They are:

- **Bottom-up approach:**
  The design starts with the lowest level components and subsystems. By using these components, the next immediate higher-level components and subsystems are created or composed. The process is continued till all the components and subsystems are composed into a single component, which is considered as the complete system. The amount of abstraction grows high as the design moves to more high levels.

  By using the basic information existing system, when a new system needs to be created, the bottom-up strategy suits the purpose.

Advantages:

- The economics can result when general solutions can be reused.
- It can be used to hide the low-level details of implementation and be merged with the top-down technique.

Disadvantages:

- It is not so closely related to the structure of the problem.
- High-quality bottom-up solutions are very hard to construct.
- It leads to the proliferation of 'potentially useful' functions rather than the most appropriate ones.

- **Top-down approach:**
  Each system is divided into several subsystems and components. Each of the subsystems is further divided into a set of subsystems and components. This process of division facilitates in forming a system hierarchy structure. The complete software system is considered as a single entity and in relation to the characteristics, the system is split into sub-system and component. The same is done with each of the sub-systems.

- This process is continued until the lowest level of the system is reached. The design is started initially by defining the system as a whole and then keeps on adding definitions of the subsystems and components. When all the definitions are combined together, it turns out to be a complete system.

  For the solutions of the software that need to be developed from the ground level, top-down design best suits the purpose.

Advantages:

- The main advantage of the top-down approach is that its strong focus on requirements helps to make a design responsive according to its requirements.

Disadvantages:

- Project and system boundaries tend to be application specification-oriented. Thus it is more likely that advantages of component reuse will be missed.
- The system is likely to miss, the benefits of a well-structured, simple architecture.
- **Hybrid Design:**
  It is a combination of both the top-down and bottom-up design strategies. In this, we can reuse the modules.

S/w engg.

Unit - 4

27-10-2020

Coding and Testing :-

- Coding is an essential part of software engineering to develop a dynamic and reliable software.
- By using Coding we can transform the design of a system into a programming language.
- we should implement a well define standard for coding known as Coding standard.
- By using Coding standards our code will be easily understandable, easy debugging, Enhance & we can extended it according to our requirement.
- The Coding standard guidlines are the rules how to write the Code, so the overall Coding becomes understandable.

The following steps describes the representation of Coding standards :-

(1) Less use of global variable.
(2) Exception handling mechanism.
(3) Naming Convention for global variables, local variables and Constant.
(4) use of Comments.
(5) A particular module should Contain Maximum 1-20 lines of Code.
(6) Don't use such type of coding i,e very clever or difficult to understand.
(7) Don't use same identifier for multiple purpose
(8) Don't use short form for name of variable & methods.

Testing :-

Testing means software testing. It finding bug in the developed s/w for Testing software.

Types of Testing :-

(1) Black Box Testing :-

→ Black box testing is that testing in which the tester does not have the knowledge of the coding.

→ In this testing we only check the functionality of the software that is input be pass and output the slw produce.

(II) Whitebox Testing :-

→ In white box Testing the tester has knowledge of Coding and internal structure of testing software whether the internal structure is Correct or not.

(III) unit Testing :-

→ unit testing is the process of checking small pieces of code to ensure that the individual parts of a program work properly on their own.

→ unit testing are used to test individual blocks of functionality.

→ unit testing is done by developer.

(iv) Integration Testing :-

→ Integration testing is Conducted to evaluate the Compliance of a system or Component with Specified functional requirements.

→ It occurs after unit testing and before system testing.

→ Types of Integration Testing.

(1) Big Bang

(2) Mixed → A Mixed integration testing follows a Combination of topdown & bottomup testing approach.

(3) Top down

(4) Bottom-up

(1) Big Bang Approach :-

→ Combing all the modules once and verifying the functionality after Completion of individual module testing.

(N) Top down Approach :-

→ In top-down approach, testing takes place from top to bottom.

→ High level modules are tested first and then low level modules and finally integrating the low level module to high level to ensure the system is working as intended.

(iii) Bottom-up Approach :—

→ Here testing takes place from bottom to up.
→ Lowest level modules are tested first and then high level modules and finally integrating the high level modules to low level to ensure the system is working as intended.

System Testing :—

→ System testing is a level of testing that validates the complete & fully integrated s/w product.
→ The purpose of system test is to evaluate the end-to-end system specification.
→ System testing is a black-box testing.
→ System testing categories based on : who is doing the testing ?
→ System testing categories based on : Functional Non-functional requirements.

User Acceptance Testing :—

→ user acceptance testing is used to determine wheather the product is working for the user correctly
→ Specific requirements which are quite often used by the customers are primarily picked for the testing

purpose. This is also termed as End-User Testing.

→ Normally the user to be tested the given s/w by using 3 testing tech.

(i) α- Testing
(ii) β- Testing
(iii) Acceptance Testing.

Performance Testing :-

→ Performance testing is carried out to check wheather the system meets the non-functional requirements identified in the SRS document.

→ There are several types of performance testing.

→ The type of performance testing to be carried out on a system depend on the different non-functional requirements of the system documented in SRS document.

→ All the performance test considered as black box tests.

- Stress testing
- volume testing
- Configuration testing
- Compatibility testing
- Regression testing
- Recovery testing
- Maintenance testing
- documentation testing
- usability testing

Usability

→ usability testing concerns checking the

user interface to see if it meets all user requirements concerning the user interface.

⇒ During usability testing, the display screens, report formats, and other aspects relating to the user interface requirements are tested.

Compatibility :-

⇒ This type of testing is required when the system interfaces with other types of system.

⇒ Compatibility check whether the interface functions perform as required. for instance.

⇒ In this testing is required to test the speed and accuracy of data retrieval.

* Black box :-
* white box :- Difference bet verification & validation

System documentation :-

⇒ when various kinds of software products are developed then not only the executable files and the source code are developed but also various kinds of documents such as users manual, software requirement specification documents, design documents, test documents, installation manual etc

→ Good documents enhance understandability and maintainability of a software product. They reduce the effort & time required for maintenance.
→ use documents help the users in effectively using the system.
→ Good documents help in effectively handling the manpower turnover problem.
→ Different types of s/w documents can broadly classified into the following.
— Internal Documentation
— External Documentation.

(i) Internal Documentation :—
→ It is provided through appropriate module headers & comments embedded in the source code.
→ Internal documentation is also provided through the useful variable names, module & fun headers, code indentation, code structuring etc.

(ii) External Documentation :—
→ It is provided through various types of supporting documents such as user's manual, software requirements specification document (SRS), design document, Test document etc.

Debugging :—

Identifying errors in a program code and then fix them up are known as debugging.
→ The following are some of the approaches popularly adopted by programmers for debugging

(i) Brute Force Method :-

→ The program is loaded with print statements to print the intermediate values with the hope that some of the printed value will help to identify the statement in error.

→ This approach becomes more systematic with the use of symbolic debugger

(ii) Backtracing which an error symptom has been observed, the source code is traced backwards until the error is discovered.

Program Analysis tools :-

→ A program analysis tool means an automated tool takes the source code the executable code of a program as input and produces reports regarding several important characteristics of the program.
such as size, complexity, adequacy of commenting etc.

→ Two broadly categories of program analysis tools.
- Static analysis tool
- Dynamic analysis tool

(i) Static Analysis Tools :-

→ It is also program analyzer tools
→ It analyze some structural representation of a program

arrive at certain analytical conclusion.

(II) Dynamic program analytic tools :-

→ This tech. requires the program to be executed and its actual behavior recorded.

Software Maintenance :-

→ S/w Maintenance is a part of s/w engg. development life cycle.

→ It refers to the process of modifying and updating a s/w system after it has been delivered to the customer.

→ This can including fixing bugs, adding new features, improving performance or updating the s/w to work with new h/w or s/w system

→ S/w maintenance is needed for
- Correct errors
- change in user requirement with time
- change h/w / s/w requirements.
- To improve system efficiency
- To optimize the code to run faster
- To modify the components
- ~~To reduce any unwanted side~~

Types of s/w maintenance :-

1. Corrective maintenance to correct any remaining errors regardless of where they may cause specification design, coding, testing & documentation etc.

2. Adaptive Maintenance :-

It contains modify the s/w to match changes in the ever changing environment.

3. Preventive

→ It involves the concept of reengineering in which an old system with old technology is re-engineered using new technology.

(4) Perfective :-

→ It defines improving processing efficiency or performance or restricting the s/w to enhance changeability.

Black Box Testing :-

→ Tester only Access on input & output of your product.

→ It didn't have any knowledge of internal Code structure, Implementation details & Internal path

→ Tester gives Input value to examine its functionality & checks wheather function is giving expected output or not.

→ If the fun^n produce Correct output, then it is passed in testing, otherwise failed.

→ It is also known as Behavioural testing, functional & Closed box Testing.

→ performed by software Tester.

Black Box Technique

① Equivalence partitioning :-

The input values that provide to System are divided into diff. classes or groups based on similarity in the outcome

→ Instead of using each & every input value, use any one value from the group to test outcome.

Ex :-

Any Registration

Age [ ] Accept a value from 18 to 60

Invalid	valid	Invalid
<= 17	18.00	>= 61

(ii) Boundary Value Analysis :-

Age [ ] Accept from 18 to 60

Ex:

Boundary value Analysis

Invalid min-1	valid min, min+1, max-1, max	Invalid max+1
17	18, 19, 55, 56	57

→ Boundary values are those that contain the upper and lower limit of a variable.

→ while entering boundary value whether the software is producing correct output or not.

(iii) Decision Table Testing :-

→ various input combinations & their respective system behavior are captured in tabular form.

→ check logical relationship betn two and more than two inputs.

Ex:- Gmail Account

Email ID	T	T	F	F
password	T	F	T	F
Decision table	Account page opens	error incorrect password	Incorrect Email	

(4) Error Guessing :-

Tester uses experience to guess the ~~pro~~ problematic areas of the s/w.

Ex: Divided by zero

(5) All pairs Testing Tech

→ It is used to test all the possible discrete combinations of values.

→ This combinational method is used for testing the application that uses checkbox input, radio button, list box, textbox etc.

Advantages :-

→ The tester doesnot need to have more functional knowledge or programming skills.

→ It is efficient for implementing the test in the larger system.

→ Tests are executed from the user's or client's point of view.

→ Disadvantages :-

→ without clear programming knowledge, test cases are difficult to implement.

→ It doesn't reveal the errors in the control structure.

→ working large sample space of inputs can be exhaustive & consumes a lot of time.

White Box Testing :-

→ white box testing techniques analyze Internal design, code structure & working of the complete s/w product.

→ It is also called clear box testing, open box testing, Transparent box testing, code based testing & Glass box testing.

- Tester has knowledge about internal structure.
- It is mostly done by s/w developers.
- perform with unit testing & Integration testing.

Why do you verify in white box testing :-

→ Internal Security of Code

→ poorly structured paths in the coding.

→ The flow of specific inputs through the code
→ Loops, decision conditions.
→ Each statement of your program.
→ Function & modules are on an individual bias basis.

Technique

(1) Path Coverage / path testing.

→ Testing is dependent on the program's control or flow structure.

→ All possible paths are defined with entry to exit points in the system.

2. Loop Testing :—

→ There are loops such as
- while
- for
- Do while etc.

→ It checks all simple
- Nested
- Concatenated loops

→ It checks ending cond^n if working correctly & if the size of the cond^n is enough

Ex:- while (condition 2)
{
statement (s);
}

while (condition 2)
{
statement (s);
}

(3) Branch Coverage / Condition Testing

→ logical cond^n for every value are checked, weather it is true or false.

→ Both the if & else cond^n are verified.

(4) Statement Coverage :-

→ This tech. tester traverse all statements in code.

each statement of the code is executed at least once

→ each line of the code is verified.

Advantages

→ White box testing optimizes code so hidden errors can be identified.

→ It can be easily automated.

→ Early detection of errors that improve code quality.

→ It covers most path of code.

Disadvantages :—

→ Complex, Expensive & Time consuming.

→ Requires professional resources with detailed knowledge of programming.

→ May provide some garbage value.