Software Engineering

INTRODUCTION

Software engineering is application of a systematic and disciplined approach to the development, operation and maintenance of software in an efficient and cost effective way.

Programmes v/s Software product

Programmes: A computer program is a sequence of instructions, written to perform a specified task with a computer. A collection of computer programs and related data is referred as software.

Software program: there are three basic entities which are generally used while defining the principles of software engineering are –

1. Software process
2. Software projects
3. Software products

- Programmes are set of instructions for computer to perform a specified task.
- Programmes are developed by individuals.
- Programmes are smaller in size.
- In case of programmes, programmer himself is the sole user.
- In programmes, very little documentation are required.

- Software product are merchandise consisting of a computer program that is offered to sale.
- Software products are developed by multiple users.
- Software products are extremely large in size.
- In software products, most user are not involved with development.
- Software products must be well documented.

Difference between programmes and software product

Emergence of software engineering

1. Early computer programming:- early commercial computer were very slow and too elementary as compared to today's standards. In 1968, key concept of modularity and information hiding were also introduced to help programmers deals with every increasing complexity of software system.

2. High level programming:- the high level language that are more powerful, easier to use and directed towards specialized classes of problems. A list of some high level language:-

- COBOL
- PL/I
- BASIC
- PASCEL
- C
- C++
- JAVA
- FORTAN

SOFTWARE DESIGN

Software design process have two levels:-

1. System design or external design
2. Internal design or detail design

There are basically three approaches to software design-

- Data structured oriented design
- Control flow based design
- Object oriented design

Control Flow-Based Design (late 60s)

- Size and complexity of programs increased further:
  - exploratory programming style proved to be insufficient.
- Programmers found:
  - very difficult to write cost-effective and correct programs.

Structured programming

- 3 types of control structure
  - Sequential structure
    - Built into Python
  - Selection structure
    - The `if` statement
    - The `if/else` statement
    - The `if/elif/else` statement
  - Repetition structure
    - The `while` repetition structure
    - The `for` repetition structure
- All have one entry point and one exit point

Data Structure Oriented Design

(Early 70s)

- Example of data structure-oriented design technique:
  - Jackson's Structured Programming(JSP) methodology
    - Developed by Michael Jackson in 1970s.

THE OBJECT-ORIENTED DESIGN PROCESS

- The object-oriented design process consists of the following-activities:
  - Apply design axioms to design classes, their attributes, methods, associations, structures, and protocols .
    - Refine and complete the static UML class diagram by adding details to the UML class diagram.
      - Refine Attributes.
      - Design methods and protocols by utilizing a UML activity diagram to represent the method's algorithm
      - Refine associations between classes (if required).
      - Refine class hierarchy and design with inheritance (if required).

THANK YOU

Software life cycle models

Various types of life cycle models have been provided. However all of these models are having same phases of working i.e. feasibility study, requirement analysis and specification, design, coding, testing & maintenance, but they only differ in methods of their implementation.

However a large no. of models have been purposed. But we will discuss only some important models that are:-

- Classical waterfall model
- Iterative waterfall model
- Evolutionary mode
- Prototyping model
- Spiral model

Classical waterfall model

This model is base of all the other model. This model divides the life cycle of software development process into following phases:-

1. Feasibility study
2. Requirements analysis and specification
3. Design
4. Coding and unit testing
5. Integration and system testing
6. maintenance

We will now discuss the various activities carried out during each phase of life cycle.

1. *Feasibility study:-* during this phase, we generally study two type of feasibility.

- ✓ Financial feasibility:- it is tested that if the project can be completed under the available resources or not or if the finance is available to fulfil the requirements.
- ✓ Technical feasibility:- it is tested that if the result is collected which forms the input to the further system.

2. Requirement Analysis and Specification:-

During the requirement analysis and specification phase, the exact requirements of the customer are analysed and a proper documentation mentioning, the requirements is prepared. Two main activities performed during this phase are:-

- Requirements Analysis
- Requirements Specification

3. Design:-

After the SRS document have been prepared in the software requirements and specification phase. In the design phase, the requirements specification mentioned in the SRS document is transformed into the structural form. This structural form is called software architecture of the project. This architecture should be such that it can be easily convert able into a coded form using implementation in some programming language. Various design approaches used in the industry can be categorized into two types:-

- Traditional design approach
- Object oriented design approach

4. Coding and Unit Testing

- *Coding testing*:- the purpose of coding and unit testing phase of software development is to translate completed, the software design into source code. The coding phase is also some times called implementation phase. Each components of design is implemented as a program modules. The end product of this phase is set of program modules that have been individually tested. After coding is each module is tested.

- *Unit Testing*:- it involves testing each modules separately from other module, then debugging and documents it. It is the most efficient way to debug errors identified at this stage.

5. Integration and system testing:-

During unit testing various test case and testing criteria are prepared for testing correct working of individual module. Now in this phase, the modules implemented in the implementation phase are integrated together and tested.

- *Integration Testing*:- In this testing, modules are integrated one by one in step and after each addition of one module, the resulting system is tested again and again. The function of testing is to verify that after the integration of new module the system is working correctly or not. When modules have been successfully integrated and tested. System testing is carried out.

- *System Testing*:- The function of system testing is to ensure that the developed system conforms to its requirement specify in SRS document. System testing consist of three different kinds of testing activities.
  - αtesting:- *α testing* is system testing performed by development team.
- βtesting:- β testing is system testing performed by a friendly set of customers.
- Acceptance testing:- This is system testing performed by the customer himself after the product delivery to determine whether to accept the delivered product or to reject it.
- System testing:- is carried out according to the system test plan document which was prepared during the requirement specification phase.

6. Maintenance :-

Maintenance of a software product requires much more effort than the effort necessary to develop the product itself. The ratio of effort of development of system and its maintenance is about 40 : 60.

There are three types of maintenance:-

- Adaptive Maintenance
- Perfective Maintenance
- Corrective Maintenance

❑ Advantages:-

1. It is simple to use and understand.
2. Each stage has well defined deliverable or milestone.
3. Required amount of resources are minimal.
4. Phase are processed and completed one at a time.
5. Works well for smaller project where requirements are very well understood.

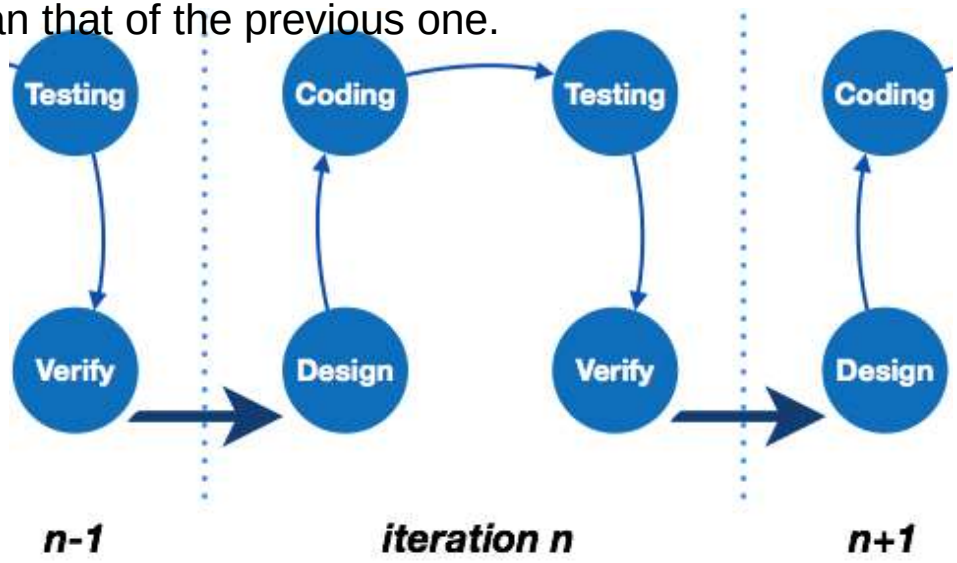
❑ Disadvantages:-

1. Small changes or errors that arise in the complete software may cause a lot of problems.
2. Client is not very clear of what he exactly wants from the software.
3. Only able to use when the requirement are fixed.
4. High amount of risk and uncertain.
5. Poor model for long and on going project.

ITERATIVE WATERFALL MODEL

- This model leads the software development process in iterations. It projects the process of development in cyclic manner repeating every step after every cycle of SDLC process.

- The software is first developed on very small scale and all the steps are followed which are taken into consideration. Then, on every next iteration, more features and modules are designed, coded, tested, and added to the software. Every cycle produces a software, which is complete in itself and has more features and capabilities than that of the previous one.



Iterative Waterfall Model

ADVANTAGE

- It is much better model of the software process.
- It allows feedback to proceeding stages.
- It can be used for project where in the requirement are not well understood.
- In Iterative model less time is spent on documenting and more time is given for designing.
- Is useful when the project is large size.

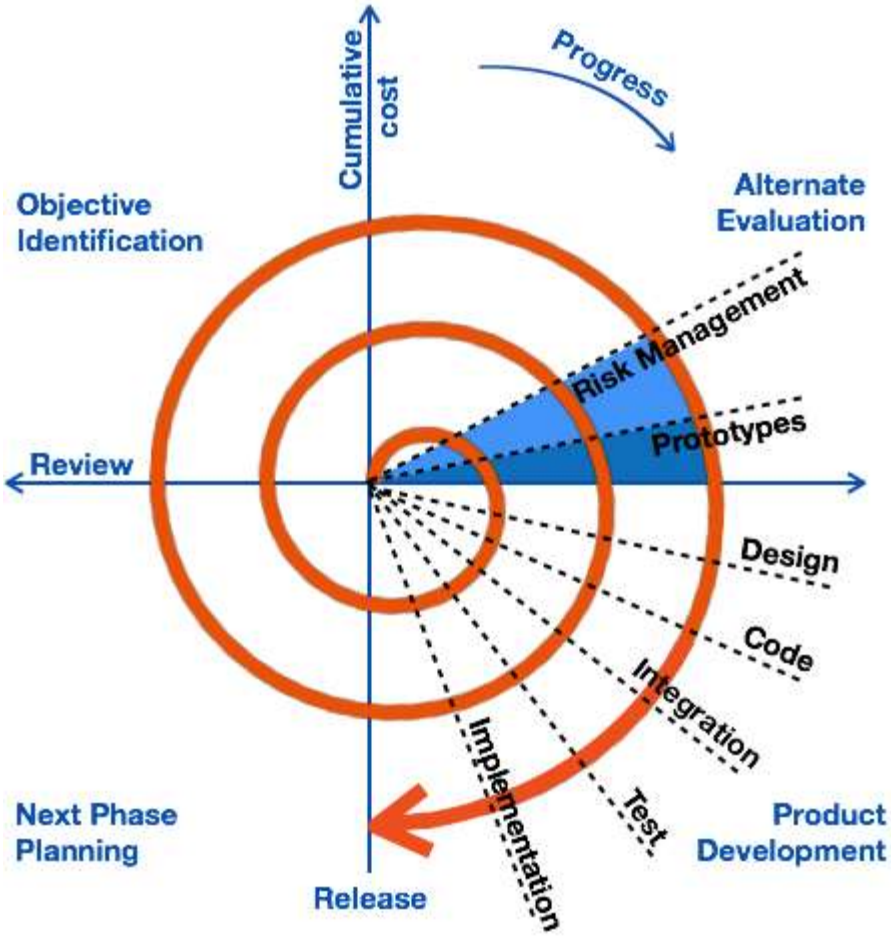
DISADVANTAGE

- It is not easy to manage this model.
- No clear milestones in the development process.
- No stage is really finished.
- The number of resources requirement are more.
- Exit criteria for the software are unknown.
- Not applicable for smaller projects.

Spiral Model

- Spiral model is a combination of both, iterative model and one of the SDLC model. It can be seen as if you choose one SDLC model and combined it with cyclic process (iterative model).

- This model considers risk, which often goes un-noticed by most other models. The model starts with determining objectives and constraints of the software at the start of one iteration. Next phase is of prototyping the software. This includes risk analysis. Then one standard SDLC model is used to build the software. In the fourth phase of the plan of next iteration is prepared.



SPIRAL MODEL

ADVANTAGE

- Every time a new prototypes is obtained, it is revaluated again and again by customer. So more customer involements is there.
- Better productivity through reuse capabilities.
- Proper control over cost, time and manpower requirement for a project work.

DISADVANTAGE

- This model requires risk identification, its projection, risk assessment and management which is not an easy task.
- Cost and time estimations are also not very easy.
- This model is not suitable for smaller project as the cost of risk analysis is greater than cost of the entire project.

Prototype Model

The **Prototyping Model** is a systems development method (SDM) in which a **prototype** (an early approximation of a final system or product) is built, tested, and then reworked as necessary until an acceptable **prototype** is finally achieved from which the complete system or product can now be developed.

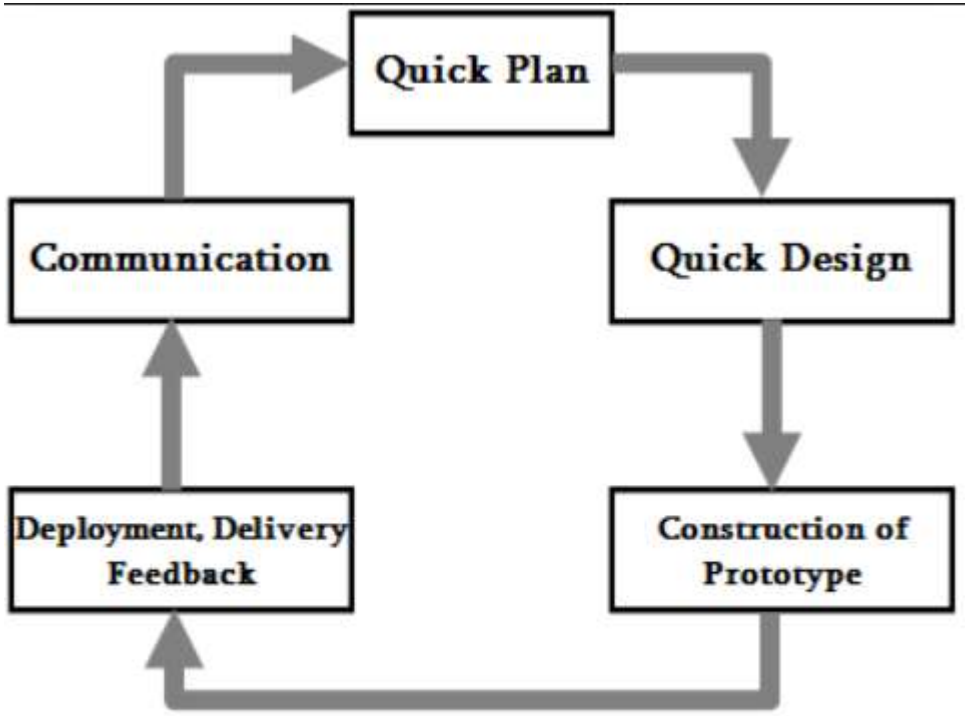


Figure : Flowchart of Prototyping Model

PROTOTYPE MODEL

ADVANTAGE

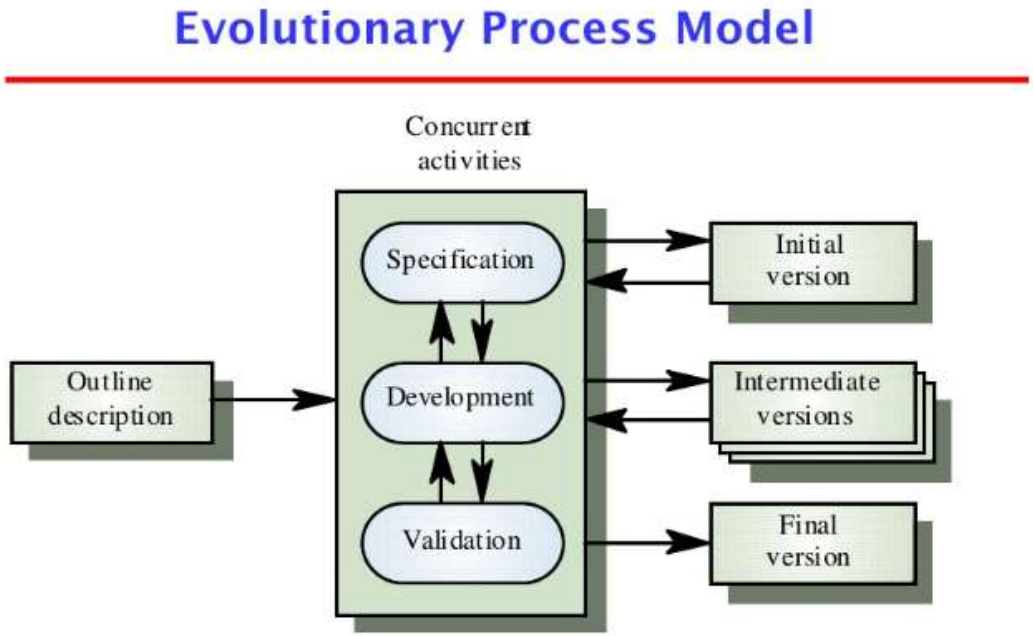
- Cost effective (Development cost reduced).
- Increase system development speed.
- Enables a higher output for user.
- Users are actively involved in the development.
- The user get a better understanding of the system being developed.

DISADVANTAGE

- Project management difficulties.
- Not suitable for large applications.
- Often lack flexibility.
- Possibility of implementing systems before they are ready.
- Possibility of causing system to be unfinished.
- Producer might produce a system inadequate for overall organization needs.

Evolutionary Model

➢**Evolutionary Model:** The Evolutionary model is suitable for large projects which can be decomposed into a set of modules for incremental development and delivery. This model is widely used in object-oriented development projects. This model is only used if incremental delivery of the system is acceptable to the customer.



Evolutionary Model

ADVANTAGE

- As each version produced in this model is a functioning system capable of performing some useful work, so when various versions of the system is delivered to the user from time to time, user got the chance to study the partially developed system.
- After development of each new versions of the system, testing takes place. As each module of the product get tested throughly, hence chances of errors in the final product get reduced.

DISADVANTAGE

- This model can only be used for very large projects which can be used for very large projects which can be subdivided into modules that can be implemented independent of each others.
- Even for very large projects, its very difficult to subdivide the problem into several functional units that can be incrementally this system.

Comparision Of Different Life Cycle Models

Comparison of various SDLC Models

Features /models	WATERFALL MODEL	SPIRAL MODEL	PROTOTYPE MODEL	ITERATIVE MODEL
Understanding requirements/Requirement specification	Beginneing	Beginneing	Not well understood at begineeing	Well understood at begineeing
Duration	long	long	long	Not very long
cost	low	expensive	high	medium
Cost control	yes	Almost yes	no	no
Documentation and training required	vital	yes	weak	yes
Guarentee of success	less	high	good	high
Initial product feel	no	no	yes	no
Client satisfaction	low	------	high	high
Risk involvement	High	Very low	low	medium
User invovement	low	high	high	high
flexibility	Rigid	flexible	flexible	Less flexible
Simplicity	Simple	Intermedite	Simple	Intermediate
Integrity and security	Least	High	Weak	Robust
Maintenance	Least glamorous	Typical routine maintenance	Routine maintenance	Promoted maintainability
Overlapping phases	No overlapping	Yes	Yes	No
Implementation	Easy	Complex	Easy	Easy

Different Terms And Concepts

- **SRS:- SRS** is a software requirement specification **SRS** is used to implement the user requirements in project.
- **Software Architecture:-**The requirements specification mentioned in SRS document is transformed into the structural form. This structural form is software arch.
- **Testing:-** ***d*** testing is system testing performed by development team.
- **Prototype:-**A prototype is like a toy of the actual system which have low reliability, less functionalities and less performance.
- **Meta Model:-**A model includes features of the all the model, so it can be viewed as a meta model.

- **Unit testing:-**In this testing, each module tested separately from other module, then debugging and documented.

- **System testing:-**The function of system testing is to ensure that the development system conforms to its requirements specify in SRS document.
- **Acceptance testing:-**Acceptance testing is system testing performed by the customer himself after the product delivery to determine whether to accept the delivered product or to reject it.
- **Integration testing:-**In Integration testing, modules are integrated one by one in step and after each addition of one module, the resulting system is to verify that after the integration of new module the system is working correctly or not.
- **Phase containment of errors:-**The principle of detecting error as close to its point of introduction as possible is called phase containment of errors.

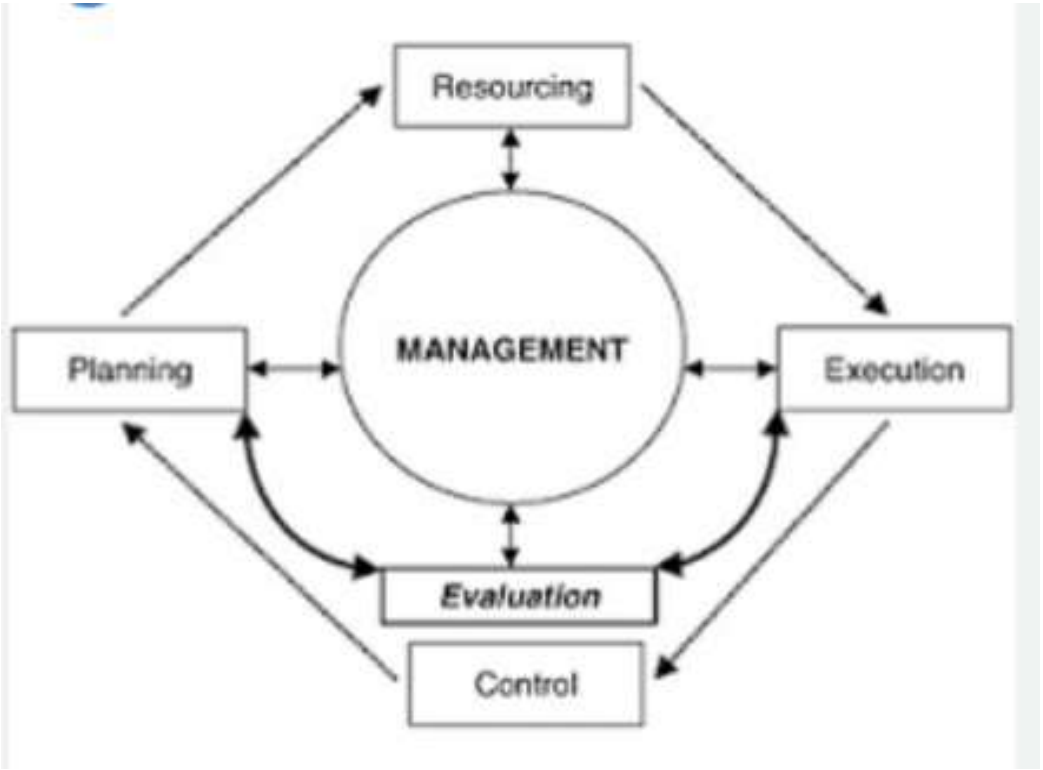
ThankYou

Software engineering

Software planning

- Project planning
- Risks identification and abatement
- Selecting development process model
- Project scheduling
- Project complexity

- Shafting plan
- Quality assurance and configuration management plan



Responsibilities of software project manager

- **Agreeing project objectives**
- **Representing the client's interest**
- **Providing independent advice on the management of project**
- **Organizing the various professional people working on the project**
- **Risk assessment**
- **Making sure that all the aims of the project are meet**
- **Using the latest approach to keep track of people and progress**
- **Recruiting specialist and subcontractors**

Project Size Estimation

- **By using the size oriented metrics**

Line of code

Effort

Cost

Pages of documentation

Defects

People

Line of code (IOC)

Loc is the simplest among all metric available to estimate project size .This metric measure the size of a project by counting the number of source instructions in the developed program while counting the number of source instructions,line used for commeting the code and leader lines are ignored.

Advantages of using LOC as key Measure

1. LOC is very simple to use, because it provides a exact numeric entity
2. Line of code can be easily counted and hence LOC is easy to measure
3. A large body of literature and data is already available on concept of LOC.

Short coming of using LOC as key measure

1. When LOC is required in planning Phase of the development,then it is very tricky for the planner to estimate LOC at beginning of projects
2. LOC measure are programming language dependent.
3. LOC only focus on the coding part of whole development.
4. Using library routine generally decrease lines of code .So if a new programmer, not having knowledge required programming language may not use the library routines and hence may increase thevLOC value Aversely

- ***Function oriented metrics***

The idea behind this approach is that the size of software product is directly dependent on the number and types of different functions it performs.

1. **Number of user inputs**
2. **Number of outputs**
3. **Number of files**
4. **Number of user inquiries**
5. **Number of interfaces**

Advantages of using FP as keyMeasure

- It overcome all the problem that we are facing while using LOC as key measure
- Using function point metrics the size of software can be directly measured from the initial problem specification.
- The function point metric is language independent

Disadvantage of using function points as key measured

- Function point metrics does not take into account the algorithmic complexity of a software.
- Function point metric is dependent on the engineering working on the project .

Example

An example of LOC based technic

- Cad software will get 2 or 3 dimensional geometric data from the engineer.
- A system interface has to be provided so that engineer can interact with this system
- All geometric data input by the user will be maintained in the database of CAD software

An example of function point based technique

- Let's now find the effort & cost for the same software system CAD using function point approach

Software Parameters	Estimated Count	Weight	Function Point Count
Number of inputs	24	4	96
Number of outputs	16	5	80
Number of inquiries	24	5	120
Number of files	5	10	50
Number of interfaces	2	7	14
Total count			360

Project-estimation Technique

- Empirical estimation-This technique is based on making a guess of project parameter using the part working experience
- Heuristic technique- In this technique the project parameter are estimated using some mathematical expression eg. cocoa models

- Analytical estimation technique - These type of technique at based on concept of making assumption.

Cocomo-A Cost Estimating model

- COCOMO (constructive cost estimation model)

It is use the Heuristics technique. It was proposed by Boehm in 1981.

Boehm software cost estimation should be done

Through three stages..

- Basic COCOMO
- Intermediate COCOMO
- Complete COCOMO

- According to the boehm software development projects can be classified into three categories based on their complexity organic, semidetached and embedded.

- Organic - If the project being developed is a well understood application
- Semidetached- If the team is a mixture is of experienced and non-experienced people.The member have experienced but unfamiliar with the aspect of the system being developed
- Embedded-Projects are considered to be of embedded type if organization has little experience and have very tight constraints from the hardware , software and personal and also the project required hardware interaction

Basic COCOMO Model

This is the most common approach for estimating effort the simple variable approach in algorithmic cost model is the basic for COCOMO Model .

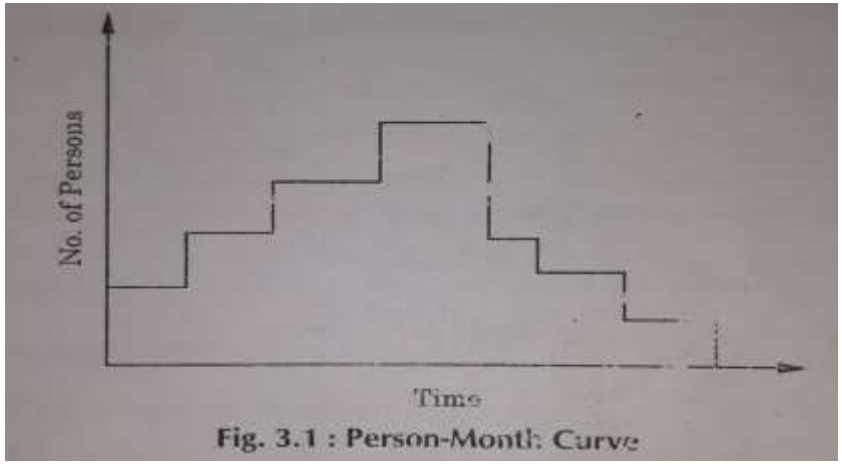


Fig. 3.1 : Person-Month Curve

Intermediate COCOMO Model

It is observed that the estimation based on single factors such as size is not much accurate

- Product- It includes the complexity of products, reliability requirement of the product etc.
- Computer- It includes the execution speed required, storage space required etc
- Personnel-It includes factors such as experience level of personal programming capability etc.
- Development Environment- The factor are such as development facilities , software tools used etc.

Complete COCOMO Model

The main problem with the previous Models were that they Considered a software as a single

Homogeneous entity.

- Interface
- Communication component
- Database handling component

Halstead's Software Science

Halstead complexity metrics were developed by ave Maurice Halstead as mean of determining a Q's quantitative measure of complexity directly from operator and operand in the module to measures a program modules complexity directly from source code.

- Overall program length
- Vocabulary size
- Program volume
- Program level
- Effort to implement E

Recognizing operator and operands

Some of guidelines regarding operators and operands are given below

1. Arithmetic, logic, assignment,operators
2. ( ) Is consider as single operators
3. Go to consider as single operator
4. (Terminator)is consider as single operator
5. A function name in function call statement is consider as operator.

- Program lenght (N)

It is sum of the total number of operators and operands in the program

N=N1+N

- Vocabulary Size (n) the vocabulary size (n) is the sum of number of unique operators and operands n=n1+n2
- Program volume (v)
- Difficult level (D)
- Program level (l)
- Efforts implement (E)
- Estimated program length (N)
- Potential Volume
- Estimated program level

Ishan's Software engineering


1. ARCHITECTURE AND DESIGN
2. APPLICATION DEVELOPMENT
3. APPLICATION CUSTOMIZATION
4. QUALITY ASSURANCE AND TESTING
5. MAINTENANCE
6. TRAINING AND SUPPORT
SOFTWARE ENGINEERING

Requirement Analysis And Specification

Software Requirement Analysis also called Requirement Engineering, is process of determining engineering, is process of determining user expectation.

Analysis is an important aspect of project management.

REQUIREMENT GATHERING

Requirement Gathering is process of gathering the information about the requirements of the user or customer which customer wants in his system or project to be developed.

By Requirement Gathering exact objective of system or product is known and what exactly needs to be accomplished and getting information about fitness of the system into the needs of business.

Many technique are available for gathering requirement

- **Interview**
- **Closed Interview**
- **Open End Interview**
- **Brainstorming**
- **Questionnaires**
- **Prototyping**
- **Use case**
- **Group Interview**

INTERVIEW

It is most commonly used method in requirement gathering. In an interview, Analyst discuss the desired product with different stockholders and develops an understanding of their requirements.

CLOSED INTERVIEW

In this interview the requirements, we have to prepare some predefined questions try to get the answer for these question for the customers.

OPEN END INTERVIEW

In this interview, we do not need to prepare any predefined question, and the information from customer in open discussion.

BRAINSTORMING

The brainstorming involves both idea generation and reduction. After all the ideas generated, the participants prioritize the ones they think are the best for this solution and that idea are used for the initial requirements.

QUESTIONNAIRES

Questionnaires are much more informal and they are good tools to gather requirements from stakeholders in remote locations or those who will have only minor input into the overall requirements.

PROTOTYPING

Prototyping is relatively modern technique for gathering requirements. In this approach, you gather preliminary requirements that you use to build an initial version of the solution.

USE CASE

Use case are basically stories that describes how discrete processes work. The stories include people and describe how the solution works from a user perspective.

GROUP INTERVIEW

Group Interview are similar to the one-on-one interview, except that more than one person is being interviewed usually two to four. Group Interviews require more preparations.

Requirement Analysis

- ✓ **To detect and resolve conflicts that arise due to unclear and unspecified requirement.**
- ✓ **To understand the problem for which software is to be developed.**
- ✓ **To check the level of the necessity of the requirement.**
- ✓ **To check if we can test the requirement after it has been implemented.**

Formal Specification Techniques

There are mainly two basis used under formal specification techniques:-

1. Relational Notations

2. State Oriented Notation

RELATIONAL NOTATIONS

Relational Notations are based on the concept of entities and attributes. Entities are named elements in a system. The name are chosen to denote the nature of element. Attribute are specified by applying function and relations to the named entities and data flow between entities.

Relational Notations

- Implicit Equations
- Recurrence Relations
- Algebraic Axioms
- Regular Expressions

○ IMPLICIT EQUATIONS

Implicit Equations state the properties of solution without starting a solution method.

For Example. Specified matrix inversion as

$$M \times M' = 1 \pm E$$

○ RECURRENCE RELATIONS

Recurrence Relations have two parts

(a) initial or basic part

(b) One or more recursive parts

○ ALGEBRAIC AXIOMS

Mathematical system are defined by AXIOMS.The AXIOMS specify fundamental properties of a system and provide a basis for deriving additional properties that are impliedby AXIOMS.These additional properties called theorems.

○ REGULAR EXPRESSION

Regular expression are the language generators. They are used to specify the syntactic structure of symbol strings. A set of symbol strings when specified by a regular expression defines a formal language.

■ STATE-ORIENTED NOTATIONS

State of a system at any particular time gives the statusof the various system entities at that time. If the current state of the system is known and current stimuli to the system is available then we can derive the next stage of state. Various state oriented notations used are :-

(1) Decision Tables

(2) Event Tables

(3) Transition Tables

(4) Finite State Mechanisms

DECISION TABLES

1. **Decision rules**
2. **Condition stub**
3. **Action stub**
4. **Condition entries**
5. **Action entries**

Decision Table	Decision Rules			
	1	2	3	4
Credit is within the limit	Yes	No	No	No
Pay is favourable	don't care	Yes	No	No
Any other clearance	don't care	don't care	Yes	No
Approve Order	Perform action	Perform action	Perform action	
Reject Order				Perform action

Fig. 4.4 : Limited entry decision table

○ Event Tables

We are representing the relation between condition and activity.

If particular condition satisfy then a particular activity should occur.

Conditions	Event Occurring				
	Event 14	Event 17	Event 20	Event x	.....
Condition 1	Activity 1	Activity 2	–	–	.....
Condition 2	–	Activity 1	–	–	.....
Condition 3	–	Activity 1, Activity 3	–	–	
Condition 4	:	:	:	:	.....

○ Transition Table

Transition tables are used to specify the changes in the state of a system when a particular condition occur or input is given .

That can be represented as :-

Current State	Current Input		
	I_{C1}	I_{C2}	I_{C2} .....
S_{C1}	S_{N1}	S_{N2}	S_{N3} .....
S_{C2}	S_{N2}	S_{N1}	S_{N5} .....
S_{C3}	:	:	:
:	:	:	:
		Next State	

Finite State Mechanisms

Finite State Mechanisms provide a powerful method for functional specification of software

System by combining the notations,Data flow diagrams, regular expression and and transition tables.

Software Requirement Specification(SRS)

During the software requirement and analysis phase mainly two activities occur Requirement Analysis, Requirement Specification. In Requirement Analysis all the requirements of user are analysed and identified. While during the requirement specification phase, the requirements identified are documented in the form of a document called Software Requirement specification.

ADVANTAGE OF SRS DOCUMENT

An srs document act as a legal contract between developers and customers. Through SRS document client can clearly specify what he expects from the developer and what are the various features required. Also developer can understand the capabilities required in the software.

Components Of SRS

① Functional Requirements
② Performance Requirements
③ Design Constraints
④ External Interface Requirements

◆ FUNCTIONAL REQUIREMENTS

Functional Requirement specify the relationship between the input and output of the system. It clearly specify the output that has to be produced from the given inputs. Functional Requirement gives the detailed description of all the data input, their source, validity of input and output required. It also specify all the operations that has to be performed on the input data to obtain the specified output.

◆ PERFORMANCE REQUIREMENTS

There are mainly two types of performance requirements :-

- ✓ **Static Performance Requirements :- are those which are not related to the execution characteristics of the system. For Example :- Number of files in the system.**
- ✓ **Dynamic Performance Requirements:- are related to the execution characteristics of the system. It usually included reponse time and throughput constraints of system.**

◆ DESIGN CONSTRAINTS

It specify the constaints that must be followed during design of project. It includes the standard to be followed like the format of reports and procedures etc.

◆ EXTERNAL INTERFACE REQUIREMENTS

In these requirements all the possible interactions of the software with people's hardware and software should be clearly specified.

Characteristics of a Good SRS

- ✓ **Correctness**
- ✓ **Completeness**
- ✓ **Verifiable**
- ✓ **Consistent**
- ✓ **Unambiguous**
- ✓ **Modifiable**
- ✓ **Traceable**

✓ CORRECTNESS

Correctness of SRS checks that every requirements included in the SRS represents something required in the system. Correctness of SRS ensures that what is specified is done correctly

✓ COMPLETENESS

An SRS is said to be complete if everything the software is supposed to do is specified in the SRS. Completeness of the SRS is most difficult property to achieve.

✓ VERIFIABLE

Verifiability of the SRS implies that every requirements specified in it must be verifiable. There should be some process to check whether the final software meets all the specified requirements or not.

✓ CONSISTENT

An SRS is said to be consistent if there is no requirements specified in SRS that conflicts with another requirement specified in SRS. For Example suppose an SRS specifies that a requirements .

✓ UNAMBIGUOUS

An SRS is said to be unambiguous if all the requirements specified in it have one and only one meaning. But if we are using the natural language for specifying the requirements then there is large chances of it to be ambiguous.

✓ MODIFIABLE

IT has been seen that, usually with the changing needs of client the requirements specified in the SRS also needs modification. Hence an SRS should be easy to modify.

✓ TRACEABLE

An SRS is said to be traceable if the origin of each of its requirement is clear. Traceability of requirements helps in getting the better understanding of system and facilitates the referencing of each requirements in future development.

However it should be noted that, completeness is the one of most important characteristics of an SRS document.

REFERENCE

ISHAN PUBLICATIONS

BY :-

(Sandeep Kumar)

Simranpreet Kaur(E- Max Institute. Of Eng & Tech)

Website:- www.ishanpublication.com

SUBJECT:

SOFTWARE DESIGN

CHAPTER-5

SOFTWARE DESIGN IMPLEMENTATION

❖ INTRODUCTION

- Software design:-Software design is the process by which an agent creates a specification of a software artifact, intended to accomplish goals, using a set of primitive components and subject to constraints.
- Implementation :-A product **software implementation** method is a systematically structured approach to effectively integrate a **software** based service or component into the workflow of an organizational structure or an individual end-user.

In other words we can say that **Software design and implementation** activities. are invariably inter-leaved. –**Software design** is a creative activity in which you. identify **software** components and their relationships, based on a customer's requirements. – **Implementation** is the process of realizing the **design** as a program.

Design framework

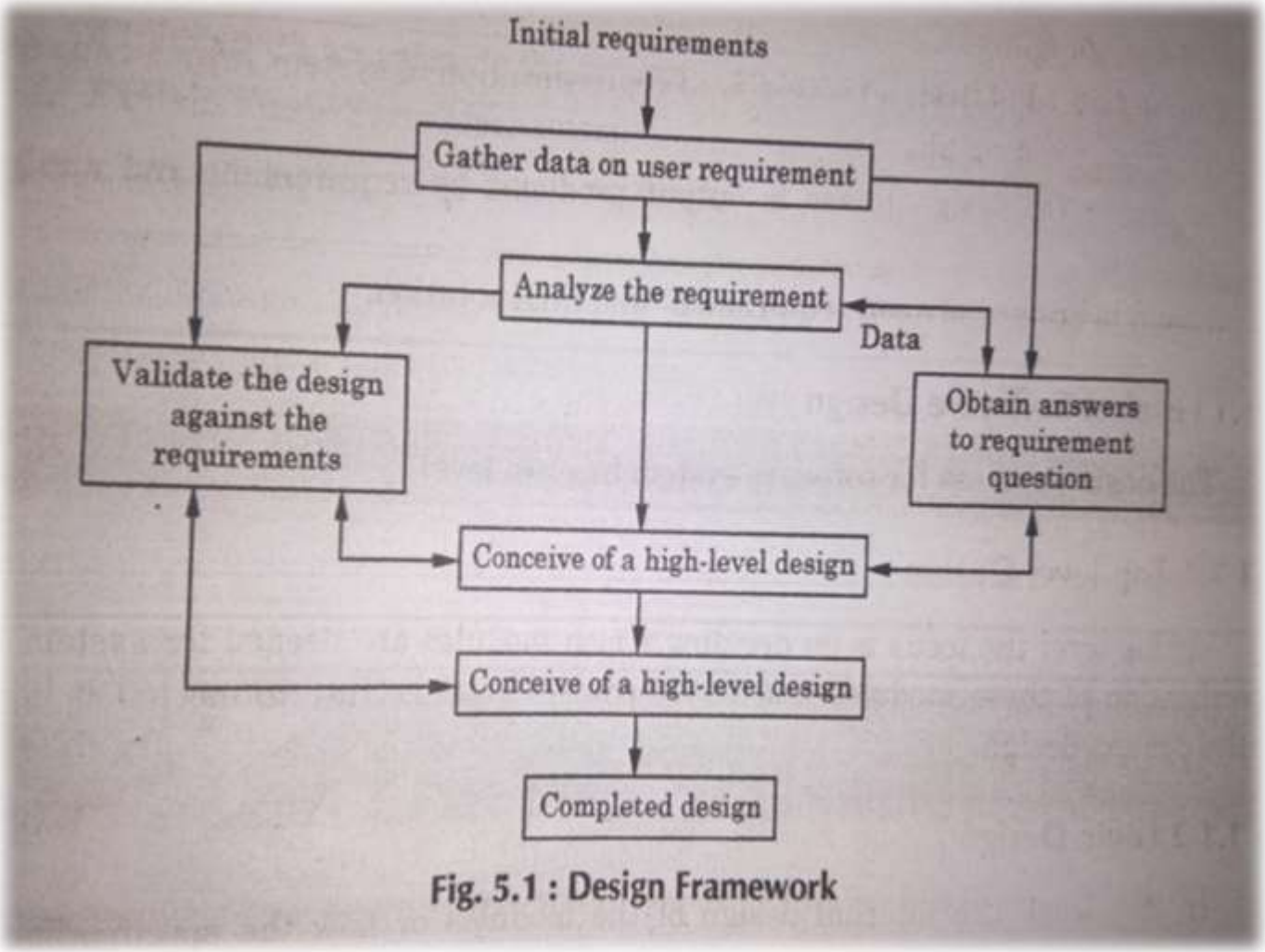


Fig. 5.1 : Design Framework

Characteristics and features of good software design

i. **correct**: design of the system is correct if system built according to satisfies the requirement of that system. The basic goal of design phases to produce correct design.

ii. **Complete:** a design should be complete by completeness of design means that it should include all the detail into design

iii. **Efficient:** a design is said to be efficient if resources are properly used by the system. if the efficiency required is more, then cost of system is also increased.resourses that are consider under efficiency are processing time and memory.

iv. **Simplicity:** simplicity is another most important quality criteria for the software system. we know that maintenance

v. **Performance:** software design performs it's tasks within a user acceptable time. The software design does not consume too much memory.

vi. **Complexity:** the complexity of design is properly handled , then effort for design can reduced . It also reduced the scope of introduction of new error during design.

vii. **Design notation:** a design should be represented using a notation that effectively communication its meaning.

viii. **Repeatable:** a design should be derived using a repeatable method that is driven by information obtained during software requirement analysis .

COHESION AND COUPLING

A good software design means the clean decomposition of problems into modules in also the next arrangements of this module. the goal of software design is to minimize the complexity of interconnection among modules . these objectives can be achieved by using the concept of coupling and cohesion . a coupling and cohesion were first describe by storens.

COUPLING

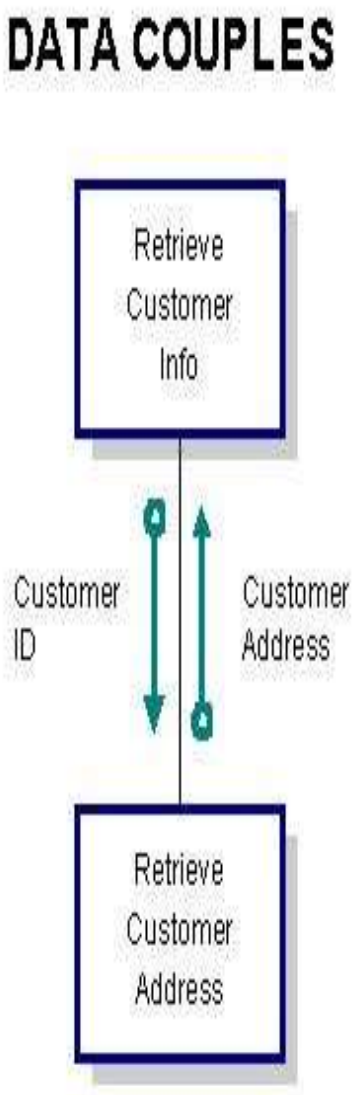
The degree of interdependence between two modules is called **coupling .**

Types of coupling:-

Data coupling	best
Stamp coupling	
Control coupling	
Common coupling	
Content coupling	worst

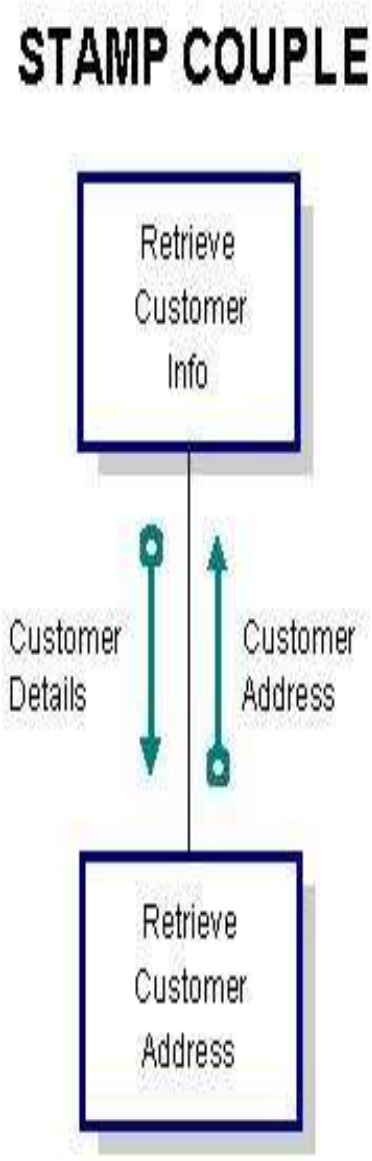
DATA COUPLING

Data coupling occurs when modules share data through, for example, parameters. Each datum is an elementary piece, and these are the only data shared (**e.g.**, passing retrieve customer info to a its address as shown in picture.)



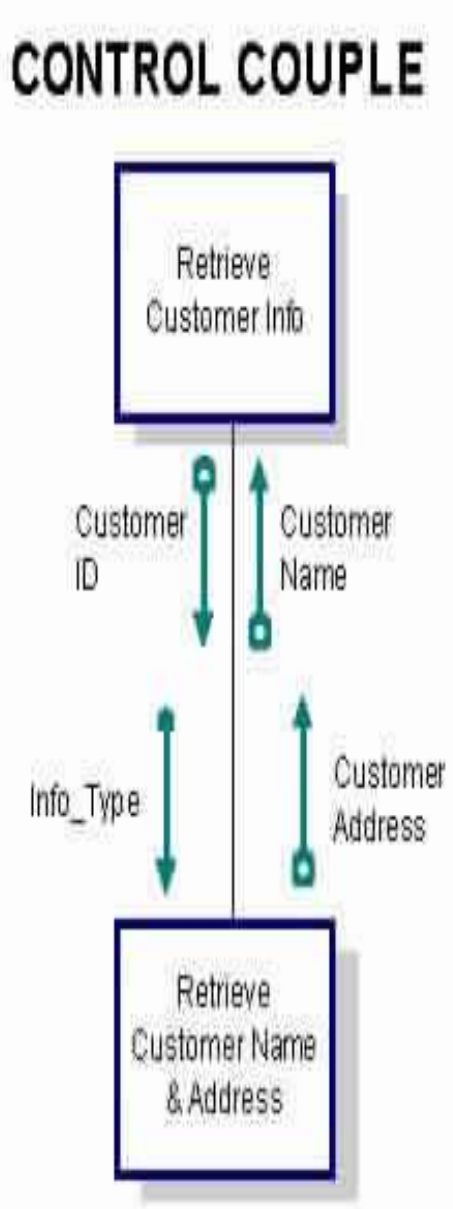
STAMP COUPLING

Stamp coupling occurs between modules when **data** are passed by parameters using a **data** structure containing fields which may or may not be used.



CONTROL COUPLING

The dependence of a software component on data not exclusively under the **control** of that software component.” ... An example of **control coupling is** a real-time software executive that initiates execution of a software component depending upon external parameters or influences.

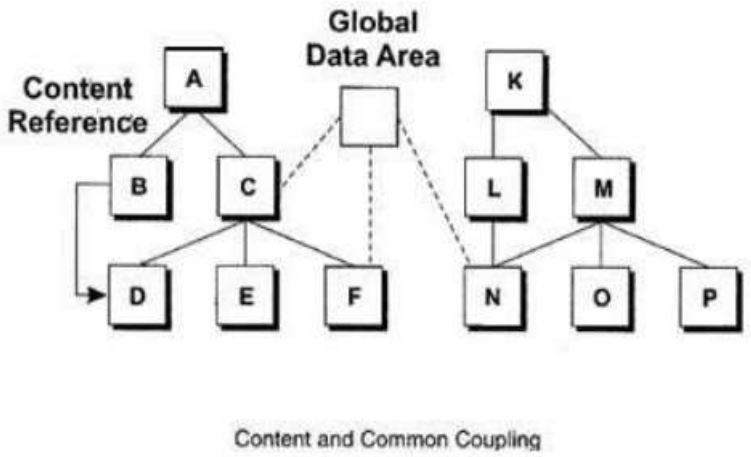


COMMON COUPLING

common coupling is a form of coupling in which multiple modules share the same global data.

Notes By Adil Aslam

Example of Common Coupling



Content and Common Coupling

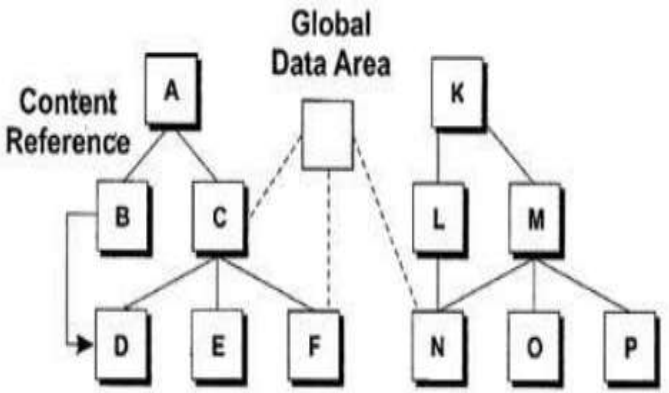
My Email Address : adilaslam5959@gmail.com

CONTENT COUPLING

Content coupling is when one module modifies or relies on the internal workings of another module (e.g., accessing local data of another module). When we change the way the second module produces data (location, type, timing) will lead to changing the dependent module

Notes By Adil Aslam

Example of Common Coupling



Content and Common Coupling

My Email Address : adilaslam5959@gmail.com

COHESION

Cohesion is measure of strength of functional relatedness of elements within a module.

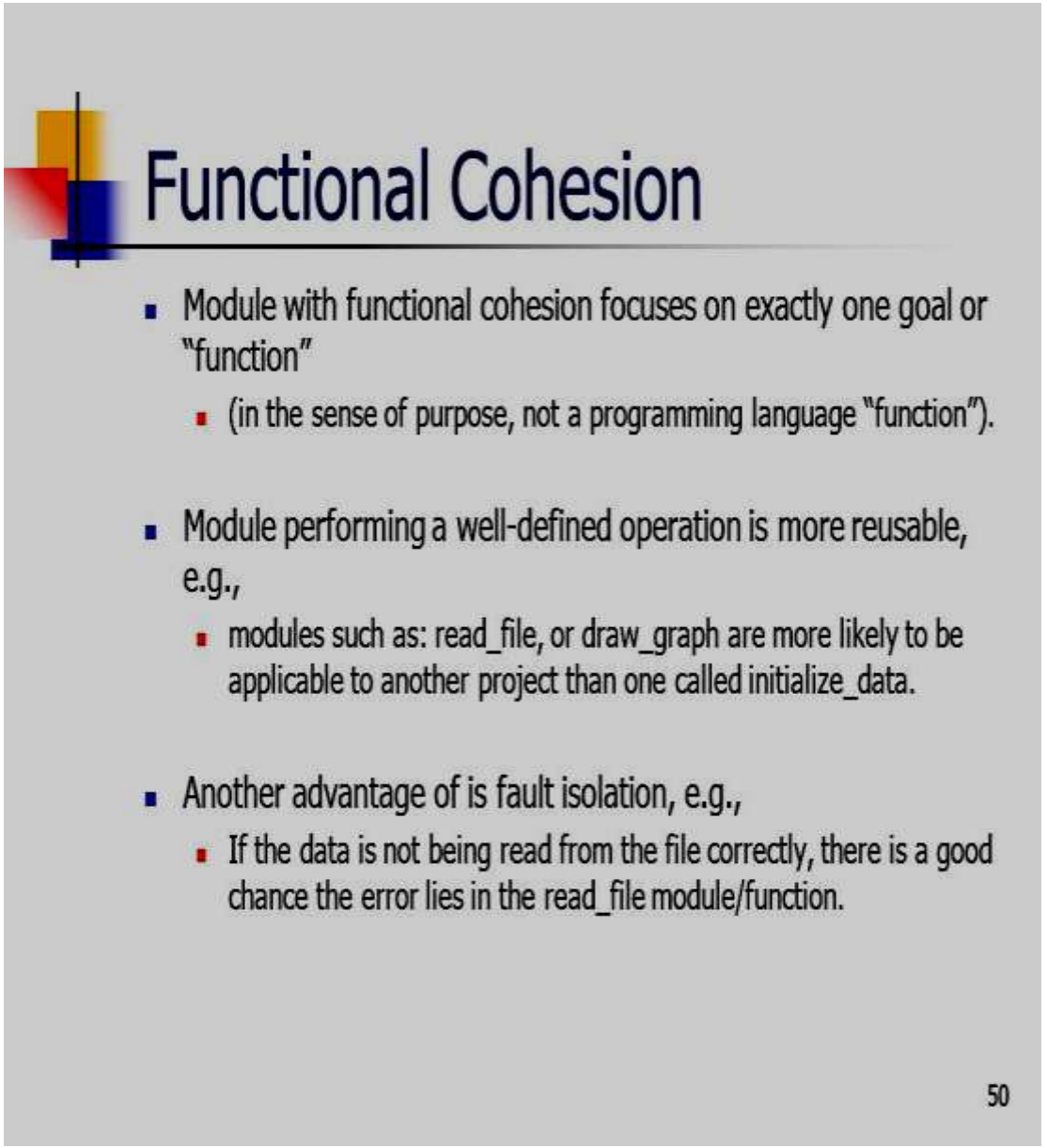
- Cohesion in module should be maximized.
- Everything in module should be related to one another, focus on take.
- Strong cohesion will reduced relation between modules minimize coupling.

TYPES OF COHESION

Functional cohesion	best
Sequential cohesion	
Communicational cohesion	
Procedural cohesion	
Temporal cohesion	
Logical cohesion	
Coincidental cohesion	worst

FUNCTIONAL COHESION

Functional cohesion. A functionally cohesive module is one in which all of the elements contribute to a single, well-defined task. Object-oriented languages tend to support this level of **cohesion** better than earlier languages do.



SEQUENTIAL COHESION

- **Element are involved in activities such that output data from one activity become input data to next .**
- **Usually has good coupling and is easily maintained.**
- **Not so ready readily reusable because activities that will not in general be useful together.**

COMMUNICATIONAL COHESION

- **Element contribute to activities that use the same input or output data.**
- **Possible links that causes activities to affect each other.**
- **Communicational cohesion is not a flexible**

PROCEDURAL COHESION

- **Element are related only by sequence , otherwise activities are unrelated.**
- **Commonly found at top of hierarchy , such as the main program module.**

TEMPORAL COHESION

- **element are involved in activities that are related in time .**
- **Commonly found in initialization and termination module.**

LOGICAL COHESION

- **element contribute to activities of some logical class.**
- **Usually have control coupling ,since one of the activity will be selected.**

COINCIDENTAL COHESION

- **element contribute to activities with meaningful relationship to one another.**

Difference between coupling and cohesion is given below

COUPLING

i. Cohesion is the indication of relationship within module.

ii. Cohesion shows the module's relative functional strength.

iii. Cohesion is the degree to which module focuses on single ting.

iv. Cohesion is intra-module concept

v. Cohesion is kind of natural extension of data hiding . For example , class having all members visible with a package having default visibility.

vi. While designing you should strive for high cohesion i.e. module focus on single task with little interaction with other modules of the system.

COHESION

i. Coupling is the indication of relationships between modules

ii. Coupling shows the relative independence among the module.

iii. Coupling is degree to which a module is connected to other module.

iv. Coupling is inter-module concept.

v. Making private fields , private method and non public classes provide loose coupling.

vi. While designing you should strive for low coupling i.e. dependency between module should be less.

SOFTWARE DESIGN APPROCHES

There are two main approaches to software analysis and design, namely, Function-Oriented Approach and Object-Oriented Approach. Both these approaches are covered in some detail in subsequent chapters of this book. The basic concepts and an overview of both these approaches are given in this chapter to serve as an introduction to subsequent chapters. It is meant to make it easier for the readers to draw similarity and distinction between these two approaches when they go through the details in subsequent chapters.

FUNCTION ORIENTED DESIGN

- Objectives
- To explain how a software design may be represented as a set of functions which share system state information.
- To introduce notations which may be used to represent a function oriented design.
- To develop an example which illustrates the process of function oriented design.
- To compare, using a common example, sequential and concurrent function-oriented design and object-oriented design. Contents 15.1 Data-flow design 15.2 Structural decomposition 15.3 Detailed design 15.4 A comparison of design strategies

ABOUT FUNCTION ORIENTED DESIGN APPROACH:-

- In function-oriented design, the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.
- Function oriented design inherits some properties of structured design where divide and conquer methodology is used.
- This design mechanism divides the whole system into smaller functions, which provides means of abstraction by concealing the information and their operation.. These functional modules can share information among themselves by means of information passing and using information available globally.
- Another characteristic of functions is that when a program calls a function, the function changes the state of the program, which sometimes is not acceptable by other modules. Function oriented design works well where the system state does not matter and program/functions work on input rather than on a state.

❖Design Process

- The whole system is seen as how data flows in the system by means of data flow diagram.

- DFD depicts how functions changes data and state of entire system.

- The entire system is logically broken down into smaller units known as functions on the basis of their operation in the system.

- Each function is then described at large.

Function-oriented design

- Practised informally since programming began
- Thousands of systems have been developed using this approach
- Supported directly by most programming languages
- Most design methods are functional in their approach
- CASE tools are available for design support

ABOUT OBJECT ORIENTED DESIGN APPROACH:-

- Object Oriented Design
- Object oriented design works around the entities and their characteristics instead of functions involved in the software system. This design strategies focuses on entities and its characteristics. The whole concept of software solution revolves around the engaged entities.

Let us see the important concepts of Object Oriented Design:

- **Objects -** All entities involved in the solution design are known as objects. For example, person, banks, company and customers are treated as objects. Every entity has some attributes associated to it and has some methods to perform on the attributes.
- **Classes -** A class is a generalized description of an object. An object is an instance of a class. Class defines all the attributes, which an object can have and methods, which defines the functionality of the object.
- In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.
- **Encapsulation -** In OOD, the attributes (data variables) and methods (operation on the data) are bundled together is called encapsulation. Encapsulation not only bundles important information of an object together, but also restricts access of the data and methods from the outside world. This is called information hiding.
- **Inheritance -** OOD allows similar classes to stack up in hierarchical manner where the lower or sub-classes can import, implement and re-use allowed variables and methods from their immediate super classes. This property of OOD is known as inheritance. This makes it easier to define specific class and to create generalized classes from specific ones.
- **Polymorphism -** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism, which allows a single interface performing tasks for different types. Depending upon how the function is invoked, respective portion of the code gets executed.

Design Process

- Software design process can be perceived as series of well-defined steps. Though it varies according to design approach (function oriented or object oriented, yet It may have the following steps involved:
- A solution design is created from requirement or previous used system and/or system sequence diagram.
- Objects are identified and grouped into classes on behalf of similarity in attribute characteristics.
- Class hierarchy and relation among them is defined.
- Application framework is defined.

STRUCTURED CODING TECHNIQUES

- The structured coding is to make the control flow through a computer program so that the execution sequence follows the same sequence in which code is written . The basic programming constructs say that flow of execution should be from top-bottom. Linear flow of control should be achieved .
- the following constructs can be used to achieve this linear flow:-

i. Single Entry , Single Exit Construct.
ii. Efficiency Consideration.
iii. Violation of construct single entry, single exit.
iv. Data Encapsulation.
v. Go to statement.
vi. Recursion.

SINGLE ENTRY SINGLE EXIT CONSTRUCT

Understanding these issues and designing the **software** in **consideration** of them will greatly increase the odds that an IBM project will be **efficient** and successful. Process. The model's process is the sequence of events that occur during a run.

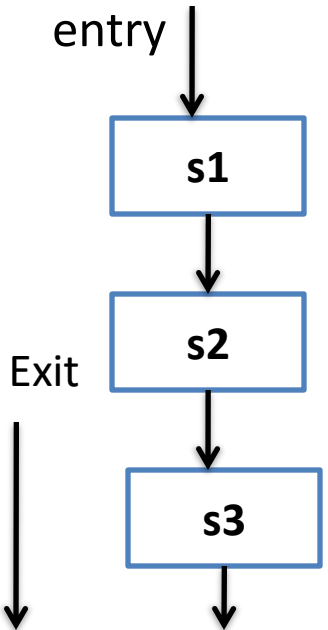
The three basic single entry , single exit constructs are :

- Sequencing
- Selection
- Iteration

It is observed that these constructs are sufficient for describing decontrol flow of every algorithm.these constructs are widely used in programming languages .single entry , single exit construct allows the nesting of these basic construct within one another.

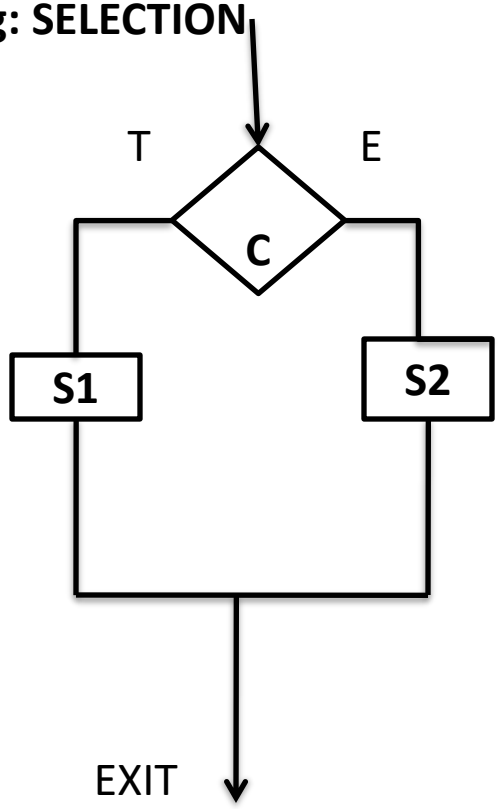
BASIC CONSTRUCTS

- Sequencing: this constructs says that the control flow should be from top to bottom i.e. statements should be executed in an order as they appear in the program.



- **<u>Selection</u>**: this constructs says that flow of control can be controlled based on some conditions . Hence either the statement **s1** or statement **s2** will execute based on the result of condition C . If the condition C is true then statement **s1** will be executed , else the statement **s2** will be executed.
- **<u>Iteration:</u>** the purpose of this construct is to repeat some portion of the program on the basis of some condition .

Fig: SELECTION



EFFICIENCY CONSIDERATION

The problem with the single entry ,single exit construct is that they result inefficient use of memory space and execution time .let us take example ,look at the following statement in C.

```
If  ( (x!=0) &&   (x%2=-0))          then
                                     {
                                     Statement- block;
                                     }
```

CODING STYLE

- Computer **Programming/Coding Style**. ... "**Programming style**" primarily refers to low-level **conventions**, such as formatting or choice of language constructs (such as goto or return), but can also refer to large-scale **code** structure or even overall**design** in **software engineering**.
- **What is the coding standard?**
- **Coding standards** are a set of guidelines, best practices, **programming** styles and conventions that developers adhere to when writing source **code** for a project.

What is coding guidelines?

Coding conventions are a set of **guidelines** for a specific programming language that recommend programming style, practices, and methods for each aspect of a program written in that language. ... **Coding** conventions are only applicable to the human maintainers and peer reviewers of a software project.

DOCUMENTATION

- **Software documentation** is written text or illustration that accompanies computer**software** or is embedded in the source code. It either explains how it operates or how to use it, and may mean different things to people in different roles.**Documentation** is an important part of **software engineering**.

What is a user documentation?

Generally, **documentation** is divided into two main areas. Process Documents guide the development, testing, maintenance and improvement of systems. They are used by managers, engineers, testers, and marketing professionals. These documents use technical terms and industry specific jargon.

What is system documentation in software engineering?

Technical **documentation in software engineering** is the umbrella term that encompasses all written documents and materials dealing with a **software** product's development and use.

All **software** development products, whether created by a small team or a large corporation, require some related **documentation**

REFERENCES

- WWW.SAFARIBOOKS.COM
- WWW.GOOGLE.COM
- WWW.QUORA.COM
- BOOK:**ISHAN'S** SOFTWARE ENGG.

SOFTWARE ENGINEERING

REFERNCE BY :-

SIMRANPREET KAUR

SANDEEP KUMAR

SOFTWARE TESTING

Testing is a necessary stage in software life cycle. It gives the programmer and user some sense of correctness. With effective testing technique, software is more easily debugged, less likely to break, ”more”correct” and better.

Defination :- testing is a process of executing a program with intent of finding error.

Concept of software testing

Need of Software testing:- Software testing help to deliver quality software products that satisfy users requirement, need and expectation.

but testing poorly, defects are found during operation and it may cause mission failure. It result in high maintenance cost and user dissatisfaction and impact on operational performance and reliability.

Software testing objectives

- To find or prevent defect .
- Determine user accept ability.
- Testing planning should be done early.

Principle of software testing

- Testing should be based on user requirement.
- testing time and resources are limited.
- Use effective resource to test.
- Testing should begin at module.
- Document test case and test result.

Verification and Validation

- **VERIFICATION:-** is a process of evaluating product of a development to phase to find out whether they specified the requirement. Following activities are involved in verification that is reviews, meetings, inspection. Execution of code is not come under verification. Verification process explains whether the output are according to input or not. Cost of error in verification is less then error found in validation. It is basically manually checking the documents and files like requirement specification.

Validation

- Validation process is evaluating software at the end of the development process to determine whether software needs the specification were correct in first place. Following activities are involved in validation testing that is white box testing and black box testing. Execution of code is not comes under validation. Validation process describe whether the software is accepted by the user or not. Cost of error caught in validation is more than error found in verification. It is basically checking of developed program based on the requirement specification documents and files.

Unit Testing

Unit testing is a process of testing a module and running it in isolation from the of the software. Product by using prepared test case and comparing the actual result which specified functionalities.

I. The size of single module is small enough that we can locate and error easily.

II. Confusing interactions of multiple error in widely different part of the software are eliminated.

- Module interface test
- Local data structure
- Boundary condition
- Independent path
- Error handling

Unit test procedure

- Scaffolding is any code that we write, not as a part of application but simply to support the process of unit and integrating testing.
- Scaffolding comes into to forms:-

✓Driver:- it is testing scaffolding that calls the module being tested. A drive module should contain non local data structure access by module under test wi5th appropriate parameter value for testing.

✓Stub:- it test scaffolding written to replace type not function used by module. Stub do minimal data manipulation, prints verification of entry and returns control to module under going testing.

Limitations of Unit Testing

- Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution in path every software application.
- It is also essential to implement a sustainable process for ensuring to that test case failure are reviewed daily and address immediately. Otherwise, it increase false positive and reducing effectiveness of test case.

Integration testing

Integration testing is phase in software testing in which individual software module are combined and tested as a group. It occurs after unit testing and before validation testing.

- **Purpose** :- the primary purpose of integration testing is to check whether the different modules of a program interface with each other properly some different type of integration testing are described below :-

- **Big bang approach**:- big bang testing, is most obvious approach to integration testing. In this approach to all modules of the system are simply linked together and tested. This technique can be used only for small system.
- **Bottom up integration testing**:- it is an approach to integrated testing where lowest level components are tested first, then used to facilitate the testing of high level components. The process is repeated until the component at the top of the hierarchy is tested. This method also help to determine the level of software developed and makes it easier to report testing progress in form of percentage.

- **Top Down Testing**:- it is an approach to integration testing where top level units are tested first and lower level units are tested step by step after that.
- **Sandwich Testing**:- it is an approach to combine top down testing with bottom up testing. The main advantages of bottom up approach is that bugs are more easily found with top down, it is easier to find a missing branch link.

BLACK BOX TESTING

Black box testing design the system as black box. Black box testing method which is used to test the software without knowing their internal structure of code of program. It is also known as functional testing. Black box testing was develop as method of analyzing client requirement specification and high level design strategies

Advantages of black box testing

- Simplicity: faciliates testing of high level design and complex applications.
- Conserves resources : tester focus on software functionality.
- Test case: focusing on software functionality to faciliate quick test case development.
- Provide flexiblity: specific programming knowledge is not required.

Disadvantages of black box testing

- Graphical User Interface (GUI) may damaged test scripts.
- Testing only covers application functions.
- This method of attempts to errors in following categories:-
  - Interface error
  - Incorrect or missing function
  - Error in data structure

Black Box Testing Techniques

- Equivalence class partitioning.
- Boundary value analysis (BVA).
- Cause effect graphs.
- Comparison testing

WHITE BOX TESTING

White box test design allow one to peek inside the box and focuses specifically on using internal knowledge of the software to guide the selection of the test data. White box testing, test cases are selected on basis of examination of the code.

White box testing is a software testing approach that examine the program structure and derive test data from the program logic. White box testing required the intimate knowledge of program internal.

Advantages of white box testing

- Force test developer to reason carefully about implementation .
- Approximates the partitioning done by execution equivalence.
- Reveals errors in hidden code.
- Optimizations.

Disadvantages of white box testing

- Expensive
- Miss cases omitted in the code
- Highly skilled resources are required.

White box testing techniques

- Basis path testing
- Structure testing
- Statement coverage testing
- Branch coverage testing
- Condition coverage testing
- Loop coverage testing
- Path coverage testing
- Data flow testing

SYSTEM TESTING

System testing is testing of behavior of a complete and fully integrated software product based on the software requirement specification (SRS) document. In main focus of this testing is to evaluate business/ functional/ end user requirements

Various types of system technique

- **Alpha Testing** : is refer to the system testing carried out by test team with in developing organization.
- **Beta Testing** : finally testing before releasing application for commercial purpose. The testing process is done by end user or other.
- **Acceptance Testing** : formal testing conduce to determine whether or not a system satisfies its acceptence criteria and to enable the customer to determine whether or not to accept the system. This testing process is usually perform by customer representation.

- **Recovery Testing** : testing technique which evaluates how well a system recovers from crashes hardware failure, or other problem. This testing process is perform by the testing team.
- **Security testing** : a process to determine that an information system protect data and maintain functionality as in intended it can be perform by testing team.
- **Load Testing** : testing technique that put demands on a system a device and measure it response. It is usually conducted by the performance engineer.

THANK YOU

SOFTWARE QUALITY AND MAINTENANCE

Introduction

Our aim of Software Engineering is to produce good quality maintable software in time and within budget. So quality is important. Software quality refers to two related but distinct notions that exists where quality is defined in business context.

- ✓ Software structural quality refers to how it meets non functional requirements, such as maintability. The degree to which software was produced correctly.

CATEGORIES OF MAINTENANCE

a).Corrective maintenance

Corrective maintenance involves developing and deploying solution to problems that arise during use of software

b). Perfectieve Maintenance

No software program contain. Zero areas for improvement. Perfectieve software involves computer programmers working to improve the way a software program function or how quickly it processes requests

C). Adaptive Maintenance

Ĺhe field of technology constantly changes through both hardware and software development. Adaptive software maintenance address these changes. A changes in a processor's speed.

D). Preventive Maintenance

It mean improving processing efficiency or performance, restructuring the software to improve changeability.

MATURITY MODEL

A maturity model provides a place to start, the benefit of a community's prior experiences, a common language and a shared vision, a frame work for prioritizing actions, a way to define what improvement mean for your organization. A maturity model can be used as bench mark for assessing different organizations for equivalent comparison.

LEVEL OF THE CMM

- Five Level of the CMM

1). Initial(Maturity level 1)

2). Repeatable(Maturity level 2)

3). Defined (Maturity level 3)

4). Managed (Maturity level 4)

5). Optimizing (Maturity level 5)

- Initial (Maturity level 1)

  Processes are usually adhoc and organisations usually does not provide a stable environment. Success in these organizations depends on the competence and good staff good.

- Repeatable (Maturity level 2)

  Software development success are repeatable. The process may not repeat for all projects in the organization.

maturity level.

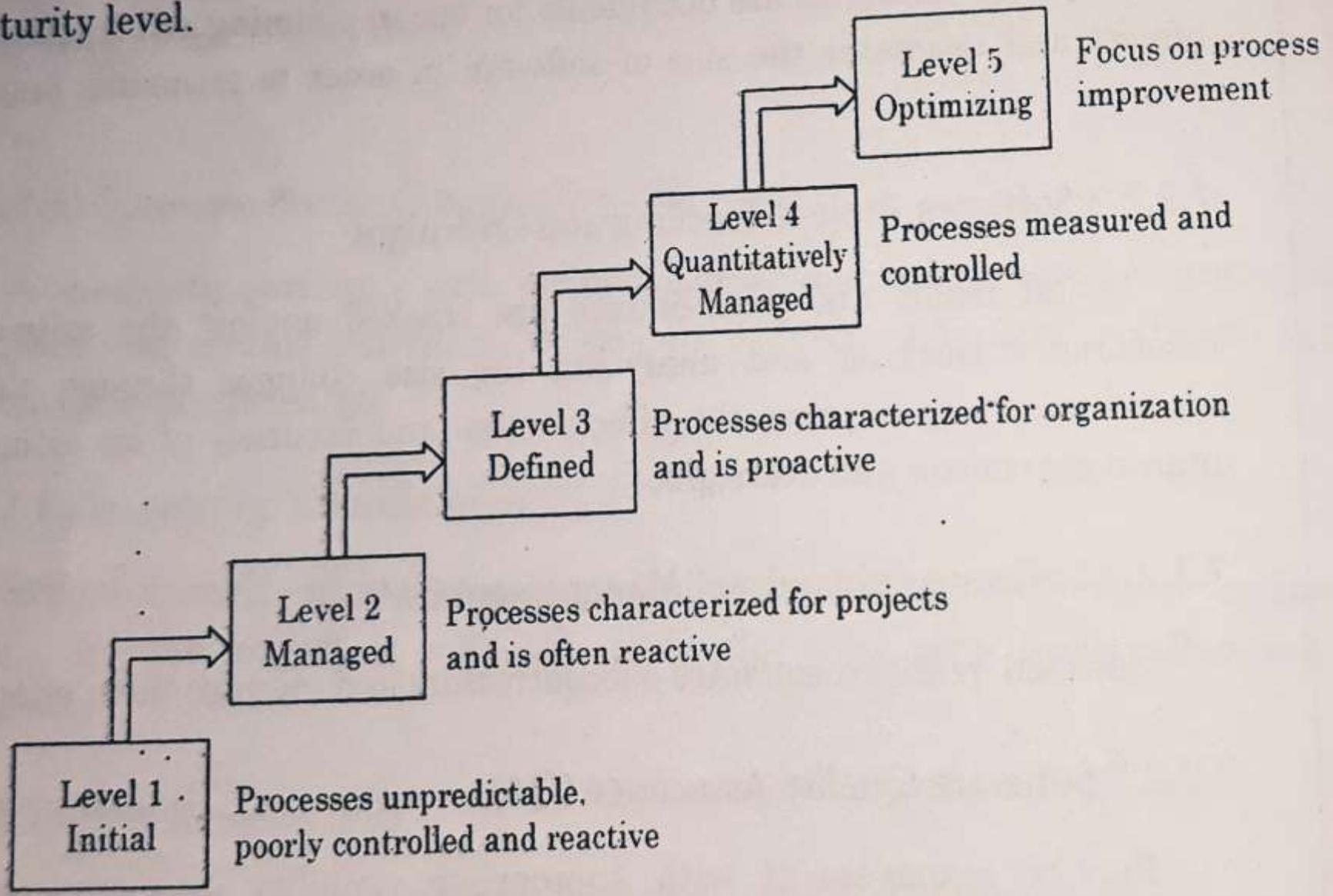


Fig. 7.2 : Characteristics of maturity level

- Defined (Maturity level 3)

The organization's set of standard processes which is basis for level 3, is established and improved over time. These standard processes are used to established consistenyacross the organization.

- Managed (Maturity level 4)

At this level the organization sets quantitative quality goal for both software products and processes productivity and quality are measured for important software process activities across all projects as part an organizational measurement program .

☐ Optimizing (Maturity level 5)

At this level, the entire organization is focused on continuous process improvement. The organizations have to mean to identify weakness and strength. The process proactively, with the goal preventing the occurrence of defeats. Data of the effectively, with the goal preventing the occurrence of defeats.

KEY PROCESS AREA

Each level represents a stage of organizational maturity describes in term of its " Key Process Area".

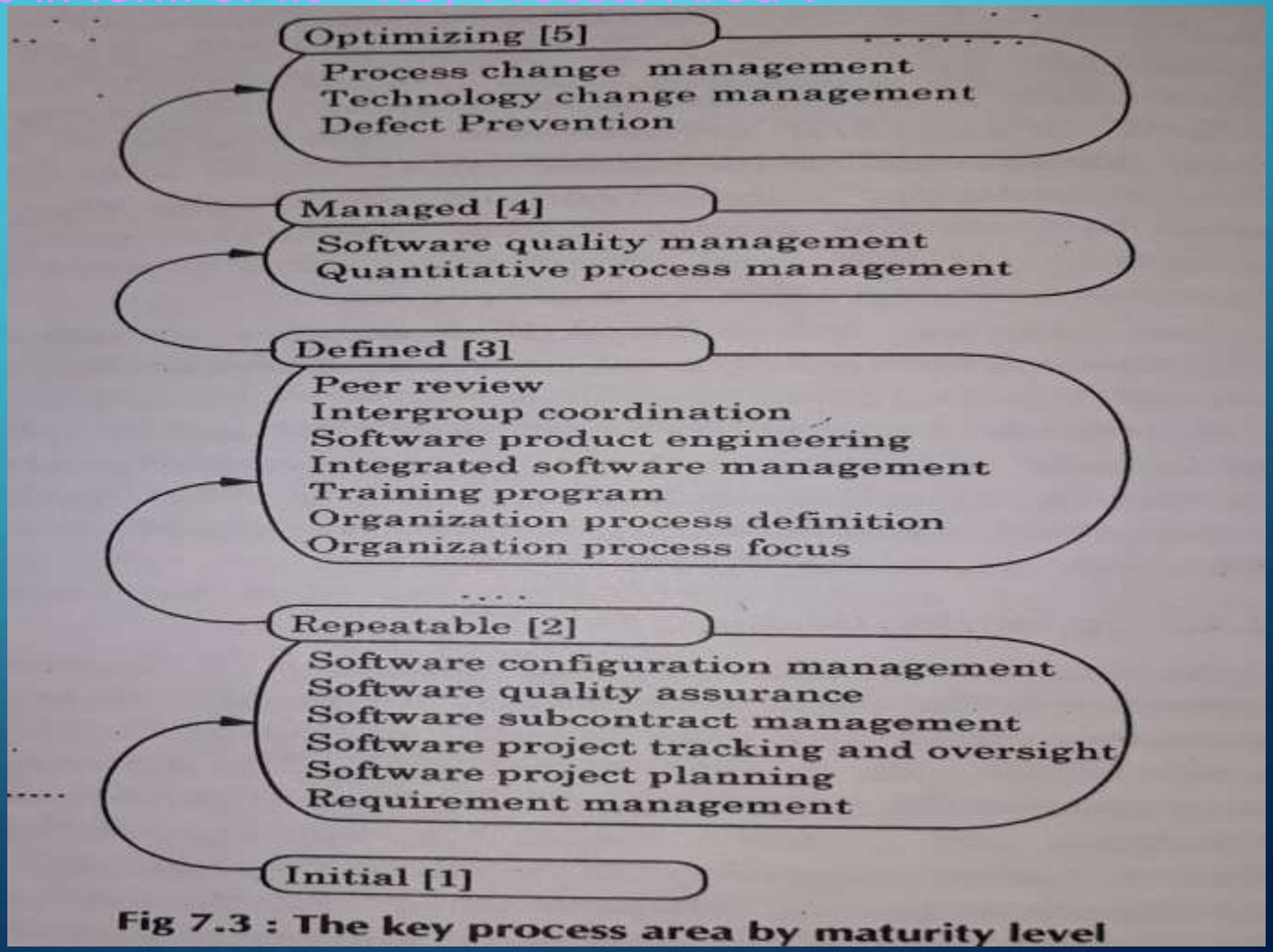


Fig 7.3 : The key process area by maturity level

- ✓ **Requirement Management:-** The software engineering group reviews the allocated requirements before they are incorporated into software project and establish a relationship between customer and development in order to understand and the requirements of project.
- ✓ **Software Project Planning:-** Software estimates are documents for use in planning and tracking the software project and estimates the size of software in order to estimates project effort and cost.

- ✓ **Software Project Tracking and Overnight:-** Actual result and performance are tracked against the software plans.
- ✓ **Software Subcontract Management(SM):-** Selected qualified software subcontractors and manage them effectively.
- ✓ **Software Quality Assurance (QA):-** provide management with appropriate visibility into process being use by software project and of products beings builts.

- ✓ Software Configuration Management(CM):- During projects software life cycle maintain the integrity of the products. At level 3 both project and organizational issues are key process area.
- ✓ Organizational Process Focus(PF):- Organization focus on software process activities that improve the organization's over all software process capability.

- ✓ Organization Process Definition:- Develop and maintain set of software process quality that improve process performance across the projects and provide a basis for successive addition, long term benefits to the organization.
- ✓ Training Program:- Conduct a training program for engineer so that they can perform their roles effectively and efficiently.

- ✓ Integrated Software Management(M1):- Integrated the software engineering and management activities into logical manner, defined software process that is design from organization's standard software process.
- ✓ Software Product Engineering(PE):-Consistently perform a well- defined engineering process that integrates all the software engineering activities to produce correct, consistent software product effectively and efficiently.

- ✓ Intergroup Coordination:- Software Engineering group to communicate actively with other engineering groups. So the project is able to satisfy the customer's needs effectively and efficiently.
- ✓ Peer Reviews (PR):- Remove defects from software work product early and efficiently.
- ✓ Quantitative Process Management:- Control the process performance of the software project quantitatively

- ✓ **Software Quality Management:-** Develop quantitative understanding of quality of the project software product and achieve specific quality goal.
- ✓ **Defect Prevention(DP):-** Identify the cause of defects and prevent them from occurring.

- ✓ Technology Change Management:- Identify beneficial new technologies and transfer them into organization in an orderly manner.
- ✓ Processes Change Management:- Continually improve the software processes used in organization with intent of improving software quality, increasing productivity and decreasing the cycle time for product development.

ISO 9000

INTRODUCTION:- Quality is something every company strives for and is often time very difficult to achieve. Complications concerning efficiency and quality present themselves everyday in business. ISO 9000 is one of the most widely recognized in the world. ISO 9000 is a quality management standard that present guide lives intended to increase business efficiency and customer satisfaction.

IMPORTANCE OF ROOT CAUSE ANALYSIS AND SYSTEMIC CORRECTIVE ACTION IN ISO 9001 MANAGEMENT SYSTEM

When problem solving, it is important to find the cause of problem in order to develop a solution. That is why ISO 9000 stresses the important of finding root cause of problem. There may be multiple reasons why a process is not working correctly and finding actual cause will lead a company one step closer to a solution and implementation of corrective action.

SIX SIGMA

- Six sigma is a set of technique and tools for process improvement. It was developed by Motorola in 1986. six sigma is based on various quality management theories. The word sigma is statistical term that measures how far a given process derivates form perfection.

KEY CONCEPT OF SIX SIGMA

- Critical to quality
- Defect
- Process capability
- Variation
- Stable operations
- Design for six sigma

SIX KEY ELEMENT

Customers:- define quality

Processes:- defining processes and defining metrics and measures for processes is the key element of six sigma quality.

Employees:- the company must involve all employees in six sigma program.

SIX SIGMA ORGANIZATION

- Leaderships
- Sponsor
- Implementation leader
- Coach
- Team leader
- Team member
- Process owner
- Belt colors
- Black belt
- Master black belt
- Green belt

DIFFERENCE BETWEEN SIX SIGMA AND ISO 9000

- Six sigma is management philosophy that focus on solving all causes for defects to improve quality.
- Six sigma is implemented and govern by completely internally.
- Six sigma has several certifications a company can obtain including black belt or green belt.

- ISO 9000 is set of standard and certification for proper use of a quality management system.
- ISO 9000 certification require and outside auditing process.
- A company to adhere to ISO 9000 standards, it must be certified. An outside assess the company.

SOFTWARE CONFIGURATION MANAGEMENT

Changes continuously take place in software project change due to the evaluation of work products as the project proceed, changes due to defects being count and then fixed, changes due to requirement changes will stop. All these are reflected in the files containing source, data. Configuration management is the discipline for systematically controlling the changes that take place during development.

REQUIREMENTS

- Inconsistency
- Concurrency
- Stability
- Accounting and maintaining status information
- Variants

CONFIGURATION MANAGEMENT ACTIVITIES

Configuration Identification:- project manager normally classifies the, object associated with software development into three main categories:-

1. Controlled
2. Pre controlled object
3. Uncontrolled object

Configuration control:- It is process of managing changes to controlled objects. Configuration control is the part of a configuration management system that most directly affects the day to day operations of developers. The configuration control system prevents unauthorized changes to any controlled object.

Change is well motivated.

Developer has considered and documented the effects of the change.

Change interact well with the changes made by other developers.