KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act, 1956)

Coimbatore - 641 021, India

FACULTY OF ARTS, SCIENCE AND HUMANITIES (FASH)

Department of CS,CA & IT

II B.Sc CS A & B   IV SEMESTER   BATCH : 2018 - 2021

18CSU402   SOFTWARE ENGINEERING   4H – 4C

Instruction Hours / week: L: 4 T: 0 P: 0   **Marks:** Int : **40** Ext : **60** Total: **100**

SCOPE

The graduates of the software engineering program shall be able to apply proper theoretical, technical, and practical knowledge of software requirements, analysis, design, implementation, verification and validation, and documentation. This course enables the students to resolve conflicting project objectives considering viable tradeoffs within limitations of cost, time, knowledge, existing systems, and organizations.

COURSE OBJECTIVES

- Apply their knowledge of mathematics, sciences, and computer science to the modeling, analysis, and measurement of software artifacts.
- Work effectively as leader/member of a development team to deliver quality software artifacts.
- Analyze, specify and document software requirements for a software system.
- Implement a given software design using sound development practices.
- Verify, validate, assess and assure the quality of software artifacts.
- Design, select and apply the most appropriate software engineering process for a given project, plan for a software project, identify its scope and risks, and estimate its cost and time.
- Express and understand the importance of negotiation, effective work habits, leadership, and good communication with stakeholders, in written and oral forms, in a typical software development environment.

COURSE OUTCOME

- The ability to analyze, design, verify, validate, implement, apply, and maintain software systems.
- An ability to use the techniques, skills, and modern engineering tools necessary for engineering practice.
- Knowledge of contemporary issues.
- An ability to identify, formulates, and solves engineering problems.
- An ability to work in one or more significant application domains

UNIT-I:

Introduction: The Evolving Role of Software, Software Characteristics, Changing Nature of Software, Software Engineering as a Layered Technology, Software Process Framework, Framework and Umbrella Activities, Process Models, Capability Maturity Model Integration (CMMI).

UNIT-II:

Requirement Analysis; Initiating Requirement Engineering Process- Requirement Analysis and Modeling Techniques- Flow Oriented Modeling- Need for SRS-Characteristics and Components of SRS- **Software Project Management**: Estimation in Project Planning Process, Project Scheduling.

UNIT-III:

Risk Management: Software Risks, Risk Identification Risk Projection and Risk Refinement, RMMM plan, **Quality Management-** Quality Concepts, Software Quality Assurance, Software Reviews, Metrics for Process and Projects

UNIT-IV:

Design Engineering:-Design Concepts, Architectural Design Elements, Software Architecture, Data Design at the Architectural Level and Component Level, Mapping of Data Flow into Software Architecture, Modeling Component Level Design

UNIT-V

Testing Strategies & Tactics: Software Testing Fundamentals, Strategic Approach to Software Testing, Test Strategies for Conventional Software, Validation Testing, System testing Black-Box Testing, White-Box Testing and their type, Basis Path Testing

SUGGESTED READINGS

1. Pressman, R.S. (2009). Software Engineering: A Practitioner‘s Approach (7th ed.). New Delhi: McGraw-Hill.
2. Jalote, P. An Integrated Approach to Software Engineering (2nd ed.). New Delhi: New Age International Publishers.
3. Aggarwal, K.K., & Singh, Y. (2008). Software Engineering (2nd ed.). New Delhi: New Age International Publishers.
4. Sommerville, I. (2006). Software Engineering (8th ed.). New Delhi: Addison Wesley.
5. Bell, D. (2005). Software Engineering for Students (4th ed.) New Delhi: Addison-Wesley.
6. Mall, R. (2004). Fundamentals of Software Engineering (2nd ed.). New Delhi: Prentice-Hall of India.

WEB SITES

1. http://en.wikipedia.org/wiki/Software_engineering
2. http://www.onesmartclick.com/engineering/software-engineering.html
3. http://www.CSU.gatech.edu/classes/AY2000/cs3802_fall/
4. www.tutorialpoint.com

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act, 1956)

Coimbatore - 641 021, India

Depatment of CS,CA & IT

STAFF NAME : D.Manjula & Dr.S.Saravana Kumar

SUBJECT NAME : SOFTWARE ENGINEERING **SUB.CODE : 18CSU402**

SEMESTER : IV **CLASS : II B.Sc (CS) A & B**

UNIT I

S.NO	Lecture Duration (Hours)	Topics To Be Covered	Support Materials/ Pg.No
		Introduction	
1	1	The Evolving Role of Software	W1
		Software Characteristics	T1 : 04 - 07
2	1	Changing Nature of Software	T1 : 10 - 12
3	1	Software Engineering as a Layered Technology	T1 : 12 - 14
4	1	Software Process Framework	T1 : 14 - 16
5	1	Framework and Umbrella Activities	W2
6	1	Process Models	T1 : 30 - 52
7	1	Capability Maturity Model Integration (CMMI).	W1
8	1	**Recapitulation and Discussion of Important Questions**	
		Total No of Hours Planned for Unit I	**8**

UNIT II

S.NO	Lecture Duration (Hours)	Topics To Be Covered	Support Materials/ Pg.No
		Requirement Analysis	
1	1	Initiating Requirement Engineering Process	T1 : 149 - 153
2	1	Requirement Analysis and Modeling Techniques	T1 : 186 - 188
3	1	Flow Oriented Modeling	
4	1	Need for SRS	W1
5	1	Characteristics and Components of SRS	T3: 65-66
		Software Project Management	
6	1	Estimation in Project Planning Process	T1 : 691 - 712
7	1	Project Scheduling	T1 : 722 - 726
8	1	**Recapitulation and Discussion of Important Questions**	
		Total No of Hours Planned for Unit II	**8**

UNIT III

S.NO	Lecture Duration (Hours)	Topics To Be Covered	Support Materials/ Pg.No
		Risk Management	
1	1	Software Risks	T1 : 745 - 757
		Risk Identification	
2	1	Risk Projection and Risk Refinement	
3	1	RMMM plan	
		Quality Management	
4	1	Quality Concepts	T1 : 399 - 413
5	1	Software Quality Assurance	T1 : 443 - 443
6	1	Software Reviews,	T1 : 417 - 424
7	1	Metrics for Process and Projects	T1 : 667 - 681
8	1	**Recapitulation and Discussion of Important Questions**	
		Total No of Hours Planned for Unit III	**8**

UNIT IV

S.NO	Lecture Duration (Hours)	Topics To Be Covered	Support Materials/ Pg.No
		Design Engineering	
1	1	Design Concepts	T1: 223-230
2	1	Architectural Design Elements	T1:233-234
3	1	Software Architecture	T1: 243-246
4	1	Data Design at the Architectural Level and Component Level	T1:234-237
5	1	Mapping of Data Flow into Software Architecture	T1:265-272
6	1	Modeling Component Level Design	T1:282 -288
7	1	**Recapitulation and Discussion of Important Questions**	
		Total No of Hours Planned for Unit IV	**7**

UNIT V

S.NO	Lecture Duration (Hours)	Topics To Be Covered	Support Materials/ Pg.No
		Testing Strategies & Tactics:	
1	1	Software Testing Fundamentals	T1: 482
		Strategic Approach to Software Testing	T1: 450-455
2	1	Test Strategies for Conventional Software	T1: 456-459
3	1	Validation Testing	T1: 467-472, W3
		System testing	
4	1	Black-Box Testing	T1:495-499, 485
5	1	White-Box Testing and their type	
6	1	Basis Path Testing	T1:485-491
7	1	**Recapitulation and Discussion of Important Questions**	
		Discussion of previous ESE question papers	
8	1	**Discussion of previous ESE question papers**	
9	1	**Discussion of previous ESE question papers**	
		Total No of Hours Planned for Unit V	**9**
Total Hours			**40**

S.NO	TEXT BOOKS
T1	Pressman, R.S. (2009). Software Engineering: A Practitioner‘s Approach (7th ed.). New Delhi: McGraw-Hill.
T2	Aggarwal, K.K., & Singh, Y. (2008). Software Engineering (2nd ed.). New Delhi: New Age International Publishers
S.NO	**WEB SITES**
W1	www.tutorialpoints.com
W2	www.studytonight.com
W3	www.geeksforgeeks.com

KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

II B.Sc(CS) (BATCH 2018-2021) IV SEMESTER

SOFTWARE ENGINEERING (18CSU402)

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

UNIT I

S.NO	QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	ANSWER
1	Software takes on a ______________role.	single	dual	triple	tetra	dual
2	Software is a ______________.	virtual	system	modifier	framework	modifier
3	Instructions that when executed provide desired function and performance is called	software	hardware	firmware	humanware	software
4	High quality of software is achieved through ________________.	testing	good design	construction	manufacture	good design
5	Software doesn't ________________.	tearout	wearout	degrade	deteriorate	wearout
6	Software is not susceptible to ______________.	hardware	defects	environmental melodies	deterioration	environmental melodies
7	Software will undergo __________.	database	testing	enhancement	manufacture	enhancement
8	____refers to the meaning and form of incoming and outgoing information.	content	software	hardware	data	content
9	____refers to the predictability of the order and timing of information.	system software	network software	information determinacy	database	information determinacy

10	_____________is not a system software.	MS Office	compiler	editor	file management	MS Office
11	Collection of programs written to service other programs are called __________.	system software	business software	embedded software	pc software	system software
12	Which one is not coming under software myths	Management myths	customer myths	product myths	practitioners myths	product myths
13	__________ is a PC Software.	MS word	LISP	CAD	C	MS word
14	Software that monitors, analyses, controls real world events is called _________.	Business software	real time software	web based software	embedded software	real time software
15	The bedrock that supports software engineering is a_______	tools	methods	process models	quality focus	quality focus
16	A complete software process by identifying a small number of ______	framework activities	umbrella activities	process framework	software process	framework activities
17	The process framework encompassess a set of ________	framework activities	umbrella activities	process framework	software process	umbrella activities
18	software engineering action is________________	design	chronic	decision	crisis	design
19	Which one is effect the outcome of the project?	Risk management	Measurement	technical reviews	Reusability	Risk management
20	Continuing indefinitely is called ___________.	crisis	decision	affliction	chronic	chronic
21	Component based development uses ____________.	functions	subroutines	procedures	objects	objects
22	UML stands for ____________.	Universal Modelling Language	User Modified Language	Unified Modelling Language	User Model Language	Unified Modelling Language
23	A model which uses formal mathematical specification is called ________.	4 GT model	Unified method model	formal methods model	component based development	formal methods model

24	A variation of formal methods model is called _____________.	component based development	4 GT model	unified method model	cleanroom software engineering	cleanroom software engineering
25	The development of formal methods is ___________.	less time consuming	quite time consuming	does not consume time	very less time consuming	quite time consuming
26	The first step to develop software is ______________.	analysis	design	requirements gathering	coding	requirements gathering
27	The waterfall model sometimes called as	classic model	classic life cycle model	life cycle model	cycle model	classic life cycle model
28	Software engineering activities include ____________	decision	affliction	hardware	maintenance	maintenance
29	all process model prescribes a ______________.	circular	elliptical	spiral	workflow	workflow
30	Component based development incorporates the characteristics of the ___________model	circular	elliptical	spiral	hierarchical	spiral
31	Prototype is a ______________.	software	hardware	computer	model	model
32	For small applications it is possible to move from requirement gathering step to____________.	analysis	implementatio n	design	modeling	implementatio n
33	Software project management begins with a set of activities that are collectively called ________________	project planning	software scope	software estimation	decomposition	project planning
34	Breaking up of a complex problem into small steps is called ____________.	project planning	software scope	software estimation	decomposition	decomposition
35	The ease with which software can be transferred from one computer to another. This quality attribute is called _______________.	portability	reliability	efficiency	accuracy	portability
36	The ability of a program to perform a required function under stated condition for a stated period of time. This quality	portability	reliability	efficiency	accuracy	reliability

37	The event to which software performs its intended function. This quality attribute is called ________________.	portability	reliability	efficiency	accuracy	efficiency
38	A qualitative assessments of freedom from errors. This quality attribute is called ____________.	portability	reliability	efficiency	accuracy	accuracy
39	The extent to which software can continue to operate correctly. This quality attribute is called ______________.	robustness	correctness	efficiency	reliability	robustness
40	The extent to which the software is free from design and coding defects ie fault free. This quality attribute is called _____________.	robustness	correctness	efficiency	reliability	correctness
41	System shall reside in 50KB of memory is an example of _____________.	quantified requirement	qualified requirement	functional requirement	performance requirement	quantified requirement
42	Accuracy shall be sufficient to support mission is an example of ___________.	quantified requirement	qualified requirement	functional requirement	performance requirement	qualified requirement
43	System shall make efficient use of memory is an example of ______________.	quantified requirement	qualified requirement	functional requirement	performance requirement	qualified requirement
44	Which level of CMM is for process control?	Initial	Repeatable	Defined	Optimizing	Optimizing
45	Product is	Deliverables	User expectations	Organization's effort in development	none of the above	Deliverables
46	To produce a good quality product, process should be	Complex	Efficient	Rigorous	none of the above	Efficient
47	Which is not a product metric?	Size	Reliability	Productivity	Functionality	Productivity
48	Which is NOT a process metric?	Productivity	Functionality	Quality	Efficiency	Functionality
49	Effort is measured in terms of:	Person- months	Rupees	Persons	Months	Person-months

50	An independently deliverable piece of functionality providing access to its services through interface is called	Software measurement	Software composition	Software measure	Software component	Software component
51	Management of software development is dependent on	People	product	Process	all of the above	all of the above
52	During software development, which factor is most crucial?	People	Product	Process	Project	People
53	Program is	Subset of software	super set of software	Software	none of the above	Subset of software
54	Milestones are used to	Know the cost of the project	know the status of the project	Know user expectations	none of the above	know the status of the project
55	Software engineering approach is used to achieve:	Better performance of hardware	Error free software	Reusable software	Quality software product	Quality software product
56	Software consists of	instructions + operating system	documentation + operating procedures	Programs + hardware manuals	Set of programs	documentation + operating procedures
57	CASE Tool is	Aided Software Engineering	Aided Software Engineering	Aided Software Engineering	Analysis Software Engineering	Aided Software Engineering
58	SDLC stands for	Software design life cycle	Software development life cycle	System development life cycle	System design life cycle	Software development life cycle
59	RAD stands for	Rapid application development	Relative application development	Ready application development	Repeated application development	Rapid application development
60	Which phase is not available in software life cycle?	Coding	Testing	Maintenance	Abstraction	Abstraction

UNIT I

Introduction: The Evolving Role of Software, Software Characteristics, Changing Nature of Software, Software Engineering as a Layered Technology, Software Process Framework, Framework and Umbrella Activities, Process Models, Capability Maturity Model Integration (CMMI).

Introduction to Software Engineering

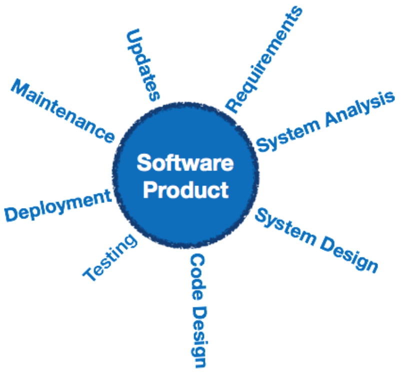
Software is a program or set of programs containing instructions which provide desired functionality. And Engineering is the processes of designing and building something that serves a particular purpose and finds a cost effective solution to problems.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**

Software Engineering *is a systematic approach to the design, development, operation, and maintenance of a software system.*

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.



Dual Role of Software:

1. As a product –

- It delivers the computing potential across network of Hardware.
- It enables the Hardware to deliver the excepted functionality.
- It acts as information transformer because it produces, manages, acquires, modifies, displays, or transmits information.

2. As a vehicle for delivering a product –

- It provides system functionality (e.g., payroll system)
- It controls other software (e.g., an operating system)
- It helps build other software (e.g., software tools)

Objectives of Software Engineering:

1. **Maintainability –**It should be feasible for the software to evolve to meet changing requirements.

2. **Correctness –**A software product is correct, if the different requirements as specified in the SRS document have been correctly implemented.

3. **Reusability –**A software product has good reusability, if the different modules of the product can easily be reused to develop new products.

4. **Testability –**Here software facilitates both the establishment of test criteria and the evaluation of the software with respect to those criteria.

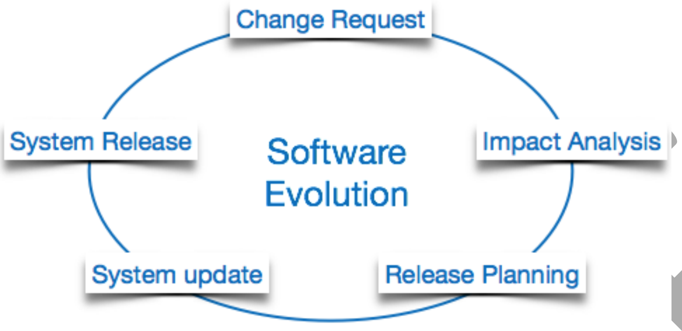
5. **Reliability –**It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.

6. **Portability –**In this case, software can be transferred from one computer system or environment to another.

7. **Adaptability –**In this case, software allows differing system constraints and user needs to be satisfied by making changes to the software.

Software Evolution

The process of developing a software product using software engineering principles and methods is referred to as **software evolution.** This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements



Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.

Even after the user has desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with requirement is not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

Software Evolution Laws

Lehman has given laws for software evolution. He divided the software into three different categories:

- **S-type (static-type) -** This is a software, which works strictly according to defined specifications and solutions. The solution and the method to achieve it, both are immediately understood before coding. The s-type software is least subjected to changes hence this is the simplest of all. For example, calculator program for mathematical computation.

- **P-type (practical-type) -** This is a software with a collection of procedures. This is defined by exactly what procedures can do. In this software, the specifications can be described but the solution is not obvious instantly. For example, gaming software.
- **E-type (embedded-type) -** This software works closely as the requirement of real-world environment. This software has a high degree of evolution as there are various changes in laws, taxes etc. in the real world situations. For example, Online trading software.

Characteristics of a software

- Software should achieve a good quality in design and meet all the specifications of the customer.
- Software does not wear out i.e. it does not lose the material.
- Software should be inherently complex.
- Software must be efficient i.e. the ability of the software to use system resources in an effective and efficient manner.
- Software must be integral i.e. it must prevent from unauthorized access to the software or data.

Its characteristics that make it different from other things human being build.

Features of such logical system:

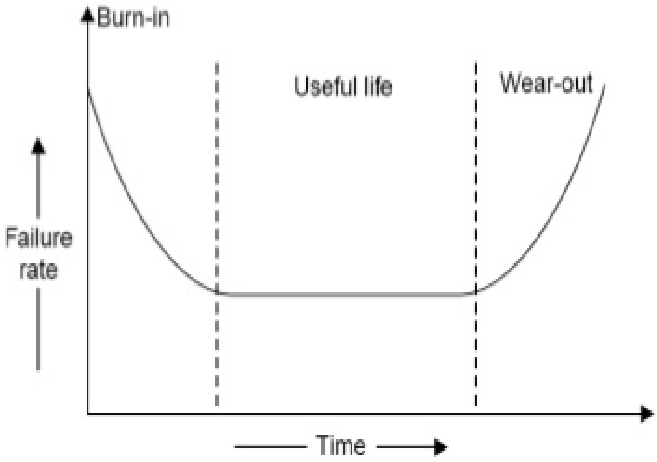
1. Software is developed or engineered, it is not manufactured in the classical sense which has quality problem.

- There is no manufacture phase (or problems during manufacture are easy to correct)
- Software costs are concentrated in engineering

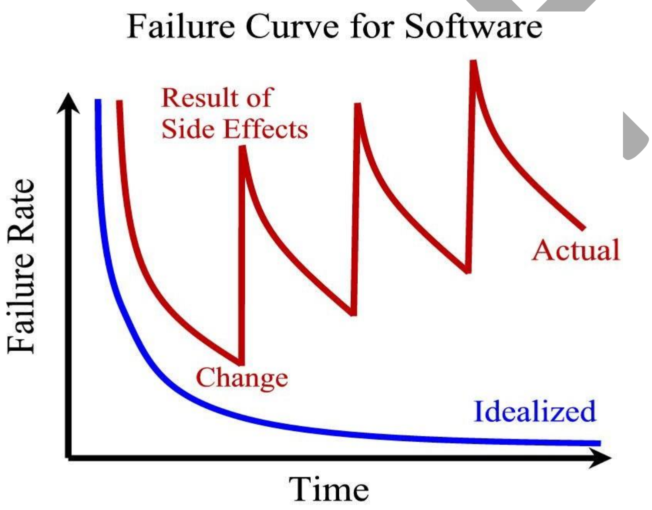
2 Software doesn't "wear out.” but it deteriorates (due to change).

Hardware has bathtub curve of failure rate (high failure rate in the beginning, then drop to steady state, then cumulative effects of dust, vibration, abuse occurs).

Failure Bath Tub curve for hardware



Failure curve for software



3. Although the industry is moving toward component-based construction (e.g. standard screws and off-the-shelf integrated circuits), most software continues to be custom-built.

Modern reusable components encapsulate data and processing into software parts to be reused by different programs. E.g. graphical user interface, window, pull-down menus in library etc.

Software Applications

- System software
- Real-time software
- Business software
- Engineering and scientific software
- Embedded software
- Personal computer software
- Web-based software

Software engineering - Layered technology

- Software engineering is a fully layered technology.
- To develop a software, we need to go from one layer to another.
- All these layers are related to each other and each layer demands the fulfillment of the previous layer.

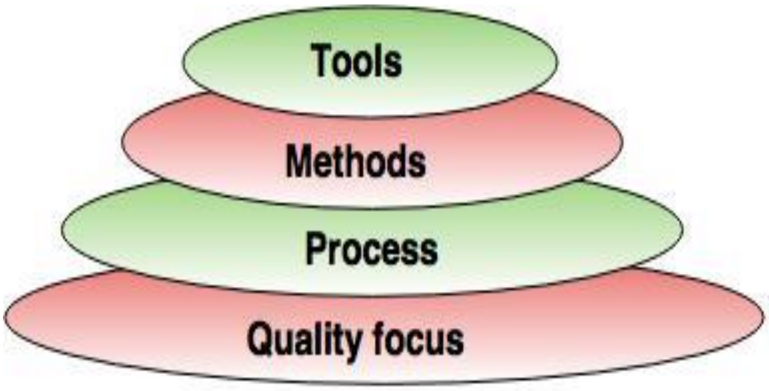


Fig. - Software Engineering Layers

The layered technology consists of:

1. Quality focus
The characteristics of good quality software are:

- Correctness of the functions required to be performed by the software.

- Maintainability of the software
- Integrity i.e. providing security so that the unauthorized user cannot access information or data.
- Usability i.e. the efforts required to use or operate the software.

2. Process

- It is the base layer or foundation layer for the software engineering.
- The software process is the key to keep all levels together.
- It defines a framework that includes different activities and tasks.
- In short, it covers all activities, actions and tasks required to be carried out for software development.

3. Methods

- The method provides the answers of all 'how-to' that are asked during the process.
- It provides the technical way to implement the software.
- It includes collection of tasks starting from communication, requirement analysis, analysis and design modelling, program construction, testing and support.

4. Tools

- The software engineering tool is an automated support for the software development.
- The tools are integrated i.e the information created by one tool can be used by the other tool.
- **For example:** The Microsoft publisher can be used as a web designing tool.

Software Process Framework

- The process of framework defines a small set of activities that are applicable to all types of projects.
- The software process framework is a collection of task sets.
- Task sets consist of a collection of small work tasks, project milestones, work productivity and software quality assurance points.

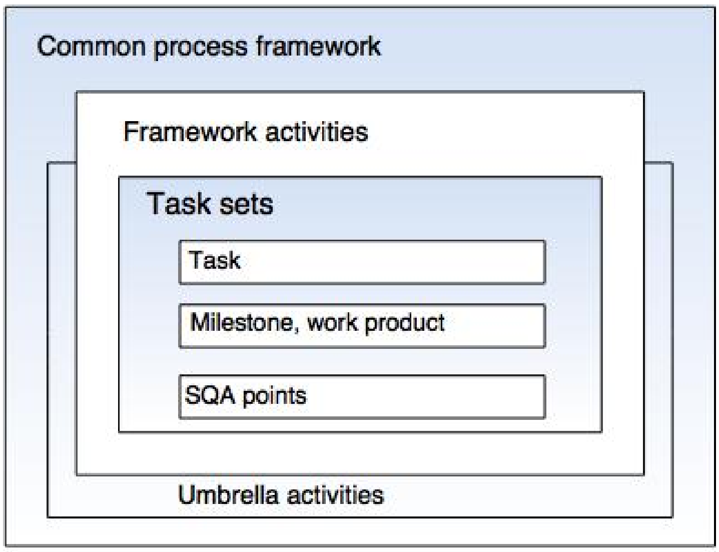


Fig.- A software process framework

Umbrella activities

Typical umbrella activities are:

1. Software project tracking and control

- In this activity, the developing team accesses project plan and compares it with the predefined schedule.
- If these project plans do not match with the predefined schedule, then the required actions are taken to maintain the schedule.

2. Risk management

- Risk is an event that may or may not occur.
- If the event occurs, then it causes some unwanted outcome. Hence, proper risk management is required.

3. Software Quality Assurance (SQA)

- SQA is the planned and systematic pattern of activities which are required to give a guarantee of software quality.

For example, during the software development meetings are conducted at every stage of development to find out the defects and suggest improvements to produce good quality software.

4. Formal Technical Reviews (FTR)

- FTR is a meeting conducted by the technical staff.
- The motive of the meeting is to detect quality problems and suggest improvements.
- The technical person focuses on the quality of the software from the customer point of view.

5. Measurement

- Measurement consists of the effort required to measure the software.
- The software cannot be measured directly. It is measured by direct and indirect measures.
- Direct measures like cost, lines of code, size of software etc.
- Indirect measures such as quality of software which is measured by some other factor. Hence, it is an indirect measure of software.

6. Software Configuration Management (SCM)

- It manages the effect of change throughout the software process.

7. Reusability management

- It defines the criteria for reuse the product.
- The quality of software is good when the components of the software are developed for certain application and are useful for developing other applications.

8. Work product preparation and production

- It consists of the activities that are needed to create the documents, forms, lists, logs and user manuals for developing a software.

Software process Models

A software process is a collection of various activities.

There are five generic process framework activities:

1. Communication:

The software development starts with the communication between customer and developer.

2. Planning:
It consists of complete estimation, scheduling for project development and tracking.

3. Modeling:

- Modeling consists of complete requirement analysis and the design of the project like algorithm, flowchart etc.
- The algorithm is the step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.

4. Construction:

- Construction consists of code generation and the testing part.
- Coding part implements the design details using an appropriate programming language.
- Testing is to check whether the flow of coding is correct or not.
- Testing also check that the program provides desired output.

5. Deployment:

- Deployment step consists of delivering the product to the customer and take feedback from them.
- If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.

The following framework activities are carried out irrespective of the process model chosen by the organization.

1. Communication
2. Planning
3. Modeling
4. Construction
5. Deployment

The name 'prescriptive' is given because the model prescribes a set of activities, actions, tasks, quality assurance and change the mechanism for every project.

There are three types of prescriptive process models. They are:

1. The Waterfall Model
2. Incremental Process model
3. RAD model

1. The Waterfall Model

- The waterfall model is also called as **'Linear sequential model'** or **'Classic life cycle model'.**
- In this model, each phase is fully completed before the beginning of the next phase.
- This model is used for the small projects.
- In this model, feedback is taken after each phase to ensure that the project is on the right path.
- Testing part starts only after the development is complete.

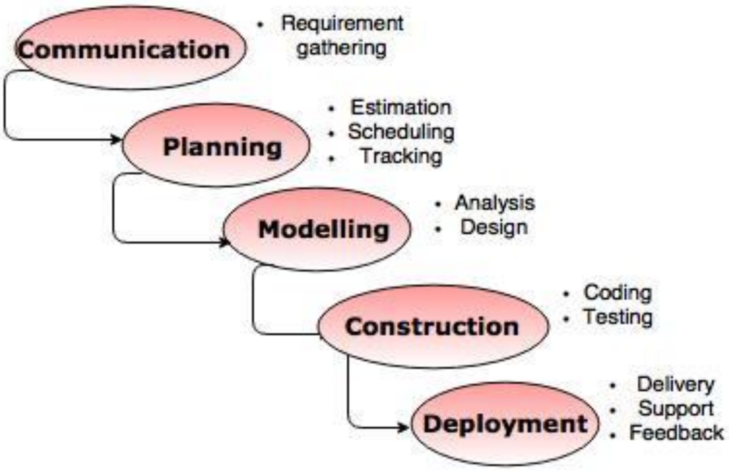


Fig. - The Waterfall model

NOTE: The description of the phases of the waterfall model is same as that of the process model.

An alternative design for 'linear sequential model' is as follows:

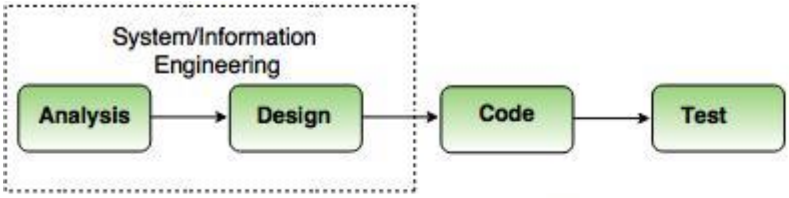


Fig. - The linear sequential model

Advantages of waterfall model

- The waterfall model is simple and easy to understand, implement, and use.
- All the requirements are known at the beginning of the project, hence it is easy to manage.
- It avoids overlapping of phases because each phase is completed at once.
- This model works for small projects because the requirements are understood very well.
- This model is preferred for those projects where the quality is more important as compared to the cost of the project.

Disadvantages of the waterfall model

- This model is not good for complex and object oriented projects.
- It is a poor model for long projects.
- The problems with this model are uncovered, until the software testing.
- The amount of risk is high.

2. Incremental Process model

- The incremental model combines the elements of waterfall model and they are applied in an iterative fashion.
- The first increment in this model is generally a core product.
- Each increment builds the product and submits it to the customer for any suggested modifications.

- The next increment implements on the customer's suggestions and add additional requirements in the previous increment.
- This process is repeated until the product is finished.

For example, the word-processing software is developed using the incremental model.

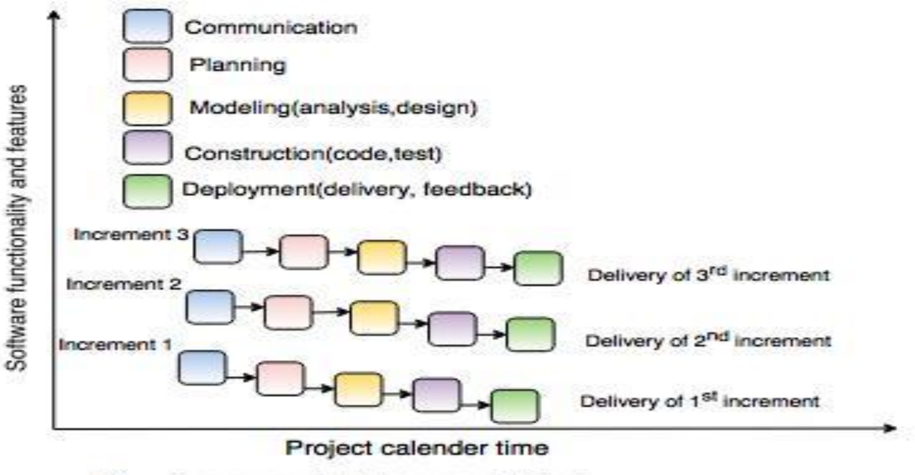

Communication
Planning
Modeling(analysis,design)
Construction(code,test)
Deployment(delivery, feedback)
Increment 3
Increment 2
Increment 1
Delivery of 3rd increment
Delivery of 2nd increment
Delivery of 1st increment
Software functionality and features
Project calender time

Fig. - Incremental Process Model

Advantages of incremental model

- This model is flexible because the cost of development is low and initial product delivery is faster.
- It is easier to test and debug during the smaller iteration.
- The working software generates quickly and early during the software life cycle.
- The customers can respond to its functionalities after every increment.

Disadvantages of the incremental model

- The cost of the final product may cross the cost estimated initially.
- This model requires a very clear and complete planning.
- The planning of design is required before the whole system is broken into small increments.
- The demands of customer for the additional functionalities after every increment causes problem during the system architecture.

3. RAD model

- RAD is a Rapid Application Development model.
- Using the RAD model, software product is developed in a short period of time.
- The initial activity starts with the communication between customer and developer.
- Planning depends upon the initial requirements and then the requirements are divided into groups.
- Planning is more important to work together on different modules.

The RAD model consist of following phases:

1. Business Modeling

- Business modeling consist of the flow of information between various functions in the project.
- For example what type of information is produced by every function and which are the functions to handle that information.
- A complete business analysis should be performed to get the essential business information.

2. Data modeling

- The information in the business modeling phase is refined into the set of objects and it is essential for the business.
- The attributes of each object are identified and define the relationship between objects.

3. Process modeling

- The data objects defined in the data modeling phase are changed to fulfil the information flow to implement the business model.
- The process description is created for adding, modifying, deleting or retrieving a data object.

4. Application generation

- In the application generation phase, the actual system is built.
- To construct the software the automated tools are used.

5. Testing and turnover

- The prototypes are independently tested after each iteration so that the overall testing time is reduced.
- The data flow and the interfaces between all the components are are fully tested. Hence, most of the programming components are already tested.

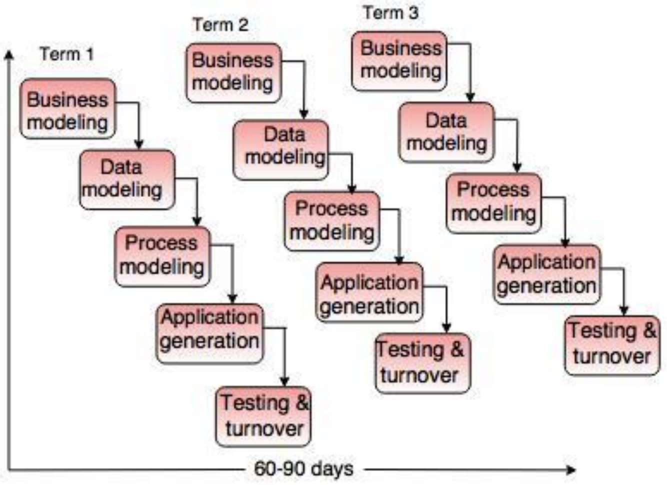


Fig. - RAD Model

Evolutionary Process Models

- Evolutionary models are iterative type models.
- They allow to develop more complete versions of the software.

Following are the evolutionary process models.

1. The prototyping model
2. The spiral model
3. Concurrent development model

1. The Prototyping model

- Prototype is defined as first or preliminary form using which other forms are copied or derived.
- Prototype model is a set of general objectives for software.
- It does not identify the requirements like detailed input, output.
- It is software working model of limited functionality.

- In this model, working programs are quickly produced.

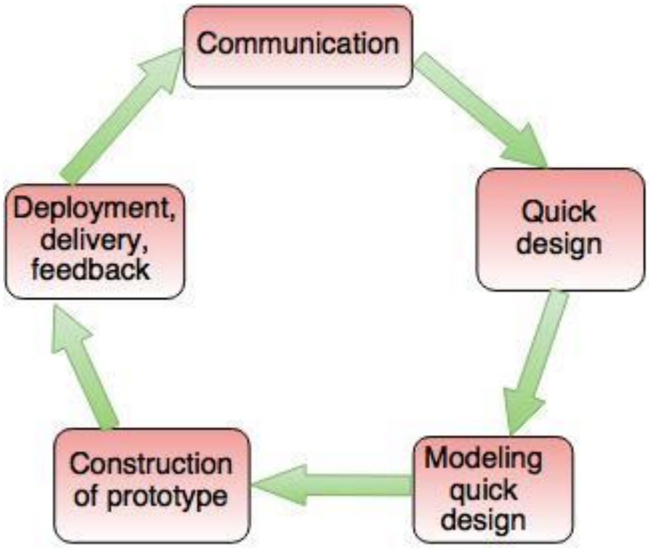


Fig. - The Prototyping Model

The different phases of Prototyping model are:

1. Communication
In this phase, developer and customer meet and discuss the overall objectives of the software.

2. Quick design

- Quick design is implemented when requirements are known.
- It includes only the important aspects like input and output format of the software.
- It focuses on those aspects which are visible to the user rather than the detailed plan.
- It helps to construct a prototype.

3. Modeling quick design

- This phase gives the clear idea about the development of software because the software is now built.
- It allows the developer to better understand the exact requirements.

4. Construction of prototype
The prototype is evaluated by the customer itself.

5. Deployment, delivery, feedback

- If the user is not satisfied with current prototype then it refines according to the requirements of the user.
- The process of refining the prototype is repeated until all the requirements of users are met.
- When the users are satisfied with the developed prototype then the system is developed on the basis of final prototype.

Advantages of Prototyping Model

- Prototype model need not know the detailed input, output, processes, adaptability of operating system and full machine interaction.
- In the development process of this model users are actively involved.
- The development process is the best platform to understand the system by the user.
- Errors are detected much earlier.
- Gives quick user feedback for better solutions.
- It identifies the missing functionality easily. It also identifies the confusing or difficult functions.

Disadvantages of Prototyping Model:

- The client involvement is more and it is not always considered by the developer.
- It is a slow process because it takes more time for development.
- Many changes can disturb the rhythm of the development team.
- It is a thrown away prototype when the users are confused with it.

2. The Spiral model

- Spiral model is a risk driven process model.
- It is used for generating the software projects.
- In spiral model, an alternate solution is provided if the risk is found in the risk analysis, then alternate solutions are suggested and implemented.
- It is a combination of prototype and sequential model or waterfall model.
- In one iteration all activities are done, for large project's the output is small.

The framework activities of the spiral model are as shown in the following figure.

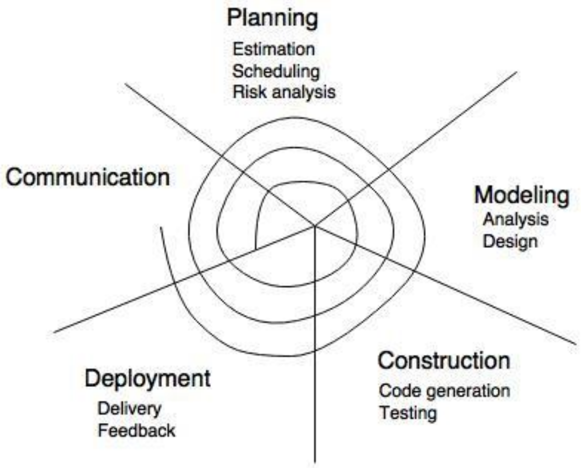


Fig. - The Spiral Model

NOTE: The description of the phases of the spiral model is same as that of the process model.

Advantages of Spiral Model

- It reduces high amount of risk.
- It is good for large and critical projects.
- It gives strong approval and documentation control.
- In spiral model, the software is produced early in the life cycle process.

Disadvantages of Spiral Model

- It can be costly to develop a software model.
- It is not used for small projects.

3. The concurrent development model

- The concurrent development model is called as concurrent model.
- The communication activity has completed in the first iteration and exits in the awaiting changes state.
- The modeling activity completed its initial communication and then go to the underdevelopment state.

- If the customer specifies the change in the requirement, then the modeling activity moves from the under development state into the awaiting change state.
- The concurrent process model activities moving from one state to another state.

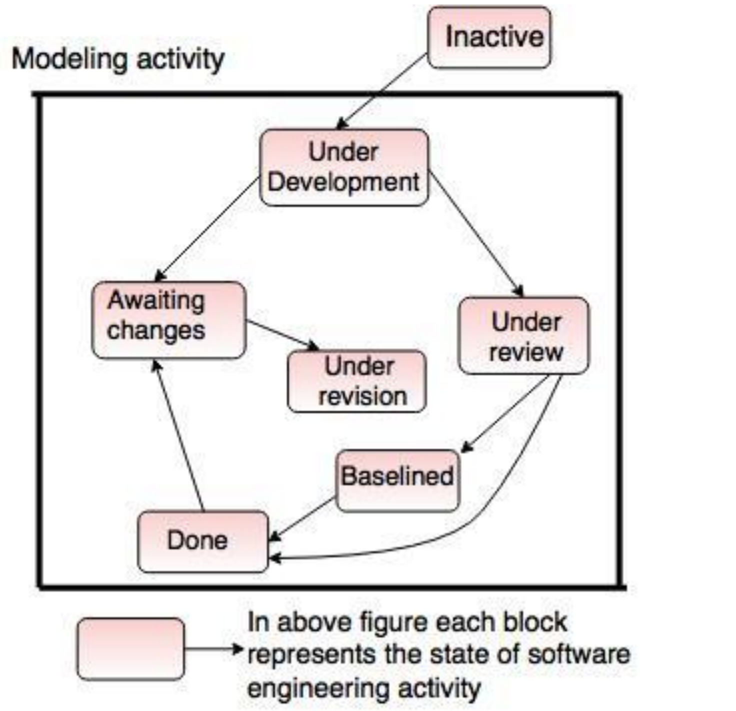


Fig. - One element of the concurrent process model

Advantages of the concurrent development model

- This model is applicable to all types of software development processes.
- It is easy for understanding and use.
- It gives immediate feedback from testing.
- It provides an accurate picture of the current state of a project.

Disadvantages of the concurrent development model

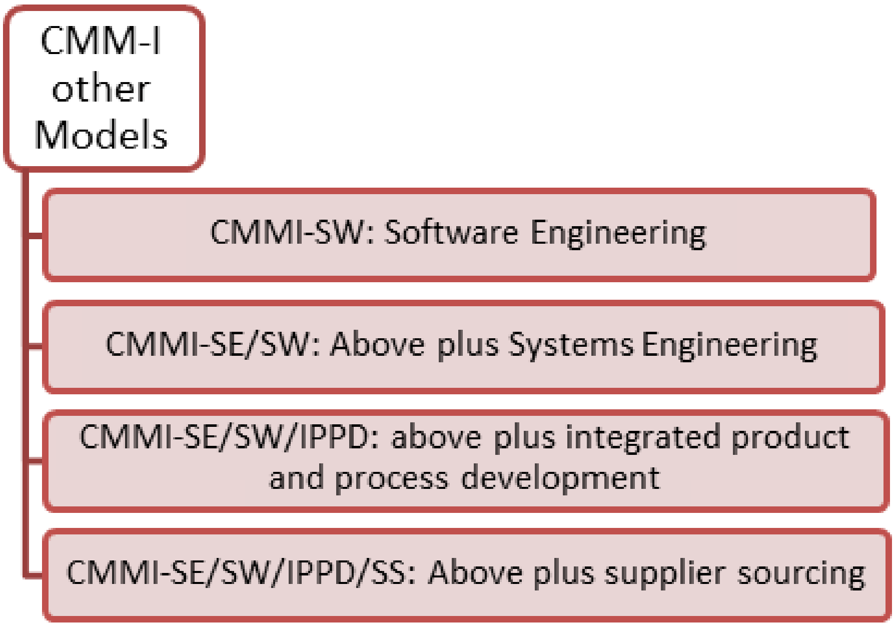
- It needs better communication between the team members. This may not be achieved all the time.
- It requires to remember the status of the different activities.

Capability Maturity Model Integration (CMMI).

What is CMM?

Capability Maturity Model is used as a benchmark to measure the maturity of an organization's software process.

CMM was developed at the Software engineering institute in the late 80's. It was developed as a result of a study financed by the U.S Air Force as a way to evaluate the work of subcontractors. Later based on the CMM-SW model created in 1991 to assess the maturity of software development, multiple other models are integrated with CMM-I they are



CMM-I other Models

CMMI-SW: Software Engineering

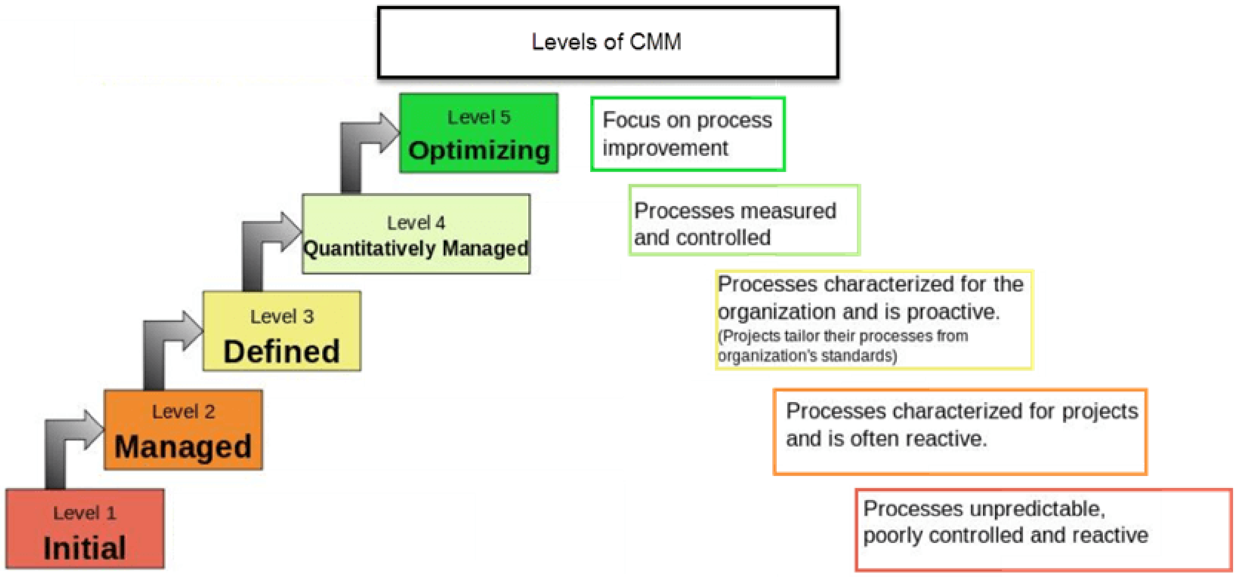
CMMI-SE/SW: Above plus Systems Engineering

CMMI-SE/SW/IPPD: above plus integrated product and process development

CMMI-SE/SW/IPPD/SS: Above plus supplier sourcing

What is Capability Maturity Model (CMM) Levels?

1. Initial
2. Repeatable/Managed
3. Defined
4. Quantitatively Managed
5. Optimizing


Levels of CMM
Level 5
Optimizing
Focus on process improvement
Level 4
Quantitatively Managed
Processes measured and controlled
Level 3
Defined
Processes characterized for the organization and is proactive.
(Projects tailor their processes from organization's standards)
Level 2
Managed
Processes characterized for projects and is often reactive.
Level 1
Initial
Processes unpredictable, poorly controlled and reactive

Level-1: Initial –

- No KPA's defined.
- Processes followed are adhoc and immature and are not well defined.
- Unstable environment for software dvelopment.
- No basis for predicting product quality, time for completion, etc.

Level-2: Repeatable –

- Focuses on establishing basic project management policies.
- Experience with earlier projects is used for managing new similar natured projects.

KPA's:

- Project Planning- It includes defining resources required, goals, constraints, etc. for the project. It presents a detailed plan to be followed systematically for successful completion of a good quality software.
- Configuration Management- The focus is on maintaining the performance of the software product, including all its components, for the entire lifecycle.
- Requirements Management- It includes the management of customer reviews and feedback which result in some changes in the requirement set. It also consists of accommodation of those modified requirements.
- Subcontract Management- It focuses on the effective management of qualified software contractors i.e. it manages the parts of the software which are developed by third parties.

- Software Quality Assurance- It guarantees a good quality software product by following certain rules and quality standard guidelines while development.

Level-3: Defined –

- At this level, documentation of the standard guidelines and procedures takes place.
- It is a well defined integrated set of project specific software engineering and management processes.

KPA's:

- Peer Reviews- In this method, defects are removed by using a number of review methods like walkthroughs, inspections, buddy checks, etc.
- Intergroup Coordination- It consists of planned interactions between different development teams to ensure efficient and proper fulfilment of customer needs.
- Organization Process Definition- It's key focus is on the development and maintenance of the standard development processes.
- Organization Process Focus- It includes activities and practices that should be followed to improve the process capabilities of an organization.
- Training Programs- It focuses on the enhancement of knowledge and skills of the team members including the developers and ensuring an increase in work efficiency.

Level-4: Managed –

- At this stage, quantitative quality goals are set for the organization for software products as well as software processes.
- The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.

KPA's:

- Software Quality Management- It includes the establishment of plans and strategies to develop a quantitative analysis and understanding of the product's quality.
- Quantitative Management- It focuses on controlling the project performance in a quantitative manner.

Level-5: Optimizing –

- This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.
- Use of new tools, techniques and evaluation of software processes is done to prevent recurrence of known defects.

KPA's:

- Process Change Management- Its focus is on the continuous improvement of organization's software processes to improve productivity, quality and cycle time for the software product.
- Technology Change Management- It consists of identification and use of new technologies to improve product quality and decrease the product development time.
- Defect Prevention- It focuses on identification of causes of defects and to prevent them from recurring in future projects by improving project defined process.

POSSIBLE QUESTIONS

PART B : 2 MARK QUESTIONS

1. Define software engineering.
2. Differentiate between software and hardware characteristics
3. List the major disadvantages of Waterfall model.
4. List the disadvantage of incremental model.
5. What is CMMI? List the five maturity levels of CMMI.

PART C: 8 MARK QUESTIONS

1. Explain in detail about Concurrent Development model.
2. Discuss in detail about Spiral Model.
3. Discuss in detail about the umbrella activities in the software process framework.
4. Illustrate Prototyping model with its phases.
5. Explain in detail about Waterfall Model with a neat sketch.
6. Discuss in detail about the Layered perspective of software engineering.
7. Describe in detail the changes acquired in the nature of software.
8. Discuss in detail about the Rapid Application Development Model with a neat sketch.
9. Explain in detail about Incremental Process Model.
10. Explain in detail about Prescriptive model.

KARPAGAM ACADEMY OF HIGHER EDUCATION

Enable | Enlighten | Enrich
KARPAGAM
ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)

Department of Computer Science

II B.Sc(CS)   (BATCH 2018-2021)   IV SEMESTER

SOFTWARE ENGINEERING (18CSU402)

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

UNIT II

S.NO	QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	ANSWER
1	_____________is a process of discovery, refinement, modeling, and specification	software engineering	software requirement engineering	software analysis	software design	software engineering
2	_________is the systematic use of proven principles, techniques, languages, and tools.	software engineering	software analysis	software design	requirements engineering	requirements engineering
3	Requirement engineering is conducted in a ______	sporadic way	random way	haphazard way	systematic use of proven approaches	systematic use of proven approaches
4	Software requirements analysis work products must be reviewed for ________	modeling	completeness	information processing	functional requirement	completeness
5	______ bridges the gap between system level requirement engineering and software design	system engineering	modeling	requirements analysis	software engineering	software engineering
6	Software requirements analysis is divided into _____ areas of effort.	2	3	4	5	4
7	Throughout evaluation and solution synthesis, the analyst's primary focus is on _________	when	where	what	how	what

8	Software applications can be collectively called as __________	data gathering	information gathering	data processing	information processing	data processing
9	________ represents the individual data and control objects that constitute some larger collection of information transformed by the software	information content	data content	data model	information model	information content
10	__________ represents the manner in which data and control change as each moves through a system.	information content	information flow	information structure	data structure	information flow
11	______ represents the internal organization of various data and control items.	information content	information flow	information structure	data structure	information structure
12	Entity is a ________	data	information	model	physical thing	physical thing
13	The first operational analysis principle requires an examination of the information domain and the creation of a ______	data model	information model	data structure	information structure	data model
14	To transform software into information, the system performs ______	input	processing	output	input, processing and output	input, processing and output
15	To transform software into information, the system must perform _________ generic functions.	2	3	4	5	3
16	There are ____________ types of models.	5	4	3	2	2
17	The horizontal partitioning of SafeHome function has _________ major functions on the first level of hierarchy	2	3	4	5	3
18	The vertical partitioning of SafeHome function has _______ major functions on the first level of hierarchy.	2	3	4	5	3

19	A model of the software to be built is called ________	data model	prototype	information model	software model	prototype
20	The________ of software requirements presents the real world manifestation of processing functions and information strucutres	implementati on view	essential view	partitioning view	evolutionary view	implementation view
21	The essential view of the SafeHome function_______ does not concern itself with the physical form of the data that is used.	identify event	read sensor status	activate sensor	deactivate sensor	read sensor status
22	A prototype is the _________	data model	information model	software model	evolution model	software model
23	Data objects are represented by _______	labeled arrows	bubbles	entity	label	labeled arrows
24	Transformations are represented by __________	labeled arrows	bubbles	entity	label	bubbles
25	__________ enables the software engineer to generate executable code quickly, they are ideal for rapid prototyping.	2 GT	3 GT	4 GT	5 GT	4 GT
26	The ________ provides a detailed description of the problem that the software must solve.	information description	software scope	function description	software description	information description
27	_______ is probably the most important and, ironically, the most often neglected section of software requirements specification.	behavioural description	processing narrative	overall structure	validation criteria	validation criteria
28	The software requirements specification includes _______	bibliography	appendix	Bibliography and appendix	review	Bibliography and appendix

29	The __________ section of the specification examines the operation of the software as a consequence of external events and internally generated control characteristics.	behavioural description	representation format	specification principles	prototyping environment	behavioural description
30	The software requirements specification is developed as a consequence of ________	review	analysis	prototyping	functional description	analysis
31	The preliminary user's manual presents the software as a _______	white box	black box	machine interface	prototype	black box
32	__________is the first technical step in the software process	requirements analysis	requirements specification	information description	information domain	requirements analysis
33	The close ended approach of the prototyping paradigm is called _______	evolutionary prototyping	simply prototyping	open ended prototyping	throwaway prototyping	throwaway prototyping
34	The information domain contains ________ different views of the data and control as each is processed by a computer program.	2	3	4	5	3
35	The content of ________ is defined by the attributes that are needed to create it.	system status	functional model	paycheck	behavioural model	paycheck
36	Building data, functional and behavioural models provide the software engineer with ________ different views	5	4	3	2	3
37	The description of each function required to solve the problem is presented in the ________	functional description	behavioural description	data description	program description	functional description
38	Software requirements analysis work products must be reviewed for ________	clarity	completeness	consistency	all of the above	all of the above
39	The overall role of software in a larger system is identified during the ___________	system engineering	software planning	software estimation	documentation	system engineering
40	The analyst finds that problems with the current manual system include________	inability to obtain the status of a	two-or-three day turn around to	multiple reorders to the same	all of the above	all of the above

41	The expansion of FAST is ________	Facilitated Application Specification	Fast Application Specification	Facilitated Application Software	Facilitated Application System	Facilitated Application Specification
42	The following come under the lists of constraints __________	cost	size	business rules	all of the above	all of the above
43	All analysis methods are related by a set of operational _________	system	software	principles	analysis	principles
44	The functions that the software is to perform must be ________	defined	described	discussed	listed	defined
45	The first step in establishing traceability back to the customer is ________	use multiple views of requirement	rank requirement	record the origin of and the reason for	work to eliminate ambiguity	record the origin of and the reason for
46	_______are used so that the characteristics of function and behaviour can be communicated in a compact fashion.	softwares	models	programs	none of the above	models
47	Requirement analysis allows the software engineer someties called an _____	analyst	team manager	progject manager	software engineer	analyst
48	The primary focus on requirement analyis is on	what not how	what	how	functions	what not how
49	The analysis model must achieve _____ primary objectives	3	2	5	4	3
50	Analysis modeling is of 2 types _____	3	2	5	4	2
51	Keep the model as it can be , the rule is sated in	analysis requiremets	waterfall model	incremental model	spiral model	analysis requiremets
52	A second approach in requirement analysis is _____	object oriented analysis	structured analysis	design	implementation	object oriented analysis

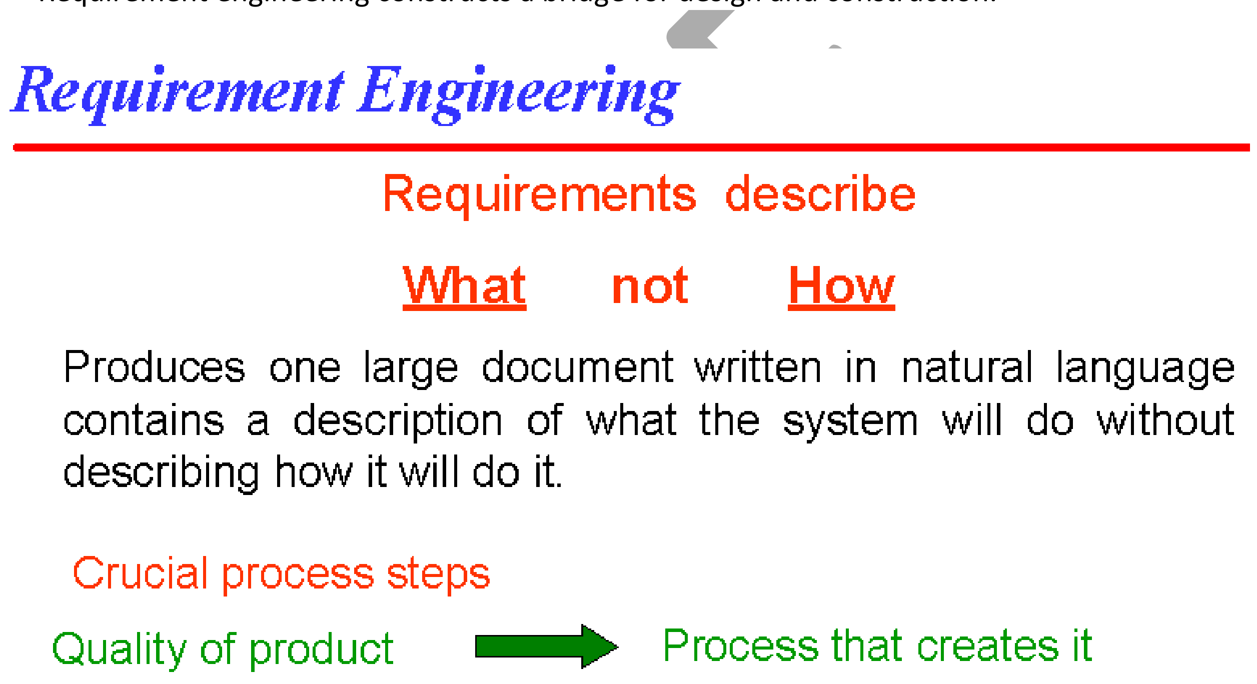
53	The software engineer or analyst defines all _____ that are processed within the system	data objects	cardinality and modality	data atttibutes	relationships	data objects
54	________ defines the properties of data objects	data objects	cardinality and modality	data atttibutes	relationships	data atttibutes
55	_______ modeling continues to be one of the most widely used analysis notations today	data flow diagram	use case diagram	context diagram	leve 0	data flow diagram
56	The control specification stands for	CSPEC	CPEC	CNPEC	CTPEEC	CSPEC
57	The process specification stands for	PSPEC	SPECP	RSPEC	SPEC	PSPEC
58	________ model indicates how software will respond to external events	behavioral	analysis	waterfall	incremental	behavioral
59	UML stands for ________	unified modern language	uniform modeling language	unified model language	unified modern linear	unified modern language
60	_______ represents a sequence of activities that involves actors and the system	use-case	dfd	waterfall	rapid model	use-case

UNIT II

Requirement Analysis; Initiating Requirement Engineering Process- Requirement Analysis and Modeling Techniques- Flow Oriented Modeling- Need for SRS- Characteristics and Components of SRS- Software Project Management: Estimation in Project Planning Process, Project Scheduling.

Introduction to requirement engineering

- The process of collecting the software requirement from the client then understand, evaluate and document it is called as requirement engineering.
- Requirement engineering constructs a bridge for design and construction.



Requirement Engineering

Requirements describe

What not How

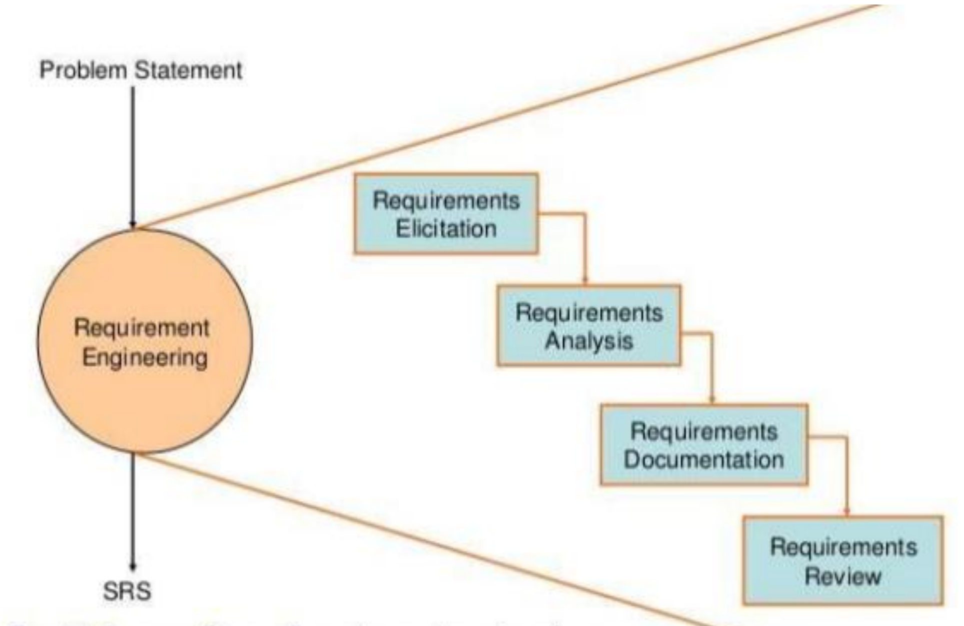
Produces one large document written in natural language contains a description of what the system will do without describing how it will do it.

Crucial process steps

Quality of product → Process that creates it

Requirement engineering consists of seven different tasks as follow:

Requirement Engineering is the disciplined application of proven principles, methods, tools, and notations to describe a proposed system's intended behavior and its associated constraints.



Crucial Process Steps of requirement engineering

1. Inception

- Inception is a task where the requirement engineering asks a set of questions to establish a software process.
- In this task, it understands the problem and evaluates with the proper solution.
- It collaborates with the relationship between the customer and the developer.
- The developer and customer decide the overall scope and the nature of the question.

2. Elicitation

Elicitation means to find the requirements from anybody. The requirements are difficult because the **following problems occur in elicitation**.

Problem of scope: The customer give the unnecessary technical detail rather than clarity of the overall system objective.

Problem of understanding: Poor understanding between the customer and the developer regarding various aspect of the project like capability, limitation of the computing environment.

Problem of volatility: In this problem, the requirements change from time to time and it is difficult while developing the project.

3. Elaboration

- In this task, the information taken from user during inception and elaboration and are expanded and refined in elaboration.
- Its main task is developing pure model of software using functions, feature and constraints of a software.

4. Negotiation

- In negotiation task, a software engineer decides the how will the project be achieved with limited business resources.
- To create rough guesses of development and access the impact of the requirement on the project cost and delivery time.

5. Specification

- In this task, the requirement engineer constructs a final work product.
- The work product is in the form of software requirement specification.
- In this task, formalize the requirement of the proposed software such as informative, functional and behavioral.
- The requirement are formalize in both graphical and textual formats.

6. Validation

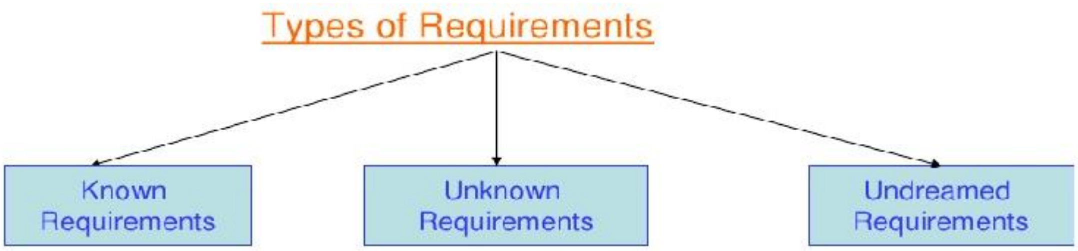
- The work product is built as an output of the requirement engineering and that is accessed for the quality through a validation step.
- The formal technical reviews from the software engineer, customer and other stakeholders helps for the primary requirements validation mechanism.

7. Requirement management

- It is a set of activities that help the project team to identify, control and track the requirements and changes can be made to the requirements at any time of the ongoing project.
- These tasks start with the identification and assign a unique identifier to each of the requirement.

- After finalizing the requirement traceability table is developed.
- The examples of traceability table are the features, sources, dependencies, subsystems and interface of the requirement.

Types of Requirements


Types of Requirements

Known Requirements

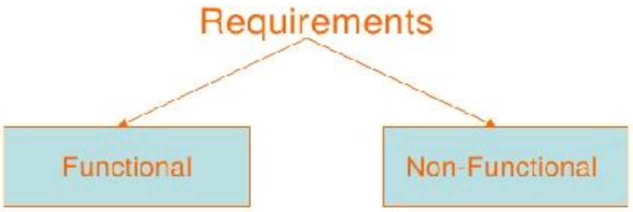
Unknown Requirements

Undreamed Requirements

Stakeholder: Anyone who should have some direct or indirect influence on the system requirements.

--- User

--- Affected persons


Requirements

Functional

Non-Functional

Functional requirements describe what the software has to do. They are often called product features.

Non Functional requirements are mostly quality requirements. That stipulate how well the software does, what it has to do.

Availability
Reliability
Usability
Flexibility
} For Users

Maintainability
Portability
Testability
} For Developers

Feasibility Study

Is cancellation of a project a bad news? As per IBM report, "31% projects get cancelled before they are completed, 53% over-run their cost estimates by an average of 189% & for every 100 projects, there are 94 restarts.

How do we cancel a project with the least work?

➡ CONDUCT A FEASIBILTY STUDY

Technical feasibility

- Is it technically feasible to provide direct communication connectivity through space from one location of globe to anotherlocation?
- Is it technically feasible to design a programming language using"Sanskrit"?

Feasibility depends upon non technical Issues like:

- Are the project's cost and schedule assumptionrealistic?
- Does the business modelrealistic?
- Is there any market for theproduct?

Purpose of feasibility study

"evaluation or analysis of the potential impact of a proposed project or program."

Focus of feasibility studies

- Is the product conceptviable?
- Will it be possible to develop a product that matches the project's vision statement?
- What are the current estimated cost and schedule for the project?

Eliciting Requirements

Eliciting requirement helps the user for collecting the requirement

Eliciting requirement steps are as follows:

1. Collaborative requirements gathering

- Gathering the requirements by conducting the meetings between developer and customer.
- Fix the rules for preparation and participation.
- The main motive is to identify the problem, give the solutions for the elements, negotiate the different approaches and specify the primary set of solution requirements in an environment which is valuable for achieving goal.

2. Quality Function Deployment (QFD)

- In this technique, translate the customer need into the technical requirement for the software.
- QFD system designs a software according to the demands of the customer.

QFD consist of three types of requirement:

Normal requirements

- The objective and goal are stated for the system through the meetings with the customer.
- For the customer satisfaction these requirements should be there.

Expected requirement

- These requirements are implicit.
- These are the basic requirement that not be clearly told by the customer, but also the customer expect that requirement.

Exciting requirements

- These features are beyond the expectation of the customer.
- The developer adds some additional features or unexpected feature into the software to make the customer more satisfied.

For example, the mobile phone with standard features, but the developer adds few additional functionalities like voice searching, multi-touch screen etc. then the customer more exited about that feature.

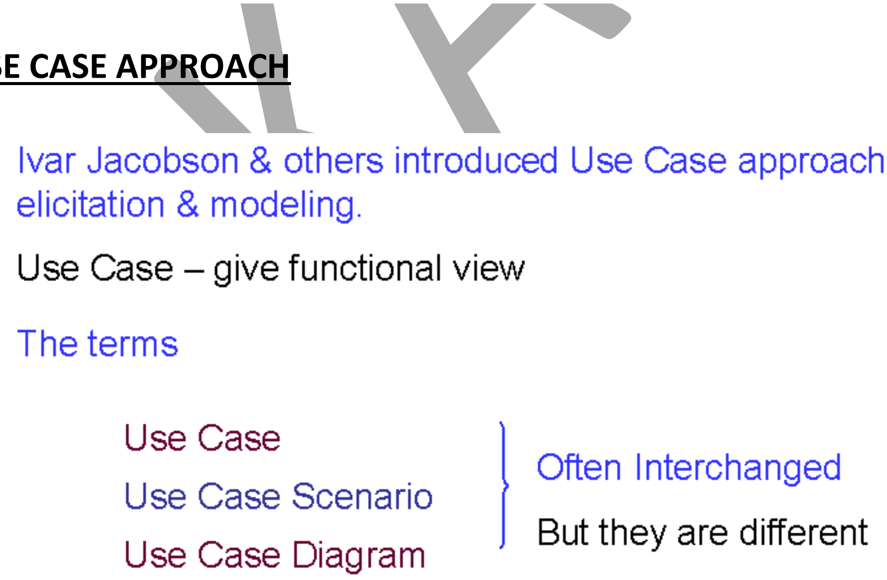
3. Usage scenarios

- Till the software team does not understand how the features and function are used by the end users it is difficult to move technical activities.
- To achieve above problem the software team produces a set of structure that identify the usage for the software.
- This structure is called as 'Use Cases'.

4. Elicitation work product

- The work product created as a result of requirement elicitation that is depending on the size of the system or product to be built.
- The work product consists of a statement need, feasibility, statement scope for the system.
- It also consists of a list of users participate in the requirement elicitation.
- Analysis model operates as a link between the 'system description' and the 'design model'.
- In the analysis model, information, functions and the behaviour of the system is defined and these are translated into the architecture, interface and component level design in the 'design modeling'.

USE CASE APPROACH



Ivar Jacobson & others introduced Use Case approach for elicitation & modeling.

Use Case – give functional view

The terms

Use Case

Use Case Scenario

Use Case Diagram

Often Interchanged

But they are different

Usecase Scenarios are unstructured descriptions of user requirements.

Use case diagrams are graphical representations that

may be decomposed into further levels of abstraction.

Components of Use Case approach

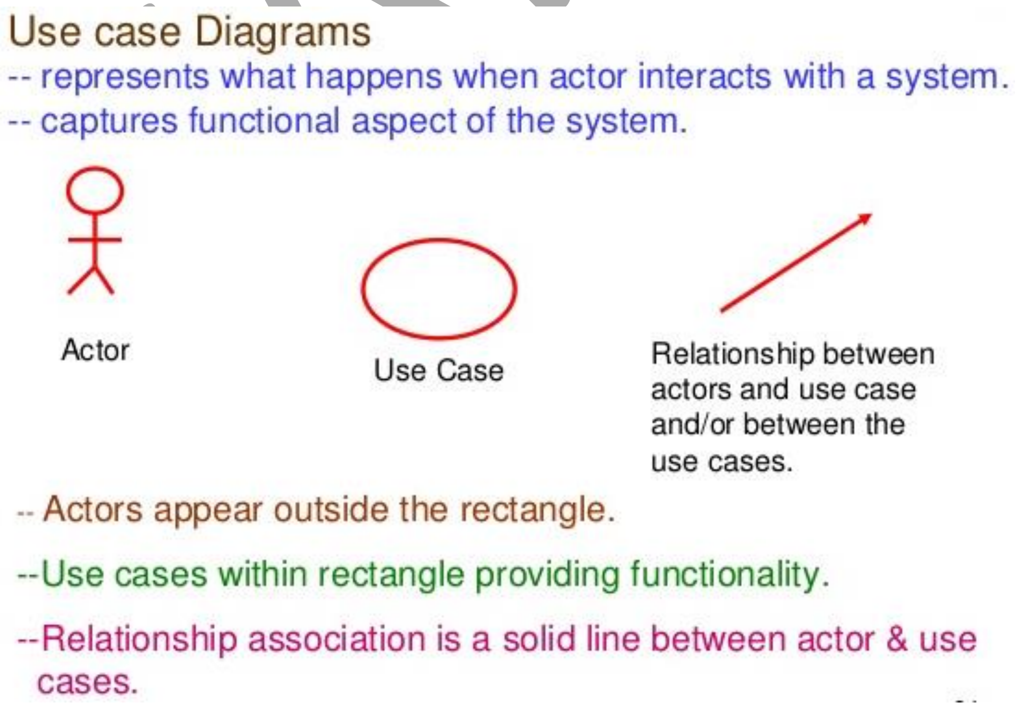
Actor:

An actor or external agent, lies outside the system model, but interacts with it in some way.

Use cases should not be used to capture all the details of the system.

- Only significant aspects of the required functionality
- No design issues
- Use Cases are for "what" the system is , not "how" the system will be designed
- Free of design characteristics

1.



Use case Diagrams

-- represents what happens when actor interacts with a system.

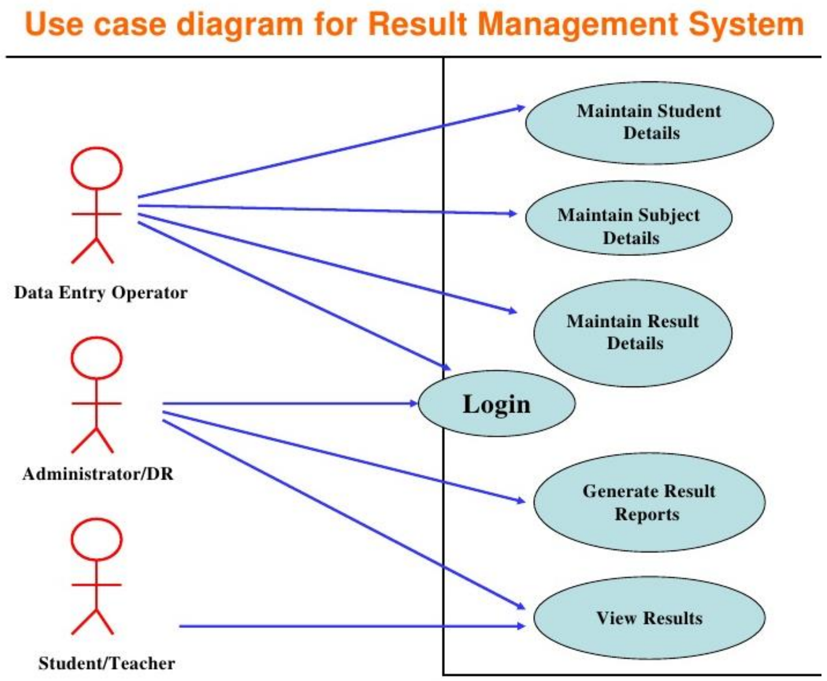
-- captures functional aspect of the system.

Actor
Use Case
Relationship between actors and use case and/or between the use cases.

-- Actors appear outside the rectangle.

--Use cases within rectangle providing functionality.

--Relationship association is a solid line between actor & use cases.



1. Maintain studentDetails

    Add/Modify/update students details like name, address.

2. Maintain subjectDetails

    Add/Modify/Update Subject information semester Wise

3. Maintain Result Details

    Include entry of marks and assignment of credit points for each paper.

4. Login

    Use to Provide way to enter through user id & password.

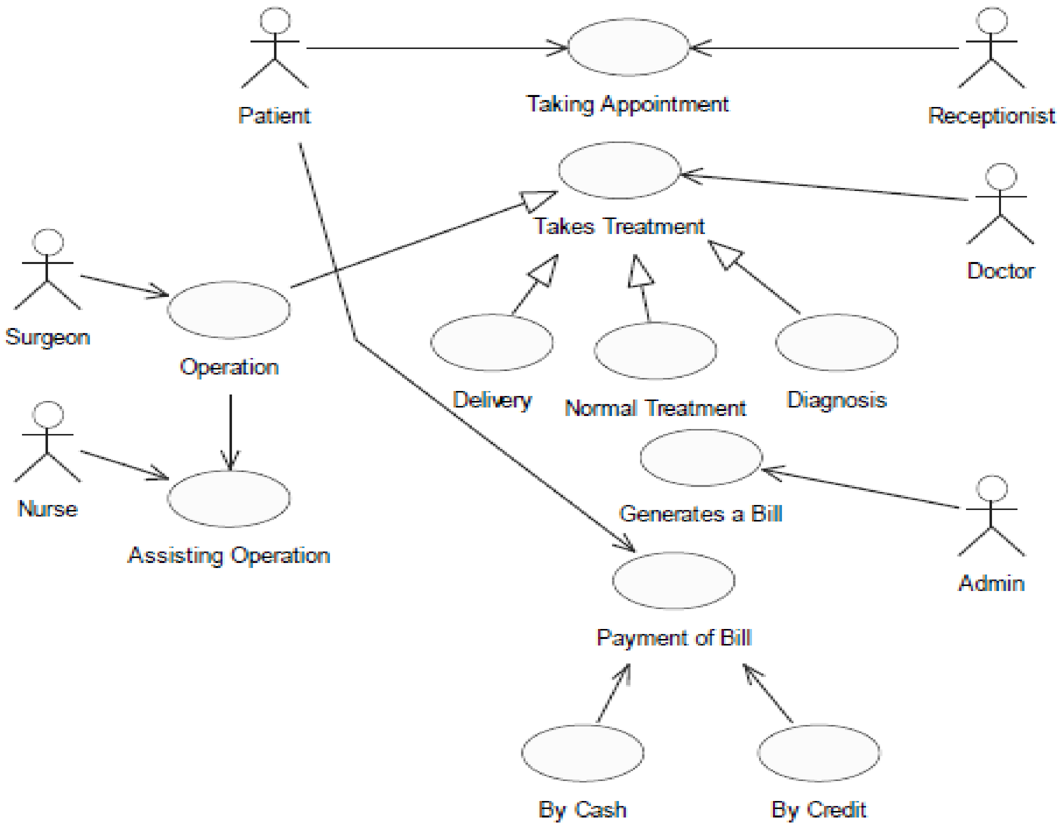
5. Generate Result Report

6. View Result

    According to coursecode

    According to Enrollment number/rollnumber

Use –Case Diagram : Hospital Management system



Requirements Analysis

Elements of the analysis model

1. Scenario based element

- This type of element represents the system user point of view.
- Scenario based elements are use case diagram, user stories.

2. Class based elements

- The object of this type of element manipulated by the system.
- It defines the object,attributes and relationship.
- The collaboration is occurring between the classes.
- Class based elements are the class diagram, collaboration diagram.

3. Behavioral elements

- Behavioral elements represent state of the system and how it is changed by the external events.
- The behavioral elements are sequenced diagram, state diagram.

4. Flow oriented elements

- An information flows through a computer-based system it gets transformed.
- It shows how the data objects are transformed while they flow between the various system functions.
- The flow elements are data flow diagram, control flow diagram.

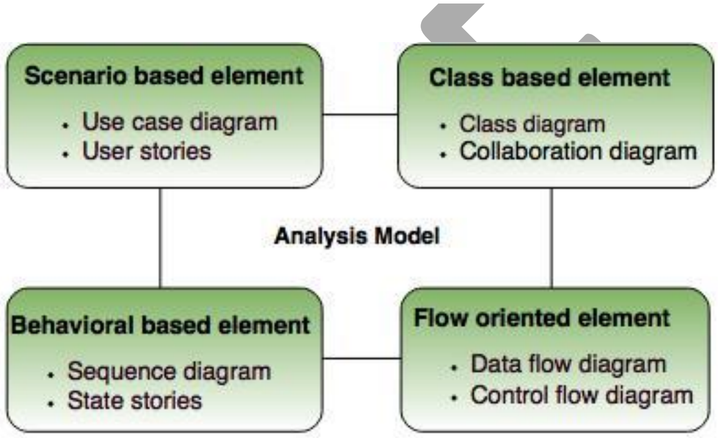


Fig. - Elements of analysis model

<u>Analysis Rules of Thumb</u>

The rules of thumb that must be followed while creating the analysis model.

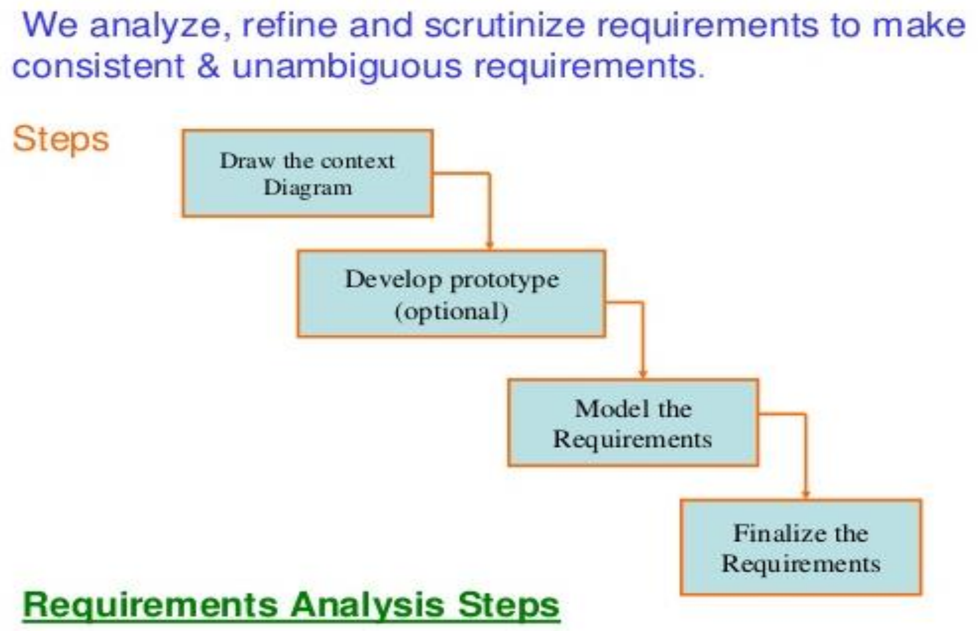
The rules are as follows:

- The model focuses on the requirements in the business domain. The level of abstraction must be high i.e there is no need to give details.
- Every element in the model helps in understanding the software requirement and focus on the information, function and behaviour of the system.
- The consideration of infrastructure and nonfunctional model delayed in the design.

For example, the database is required for a system, but the classes, functions and behavior of the database are not initially required. If these are initially considered then there is a delay in the designing.

- Throughout the system minimum coupling is required. The interconnections between the modules is known as 'coupling'.
- The analysis model gives value to all the people related to model.
- The model should be simple as possible. Because simple model always helps in easy understanding of the requirement.

Analysis model



We analyze, refine and scrutinize requirements to make consistent & unambiguous requirements.

Steps

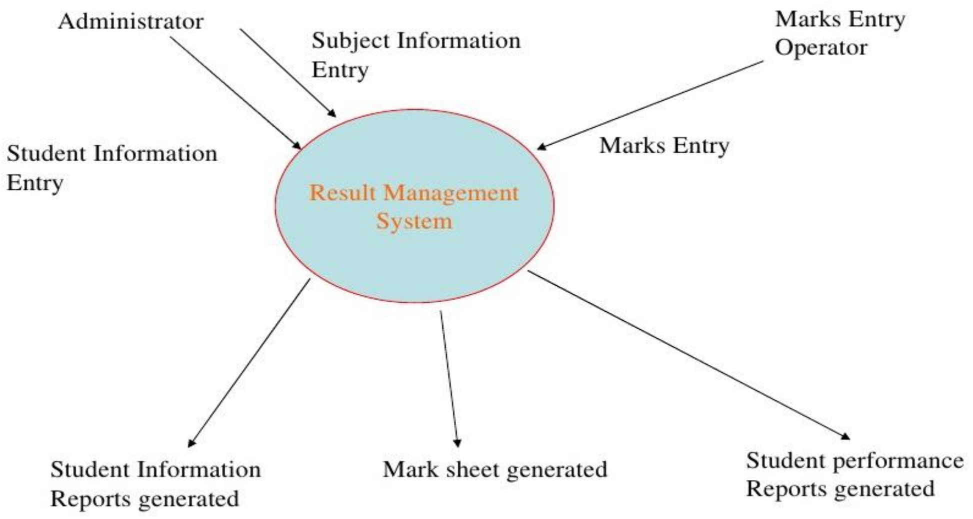
Draw the context Diagram

Develop prototype (optional)

Model the Requirements

Finalize the Requirements

Requirements Analysis Steps


Administrator
Subject Information Entry
Marks Entry Operator
Student Information Entry
Marks Entry
Result Management System
Student Information Reports generated
Mark sheet generated
Student performance Reports generated

Flow Oriented Modeling

It shows how data objects are transformed by processing the function.

The Flow oriented elements are:

i. Data flow model

- It is a graphical technique. It is used to represent information flow.
- The data objects are flowing within the software and transformed by processing the elements.
- The data objects are represented by labeled arrows. Transformation are represented by circles called as bubbles.
- DFD shown in a hierarchical fashion. The DFD is split into different levels. It also called as 'context level diagram'.

ii. Control flow model

- Large class applications require a control flow modeling.
- The application creates control information instated of reports or displays.
- The applications process the information in specified time.
- An event is implemented as a boolean value.

For example, the boolean values are true or false, on or off, 1 or 0.

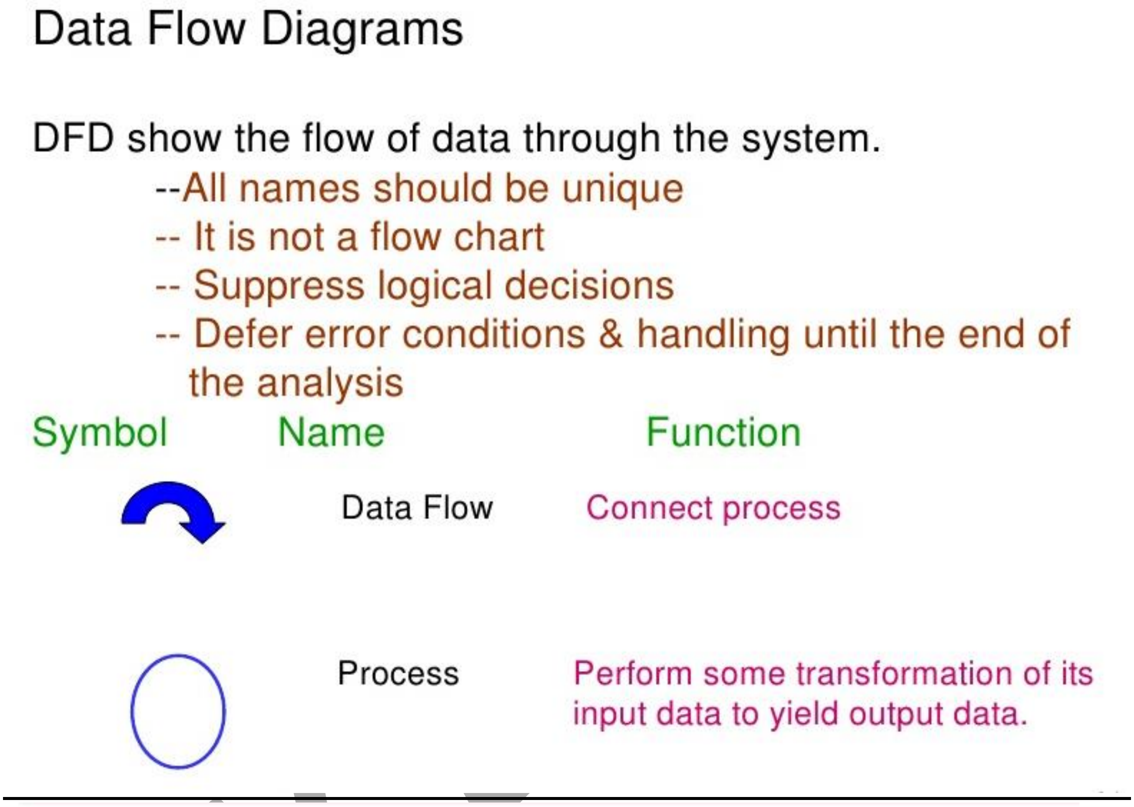
iii. Control Specification

- A short term for control specification is CSPEC.
- It represents the behaviour of the system.
- The state diagram in CSPEC is a sequential specification of the behaviour.
- The state diagram includes states, transitions, events and activities.
- State diagram shows the transition from one state to another state if a particular event has occurred.

iv. Process Specification

- A short term for process specification is PSPEC.
- The process specification is used to describe all flow model processes.

- The content of process specification consists narrative text, Program Design Language(PDL) of the process algorithm, mathematical equations, tables or UML activity diagram.



Data Flow Diagrams

DFD show the flow of data through the system.

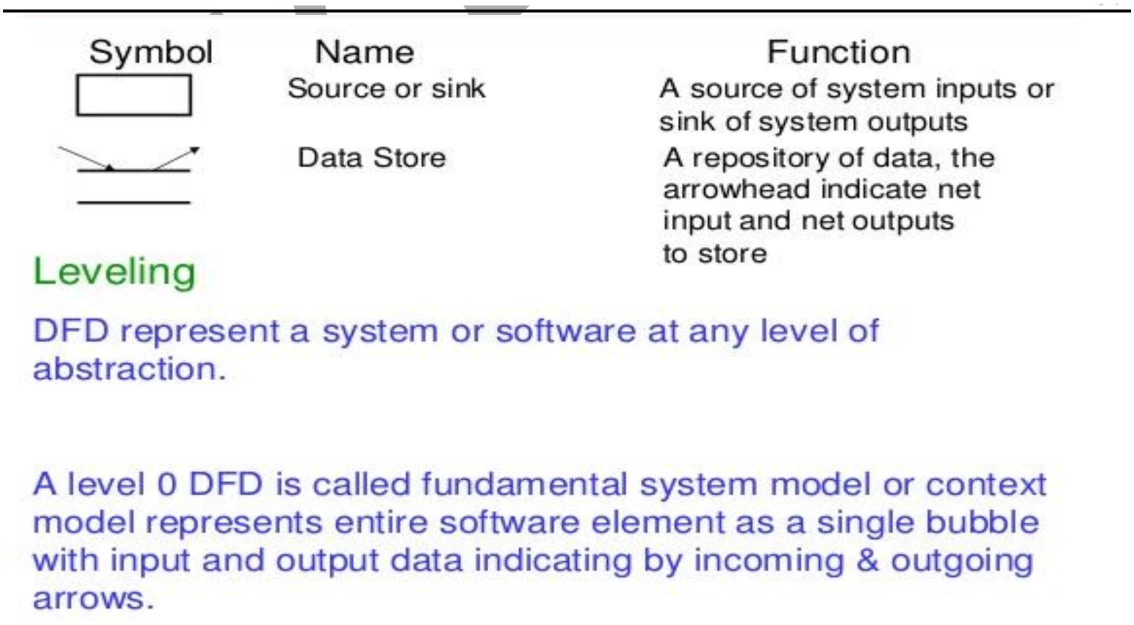
--All names should be unique

-- It is not a flow chart

-- Suppress logical decisions

-- Defer error conditions & handling until the end of the analysis

Symbol	Name	Function
	Data Flow	Connect process
	Process	Perform some transformation of its input data to yield output data.



Symbol	Name	Function
	Source or sink	A source of system inputs or sink of system outputs
	Data Store	A repository of data, the arrowhead indicate net input and net outputs to store

Leveling

DFD represent a system or software at any level of abstraction.

A level 0 DFD is called fundamental system model or context model represents entire software element as a single bubble with input and output data indicating by incoming & outgoing arrows.

Data objects

- The data object is the representation of composite information.
- The composite information means an object has a number of different properties or attribute.

For example, Height is a single value so it is not a valid data object, but dimensions contain the height, the width and depth these are defined as an object.

Data Attributes

Each of the data object has a set of attributes.

Data object has the following characteristics:

- Name an instance of the data object.
- Describe the instance.
- Make reference to another instance in another table.

Relationship

Relationship shows the relationship between data objects and how they are related to each other.

Cardinality

Cardinality state the number of events of one object related to the number of events of another object.

The cardinality expressed as:

One to one (1:1)

One event of an object is related to one event of another object.
For example, one employee has only one ID.

One to many (1:N)

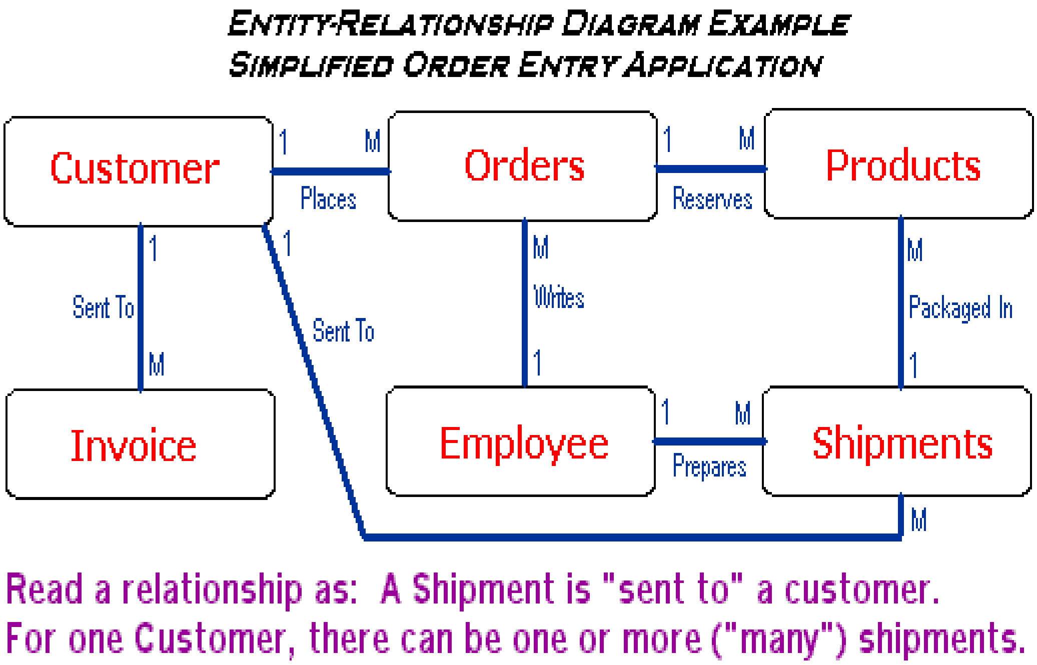
One event of an object is related to many events.
For example, One collage has many departments.

Many to many(M:N)

Many events of one object are related to many events of another object.
For example, many customer place order for many products.

Modality

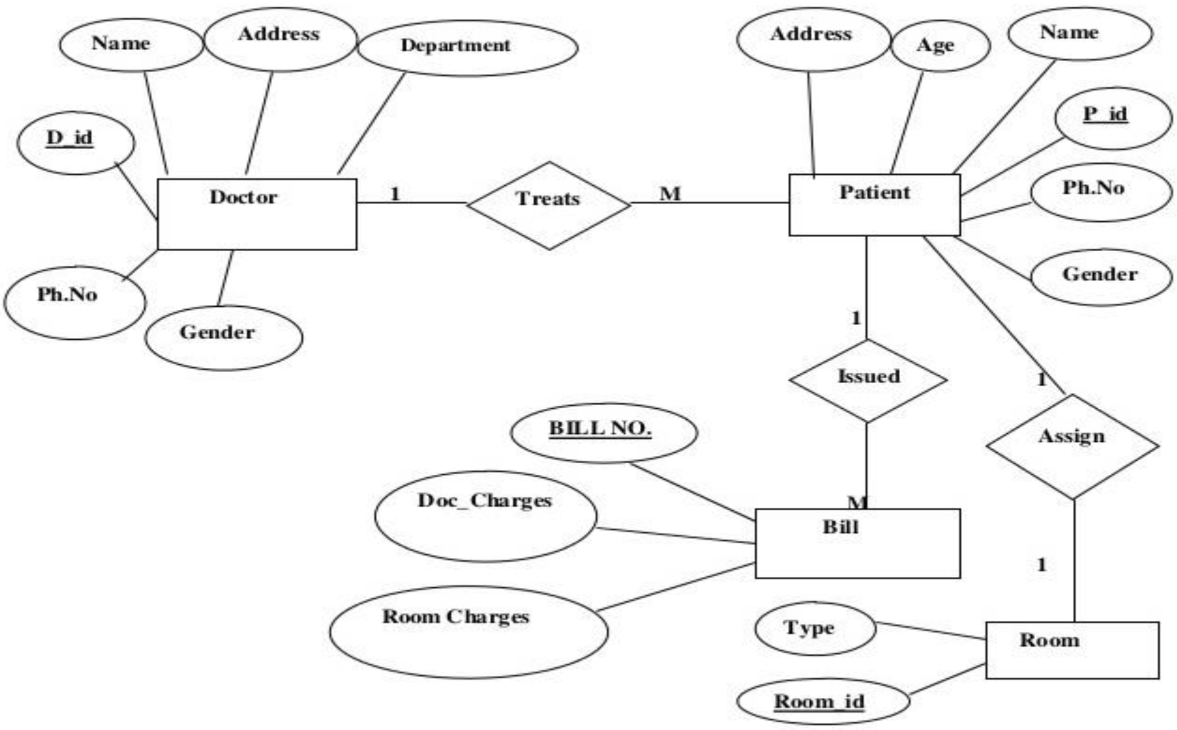
- If an event relationship is an optional then the modality of relationship is zero.
- If an event of relationship is compulsory then modality of relationship is one.



ENTITY-RELATIONSHIP DIAGRAM EXAMPLE
SIMPLIFIED ORDER ENTRY APPLICATION

Read a relationship as: A Shipment is "sent to" a customer.
For one Customer, there can be one or more ("many") shipments.

E-R Diagram of Hospital Management System



Software Requirement Specification (SRS)

- The requirements are specified in specific format known as SRS.
- This document is created before starting the development work.
- The software requirement specification is an official document.
- It shows the detail about the performance of expected system.
- SRS indicates to a developer and a customer what is implemented in the software.
- SRS is useful if the software system is developed by the outside contractor.
- SRS must include an interface, functional capabilities, quality, reliability, privacy etc.

Characteristics of SRS

Correctness:
User review is used to ensure the correctness of requirements stated in the SRS. SRS is said to be correct if it covers all the requirements that are actually expected from the system.

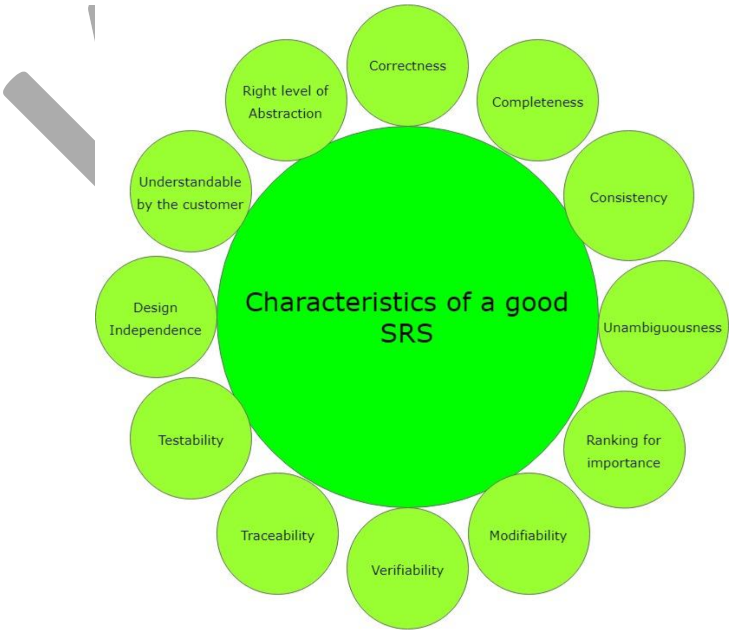
Completeness:
Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.

Consistency:
Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements. Examples of conflict include differences in terminologies used at separate places, logical conflicts like time period of report generation, etc.

Unambiguousness:
An SRS is said to be unambiguous if all the requirements stated have only 1 interpretation. Some of the ways to prevent unambiguousness include the use of modelling techniques like ER diagrams, proper reviews and buddy checks, etc.

Ranking for importance and stability:
There should a criterion to classify the requirements as less or more important or more specifically as desirable or essential. An identifier mark can be used with every requirement to indicate its rank or stability.

Modifiability:
SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent. Modifications should be properly indexed and cross-referenced.



Verifiability:
An SRS is verifiable if there exists a specific technique to quantifiably measure the extent to which every requirement is met by the system. For example, a requirement stating that the system must be user-friendly is not verifiable and listing such requirements should be avoided.

Traceability:
One should be able to trace a requirement to a design component and then to a code segment in the program. Similarly, one should be able to trace a requirement to the corresponding test cases.

Design Independence:
There should be an option to choose from multiple design alternatives for the final system. More specifically, the SRS should not include any implementation details.

Testability:
An SRS should be written in such a way that it is easy to generate test cases and test plans from the document.

Understandable by the customer:
An end user maybe an expert in his/her specific domain but might not be an expert in computer science. Hence, the use of formal notations and symbols should be avoided to as much extent as possible. The language should be kept easy and clear.

Right level of abstraction:
If the SRS is written for the requirements phase, the details should be explained explicitly. Whereas, for a feasibility study, fewer details can be used. Hence, the level of abstraction varies according to the purpose of the SRS.

Software Project Management:

The job pattern of an IT company engaged in software development can be seen split in two parts:

- Software Creation
- Software Project Management

A project is well-defined task, which is a collection of several operations done in order to achieve a goal (for example, software development and delivery). A Project can be characterized as:

- Every project may has a unique and distinct goal.
- Project is not routine activity or day-to-day operations.
- Project comes with a start time and end time.
- Project ends when its goal is achieved hence it is a temporary phase in the lifetime of an organization.
- Project needs adequate resources in terms of time, manpower, finance, material and knowledge-bank.

Software Project Management

A software project manager is the most important person inside a team who takes the overall responsibilities to manage the software projects and play 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

Project planning

Project planning is undertaken immediately after the feasibility study phase and before the starting of the requirement analysis and specification phase. Once a project has been found to be feasible, Software project managers started project planning. Project planning is completed before any development phase starts. Project planning involves estimating several characteristics of a project and then plan the project activities based on these estimations. Project planning is done with most care and attention. A wrong estimation can result in schedule slippage. Schedule delay can cause customer dissatisfaction, which may lead to a project failure. For effective project planning, in addition to a very good knowledge of various estimation techniques, past experience is also very important. During the project planning the project manager performs the following activities:

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.

   The effectiveness of all later planning activities is dependent on the accuracy of these three estimations.

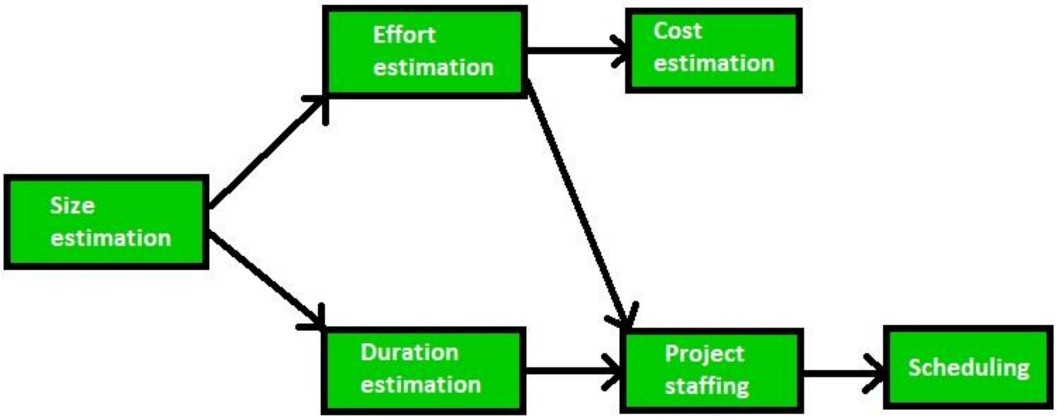
2. **Scheduling:** After completion of estimation of all the project parameters, scheduling for manpower and other resources are done.

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, analysis the damage might cause these risks and take risk reduction plan to cope up with these risks.

5. **Miscellaneous plans:** This includes making several other plans such as quality assurance plan, configuration management plan, etc.

The order in which the planning activities are undertaken is shown in the below figure:



Project monitoring and control

Project monitoring and control activities are undertaken once the development activities start. The main focus of project monitoring and control activities is to ensure that the software development proceeds as per plan. This includes checking whether the project is going on as per plan or not if any problem created then the project manager must take necessary action to solve the problem.

Role of a software project manager: There are many roles of a project manager in the development of software.

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 task.

Motivate the team-member: One of the key roles of a software project manager is to encourage team member 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 necessary action to solve the problem. Moreover, check whether the product is developed by maintaining correct coding standards or not.

Liaison: 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 requirements or not.

Documenting project report: The project manager prepares the documentation of the project for future purpose. 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.

Necessary skills of software project manager: A good theoretical knowledge of various project management technique is needed to become a successful project manager, but only theoretical knowledge is not enough. Moreover, a project manager must have good decision-making abilities, good communication skills and the ability to control the team members with keeping a good rapport with them and the ability to get the work done by them. Some skills such as tracking and controlling the progress of the project, customer interaction, good knowledge of estimation techniques and previous experience are needed.

Skills that are the most important to become a successful project manager are given below:

- Knowledge of project estimation techniques
- Good decision-making abilities at the right time
- Previous experience of managing a similar type of projects
- Good communication skill 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 for these problems

POSSIBLE QUESTIONS

PART B : 2 MARK QUESTIONS

1. What is requirement engineering process?
2. List the analysis rules of thumb.
3. Draw the context level DFD of Safe Home Alarm System.
4. What are the four major approaches used in analysis modeling
5. Differentiate between cardinality and modality

PART C: 8 MARK QUESTIONS

1. Illustrate the Guidelines for drawing a DFD and explain it with an example.
2. Describe Sequence diagram with an example.
3. Compare and contrast Process Specification and Control Specification.
4. Illustrate the identification of Events with Use-Case while creating a Behavioral Model.
5. Write Short notes on Attributes and Relationships.
6. Give a detailed note on active state and passive state.

7. Discuss the Control Specification in Flow Oriented Modeling.
8. Narrate the steps involved in creating a behavioral model.
9. Describe in detail about analysis modeling concepts.
10. Elucidate Requirement Analysis process in analysis model.
11. Explain State diagram with an example.
12. Compare Cardinality and Modality.
13. Describe the Analysis Modeling Approaches and explain the Rules of Thumb.
14. Elucidate Creation of Flow Oriented Modeling in software engineering.

KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

II B.Sc(CS)   (BATCH 2018-2021)   IV SEMESTER

SOFTWARE ENGINEERING (18CSU402)

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

UNIT III

S.NO	QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	ANSWER
1	There are ___________ major phases to any design process	2	3	4	5	2
2	Diversification is the _____________ of a repertoire of alternatives.	component	solution	acquisition	knowledge	acquisition
3	During _____________, the designer chooses and combines appropriate elements from the repertoire to meet the design objectives.	diversification	convergence	elimination	creation	convergence
4	_________ and ___________ combine intuition and judgement based on experience in building similar entities.	elimination, convergence	creation, convergence	acquisition, creation	diversification and convergence	diversification and convergence
5	___________ can be traced to a customer's requirements and at the same time assessed for quality against a set of predefined criteria.	design	analysis	principles	testing	design
6	The ___________ must implement all of the explicit requirements contained in the analysis model	principles	testing	design	component	design
7	A ____________ should exhibit an architectural structure that has been created using recognizable design patterns.	principles	testing	component	design	design

8	A ____________ is composed of components that exhibit good design characteristics.	principles	testing	component	design	design
9	A ____________ can be implemented in an evolutionary fashion thereby facilitating implementation and testing.	principles	testing	component	design	design
10	A ____________ should be modular that is the software should be logically partitioned into elements that perform specific functions and sub functions.	design	principles	component	testing	design
11	A ____________ should contain distinct representations of data, architecture, interfaces, and components.	design	principles	component	testing	design
12	A ____________ should lead to data structures that are appropriate for the objects to be implemented and are drawn from recognizable data patterns.	design	principles	component	testing	design
13	. A ______________ should lead to interfaces that reduce the complexity of connections between modules and with the external environment.	design	principles	component	testing	design
14	A ____________ should be derived using a repeatable method that is driven by information obtained during software requirements analysis	principles	component	design	testing	design
15	The software ___________ process encourages good design through the application of fundamental design principles, systematic methodology and thorough review.	principles	component	design	testing	design
16	The ___________ must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.	principles	component	design	testing	design
17	The ___________ should provide a complete picture of the software addressing the data, functional and behavioral domains from an implementation perspective	principles	component	design	testing	design

18	The evolution of software __________ is a continuing process that has spanned the past four decades.	principles	component	design	testing	design
19	Procedural aspects of design definition evolved into a philosophy called ____________.	top down programming	bottom up programming	structured programming	object oriented programming	structured programming
20	The design process should not suffer from ___________.	analysis	tunnel vision	conceptual errors	integrity	tunnel vision
21	The design should be __________ to the analysis model.	consistent	related	traceable	relevant	traceable
22	The design should not ___________ the wheel.	minimize	maximize	integrate	reinvent	reinvent
23	The design should ___________ the intellectual distance	maximize	minimize	integrate	analyse	minimize
24	. The ___________ is represented at a high level of abstraction	specification	analysis	quality	design specification	design specification
25	The design should exhibit ___________ and integration.	uniformity	analysis	quality	review	uniformity
26	The design should be ____________ to accommodate change.	reviewed	analysed	assessed	structured	structured
27	The design should be ___________ to degrade gently, even when aberrant data, events, or operating conditions are encountered.	reviewed	analysed	assessed	structured	structured

28	Design is not ____________, coding is not design	coding	analysis	review	event	coding
29	Design is not coding, ___________ is not design.	coding	analysis	review	event	coding
30	The design should be ___________ for quality as it is being created not after the fact.	reviewed	assessed	structured	integrated	assessed
31	The design should be ____________ to minimize conceptual errors.	reviewed	assessed	structured	integrated	reviewed
32	Software design is both a __________ and a model.	model	process	data	function	process
33	___________ is the only way that we can accurately translate a customer's requirements into a finished software product or system.	specification	design	data	prototype	design
34	The design ____________ is the equivalent of an architect's plan for a house.	analysis	process	model	function	model
35	At the highest level of __________, a solution is stated in broad terms, using the language of the problem environment.	refinement	modularity	abstraction	continuity	abstraction
36	. A ___________ is a named sequence of instructions that has a specific and limited function.	procedural abstraction	data abstraction	control abstraction	Process abstraction	procedural abstraction
37	A ___________ is a named collection of data that describes a data object.	procedural abstraction	data abstraction	control abstraction	Process abstraction	data abstraction
38	__________ implies a program control mechanism without specifying internal detail.	procedural abstraction	data abstraction	control abstraction	Process abstraction	control abstraction
39	____________ is used to coordinate activities in an operating system.	synchronization semaphore	control abstraction	data abstraction	procedural abstraction	synchronization semaphore

40	_________ is a top down design strategy originally proposed by Niklaus Wirth.	stepwise refinement	control abstraction	data abstraction	procedural abstraction	stepwise refinement
41	The designer's goal is to produce a model or representation of a __________ that will later be built	component	entity	data	raw material	component
42	The second phase of any design process is the gradual ___________ of all but one particular configuration of components, and thus the creation of the final product.	acquisition	addition	elimination	creation	elimination
43	Design begins with the __________ model.	data	requirements	specification	code	requirements
44	Software design methodologies lack the __________ that are normally associated with more classical engineering design disciplines.	depth	flexibility	quantitative nature	all of the above	all of the above
45	Software requirements, manifested by the ___________ models, feed the design task.	data	functional	behavioral	all of the above	all of the above
46	___________ is the place where quality is fostered in software engineering	model	data	design	specification	design
47	________ provides us with representations of software that can be assessed for quality.	design	specification	data	prototype	design
48	Procedural aspects of design definition evolved into a philosophy called __________.	procedural programming	object oriented programming	structured programming	all of the above	structured programming
49	Meyer defines __________ criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system.	2	3	4	5	5

50	. If a design method provides a systematic mechanism for decomposing the problem into sub problems, it will reduce the complexity of the overall problem, thereby achieving an effective modular solution. This is called .	modular decomposability	modular composability	modular understandability	modular continuity	modular decomposability
51	If a design method enables existing (reusable) design components to be assembled into a new system, it will yield a modular solution that does not reinvent the wheel. This is called	modular decomposability	modular composability	modular understandability	modular continuity	modular composability
52	If a module can be understood as a stand alone unit (without reference to other modules), it will be easier to build and easier	modular decomposability	modular composability	modular understandabil	modular continuity	modular understandabili
53	If small changes to the system requirements result in changes to individual modules, rather than system wide changes, the impact of change-induced side effects will be minimized. This is	modular decomposability	modular composability	modular understandability	modular continuity	modular continuity
54	If an aberrant condition occurs within a module and its effects are constrained within that module, the impact of error-induced side effects will be minimized. This is called	modular protection	modular composability	modular understandability	modular continuity	modular protection
55	The aspect of the architectural design representation defines the components of a system and the manner in which those components are packaged and interact with one another. This property is called	extra functional property	structural property	families of related systems	operational property	structural property
56	____________ represent architecture as an organized collection of program components.	dynamic models	functional models	framework models	structural models	structural models
57	____________ increases the level of design abstraction by attempting to identity repeatable architectural design frameworks that are encountered in similar types of applications.	framework models	dynamic models	process models	functional models	framework models
58	_________ address the behavioural aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events.	framework models	dynamic models	process models	functional models	dynamic models

59	____________ focus on the design of the business or technical process that the system must accommodate.	framework models	dynamic models	process models	functional models	process models
60	______________ can be used to represent the functional hierarchy of a system.	framework models	dynamic models	process models	functional models	functional models

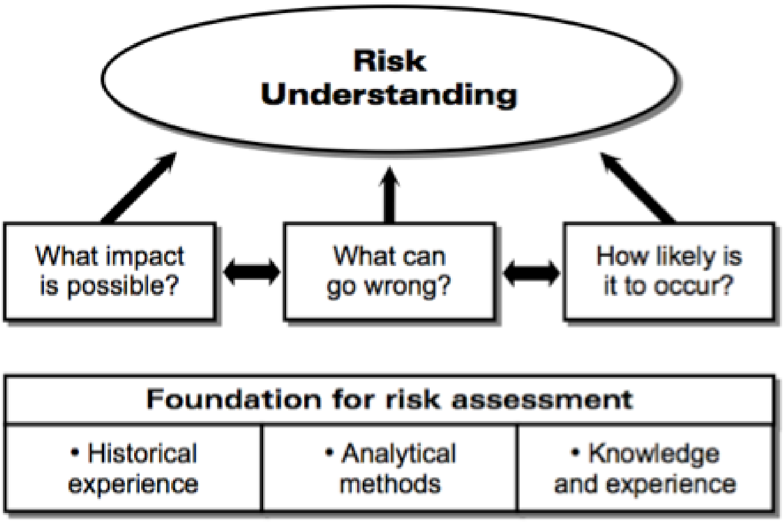
Risk Management: Software Risks, Risk Identification Risk Projection and Risk Refinement, RMMM plan, **Quality Management-** Quality Concepts, Software Quality Assurance, Software Reviews, Metrics for Process and Projects

Risk Management

Introduction

Risk is inevitable in a business organization when undertaking projects. However, the project manager needs to ensure that risks are kept to a minimal. Risks can be mainly divided between two types, negative impact risk and positive impact risk.

Not all the time would project managers be facing negative impact risks as there are positive impact risks too. Once the risk has been identified, project managers need to come up with a mitigation plan or any other solution to counter attack the risk.



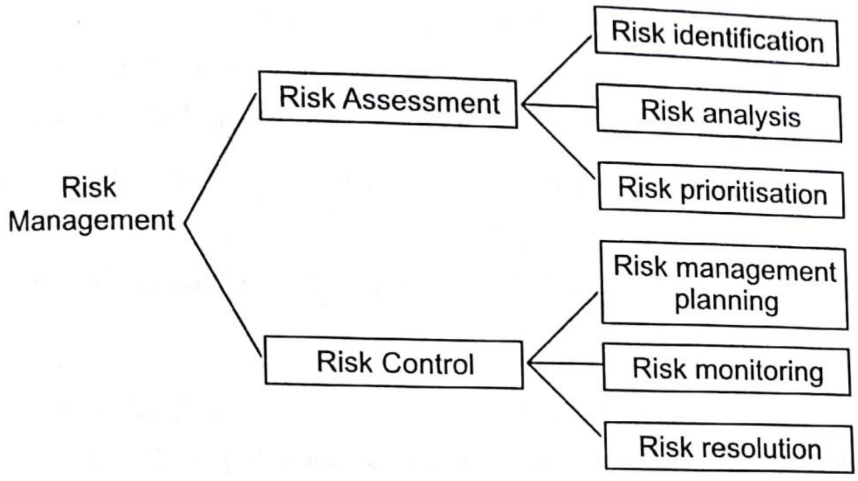
Project Risk Management

Managers can plan their strategy based on four steps of risk management which prevails in an organization. Following are the steps to manage risks effectively in an organization:

- Risk Identification
- Risk Quantification
- Risk Response

- Risk Monitoring and Control

Let's go through each of the step in project risk management:



Risk management activities

There are two characteristics of risk i.e. uncertainty and loss.

Following are the categories of the risk:

1. Project risk

- If the project risk is real then it is probable that the project schedule will slip and the cost of the project will increase.
- It identifies the potential schedule, resource, stakeholders and the requirements problems and their impact on a software project.

2. Technical risk

- If the technical risk is real then the implementation becomes impossible.
- It identifies potential design, interface, verification and maintenance of the problem.

3. Business risk

- If the business risk is real then it harms the project or product.

There are five sub-categories of the business risk:

1. Market risk - Creating an excellent system that no one really wants.

2. Strategic risk - Creating a product which no longer fit into the overall business strategy for companies.

3. Sales risk - The sales force does not understand how to sell a creating product.

4. Management risk - Loose a support of senior management because of a change in focus.

5. Budget risk - losing a personal commitment.

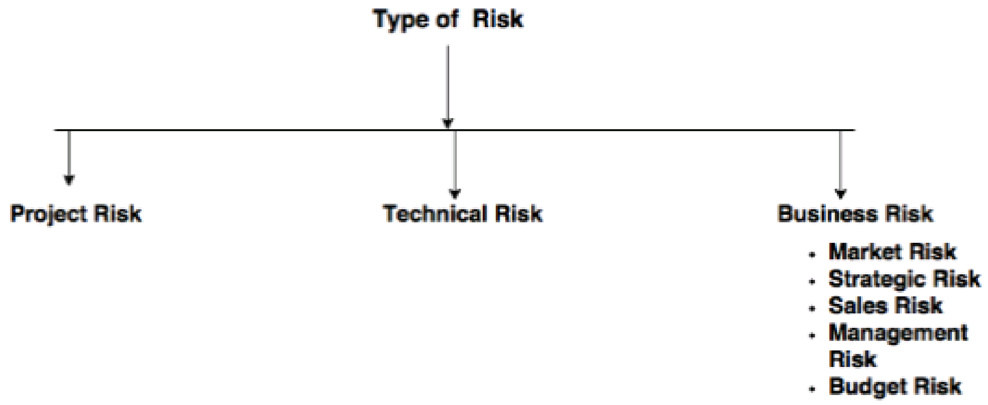
Other risk categories

These categories suggested by Charette.

1. Known risks : These risk are unwrapped after the project plan is evaluated.

2. Predictable risks : These risks are estimated from previous project experience.

3. Unpredictable risks : These risks are unknown and are extremely tough to identify in advance.



Principles of risk management

Maintain a global perspective - View software risks in the context of a system and the business problem planned to solve.

Take a forward looking view – Think about the risk which may occur in the future and create future plans for managing the future events.

Encourage open communication – Encourage all the stakeholders and users for suggesting risks at any time.

Integrate – A consideration of risk should be integrated into the software process.

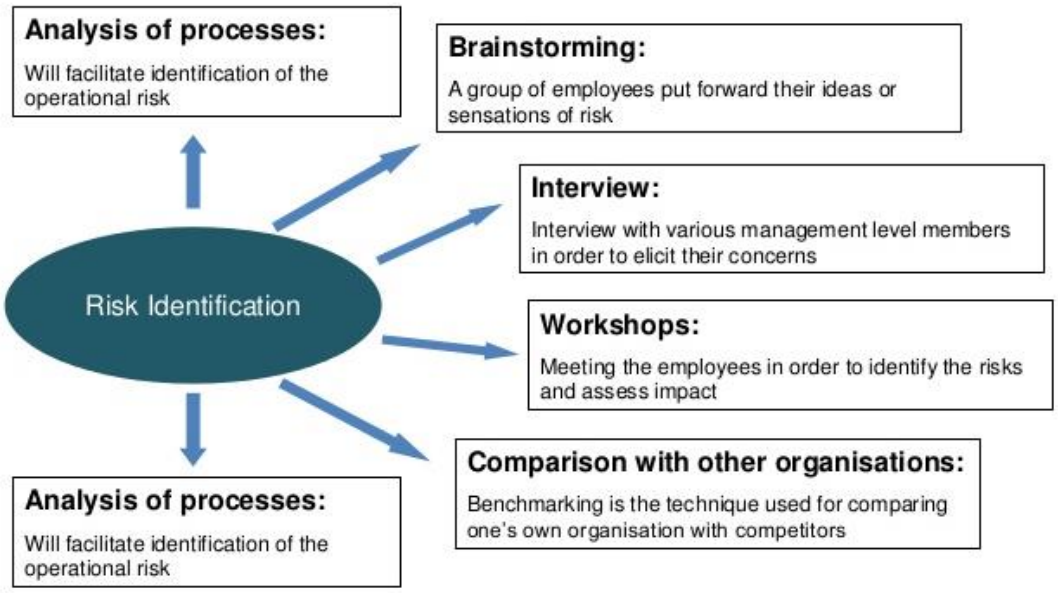
Emphasize a continuous process – Modify the identified risk than the more information is known and add new risks as better insight is achieved.

Develop a shared product vision – If all the stakeholders share the same vision of the software then it is easier for better risk identification.

Encourage teamwork – While conducting risk management activities pool the skills and experience of all stakeholders.

Risk Identification

It is a systematic attempt to specify threats to the project plans.


Risk Identification
Analysis of processes:
Will facilitate identification of the operational risk
Brainstorming:
A group of employees put forward their ideas or sensations of risk
Interview:
Interview with various management level members in order to elicit their concerns
Workshops:
Meeting the employees in order to identify the risks and assess impact
Comparison with other organisations:
Benchmarking is the technique used for comparing one's own organisation with competitors
Analysis of processes:
Will facilitate identification of the operational risk

Two different types of risk:

1. Generic risks

- These risks are a potential threat to each software project.

2. Product-specific risks

- These risks are recognized by those with a clear understanding of the technology, the people and the environment which is specific to the software that is to be built.
- A method for recognizing risks is to create item checklist.
- The checklist is used for risk identification and focus is at the subset of known and predictable risk in the following categories:

  1. Product size
  2. Business impact
  3. Customer characteristic
  4. Process definition
  5. Development environment
  6. Technology to be built
  7. staff size and experience

Contents of a Risk Table

A risk table provides a project manager with a simple technique for risk projection

It consists of five columns

- Risk Summary – short description of the risk
- Risk Category – one of seven risk categories (slide 12)
- Probability – estimation of risk occurrence based on group input
- Impact – (1) catastrophic (2) critical (3) marginal (4) negligible
- RMMM – Pointer to a paragraph in the Risk Mitigation, Monitoring, and Management Plan

Risk Summary	Risk Category	Probability	Impact (1-4)	RMMM

Developing a Risk Table

- List all risks in the first column (by way of the help of the risk item checklists)
- Mark the category of each risk
- Estimate the probability of each risk occurring
- Assess the impact of each risk based on an averaging of the four risk components to determine an overall impact value (See next slide)
- Sort the rows by probability and impact in descending order
- Draw a horizontal cutoff line in the table that indicates the risks that will be given further attention

Risk Mitigation, Monitoring and Management (RMMM)

The Risk Mitigation, Monitoring and Management, RMMM, plan documents all work performed as part of risk analysis and is used by the project manager as part of overall project plan.

Once RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence. Risk analysis supports the project team in constructing a strategy to deal with risks.

There are three important issues considered in developing an effective strategy:

Risk avoidance or mitigation - It is the primary strategy which is fulfilled through a plan. Risk Mitigation covers efforts taken to reduce either the probability or consequences of a threat.

Risk monitoring - The project manager monitors the factors and gives an indication whether the risk is becoming more or less. Risk monitoring and control is the process of identifying, analyzing, and planning for newly discovered risks and managing identified risks.

Risk management and planning - It assumes that the mitigation effort failed and the risk is a reality. Risk management is the identification,assessment, and prioritization of risks

RMMM Plan

- It is a part of the software development plan or a separate document.

- The RMMM plan documents all work executed as a part of risk analysis and used by the project manager as a part of the overall project plan.
- The risk mitigation and monitoring starts after the project is started and the documentation of RMMM is completed.

Quality Management

Software Quality Management

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

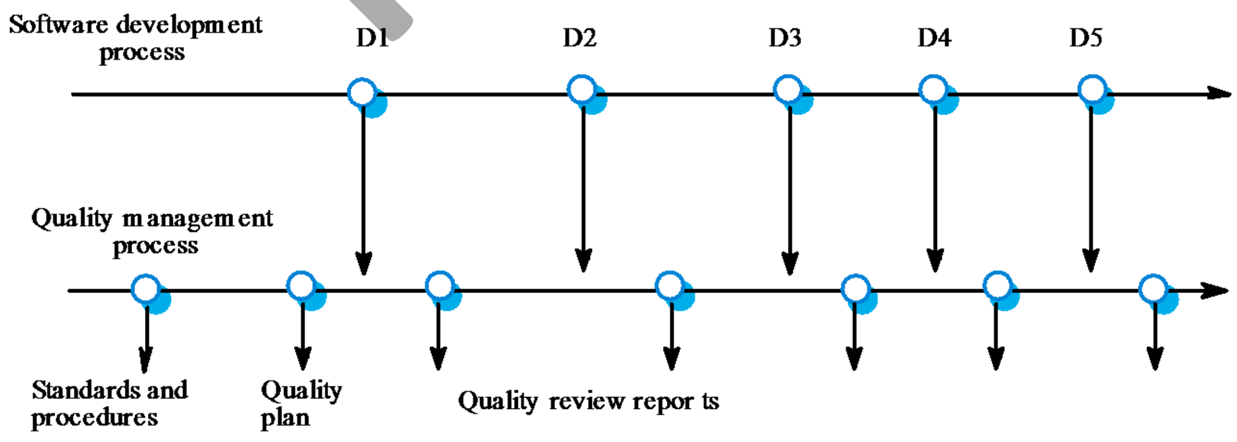
Software Quality management should be independent of project management to ensure independence of cost and schedule adherences. It directly affects the process quality and indirectly affects the product quality.

Activities of Software Quality Management:

Quality Assurance - QA aims at developing Organizational procedures and standards for quality at Organizational level.

Quality Planning - Select applicable procedures and standards for a particular project and modify as required to develop a quality plan.

Quality Control - Ensure that best practices and standards are followed by the software development team to produce quality products.



Quality Concepts

What is Quality?

Software Quality is *conformance to*:

- explicitly stated *functional and performance requirements*,
- explicitly documented *development standards*,
- *implicit characteristics* that are expected of all professionally developed software.

Problems with Software Quality

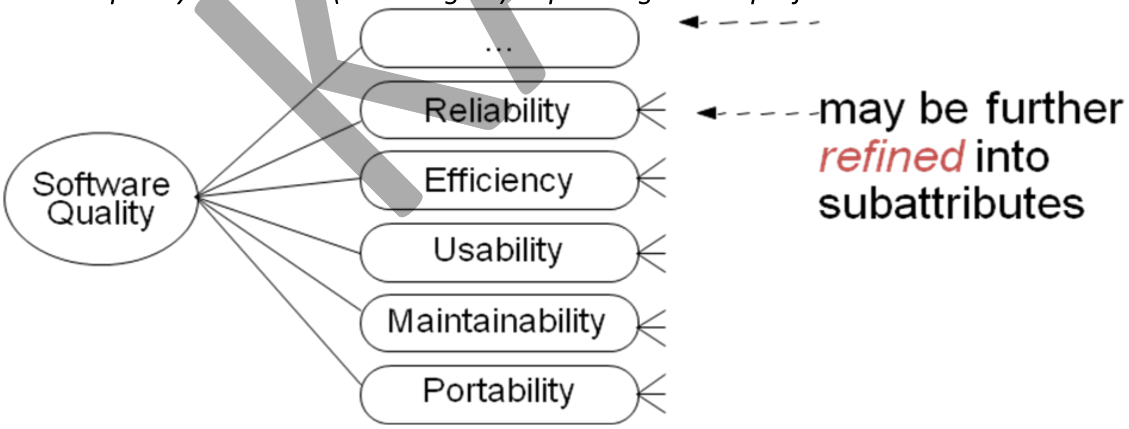
- Software specifications are usually *incomplete and often inconsistent*
- There is *tension* between:
  - customer quality requirements (efficiency, reliability, etc.)
  - developer quality requirements (maintainability, reusability, etc.)
- Some quality requirements are *hard to specify* in an unambiguous way
  - directly measurable qualities (e.g., errors/KLOC),
  - indirectly measurable qualities (e.g., usability).

Quality management is not just about reducing defects!

Hierarchical Quality Model

Define quality via hierarchical quality model, i.e. a number of *quality attributes* (a.k.a. quality factors, quality aspects, ...)

Choose quality attributes (and weights) depending on the project context



Quality Attributes

Quality attributes apply both to the product and the process.

- ***product***: delivered to the customer

- ***process:*** produces the software product
- ***resources:*** (both the product and the process require resources)
  - Underlying assumption: a quality process leads to a quality product (cf. metaphor of manufacturing lines)

Quality attributes can be external or internal.

- ***External:*** Derived from the relationship between the environment and the system (or the process). (To derive, the system or process must run)
  - e.g. Reliability, Robustness
  - ***Internal:*** Derived immediately from the product or process description (To derive, it is sufficient to have the description)
  - Underlying assumption: internal quality leads to external quality (cfr. metaphor manufacturing lines)
  - e.g. Efficiency

Quality attributes

Correctness

- A system is correct if it *behaves according to its specification*
  - An *absolute property* (i.e., a system cannot be “almost correct”)
  - ... in theory and practice *undecidable*

Reliability

- The user may rely on the system behaving properly
- Reliability is the *probability* that the system will operate as expected over a specified interval
  - A *relative property* (a system has a mean time between failure of 3 weeks)

Robustness

- A system is robust if it behaves reasonably *even in circumstances that were not specified*
- A *vague property* (once you specify the abnormal circumstances they become part of the requirements)

Efficiency (Performance)

- *Use of resources* such as computing time, memory
  - Affects user-friendliness and scalability
  - Hardware technology changes fast!
  - *First do it, then do it right, then do it fast*
- For process, resources are manpower, time and money
  - relates to the “productivity” of a process

Usability (User Friendliness, Human Factors)

- The *degree* to which the human users find the system (process) *both "easy to use" and useful*
  - Depends a lot on the target audience (novices vs. experts)
  - Often a system has various kinds of users (end-users, operators, installers)
  - Typically expressed in "amount of time to learn the system"

Maintainability

- *External product attributes* (evolvability also applies to process)
- How easy it is to *change* a system after its initial release
  - software entropy ⇒ maintainability gradually decreases over time

Software Quality Assurance

What is Assurance?

Assurance is nothing but a positive declaration on a product or service, which gives confidence. It is certainty of a product or a service, which it will work well. It provides a guarantee that the product will work without any problems as per the expectations or requirements.

What is Quality Assurance?

Quality Assurance is popularly known as QA Testing, is defined as an activity to ensure that an organization is providing the best possible product or service to customers. QA focuses on improving the processes to deliver Quality Products to the customer. An organization has to ensure, that processes are efficient and effective as per the quality standards defined for software products.

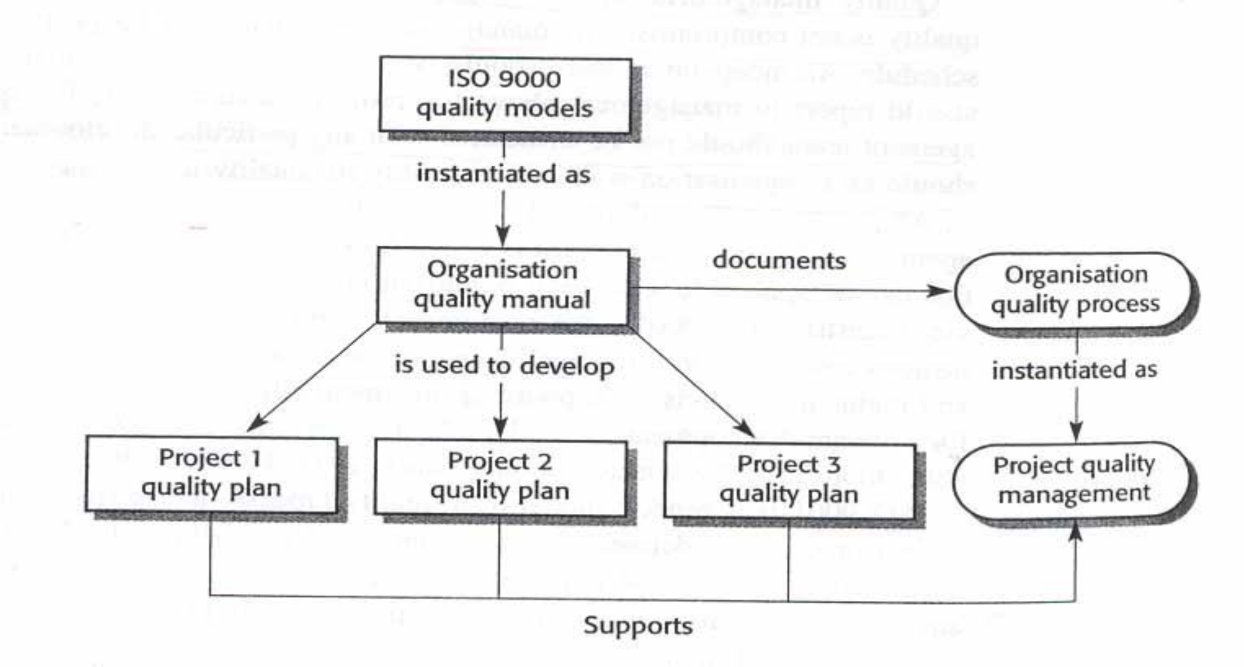
Elements of SQA

- Standards
- Reviews and Audits
- Testing
- Error/defect collection and analysis
- Change management
- Education
- Vendor management
- Security management
- Safety
- Risk management

Why are Standards Important?

- Standards provide encapsulation of best, or at least most appropriate, practice
- • Standards provide a framework around which the quality assurance process may be implemented
- • Standards assist in continuity of work when it's carried out by different people throughout the software product lifecycle
- **Standards should not be avoided. If they are too extensive for the task at hand, then they should be tailored.**

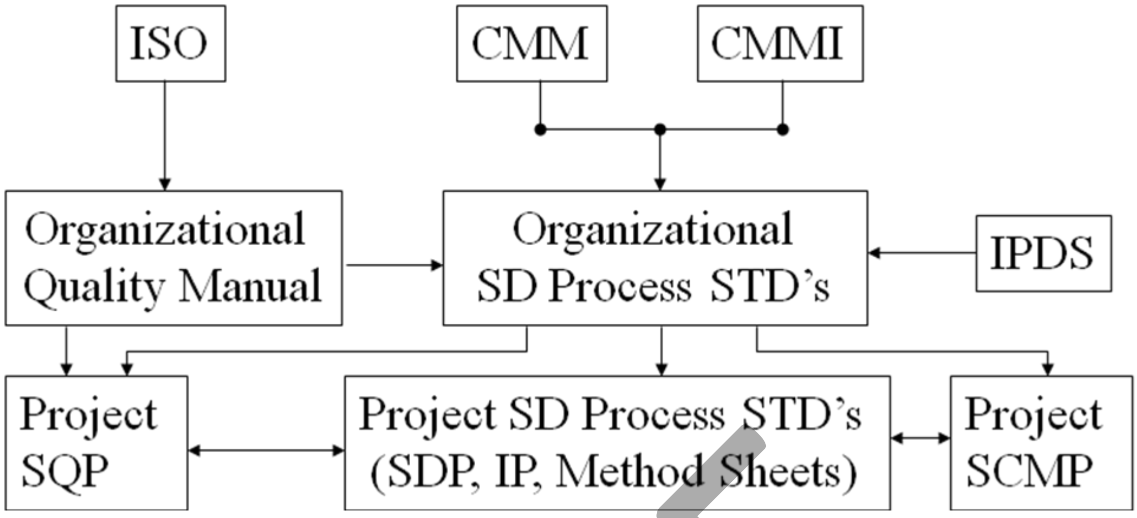
SDS a Simplistic approach



In most mature organizations:
• ISO is not the only source of SDS
• Process and Product standards are derived independently
• Product standards are not created by SQA

Process Standards

Process Standards – standards that define the process which should be followed during software development


ISO
CMM
CMMI
Organizational Quality Manual
Organizational SD Process STD's
IPDS
Project SQP
Project SD Process STD's (SDP, IP, Method Sheets)
Project SCMP

The Capability Maturity Model (CMM) is a methodology used to develop and refine an organization's software development process. .

The Capability Maturity Model Integration (CMMI) is a capability maturity model developed by the Software Engineering Institute, part of Carnegie Mellon University in Pittsburgh, USA.
The CMMI principal is that "the quality of a system or product is highly influenced by the process used to develop and maintain it"

Integrated Product Development or IPDS manages products from concept to closure through a series of gates. Each IPDS gate refers to a specific phase of the project.
Product Standards

SQA (SOFTWARE QUALITY ASSURANCE) is enforcing quality procedures and the **SQP (SOFTWARE QUALITY PLAN)** are the directions on how you go about enforcing the quality assurance.

Software Configuration Management Plan (SCMP)

Software Configuration Management encompasses the disciplines and techniques of initiating, evaluating and controlling change to software products during and after the software engineering process.

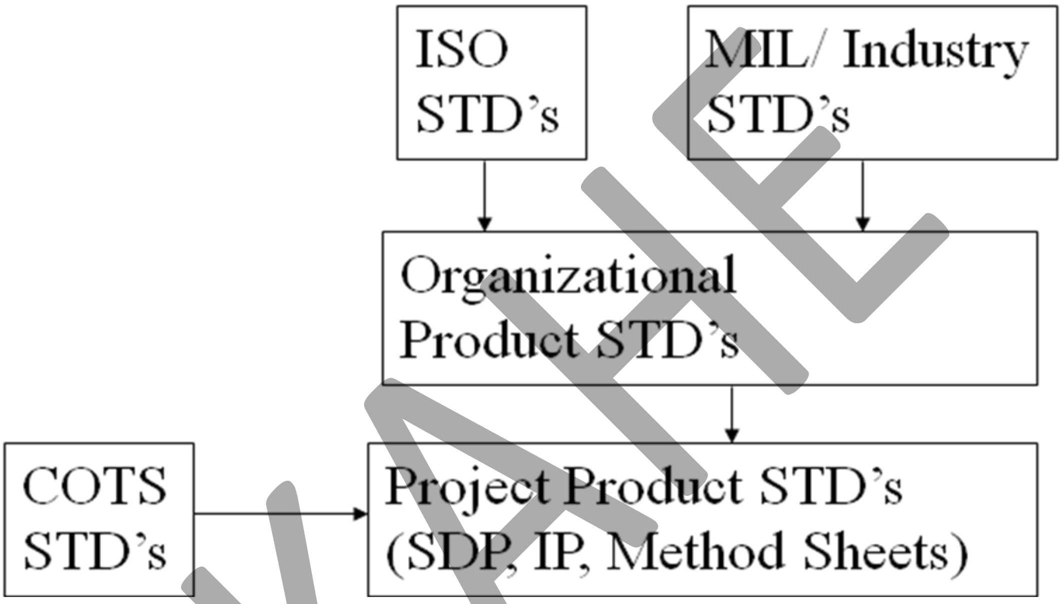
The problem:

- Multiple people have to work on software that is changing
- More than one version of the software has to be supported:
- Released systems

- Custom configured systems (different functionality) System(s) under development
- Software must run on different machines and operating systems
- Need for coordination

Software Configuration Management
  - manages evolving software systems
  - controls the costs involved in making changes to a system

Product Standards – standards that apply to software product being developed


ISO STD's
MIL/ Industry STD's
Organizational Product STD's
COTS STD's
Project Product STD's (SDP, IP, Method Sheets)

Intellectual property (IP) is a term for any intangible asset -- something proprietary that doesn't exist as a physical object but has value. Examples of intellectual property include designs, concepts, software, inventions, trade secrets, formulas and brand names, as well as works of art. Intellectual property can be protected by copyright, trademark, patent or other legal measure.

Software Reviews

What Are Reviews?

- a meeting conducted by technical people for technical people
- a technical assessment of a work product created during the software engineering process
- a software quality assurance mechanism
- a training ground

What Reviews Are Not

- A project summary or progress assessment
- A meeting intended solely to impart information
- A mechanism for political or personal reprisal!

Informal Reviews

Informal reviews include:

- a simple desk check of a software engineering work product with a colleague
- a casual meeting (involving more than 2 people) for the purpose of reviewing a work product, or the review-oriented aspects of pair programming

pair programming encourages continuous review as a work product (design or code) is created.

- The benefit is immediate discovery of errors and better work product quality as a consequence.

Formal Technical Reviews

The objectives of an FTR are:

- to uncover errors in function, logic, or implementation for any representation of the software
- to verify that the software under review meets its requirements
- to ensure that the software has been represented according
- to predefined standards
- to achieve software that is developed in a uniform manner
- to make projects more manageable

The FTR is actually a class of reviews that includes **walkthroughs and inspections**

The Review Meeting

- Between three and five people (typically) should be involved in the review.
- Advance preparation should occur but should require no more than two hours of work for each person.
- The duration of the review meeting should be less than two hours.
- Focus is on a work product (e.g., a portion of a requirements model, a detailed component design, source code for a component)

The Players

Producer—the individual who has developed the work product

- informs the project leader that the work product is complete and that a review is required

Review leader—evaluates the product for readiness, generates copies of product materials, and distributes them to two or three reviewers for advance preparation.

Reviewer(s)—expected to spend between one and two hours reviewing the product, making notes, and otherwise becoming familiar with the work.

Recorder—reviewer who records (in writing) all important issues raised during the review.

The Players
review leader
standards bearer (SQA)
producer
maintenance oracle
recorder
user rep
reviewer

Metrics for Process and Projects

What are Metrics?

- Software process and project metrics are quantitative measures
- They are a management tool
- They offer insight into the effectiveness of the software process and the projects that are conducted using the process as a framework
- Basic quality and productivity data are collected.
- These data are analyzed, compared against past averages, and assessed
  The goal is to determine whether quality and productivity improvements have occurred
- The data can also be used to pinpoint problem areas
- Remedies can then be developed and the software process can be improved.

Reasons to Measure

To characterize in order to

- Gain an understanding of processes, products, resources, and environments
- Establish baselines for comparisons with future assessments
- To evaluate in order to
- Determine status with respect to plans
- To predict in order to
- Gain understanding of relationships among processes and products
- Build models of these relationships
- To improve in order to
- Identify roadblocks, root causes, inefficiencies, and other opportunities for improving product quality and process performance

Metrics In The Process And Project Domains

Process metrics are collected across all projects and over long periods of time.

- Their intent is to provide a set of process indicators that lead to long-term software process improvement.
- Project metrics enable a software project manager to
  (1) assess the status of an ongoing project,
  (2) track potential risks,
  (3) uncover problem areas before they go “critical,”

(4) adjust work flow or tasks,
(5) evaluate the project team’s ability to control quality of software work products

Process Metrics and Software Process Improvement:-

- Software process improvement, it is important to note that process is only one of a number of “controllable factors in improving software quality and organizational performance”.
- Process sits at the center of a triangle connecting three factors that have a profound influence on software quality and organizational performance.

Process Metrics and Software Process Improvement

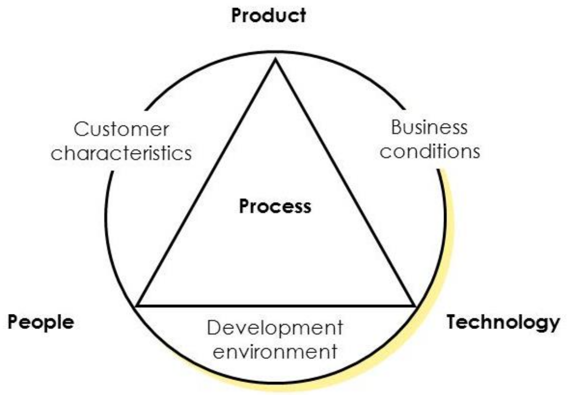


Fig: Determinants for s/w quality and organizational effectiveness

We measure the effectiveness of a process by deriving a set of metrics based on outcomes of the process such as

- Errors uncovered before release of the software
- Defects delivered to and reported by the end users
- Work products delivered
- Human effort expended
- Calendar time expended
- Conformance to the schedule
- Time and effort to complete each generic activity.

Etiquette(good manners) of Process Metrics:

- Use common sense and organizational sensitivity when interpreting metrics data
- Provide regular feedback to the individuals and teams who collect measures and metrics
- Don't use metrics to evaluate individuals
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them
- Never use metrics to pressure individuals or teams
- Metrics data that indicate a problem should not be considered "negative"
  - Such data are merely an indicator for process improvement
- Don't obsess on a single metric to the exclusion of other important metrics

Project Metrics:-

- Many of the same metrics are used in both the process and project domain
- Project metrics are used for making tactical decisions
  - They are used to adapt project workflow and technical activities.
- The first application of project metrics occurs during estimation
  - Metrics from past projects are used as a basis for estimating time and effort
- As a project proceeds, the amount of time and effort expended are compared to original estimates
- As technical work commences, other project metrics become important
  - Production rates are measured (represented in terms of models created, review hours, function points, and delivered source lines of code)
  - Error uncovered during each generic framework activity (i.e, communication, planning, modeling, construction, deployment) are measured

Possible Questions

Part – B (2 Mark)

1. Write about generic risk and product risk.

2. What is risk projection?

3. Define RMMM plan and its use.

4. What are the elements of software quality assurance

5. What are project metrics?

Part – C (6 Mark)

1. Explain in detail about Software risks that are faced by developers.

2. What is the use of software reviews? Explain in detail.

3. Describe in detail how are risks identified.

4. What is risk refinement? Explain in detail the steps to refine a risk if it occurs.

5. Illustrate risk projection mechanism in software engineering

6. What are Formal technical reviews? How are they conducted in software engineering?

7. Describe in detail about RMMM plan

8. Explain in detail about project matrices

9. Enumerate in detail the quality concepts that must be considered in developing a software

10. Explain the software quality assurance standards in detail.

KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

II B.Sc(CS) **(BATCH 2018-2021)** **IV SEMESTER**

SOFTWARE ENGINEERING (18CSU402)

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

UNIT IV

S.NO	QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	ANSWER
1	Interface design focuses on __________ areas of concern.	2	3	4	5	3
2	Frustration and ___________ are part of daily life for many users of computerized information system	sadness	happiness	enjoyment	anxiety	anxiety
3	___________ creates effective communication medium between a human and a computer.	user interface design	architectural design	code design	procedure design	user interface design
4	__________ identifies interface objects and actions and then creates a screen layout that form the basis for a user interface prototype.	design	coding	testing	analysis	design
5	___________ begins with the identification of user, task and environmental requirements.	user interface design	architectural design	code design	procedure design	user interface design
6	There are _________ golden rules.	2	3	4	5	3
7	We should define interaction modes in a way that does not force a user into unnecessary or undesired actions.	interaction modes	interface constraints	design principles	design analysis	interaction modes

8	We should provide ____________ interaction.	rigid	flexible	encouraging	enthusiastic	flexible
9	We should design for direct interaction with _________ that appear on the screen	code	class	objects	user	objects
10	We should hide technical ____________ from the casual user	reactions	actions	internals	interactions	internals
11	We should streamline ____________ as skill levels advance and allow the interaction to be customized.	internals	interaction	actions	reactions	interaction
12	We should allow user interaction to be ___________ and undoable	interruptible	flexible	rigid	encouraging	interruptible
13	We should allow user interaction to interruptible and ___________.	undoable	flexible	rigid	encouraging	undoable
14	We should define shortcuts that are ______________.	encouraging	intuitive	default	past actions	intuitive
15	We should define ___________ that are intuitive.	shortcuts	broad area	interruptible actions	interactions	shortcuts
16	We should disclose information in a ____________ fashion.	open	progressive	streamline	flexible	progressive
17	The visual layout of the ___________ should be based on a real world metaphor.	interaction modes	interface	design	structure	interface
18	The interface should present and acquire ______________ in a consistent fashion.	information	task	knowledge	idea	information

19	The interface should present and acquire information in a ___________ fashion.	consistent	inconsistent	rigid	flexible	consistent
20	A ____________ of the entire system incorporates data, architectural interface, and procedural representations of the software	data model	design model	user model	system image	design model
21	The software engineer creates a ________________.	design model	data model	interface model	system image	design model
22	The end user develops a mental image that is often called the ____________.	design model	user model	data model	system image	user model
23	The implementers of the system create a _____________.	design model	system image	data model	user model	system image
24	Users are categorized into __________ types.	2	3	4	5	3
25	Users with no syntactic knowledge of the system and little semantic knowledge of the application or computer usage are called ___________.	knowledgeable intermittent users	knowledgeable frequent users	novices	all of the above	novices
26	Users with reasonable semantic knowledge of the application but relatively low recall of syntactic information necessary to use the interface are called ___________.	novices	knowledgeable, intermittent users	knowledgeable, frequent users	all of the above	knowledgeable, intermittent users
27	Users with good semantic and syntactic knowledge that often leads to the "power-user syndrome" are called _________.	novices	knowledgeable, intermittent users	knowledgeable, frequent users	all of the above	knowledgeable, frequent users
28	Individuals who look for shortcuts and abbreviated modes of interaction are called ___________.	novices	knowledgeable, intermittent users	knowledgeable, frequent users	Testers	knowledgeable, frequent users

29	The ___________ is the image of the system that end-users carry in their heads.	user’s model	data model	design model	system image	user’s model
30	Stepwise elaboration is called ___________.	functional decomposition	data abstraction	modularity	modular protection	functional decomposition
31	____________ is the only way that we can accurately translate a customer’s requirements into a finished software product or	specification	design	data	prototype	design
32	Validation focuses on ____________ criteria.	2	3	4	5	2
33	Task analysis can be applied in _________ ways.	2	3	4	5	3
34	Task analysis for interface design used ____________ approach.	object oriented approach	top down approach	bottom up approach	all of the above	object oriented approach
35	The overall approach to task analysis, a human engineer must first _________ and classify tasks.	discuss	define	describe	list	define
36	There are ____________ steps in interface design activities.	4	5	6	7	7
37	___________ refers to the deviation from average time.	system response time	variability	system mean time	all of the above	variability
38	System response time has __________ important characteristics.		3	4	5	2
39	A ____________ is designed into the software from the beginning.	integrated help facility	system response time	variability	all of the above	integrated help facility
40	Component level design also called ___________.	procedural abstraction	procedural design	stepwise refinement	decomposition	procedural design

41	___________ must be translated into operational software	data	architectural	interface design	all of the above	all of the above
42	A _________ performs component level design.	user	top level management	software engineer	middle level management	software engineer
43	The ___________ represents the software in a way that allows one to review the details of the design for correctness and consistency with earlier design representations.	component level design	procedural design	data design	data design	component level design
44	Design, representations of data, architecture, and interfaces form the foundation for _____________.	procedural design	component level design	data design	code design	component level design
45	__________ notation is used to represent the design.	graphical	tabular	text-based	all of the above	graphical
46	Any program, regardless of application area or technical complexity, can be designed and implemented using only the __________ structured constructs.	2	3	4	5	3
47	A box in a flowchart is used to indicate a ___________.	processing step	logical condition	flow of control	start	processing step
48	A diamond in a flowchart is used to indicate a _________.	processing step	logical condition	flow of control	start	logical condition
49	The arrows in a flowchart is used to indicate a __________.	processing step	logical condition	flow of control	start	flow of control
50	A picture is worth a __________ words.	100	1000	10000	100000	1000
51	The following construct is fundamental to structured programming.	sequence	condition	repetition	all of the above	all of the above

52	___________ implements processing steps that are essential in the specification of any algorithm.	sequence	condition	repetition	selection	sequence
53	__________ provides the facility for selected processing steps that are essential in the specification of any algorithm	sequence	condition	repetition	selection	condition
54	_________ allows for looping.	sequence	condition	repetition	selection	repetition
55	Another graphical design tool, the ________ evolved from a desire to develop a procedural design representation that would not allow violation of the structured constructs.	box diagram	flowchart	transition diagram	decision table	box diagram
56	PDL is the abbreviation of _____________.	Process Design Language	Program Design Language	Program Document Language	Program Document Language	Program Design Language
57	A design language should have the ___________ characters.	2	3	4	5	4
58	Design notation should support the development of modular software and provide a means for interface specification. This attribute of design notation is called ___________.	modularity	simplicity	ease of editing	maintainability	modularity
59	Design notation should be relatively simple to learn, relatively easy to use, and generally easy to read. This attribute of the design notation is called __________.	modularity	simplicity	ease of editing	maintainability	simplicity
60	The procedural design may require modification as the software process proceeds. The ease with which a design representation can be edited can help facilitate each software engineering task is called ___________.	modularity	simplicity	ease of editing	maintainability	ease of editing

Design Engineering:-Design Concepts, Architectural Design Elements, Software Architecture, Data Design at the Architectural Level and Component Level, Mapping of Data Flow into Software Architecture, Modeling Component Level Design

Introduction to design process

- The main aim of design engineering is to generate a model which shows firmness, delight and commodity.
- Software design is an iterative process through which requirements are translated into the blueprint for building the software.

Software quality guidelines

- A design is generated using the recognizable architectural styles and compose a good design characteristic of components and it is implemented in evolutionary manner for testing.
- A design of the software must be modular i.e the software must be logically partitioned into elements.
- In design, the representation of data , architecture, interface and components should be distinct.
- A design must carry appropriate data structure and recognizable data patterns.
- Design components must show the independent functional characteristic.
- A design creates an interface that reduce the complexity of connections between the components.
- A design must be derived using the repeatable method.
- The notations should be use in design which can effectively communicates its meaning.

Quality attributes

The attributes of design name as 'FURPS' are as follows:

Functionality:
It evaluates the feature set and capabilities of the program.

Usability:
It is accessed by considering the factors such as human factor, overall aesthetics, consistency and documentation.

Reliability:
It is evaluated by measuring parameters like frequency and security of failure, output result accuracy, the mean-time-to-failure(MTTF), recovery from failure and the the program predictability.

Performance:
It is measured by considering processing speed, response time, resource consumption, throughput and efficiency.

Supportability:

- It combines the ability to extend the program, adaptability, serviceability. These three term defines the maintainability.
- Testability, compatibility and configurability are the terms using which a system can be easily installed and found the problem easily.
- Supportability also consists of more attributes such as compatibility, extensibility, fault tolerance, modularity, reusability, robustness, security, portability, scalability.

Stages of Design

- Problem understanding
  - *Look at the problem from different angles to discover the design requirements.*
- *Identify one or more solutions*
  - *Evaluate possible solutions and choose the most appropriate depending on the designer's experience and available resources.*
- *Describe solution abstractions*
  - *Use graphical, formal or other descriptive notations to describe the components of the design.*
- *Repeat process for each identified abstraction until the design is expressed in primitive terms.*

The Design Process

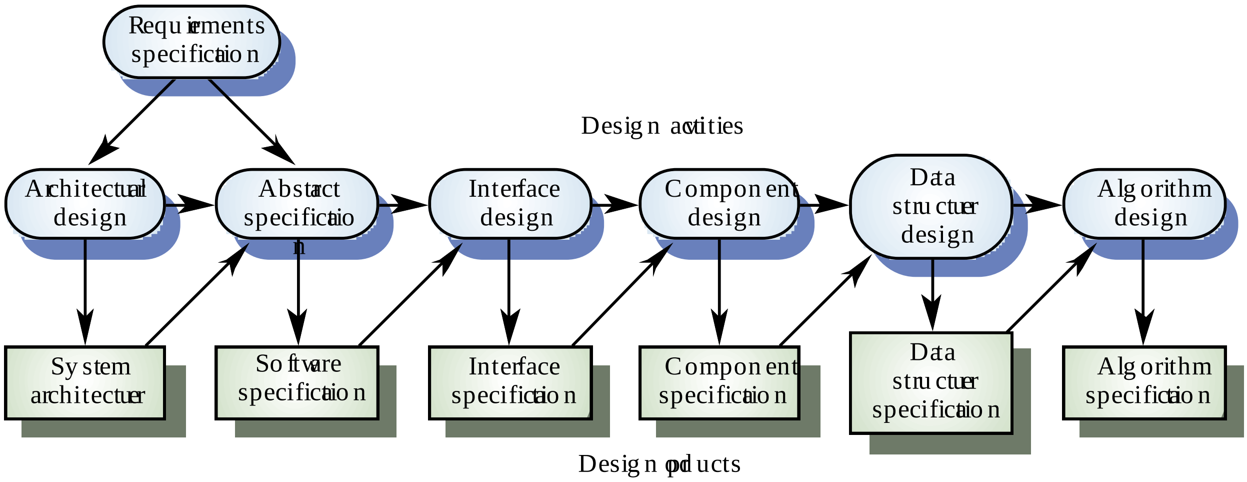
- *Any design may be modelled as a directed graph made up of entities with attributes which participate in relationships.*

- *The system should be described at several different levels of abstraction.*

Design takes place in overlapping stages. It is artificial to separate it into distinct phases but some separation is usually necessary

Design concepts

Phases in the Design Process



The set of fundamental software design concepts are as follows:

1. Abstraction

- A solution is stated in large terms using the language of the problem environment at the highest level abstraction.
- The lower level of abstraction provides a more detail description of the solution.
- A sequence of instruction that contain a specific and limited function refers in a procedural abstraction.
- A collection of data that describes a data object is a data abstraction.

2. Architecture

- The complete structure of the software is known as software architecture.
- Structure provides conceptual integrity for a system in a number of ways.
- The architecture is the structure of program modules where they interact with each other in a specialized way.
- The components use the structure of data.
- The aim of the software design is to obtain an architectural framework of a system.
- The more detailed design activities are conducted from the framework.

3. Patterns

- A design pattern describes a design structure and that structure solves a particular design problem in a specified content.

4. Modularity

- A software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the problem requirements.
- Modularity is the single attribute of a software that permits a program to be managed easily.

5. Information hiding

- Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

6. Functional independence

- The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.
- The functional independence is accessed using two criteria i.e Cohesion and coupling.

Cohesion

- Cohesion is an extension of the information hiding concept.
- A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.

Coupling

Coupling is an indication of interconnection between modules in a structure of software.

7. Refinement

- Refinement is a top-down design approach.
- It is a process of elaboration.
- A program is established for refining levels of procedural details.
- A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.

8. Refactoring

- It is a reorganization technique which simplifies the design of components without changing its function behaviour.
- Refactoring is the process of changing the software system in a way that it does not change the external behaviour of the code still improves its internal structure.

9. Design classes

- The model of software is defined as a set of design classes.
- Every class describes the elements of problem domain and that focus on features of the problem which are user visible.

Following are the types of design elements:

1. Data design elements

- The data design element produced a model of data that represent a high level of abstraction.
- This model is then more refined into more implementation specific representation which is processed by the computer based system.
- The structure of data is the most important part of the software design.

2. Architectural design elements

- The architecture design elements provides us overall view of the system.
- The architectural design element is generally represented as a set of interconnected subsystem that are derived from analysis packages in the requirement model.

The architecture model is derived from following sources:

- The information about the application domain to built the software.
- Requirement model elements like data flow diagram or analysis classes, relationship and collaboration between them.
- The architectural style and pattern as per availability.

3. Interface design elements

- The interface design elements for software represents the information flow within it and out of the system.
- They communicate between the components defined as part of architecture.

Following are the important elements of the interface design:

1. The user interface

2. The external interface to the other systems, networks etc.

3. The internal interface between various components.

4. Component level diagram elements

- The component level design for software is similar to the set of detailed specification of each room in a house.
- The component level design for the software completely describes the internal details of the each software component.
- The processing of data structure occurs in a component and an interface which allows all the component operations.
- In a context of object-oriented software engineering, a component shown in a UML diagram.

The UML diagram is used to represent the processing logic.

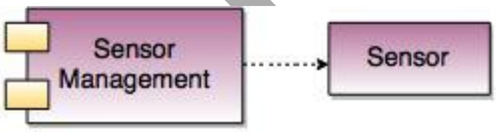


Fig. - UML component diagram for sensor managemnet

5. Deployment level design elements

- The deployment level design element shows the software functionality and subsystem that allocated in the physical computing environment which support the software.

- Following figure shows three computing environment as shown. These are the personal computer, the CPI server and the Control panel.

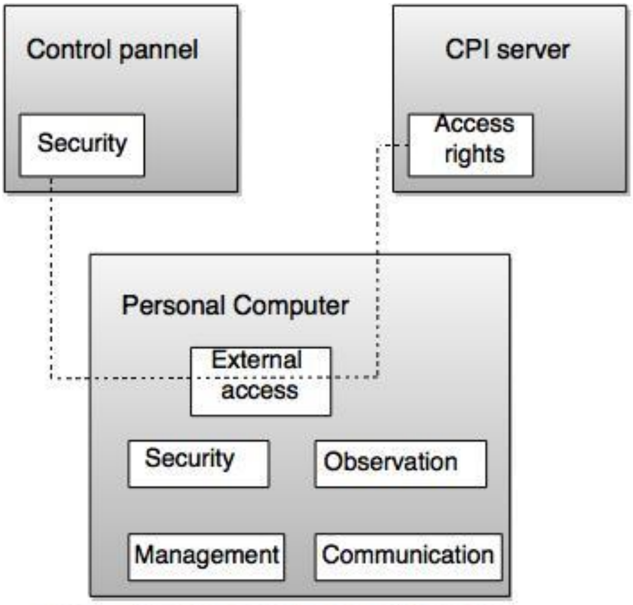


Fig. - Deployment level diagram

Architectural Design

Software Architecture

This section defines the term "software architecture" as a framework made up of the system structures that comprise the software components, their properties, and the relationships among these components. The goal of the architectural model is to allow the software engineer to view and evaluate the system as a whole before moving to component design.

What is Architecture?

The architecture is not the operational software. Rather, it is a representation that enables a software engineer to:

(1) Analyze the effectiveness of the design in meeting its stated requirements,

(2) Consider architectural alternatives at a stage when making design changes is still relatively easy, and

(3) Reduce the risks associated with the construction of the software.

Why is Architecture Important?

- Representations of software architecture are an enabler for communication between all parties (stakeholders) interested in the development of a computer-based system.

- The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.
- Architecture "constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together" [BAS03].

Data Design
This section describes data design at both the architectural and component levels. At the architecture level, data design is the process of creating a model of the information represented at a high level of abstraction (using the customer's view of data).

Data Design at the Architectural Level

The challenge is extract useful information from the data environment, particularly when the information desired is cross-functional.

To solve this challenge, the business IT community has developed *data mining* techniques, also called *knowledge discovery in databases* (KDD), that navigate through existing databases in an attempt to extract appropriate business-level information.

However, the existence of multiple databases, their different structures, the degree of detail contained with the databases, and many other factors make data mining difficult within an existing database environment.

An alternative solution, called a *data warehouse*, adds on additional layer to the data architecture.
A data warehouse is a separate data environment that is not directly integrated with day-to-day applications that encompasses all data used by a business.
In a sense, a data warehouse is a large, independent database that has access to the data that are stored in databases that serve as the set of applications required by a business.

Data Design at the Component Level
At the component level, data design focuses on specific data structures required to realize the data objects to be manipulated by a component.

- refine data objects and develop a set of data abstractions
- implement data object attributes as one or more data structures
- review data structures to ensure that appropriate relationships have been established
- simplify data structures as required

Set of principles for data specification:
1. The systematic analysis principles applied to function and behavior should also be applied to data.
2. All data structures and the operations to be performed on each should be identified.

3. A data dictionary should be established and used to define both data and program design.
4. Low level data design decisions should be deferred until late in the design process.
5. The representation of data structure should be known only to those modules that must make direct use of the data contained within the structure.
6. A library of useful data structures and the operations that may be applied to them should be developed.
7. A software design and programming language should support the specification and realization of abstract data types.

Architectural Styles and Patterns

Each style describes a system category that encompasses:
(1) A set of components (e.g., a database, computational modules) that perform a function required by a system,
(2) A set of connectors that enable "communication, coordination and cooperation" among components,
(3) Constraints that define how components can be integrated to form the system, and
(4) Semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

An **architectural style** is a transformation that is imposed on the design of an entire system. An *architectural pattern,* like an architectural style, imposes a transformation on the design of an architecture.

A **pattern differs** from a style in a number of fundamental ways:

1. The scope of a pattern is less broad, focusing on one aspect of the architecture rather than the architecture in its entirety.
2. A pattern imposes a rule on the architecture, describing how the S/W will handle some aspect of its functionality at the infrastructure level.
3. Architectural patterns tend to address specific behavioral issues within the context of the architectural.

.

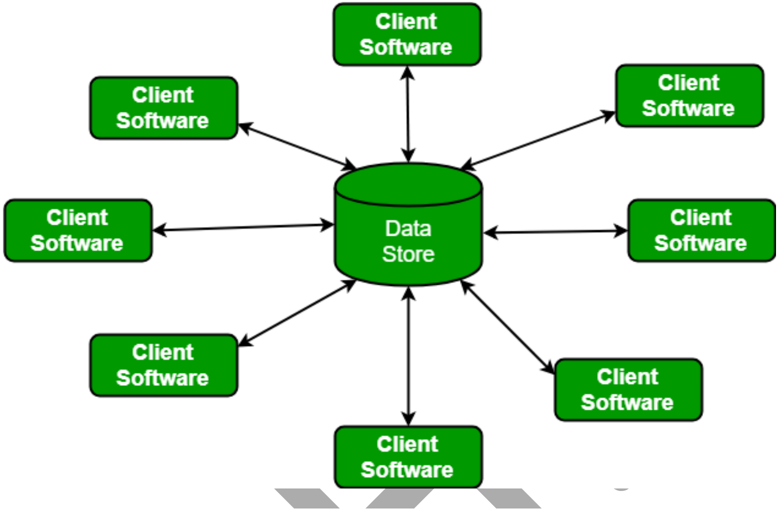
Taxonomy of Architectural styles:

1. **Data centred architectures:**

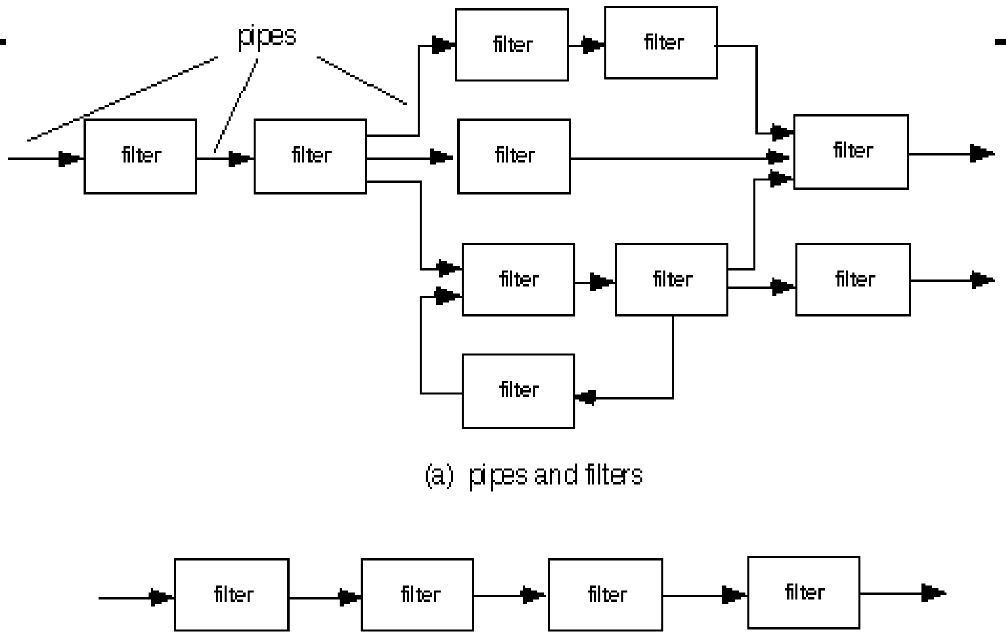
    - A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.
    - The figure illustrates a typical data centered style. The client software access a central repository. Variation of this approach are used to transform the repository

into a blackboard when data related to client or data of interest for the client change the notifications to client software.

- This data-centered architecture will promote integrability. This means that the existing components can be changed and new client components can be added to the architecture without the permission or concern of other clients.
- Data can be passed among clients using blackboard mechanism.



2. **Data flow architectures:**
    - This kind of architecture is used when input data to be transformed into output data through a series of computational manipulative components.
    - The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by pipes.
    - Pipes are used to transmit data from one component to the next.
    - Each filter will work independently and is designed to take data input of a certain form and produces data output to the next filter of a specified form. The filters don’t require any knowledge of the working of neighboring filters.
    - If the data flow degenerates into a single line of transforms, then it is termed as batch sequential. This structure accepts the batch of data and then applies a series of sequential components to transform it.
    -

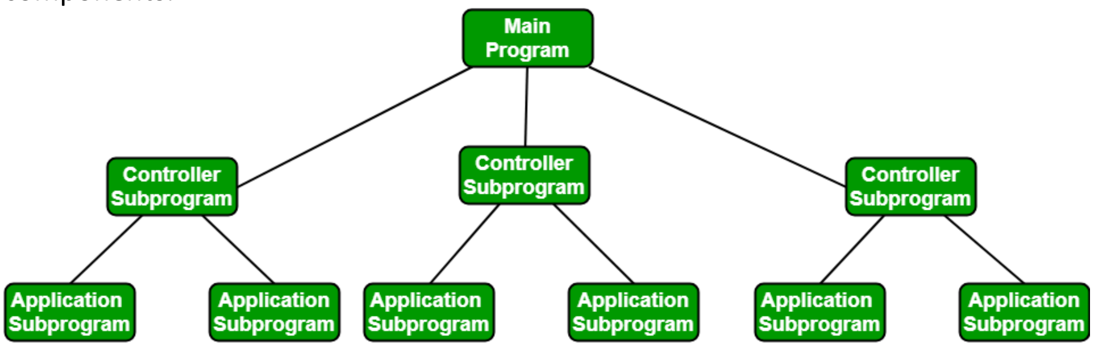


(a) pipes and filters

(b) batch sequential

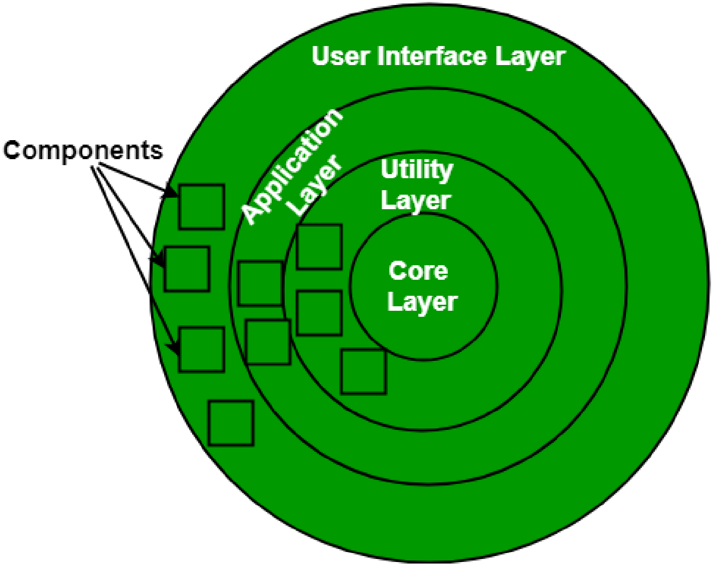
3. **Call and Return architectures:** It is used to create a program that is easy to scale and modify. Many sub-styles exist within this category. Two of them are explained below.

- **Remote procedure call architecture:** This components is used to present in a main program or sub program architecture distributed among multiple computers on a network.
- **Main program or Subprogram architectures:** The main program structure decomposes into number of subprograms or function into a control hierarchy. Main program contains number of subprograms that can invoke other components.



4. **Object Oriented architecture:** The components of a system encapsulate data and the operations that must be applied to manipulate the data. The coordination and communication between the components are established via the message passing.

5. **Layered architecture:**
   - A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively.
   - At the outer layer, components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing(communication and coordination with OS)
   - Intermediate layers to utility services and application software functions.



Architectural Patterns

A S/W architecture may have a number of architectural patterns that address issues such as concurrency, persistence, and distribution.

- Concurrency—applications must handle multiple tasks in a manner that simulates parallelism
  - *operating system process management* pattern
  - *task scheduler* pattern
- Persistence—Data persists if it survives past the execution of the process that created it. Persistent data are stored in a database or file and may be read and modified by other processes at a later time.

  Two patterns are common:
  - a *database management system* pattern that applies the storage and retrieval capability of a DBMS to the application architecture
  - an *application level persistence* pattern that builds persistence features into the application architecture

- Distribution— the manner in which systems or components within systems communicates with one another in a distributed environment, and the nature of the communication that occurs.

A *broker* acts as a 'middle-man' between the client component and a server component.

Mapping Data Flow into Software Architecture

This section describes the general process of mapping requirements into software architectures during the structured design process.

An Architectural Design Method

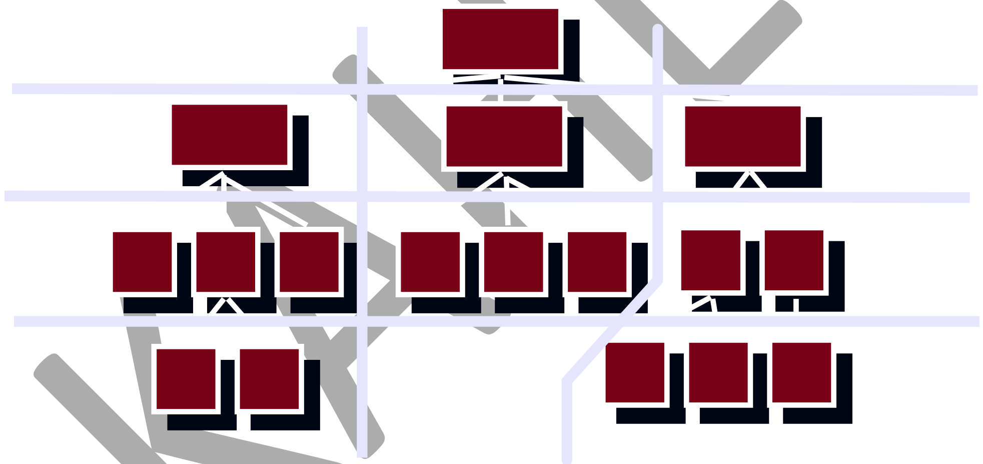
customer requirements

four bedrooms, three baths, lots of glass…

Deriving Program Architecture

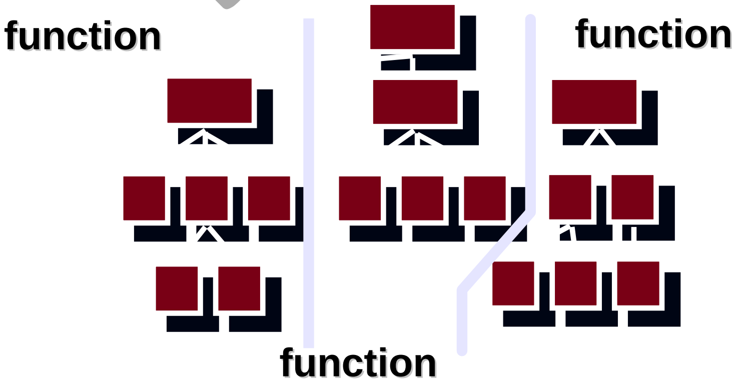
Partitioning the Architecture

horizontal" and "vertical" partitioning are required



Horizontal Partitioning

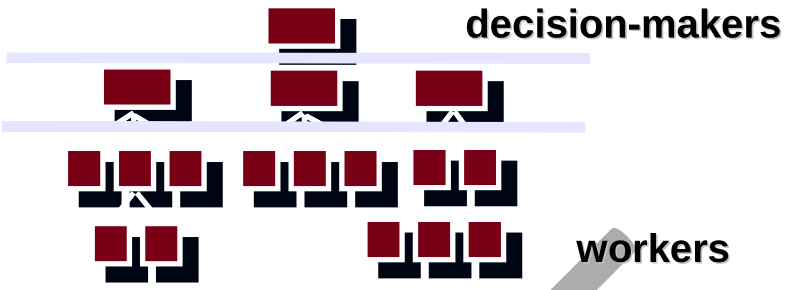
- define separate branches of the module hierarchy for each major function
- use control modules to coordinate communication between functions



Vertical Partitioning:

Factoring

- design so that decision making and work are stratified
- decision making modules should reside at the top of the architecture



Why

Partitioned Architecture?

- results in software that is easier to test
- leads to software that is easier to maintain
- results in propagation of fewer side effects
- results in software that is easier to extend

- objective: to derive a program architecture that is partitioned
- approach:
  - the DFD is mapped into a program architecture
  - the PSPEC and STD are used to indicate the content of each module
- notation: structure chart

Flow Characteristics

Transform Mapping

Transform mapping is a technique in which Data Flow Diagrams (DFD's) are mapped to a specific scenario. It is a data flow-oriented mapping technique that uses DFDs to map real life scenarios to a software architecture. These real life scenarios are converted to what we call DFDs which can be applied to a software architecture. This process of converting a real-life situation (termed as system in software engineering) with flow of data to a DFD is called transform mapping. In this lesson, transform mapping has been described using the scenario of an airline reservation system.

Data Flow Diagram

A Data Flow Diagram(DFD) shows the flow of data through the system. It is also used for modeling the requirements. DFD is often called as a data flow graph. The data flow diagram is created with the help of various symbols which represent a process, data repository etc.

General Mapping Approach

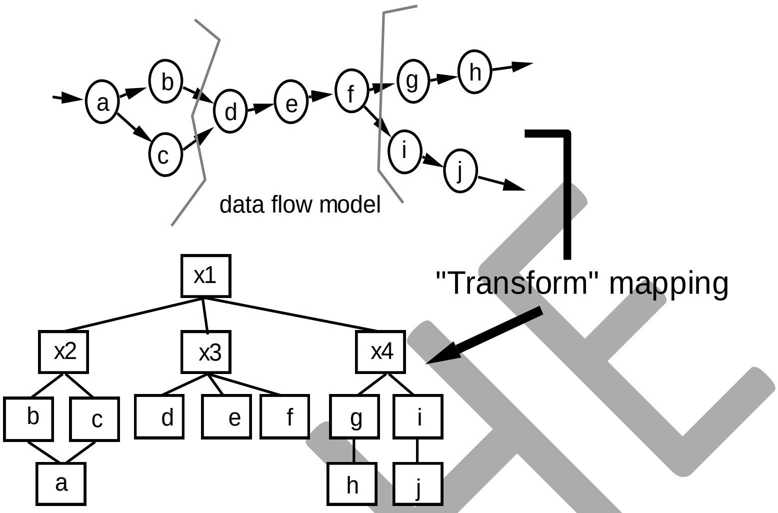
Isolate incoming and outgoing flow boundaries; for transaction flows, isolate the transaction center.

Working from the boundary outward, map DFD transforms into corresponding modules.

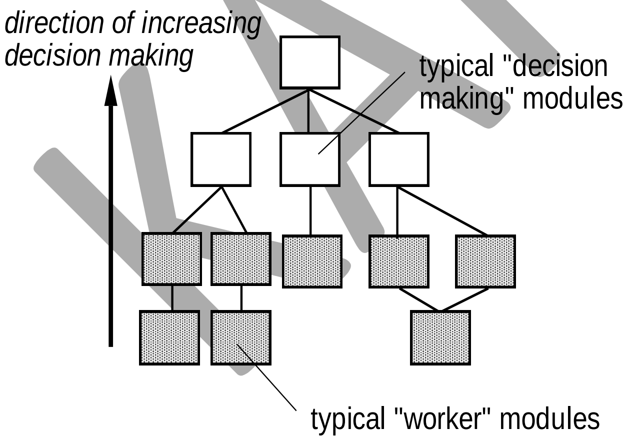
Add control modules as required.

Refine the resultant program structure using effective modularity concepts.

Factoring



First Level


direction of increasing decision making
typical "decision making" modules
typical "worker" modules

Factoring

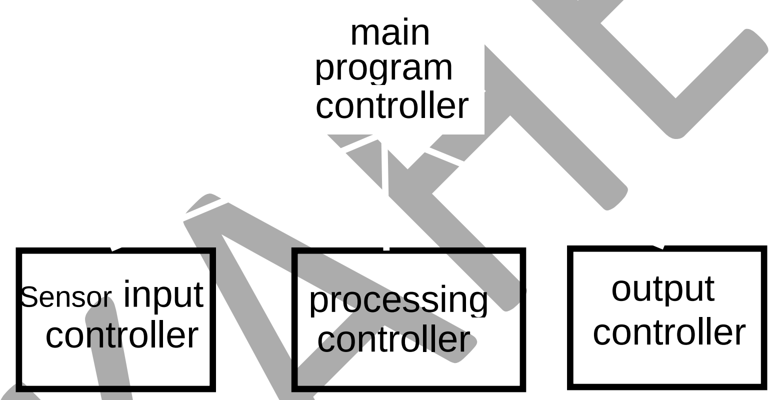
Program structure represents a top-down distribution of control. Factoring results in a program structure in which top-level modules perform decision making and low-level modules perform most input, computation, and output work. Middle-level modules perform some control and do moderate amounts of work. When transform flow is encountered, a DFD is mapped to a specific structure (a call and return architecture) that provides control for incoming, transform, and outgoing information processing.

This first-level factoring for the *monitor sensors* subsystem is illustrated in Figure 14.9. A main controller (called *monitor sensors executive*)resides at the top of the program structure and coordinates the following subordinate control functions:

• An incoming information processing controller, called *sensor input controller,* coordinates receipt of all incoming data.

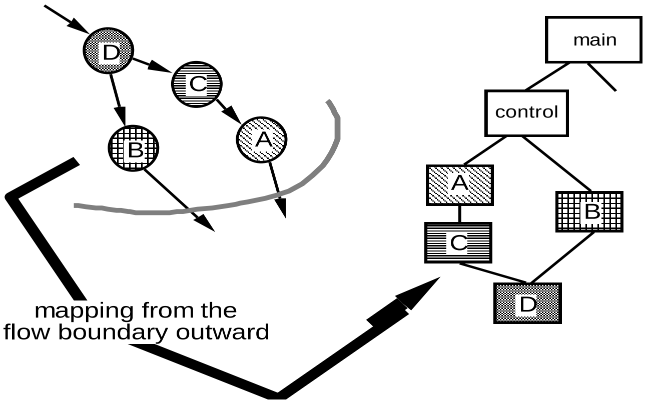
• A transform flow controller, called *alarm conditions controller,* supervises all operations on data in internalized form (e.g., a module that invokes various data transformation procedures).

• An outgoing information processing controller, called *alarm output controller,* coordinates production of output information

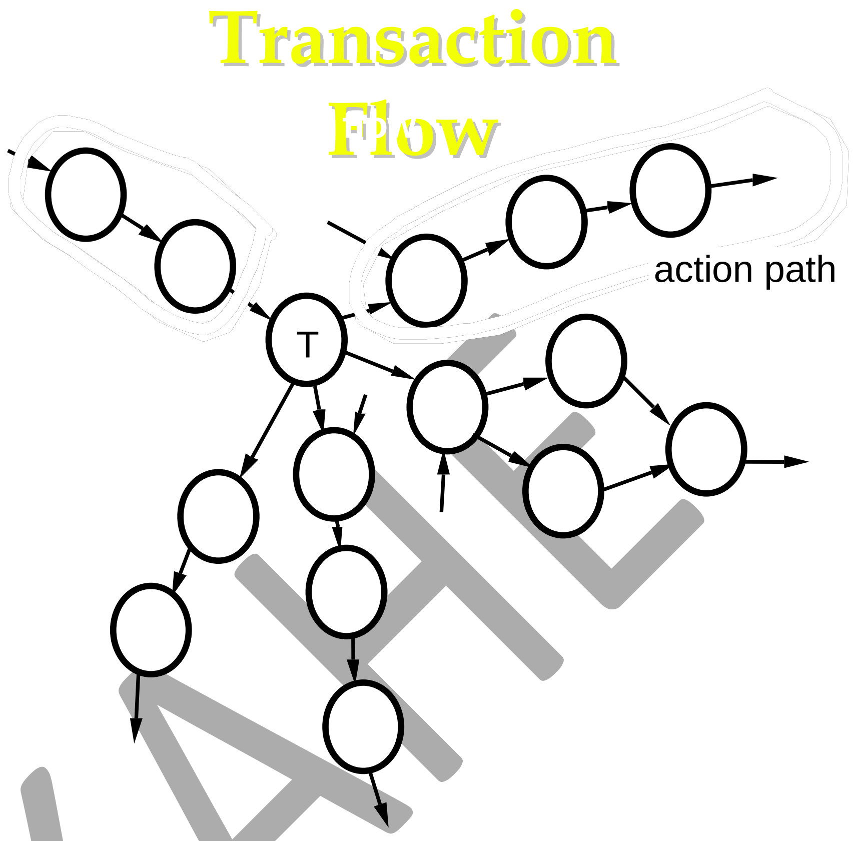


Second Level Mapping

Second-level factoring is accomplished by mapping individual transforms (bubbles) of a DFD into appropriate modules within the architecture. Beginning at the transform center boundary and moving outward along incoming and then outgoing paths, transforms are mapped into subordinate levels of the software structure



Transaction Flow



Modeling Component Level Design

Overview

The purpose of component-level design is to define data structures, algorithms, interface characteristics, and communication mechanisms for each software component identified in the architectural design. Component-level design occurs after the data and architectural designs are established. The component-level design represents the software in a way that allows the designer to review it for correctness and consistency, before it is built. The work product produced is a design for each software component, represented using graphical, tabular, or text -based notation. Design walkthroughs are conducted to determine correctness of the data transformation or control transformation allocated to each component during earlier design steps.

Component Definitions

- Component is a modular, deployable, replaceable part of a system that encapsulates implementation and exposes a set of interfaces

   - Object-oriented view is that component contains a set of collaborating classes
   - Each elaborated class includes all attributes and operations relevant to its implementation All interfaces communication and collaboration with other design classes are also defined
   - Analysis classes and infrastructure classes serve as the basis for object-oriented elaboration
- Traditional view is that a component (or module) reside in the software and serves one of three roles
   - Control components coordinate invocation of all other problem domain components
   - Problem domain components implement a function required by the customer
   - Infrastructure components are responsible for functions needed to support the processing required in a domain application
   - The analysis model data flow diagram is mapped into a module hierarchy as the starting point for the component derivation

Class-based Component Design

- Focuses on the elaboration of domain specific analysis classes and the definition of infrastructure classes
- Detailed description of class attributes, operations, and interfaces is required prior to beginning construction activities

Class-based Component Design Principles

- Open-Closed Principle (OCP) – class should be open for extension but closed for modification
- Liskov Substitution Principle (LSP) – subclasses should be substitutable for their base classes
- Dependency Inversion Principle (DIP) – depend on abstractions, do not depend on concretions
- Interface Segregation Principle (ISP) – many client specific interfaces are better than one general purpose interface
- Release Reuse Equivalency Principle (REP) – the granule of reuse is the granule of release
- Common Closure Principle (CCP) – classes that change together belong together
- Common Reuse Principle (CRP) – Classes that can’t be used together should not be grouped together

Possible Questions

Part – B (2 Mark)

1. Define abstraction.
2. Differentiate between refinement and refactoring
3. Write the difference between transform flow and transaction flow
4. What is transform mapping?
5. Define transaction mapping.

Part – C (6 Mark)

1. Explain in detail the process of data design at
    a. Architectural level ii) Component level
2. Write in detail the approach used to design class based components
3. Discuss in detail about the Architectural components of software.
4. Write short notes on
    a. Transform mapping ii) Transaction mapping
5. Write short notes on the following design concepts
    a. Information hiding ii) Refinement iii) Refactoring
6. Describe in detail the procedure to refine an architecture into components.
7. Write short notes on the following design concepts
    a. Abstraction ii) Architecture iii) Modularity
8. Write in detail the approach used to design conventional components
9. Explain in detail about design process and design quality
10. Write short notes on
    a. Transform flow ii) Transaction flow

KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

II B.Sc(CS) (BATCH 2018-2021) IV SEMESTER

SOFTWARE ENGINEERING (18CSU402)

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

UNIT V

S.NO	QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	ANSWER
1	___________ is a critical element of software quality assurance and represents the ultimate review of specification, design, and code generation.	software specification	software generation	software coding	software testing	software testing
2	Software is tested from ___________ different perspectives.	2	3	4	5	2
3	Software engineers are by their nature ___________ people.	pessimistic	optimistic	constructive	destructive	constructive
4	___________ is a process of executing a program with the intent of finding an error.	coding	testing	debugging	designing	testing
5	All tests should be _________ to customer requirements.	traceable	designed	tested	coded	traceable
6	Tests should be planned long before ______________ begins.	testing	coding	specification	requirements	testing
7	Testing should begin in the _________ and progress toward testing in the large.	design	beginning	small	big	small

8	The less there is to test, the more _________ we can test it.	quickly	shortly	automatically	hardly	quickly
9	________ is a process of executing a program with the intend of finding an error.	testing	coding	planning	designing	testing
10	A good _________ is one that has a high probability of finding an as-yet-undiscovered error	planning	test case	objective	goal	test case
11	All _________ should be traceable to customer-requirements.	analysis	designs	tests	plans	tests
12	__________ is simple how easily a computer program can be tested.	software operability	software simplicity	software decomposability	software testability	software testability
13	The better it works, the more efficiently it can be testing. This characteristic is called ___________.	operability	observability	controllability	decomposability	operability
14	There are _________ characteristics in testability	5	6	7	8	7
15	What you see is what you test. This characteristic is called __________.	controllability	observability	decomposability	stability	observability
16	The better we can control the software, the more the testing can be automated and optimized. This characteristic is called __________.	operability	stability	understandability	controllability	controllability
17	By controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting. This characteristic is called _________.	decomposability	simplicity	stability	understandability	decomposability
18	. The less there is to test, the more quickly we can test it. This characteristic is called _________.	controllability	simplicity	operability	observability	simplicity

19	The fewer the changes, the fewer the disruptions to testing. This characteristic is called __________.	controllability	decomposability	stability	understandability	stability
20	The more information we have, the smarter we will test. This characteristic is called _________.	controllability	decomposability	stability	understandability	understandability
21	A good test has a high ___________ of finding an error.	probability	simplicity	understandability	stability	probability
22	A good test is not _________.	stable	redundant	simple	complex	redundant
23	White-box testing sometimes called _________.	control structure testing	condition testing	glass-box testing	black-box testing	glass-box testing
24	Logic errors and incorrect assumptions are inversely proportional to the ___________ that a program path will be executed	simplicity	probability	understandability	stability	probability
25	Typographical errors are _________.	redundant	simple	random	complex	random
26	One often believes that a _________ path is not likely to be executed when, in fact, it may be executed on a regular basis.	control	structural	physical	logical	logical
27	Basic path testing is a __________.	black-box testing	white-box testing	control structure testing	control path testing	white-box testing

28	__________ is a software metric that provides a quantitative measure of the logical complexity of a program.	cyclomatic complexity	flow graph	deriving test cases	graph matrices	cyclomatic complexity
29	An __________ is any path through the program that introduces atleast one new set of processing statements or a new condition.	dependent path	independent path	basic path	control path	independent path
30	There are _________ steps to be applied to derive the basis set.	2	3	4	5	4
31	There are _________ test cases that satisfy the basis set.	3	4	5	6	6
32	. A ________ is a square matrix whose size is equal to the number of nodes on the flow graph.	graph matrix	matrix	flow graph	cyclomatic complexity	graph matrix
33	To develop a software tool that assists in basis path testing, a data structure called a ___________ is useful.	matrix	flow graph	graph matrix	cyclomatic omplexity	graph matrix
34	____________ requires three or four tests to be derived for a relational expression.	branch testing	data flow testing	data control testing	domain testing	domain testing
35	__________ is probably the simplest condition testing strategy.	branch testing	data flow testing	condition testing	domain testing	branch testing
36	The __________ method selects test paths of a program according to the locations of definitions and uses of variables in the program	data flow testing	condition testing	loop testing	black box testing	data flow testing
37	__________ is a white box testing technique that focuses exclusively on the validity of loop constructions	data flow testing	loop testing	condition testing	control path testing	loop testing
38	___________ is a test case design method that exercises the logical conditions contained in a program module	black box testing	loop testing	data flow testing	condition testing	condition testing
39	_____________ is called behavioral testing.	black box testing	loop testing	data flow testing	condition testing	black box testing

40	The first step in ___________ is to understand the objects that are modeled in software and the relationships that connect these objects	black box testing	loop testing	data flow testing	condition testing	black box testing
41	Equivalence partitioning is a ____________ method that divides the input domain of a program into classes of data.	black box testing	loop testing	data flow testing	condition testing	black box testing
42	Comparison testing is also called _____________.	black box testing	loop testing	behavioral testing	back-to-back testing	back-to-back testing
43	___________ testing can be applied to problems in which the input domain is relatively small but too large to accommodate exhaustive testing.	orthogonal array	loop	behavioral	back-to-back	orthogonal array
44	___________ focuses verification effort on the smallest unit of software design – the software component or module.	module testing	unit testing	structure testing	system testing	unit testing
45	A driver is nothing more than a ___________.	subprogram	main program	stub	subroutine	main program
46	______________ serve to replace modules that are subordinate called by the component to be tested.	subprograms	main programs	stubs	subroutines	stubs
47	Drivers and __________ represent overhead.	subprograms	main programs	stubs	subroutines	stubs
48	____________ of execution paths is an essential task during the unit test.	unit testing	module testing	selective testing	white box testing	selective testing

49	Good _________ dictates that error conditions be anticipated and error-handling paths set up to reroute or cleanly terminate processing when an error does occur	design	testing	code	module	design
50	_________ is completely assembled as a package, interfacing errors have been uncovered and corrected.	software	program	code	all of the above	software
51	The Process of Configuration identification involves the specification of components in the software project are known as ________.	Configuration Items	Change Control	Configuration Control	Project control	Configuration Items
52	Implementing a quality system is spent on writing documents which specify how certain tasks are to be carried out is known as _______.	Procedures	Policies	Function	Definitions	Procedures
53	________ task involves the programmer to receive a specification of a module.	Integration Programming	System Programming	Unit Testing	Configuration Control	Unit Testing
54	Quality Assurance follows _______ methodology	Defect analysis	Defect Prevention	Error detection	Error correction	Defect Prevention
55	Quality Assurance is based on _________ work	Product Oriented	Function Oriented	Process Oriented	Design Oriented	Process Oriented
56	SEI means	Software Engineering Institute	Software Engineering International	Software Engineering Independent	System Engineering Institute	Software Engineering Institute
57	ISO means	Internal Organizations for standards	Intermediate Organizations for standards	International Organizations for standards	Internal optimization standards	International Organizations for standards

58	PCMM means	Personal Capability Maturity Model	People Capability Maturity Model	Professional Capability Maturity Model	Project Capability Maturity Model	People Capability Maturity Model
59	Quality Control is based on ______ methodology	Defect Prevention	Process Oriented	Defect Detection	debugging	Defect Detection
60	the document shows the relationship between requirement specification and test case is called ________	Matrix	Traceability Matrix	Defect Analysis	Matrix Analysis	Traceability Matrix

Testing Strategies & Tactics: Software Testing Fundamentals, Strategic Approach to Software Testing, Test Strategies for Conventional Software, Validation Testing, System testing Black-Box Testing, White-Box Testing and their type, Basis Path Testing

Software Testing Fundamentals

What is Software Testing:

Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect free in order to produce the quality product.

Definition:

According to **ANSI/IEEE 1059** standard – A process of analyzing a software item to detect the differences between existing and required conditions (i.e., defects) and to evaluate the features of the software item.

Software Testing Types:

Manual Testing: Manual testing is the process of testing the software manually to find the defects. Tester should have the perspective of an end users and to ensure all the features are working as mentioned in the requirement document. In this process, testers execute the test cases and generate the reports manually without using any automation tools.

Automation Testing: Automation testing is the process of testing the software using an automation tools to find the defects. In this process, testers execute the test scripts and generate the test results automatically by using automation tools. Some of the famous automation testing tools for functional testing are QTP/UFT and Selenium.

Types of testing

There are many types of testing like

- Unit Testing
- Integration Testing
- Functional Testing
- System Testing

- Stress Testing
- Performance Testing
- Usability Testing
- Acceptance Testing
- Regression Testing
- Beta Testing

Testing Methods:

1. Static Testing
2. Dynamic Testing

Verification And Validation In Software Testing

Verification And Validation:

In software testing, verification and validation are the processes to check whether a software system meets the specifications and that it fulfills its intended purpose or not. Verification and validation is also known as V & V. It may also be referred to as software quality control. It is normally the responsibility of software testers as part of the Software Development Life Cycle.

VERIFICATION: (Static Testing)

Verification is the process, to ensure that whether we are building the product right i.e., to verify the requirements which we have and to verify whether we are developing the product accordingly or not.

Activities involved here are Inspections, Reviews, Walkthroughs

VALIDATION: (Dynamic Testing)

Validation is the process, whether we are building the right product i.e., to validate the product which we have developed is right or not.

Software Testing - Validation Testing

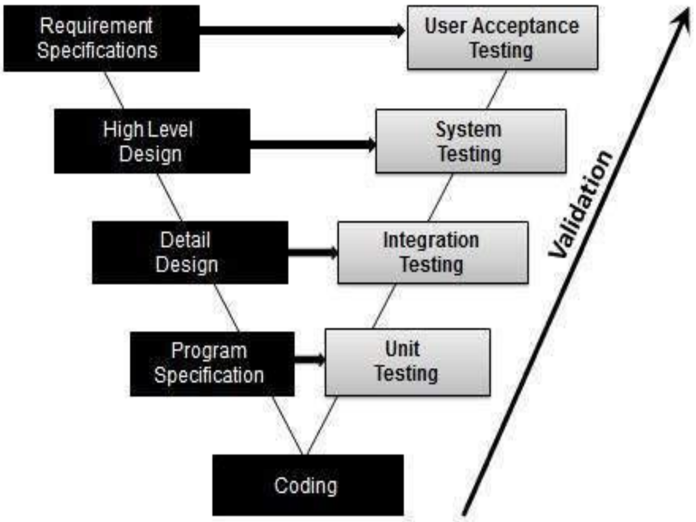
The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

It answers to the question, Are we building the right product?

Validation Testing - Workflow:

Validation testing can be best demonstrated using V-Model. The Software/product under test is evaluated during this type of testing.



Activities:

- Unit Testing
- Integration Testing
- System Testing
- User Acceptance Testing

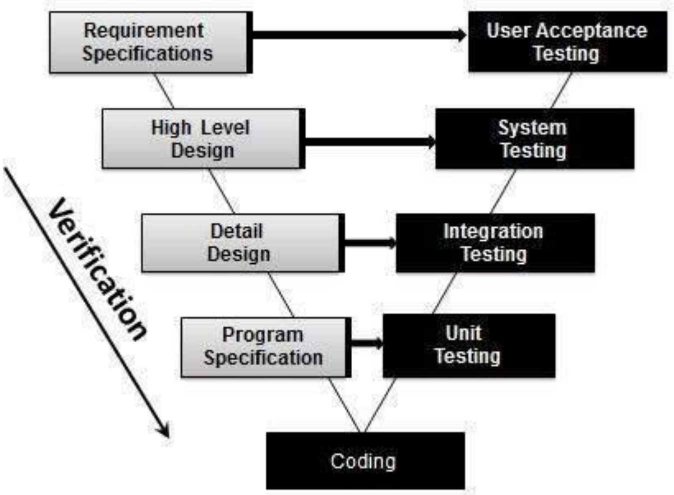
What is Verification Testing ?

Verification is the process of evaluating work-products of a development phase to determine whether they meet the specified requirements.

verification ensures that the product is built according to the requirements and design specifications. It also answers to the question, Are we building the product right?

Verification Testing - Workflow:

verification testing can be best demonstrated using V-Model. The artefacts such as test Plans, requirement specification, design, code and test cases are evaluated.



Activities:

- Reviews
- Walkthroughs
- Inspection

What is Unit Testing?

Unit testing, a testing technique using which individual modules are tested to determine if there are any issues by the developer himself. It is concerned with functional correctness of the standalone modules.

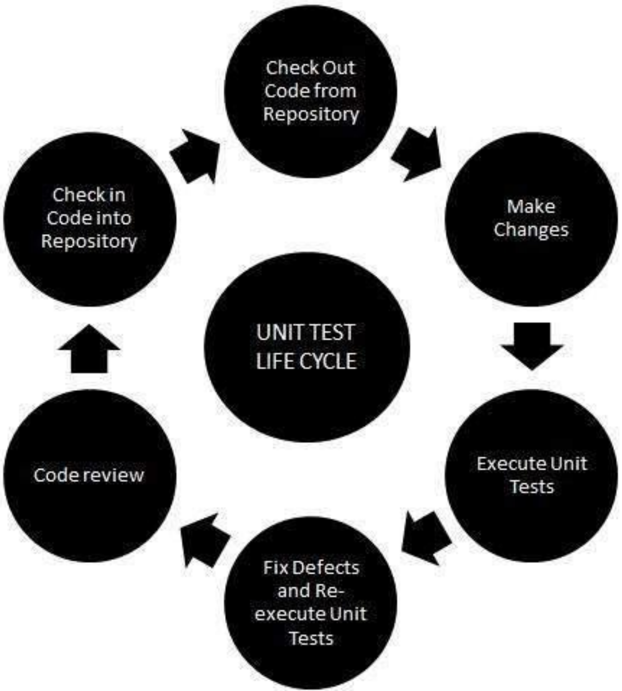
The main aim is to isolate each unit of the system to identify, analyze and fix the defects.

Unit Testing - Advantages:

- Reduces Defects in the Newly developed features or reduces bugs when changing the existing functionality.

- Reduces Cost of Testing as defects are captured in very early phase.
- Improves design and allows better refactoring of code.
- Unit Tests, when integrated with build gives the quality of the build as well.

Unit Testing LifeCyle:


Check Out Code from Repository
Make Changes
Execute Unit Tests
Fix Defects and Re-execute Unit Tests
Code review
Check in Code into Repository
UNIT TEST LIFE CYCLE

Unit Testing Techniques:

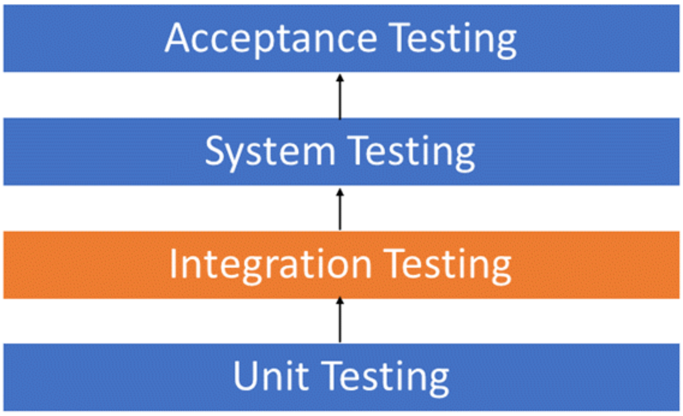
- **Black Box Testing -** Using which the user interface, input and output are tested.
- **White Box Testing -** used to test each one of those functions behaviour is tested.
- **Gray Box Testing -** Used to execute tests, risks and assessment methods.

What is Integration Testing?

Upon completion of unit testing, the units or modules are to be integrated which gives raise to integration testing. The purpose of integration testing is to verify the functional, performance, and reliability between the modules that are integrated.

Integration Strategies:

- Big-Bang Integration
- Top Down Integration
- Bottom Up Integration
- Hybrid Integration



Big Bang Approach:

Here all component are integrated together at **once** and then tested.

Advantages:

- Convenient for small systems.

Incremental Approach

In this approach, testing is done by joining two or more modules that are ***logically related***. Then the other related modules are added and tested for the proper functioning. The process continues until all of the modules are joined and tested successfully.

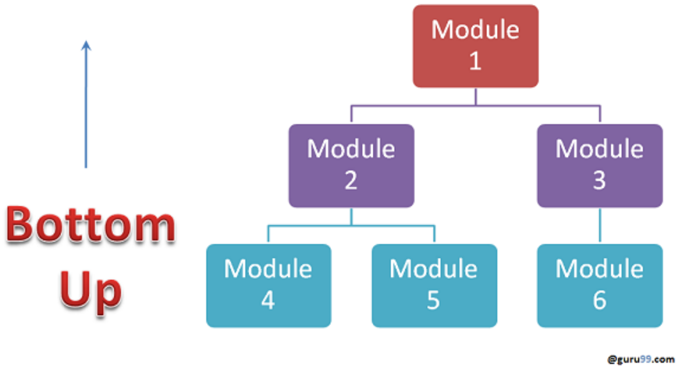
Incremental Approach, in turn, is carried out by two different Methods:

- Bottom Up
- Top Down

Bottom-up Integration

In the bottom-up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing

Diagrammatic Representation:



Advantages:

- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

Disadvantages:

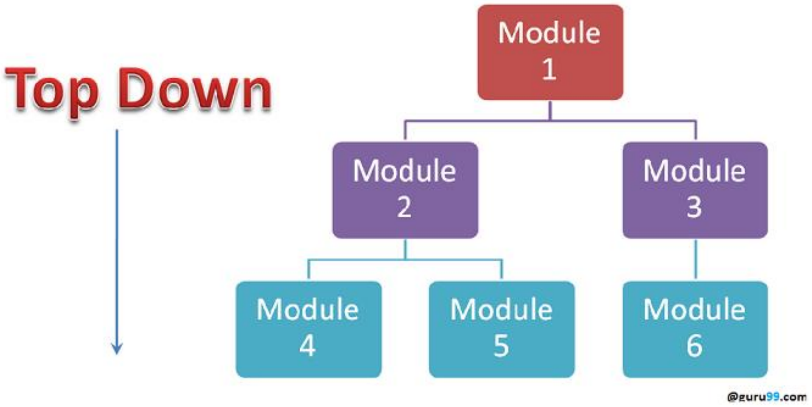
- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- An early prototype is not possible

Top-down Integration:

In Top to down approach, testing takes place from top to down following the control flow of the software system.

Takes help of stubs for testing.

Diagrammatic Representation:



Advantages:

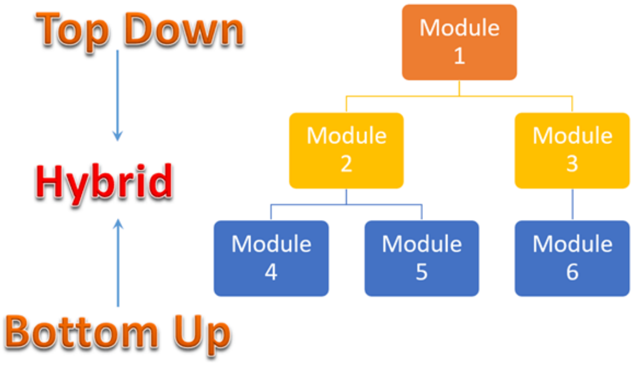
- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

Disadvantages:

- Needs many Stubs.
- Modules at a lower level are tested inadequately.

Hybrid/ Sandwich Integration

In the sandwich/hybrid strategy is a combination of Top Down and Bottom up approaches. Here, top modules are tested with lower modules at the same time lower modules are integrated with top modules and tested. This strategy makes use of stubs as well as drivers.



How to do Integration Testing?

The Integration test procedure irrespective of the Software testing strategies (discussed above):

1. Prepare the Integration Tests Plan
2. Design the Test Scenarios, Cases, and Scripts.
3. Executing the test Cases followed by reporting the defects.
4. Tracking & re-testing the defects.
5. Steps 3 and 4 are repeated until the completion of Integration is successful.

Strategic Approach to Software Testing

What is Test Strategy?

Test Strategy is also known as test approach defines how testing would be carried out. Test approach has two techniques:

- **Proactive -** An approach in which the test design process is initiated as early as possible in order to find and fix the defects before the build is created.
- **Reactive -** An approach in which the testing is not started until after design and coding are completed.

A strategy of software testing is shown in the context of spiral.

Following figure shows the testing strategy:

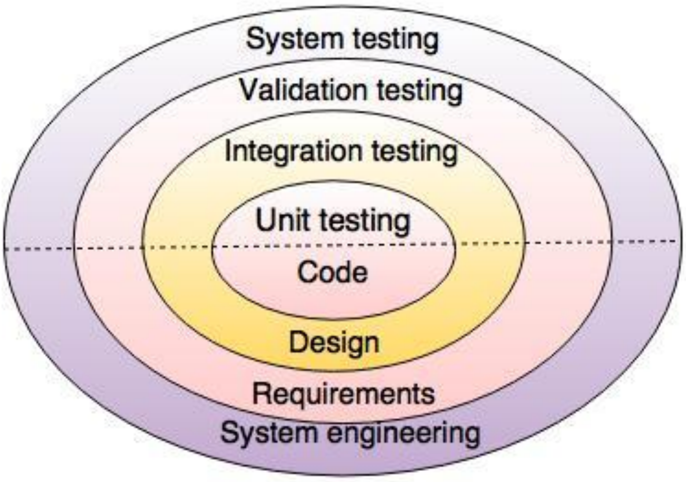


Fig. - Testing Strategy

Unit testing

Unit testing starts at the centre and each unit is implemented in source code.

Integration testing

An integration testing focuses on the construction and design of the software.

Validation testing

Check all the requirements like functional, behavioral and performance requirement are validate against the construction software.

System testing

System testing confirms all system elements and performance are tested entirely.

Testing strategy for procedural point of view

As per the procedural point of view the testing includes following steps.

1) Unit testing
2) Integration testing
3) High-order tests
4) Validation testing

These steps are shown in following figure:

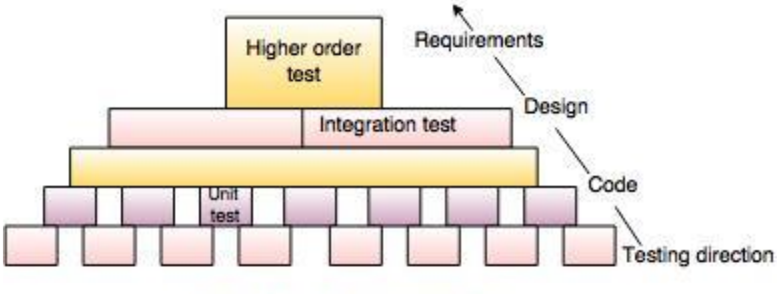


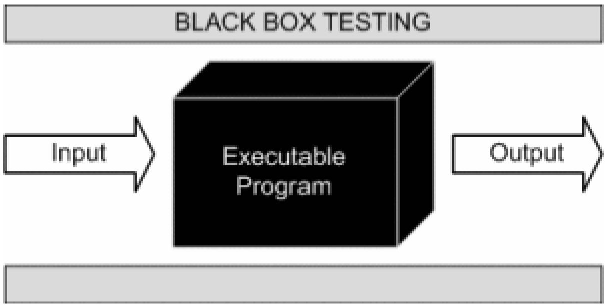
Fig.- Steps of software testing

Testing Approaches:

1. White Box Testing
2. Black Box Testing
3. Grey Box Testing

BLACK BOX TESTING

BLACK BOX TESTING, also known as Behavioral Testing, is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.



This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following **categories:**

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Behavior or performance errors
- Initialization and termination errors

Definition by ISTQB

- **black box testing:** Testing, either functional or non-functional, without reference to the internal structure of the component or system.
- **black box test design technique:** Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.

Example

A tester, without knowledge of the internal structures of a website, tests the web pages by using a browser; providing inputs (clicks, keystrokes) and verifying the outputs against the expected outcome.

Levels Applicable To

Black Box Testing method is applicable to the following levels of software testing:

- Integration Testing
- System Testing
- Acceptance Testing

The higher the level, and hence the bigger and more complex the box, the more black-box testing method comes into use.

Techniques

Following are some techniques that can be used for designing black box tests.

- *Equivalence Partitioning:* It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.
- *Boundary Value Analysis:* It is a software test design technique that involves the determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.
- *Cause-Effect Graphing:* It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly.

Advantages

- Tests are done from a user's point of view and will help in exposing discrepancies in the specifications.
- Tester need not know programming languages or how the software has been implemented.
- Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias.
- Test cases can be designed as soon as the specifications are complete.

Disadvantages

- Only a small number of possible inputs can be tested and many program paths will be left untested.
- Without clear specifications, which is the situation in many projects, test cases will be difficult to design.
- Tests can be redundant if the software designer/developer has already run a test case.
- Ever wondered why a soothsayer closes the eyes when foretelling events? So is almost the case in Black Box Testing.

BLACK BOX TESTING:

It is also called as Behavioral/Specification-Based/Input-Output Testing

Black Box Testing is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure. This can be applied to every level of software testing such as Unit, Integration, System and Acceptance Testing.

Testers create test scenarios/cases based on software requirements and specifications. So it is AKA Specification Based Testing.

Tester performs testing only on the functional part of an application to make sure the behavior of the software is as expected. So it is AKA Behavioral Based Testing.

The tester passes input data to make sure whether the actual output matches the expected output. So it is AKA Input-Output Testing.

Black Box Testing Techniques:

1. Equivalence Partitioning
2. Boundary Value Analysis
3. Decision Table
4. State Transition

Equivalence Partitioning:

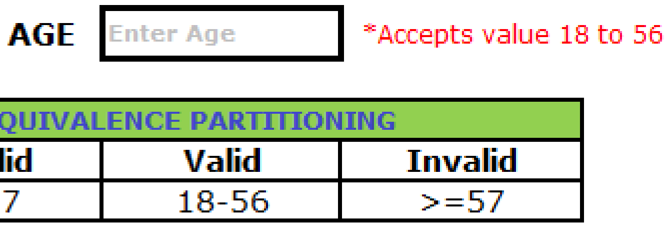
Equivalence Partitioning is also known as Equivalence Class Partitioning. In equivalence partitioning, inputs to the software or system are divided into groups that are expected to exhibit similar behavior, so they are likely to be proposed in the same way. Hence selecting one input from each group to design the test cases. Click here to see detailed post on equivalence partitioning.

Equivalence partitioning is applicable at all levels of testing.

Example on Equivalence Partitioning Test Case Design Technique:

Example 1:

Assume, we have to test a field which accepts Age 18 – 56



EQUIVALENCE PARTITIONING		
Invalid	Valid	Invalid
<=17	18-56	>=57

Valid Input: 18 – 56

Invalid Input: less than or equal to 17 (<=17), greater than or equal to 57 (>=57)

Valid Class: 18 – 56 = Pick any one input test data from 18 – 56

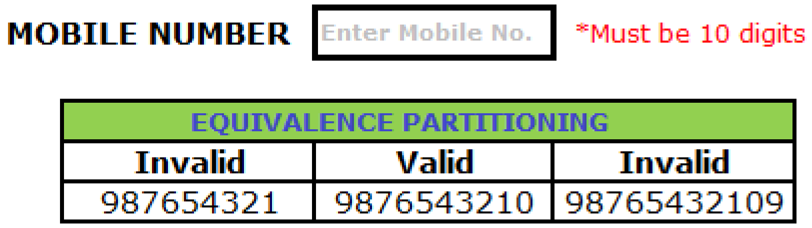
Invalid Class 1: <=17 = Pick any one input test data less than or equal to 17

Invalid Class 2: >=57 = Pick any one input test data greater than or equal to 57

We have one valid and two invalid conditions here.

Example 2:

Assume, we have to test a filed which accepts a Mobile Number of ten digits.



MOBILE NUMBER Enter Mobile No. *Must be 10 digits

EQUIVALENCE PARTITIONING		
Invalid	Valid	Invalid
987654321	9876543210	98765432109

Valid input: 10 digits

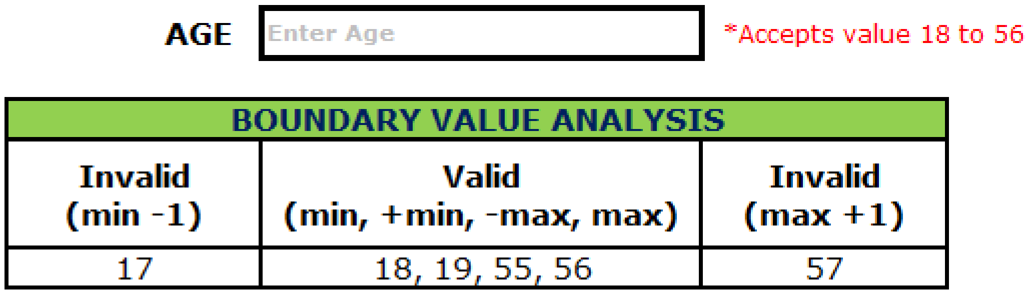
Invalid Input: 9 digits, 11 digits

Boundary Value Analysis:

Boundary value analysis (BVA) is based on testing the boundary values of valid and invalid partitions. The Behavior at the edge of each equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects. Click here to see detailed post on boundary value analysis.

Example on Boundary Value Analysis Test Case Design Technique:

Assume, we have to test a field which accepts Age 18 – 56



AGE Enter Age *Accepts value 18 to 56

BOUNDARY VALUE ANALYSIS		
Invalid (min -1)	Valid (min, +min, -max, max)	Invalid (max +1)
17	18, 19, 55, 56	57

Minimum boundary value is 18

Maximum boundary value is 56

Valid Inputs: 18,19,55,56

Invalid Inputs: 17 and 57

Test case 1: Enter the value 17 (18-1) = Invalid

Test case 2: Enter the value 18 = Valid

Test case 3: Enter the value 19 (18+1) = Valid

Test case 4: Enter the value 55 (56-1) = Valid

Test case 5: Enter the value 56 = Valid

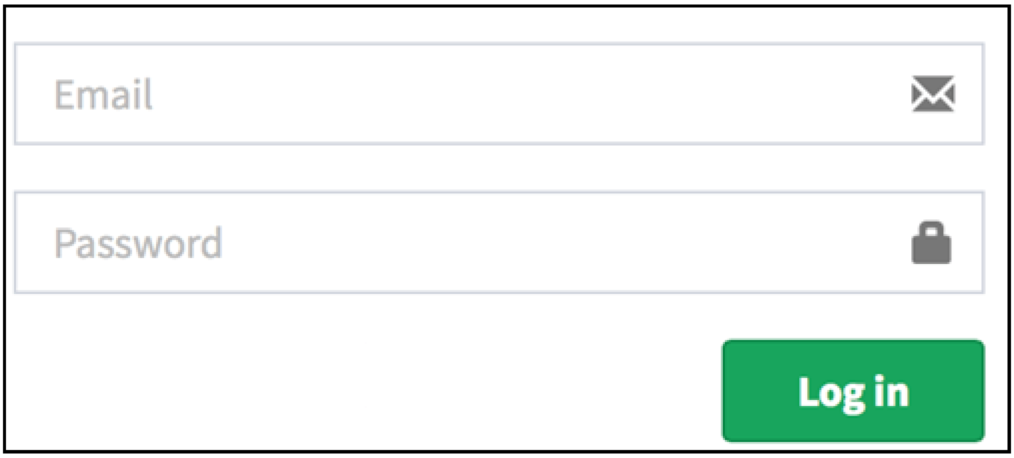
Test case 6: Enter the value 57 (56+1) =Invalid

Decision Table:

Decision Table is aka Cause-Effect Table. This test technique is appropriate for functionalities which has logical relationships between inputs (if-else logic). In Decision table technique, we deal with combinations of inputs. To identify the test cases with decision table, we consider conditions and actions. We take conditions as inputs and actions as outputs. Click here to see detailed post on decision table.

Example 1: How to make Decision Base Table for Login Screen

Let's create a decision table for a login screen.



The condition is simple if the user provides correct username and password the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.

Conditions	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

Legend:

- T – Correct username/password
- F – Wrong username/password
- E – Error message is displayed
- H – Home screen is displayed

Interpretation:

- Case 1 – Username and password both were wrong. The user is shown an error message.
- Case 2 – Username was correct, but the password was wrong. The user is shown an error message.
- Case 3 – Username was wrong, but the password was correct. The user is shown an error message.
- Case 4 – Username and password both were correct, and the user navigated to homepage

State Transition:

Using state transition testing, we pick test cases from an application where we need to test different system transitions. We can apply this when an application gives a different output for the same input, depending on what has happened in the earlier state. Click here to see detailed post on state transition technique.

Types of Black Box Testing:

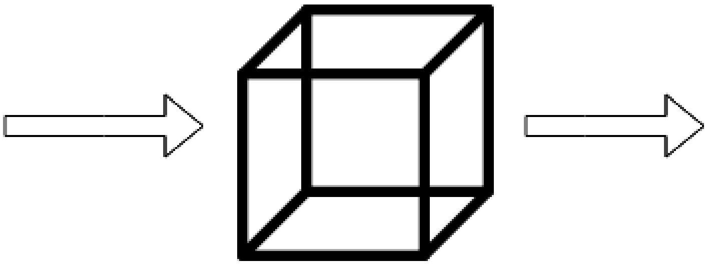
Functionality Testing: In simple words, what the system actually does is functional testing

Non-functionality Testing: In simple words, how well the system performs is non-functionality testing

WHITE BOX TESTING

WHITE BOX TESTING (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty-gritty of a system.

This method is named so because the software program, in the eyes of the tester, is like a white/transparent box; inside which one clearly sees.



Whitebox Testing

Definition by ISTQB

- **white-box testing:** Testing based on an analysis of the internal structure of the component or system.
- **white-box test design technique:** Procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.

Example

A tester, usually a developer as well, studies the implementation code of a certain field on a webpage, determines all legal (valid and invalid) AND illegal inputs and verifies the outputs against the expected outcomes, which is also determined by studying the implementation code.

White Box Testing is like the work of a mechanic who examines the engine to see why the car is not moving.

Levels Applicable To

White Box Testing method is applicable to the following levels of software testing:

- Unit Testing: For testing paths within a unit.
- Integration Testing: For testing paths between units.
- System Testing: For testing paths between subsystems.

However, it is mainly applied to Unit Testing.

Advantages

- Testing can be commenced at an earlier stage. One need not wait for the GUI to be available.
- Testing is more thorough, with the possibility of covering most paths.

Disadvantages

- Since tests can be very complex, highly skilled resources are required, with a thorough knowledge of programming and implementation.
- Test script maintenance can be a burden if the implementation changes too frequently.
- Since this method of testing is closely tied to the application being tested, tools to cater to every kind of implementation/platform may not be readily available.

WHITE BOX TESTING:

It is also called as Glass Box, Clear Box, Structural Testing.

White Box Testing is based on applications internal code structure. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. This testing usually done at the unit level.

White Box Testing Techniques:

1. Statement Coverage
2. Branch Coverage
3. Path Coverage

Coverage

White Box Testing is coverage of the specification in the code:

1. Code coverage
2. Segment coverage: Ensure that each code statement is executed once.
3. Branch Coverage or Node Testing: Coverage of each code branch in from all possible was.
4. Compound Condition Coverage: For multiple conditions test each condition with multiple paths and combination of the different path to reach that condition.
5. Basis Path Testing: Each independent path in the code is taken for testing.
6. Data Flow Testing (DFT): In this approach you track the specific variables through each possible calculation, thus defining the set of intermediate paths through the code.DFT tends to reflect dependencies but it is mainly through sequences of data manipulation. In short, each data variable is tracked and its use is verified. This approach tends to uncover bugs like variables used but not initialize, or declared but not used, and so on.
7. Path Testing: Path testing is where all possible paths through the code are defined and covered. It's a time-consuming task.
8. Loop Testing: These strategies relate to testing single loops, concatenated loops, and nested loops. Independent and dependent code loops and values are tested by this approach.

White Box Testing Example

Consider the below simple pseudocode:

```
INPUT A & B

C = A + B

IF C>100

PRINT "ITS DONE"
```

Statement Coverage

For ***Statement Coverage*** – we would only need one test case to check all the lines of the code.

That means:
If I consider *TestCase_01 to be (A=40 and B=70),* then all the lines of code will be executed.

Now the question arises:

1. Is that sufficient?

2. What if I consider my Test case as A=33 and B=45?

Because Statement coverage will only cover the true side, for the pseudo code, only one test case would NOT be sufficient to test it. As a tester, we have to consider the negative cases as well.

Branch Coverage

Hence for maximum coverage, we need to consider **"*Branch Coverage*"**, which will evaluate the "FALSE" conditions.
In the real world, you may add appropriate statements when the condition fails.

So now the pseudocode becomes:

```
INPUT A & B
C = A + B
IF C>100
PRINT "ITS DONE"
ELSE
PRINT "ITS PENDING"
```

Since Statement coverage is not sufficient to test the entire pseudo code, we would require Branch coverage to ensure maximum coverage**.**
So for Branch coverage, we would require two test cases to complete the testing of this pseudo code.

TestCase_01: A=33, B=45
TestCase_02: A=25, B=30
With this, we can see that each and every line of the code is executed at least once.

Here are the Conclusions that are derived so far:

- Branch Coverage ensures more coverage than Statement coverage.
- Branch coverage is more powerful than Statement coverage.
- 100% Branch coverage itself means 100% statement coverage.
- But 100 % statement coverage does not guarantee 100% branch coverage.

Path Coverage

Now let's move on to ***Path Coverage:***
As said earlier, Path coverage is used to test the complex code snippets, which basically involve loop statements or combination of loops and decision statements.

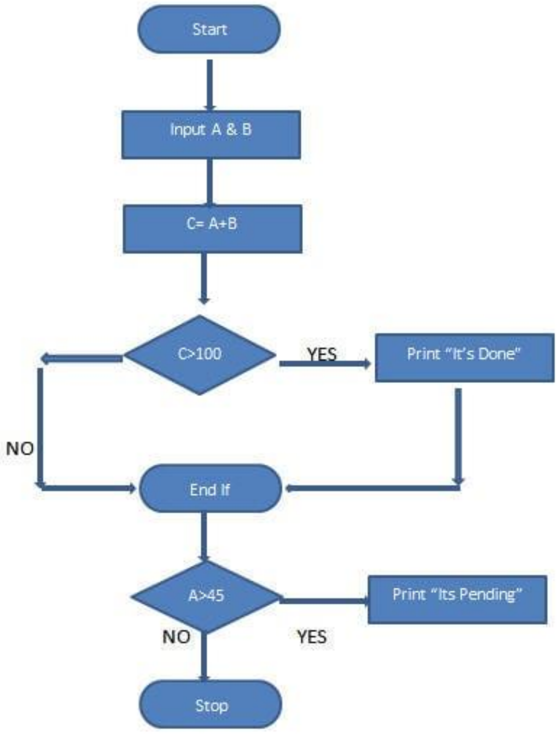
Consider this pseudocode:

```
INPUT A & B

C = A + B

IF C>100

PRINT “ITS DONE”

END IF

IF A>50

PRINT “ITS PENDING”

END IF
```

Now to ensure maximum coverage, we would require 4 test cases.

How? Simply – there are 2 decision statements, so for each decision statement, we would need two branches to test. One for true and the other for the false condition. So for 2 decision statements, we would require 2 test cases to test the true side and 2 test cases to test the false side, which makes a total of 4 test cases.

To simplify these let's consider below flowchart of the pseudo code we have:

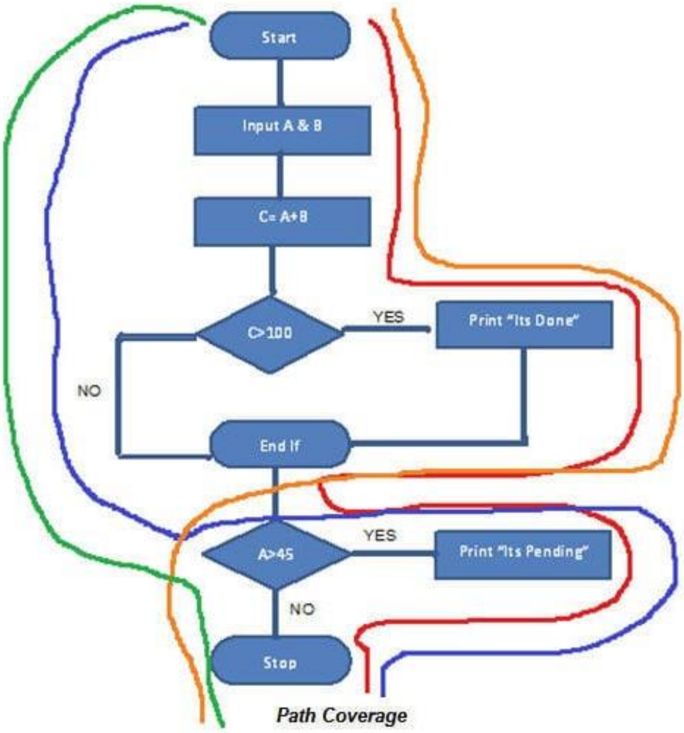


Path Coverage

In order to have the full coverage, we would need following test cases:

TestCase_01: A=50, B=60
TestCase_02: A=55, B=40
TestCase_03: A=40, B=65
TestCase_04: A=30, B=30

So the path covered will be:



Path Coverage

Red Line – TestCase_01 = (A=50, B=60)

Blue Line = TestCase_02 = (A=55, B=40)

Orange Line = TestCase_03 = (A=40, B=65)

Green Line = TestCase_04 = (A=30, B=30)

GRAY BOX TESTING

GRAY BOX TESTING is a software testing method which is a combination of Black Box Testing method and White Box Testing method. In Black Box Testing, the internal structure of the item being tested is unknown to the tester and in White Box Testing the internal structure is known. In Gray Box Testing, the internal structure is partially known. This involves having access to internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level.

Gray Box Testing is named so because the software program, in the eyes of the tester is like a gray/semi-transparent box; inside which one can partially see.

Example

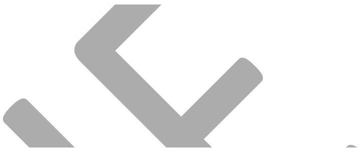
An example of Gray Box Testing would be when the codes for two units/modules are studied (White Box Testing method) for designing test cases and actual tests are conducted using the exposed interfaces (Black Box Testing method).

Levels Applicable To

Though Gray Box Testing method may be used in other levels of testing, it is primarily used in Integration Testing.

Spelling

Note that *Gray* is also spelled as *Grey*. Hence Grey Box Testing and Gray Box Testing mean the same.



Basic Path Testing

What is Path Testing?

Path testing is a structural testing method that involves using the source code of a program in order to find every possible executable path. It helps to determine all faults lying within a piece of code. This method is designed to execute all or selected path through a computer program.

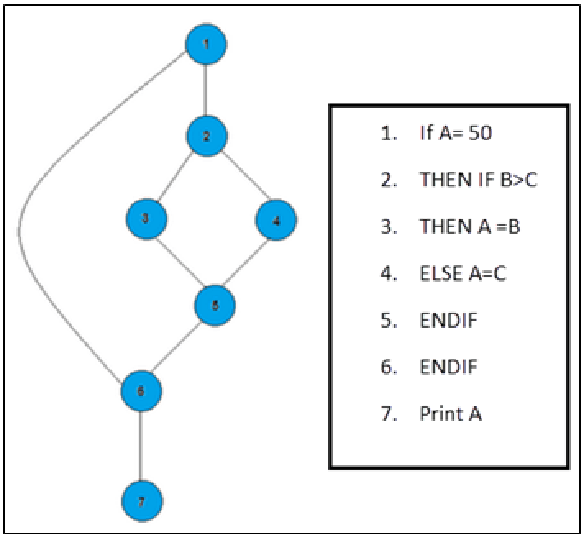
Any software program includes, multiple entry and exit points. Testing each of these points is a challenging as well as time-consuming. In order to reduce the redundant tests and to achieve maximum test coverage, basis path testing is used.

What is Basis Path Testing?

The basis path testing is same, but it is based on a White Box Testing method, that defines test cases based on the flows or logical path that can be taken through the program. In software engineering, Basis path testing involves execution of all possible blocks in a program and achieves maximum path coverage with the least number of test cases. It is a hybrid of branch testing and path testing methods.

The objective behind basis path in software testing is that it defines the number of independent paths, thus the number of test cases needed can be defined explicitly (maximizes the coverage of each test case).

Here we will take a simple example, to get a better idea what is basis path testing include



```
1. If A= 50
2. THEN IF B>C
3. THEN A =B
4. ELSE A=C
5. ENDIF
6. ENDIF
7. Print A
```

In the above example, we can see there are few conditional statements that is executed depending on what condition it suffice. Here there are 3 paths or condition that need to be tested to get the output,

- **Path 1**: 1,2,3,5,6, 7
- **Path 2**: 1,2,4,5,6, 7
- **Path 3**: 1, 6, 7

Steps for Basis Path testing

The basic steps involved in basis path testing include

- Draw a control graph (to determine different program paths)
- Calculate Cyclomatic complexity (metrics to determine the number of independent paths)
- Find a basis set of paths
- Generate test cases to exercise each path

Advantages of Basic Path Testing

- It helps to reduce the redundant tests
- It focuses attention on program logic
- It helps facilitates analytical versus arbitrary case design
- Test cases which exercise basis set will execute every statement in a program at least once
- Basis path testing helps to determine all faults lying within a piece of code.

UNIT V

PART A(Online)

PART B (2 Marks)

1. Define abstraction.
2. What do you mean by an error?
3. Differentiate between refinement and refactoring
4. Compare black box and white box testing
5. Write the difference between transform flow and transaction flow
6. List the different types of loops in testing
7. What is transform mapping?
8. What is validation testing?
9. Define transaction mapping.
10. What is the use of system testing?

PART C (6 Marks)

1. Explain Graph based testing methods in Black Box testing.
2. Demonstrate Flow graph notation and Independent program path in Basis path testing.
3. Demonstrate in detail about Validation testing
4. Explain in detail about Equivalence Partitioning
5. Discuss about Boundary value analysis.
6. Write in detail about Software Testing Fundamentals.
7.Illustrate in detail about System testing.
8. Write short notes on condition testing.
9. Illustrate the use of dataflow testing in software engineering process.
10. Discuss in detail about orthogonal array testing.
11. Illustrate loop testing and its types.