MODULE l

SOFTWARE ENGINEERING

Lecture 1

SOFTWARE ENGINEERING

Let us first understand what software engineering stands for. The term is made of two words, software and engineering.

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.

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.

Need of Software Engineering

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- **Large software -** It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
- **Scalability-** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- **Cost-** As hardware industry has shown its skills and huge manufacturing has lower down he price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
- **Dynamic Nature-** The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- **Quality Management-** Better process of software development provides better and quality software product.

Characteristics of a Good Software

1) Operational Characteristics
2) Transition Characteristics
3) Revision Characteristics

What Operational Characteristics should a software have?
These are functionality-based factors and related to 'exterior quality' of software. Various Operational Characteristics of software are

a) **Correctness:** The software which we are making should meet all the specifications stated by the customer.

b) **Usability/Learnability:** The amount of efforts or time required to learn how to use the software should be less. This makes the software user-friendly even for IT-illiterate people.

c) **Integrity:** Just like medicines have side-effects, in the same way a software may have a side-effect i.e. it may affect the working of another application. But a quality software should not have side effects.

d) **Reliability:** The software product should not have any defects. Not only this, it shouldn't fail while execution.

e) **Efficiency:** This characteristic relates to the way software uses the available resources. The software should make effective use of the storage space and execute command as per desired timing requirements.

f) **Security:** With the increase in security threats nowadays, this factor is gaining importance. The software shouldn't have ill effects on data / hardware. Proper measures should be taken to keep data secure from external threats.

g) **Safety:** The software should not be hazardous to the environment/life.

What are the Revision Characteristics of software?
These engineering-based factors of the relate to 'interior quality' of the software like efficiency, documentation and structure. These factors should be in-build in any good software. Various Revision Characteristics of software are: -

a) **Maintainability:** Maintenance of the software should be easy for any kind of user.

b) **Flexibility:** Changes in the software should be easy to make.

c) **Extensibility:** It should be easy to increase the functions performed by it.

d) **Scalability:** It should be very easy to upgrade it for more work (or for more number of users).

e) **Testability:** Testing the software should be easy.

f) **Modularity:** Any software is said to made of units and modules which are independent of each other. These modules are then integrated to make the final software. If the software is divided into separate independent parts that can be modified, tested separately, it has high modularity.

Transition Characteristics of the software:

a) **Interoperability:** Interoperability is the ability of software to exchange information with other applications and make use of information transparently.
b) **Reusability:** If we are able to use the software code with some modifications for different purpose then we call software to be reusable.
c) **Portability:** The ability of software to perform same functions across all environments and platforms, demonstrate its portability.

Importance of any of these factors varies from application to application. In systems where, human life is at stake, integrity and reliability factors must be given prime importance. In any business-related application usability and maintainability are key factors to be considered. Always remember in Software Engineering, quality of software is everything, therefore try to deliver a product which has all these characteristics and qualities.

Lecture 2

SOFTWARE APPLICATION

System software: This class of software manages and controls the internal operations of a computer system. It is a group of programs, which is responsible for using computer resources efficiently and effectively. For example, an operating system ′
• Real-time software: This class of software observes, analysis, and controls real world events as they occur an example of real-time software is the software used for weather forecasting that collects and processes parameters like temperature and humidity ′
• Business software: This class of software is widely used in areas where management and control of financial activities is of utmost importance. The fundamental component of a business system comprises payroll, inventory, and accounting software
• Engineering and scientific software: This class of software has emerged as a powerful tool in the research and development of next generation technology ′
• Artificial intelligence (AI) software: This class of software is used where the problem-solving technique is non-algorithmic in nature. ′
• Web-based software: This class of software acts as an interface between the user and the Internet
• Personal computer (PC) software: This class of software is used for both official and personal use.

Software life cycle

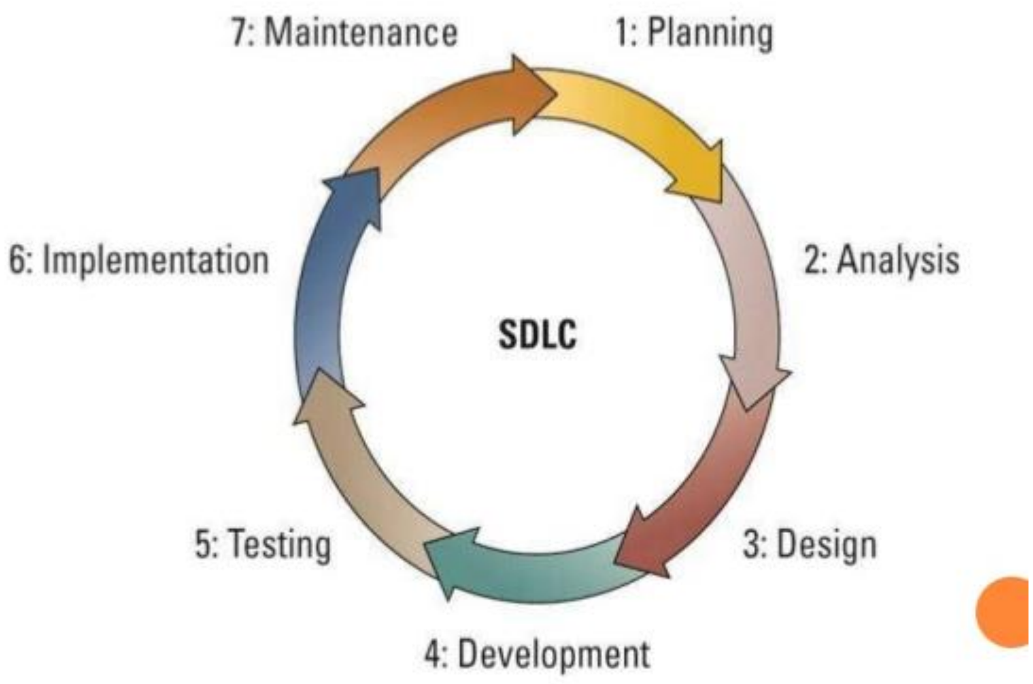
The life cycle of a software represents the series of identifiable stages through which it evolves during its life time

software life cycle has been defined to imply the different stages (or
phases) over which a software evolves from an initial customer request
for it, to a fully developed software, and finally to a stage where it is no longer useful to any user, and then it is discarded.

Software development life cycle (SDLC) model

A software development life cycle (SDLC) model (also called software life cycle model and software development process model) describes the different activities that need to be carried out for the software to evolve in its life cycle. Throughout our discussion, we shall use the terms software development lifecycle (SDLC) and software development process interchangeably.

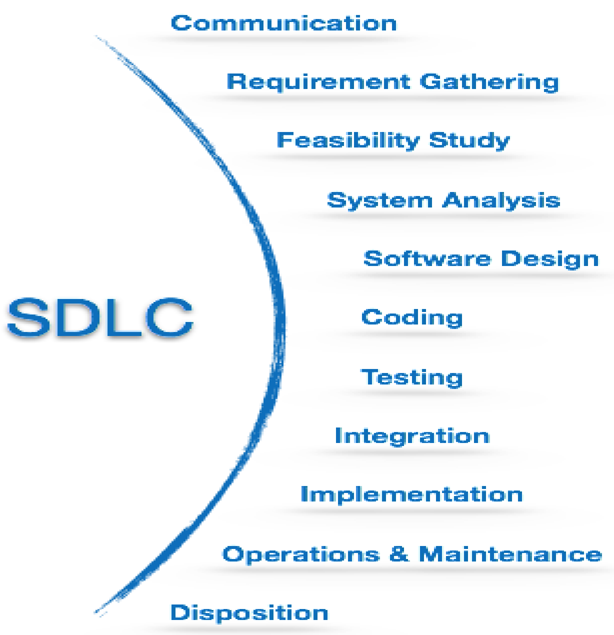
SDLC PHASES



An SDLC graphically depicts the different phases through which a software evolves. It is usually accompanied by a textual description of the different activities that need to be carried out during each phase.

SDLC Activities

SDLC provides a series of steps to be followed to design and develop a software product efficiently. SDLC framework includes the following steps:



Communication

This is the first step where the user initiates the request for a desired software product. He contacts the service provider and tries to negotiate the terms. He submits his request to the service providing organization in writing.

Requirement Gathering

This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and tries to bring out as much information as possible on their requirements. The requirements are contemplated and segregated into user requirements, system requirements and functional requirements. The requirements are collected using a number of practices as given -

- studying the existing or obsolete system and software,
- conducting interviews of users and developers,
- referring to the database or
- collecting answers from the questionnaires.

Feasibility Study

After requirement gathering, the team comes up with a rough plan of software process. At this steps the team analysis if software can be made to fulfil all requirements of the user and if there is any possibility of software being no more useful. It is found out, if the project is financially, practically and technologically feasible for the organization to take up. There are many algorithms available, which help the developers to conclude the feasibility of a software project.

System Analysis

At this step the developers decide a roadmap of their plan and try to bring up the best software model suitable for the project. System analysis includes Understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc. The project team analyses the scope of the project and plans the schedule and resources accordingly.

Software Design

Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs; logical design and physical design. Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams and in some cases pseudo codes.

Coding

This step is also known as programming phase. The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently.

Testing

An estimate says that 50% of whole software development process should be tested. Errors may ruin the software from critical level to its own removal. Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing, program testing, product testing, in-house testing and testing the product at user’s end. Early discovery of errors and their remedy is the key to reliable software.

Integration

Software may need to be integrated with the libraries, databases and other program(s). This stage of SDLC is involved in the integration of software with outer world entities.

Implementation

This means installing the software on user machines. At times, software needs post-installation configurations at user end. Software is tested for portability and adaptability and integration related issues are solved during implementation.

Operation and Maintenance

This phase confirms the software operation in terms of more efficiency and less errors. If required, the users are trained on, or aided with the documentation on how to operate the software and how to keep the software operational. The software is maintained timely by updating the code according to the changes taking place in user end environment or technology. This phase may face challenges from hidden bugs and real-world unidentified problems.

Disposition

As time elapses, the software may decline on the performance front. It may go completely obsolete or may need intense upgradation. Hence a pressing need to eliminate a major portion of the system arises. This phase includes archiving data and required software components, closing down the system, planning disposition activity and terminating system at appropriate end-of-system time.

Lecture 3

WATERFALL MODEL AND ITS EXTENSIONS

The waterfall model is possibly the most obvious and intuitive
way in which software can be developed through team effort. We can think of the waterfall model as a generic model that has been extended in many ways for catering to certain specific software development situations to realise all other software life cycle models.

Classical Waterfall Model

Classical waterfall model is the basic **software development life cycle** model. It is very simple but idealistic. Earlier this model was very popular but nowadays it is not used. But it is very important because all the other software development life cycle models are based on the classical waterfall model.
Classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after completion of the previous phase. That is the output of one phase will be the input to the next phase. Thus, the development process can be considered as a sequential flow in the waterfall. Here the phases do not overlap with each other. The different sequential phases of the classical waterfall model are shown in the below figure:

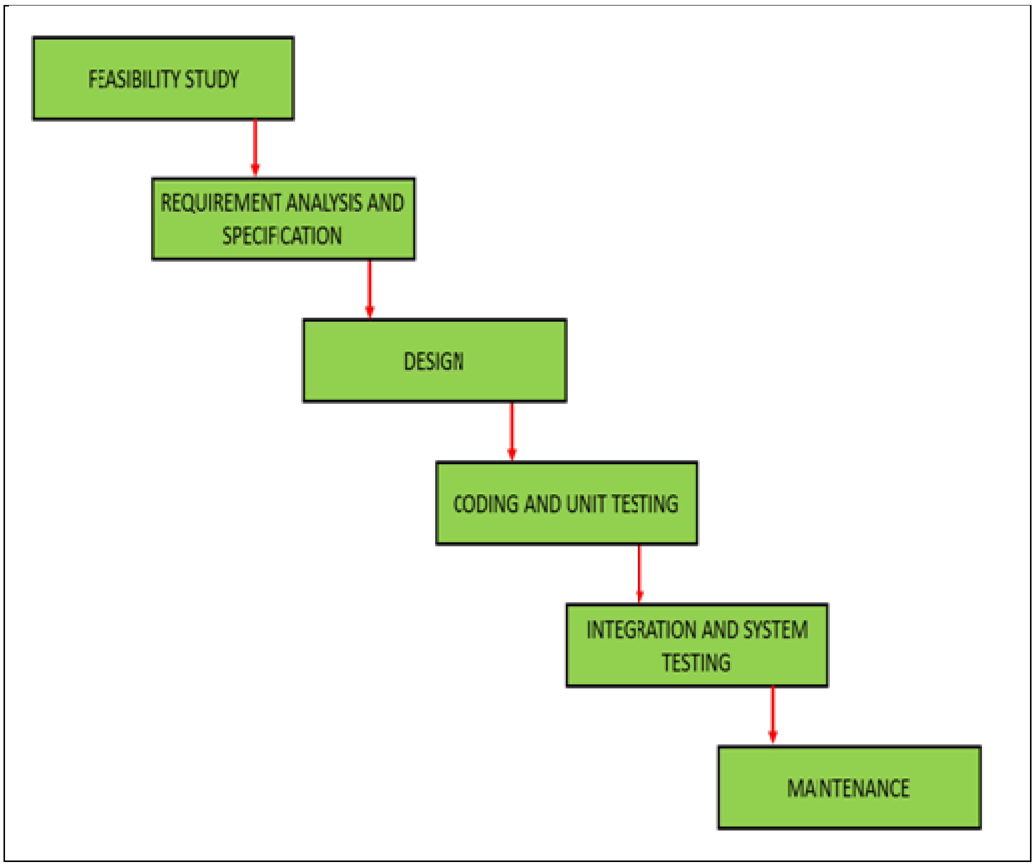

FEASIBILITY STUDY
REQUIREMENT ANALYSIS AND SPECIFICATION
DESIGN
CODING AND UNIT TESTING
INTEGRATION AND SYSTEM TESTING
MAINTENANCE

Fig. 1.1 Classical waterfall model.

1. **Feasibility Study**: The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software. The feasibility studies
2. involves understanding the problem and then determine the various possible strategies to solve the problem. These different identified solutions are analysed based on their

benefits and drawbacks, the best solution is chosen and all the other phases are carried out as per this solution strategy.

Requirements analysis and specification: The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.

1.
   - **Requirement gathering and analysis:** Firstly, all the requirements regarding the software are gathered from the customer and then the gathered requirements are analysed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (inconsistent requirement is one in which some part of the requirement contradicts with some other part).
   - **Requirement specification:** These analysed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

2. **Design**: The aim of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.

3. **Coding and Unit testing**: In coding phase software design is translated into source code using any suitable programming language. Thus, each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.

4. **Integration and System testing**: Integration of different modules are undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this.
   System testing consists three different kinds of testing activities as described below:
   - **α-testing:** α-testing is the system testing performed by the development team.
   - **β-testing:** β-testing is the system testing performed by a friendly set of customers.
   - **Acceptance testing:** After the software has been delivered, the customer performed the acceptance testing to determine whether to accept the delivered software or to reject it.

5. **Maintenance:** Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is the 60% of the total effort spent to develop a full software. There are basically three types of maintenance:
   - **Corrective Maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
   - **Perfective Maintenance:** This type of maintenance is carried out to enhance the functionalities of the system based on the customer's request.
   - **Adaptive Maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment such as work on a new computer platform or with a new operating system.

Advantages of Classical Waterfall Model

Classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered as the basis for other software development life cycle models. Below are some of the major advantages of this SDLC model:

- This model is very simple and is easy to understand.
- Phases in this model are processed one at a time.
- Each stage in the model is clearly defined.
- This model has very clear and well understood milestones.
- Process, actions and results are very well documented.
- This model works well for smaller projects.

Drawbacks of Classical Waterfall Model

Classical waterfall model suffers from various shortcomings, basically we can’t use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model:

- **No feedback path:** In classical waterfall model evolution of a software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phases. Therefore, it does not incorporate any mechanism for error correction.
- **Difficult to accommodate change requests:** This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but actually customers’ requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.
- **No overlapping of phases:** This model recommends that new phase can start only after the completion of the previous phase. But in real projects, this can’t be maintained. To increase the efficiency and reduce the cost, phases may overlap.

Software Engineering | Iterative Waterfall Model

In a practical software development project, the classical waterfall model is hard to use. So, Iterative waterfall model can be thought of as incorporating the necessary changes to the classical waterfall model to make it usable in practical software development projects. It is almost same as the classical waterfall model except some changes are made to increase the efficiency of the software development.
The iterative waterfall model provides feedback paths from every phase to its preceding phases, which is the main difference from the classical waterfall model.

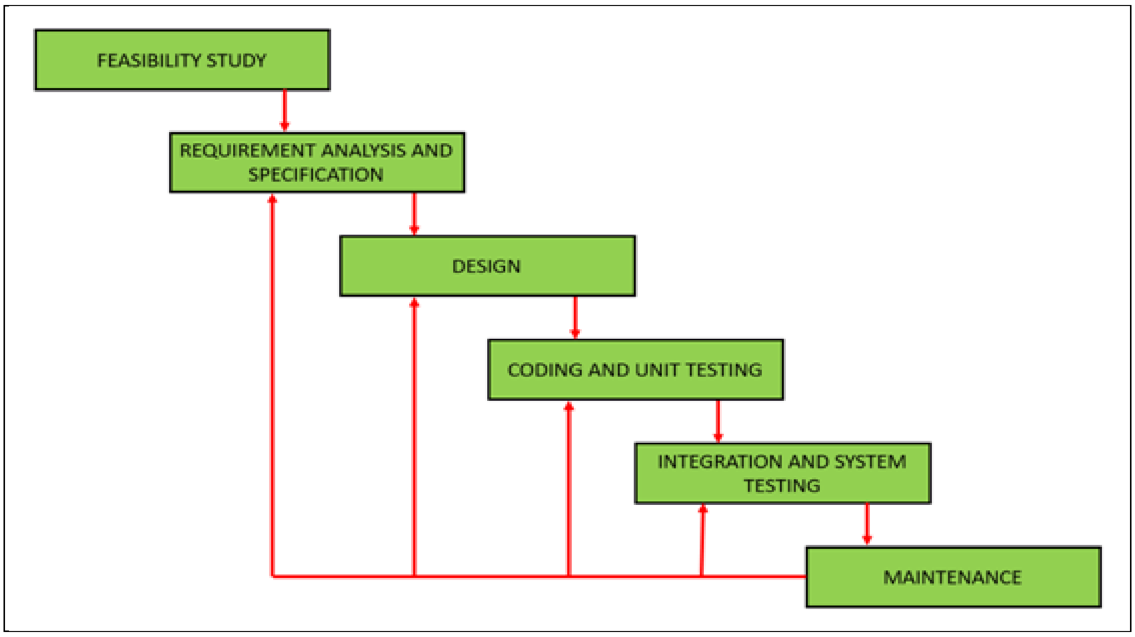

FEASIBILITY STUDY
REQUIREMENT ANALYSIS AND SPECIFICATION
DESIGN
CODING AND UNIT TESTING
INTEGRATION AND SYSTEM TESTING
MAINTENANCE

Fig. 1.2 - Iterative waterfall model.

When errors are detected at some later phase, these feedback paths allow correcting errors committed by programmers during some phase. The feedback paths allow the phase to be reworked in which errors are committed and these changes are reflected in the later phases. But, there is no feedback path to the stage – feasibility study, because once a project has been taken, does not give up the project easily.

It is good to detect errors in the same phase in which they are committed. It reduces the effort and time required to correct the errors.

Advantages of Iterative Waterfall Model

- **Feedback Path:** In the classical waterfall model, there are no feedback paths, so there is no mechanism for error correction. But in iterative waterfall model feedback path from one phase to its preceding phase allows correcting the errors that are committed and these changes are reflected in the later phases.
- **Simple:** Iterative waterfall model is very simple to understand and use. That's why it is one of the most widely used software development models.

Spiral model

Spiral model is one of the most important Software Development Life Cycle models, which provides support for **Risk Handling**. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. **Each loop of the spiral is called a Phase of the software development process.** The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using spiral model.

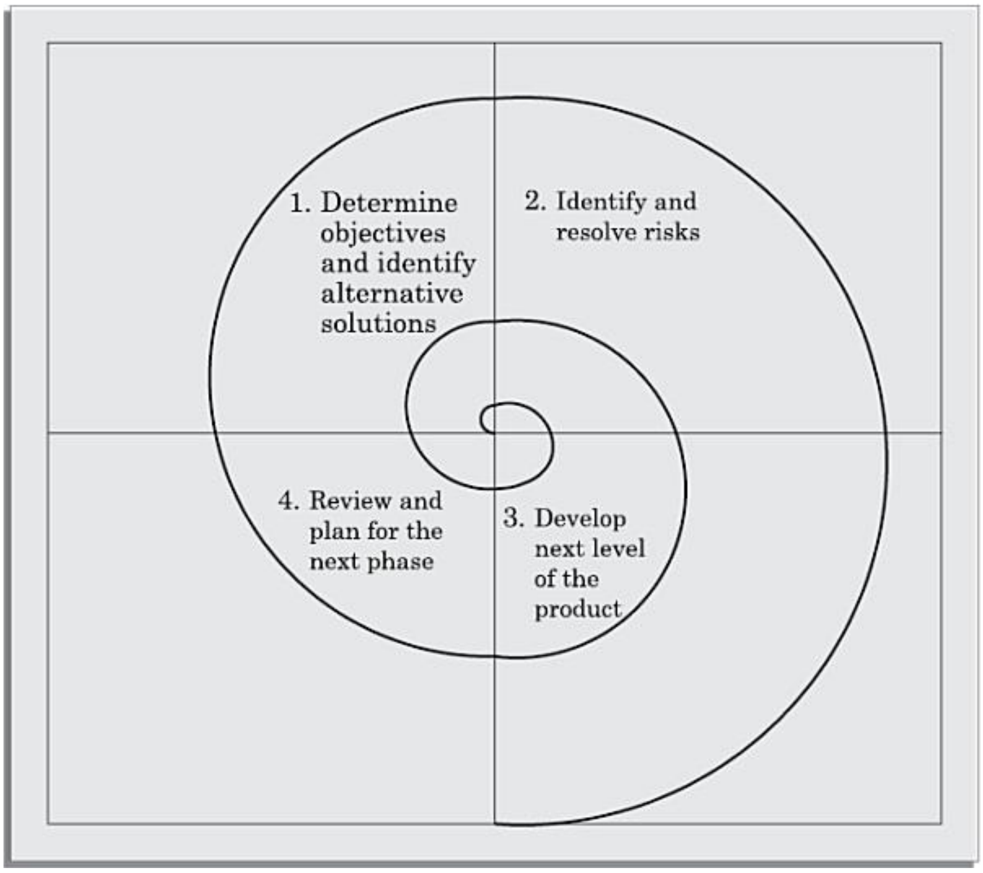

1. Determine objectives and identify alternative solutions
2. Identify and resolve risks
3. Develop next level of the product
4. Review and plan for the next phase

Fig. 1.3 - Spiral model.

1. **Objectives determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated and analysed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
2. **Identify and resolve Risks:** During the second quadrant all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution is identified and the risks are resolved using the best possible strategy. At the end of this quadrant, Prototype is built for the best possible solution.
3. **Develop next version of the Product:** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
4. **Review and plan for the next Phase:** In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

Why Spiral Model is called Meta Model?

The Spiral model is called as a Meta Model because it subsumes all the other SDLC models. For example, a single loop spiral actually represents the Iterative Waterfall Model. The spiral model incorporates the stepwise approach of the Classical Waterfall Model. The spiral model uses the approach of **Prototyping Model** by building a prototype at the start of each phase as a risk handling technique. Also, the spiral model can be considered as supporting the evolutionary model – the iterations along the spiral can be considered as evolutionary levels through which the complete system is built.

Advantages of Spiral Model: Below are some of the advantages of the Spiral Model.

- **Risk Handling:** The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
- **Good for large projects:** It is recommended to use the Spiral Model in large and complex projects.
- **Flexibility in Requirements:** Change requests in the Requirements at later phase can be incorporated accurately by using this model.
- **Customer Satisfaction:** Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

Disadvantages of Spiral Model: Below are some of the main disadvantages of the spiral model.

- **Complex:** The Spiral Model is much more complex than other SDLC models.
- **Expensive:** Spiral Model is not suitable for small projects as it is expensive.
- **Too much dependable on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced expertise, it is going to be a failure to develop a project using this model.
- **Difficulty in time management:** As the number of phases is unknown at the start of the project, so time estimation is very difficult.

Lecture 4

Prototyping Model

The prototype model is also a popular life cycle model. The prototyping model can be considered to be an extension of the waterfall model.
This model suggests building a working prototype of the system, before development of the actual software. A prototype is a toy and crude implementation of a system. It has limited functional capabilities, low reliability, or inefficient performance as compared to the actual software. A prototype can be built very quickly by using several shortcuts. The shortcuts usually involve developing inefficient, inaccurate, or dummy functions.

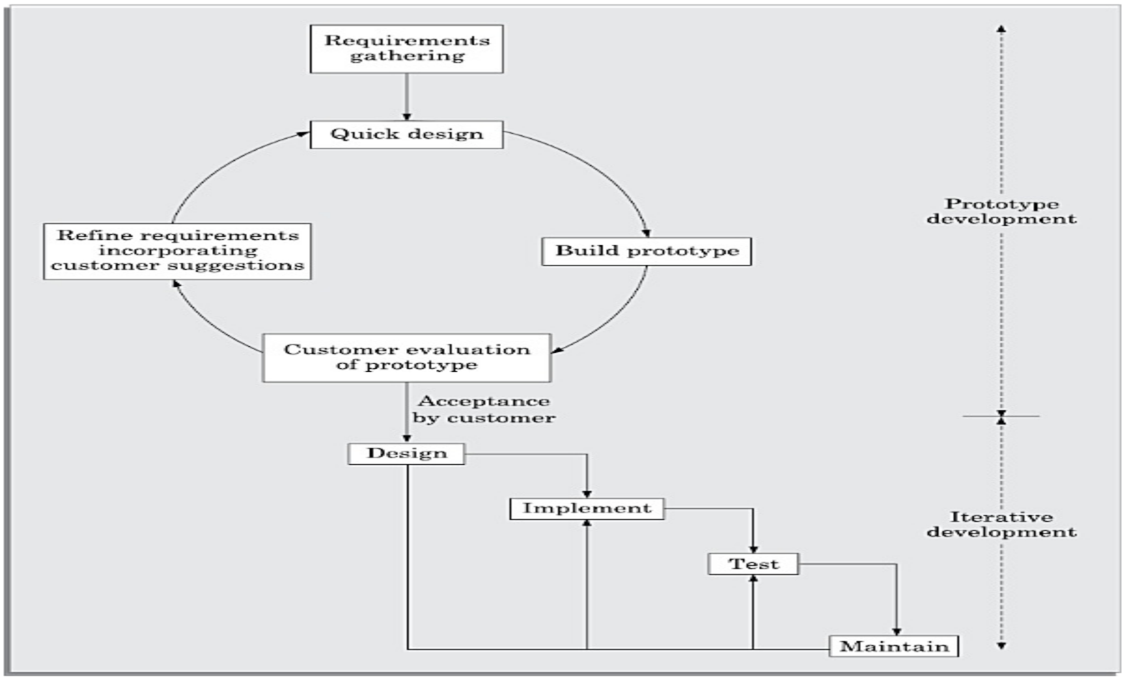


Fig. 1.4 - Prototyping model

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.

Incremental process model

Incremental process model is also known as Successive version model.

First, a simple working system implementing only a few basic features is built and then that is delivered to the customer. Then thereafter many successive iterations/ versions are implemented and delivered to the customer until the desired system is realized.

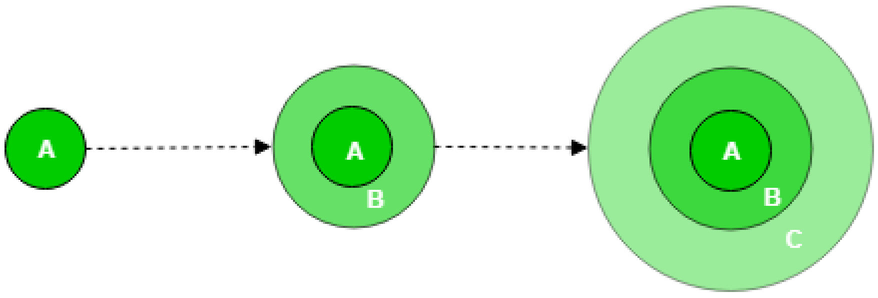


Fig. 1.5 - Incremental process model

A, B, C are modules of Software Product that are incrementally developed and delivered.

Lifecycle activities

Requirements of Software are first broken down into several modules that can be incrementally constructed and delivered. At any time, the plan is made just for the next increment and not for any kind of long term plans. Therefore, it is easier to modify the version as per the need of the customer. Development Team first undertakes to develop core features (these do not need services from other features) of the system.
Once the core features are fully developed, then these are refined to increase levels of capabilities by adding new functions in Successive versions. Each incremental version is usually developed using an iterative waterfall model of development.

As each successive version of the software is constructed and delivered, now the feedback of the Customer is to be taken and these were then incorporated in the next version. Each version of the software has more additional features over the previous ones.

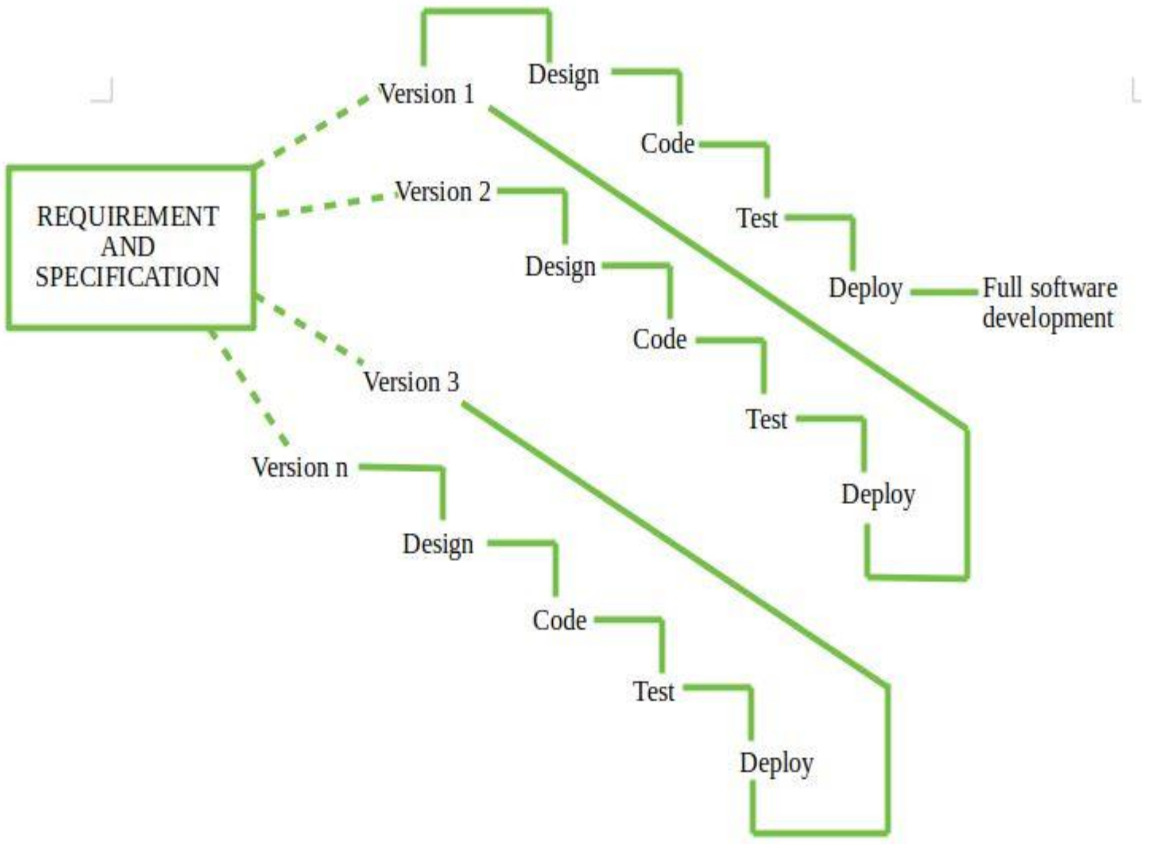


Fig. 1.6– Life Cycle activities

After Requirements gathering and specification, requirements are then spitted into several different versions starting with version-1, in each successive increment, next version is constructed and then deployed at the customer site. After the last version (version n), it is now deployed at the client site.

When to use this –

1. Funding Schedule, Risk, Program Complexity, or need for early realization of benefits.
2. When Requirements are known up-front.
3. When Projects having lengthy developments schedules.
4. Projects with new Technology.

Advantages –

- Error Reduction (core modules are used by the customer from the beginning of the phase and then these are tested thoroughly)
- Uses divide and conquer for breakdown of tasks.
- Lowers initial delivery cost.
- Incremental Resource Deployment.

Disadvantages –

- Requires good planning and design.
- Total cost is not lower.
- Well defined module interfaces are required.

Rapid application development model (RAD)

The Rapid Application Development Model was first proposed by IBM in 1980's. The critical feature of this model is the use of powerful development tools and techniques.

A software project can be implemented using this model if the project can be broken down into small modules wherein each module can be assigned independently to separate teams. These modules can finally be combined to form the final product.

Development of each module involves the various basic steps as in waterfall model i.eanalyzing, designing, coding and then testing, etc. as shown in the figure. Another striking feature of this model is a short time span i.e the time frame for delivery(time-box) is generally 60-90 days.

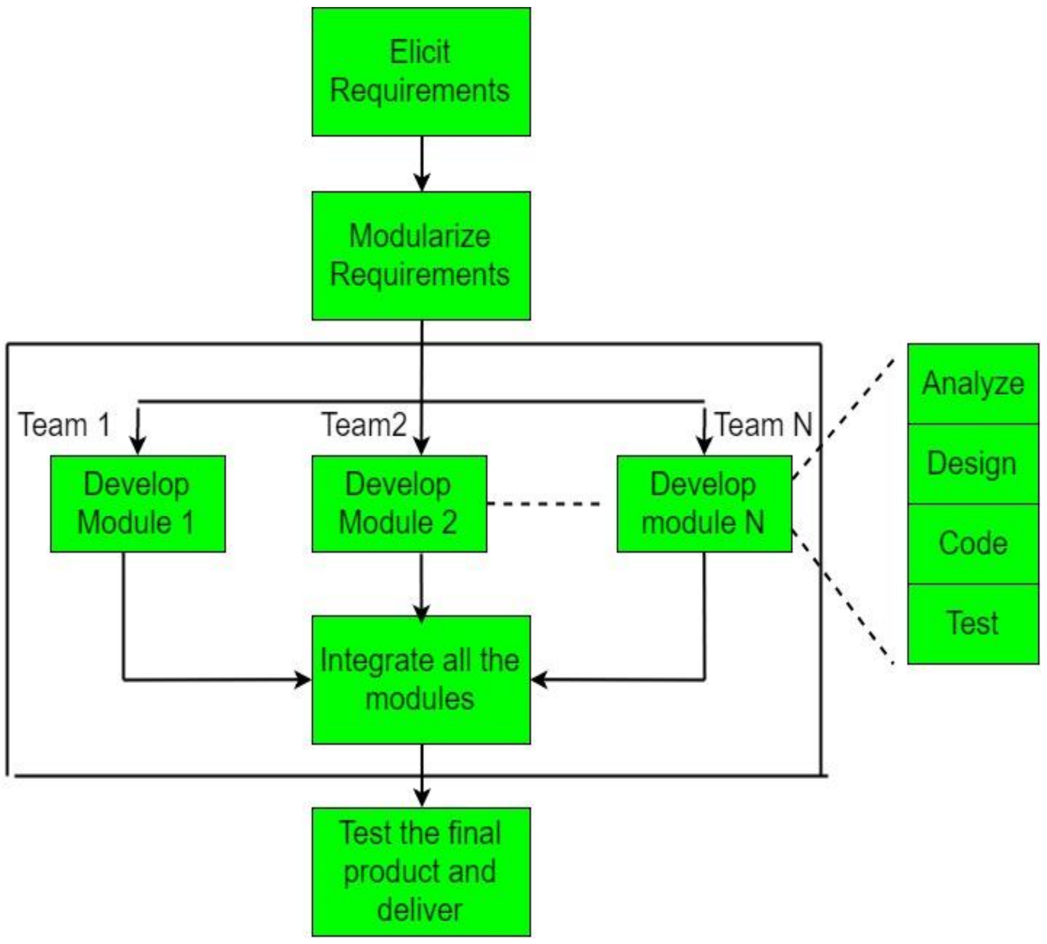


Fig. 1.7 - RAD model

This model consists of 4 basic phases:

1. **Requirements Planning –**
   It involves the use of various techniques used in requirements elicitation like brainstorming, task analysis, form analysis, user scenarios, FAST (Facilitated Application Development Technique), etc. It also consists of the entire structured plan describing the critical data, methods to obtain it and then processing it to form final refined model.

2. **User Description –**
   This phase consists of taking user feedback and building the prototype using developer tools. In other words, it includes re-examination and validation of the data collected in the first phase. The dataset attributes are also identified and elucidated in this phase.

3. **Construction**
   In this phase, refinement of the prototype and delivery takes place. It includes the actual use of powerful automated tools to transform process and data models into the final working product. All the required modifications and enhancements are too done in this phase.

4. **Cutover**
   All the interfaces between the independent modules developed by separate teams have to be tested properly. The use of powerfully automated tools and subparts makes testing easier. This is followed by acceptance testing by the user. The process involves building a rapid prototype, delivering it to the customer and the taking feedback. After validation by the customer, SRS document is developed and the design is finalised.

Advantages –

- Use of reusable components helps to reduce the cycle time of the project.
- Feedback from the customer is available at initial stages.
- Reduced costs as fewer developers are required.
- Use of powerful development tools results in better quality products in comparatively shorter time spans.
- The progress and development of the project can be measured through the various stages.
- It is easier to accommodate changing requirements due to the short iteration time spans.

Disadvantages –

- The use of powerful and efficient tools requires highly skilled professionals.
- The absence of reusable components can lead to failure of the project.
- The team leader must work closely with the developers and customers to close the project in time.
- The systems which cannot be modularized suitably cannot use this model.
- Customer involvement is required throughout the life cycle.
- It is not meant for small scale projects as for such cases, the cost of using automated tools and techniques may exceed the entire budget of the project.

Lecture 5

Project Planning in Software Engineering

Before starting a software project, it is essential to determine the tasks to be performed and properly manage allocation of tasks among individuals involved in the software development. Hence, planning is important as it results in effective software development.

Project planning is an organized and integrated management process, which focuses on activities required for successful completion of the project. It prevents obstacles that arise in the project such as changes in projects or organization's objectives, non-availability of resources, and so on. Project planning also helps in better utilization of resources and optimal usage of the allotted time for a project.

Once a project has been found to be feasible, software project managers
undertake project planning.
Project planning is undertaken and completed before any development activity starts.
Project planning requires utmost care and attention since commitment to
unrealistic time and resource estimates result in schedule slippage. Schedule
delays can cause customer dissatisfaction and adversely affect team morale.
It can even cause project failure. For this reason, project planning is

undertaken by the project managers with utmost care and attention.

However, for effective project planning, in addition to a thorough knowledge
of the various estimation techniques, past experience is crucial.
During project planning, the project manager performs the following
activities. Note that we have given only a very brief description of the
activities. We discuss these in the later section in more detail.

Estimation: The following project attributes are estimated.

• **Cost:** How much is it going to cost to develop the software product?
• **Duration:** How long is it going to take to develop the product?
• **Effort:** How much effort would be necessary to develop the product?
The effectiveness of all later planning activities such as scheduling and
staffing is dependent on the accuracy with which these three estimations
have been made.

Scheduling: After all the necessary project parameters have been
estimated, the schedules for manpower and other resources are developed.

Staffing: Staff organisation and staffing plans are made.

Risk management: This includes risk identification, analysis, and abatement planning.

Miscellaneous plans: This includes making several other plans such as
quality assurance plan, and configuration management plan, etc.
Observe that size estimation is the first activity that a project manager
undertakes during project planning.
Size is the most fundamental parameter based on which all other estimations and project plans are made.
required to complete a project and the duration over which the development

is to be carried out are estimated. Based on the effort estimation, the cost of the project is computed. The estimated cost forms the basis on which price negotiations with the customer is carried out. Other planning activities such as staffing, scheduling etc. are undertaken based on the effort and duration estimates made.

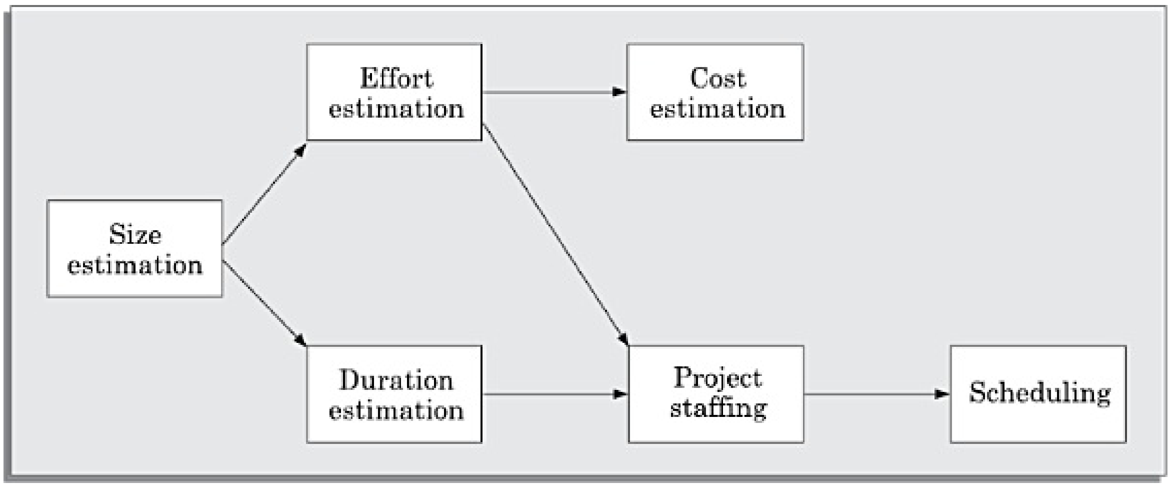


Fig. 1.8– Project Estimation metrics

Feasibility analysis

Feasibility is defined as the practical extent to which a project can be performed successfully. To evaluate feasibility, a feasibility study is performed, which determines whether the solution considered to accomplish the requirements is practical and workable in the software. Information such as resource availability, cost estimation for software development, benefits of the software to the organization after it is developed and cost to be incurred on its maintenance are considered during the feasibility study. The objective of the feasibility study is to establish the reasons for developing the software that is acceptable to users, adaptable to change and conformable to established standards. Various other objectives of feasibility study are listed below.

- To analyse whether the software will meet organizational requirements
- To determine whether the software can be implemented using the current technology and within the specified budget and schedule
- To determine whether the software can be integrated with other existing software.

Types of Feasibility

Various types of feasibility that are commonly considered include technical feasibility, operational feasibility, and economic feasibility.

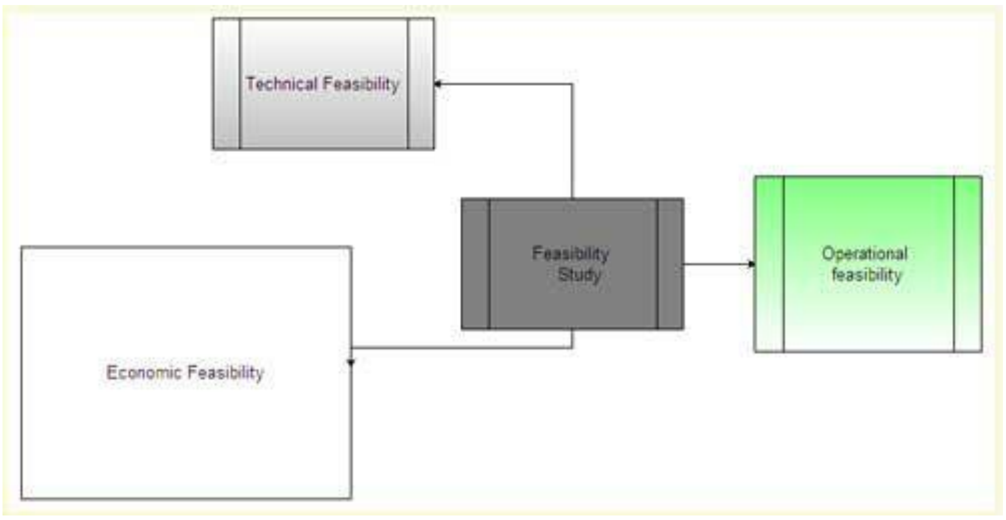


Fig. 1.9 – Types of Feasibility

Technical feasibility

Technical feasibility assesses the current resources (such as hardware and software) and technology, which are required to accomplish user requirements in the software within the allocated time and budget. For this, the software development team ascertains whether the current resources and technology can be upgraded or added in the software to accomplish specified user requirements. Technical feasibility also performs the following tasks.

- Analyzes the technical skills and capabilities of the software development team members
- Determines whether the relevant technology is stable and established
- Ascertains that the technology chosen for software development has a large number of users so that they can be consulted when problems arise or improvements are required.

Operational feasibility

assesses the extent to which the required software performs a series of steps to solve business problems and user requirements. This feasibility is dependent on human resources (software development team) and involves visualizing whether the software will operate after it is developed and be operative once it is installed. Operational feasibility also performs the following tasks.

- Determines whether the problems anticipated in user requirements are of high priority
- Determines whether the solution suggested by the software development team is acceptable
- Analyzes whether users will adapt to a new software
- Determines whether the organization is satisfied by the alternative solutions proposed by the software development team.

Economic feasibility

Economic feasibility determines whether the required software is capable of generating financial gains for an organization. It involves the cost incurred on the software development team, estimated cost of hardware and software, cost of performing feasibility study, and so on. For this, it is essential to consider expenses made on purchases (such as hardware purchase) and activities required to carry out software development. In addition, it is necessary to consider the benefits that can be achieved by developing the software. Software is said to be economically feasible if it focuses on the issues listed below.

- Cost incurred on software development to produce long-term gains for an organization
- Cost required to conduct full software investigation (such as requirements elicitation and requirements analysis)
- Cost of hardware, software, development team, and training.

Lecture 6

COCOMO Model

COCOMO (Constructive Cost Model) is a regression model based on LOC, i.e. **number of Lines of Code**. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality. It was proposed by Barry Boehm in 1970 and is based on the study of 63 projects, which make it one of the best-documented models.
The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:

- **Effort:** Amount of labour
- that will be required to complete a task. It is measured in person-months units.
- **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

Different models of Cocomo have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. All of these models can be applied to a variety of projects, whose characteristics determine the value of constant to be used in subsequent calculations. These characteristics pertaining to different system types are mentioned below.

Boehm's definition of organic, semidetached, and embedded systems:

1. **Organic** – A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.
2. **Semi-detached** – A software project is said to be a Semi-detached type if the vital characteristics such as team-size, experience, knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered of Semi-Detached type.
3. **Embedded** – A software project with requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

All the above system types utilize different values of the constants used in Effort calculations.

Types of Models: COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adopted according to our requirements. These are types of COCOMO model:

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

The first level, **Basic COCOMO** can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations.
Intermediate COCOMO takes these Cost Drivers into account and **Detailed COCOMO** additionally accounts for the influence of individual project phases, i.e. in case of Detailed it

accounts for both these cost drivers and also calculations are performed phase wise henceforth producing a more accurate result. These two models are further discussed below.

Estimation of Effort: Calculations –

4. **Basic Model**

   The above formula is used for the cost estimation of for the basic COCOMO model, and also is used in the subsequent models. The constant values a and b for the Basic Model for the different categories of system:

SOFTWARE PROJECTS	A	B
Organic	2.4	1.05
Semi Detached	3.0	1.12
Embedded	3.6	1.20

   The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, expertise is taken into account, henceforth the estimate is rough.

5. **Intermediate Model –**
   The basic Cocomo model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software system. However, in reality, no system's effort and schedule can be solely calculated on the basis of Lines of Code. For that, various other factors such as reliability, experience, Capability. These factors are known as Cost Drivers and the Intermediate Model utilizes 15 such drivers for cost estimation.

   Classification of Cost Drivers and their attributes:

   (i) Product attributes –
   - Required software reliability extent
   - Size of the application database
   - The complexity of the product

   (ii) Hardware attributes –
   - Run-time performance constraints
   - Memory constraints
   - The volatility of the virtual machine environment
   - Required turnabout time

   (iii) Personnel attributes –
   - Analyst capability
   - Software engineering capability
   - Applications experience
   - Virtual machine experience
   - Programming language experience

(iv) Project attributes –

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

Detailed Model –

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.
The Six phases of detailed COCOMO are:

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

The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software lifecycle.

Problem: Assume that the size of an organic type software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software engineers be Rs. 15,000/- per month. Determine the effort required to develop the software product and the nominal development time.

Solution: As per the basic COCOMO estimation formula for organic software:

Effort = 2.4 * $(32)^{1.05}$ = 91 PM

Nominal development time = 2.5 * $(91)^{0.38}$ = 14 months

Cost required to develop the product = 14 * 15,000 = Rs. 210,000/-

Example: Suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three modes i.e. organic, semidetached and embedded.

Solution The basic COCOMO equations take the form:

E = ab (KLOC)bb

D = cb (E)db

Estimated size of the project = 400 KLOC

1. Organic Mode

E = 2.4 (400)1.05 = 1295.31 PM

D = 2.5 (1295.31)0.38 = 38.07 M

2. Semi-detached Mode

E = 3.0 (400)1.12 = 2462.79 PM

D = 2.5 (2462.79)0.35 = 38.45 M

3. Embedded Mode

E = 3.6 (400)1.20 = 4772.81 PM

D = 2.5 (4772.81)0.32 = 37.59 M

MODULE: 1

SOFTWARE ENGINEERING

References:

1. https://www.tutorialspoint.com/software_engineering/pdf/software_engineering_interview_questions.pdf
2. https://www.techtud.com/sites/default/files/public/share/SOFTWARE%20ENGINEERING%20OBJECTIVE%20QUESTIONS%20WITH%20ANSWERS.pdf
3. https://www.cmpe.boun.edu.tr/sites/default/files/introduction.pdf
4. https://nptel.ac.in/courses/106105087/pdf/m01L01.pdf
5. https://www.tutorialspoint.com/sdlc/sdlc_tutorial.pdf
6. https://www.softwaretestinggenius.com/download/Lifecycle.pdf
7. https://nptel.ac.in/courses/Webcourse-contents/IIT%20Kharagpur/Soft%20Engg/pdf/m11L28.pdf
8. https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/WIKIPEDI/W120626C.pdf
9. https://nptel.ac.in/courses/Webcourse-contents/IIT%20Kharagpur/Soft%20Engg/pdf/m11L27.pdf
10. http://www.suranacollege.edu.in/surana-pg/pdf/mca/Estimation-for-Software-Projects.pdf
11. https://www.techtud.com/sites/default/files/public/share/SOFTWARE%20ENGINEERING%20OBJECTIVE%20QUESTIONS%20WITH%20ANSWERS.pdf

MODULE: 1
SOFTWARE ENGINEERING

Multiple Choice Questions

1. Which model can be selected if user is involved in all the phases of SDLC?
a) Waterfall Model
b) Prototyping Model
c) RAD Model
d) both Prototyping Model & RAD Model

2. What is the major drawback of using RAD Model?
a) Highly specialized & skilled developers/designers are required
b) Increases reusability of components
c) Encourages customer/client feedback
d) Increases reusability of components, highly specialized & skilled developers/designers are required

3. Which of the following statements regarding Build & Fix Model is wrong?
a) No room for structured design
b) Code soon becomes unfixable & unchangeable
c) Maintenance is practically not possible
d) It scales up well to large projects

4. Which is not one of the types of prototype of Prototyping Model?
a) Horizontal Prototype
b) Vertical Prototype
c) Diagonal Prototype
d) Domain Prototype

5. Build & Fix Model is suitable for programming exercises of ___________ LOC (Line of Code).
a) 100-200
b) 200-400
c) 400-1000
d) above 1000

6. What are the characteristics of software?
a. Software is developed or engineered; it is not manufactured in the classical sense.

b. Software doesn’t “wear out ”.
c. Software can be custom built or custom build.
d. All mentioned above

7. Software project management comprises of a number of activities, which contains

a. Project planning
b. Scope management
c. Project estimation
d. All mentioned above

8. COCOMO stands for ______ .
a. Consumed Cost Model
b. Constructive Cost Model
c. Common Control Model
d. Composition cost Model

9. Which of the following is not defined in a good Software Requirement Specification (SRS) document?
a. Functional Requirement.
b. Non-functional Requirement.
c. Goals of implementation.
d. Algorithm for software implementation.

10. What is the simplest model of software development paradigm?
a. Spiral model
b. Big Bang model
c. V-model
d. Waterfall model

MODULE: 1

SOFTWARE ENGINEERING

Short Answer Type Question

1. When you know programming, what is the need to learn software engineering concepts?
2. What is software process or Software Development Life Cycle (SDLC)?
3. What are various phases of SDLC?
4. What is software project management?
5. What does software project manager do?
6. What is project estimation?
7. How can we derive the size of software product?
8. What is feasibility study?
9. Why prototype model is called Meta model?
10. What are the limitations of Waterfall model?
11. Explain advantages of spiral model.
12. Explain COCOMO Model.

MODULE: 1

SOFTWARE ENGINEERING

Assignment:

1. Explain Spiral Model in detail. Also differentiate between prototype Model and Water Fall Model.
2. Discuss the merits & demerits of various Model of software Engineering Models?
3. Explain all the phases of Software Development Life cycle?
4. Explain the prototype Model? Under what circumstances is it beneficial to construct a prototype model?
5. What are the advantages and disadvantages of Waterfall Model of software life cycle?
6. How does the risk factor affect the spiral model of software development? Why prototype model is called meta model?
7. Compare the basic COCOMO model with the detailed COCOMO model.
8. Differentiate between iterative Enhancement Model and Evolutionary Development model.
9. Short notes – Iterative waterfall model, software project estimation,

MODULE: 1

SOFTWARE ENGINEERING

Web/Video links:

1. https://www.youtube.com/watch?v=tZreaH_FyMs&list=PLV8vIYTIdSnat3WCO9jfehtZyjnxb74wm
2. https://www.youtube.com/watch?v=TtC_OFjknWI&index=4&list=PLV8vIYTIdSnat3WCO9jfehtZyjnxb74wm
3. https://www.youtube.com/watch?v=G-6qDY8UltU&t=30s
4. https://www.youtube.com/watch?v=DRDD7UWX2y4
5. https://www.youtube.com/watch?v=6ySF9wNDAZo
6. https://www.youtube.com/watch?v=9pfInykZdwE
7. https://www.youtube.com/watch?v=Fu3PLcgZHE4
8. https://www.youtube.com/watch?v=wEiatXC52Qw
9. https://www.youtube.com/watch?v=Xe5qQBI8MeM

MODULE: II

SYSTEM ANALYSIS & DESIGN

Lecture 7

System Analysis

To understand System Analysis and Design, one has to first understand what exactly systems are. In this session, we explore the meaning of system in accordance with analysts and designers. This session gives the reader basic concepts and terminology associated with the Systems.

It also gives the overview of various types of systems. In the broadest sense, a system is simply a set of components that interact to accomplish some purpose. They are all around us. For example, human body is a biological system. We experience physical sensations by means of a complex nervous system, a set of parts, including brain, spinal cord, nerves, and special sensitive cells under our skin, that work together to make us feel hot, cold, itchy, and so on.

Language is another example of a system. Each language has got its set of alphabets, vocabulary and grammar rules. Combination of all these make possible for one person to convey thoughts to other persons.

An organization may also be viewed as a system where all the employees interact with each other and also with the employer to make the organization a functional unit. The organization also interacts with their customers to make a complete business system.

In today's world most of the people study. To make this possible, there are education systems. Each education system contains educational institutes like preparatory schools, middle and high schools and colleges. It also contains governing bodies, people (teachers and students) and some commercial bodies, which fulfil the other needs like stationery, transportation, furniture, etc.

In our day to day life, we see many business systems. These businesses have varied objectives, which range from producing a notebook to producing aircraft. These systems have their information needs. It can be for maintaining the records for employee for their wages calculations, keeping track of their leave status, maintaining company's expenses, inquiries from customers in case the business provide some service, or for keeping track for some particular function. So maintaining data is an important and essential activity in any business. The overall data maintained constitutes what is known as Information system.

Information system is the means by which data flow from one person or department to another and can encompass everything from interoffice mail and telephone links to a computer system that generates periodic reports for various users. Information systems serve all the systems of a business, linking the different components in such a way that they effectively work towards the same purpose.

System Concepts

The term "System" is derived from the Greek word systema. It means an organized relationship among functioning units or components. We can define a System as a combination of resources or functional units working together to accomplish a given task.

The term "working together" in system definition is very important as all the components are interrelated and interdependent and cannot exist independently. As the definition says, these components interact with each other to accomplish a given task, which is actually the objective of the system.

Principle of structure Analysis

A big system may be seen as a set of interacting smaller systems known as subsystems or functional units each of which have its defined tasks. All these work in coordination to achieve the overall objective of the system.

As discussed above, a system is a set of components working together to achieve some goal. The basic elements of the system may be listed as:

- Resources
- Procedures
- Data/Information
- Processes

Resources

Every system requires certain resources for the system to exist. Resources can be hardware, software or liveware. Hardware resources may include the computer, its peripherals, stationery etc. Software resources would include the programs running on these computers and the liveware would include the human beings required to operate the system and make it functional.

Thus these resources make an important component of any system. For instance, a Banking system cannot function without the required stationery like cheque books, pass books etc. such systems also need computers to maintain their data and trained staff to operate these computers and cater to the customer requirements.

Procedures

Every system functions under a set of rules that govern the system to accomplish the defined goal of the system. This set of rules defines the procedures for the system to Chapter 1-Introduction to Systems operate. For instance, the Banking systems have their predefined rules for providing interest at different rates for different types of accounts.

Data/Information

Every system has some predefined goal. For achieving the goal the system requires certain inputs, which are converted into the required output. The main objective of the System is to produce some useful output. Output is the outcome of processing. Output can be of any nature e.g. goods, services or information.

However, the Output must conform to the customer's expectations. Inputs are the elements that enter the system and produce Output. Input can be of various kinds, like material, information, etc.

Intermediate Data

Various processes process system's Inputs. Before it is transformed into Output, it goes through many intermediary transformations. Therefore, it is very important to identify the Intermediate Data. For example, in a college when students register for a new semester, the initial form submitted by student goes through many departments. Each department adds their validity checks on it.

Finally the form gets transformed and the student gets a slip that states whether the student has been registered for the requested subjects or not. It helps in building the System in a better way. Intermediate forms of data occur when there is a lot of processing on the input data. So, intermediate data should be handled as carefully as other data since the output depends upon it.

Processes

The systems have some processes that make use of the resources to achieve the set goal under the defined procedures. These processes are the operational element of the system.

For instance in a Banking System there are several processes that are carried out. Consider for example the processing of a cheque as a process. A cheque passes through several stages before it actually gets processed and converted. These are some of the processes of the Banking system. All these components together make a complete functional system.

Systems also exhibit certain features and characteristics, some of which are:

- Objective
- Standards
- Environment
- Feedback
- Boundaries and interfaces

Objective

Every system has a predefined goal or objective towards which it works. A system cannot exist without a defined objective. For example an organization would have an objective of earning maximum possible revenues, for which each department and each individual has to work in coordination.

Standards

It is the acceptable level of performance for any system. Systems should be designed to meet standards. Standards can be business specific or organization specific.

For example take a sorting problem. There are various sorting algorithms. But each has its own complexity. So such algorithm should be used that gives most optimum efficiency. So there should be a standard or rule to use a particular algorithm. It should be seen whether that algorithm is implemented in the system.

Environment

Every system whether it is natural or manmade co-exists with an environment. It is very important for a system to adapt itself to its environment. Also, for a system to exist it should change according to the changing environment. For example, we humans live in a particular environment. As we move to other places, there are changes in the surroundings but our body gradually adapts to the new environment. If it were not the case, then it would have been very difficult for human to survive for so many thousand years.

Another example can be Y2K problem for computer systems. Those systems, which are not Y2K compliant, will not be able to work properly after year 2000. For computer systems to survive it is important these systems are made Y2K compliant or Y2K ready.

Feed Back

Feedback is an important element of systems. The output of a system needs to be observed and feedback from the output taken so as to improve the system and make it achieve the laid standards. In fig 2.1, it is shown that a system takes input. It then transforms it into output. Also some feedback can come from customer (regarding quality) or it can be some intermediate data (the output of one process and input for the other) that is required to produce final output

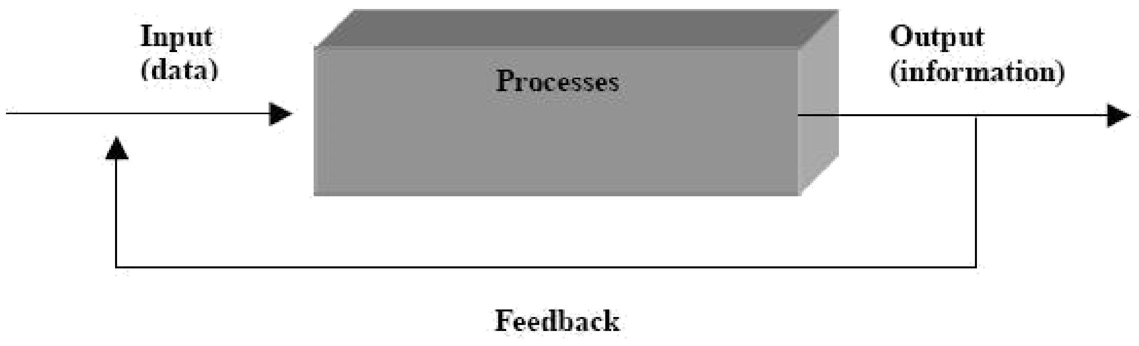


Fig 2.1
A system converts input data into output information

Requirement Analysis

IEEE defines requirements analysis as (1) the process of studying user needs to arrive at a definition of a system, hardware or software requirements. (2) The process of studying and refining system, hardware or software requirements.' Requirements analysis helps to understand, interpret, classify, and organize the software requirements in order to assess the feasibility, completeness, and consistency of the requirements. Various other tasks performed using requirements analysis are listed below.

1. To detect and resolve conflicts that arise due to unclear and unspecified requirements
2. To determine operational characteristics of the software and how they interact with the environment
3. To understand the problem for which the software is to be developed

4. To develop an analysis model to analyze the requirements in the software.

Software engineers perform analysis modeling and create an analysis model to provide information of 'what' software should do instead of 'how' to fulfil the requirements in software. This model emphasizes information such as the functions that software should perform, behavior it should exhibit, and constraints that are applied on the software. This model also determines the relationship of one component with other components. The clear and complete requirements specified in the analysis model help the software development team to develop the software according to those requirements. An analysis model is created to help the development team to assess the quality of the software when it is developed. An analysis model helps to define a set of requirements that can be validated when the software is developed.

Let us consider an example of constructing a study room, where the user knows the dimensions of the room, the location of doors and windows, and the available wall space. Before constructing the study room, he provides information about flooring, wallpaper, and so on to the constructor. This information helps the constructor to analyze the requirements and prepare an analysis model that describes the requirements. This model also describes what needs to be done to accomplish those requirements. Similarly, an analysis model created for the software facilitates the software development team to understand what is required in the software and then they develop it.

In Figure the analysis model connects the system description and design model. System description provides information about the entire functionality of the system, which is achieved by implementing the software, hardware and data. In addition, the analysis model specifies the software design in the form of a design model, which provides information about the software's architecture, user interface, and component level structure.

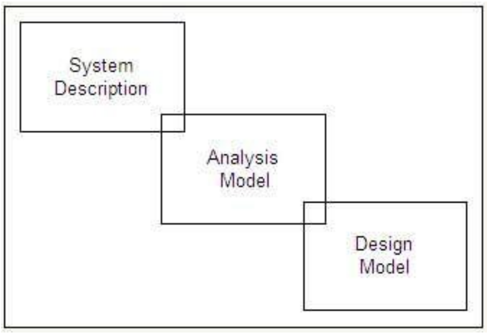


Fig 2.2

The guidelines followed while creating an analysis model are listed below.

1. The model should concentrate on requirements in the problem domain that are to be accomplished. However, it should not describe the procedure to accomplish the requirements in the system.

2. Every element of the analysis model should help in understanding the software requirements. This model should also describe the information domain, function, and behavior of the system.

3. The analysis model should be useful to all stakeholders because every stakeholder uses this model in his own manner. For example, business stakeholders use this model to validate requirements whereas software designers view this model as a basis for design.

4. The analysis model should be as simple as possible. For this, additional diagrams that depict no new or unnecessary information should be avoided.

DFD

A Data Flow Diagram (DFD) is used to describe the logical operation of a system, i.e. what a system does. DFD shows the flow of data through a system and the work or processing performed by that system.

DFDs only represent the flow of data through the system and do not describe the physical functioning of the system. DFDs are basically of 2 types: Physical and logical ones.

Physical DFDs are used in the analysis phase to study the functioning of the current system. Logical DFDs are used in the design phase for depicting the flow of data in proposed system.

Elements of Data Flow Diagrams

Data Flow Diagrams are composed of the four basic symbols shown below.

The External Entity symbol represents sources of data to the system or destinations of data from the system.

The Data Flow symbol represents movement of data.

The Data Store symbol represents data that is not moving (delayed data at rest).

The Process symbol represents an activity that transforms or manipulates the data (combines, reorders, converts, etc.).

Any system can be represented at any level of detail by these four symbols.

Processes

Processes are work or actions performed on incoming data flows to produce outgoing data flows. These show data transformation or change. Data coming into a process must be "worked on" or transformed in some way. Thus, all processes must have inputs and outputs. In some (rare) cases, data inputs or outputs will only be shown at more detailed levels of the diagrams. Each process in always "running" and ready to accept data.

Major functions of processes are computations and making decisions. Each process may have dramatically different timing: yearly, weekly, daily.

Naming Processes

Processes are named with one carefully chosen verb and an object of the verb. There is no subject. Name is not to include the word "process". Each process should represent one function or action. If there is an "and" in the name, you likely have more than one function (and process). For example, get invoice ,update customer and create Order Processes are numbered within the diagram as convenient. Levels of detail are shown by decimal notation. For example, top level process would be Process 14, next level of detail Processes 14.1-14.4, and next level with Processes 14.3.1-14.3.6. Processes should generally move from top to bottom and left to right.

External Entities

External entities determine the system boundary. They are external to the system being studied. They are often beyond the area of influence of the developer. These can represent another system or subsystem. These go on margins/edges of data flow diagram. External entities are named with appropriate name.

Data Flow

Data flow represents the input (or output) of data to (or from) a process ("data in motion"). Data flows only data, not control. Represent the minimum essential data the process needs. Using only the minimum essential data reduces the dependence between processes. Data flows must begin and/or end at a process. Data flows are always named. Name is not to include the word "data" Should be given unique names. Names should be some identifying noun. For example, order, payment, complaint.

Data Stores

Data Stores are repository for data that are temporarily or permanently recorded within the system. It is an "inventory" of data. These are common link between data and process models. Only processes may connect with data stores.

There can be two or more systems that share a data store. This can occur in the case of one system updating the data store, while the other system only accesses the data.

Data stores are named with an appropriate name, not to include the word "file", Names should consist of plural nouns describing the collection of data. Like customers, orders, and products. These may be duplicated. These are detailed in the data dictionary or with data description diagrams.

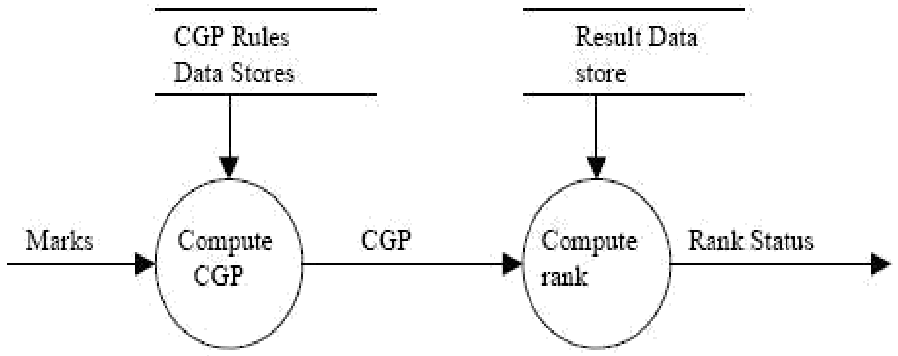


Fig.2.3

DFD showing rank calculation process of a university

Different Levels of DFDs

The DFD may be used to represent a system or software at any level of abstraction. In fact, DFDs may be partitioned into levels that represent increasing information flow and functional details.

A 0 level DFD, also called a context model, represents the entire software elements as a single bubble with input and output data indicated by incoming and outgoing arrows, respectively. Additional processes and information flow paths are represented as the 0 level DFD is partitioned to reveal more information. For example, a 1 level DFD might contain five or six bubbles with interconnecting arrows.

Making DFDs (Data Flow Diagrams)

Data Flow Diagramming is a means of representing a system at any level of detail with a graphic network of symbols showing data flows, data stores, data processes, and data sources/destinations.

The goal of data flow diagramming is to have a commonly understood model of a system. The diagrams are the basis of structured systems analysis. Data flow diagrams are supported by other techniques of structured systems analysis such as data structure diagrams, data dictionaries, and procedure-representing techniques such as decision tables, decision trees, and structured English.

The purpose of data flow diagrams is to provide a semantic bridge between users and systems developers. The diagrams are graphical, eliminating thousands of words. These are logical representations, modeling what a system does, rather than physical models showing how it does it. DFDs are hierarchical, showing systems at any level of detail. Finally, it should be jargon less, allowing user understanding and reviewing.

Also, data flow diagrams have the objective of avoiding the cost of:

- User/developer misunderstanding of a system, resulting in a need to redo systems or in not using the system.
- Having to start documentation from scratch when the physical system changes since the logical system, WHAT gets done, often remains the same when technology changes.
- Systems inefficiencies because a system gets "computerized" before it gets "systematized".
- Being unable to evaluate system project boundaries or degree of automation, resulting in a project of inappropriate scope.

Notation

Data flow analysis was developed and promoted simultaneously by two organizations. Yourdon, Inc., a consulting firm, has vigorously promoted the method publicly. Mc Donnell-Douglas, through the work and writings of Gane and Sarson, has also influenced the popularity of data flow analysis.

Data flow diagram can be completed using only four simple notations. The use of specific icons associated with each element depends on whether the Yourdon or Gane and Sarson approach is used.

1) People, procedures, or devices that use or produce(transform) data. The physical component is not identified Yourdon Gane and Sarson

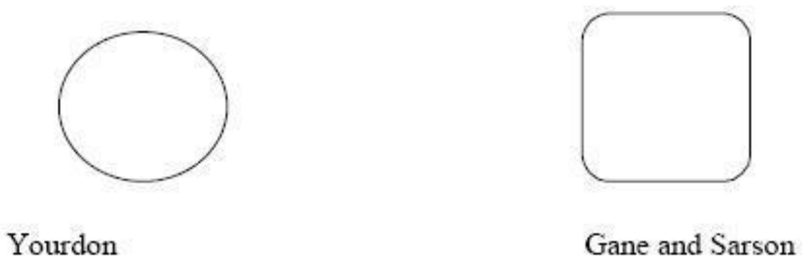


Fig.2.4
Process or transform

2) Data flow: Data move in a specific direction from origin to a destination in the form of a document, letter, telephone call, or virtually any other medium. The data flow is a "packet" of data.

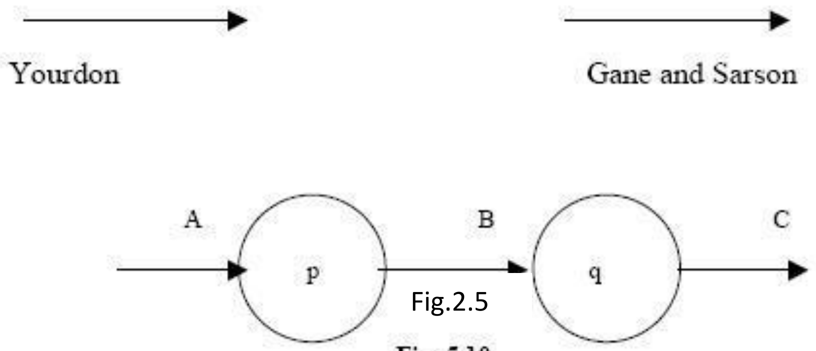


Fig.2.5
Fig. 5.10
Data flows

3) Source or destination of data: external sources or destination of data, which may be people, programs, organizations, or other entities, interact with the system but are outside its boundary. The term source and sink are interchangeable with origin and destination.

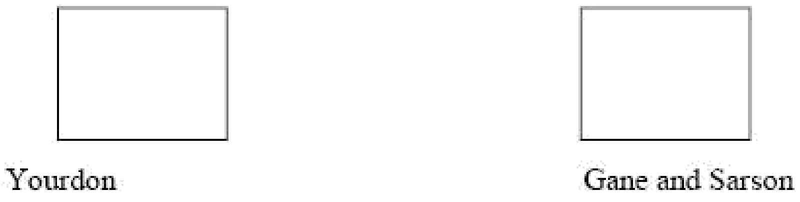


Fig 2.6
Source or destination of data (external)

4) Here data are stored or referenced by a process in the system. The data store may represent computerized or non-computerized devices.

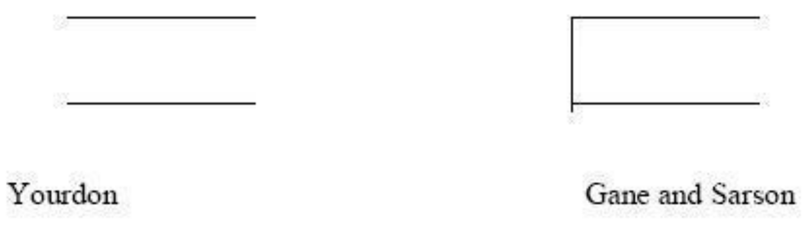


Fig 2.7
Data stores

Each component in a data flow diagram is labeled with a descriptive name. Process names are further identified with a number that will be used for identification purposes. The number assigned to a specific process does not represent the sequence of processes.

Multiple input data streams and multiple data output streams are possible.

If two adjacently placed input are both required then a star sign * is placed between these.

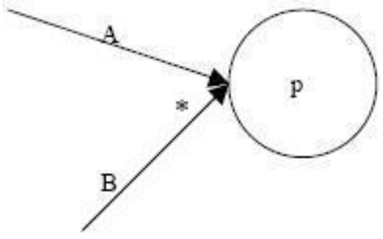


Fig.2.8
When two inputs are required for a transform: star sign

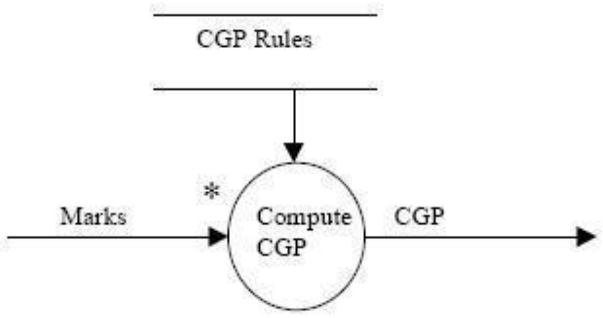


Fig 5.14
DFD showing when two inputs are both required.

If either of two adjacent placed inputs then a ring sum is placed between these.

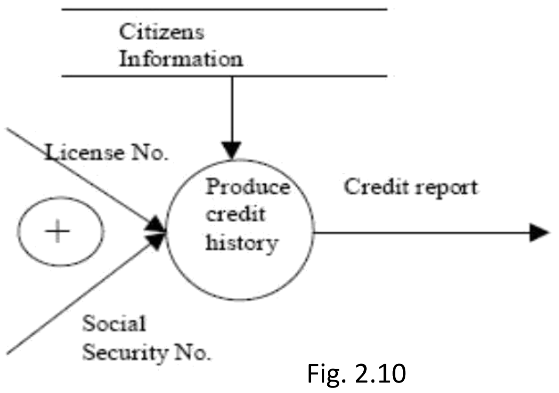


Fig. 2.10

DFD showing when either of two inputs is required.

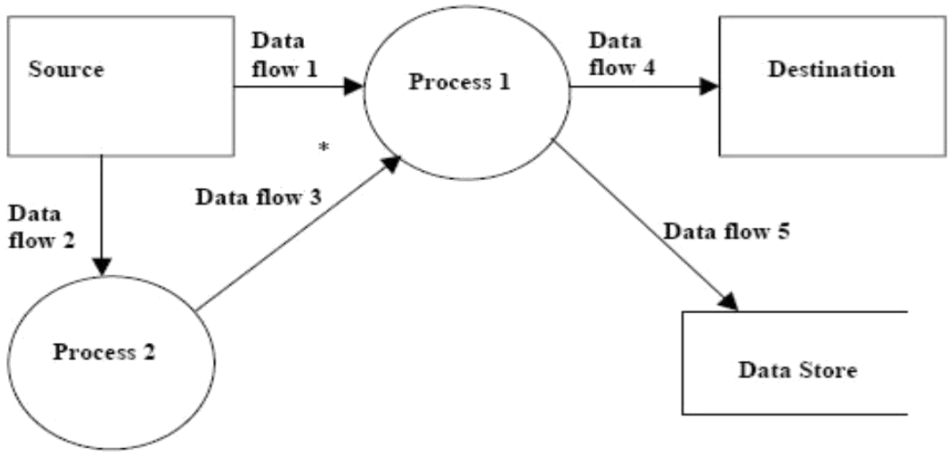


Fig. 5.17
Data Flow diagram using Yourdon notation

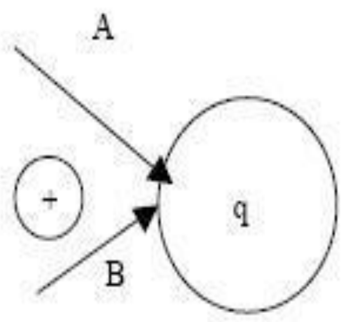


Fig. 5.15
When either of two inputs is required: Ring Sum

The procedure for producing a data flow diagram is to:

Identify and list external entities providing inputs/receiving outputs from system; -identify and list inputs from/outputs to external entities;

Draw a context DFD

Defines the scope and boundary for the system and project.

1. Think of the system as a container (black box)
2. Ignore the inner workings of the container
3. Ask end-users for the events the system must respond to
4. For each event, ask end-users what responses must be produced by the system
5. Identify any external data stores
6. Draw the context diagram

i. Use only one process

ii. Only show those data flows that represent the main objective or most common

Inputs/outputs

- identify the business functions included within the system boundary;
- identify the data connections between business functions;
- confirm through personal contact sent data is received and vice-versa;
- trace and record what happens to each of the data flows entering the system (data movement, data storage, data transformation/processing)
- Draw an overview DFD

>>Shows the major subsystems and how they interact with one another >>Exploding processes should add detail while retaining the essence of the details from the more general diagram

>>Consolidate all data stores into a composite data store

- Draw middle-level DFDs >>Explode the composite processes
- Draw primitive-level DFDs

>>Detail the primitive processes

>>Must show all appropriate primitive data stores and data flows

- verify all data flows have a source and destination;
- verify data coming out of a data store goes in;
- review with "informed";
- Repeat above steps as needed.

Rules for Drawing DFDs

- A process must have at least one input and one output data flow
- A process begins to perform its tasks as soon as it receives the necessary input data flows
- A primitive process performs a single well-defined function
- Never label a process with an IF-THEN statement
- Never show time dependency directly on a DFD
- Be sure that data stores, data flows, data processes have descriptive titles. Processes should use imperative verbs to project action.
- All processes receive and generate at least one data flow.
- Begin/end data flows with a bubble.

Rules for Data Flows

1. A data store must always be connected to a process
2. Data flows must be named
3. Data flows are named using nouns
4. Data that travel together should be one data flow
5. Data should be sent only to the processes that need the data

Lecture 8

DFDs.

Identify the key processing steps in a system. A processing step is an activity that transforms one piece of data into another form.

Process bubbles should be arranged from top left to bottom right of page.

Number each process (1.0, 2.0, etc). Also name the process with a verb that describes the information processing activity.

Name each data flow with a noun that describes the information going into and out of a process.

What goes in should be different from what comes out.

Data stores, sources and destinations are also named with nouns.

Realize that the highest level DFD is the context diagram. It summarizes the entire system as one bubble and shows the inputs and outputs to a system

.Think of data flow not control flow. Data flows are pathways for data. Think about what data is needed to perform a process or update a data store. A data flow diagram is not a flowchart and should not have loops or transfer of control. Think about the data flows, data processes, and data storage that are needed to move a data structure through a system.

Entity Relationship Diagram

The ER data modeling techniques is based on the perception of a real world that consists of a set of basic objects called entities, and of relationships among these objects. In ER modeling, data is described as entities, relationships, and attributes. In the following section, entities and attributes are discussed. Later, entity types, their key attributes, relationship types, their structural constraints, and weak entity types are discussed. In the last, we will apply ER modeling to our case study problem "Library management system".

Entities and Attributes

One of the basic components of ER model is entity. An entity is any distinguishable object about which information is stored. These objects can be person, place, thing, event or a concept. Entities contain descriptive information. Each entity is distinct.

An entity may be physical or abstract. A person, a book, car, house, employee etc. are all physical entities whereas a company, job, or a university course, are abstract entities.

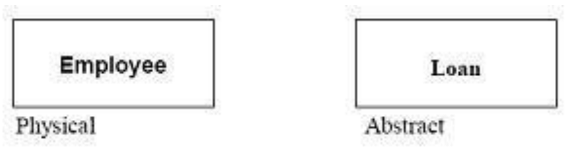


Fig. 2.11 Physical and Abstract Entity

Another classification of entities can be independent or dependent (strong or weak) entity. Entities are classified as independent or dependent (in some methodologies, the terms used are strong and weak, respectively). An independent entity is one, which does not rely on another entity for identification. A dependent entity is one that relies on another entity for identification. An independent entity exists on its own whereas dependent entity exists on the existence of some other entity. For example take an organization scenario. Here department is independent entity. Department manager is a dependent entity. It exists for existing depts. There won't be any department manager for which there is no dept.

Some entity types may not have any key attributes of their own. These are called weak entity types. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with some of their attribute values. For example, take the license entity. It can't exist unless it is related to a person entity.

Attributes

After you identify an entity, then you describe it in real terms, or through its attributes. Attributes are basically properties of entity. We can use attributes for identifying and expressing entities. For example, Dept entity can have DeptName, DeptId, and DeptManager as its attributes. A car entity can have modelno, brandname, and color as its attributes.

A particular instance of an attribute is a value. For example, "Bhaskar" is one value of the attribute Name. Employee number 8005 uniquely identifies an employee in a company.

The value of one or more attributes can uniquely identify an entity.

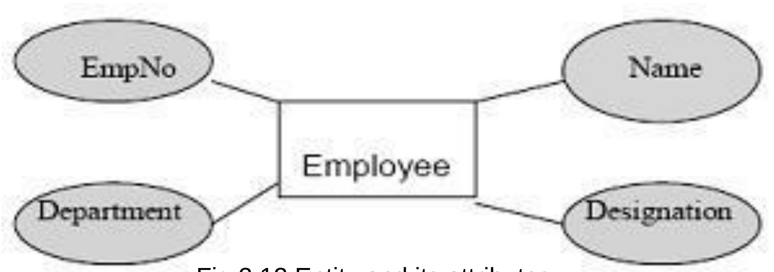


Fig 2.12 Entity and its attributes

In the above figure, employee is the entity. EmpNo, Name, Designation and Department are its attributes. An entity set may have several attributes. Formally each entity can be described by set of <attribute, data value> pairs.

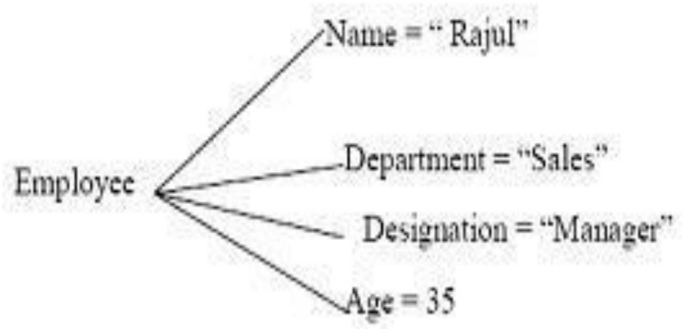


Fig. 2.13 Employee entity and its attribute values

Types of Attributes

Attributes can be of various types. In this section, we'll look at different types of attributes. Attributes can be categorized as:

Key or non-key attributes

Attributes can be classified as identifiers or descriptors. Identifiers, more commonly called keys or key attributes uniquely identify an instance of an entity. If such an attribute doesn't exist naturally, a new attribute is defined for that purpose, for example an ID number or code. A descriptor describes a non-unique characteristic of an entity instance.

An entity usually has an attribute whose values are distinct for each individual entity. This attribute uniquely identifies the individual entity. Such an attribute is called a key attribute. For example, in the Employee entity type, EmpNo is the key attribute since no two employees can have same employee number. Similarly, for Product entity type, ProdId is the key attribute.

There may be a case when one single attribute is not sufficient to identify entities. Then a combination of attributes can solve this purpose. We can form a group of more than one attribute and use this combination as a key attribute. That is known as a composite key attribute. When identifying attributes of entities, identifying key attribute is very important.

Required or optional Attributes

An attribute can be required or optional. When it's required, we must have a value for it, a value must be known for each entity occurrence. When it's optional, we could have a value for it, a value may be known for each entity occurrence. For example, there is an attribute EmpNo (for employee no.) of entity employee. This is required attribute since here would be no employee having no employee no. Employee's spouse is optional attribute because an employee may or may not have a spouse.

Simple and composite Attributes

Composite attributes can be divided into smaller subparts. These subparts represent basic attributes with independent meanings of their own. For example, take Name attributes. We can divide it into sub-parts like First_name, Middle_name, and Last_name.

Attributes that can’t be divided into subparts are called Simple or Atomic attributes. For example, EmployeeNumber is a simple attribute. Age of a person is a simple attribute.

Single-valued and multi-valued attributes

Attributes that can have single value at a particular instance of time are called singlevalued. A person can’t have more than one age value. Therefore, age of a person is a single-values attribute. A multi-valued attribute can have more than one value at one time. For example, degree of a person is a multi- valued attribute since a person can have more than one degree. Where appropriate, upper and lower bounds may be placed on the number of values in a multi-valued attribute. For example, a bank may limit the number of addresses recorded for a single customer to two.

Stored, coded, or derived Attributes

There may be a case when two or more attributes values are related. Take the example of age. Age of a person can be calculated from person's date of birth and present date. Difference between the two gives the value of age. In this case, age is the derived attribute.

The attribute from which another attribute value is derived is called stored attribute. In the above example, date of birth is the stored attribute. Take another example, if we have to calculate the interest on some principal amount for a given time, and for a particular rate of interest, we can simply use the interest formula

Interest=NPR/100;

In this case, interest is the derived attribute whereas principal amount(P), time(N) and rate of interest(R) are all stored attributes.

Derived attributes are usually created by a formula or by a summary operation on other attributes.

A coded value uses one or more letters or numbers to represent a fact. For example, the value Gender might use the letters "M" and "F" as values rather than "Male" and "Female".

The attributes reflect the need for the information they provide. In the analysis meeting, the participants should list as many attributes as possible. Later they can weed out those that are not applicable to the application, or those clients are not prepared to spend the resources on to collect and maintain. The participants come to an agreement, on which attributes belong with an entity, as well as which attributes are required or optional.

Entity Types

An entity set is a set of entities of the same type that share the same properties, or attributes. For example, all software engineers working in the department involved in the Internet projects can be defined as the entity set Internet Group. The individual entities that constitute a set are called extension of the entity set. Thus, all individual software engineers of in the Internet projects are the extensions of the entity set Internet Group.

Entity sets don't need to be disjointed. For example, we can define an entity set Employee. An employee may or may not be working on some Internet projects. In Internet Group we will have some entries that are there in Employee entity set. Therefore, entity sets Employee and Internet Group are not disjoint.

A database usually contains groups of entities that are similar. For example, employees of a company share the same attributes. However, every employee entity has its own values for each attribute. An entity type defines a set of entities that have same attributes. A name and a list of attributes describe each entity type.

Fig. 2.14 shows two entity types Employee and Product. Their attribute list is also shown. A few members of each entity type are shown.

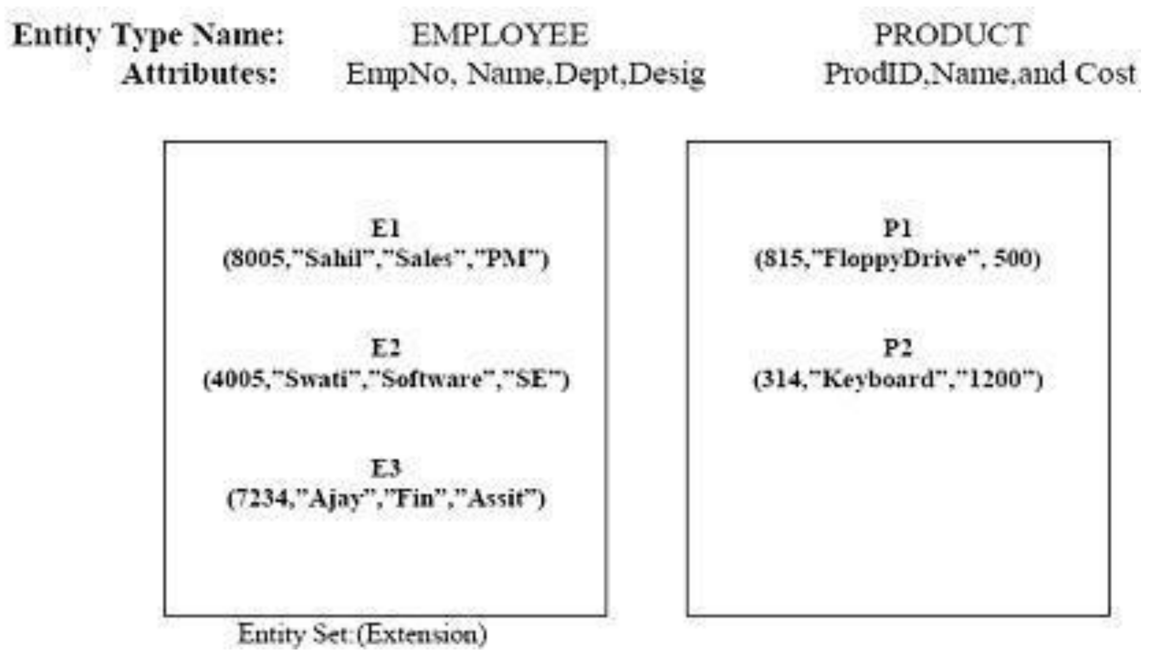


Fig. 2.14 Two entity types and some of the member entities of each

An entity type is represented in ER diagrams as rectangular box and the corresponding attributes are shown in ovals attached to the entity type by straight lines.

An entity type is basically the schema or intension or structure for the set of entities that share the same structure whereas the individual entities of a particular entity type are collectively called entity set. The entity set is also called the extension of the entity type.

Value Sets (domain) of Attributes

Each attribute of an entity type is associated with a value set. This value set is also called domain.

The domain of an attribute is the collection of all possible values an attribute can have. The value set specifies the set of values that may be assigned for each individual entity. For example, we can specify the value set for designation attribute as <"PM", "Assit", "DM", "SE">. We can specify "Name" attribute value set as <strings of alphabetic characters separated by blank characters>. The domain of Name is a character string.

Entity Relationships

After identification of entities and their attributes, the next stage in ER data modeling is to identify the relationships between these entities.

We can say a relationship is any association, linkage, or connection between the entities of interest to the business. Typically, a relationship is indicated by a verb connecting two or more entities. Suppose there are two entities of our library system, member and book, then the relationship between them can be "borrows".

Member borrows book

Each relationship has a name, degree and cardinality. These concepts will be discussed next.

Degree of an Entity Relationship Type

Relationships exhibit certain characteristics like degree, connectivity, and cardinality. Once the relationships are identified their degree and cardinality are also specified.

Degree: The degree of a relationship is the number of entities associated with the relationship. The n-ary relationship is the general form for degree n. Special cases are the binary, and ternary, where the degree is 2, and 3, respectively.

Binary relationships, the association between two entities are the most common type in the real world.

Fig 2.15 shows a binary relationship between member and book entities of library system

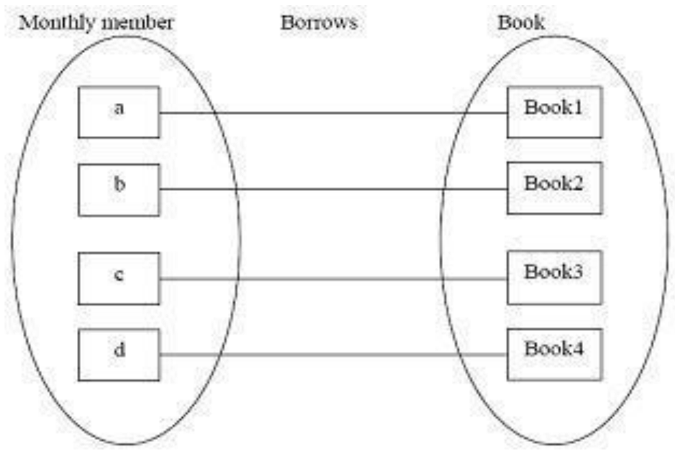


Fig. 2.15 Binary Relationship

A ternary relationship involves three entities and is used when a binary relationship is inadequate. Many modeling approaches recognize only binary relationships. Ternary or n-ary relationships are decomposed into two or more binary relationships.

Connectivity and Cardinality

By connectivity we mean how many instances of one entity are associated with how many instances of other entity in a relationship. Cardinality is used to specify such connectivity. The connectivity of a relationship describes the mapping of associated entity instances in the relationship. The values of connectivity are "one" or "many". The cardinality of a relationship is the actual number of related occurrences for each of the two entities. The basic types of connectivity for relations are: one-to-one, one-to-many, and many-to many.

A **one-to-one (1:1)** relationship is when at most one instance of an entity A is associated with one instance of entity B. For example, each book in a library is issued to only one member at a particular time.

A **one-to- many (1:N)** relationship is when for one instance of entity A, there are zero, one, or many instances of entity B but for one instance of entity B, there is only one instance of entity A. An example of a 1:N relationships is a department has many employees;

each employee is assigned to one department.

A **many-to-many (M:N)** relationship, sometimes called non-specific, is when for one instance of entity A, there are zero, one, or many instances of entity B and for one instance of entity B there are zero, one, or many instances of entity A. An example is employees may be assigned to no more than three projects at a time; every project has at least two employees assigned to it.

Here the cardinality of the relationship from employees to projects is three; from projects to employees, the cardinality is two. Therefore, this relationship can be classified as a many-to-many relationship.

If a relationship can have a cardinality of zero, it is an optional relationship. If it must have a cardinality of at least one, the relationship is mandatory. Optional relationships are typically indicated by the conditional tense. For example,

An employee may be assigned to a project.

Mandatory relationships, on the other hand, are indicated by words such as must have. For example, a student must register for at least three courses in each semester.

Designing basic model and E-R Diagrams

E-R diagrams represent the schemas or the overall organization of the system. In this section, we'll apply the concepts of E-R modeling to our "Library Management System" and draw its E-R diagram.

In order to begin constructing the basic model, the modeler must analyze the information gathered during the requirement analysis for the purpose of: and

- classifying data objects as either entities or attributes,
- identifying and defining relationships between entities,
- naming and defining identified entities, attributes, and relationships,
- documenting this information in the data document.
- Finally draw its ER diagram.

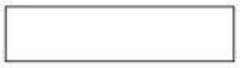
To accomplish these goals the modeler must analyze narratives from users, notes from meeting, policy and procedure documents, and, if lucky, design documents from the current information system.

E-R diagrams constructs

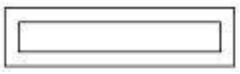
In E-R diagrams, entity types are represented by squares. See the table below. Relationship types are shown in diamond shaped boxes attached to the participating entity types with straight lines. Attributes are shown in ovals, and each attribute is attached to its entity type or relationship type by a straight line. Multi valued attributes are shown in double ovals. Key attributes have their names underlined. Derived attributes are shown in dotted ovals.

Weak entity types are distinguished by being placed in double rectangles and by having their identifying relationship placed in double diamonds.

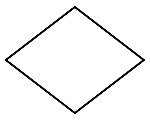
Attaching a 1, M, or N on each participating edge specifies cardinality ratio of each binary relationship type. The participation constraint is specified by a single line for partial participation and by double lines for total participation. The participation constraints specify whether the existence of an entity depends on its being related to another entity via the relationship type. If every entity of an entity set is related to some other entity set via a relationship type, then the participation of the first entity type is total. If only few member of an entity type is related to some entity type via a relationship type, the participation is partial.



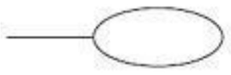
ENTITY TYPE



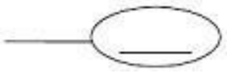
WEAK ENTITY TYPE



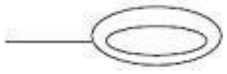
RELATIONSHIP TYPE



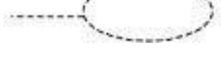
ATTRIBUTE



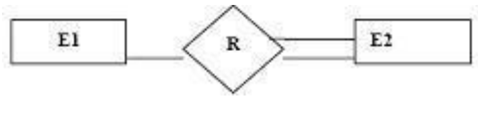
KEY ATTRIBUTE



MULTIVALUED ATTRIBUTE

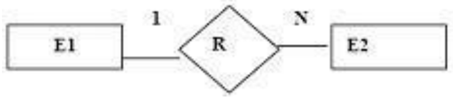


DERIVED ATTRIBUTE



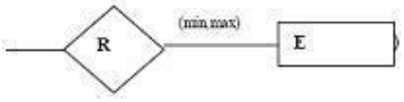
TOTAL PARTICIPATION OF

E2 IN R



Cardinality Ratio 1:N FOR

E1:E2 IN R



Structural

Constraint (Min, Max) on participation of E in R

Naming Data Objects

The names should have the following properties:

- unique,
- Have meaning to the end-user.
- contain the minimum number of words needed to uniquely and accurately describe the object

.

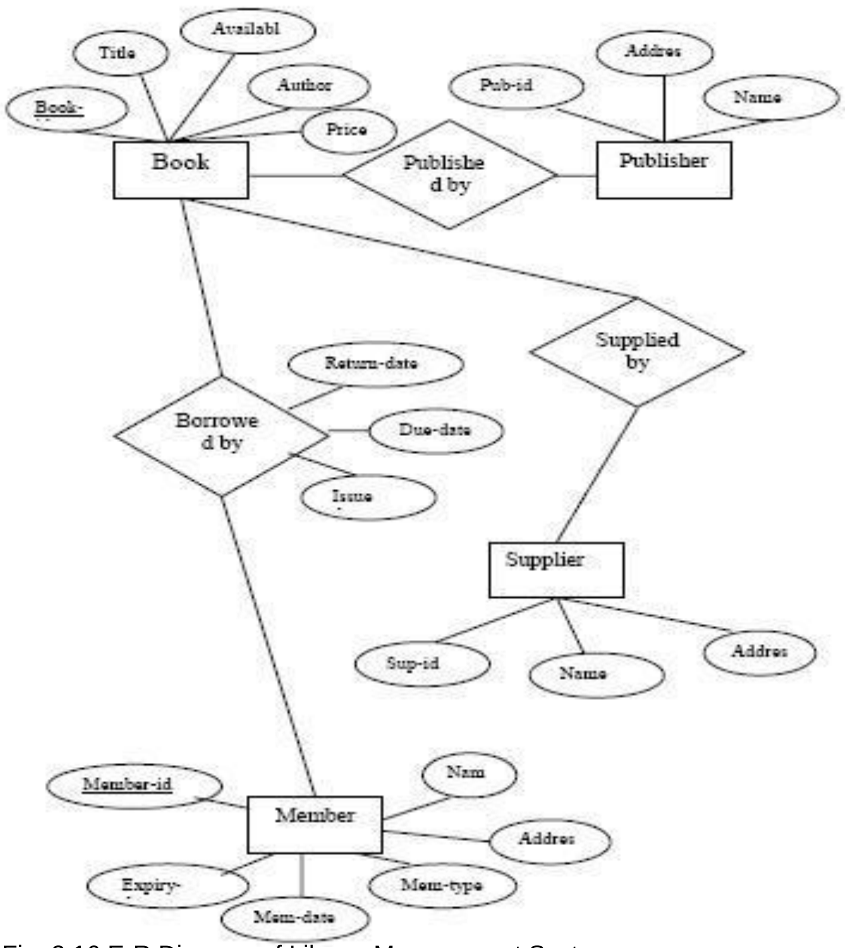


Fig. 2.16 E-R Diagram of Library Management System

Lecture 9

Data Dictionary

The analysis model encompasses representations of data objects, functions, and control. In each representation data objects and/or control items play a role. Therefore, it is necessary to provide an organized approach for representing the characteristics of each data objects and control items. This is accomplished with the data dictionary.

Formally, data dictionary can be defined as:

The data dictionary is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that both user and system analyst will have a common understanding of inputs, outputs, components of stores and even intermediate calculations.

Data dictionary contains the following information:

Name-the primary name of data or control item, the data store, or an external entity. Alias - other names used for first entry. Where- used/how-used -a listing of the processes that use the data or control item and how it is used. For example, input to a process, output from the process, as a store, as an external entity. Central description- a notation for representing content. Supplementary information - other information about data types, preset values, restrictions or limitations, etc.

The logical characteristics of current data stores, including name, description, aliases, contents, and organization. Identifies processes where the data are used and where immediate access to information is needed. Serves as the basis for identifying database requirements during system design.

Data Modeling

Data Requirements and Data modeling

Last chapter discusses about one part of the conceptual design process, the functional model. The other is the data model, which discusses the data related design issues of the system. See fig 7.1.

The data model focuses on what data should be stored in the database while the function model deals with how the data is processed. In this chapter, we'll look into details of data modeling.

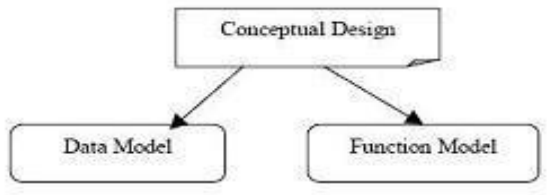


Fig 2.17 Elements of Conceptual Design

We have already discussed the Data Flow Diagrams, which make the foundation of the system under development. While the system is being studied, the physical DFDs are prepared whereas at the design phase, the basic layout of the proposed system is depicted in the form of a logical DFD. Taking this DFD as the basis the system is further developed. Even at the Data Modeling phase, the DFD can provide the basis in the form of the data flows and the Data Stores depicted in the DFD of the proposed system. The Data Stores from the

DFD are picked up and based on the data being stored by them the Data Model of the system is prepared.

Prior to data modeling, we'll talk of basics of database design process. The database design process can be described as a set of following steps.

- Requirement collection: Here the database designer interviews database users. By this process they are able to understand their data requirements. Results of this process are clearly documented. In addition to this, functional requirements are also specified. Functional requirements are user defined operations or transaction like retrievals, updates, etc, that are applied on the database.
- Conceptual schema: Conceptual schema is created. It is the description of data requirements of the users. It includes description of data types, relationships and constraints.
- Basic data model operations are used to specify user functional requirements.
- Actual implementation of database.
- Physical database design. It includes design of internal storage structures and files.

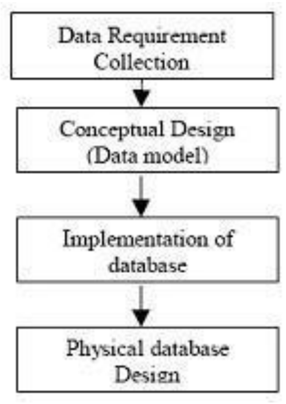

Data Requirement Collection
Conceptual Design (Data model)
Implementation of database
Physical database Design

Fig 2.18 Overall database design process

In this chapter, our main concern is data model. There are various data models available. They fall in three different groups.

· Object-based logical models

· Records-based logical models

· Physical-models

Object-Based Logical Models

Object-based logical models are used in describing data at the logical and view levels. The main characteristic of these models is that they provide flexible structuring capabilities and allows data constraints to be specified explicitly. Many different models fall into this group. They are following.

· Entity-relationship model

· Object-oriented model

In this chapter, we'll discuss Entity-Relationship model in detail. The object-oriented model is covered in the next chapter.

Record-Based Logical Models

Records-based logical models are used in describing data at the logical and view levels. They are used to specify the overall logical structure of the database and to provide a higher-level description of the implementation.

In record-based models, the database is structured in fixed-format records of several types. Each record type defines a fixed number of fields, or attributes, and each field is usually of a fixed length. The use of fixed-length records simplifies the physical-level implementation of the database.

The following models fall in this group.

· Relational model

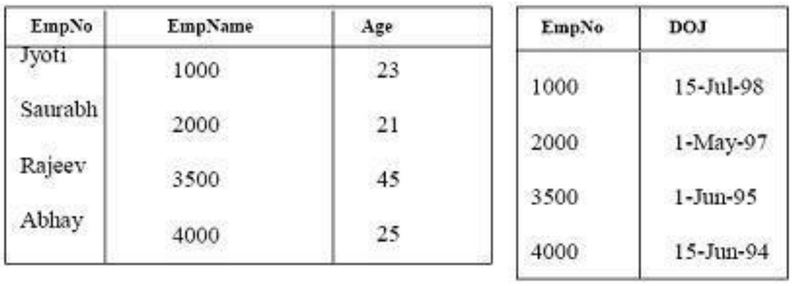
· Network model

-Hierarchical model

Relational Model

This model uses a collection of tables to represent both data and relationship among those data. Each table has multiple columns, and each column has a unique name. Figure shows a simple relational database.

·



EmpNo	EmpName	Age
Jyoti	1000	23
Saurabh	2000	21
Rajeev	3500	45
Abhay	4000	25

EmpNo	DOJ
1000	15-Jul-98
2000	1-May-97
3500	1-Jun-95
4000	15-Jun-94

Fig. 2.19 A sample relational model

Network Model

In network database, data is represented by collection of records, and relationships among data are represented by links. The records are organized as a collection of arbitrary graphs. Figure 2.20 represent a simple network database

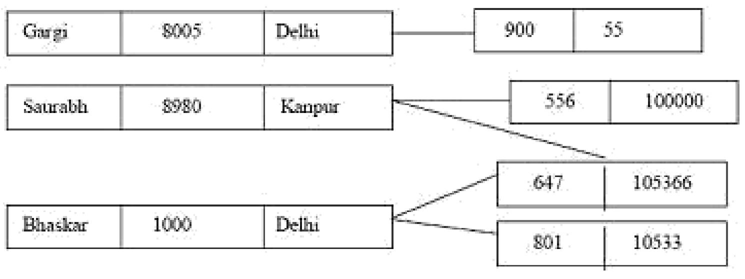


Fig. 2.20sample network model

Hierarchical Model

The hierarchical model is similar to the network model. Like network model, records and links represent data and relationships among data respectively. It differs from the network model in that the records are organized as collections of trees rather than arbitrary graphs. Fig 7.5 represents a simple database.

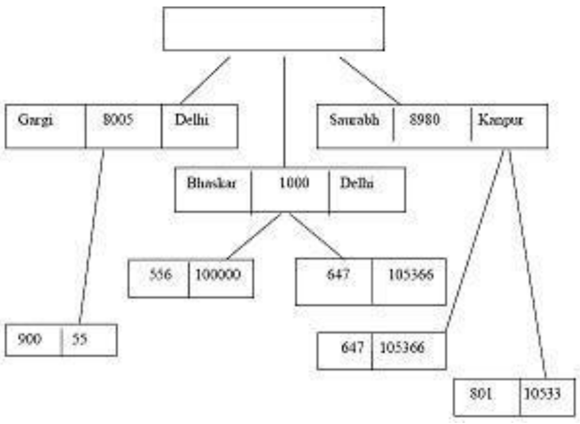


Fig. 2.21 A sample hierarchical database

Software Requirement Specification

A **software requirements specification** (SRS) is a detailed description of a software system to be developed with its functional and non-functional requirements. The SRS is developed based the agreement between customer and contractors. It may include the use cases of how user is going to interact with software system. The software requirement specification document consistent of all necessary requirements required for project development. To develop the software system we should have clear understanding of Software system. To achieve this we need to continuous communication with customers to gather all requirements.

A good SRS defines the how Software System will interact with all internal modules, hardware, communication with other programs and human user interactions with wide range of real life scenarios. Using the *Software requirements specification* (SRS) document on QA lead, managers creates test plan. It is very important that testers must be cleared with every detail specified in this document in order to avoid faults in test cases and its expected results.

It is highly recommended to review or test SRS documents before start writing test cases and making any plan for testing. Let’s see how to test SRS and the important point to keep in mind while testing it.

1. Correctness of SRS should be checked. Since the whole testing phase is dependent on SRS, it is very important to check its correctness. There are some standards with which we can compare and verify.

2. Ambiguity should be avoided. Sometimes in SRS, some words have more than one meaning and this might confused testers making it difficult to get the exact reference. It is advisable to check for such ambiguous words and make the meaning clear for better understanding.

3. Requirements should be complete. When tester writes test cases, what exactly is required from the application, is the first thing which needs to be clear. For e.g. if application needs to send the specific data of some specific size then it should be clearly mentioned in SRS that how much data and what is the size limit to send.

4. Consistent requirements. The SRS should be consistent within itself and consistent to its reference documents. If you call an input “Start and Stop” in one place, don’t call it “Start/Stop” in another. This sets the standard and should be followed throughout the testing phase.

5. Verification of expected result: SRS should not have statements like “Work as expected”, it should be clearly stated that what is expected since different testers would have different thinking aspects and may draw different results from this statement.

6. Testing environment: some applications need specific conditions to test and also a particular environment for accurate result. SRS should have clear documentation on what type of environment is needed to set up.

7. Pre-conditions defined clearly: one of the most important part of test cases is pre-conditions. If they are not met properly then actual result will always be different expected result. Verify that in SRS, all the pre-conditions are mentioned clearly.

8. Requirements ID: these are the base of test case template. Based on requirement Ids, test case ids are written. Also, requirements ids make it easy to categorize modules so just by looking at them, tester will know which module to refer. SRS must have them such as id defines a particular module.

9. Security and Performance criteria: security is priority when a software is tested especially when it is built in such a way that it contains some crucial information when leaked can cause harm to business. Tester should check that all the security related requirements are properly defined and are clear to him. Also, when we talk about performance of a software, it plays a very important role in business so all the requirements related to performance must be clear to the tester and he must also know when and how much stress or load testing should be done to test the performance.

10. Assumption should be avoided: sometimes when requirement is not cleared to tester, he tends to make some assumptions related to it, which is not a right way to do testing as assumptions could go wrong and hence, test results may vary. It is better to avoid assumptions and ask clients about all the “missing requirements” to have a better understanding of expected results.

11. Deletion of irrelevant requirements: there are more than one team who work on SRS so it might be possible that some irrelevant requirements are included in SRS. Based on the understanding of the software, tester can find out which are these requirements and remove them to avoid confusions and reduce work load.

12. Freeze requirements: when an ambiguous or incomplete requirement is sent to client to analyze and tester gets a reply, that requirement result will be updated in the next SRS version and client will freeze that requirement. Freezing here means that result will not change again until and unless some major addition or modification is introduced in the software.

Lecture 10

Software Design Aspects: Objective, Principles concepts

Once the requirements document for the software to be developed is available, the software design phase begins. While the requirement specification activity deals entirely with the problem domain, design is the first phase of transforming the problem into a solution. In the design phase, the customer and business requirements and technical considerations all come together to formulate a product or a system.

The design process comprises a set of principles, concepts and practices, which allow a software engineer to model the system or product that is to be built. This model, known as design model, is assessed for quality and reviewed before a code is generated and tests are conducted. The design model provides details about software data structures, architecture, interfaces and components which are required to implement the system. This chapter discusses the design elements that are required to develop a software design model. It also discusses the design patterns and various software design notations used to represent a software design

Basic of Software Design

Software design is a phase in software engineering, in which a blueprint is developed to serve as a base for constructing the software system. **IEEE** defines software design as 'both a process of defining, the architecture, components, interfaces, and other characteristics of a system or component and the result of that process.'

In the design phase, many critical and strategic decisions are made to achieve the desired functionality and quality of the system. These decisions are taken into account to successfully develop the software and carry out its maintenance in a way that the quality of the end product is improved.

Principles of Software Design

Developing design is a cumbersome process as most expansive errors are often introduced in this phase. Moreover, if these errors get unnoticed till later phases, it becomes more difficult to correct them. Therefore, a number of principles are followed while designing the software. These principles act as a framework for the designers to follow a good design practice.

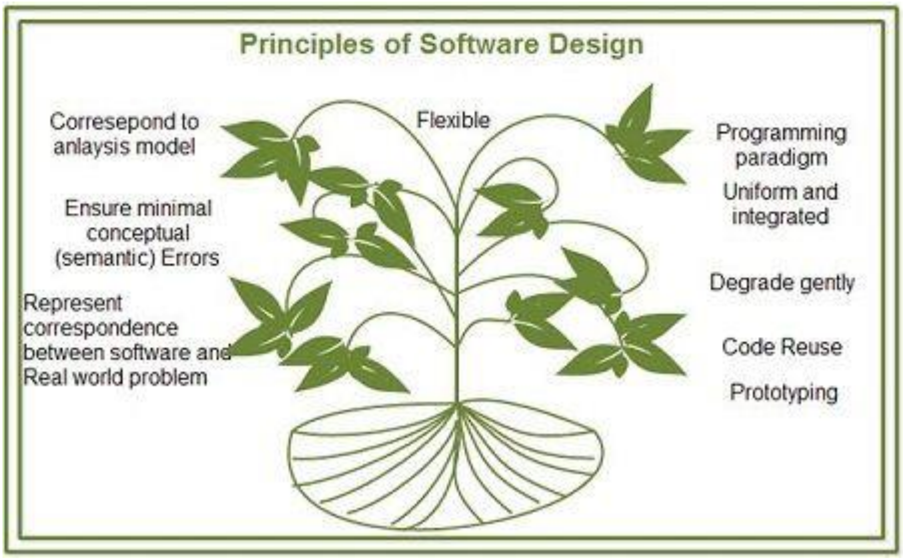

Principles of Software Design
Corresepond to anlaysis model
Flexible
Programming paradigm
Uniform and integrated
Ensure minimal conceptual (semantic) Errors
Degrade gently
Represent correspondence between software and Real world problem
Code Reuse
Prototyping

Fig 2.22 principles of software design

Some of the commonly followed design principles are as following.

1. **Software design should correspond to the analysis model:** Often a design element corresponds to many requirements, therefore, we must know how the design model satisfies all the requirements represented by the analysis model.
2. **Choose the right programming paradigm:** A programming paradigm describes the structure of the software system. Depending on the nature and type of application, different programming paradigms such as procedure oriented, object-oriented, and prototyping paradigms can be used. The paradigm should be chosen keeping constraints in mind such as time, availability of resources and nature of user's requirements.
3. **Software design should be uniform and integrated:** Software design is considered uniform and integrated, if the interfaces are properly defined among the design components. For this, rules, format, and styles are established before the design team starts designing the software.
4. **Software design should be flexible:** Software design should be flexible enough to adapt changes easily. To achieve the flexibility, the basic design concepts such as abstraction, refinement, and modularity should be applied effectively.
5. **Software design should ensure minimal conceptual (semantic) errors:** The design team must ensure that major conceptual errors of design such as ambiguousness and inconsistency are addressed in advance before dealing with the syntactical errors present in the design model.
6. **Software design should be structured to degrade gently:** Software should be designed to handle unusual changes and circumstances, and if the need arises for termination, it must do so in a proper manner so that functionality of the software is not affected.
7. **Software design should represent correspondence between the software and real-world problem:** The software design should be structured in such away that it always relates with the real-world problem.
8. **Software reuse:** Software engineers believe on the phrase: 'do not reinvent the wheel'. Therefore, software components should be designed in such a way that they can be effectively reused to increase the productivity.
9. **Designing for testability:** A common practice that has been followed is to keep the testing phase separate from the design and implementation phases. That is, first the software is developed (designed and implemented) and then handed over to the testers who subsequently determine whether the software is fit for distribution and subsequent use by the customer. However, it has become apparent that the process of separating testing is seriously flawed, as if any type of design or implementation errors are found after implementation, then the entire or a substantial part of the software requires to be redone. Thus, the test engineers should be involved from the initial stages. For example, they should be involved with analysts to prepare tests for determining whether the user requirements are being met.
10. **Prototyping:** Prototyping should be used when the requirements are not completely defined in the beginning. The user interacts with the developer to expand and refine the requirements as the development proceeds. Using prototyping, a quick 'mock-up' of the system can be developed. This mock-up can be used as a effective means to give the users a feel of what the system will look like and demonstrate functions that will be included in the developed system. Prototyping also helps in reducing risks of designing software that is not in accordance with the customer's requirements.

Software Design Concepts

Every software process is characterized by basic concepts along with certain practices or methods. Methods represent the manner through which the concepts are applied. As new technology replaces older technology, many changes occur in the methods that are used to apply the concepts for the development of software. However, the fundamental concepts underlining the software design process remain the same, some of which are described here.

Abstraction

Abstraction refers to a powerful design tool, which allows software designers to consider components at an abstract level, while neglecting the implementation details of the components. **IEEE** defines abstraction as 'a view of a problem that extracts the essential information relevant to a particular purpose and ignores the remainder of the information.' The concept of abstraction can be used in two ways: as a process and as an entity. As a **process,** it refers to a mechanism of hiding irrelevant details and representing only the essential features of an item so that one can focus on important things at a time. As an **entity,** it refers to a model or view of an item.

Each step in the software process is accomplished through various levels of abstraction. At the highest level, an outline of the solution to the problem is presented whereas at the lower levels, the solution to the problem is presented in detail. For example, in the requirements analysis phase, a solution to the problem is presented using the language of problem environment and as we proceed through the software process, the abstraction level reduces and at the lowest level, source code of the software is produced.

There are three commonly used abstraction mechanisms in software design, namely, functional abstraction, data abstraction and control abstraction. All these mechanisms allow us to control the complexity of the design process by proceeding from the abstract design model to concrete design model in a systematic manner.

1. **Functional abstraction:** This involves the use of parameterized subprograms. Functional abstraction can be generalized as collections of subprograms referred to as 'groups'. Within these groups there exist routines which may be visible or hidden. Visible routines can be used within the containing groups as well as within other groups, whereas hidden routines are hidden from other groups and can be used within the containing group only.
2. **Data abstraction:** This involves specifying data that describes a data object. For example, the data object *window* encompasses a set of attributes (window type, window dimension) that describe the window object clearly. In this abstraction mechanism, representation and manipulation details are ignored.
3. **Control abstraction:** This states the desired effect, without stating the exact mechanism of control. For example, if and while statements in programming languages (like C and C++) are abstractions of machine code implementations, which involve conditional instructions. In the architectural design level, this abstraction mechanism permits specifications of sequential subprogram and exception handlers without the concern for exact details of implementation.

Architecture

Software architecture refers to the structure of the system, which is composed of various components of a program/ system, the attributes (properties) of those components and the relationship amongst them. The software architecture enables the software engineers to analyze the software design efficiently. In addition, it also helps them in decision-making and handling risks. The software architecture does the following.

- Provides an insight to all the interested stakeholders that enable them to communicate with each other
- Highlights early design decisions, which have great impact on the software engineering activities (like coding and testing) that follow the design phase
- Creates intellectual models of how the system is organized into components and how these components interact with each other.

Currently, software architecture is represented in an informal and unplanned manner. Though the architectural concepts are often represented in the infrastructure (for supporting particular architectural styles) and the initial stages of a system configuration, the lack of an explicit independent characterization of architecture restricts the advantages of this design concept in the present scenario.

Note that software architecture comprises two elements of design model, namely, data design and architectural design.

Patterns

A pattern provides a description of the solution to a recurring design problem of some specific domain in such a way that the solution can be used again and again. The objective of each pattern is to provide an insight to a designer who can determine the following.

1. Whether the pattern can be reused
2. Whether the pattern is applicable to the current project
3. Whether the pattern can be used to develop a similar but functionally or structurally different design pattern.

Types of Design Patterns

Software engineer can use the design pattern during the entire software design process. When the analysis model is developed, the designer can examine the problem description at different levels of abstraction to determine whether it complies with one or more of the following types of design patterns.

1. **Architectural patterns:** These patterns are high-level strategies that refer to the overall structure and organization of a software system. That is, they define the elements of a software system such as subsystems, components, classes, etc. **In** addition, they also indicate the relationship between the elements along with the rules and guidelines for specifying these relationships. Note that architectural patterns are often considered equivalent to software architecture.
2. **Design patterns:** These patterns are medium-level strategies that are used to solve design problems. They provide a means for the refinement of the elements (as defined by architectural pattern) of a software system or the relationship among them. Specific design elements such as relationship among components or mechanisms that affect component-to-component interaction are addressed by design patterns. Note that design patterns are often considered equivalent to software components.
3. **Idioms:** These patterns are low-level patterns, which are programming-language specific. They describe the implementation of a software component, the method used for interaction among software components, etc., in a specific programming language. Note that idioms are often termed as coding patterns.

Modularity

Modularity is achieved by dividing the software into uniquely named and addressable components, which are also known as **modules.** A complex system (large program) is partitioned into a set of discrete modules in such a way that each module can be developed independent of other modules. After developing the modules, they are integrated together to meet the software requirements. Note that larger the number of modules a system is divided into, greater will be the effort required to integrate the modules.

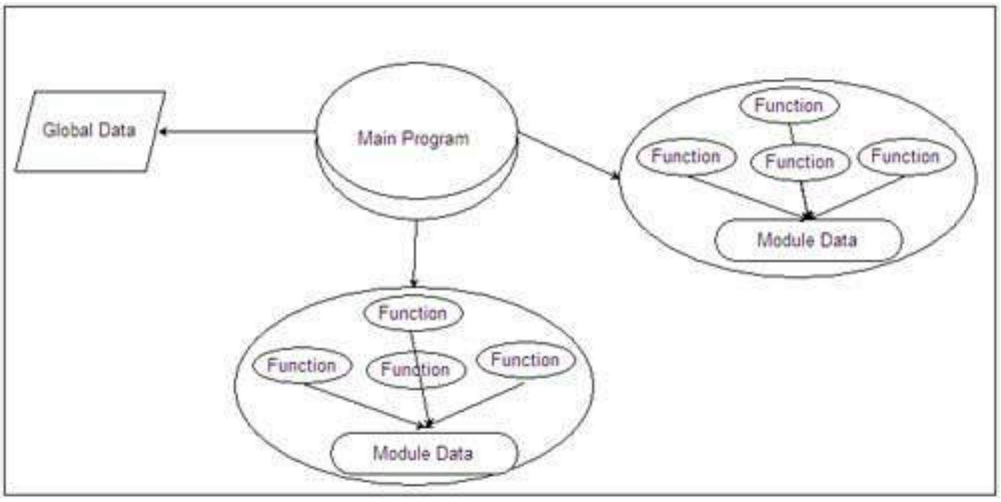


Fig 2.23 Modularity

Modularizing a design helps to plan the development in a more effective manner, accommodate changes easily, conduct testing and debugging effectively and efficiently, and conduct maintenance work without adversely affecting the functioning of the software.

Information Hiding

Modules should be specified and designed in such a way that the data structures and processing details of one module are not accessible to other modules. They pass only that much information to each other, which is required to accomplish the software functions. The way of hiding unnecessary details is referred to as **information hiding. IEEE** defines information hiding as 'the technique of encapsulating software design decisions in modules in such a way that the module's interfaces reveal as little as possible about the module's inner workings; thus each module is a 'black box' to the other modules in the system.

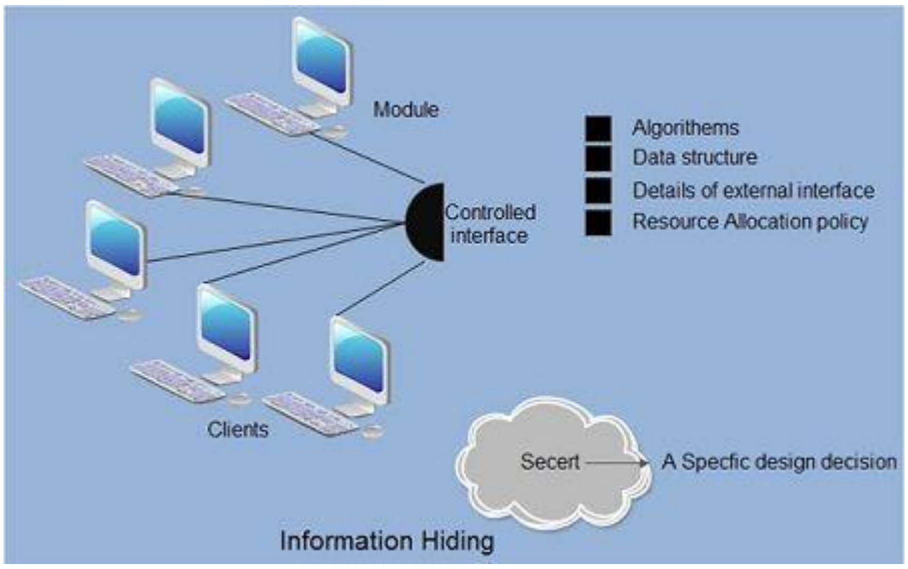


Fig 2.24 Information Hiding

Information hiding is of immense use when modifications are required during the testing and maintenance phase. Some of the advantages associated with information hiding are listed below.

1. Leads to low coupling
2. Emphasizes communication through controlled interfaces

3. Decreases the probability of adverse effects
4. Restricts the effects of changes in one component on others
5. Results in higher quality software.

Structural Partitioning

When the architectural style of a design follows a hierarchical nature, the structure of the program can be partitioned either horizontally or vertically. In **horizontal partitioning,** the control modules are used to communicate between functions and execute the functions. Structural partitioning provides the following benefits.

- The testing and maintenance of software becomes easier.
- The negative impacts spread slowly.
- The software can be extended easily.

Besides these advantages, horizontal partitioning has some disadvantage also. It requires to pass more data across the module interface, which makes the control flow of the problem more complex. This usually happens in cases where data moves rapidly from one function to another.

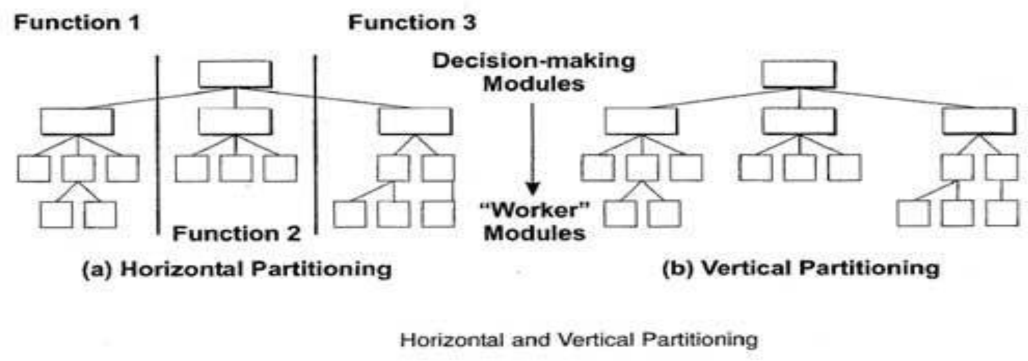

Function 1
Function 3
Function 2
Decision-making Modules
"Worker" Modules
(a) Horizontal Partitioning
(b) Vertical Partitioning
Horizontal and Vertical Partitioning

Fig.2.25

In **vertical partitioning**, the functionality is distributed among the modules--in a top-down manner. The modules at the top level called **control modules** perform the decision-making and do little processing whereas the modules at the low level called **worker modules** perform all input, computation and output tasks.

Concurrency

Computer has limited resources and they must be utilized efficiently as much as possible. To utilize these resources efficiently, multiple tasks must be executed concurrently. This requirement makes concurrency one of the major concepts of software design. Every system must be designed to allow multiple processes to execute concurrently, whenever possible. For example, if the current process is waiting for some event to occur, the system must execute some other process in the mean time.

However, concurrent execution of multiple processes sometimes may result in undesirable situations such as an inconsistent state, deadlock, etc. For example, consider two processes A and B and a data item Q1 with the value '200'. Further, suppose A and B are being executed concurrently and firstly A reads the value of Q1 (which is '200') to add '100' to it. However, before A updates as the value of Q1, B reads the value ofQ1 (which is still '200') to add '50' to it. In this situation, whether A or B first updates the value of Q1, the value of would definitely be wrong resulting in an inconsistent state of the system. This is because the actions of A and B are not synchronized with each other. Thus, the system must control the concurrent execution and synchronize the actions of concurrent processes.

One way to achieve synchronization is mutual exclusion, which ensures that two concurrent processes do not interfere with the actions of each other. To ensure this, mutual exclusion may use locking technique. In this technique, the processes need to lock the data item to be read or updated. The data item locked by some process cannot be accessed by other processes until it is unlocked. It implies that the process, that needs to access the data item locked by some other process, has to wait.

Developing a Design Model

To develop a complete specification of design (design model), four design models are needed. These models are listed below.

1. **Data design:** This specifies the data structures for implementing the software by converting data objects and their relationships identified during the analysis phase. Various studies suggest that design engineering should begin with data design, since this design lays the foundation for all other design models.
2. **Architectural design:** This specifies the relationship between the structural elements of the software, design patterns, architectural styles, and the factors affecting the ways in which architecture can be implemented.
3. **Component-level design:** This provides the detailed description of how structural elements of software will actually be implemented.
4. **Interface design:** This depicts how the software communicates with the system that interoperates with it and with the end-users.

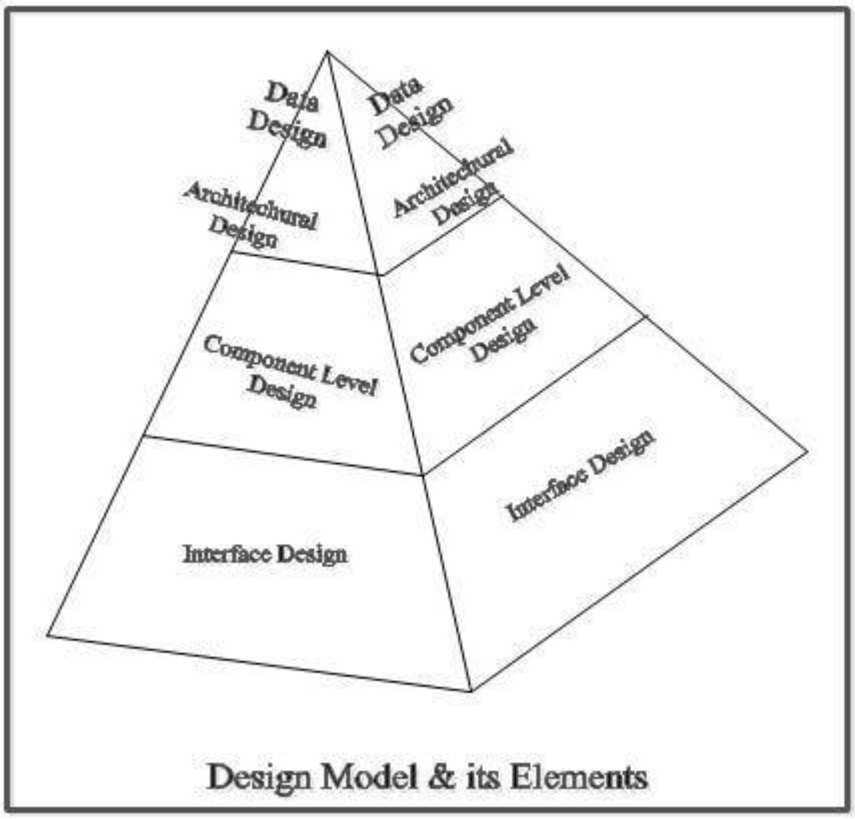


Fig. 2.26

Lecture 11

HLD and LLD

High – level Design gives the overall System Design in terms of Functional Architecture and Database design. It designs the over all architecture of the entire system from main module to all sub module. This is very useful for the developers to understand the flow of the system. In this phase design team, review team (testers) and customers plays a major role. For this the entry criteria are the requirement document that is SRS. And the exit criteria will be HLD, projects standards, the functional design documents, and the database design document. Further, High level deign gives the overview of the development of product. In other words how the program is going to be divided into functions, modules, subdivision etc.

Low – Level Design (LLD): During the detailed phase, the view of the application developed during the high level design is broken down into modules and programs. Logic design is done for every program and then documented as program specifications. For every program, a unit test plan is created. The entry criteria for this will be the HLD document. And the exit criteria will the program specification and unit test plan (LLD).

The Low Level Design Document gives the design of the actual program code which is designed based on the High Level Design Document. It defines Internal logic of corresponding sub module designers are preparing and mapping individual LLD's to Every module. A good Low Level Design Document developed will make the program very easy to be developed by developers because if proper analysis is made and the Low Level Design Document is prepared then the code can be developed by developers directly from Low Level Design Document with minimal effort of debugging and testing.

Top down and Bottom up Design

Top Down Design

We know that a system is composed of more than one sub-systems and it contains a number of components. Further, these sub-systems and components may have their onset of sub-system and components and creates hierarchical structure in the system.

Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics. Each sub-system or component is then treated as a system and decomposed further. This process keeps on running until the lowest level of system in the top-down hierarchy is achieved.

Top-down design starts with a generalized model of system and keeps on defining the more specific part of it. When all components are composed the whole system comes into existence.

Top-down design is more suitable when the software solution needs to be designed from scratch and specific details are unknown.

Bottom-up Design

The bottom up design model starts with most specific and basic components. It proceeds with composing higher level of components by using basic or lower level components. It keeps creating higher level components until the desired system is not evolved as one single component. With each higher level, the amount of abstraction is increased.

Bottom-up strategy is more suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system.

Both, top-down and bottom-up approaches are not practical individually. Instead, a good combination of both is used.

User interface is the front-end application view to which user interacts in order to use the software. User can manipulate and control the software as well as hardware by means of user interface. Today, user interface is found at almost every place where digital technology exists, right from computers, mobile phones, cars, music players, airplanes, ships etc.

User interface is part of software and is designed such a way that it is expected to provide the user insight of the software. UI provides fundamental platform for human-computer interaction.

UI can be graphical, text-based, audio-video based, depending upon the underlying hardware and software combination. UI can be hardware or software or a combination of both.

The software becomes more popular if its user interface is:

- Attractive
- Simple to use
- Responsive in short time
- Clear to understand
- Consistent on all interfacing screens

UI is broadly divided into two categories:

- Command Line Interface
- Graphical User Interface

Decision Tree

Decision tree is a tree like structure that represents the various conditions and the subsequent possible actions. It also shows the priority in which the conditions are to be tested or addressed. Each of its branches stands for any one of the logical alternatives and because of the branch structure, it is known as a tree.

The decision sequence starts from the root of the tree that is usually on the left of the diagram. The path to be followed to traverse the branches is decided by the priority of the conditions and the respectable actions. A series of decisions are taken, as the branches are traversed from left to right. The nodes are the decision junctions. After each decision point there are next set of decisions to be considered. Therefore at every node of the tree represented conditions are considered to determine which condition prevails before moving further on the path.

This decision tree representation form is very beneficial to the analyst. The first advantage is that by using this form the analyst is able to depict all the given parameters in a logical format which enables the simplification of the whole decision process as now there is a very remote chance of committing an error in the decision process as all the options are clearly specified in one of the most simplest manner.

Secondly it also aids the analyst about those decisions, which can only be taken when couple or more conditions should hold true together for there may be a case where other conditions are relevant only if one basic condition holds true.

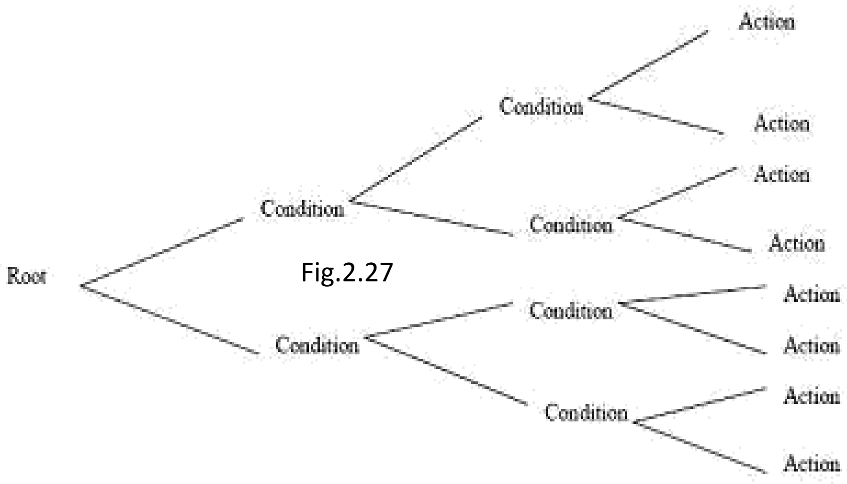


Fig.2.27

In our day-to-day life, many a times we come across complex cases where the most appropriate action under several conditions is not apparent easily and for such a case a decision tree is a great aid. Hence this representation is very effective in describing the business problems involving more than one dimension and parameters.

They also point out the required data, which surrounds the decision process. All the data used in the decision making should be first described and defined by the analyst so that the system can be designed to produce correct output data.

Consider for example the discount policy of a saree manufacturer for his customers.

According to the policy the saree manufacturer give discount to his customers based on the type of customer and size of their order. For the individual, only if the order size is 12 or more, the manufacturer gives a discount of 50% and for less than 12 sarees the discount is 30%. Whereas in case of shopkeeper or retailers, the discount policy is different. If the order is less than 12 then there is 15% discount. For 13 to 48 sarees order, the discount is 30%, for

49 to 84 sarees 40% and for more than 85 sarees the discount is 50%. The decision policy for discount percentage can be put in the form of a decision tree displayed in fig 4.2

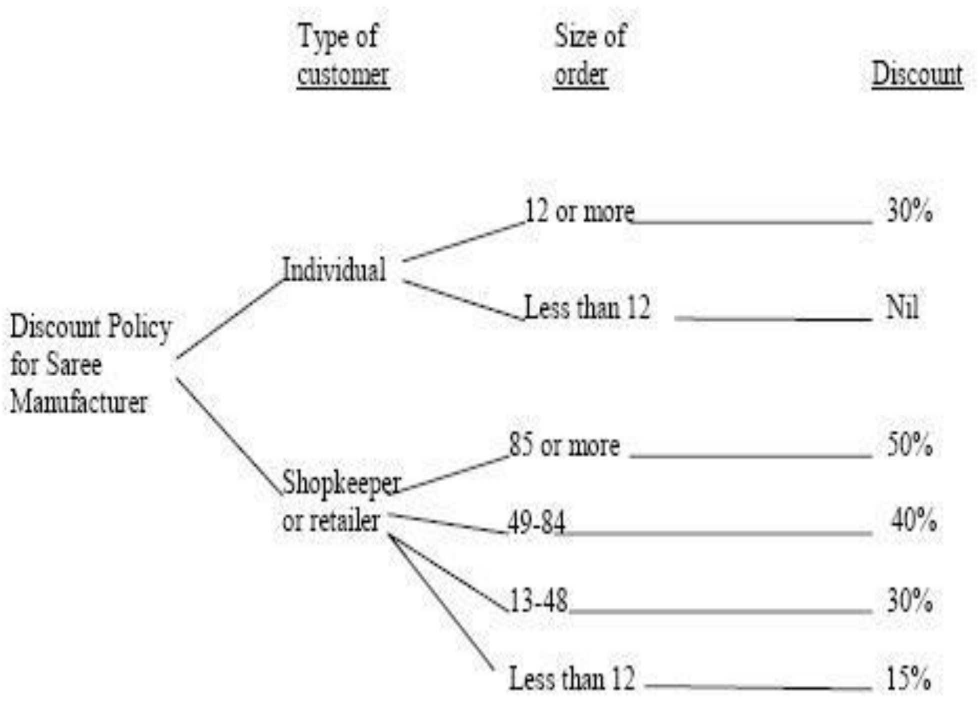


Fig.2.28

A Sample Decision Tree

The decision trees are not always the most appropriate and the best tool for the decision making process. Representing a very complex system with this tool may lead to a huge number of branches with a similar number of possible paths and options.

Decision Table

A decision table is a table with various conditions and their corresponding actions. Decision tree is a two dimensional matrix. It is divided into four parts, condition stub, action stub, condition entry, and action entry. See fig 4.3. Condition stub shows the various possible conditions.

Condition entry is used for specifying which condition is being analyzed. Action stub shows the various actions taken against different conditions.

And action entry is used to find out which action is taken corresponding to a particular set of conditions.

The steps to be taken for a certain possible condition are listed by action statements. Action entries display what specific actions to be undertaken when selected conditions or combinations of conditions are true. At times notes are added below the table to indicate when to use the table or to distinguish it from other decisions tables.

The right side columns of the table link conditions and actions and form the decision rules hence they state the conditions that must be fulfilled for a particular set of actions to be taken. In the decision trees, a fixed ordered sequence is followed in which conditions are examined. But this is not the case here as the decision rule incorporates all the required conditions, which must be true.

Developing Decision Tables

Before describing the steps involved in building the decision table it is important to take a note of few important points. Every decision should be given a name and the logic of the decision table is independent of the sequence in which condition rules are written but the action takes place in the order in which events occur. Wherever possible, duplication of terms and meaning should be avoided and only the standardized language must be used.

The steps of building the concerned tables are given below.

1. Firstly figure out the most essential factors to be considered in making a decision.

This will identify the conditions involved in the decision. Only those conditions should be selected which have the potential to either occur or not but partial occurrences are not permissible.

2. Determine the most possible steps that can take place under varying conditions and not just under current condition. This step will identify the actions.

3. Calculate all the possible combinations of conditions.

For every N number of conditions there are 2*2*2…. (N times) combinations to be considered.

4. Fill the decision rules in the table.

Entries in a decision table are filled as Y/N and action entries are generally marked as "X". For the conditions that are immaterial a hyphen "-" is generally put. Decision table is further simplified by eliminating and consolidating certain rules. Impossible rules are eliminated. There are certain conditions whose values do not affect the decision and always result in the same action. These rules can be consolidated into a single rule.

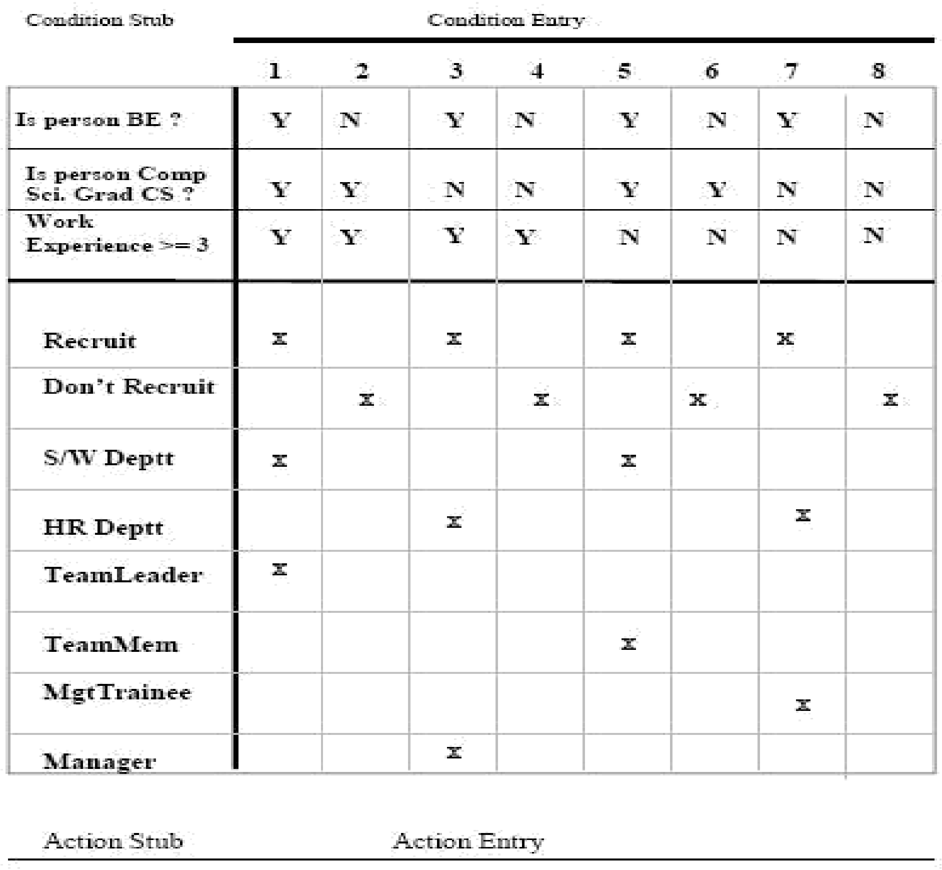
Example: Consider the recruitment policy of ABC Software Ltd.

It the applicant is a BE then recruit otherwise not. If the person is from Computer Science, put him/her in the software development department and if the person is from non-computer science background put him/her in HR department. If the Person is from Computer Science and having experience equal to or greater than three years, take him/her as Team leader and if the experience is less than that then take the person as Team member. If the person recruited is from non Computer Science background, having experience less than three years, make him/her Management Trainee otherwise Manager

Condition Stub	**Condition Entry**					
	1	**2**	**3**	**4**	**5**	**6**
Customer is individual ?	*Y*	*Y*				
Customer shopkeeper or retailer ?			Y	Y	Y	Y
Order-size 85 copies or more ?			Y			
Order-size 49-84 sarees ?				Y		
Order-size 13-48 copies ?					Y	
Order-size 12 or more ?		Y				
Order-size less than 12?	Y					Y
Allow 50% discount		X	X			
Allow 40% discount				X		
Allow 30% discount	X				X	
Allow 15% discount						X

Fig 2.28 Decision table-Discount Policy

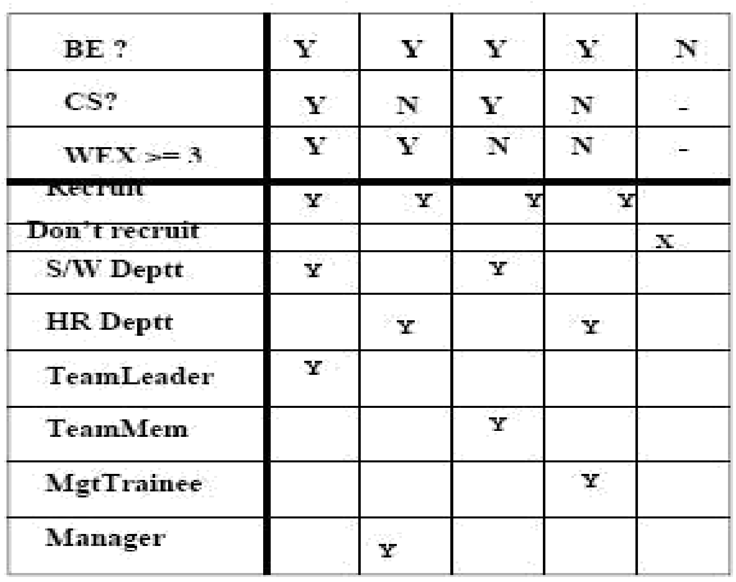
The first decision table for the problem stated above can be drawn as shown



Condition Stub	Condition Entry							
	1	2	3	4	5	6	7	8
Is person BE ?	Y	N	Y	N	Y	N	Y	N
Is person Comp Sci. Grad CS ?	Y	Y	N	N	Y	Y	N	N
Work Experience >= 3	Y	Y	Y	Y	N	N	N	N
Recruit	x		x		x		x	
Don't Recruit		x		x		x		x
S/W Deptt	x				x			
HR Deptt			x				x	
TeamLeader	x							
TeamMem					x			
MgtTrainee							x	
Manager			x					
Action Stub	Action Entry							

Fig.2.29

This table can further be refined by combining condition entries 2, 4, 6, and 8. The simplified table is displayed in figure 4.5.



BE ?	Y	Y	Y	Y	N
CS?	Y	N	Y	N	-
WEX >= 3	Y	Y	N	N	-
Recruit	Y	Y	Y	Y	
Don't recruit					X
S/W Deptt	Y		Y		
HR Deptt		Y		Y	
TeamLeader	Y				
TeamMem			Y		
MgtTrainee				Y	
Manager		Y			

Fig.2.30 .5 ision Table

Structured English

Structured English is one more tool available to the analyst. It comes as an aid against the problems of ambiguous language in stating condition and actions in decisions and procedures. Here no trees or tables are employed, rather with narrative statements a procedure is described. Thus it does not show but states the decision rules. The analyst is first required to identify the conditions that occur in the process, subsequent decisions, which are to be made and the alternative actions to be taken.

Here the steps are clearly listed in the order in which they should be taken. There are no special symbols or formats involved unlike in the case of decision trees and tables, also the entire procedure can be stated quickly as only English like statements are used.

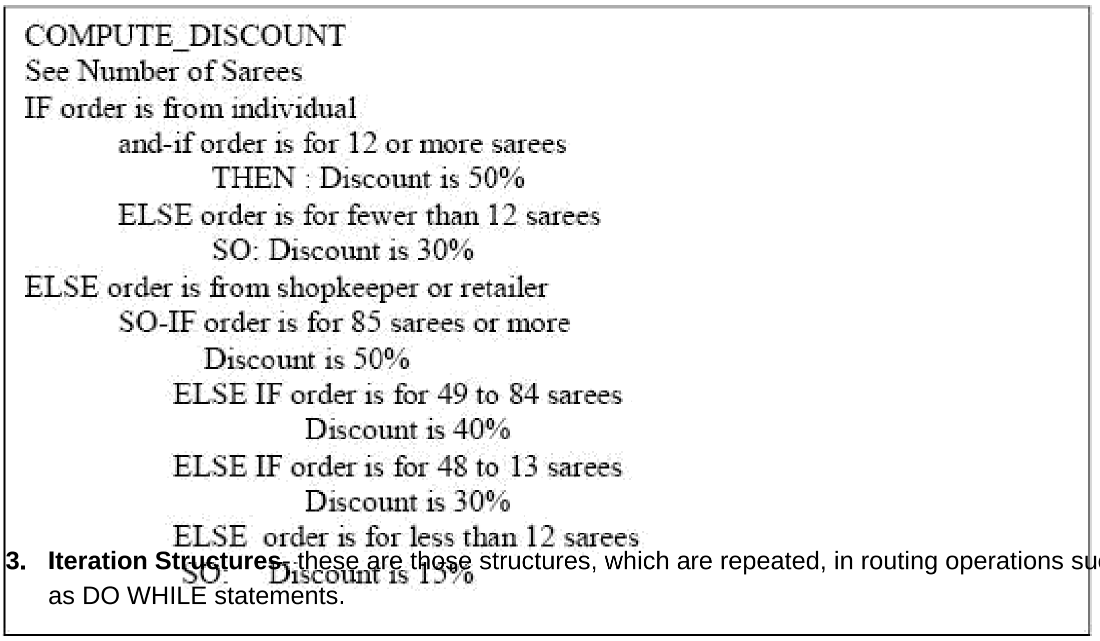
Structured English borrows heavily from structured programming as it uses logical construction and imperative statements designed to carry out instructions for actions. Using "IF", "THEN", "ELSE" and "So" statement decisions are made. In this structured description terms from the data dictionary are widely used which makes the description compact and straight.

Developing Structured Statements

Three basic types of statements are employed to describe the process.

1. **Sequence Structures** - A sequence structure is a single step or action included in a process. It is independent of the existence of any condition and when encountered it is always taken. Usually numerous such instructions are used together to describe a process.

2. **Decision Structures** - Here action sequences described are often included within decision structures that identify conditions. Therefore these structures occur when two or more actions can be taken as per the value of a specific condition. Once the condition is determined the actions are unconditional.



```
COMPUTE_DISCOUNT
See Number of Sarees
IF order is from individual
          and-if order is for 12 or more sarees
                    THEN : Discount is 50%
          ELSE order is for fewer than 12 sarees
                    SO: Discount is 30%
ELSE order is from shopkeeper or retailer
          SO-IF order is for 85 sarees or more
                   Discount is 50%
                ELSE IF order is for 49 to 84 sarees
                             Discount is 40%
                ELSE IF order is for 48 to 13 sarees
                             Discount is 30%
                ELSE  order is for less than 12 sarees
                   SO:     Discount is 15%
```

3. **Iteration Structures**- these are those structures, which are repeated, in routing operations such as DO WHILE statements.

Fig.2.31

An example of Structured English

Lecture 12

Structure Chart

Once the flow of data and control in the system is decided using tools like DFDs and CFDs, the system is given shape through programming. Prior to this, the basic infrastructure of the program layout is prepared based on the concepts of modular programming.

In modular programming, the complete system is coded as small independent interacting modules. Each module is aimed at doing one specific task. The design for these modules is prepared in the form of structure charts.

A structure chart is a design tool that pictorially shows the relation between processing modules in computer software. Describes the hierarchy of components modules and the data are transmitted between them. Includes analysis of input-to-output transformations and analysis of transaction.

Structure charts show the relation of processing modules in computer software. It is a design tool that visually displays the relationships between program modules. It shows which module within a system interacts and graphically depicts the data that are communicated between various modules.

Structure charts are developed prior to the writing of program code. They identify the data passes existing between individual modules that interact with one another.

They are not intended to express procedural logic. This task is left to flowcharts and pseudocode.

They don't describe the actual physical interface between processing functions.

Notation

Program modules are identified by rectangles with the module name written inside the rectangle.

Arrows indicate calls, which are any mechanism used to invoke a particular module.

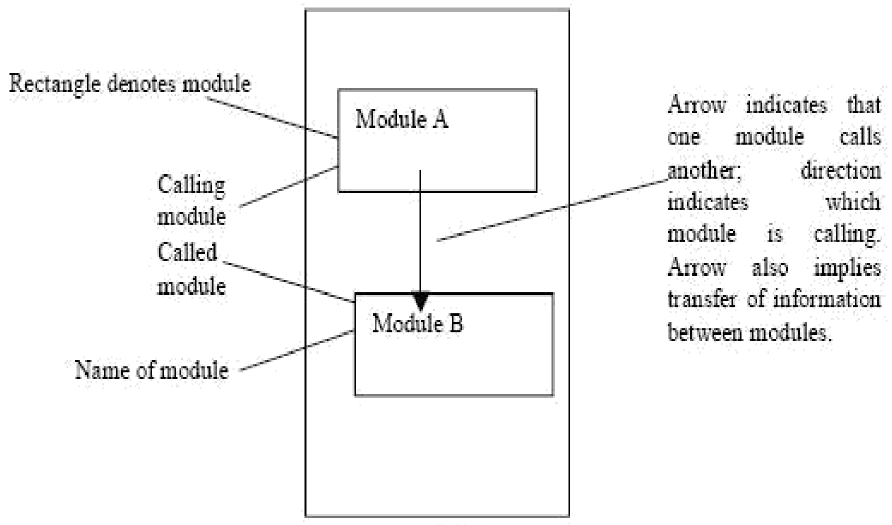

Rectangle denotes module
Module A
Calling module
Called module
Module B
Name of module
Arrow indicates that one module calls another; direction indicates which module is calling. Arrow also implies transfer of information between modules.

Fig. 6.2
Notation used in structure charts.

Annotations on the structure chart indicate the parameter that are passed and the direction of the data movement. In fig. 2.33, we see that modules A and B interact. Data identified as X and Y are passed to module B, which in turn passes back Z.

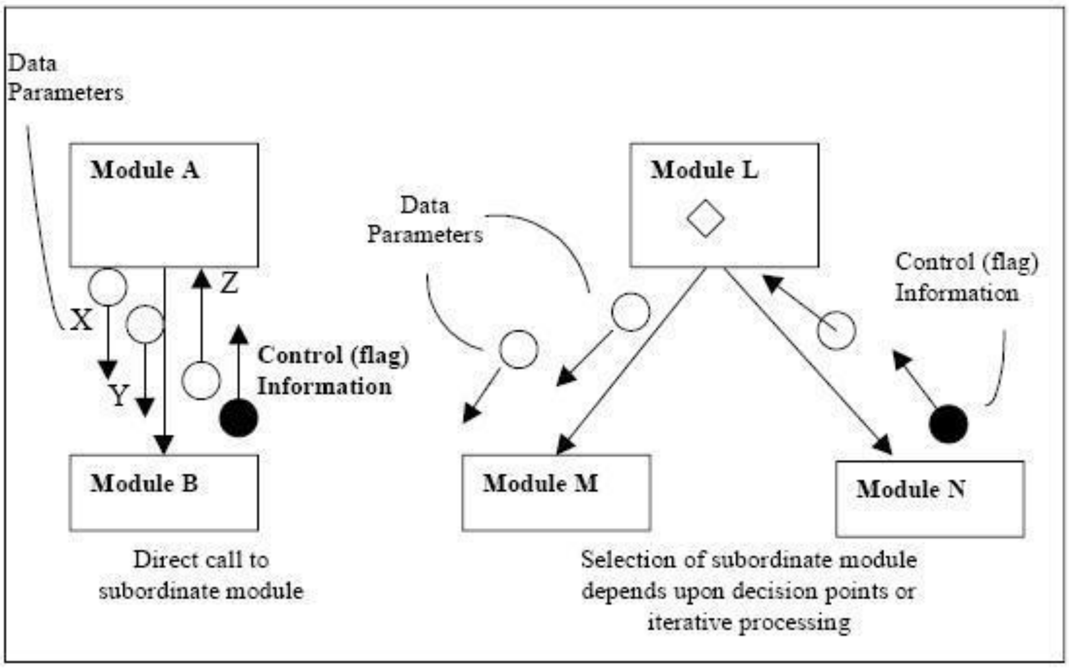

Data Parameters
Module A
X
Y
Z
Control (flag) Information
Module B
Direct call to subordinate module
Data Parameters
Module L
Control (flag) Information
Module M
Module N
Selection of subordinate module depends upon decision points or iterative processing

Fig.2.33

Annotations and data passing in structure charts.

A calling module can interact with more than one subordinate module. Fig. 6.3 also shows module L calling subordinate modules M and N. M is called on the basis of a decision point in L (indicated by the diamond notation), while N is called on the basis of the iterative processing loop (noted by the arc at the start of the calling arrow.

Data passing

When one module calls another, the calling module can send data to the called module so that it can perform the function described in its name. The called module can produce data that are passed back to the calling module.

Two types of data are transmitted. The first, parameter data, are items of data needed in the called module to perform the necessary work. A small arrow with an open circle at the end is used to note the passing of data parameters. In addition, control information (flag data) is also passed. Its purpose is to assist in the control of processing by indicating the occurrence of, say, errors or end-of-conditions. A small arrow with a closed circle indicates the control information. A brief annotation describes the type of information passed. Structure chart is a tool to assist the analyst in developing software that meets the objectives of good software design.

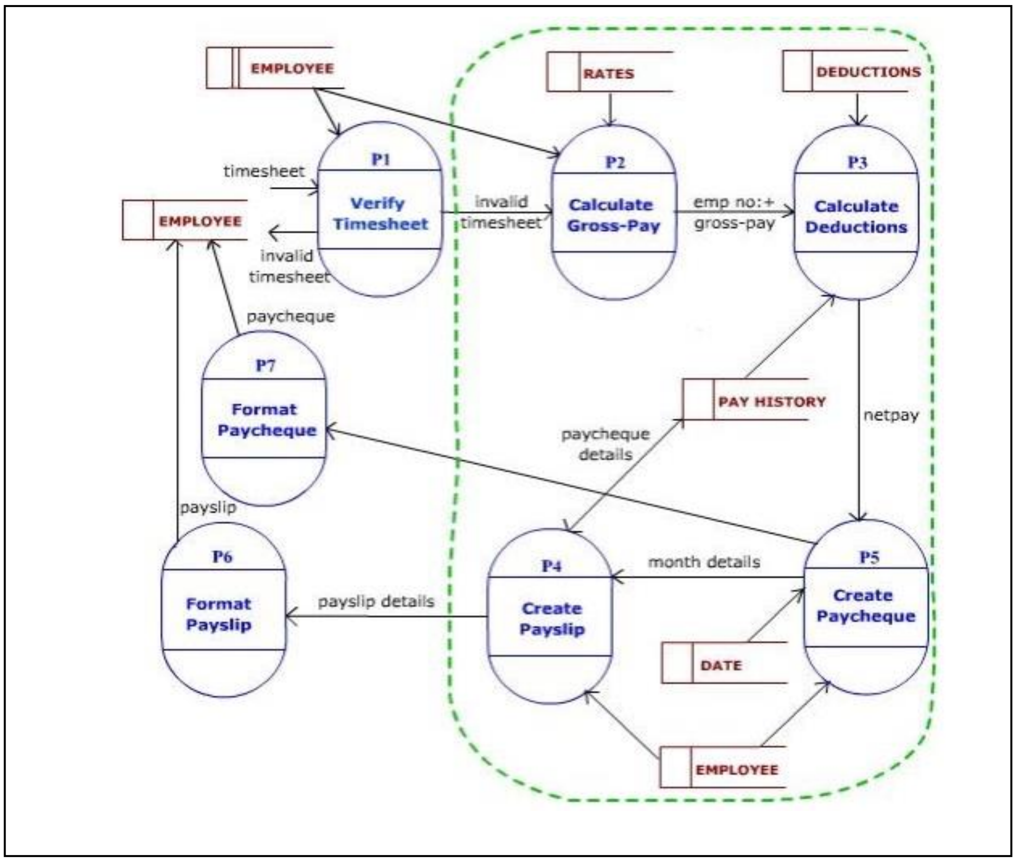
Transform Analysis

Transform analysis is strategy of converting each piece of DFD (may be from level 2 or level 3, etc.) for all the identified transaction centers. Draw a DFD of a transaction type (usually done during analysis phase)

- Find the central functions of the DFD
- Convert the DFD into a first-cut structure chart

- Refine the structure chart
- Verify that the final structure chart meets the requirements of the original DFD Let us understand these steps through a payroll system example:

Identifying the central transform



The central transform is the portion of DFD that contains the essential functions of the system and is independent of the particular implementation of the input and output. One way of identifying central transform (Page-Jones, 1988) is to identify the centre of the DFD by pruning off its afferent and efferent branches. Afferent stream is traced from outside of the DFD to a flow point inside, just before the input is being transformed into some form of output (For example, a format or validation process only refines the input – does not transform it). Similarly an efferent stream is a flow point from where output is formatted for better presentation. The processes between afferent and efferent stream represent the central transform (marked within dotted lines above). In the above example, P1 is an input process, and P6 & P7 are output processes. Central transform processes are P2, P3, P4 & P5 - which transform the given input into some form of output.

Functional vs Object oriented approach

Structured Design:

Structured design is a conceptualization of problem into several well-organized elements of solution. It is basically concerned with the solution design. Benefit of structured design is, it gives better understanding of how the problem is being solved. Structured design also makes it simpler for designer to concentrate on the problem more accurately.

Structured design is mostly based on ‘divide and conquer’ strategy where a problem is broken into several small problems and each small problem is individually solved until the whole problem is solved.

The small pieces of problem are solved by means of solution modules. Structured design emphasis that these modules be well organized in order to achieve precise solution.

These modules are arranged in hierarchy. They communicate with each other. A good structured design always follows some rules for communication among multiple modules, namely -

Cohesion - grouping of all functionally related elements.

Coupling - communication between different modules.

A good structured design has high cohesion and low coupling arrangements.

.

Function Oriented Design

In function-oriented design, the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.

Function oriented design inherits some properties of structured design where divide and conquer methodology is used.

This design mechanism divides the whole system into smaller functions, which provides means of abstraction by concealing the information and their operation.. These functional modules can share information among themselves by means of information passing and using information available globally.

Another characteristic of functions is that when a program calls a function, the function changes the state of the program, which sometimes is not acceptable by other modules. Function oriented design works well where the system state does not matter and program/functions work on input rather than on a state.

Design Process

- The whole system is seen as how data flows in the system by means of data flow diagram.
- DFD depicts how functions changes data and state of entire system.

- The entire system is logically broken down into smaller units known as functions on the basis of their operation in the system.
- Each function is then described at large.

Object Oriented Design

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

Let us see the important concepts of Object Oriented Design:

- **Objects -** All entities involved in the solution design are known as objects. For example, person, banks, company and customers are treated as objects. Every entity has some attributes associated to it and has some methods to perform on the attributes.
- **Classes -** A class is a generalized description of an object. An object is an instance of a class. Class defines all the attributes, which an object can have and methods, which defines the functionality of the object.

  In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.

- **Encapsulation -** In OOD, the attributes (data variables) and methods (operation on the data) are bundled together is called encapsulation. Encapsulation not only bundles important information of an object together, but also restricts access of the data and methods from the outside world. This is called information hiding.
- **Inheritance -** OOD allows similar classes to stack up in hierarchical manner where the lower or sub-classes can import, implement and re-use allowed variables and methods from their immediate super classes. This property of OOD is known as inheritance. This makes it easier to define specific class and to create generalized classes from specific ones.
- **Polymorphism -** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism, which allows a single interface performing tasks for different types. Depending upon how the function is invoked, respective portion of the code gets executed.

MODULE: II

SYSTEM ANALYSIS & DESIGN

References:

[1] https://www.tutorialspoint.com/system_analysis_and_design/system_analysis_and_design_tutorial.pdf

[2] https://pdfs.semanticscholar.org/7ac0/d50dae144f94ef58dcd50bc206d3e4c7b012.pdf

[3] http://download.nos.org/cca/cca1.pdf

[4] http://www.saigontech.edu.vn/faculty/huynq/SAD/Systems_Analysis_Design_UML_5th%20ed.pdf

[5] http://www.monmouthcountyparks.com/Documents/13/Systems%20Analysis%20and%20Design.pdf

[6] http://www.arxen.com/descargas/PulzarCloud/Books/systems-analysis-and-design-with-uml-5th-edition.pdf

[7] https://www.cl.cam.ac.uk/teaching/1112/SWDesign/softwaredesign01.pdf

[8] file:///C:/Users/Guest/Downloads/System%20Development%20Models.pdf

[9] http://www.sts.tu-harburg.de/teaching/ws-99.00/OOA+D/AnalysisDesignImplementation.pdf

[10] http://web.cerritos.edu/dwhitney/SitePages/CIS201/LectureNotesOnTalonNet/Chapter08Lecture.pdf

MODULE: II
SYSTEM ANALYSIS & DESIGN

MCQ Questions:

i. The primary tool used in structured design is a
A) Module
B) Structure chart
C) DFD
D) Program Flow chart

Ii In a DFD external entities are represented by a
A) Rectangle
B) Ellipse
C) Diamond shaped box
D) Circle

iii. A data flow can
A) Only emanate from an external entity
B) Only terminate in an external entity
C) May emanate and terminate in an external entity
D) May either emanate or terminate in an external entity but not both

iv………….. can be defined as most recent and perhaps the most comprehensive technique for solving computer problems.
A) System Analysis
B) System Data
C) System Procedure
D) System Record

v. …………………………. is an important factor of a management information system.
A) System
B) Data
C) Process
D) All

vi. Which are the following is /are the level(s) of documentation?
A) Documentation for management
B) Documentation for user
C) Documentation for data processing department
D) All of the above

vii………………………….. Level supply information to strategic tier for the use of top management.
A) Operational
B) Environmental
C) Competitive
D) Tactical

viii …………… can be defined as data that has been processed into a form that is meaningful to the recipient and is of real or perceived value in current or prospective decisions.
A) System
B) Information
C) Technology
D) Service

ix Use the new system at the same time as the old system to compare the results. This is known as
A) Procedure Writing
B) Simultaneous processing
C) Parallel Operation
D) File Conversion

x. SRS stands for
A) Software requirement specification
B) Software requirement scale
C) System requirement specification
D) System requirement standard

MODULE: II
SYSTEM ANALYSIS & DESIGN

Short Answer Type Questions:

1. What is SRS?
2. What is Data Modeling?
3. Distinguish between coupling and cohesion?
4. What is system analysis?
5. What is physical DFD?
6. What is logical DFD?
7. Distinguish between flowchart and structure chart.
8. What is problem partitioning?
9. What is Decision Tree?
10. What is Decision Table?

MODULE: II
SYSTEM ANALYSIS & DESIGN

Assignment:

Q1.A university registrar's office maintains data about the following entities:

- courses, including number, title, credits, syllabus, and prerequisites;
- course offerings, including course number, year, semester, section number, instructor(s), timings, and classroom;
- students, including student-id, name, and program;
- Instructors, including identification number, name, department, and title.

Further, the enrolment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modelled. Construct an E-R diagram for the registrar's office. Document all assumptions that you make about the mapping constraints.

Q2. Discount policy has following conditions for the customers.
If customer is book store:
Get a trade discount of 25% if orders for 6 or more copies per book title.
If customer is libraries and individuals:
5% allowed on order of 6 to 19 copies per book title.
10% allowed on order of 20 to 49 copies per book title.
15% allowed on order of 50 copies per book title.
Develop process description in decision Table.

Q3. Doctors' Surgery A doctors' surgery consists of five doctors a receptionist and a manager. They need an information system to help them to run the facility. A patient may ring the surgery to make an appointment with a doctor. Each patient nominally has a doctor associated with him or her but they may often opt to see any doctor in the surgery that is available. The receptionist sees which doctors are on duty on which days and offers appointment alternatives from which the patient may choose. If an appointment is not available within a short time and the patient must be seen quickly they are asked to attend an emergency surgery that takes place every evening between 5 and 6 p.m. The appointment can be 5, 10 or 20 minutes long, dependent on the reported reason for seeing the doctor. This reason is recorded on the system. Sometimes patients ring to cancel appointments. Appointments may be made for up to six weeks in advance. Appointments that are more than 3 weeks old are automatically deleted from the system. Some appointments are for a doctor to go and visit a patient at home when the patient cannot come to the surgery. Every day one of the doctors is available for home visits in the afternoon. A record is kept of each patient and the treatments they have received for any ailments they may have had. Here are recorded many details such as allergies, details of which drugs patients have been administered in which quantities and when. Also relevant personal details of each patient are recorded. Typically the doctor who sees a patient will want access to this information before deciding on the relevant treatment to give. When the doctor prescribes treatment, details will be recorded in the patient's record. Repeat prescriptions are automatically produced by the system and are available for collection at the surgery by the patient. At any time a doctor may suspend or cancel the prescriptions. Patients may register with the surgery providing the number registered to each doctor is not above a certain maximum. Sometimes patients die or leave the area. In this case the patient is removed from the system and their details are archived. The manager is responsible for dealing with this aspect.

Construct:

a) A context diagram
b) A level 1 data flow diagram
c) One necessary level 2 data flow diagram

Q4. What is Data Dictionary? Explain Briefly.

Q5. Briefly explain about different coupling and cohesion in system design.

MODULE: II
SYSTEM ANALYSIS & DESIGN

Web/Video links:

[1] https://www.youtube.com/watch?v=F440S-HibGQ

[2] https://www.youtube.com/watch?v=yurSdXxaIFo&list=PL4F47209691234D1D

[3] https://www.youtube.com/watch?v=Z6f9ckEElsU&list=PL0eXJqlwBJ19L1gIo_94r93vvrphmcz8f

[4] https://www.youtube.com/watch?v=X7EJck9XnBE

[5] https://www.youtube.com/watch?v=xu2Sug6ztk8

[6] https://www.youtube.com/watch?v=XCx6ol18544

[7] https://www.youtube.com/watch?v=PfaOQeLnU0w

[8] https://www.youtube.com/watch?v=PF40PTJxn4Q

[9] https://www.youtube.com/watch?v=QpdhBUYk7Kk

[10] https://www.youtube.com/watch?v=a5yWr1hr6QY

MODULE III
AGILE METHODOLOGY

Lecture 13

Agile is one form of software development methodology. Its main focus is on client satisfaction through continuous delivery. The focus of Agile is more on limiting the project scope. An agile project sets a minimum number of requirements and turns them into a deliverable product.

An Agile project sets a minimum number of requirements and turns them into a deliverable product. Agile means what it sounds like: fast and efficient; small; lower cost; fewer features; shorter projects.

In February 2001, the Manifesto for Agile Software Development (The Agile Manifesto, 2001) was created by seventeen people with desires to find alternative approaches to software development. Each of them played a prominent part in the opposition of the prevailing software development processes, which they considered rigid, heavyweight and too focused on documentation. Their response, summarized in the manifesto, clarifies their focus by valuing:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

In Agile literature, Agile methods generally denote a family of methods under the umbrella of the Agile Alliance, including: extreme Programming, Scrum, Dynamic Systems Development Method, Crystal Methods, Feature-Driven Development, Lean Development and Adaptive Software Development. Although differing in specific techniques, these methods have much in common, including short iterative life cycles, quick and frequent feedback from customers, and constant learning. Among them, Scrum and XP/Scrum hybrid are by far the most widely adopted in the past decade. Agile processes bring about a dramatic increase in productivity and quality. This is achieved through a high degree of communication and interaction, short iterative development and a strong sense of team responsibility.

However, there has been some criticism of Agile as well. (Kruchten, 2011) compared Agile methodology to a teenager: very self-conscious, checking constantly its appearance in a mirror, accepting few criticisms, only interested in being with its peers, rejecting en bloc all wisdom from the past, just because it is from the past, adopting fads and new jargon, at times cocky and arrogant.

(Boehm & Turner, 2004) convincingly argue that there is a pragmatic need to balance stability and agility. They analyze the home grounds of Agile and traditional approaches based on application characteristics, management characteristics, technical characteristics, and personnel characteristics. Further, they assert that the choice of traditional or agile methods for a given project is largely contingent on five factors:

1. The size of the systems development project and team
2. The consequences of failure (i.e., criticality)
3. The degree of dynamism or volatility of the environment
4. The competence of personnel
5. Compatibility with the prevailing culture

(Krill, 2013) also pointed out that barriers to Agile adoption include an inability to change an organization's culture, followed by general resistance to change and trying to fit agile into a non-agile framework. The framework for organizational change articulated by Adler and Shenhar (1990) is useful for assessing the effort required to meet these challenges. The biggest concerns about Agile include lack of upfront planning, loss of management control, and management opposition. Other reasons include communication problems between development teams and other areas of the business and problems with the Scrum master.

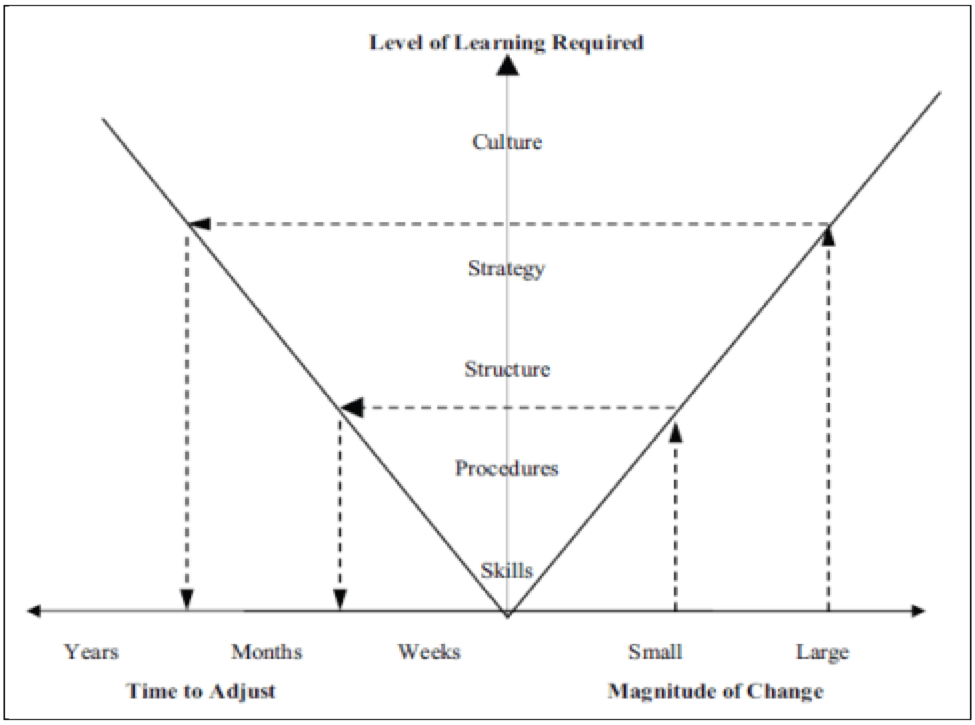


Fig. 3.1- Framework for Organizational Change (Adler & Shenhar, 1990)

Scrum

The founder of Scrum (Schwaber & Sutherland, 1995) described Scrum as a process framework that has been used to manage complex product development since the early 1990s. Scrum is not a process or a technique for building products; rather, it is a framework within which you can employ various processes and techniques.

Scrum is about organizing people and work into short "sprints" of activity, to develop code in short, small chunks, rather than building one big monolithic blob of code that takes forever to build, test and "drop" into the system.

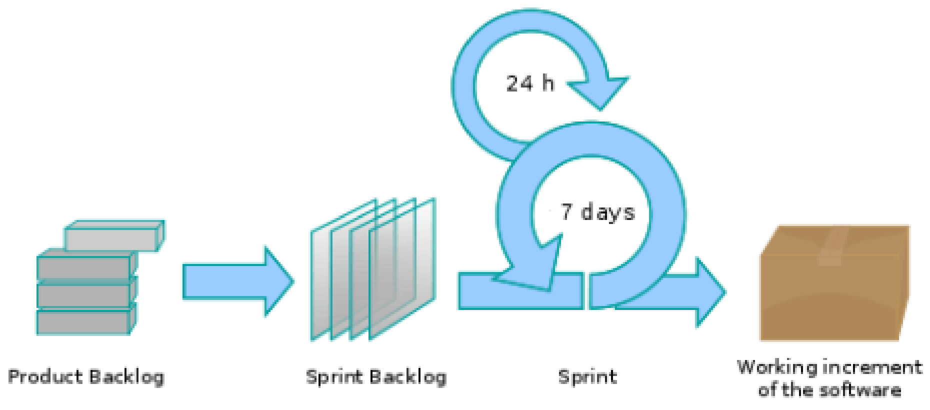


Fig. 3.2- Illustration of Scrum framework

Three distinct roles are identified within the Scrum methodology:

1. The Scrum master, who ensures the process is followed, removes impediments, and protects the Development Team from disruption
2. The Product Owner, who represents the stakeholders and the business
3. The Development Team, a cross-functional, self-organizing team who do the actual analysis, design, implementation, testing, etc.

Scrum is a suitable method to support development process. It is very efficient in creating solutions with the highest business value in the shortest possible time. Due to the daily scrums, any impediments are known to everyone as they occur making it possible to resolve them as quickly as possible. Furthermore, it adds support to prioritize work and closely monitor the progress of a project with little to no overhead.

Lecture 14

Agile Testing – Principles, methods & advantages

Agile testing is a software testing process that follows the principles of agile software development. Agile testing aligns with iterative Development Methodology in which requirements develop gradually from customers and testing teams. The development is aligned with customer requirements.

Agile testing is a continuous process rather than being sequential. The testing begins at the start of the project and there is ongoing integration between testing and development. The common objective of agile development and testing is to achieve a high product quality.

Agile testing vs. Waterfall testing

Agile te sting is adopted while working with agile development approach whereas waterfall testing is used in the waterfall development model. Below are some high-level Differences Between Agile Testing and Waterfall Testing.

Agile testing	**Waterfall testing**
Agile testing is unstructured as compared to the waterfall approach and there is minimal planning.	In the Waterfall model, the testing process is more structured and there is a detailed description of the testing phase.
Agile testing is well suited for small projects.	Waterfall testing can be adopted for all sorts of projects.
As testing begins at the start of the project, errors can be fixed in the middle of the project.	In the waterfall model, the product is tested at the end of the development. For any changes, the project has to start from the beginning.
There is very less documentation required for agile testing.	The testing in the waterfall approach requires elaborate documentation.
In this approach, every iteration has its own testing phase. The regression tests can be run every time new functions or logic are released.	The testing begins only after the completion of the development phase.
In agile testing shippable features of the product are delivered to the customer at the	In this traditional approach, all features developed are delivered altogether after

end of an iteration.	the implementation phase.
Testers and developers work closely in Agile testing.	Testers and developers work separately.
User acceptance is performed at the end of every sprint.	User acceptance can only be performed at the end of the project.
The testers need to establish communication with developers to analyze requirements and planning.	Developers are not involved in analysing requirements and planning process.

Principles of Agile Testing

Testing is continuous: Agile team tests continuously because it is the only way to ensure continuous progress of the product.

Continuous feedback- Agile testing provides feedback on an ongoing basis and this is how your product meets the business needs.

Tests performed by the whole team: In a traditional software development life cycle, only the test team is responsible for testing but in agile testing, the developers and the business analysts also test the application.

Decrease time of feedback response: The business team is involved in each iteration in agile testing & continuous feedback shortens the time of feedback response.

Simplified & clean code: All the defects which are raised by the agile team are fixed within the same iteration and it helps in keeping the code clean and simplified.

Less documentation: Agile teams use a reusable checklist; the team focuses on the test instead of the incidental details.

Test Driven: In agile methods, testing is performed at the time of implementation whereas, in the traditional process, the testing is performed after implementation.

Lecture 15

Agile testing methods

There are various agile testing methods as follows:

Behavior Driven Development (BDD)

Acceptance Test Driven Development (ATDD)

Exploratory Testing

Behaviour Driven Development (BDD)

Behaviour Driven Development (BDD) improves communication amongst project stakeholders so that all members correctly understand each feature before the development process starts. There is continuous example-based communication between developers, testers, and business analysts.

Acceptance Test Driven Development (ATDD)

ATDD focuses on involving team members with different perspectives such as the customer, developer, and tester. Three Amigos meetings are held to formulate acceptance tests incorporating perspectives of the customer, development, and testing. The customer is focused on the problem that is to be solved; the development is focused on how the problem will be solved whereas the testing is focused on what could go wrong. The acceptance tests are a representation of the user's point of view and it describes how the system will function. It also helps to verify that the system functions as it is supposed to. In some instances, acceptance tests are automated.

Exploratory Testing

In this type of testing, the test design and test execution phase go hand in hand. Exploratory testing emphasizes working software over comprehensive documentation. The individuals and interactions are more important than the process and tools. Customer collaboration holds greater value than contract negotiation. Exploratory testing is more adaptable to changes. In this testers identify the functionality of an application by exploring the application. The testers try to learn the application, and design & execute the test plans according to their findings. Advantages of Agile Testing

The benefits of the agile testing approach are as follows:

It saves time and money

Agile testing reduces documentation

It is flexible and highly adaptable to changes

It provides a way for receiving regular feedback from the end user

Better determination of issues through daily meetings

Test Plan for Agile

In agile testing, the test plan is written as well as updated for every release. A test plan in agile includes:

The scope of the testing

Consolidating new functionalities to be tested

Types of testing/Levels of testing

Performance & load testing

Consideration of infrastructure

Risks Plan

Planning of resources

Deliverables & Milestone

QUALITY IN AGILE DEVELOPMENT

The adoption of Agile methodologies and Agile frameworks in software development projects has become common practice over the last few years. This new software development process brings many of the key development areas into focus, and one of the areas that must be considered is Quality Assurance and Test (QA). QA in Agile methodologies will not succeed if addressed as it has been during traditional Waterfall processes, i.e., as a separate activity during the development process. QA is something that must be integrated into the process from the start, rather than be added as an afterthought.

The Agile approach fosters collaborative and proactive behaviors by team members: test, development, product owners, etc. Everyone is responsible for the eventual quality of the product, everyone has ownership.
However, that doesn’t mean that in an Agile project the QA engineer has no role to play! It is actually the opposite. An Agile team benefits tremendously from having QA personnel as team members. The QA engineer will be integral in the definition and clarification of acceptance criteria, as well as in refining the requirements.

Although the product owner (PO) is initially responsible for creating and prioritizing the requirements, as well as writing the initial acceptance criteria for them, the whole team should work together, with the PO, to review and document the acceptance criteria for each of the requirements. The QA Engineer can ensure QA is part of the project DNA. A good Agile technique that helps in documenting acceptance criteria clearly is behavior driven development.

The QA role in Agile methodologies is not restricted to testing the software being built. In fact, the underlying idea behind an Agile team is that none of the members are specialists. That means that QA will eventually help put together automated testing environments, and will also work on exploratory tests, which are usually manual. Development will see increased quality gains if a test driven development (TDD) process is used. This will guarantee a good automated unit test coverage, which is usually, but not exclusively, created by the developers with aid from QA engineers. Additional improvement in quality can be achieved by simple and common techniques, such as peer reviews. There are a couple of Extreme Programming techniques that can also improve the quality of the code, which usually eases testing times.

In order to increase automated testing, it is essentially mandatory that a continuous integration test environment be running. This environment will not only run TDD unit tests, but also extra automated tests, usually devised by QA engineers, such as service layer testing, acceptance criteria validation, and integrated testing.

Bottom-line: QA in Agile methodologies is not what it has been known to be in the past; there must be more integration, more collaboration, more pro activity, and essentially more team work from the start to ensure a successful and high quality deliverable.

At Daitan Group we foster agile techniques and Agile culture, with a belief in continuous never ending improvement.

MODULE: III

AGILE METHODOLOGY

Reference:

1. https://www.tutorialspoint.com/agile/agile_tutorial.pdf
2. http://agilehandbook.com/agile-handbook.pdf
3. https://matthewrenze.com/presentations/introduction-to-agile-and-scrum.pdf
4. http://www.danube.com/docs/Intro_to_Agile.pdf
5. http://www-public.imtbs-tsp.eu/~gibson/Teaching/Agile/AgileMethods.pdf

MODULE: III

AGILE METHODOLOGY

Multiple Choice Questions

1. When is Acceptance Testing performed in Agile development?

a. On request of customer
b. After system is ready
c. At the end of each iteration
d. Daily

2. In agile development, lengthy documentation is created.

a. True
b. False

3. What are the skills that are required by the Agile Tester?

a. Domain knowledge
b. Keen to learn and adopt new technology
c. Effective communicator who maintains good relationship with development team
d. All the above

4. Agile is a ______.

a. Sequential
b. Iterative
c. Incremental
d. Both b & c

5. Agile Development Testing is treated as a separate phase.

a. True
b. False

MODULE: III

AGILE METHODOLOGY

Short question

1. What is Agile Testing?
2. Define the roles in Scrum?
3. Explain the difference between traditional waterfall model and Agile testing?
4. Explain the Iterative and Incremental Development in Agile?
5. What qualities should a good Agile tester have?

6. Name some Agile quality strategies.

MODULE: III

AGILE METHODOLOGY

Assignment:

1. What is Agile Testing? What is the difference between burn-up and burn-down chart?
2. Define the roles in Scrum?
3. Explain Velocity in Agile?
4. Explain the difference between traditional Waterfall model and Agile testing? Explain the Iterative and Incremental Development in Agile?
5. What qualities should a good Agile tester have?

MODULE: III

AGILE METHODOLOGY

Web/Video links:

1. https://www.youtube.com/watch?v=NpCEjtKAa20
2. https://www.youtube.com/watch?v=cWQrJ7DGgXA&list=PLWPirh4EWFpF8LjfSxHPuTnyNhqhe4XDc
3. https://www.youtube.com/watch?v=9TycLR0TqFA
4. https://www.youtube.com/watch?v=PSorv6xE79c
5. https://www.youtube.com/watch?v=gLd4NZLVrFw

MODULE IV

UNIFIED MODELLING LANGUAGE

Lecture16

Unified Modeling Language:

Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artefacts of a software-intensive system. It offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components

Introduction

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of object-oriented software engineering. UML includes a set of graphic notation techniques to create visual models of object-oriented software systems. UML combines techniques from data modeling, business modeling, object modeling, and component modeling and can be used throughout the software development life-cycle and across different implementation technologies.

Modeling

There is a difference between a UML model and the set of diagrams of a system. A diagram is a partial graphic representation of a system's model. The model also contains documentation that drives the model elements and diagrams (such as written use cases).

UML diagrams represent two different views of a system model:

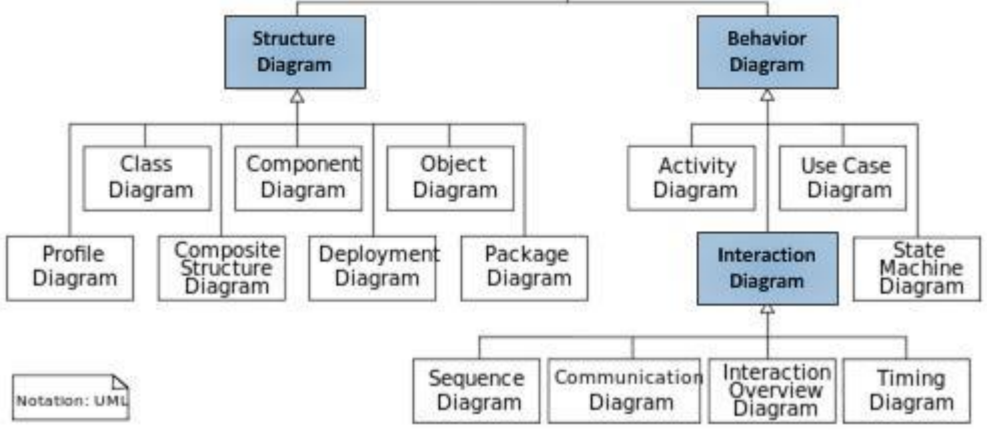
Static (or structural) view

This view emphasizes the static structure of the system using objects, attributes, operations, and relationships. Ex: Class diagram, Composite Structure diagram.

Dynamic (or behavioral) view

This view emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. Ex: Sequence diagram, Activity diagram, State Machine diagram.

Diagrams Overview: UML 2.2 has 14 types of diagrams divided into multiple categories as shown in the figure below.



Class Diagram

Describes the structure of a system by showing the system's classes, their attributes, and the relationships among the classes**.**

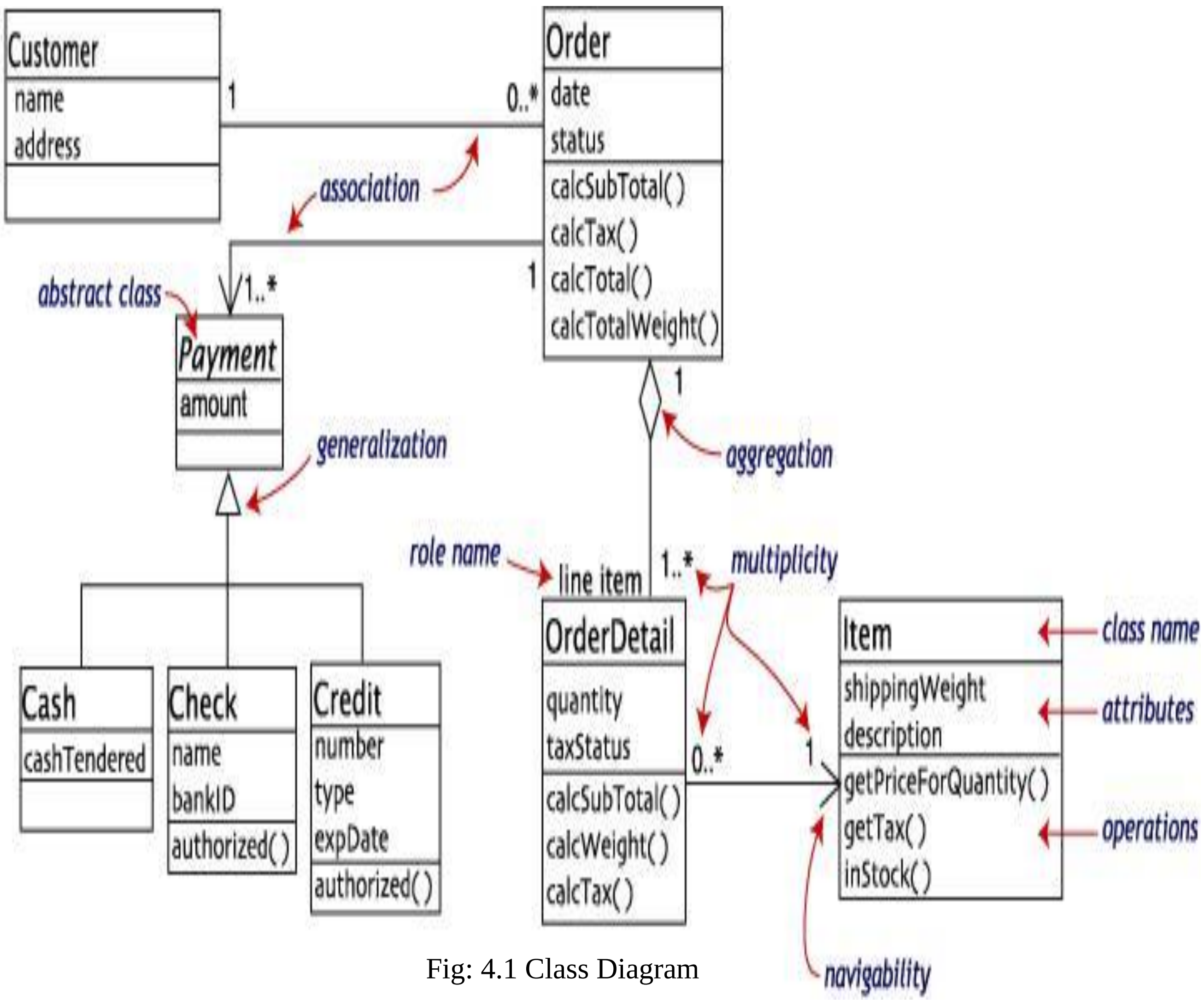


Fig: 4.1 Class Diagram

Interaction Diagram

These diagrams are a subset of behavior diagrams, emphasizing the flow of control and data among the things in the system being modeled.

1. **Communication Diagram**

Shows the interactions between objects or parts in terms of sequenced messages. They represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system.

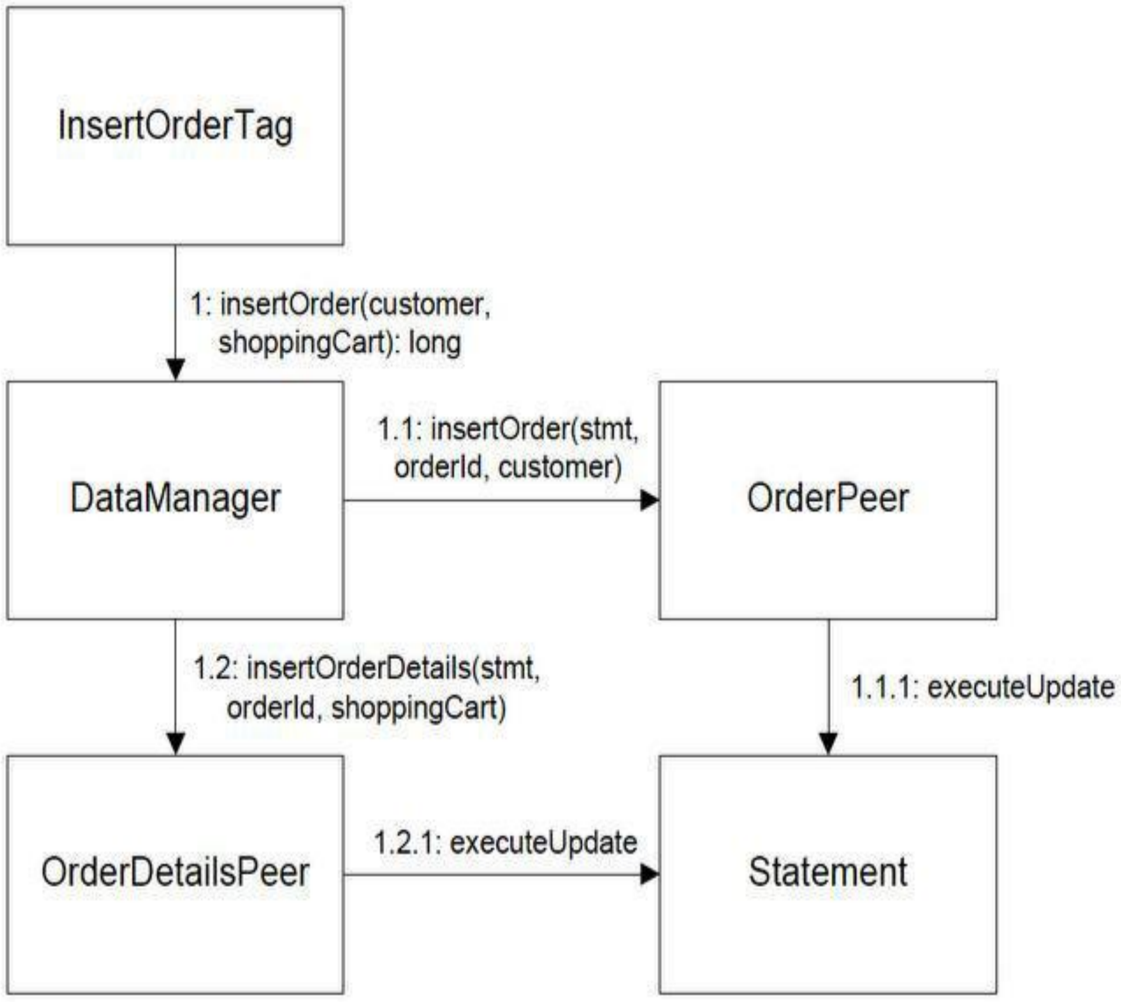

InsertOrderTag
1: insertOrder(customer, shoppingCart): long
DataManager
1.1: insertOrder(stmt, orderId, customer)
OrderPeer
1.2: insertOrderDetails(stmt, orderId, shoppingCart)
1.1.1: executeUpdate
OrderDetailsPeer
1.2.1: executeUpdate
Statement

Fig: 4.2 Communication Diagram

Lecture 17

2. Interaction Overview Diagram

Provides an overview in which the nodes represent communication diagrams. They are activity diagrams in which every node, instead of being an activity, is a rectangular frame containing an interaction diagram (i.e., a communication, interaction overview, sequence, or UML timing diagram).

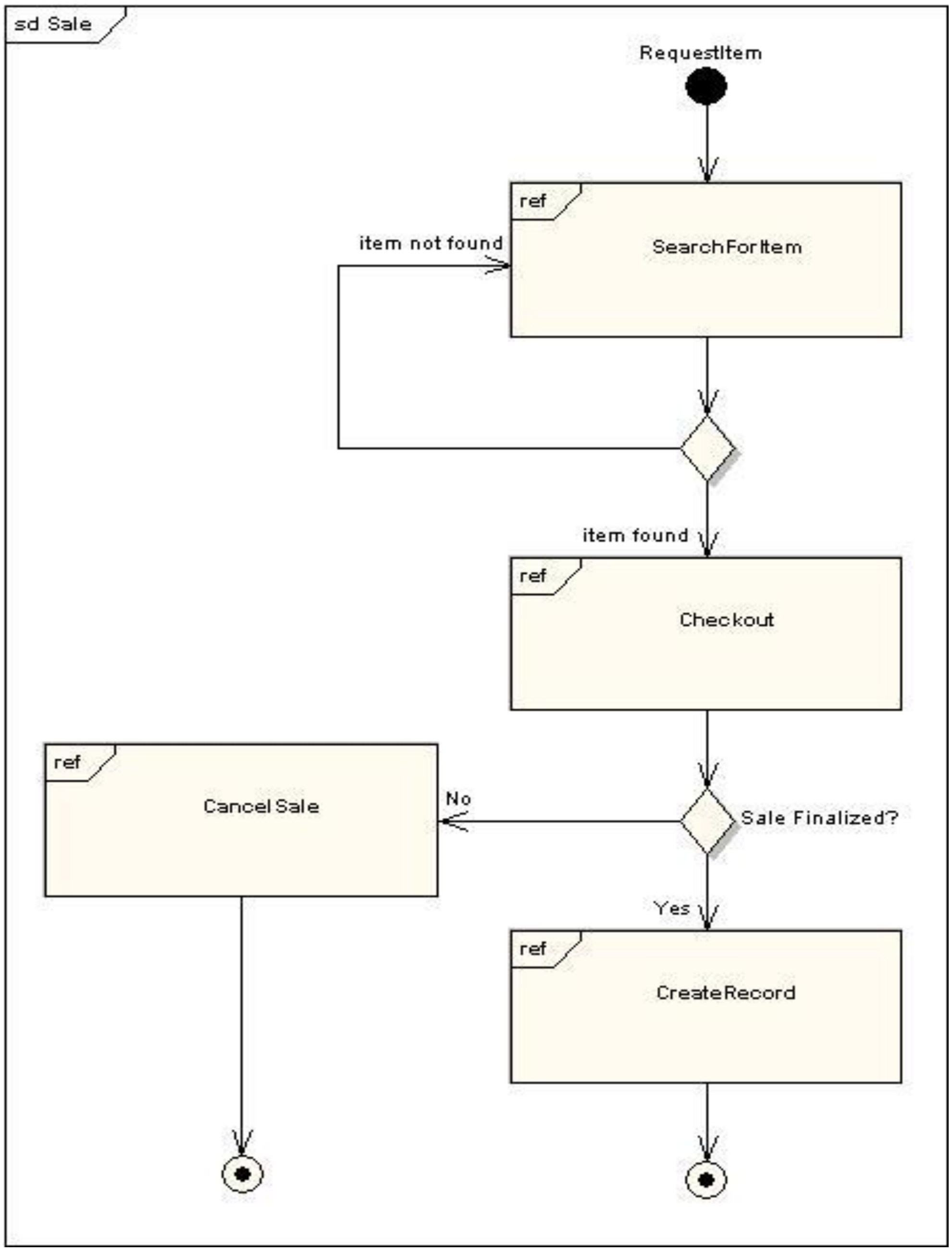


Fig: 4.3 Interaction Diagram

Sequence Diagram

Shows how objects communicate with each other in terms of a sequence of messages. Also indicates the life spans of objects relative to those messages.

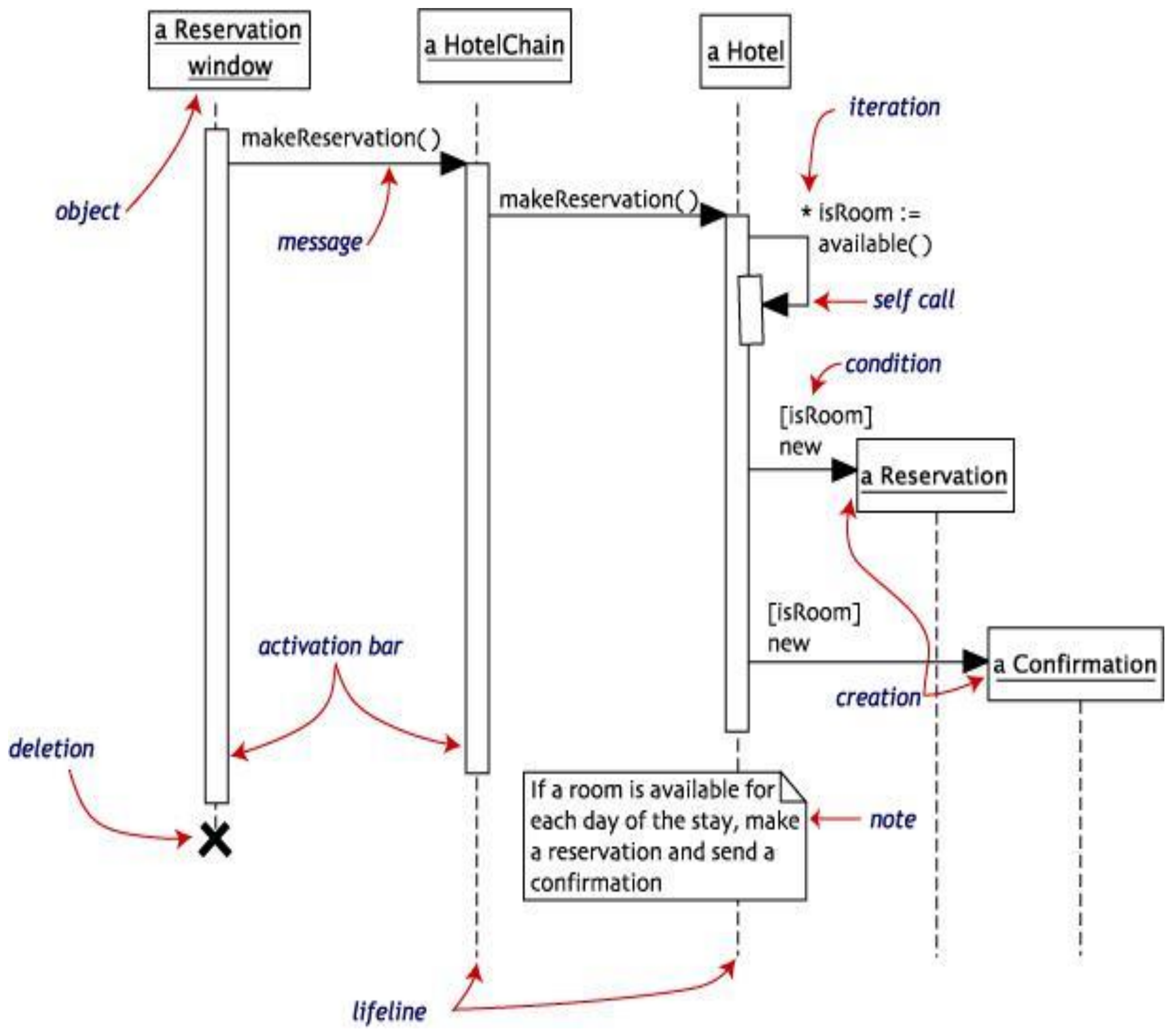


Fig: 4.4 Sequence Diagram

Lecture 18

State Chart Diagram

Describes the states and state transitions of the system.

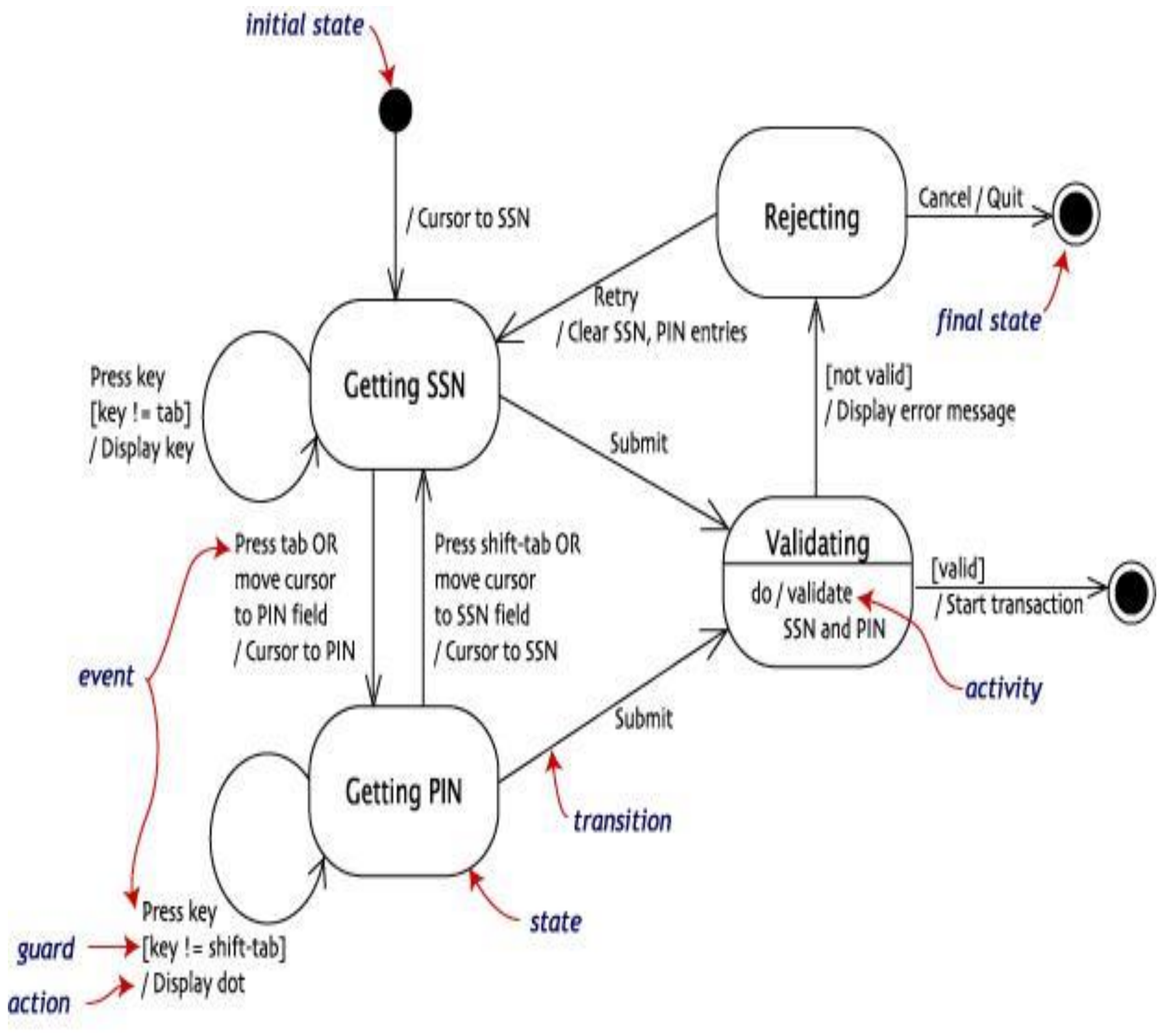


Fig: 4.5 State Chart Diagram

Activity Diagram

Describes the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

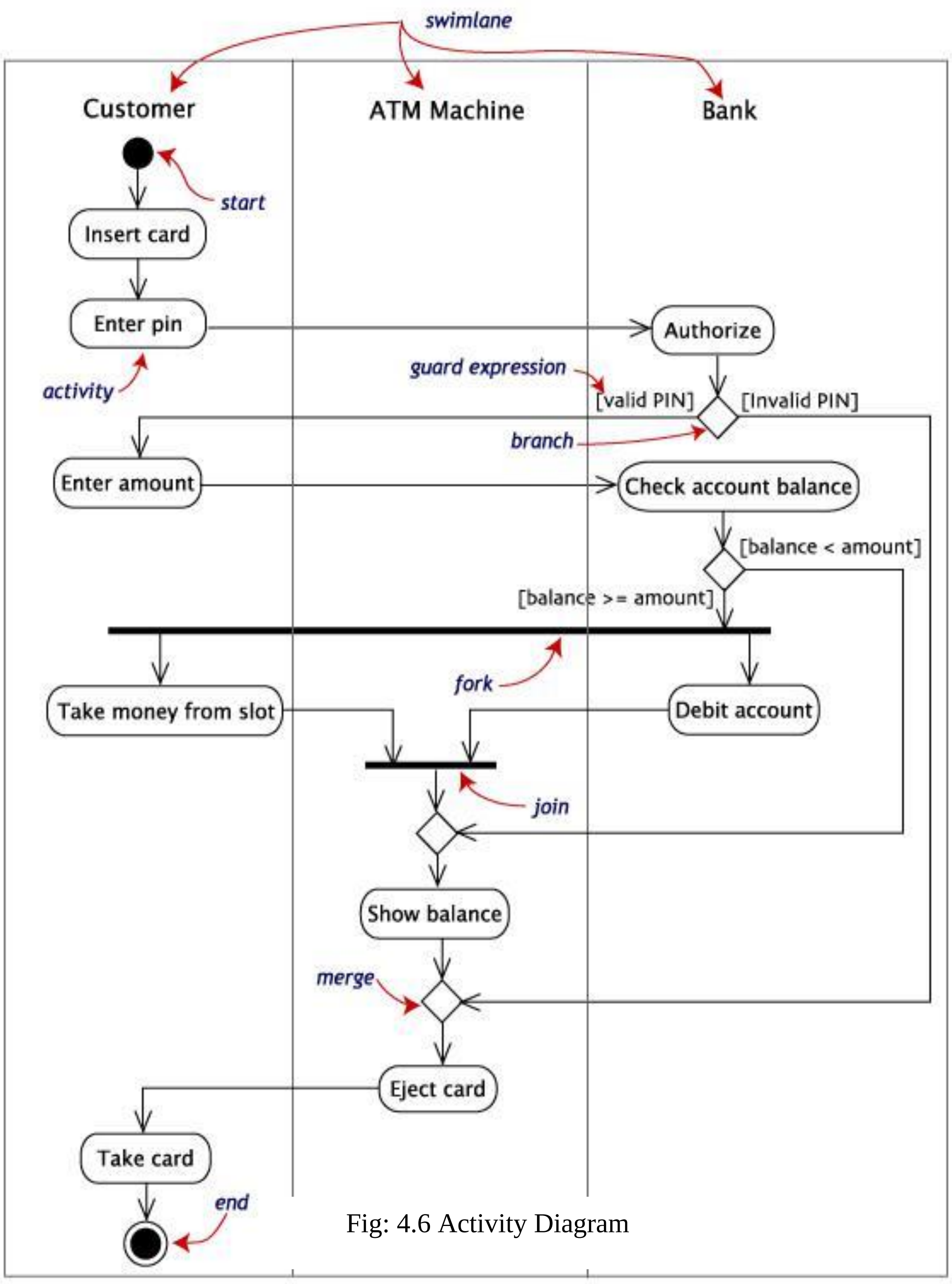


Fig: 4.6 Activity Diagram

Lecture 19

Implementation Diagram

4. Deployment Diagram

Describes the hardware used in system implementations and the execution environments and artefacts deployed on the hardware.

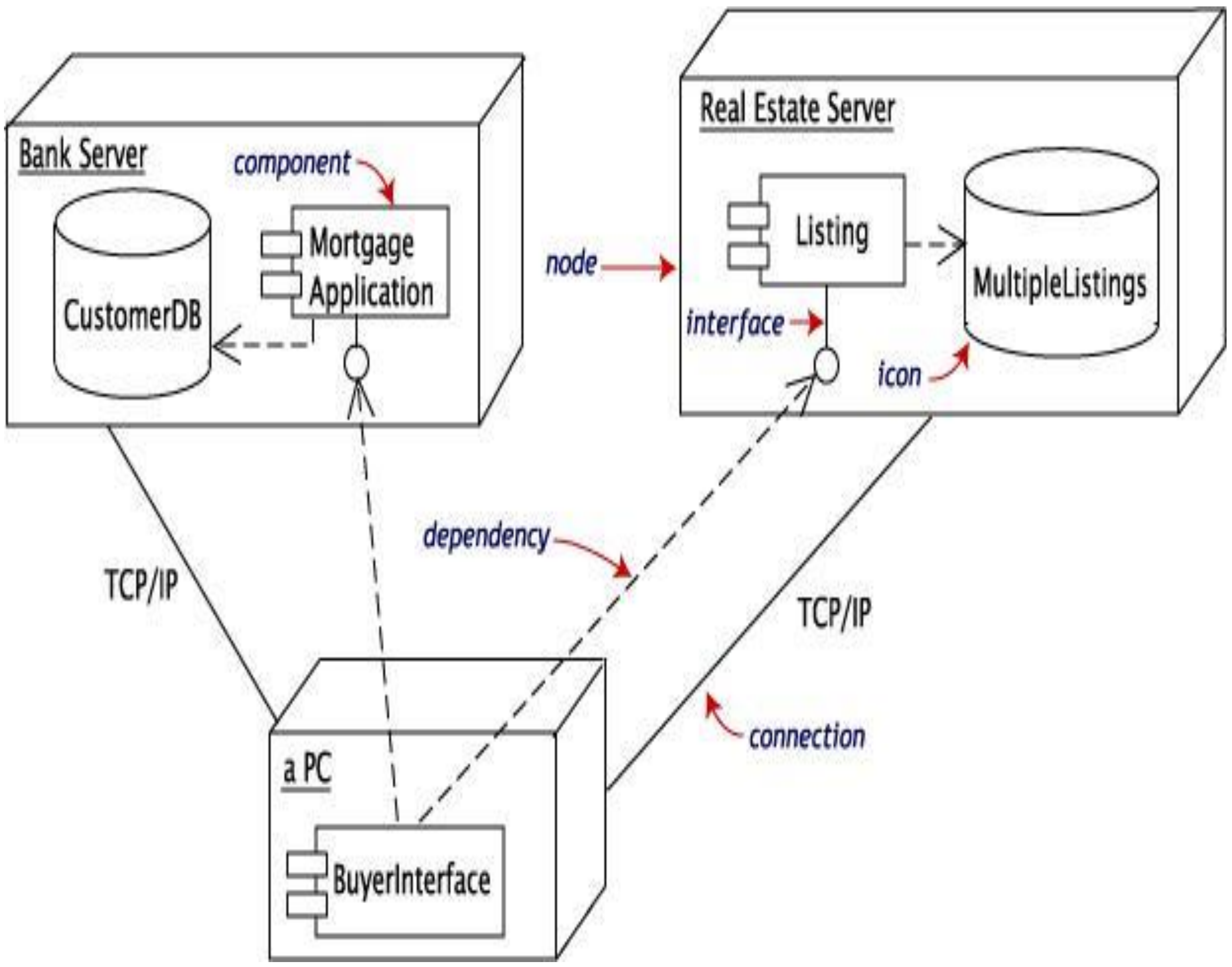
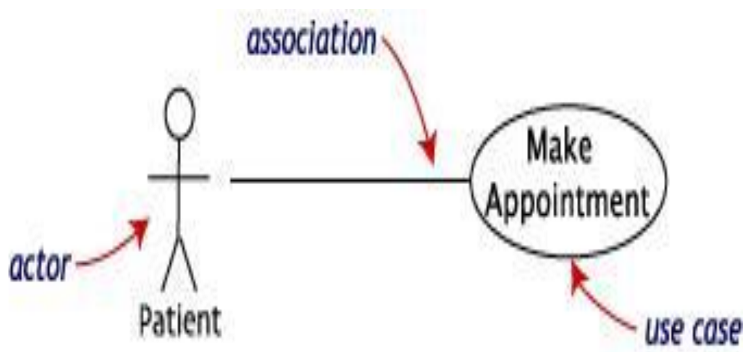


Fig: 4.7 Communication Diagram

Use Case Diagram

Describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases.



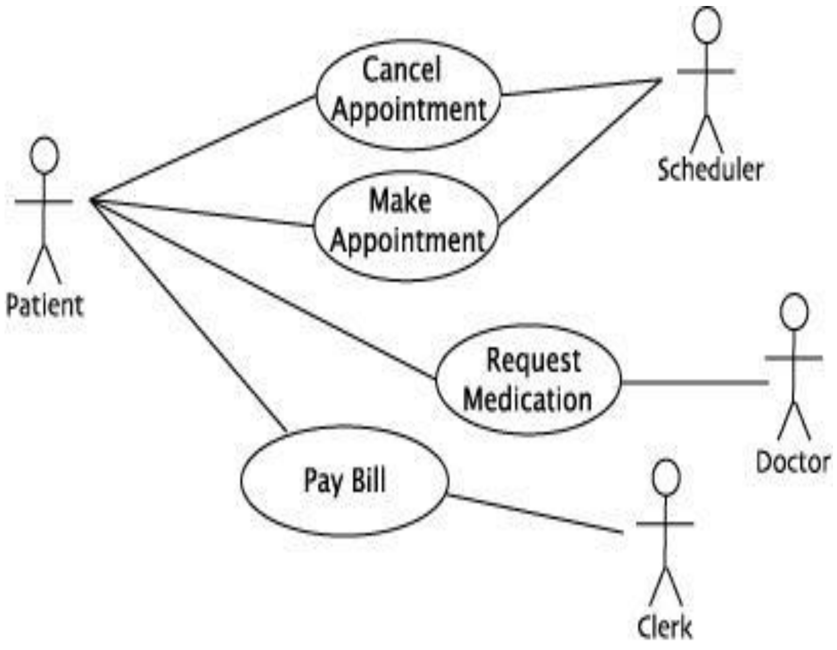


Fig: 4.8 Use Case Diagrams

MODULE: IV

UNIFIED MODELING LANGUAGE

References:

[1] https://www.tutorialspoint.com/uml/uml_tutorial.pdf

[2] https://people.eecs.ku.edu/~hossein/Teaching/Fa13/810/Readings/UML-diagrams.pdf

[3] https://courses.cs.washington.edu/courses/cse403/11sp/lectures/lecture08-uml1.pdf

[4] http://www.temida.si/~bojan/IPIT_2014/literatura/UML_Reference_Manual.pdf

[5] http://www.peter-lo.com/Teaching/U08182/Types%20of%20UML%20Diagrams.pdf

[6] http://www.nyu.edu/classes/jcf/g22.2440-001_sp09/handouts/UMLBasics.pdf

[7] http://ima.udg.edu/~sellares/EINF-ES2/uml2_diagrams.pdf

[8] http://stlab.istc.cnr.it/documents/swe0910/UMLBasics.pdf

[9] https://www.oasis-open.org/committees/download.php/23589/umlnotetemp.xml.pdf

[10] http://homepage.divms.uiowa.edu/~tinelli/classes/181/Spring08/Notes/04-UML-intro.pdf

MODULE: IV
UNIFIED MODELING LANGUAGE

MCQ Questions:

i. A UML diagram includes which of the following?

A. Class name
B. List of attributes
C. List of operation
D. All of the above.

ii. An object can have which of the following multiplicities?

A. Zero
B. One
C. More than one
D. All of the above

iii. Which of the following applies to a class rather than an object?

A. Query
B. Update
C. Scope
D. Constructor

iv. The term Incomplete for a UML has the same meaning as which of the following for an EER diagram?

A. Overlapping rule
B. Disjoint rule
C. Total specialization rule
D. Partial specialization rule

v. What does a simple name in UML Class and objects consists of ?

A. Letters
B. Digits
C. Punctuation Characters
D. All of the mentioned

vi. A Class consists of which of these abstractions?

A. Set of objects
B. Operations
C. Attributes
D. All of the above

vii A class is divided into which of these compartments ?

A. Name compartment
B. Attribute compartment
C. Operation Compartment
D. All of the above

viii An attribute is a data item held by which of the following ?

A. Class
B. Object
C. All of the above
D. None of the above

ix Which of these are part of class operation specification format ?

A. Name
B. Parameter list
C. Return type list
D. All of the above

x An operation can be described as?

A. Object behavior
B. Class behavior
C. Functions
D. Object & Class behavior

MODULE: IV
UNIFIED MODELING LANGUAGE

Short Answer Type Questions:

1. What is unified modeling language?
2. What do you mean object?
3. Give the diagram of sequence diagram.
4. What is modeling?
5. What is state chart diagram?
6. Explain activity diagram.
7. What is class diagram?

MODULE: IV
UNIFIED MODELING LANGUAGE

Assignment:

1. What is the difference between sequence diagram and collaboration diagram?
2. Short note on use case diagram?
3. Explain how Sequence Diagram differs from Component Diagram?
4. What is UML? How it is useful?
5. Explain about interaction diagram with proper diagram.
6. What is Activity diagram? Explain with help of diagram.
7. What is interaction diagram? Explain with help of diagram.
8. Explain deployment diagram with proper diagram.

MODULE: IV

UNIFIED MODELING LANGUAGE

Web/Video links:

[1] https://www.youtube.com/watch?v=gUEizau0UQ&list=PLWPirh4EWFpF9Gbnu4_DdF4ITHSN6MSsk

[2] https://www.youtube.com/watch?v=F5UJkENKc50

[3] https://www.youtube.com/watch?v=3cmzqZzwNDM

[4] https://www.youtube.com/watch?v=RRXe1omEGWQ&list=PLD4EF3E3AD055F3C7

[5] https://www.youtube.com/watch?v=4WDbte6cPa8

[6] https://www.youtube.com/watch?v=xiUFTLIU-lw

[7] https://www.youtube.com/watch?v=UI6lqHOVHic

[8] https://www.youtube.com/watch?v=pCK6prSq8aw

[9] https://www.youtube.com/watch?v=6oz8MKShCVE

[10] https://www.youtube.com/watch?v=qxWhw6zAgdI

MODULE: V

CODING AND DOCUMENTATION

Lecture: 20

Structured Programming

Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

Structured programming is a logical programming method that is considered a precursor to object-oriented programming (OOP). Structured programming facilitates program understanding and modification and has a top-down design approach, where a system is divided into compositional subsystems.

Structured Programming is designed which focuses on process/ logical structure and then data required for that process. Object Oriented Programming is designed which focuses on data. Object Oriented Programming supports inheritance, encapsulation, abstraction, polymorphism, etc.

C is called a structured programming language because to solve a large problem, C programming language divides the problem into smaller modules called functions or procedures each of which handles a particular responsibility. The program which solves the entire problem is a collection of such functions.

Structured and unstructured programming. Unstructured programming predates structured approaches, and can be found in languages such as BASIC, FORTRAN and machine code. It involves a single code block, and makes extensive use of GOTO in order to control the flow of the execution and add complexity to the program

Modular Programming

Modular programming is the process of subdividing a computer program into separate sub-programs. A module is a separate software component. It can often be used in a variety of applications and functions with other components of the system.

As programs grow larger and larger, they should be split into sections, or modules. C++ allows programs to be split into multiple files, compiled separately, and then combined (linked) to form a single program.

The benefits of modular programming are:

- Efficient Program Development. Programs can be developed more quickly with the modular approach since small subprograms are easier to understand, design, and test than large programs.
- Multiple Uses of Subprograms.

- Ease of Debugging and Modifying.

  Modular design, or "modularity in design", is a design approach that subdivides a system into smaller parts called modules or skids that can be independently created and then used in different systems.

Lecture: 21

Module Relationship

Coupling and Cohesion

In software engineering, coupling is the degree of interdependence between software modules; a measure of how closely connected two routines or modules are; the strength of the relationships between modules. Coupling is usually contrasted with cohesion. Low coupling often correlates with high cohesion, and vice versa.

Cohesion is the indication of the relationship within module. Coupling is the indication of the relationships between modules. Coupling shows the relative independence among the modules. Cohesion is a degree (quality) to which a component / module focuses on the single thing.

Definition of Data Coupling. Data coupling occurs between two modules when data are passed by parameters using a simple argument list and every item in the list is used. An example of data coupling is illustrated as a module which retrieves customer address using customer id.

In computer programming, cohesion refers to the degree to which the elements inside a module belong together. In one sense, it is a measure of the strength of relationship between the methods and data of a class and some unifying purpose or concept served by that class.

Coupling and cohesion are two often misunderstood terms in software engineering. These are terms that are used to indicate the qualitative analysis of the modularity in a system, and they help us to identify and measure the design complexity of object oriented systems.

Cohesion refers to the degree to which the elements inside a module belong together. In one sense, it is a measure of the strength of relationship between the methods and data of a class and some unifying purpose or concept served by that class.

In software engineering, coupling is the degree of interdependence between software modules; a measure of how closely connected two routines or modules are; the strength of the relationships between modules. Coupling is usually contrasted with cohesion. Low coupling often correlates with high cohesion, and vice versa.

Cohesion is the indication of the relationship within module. Coupling is the indication of the relationships between modules. Coupling shows the relative independence among the modules. Cohesion is a degree (quality) to which a component / module focuses on the single thing.

A communication cohesive module is one which performs several functions on the same input or output data. For example, obtain author, title, or price of book from bibliographic record, based on a passed flag.

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

Types of Cohesion

- Functional cohesion (Most Required)
- Sequential cohesion.
- Communicational cohesion.
- Procedural cohesion.
- Temporal cohesion.
- Logical cohesion.
- Coincidental cohesion (Least Required)

Cohesion refers to the degree to which the elements inside a module belong together. In one sense, it is a measure of the strength of relationship between the methods and data of a class and some unifying purpose or concept served by that class.

Adhesion vs. Cohesion.--The difference between them is that adhesion refers to the clinging of unlike molecules and cohesion refers to the clinging of like molecules. Adhesion is the mutual attraction between unlike molecules that causes them to cling to one another.

Lecture: 22

OO Programming

Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.

Encapsulation, inheritance, and polymorphism are usually given as the three fundamental principles of object-oriented languages (OOLs) and object-oriented methodology.

Object Oriented programming is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation etc.

There are four pillars of oops as follows.

- Abstraction.
- Encapsulation.
- Polymorphism.
- Inheritance.

Object oriented programming aims to implement real world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of code can access this data except that function.

The three major principles of OOP are;

- Encapsulation – this is concerned with hiding the implementation details and only exposing the methods.
- Inheritance – this is concerned with the relationship between classes. ...
- Polymorphism – this is concerned with having a single form but many different implementation ways.

FEATURES OF OOP:

- Object.
- Class.
- Data Hiding and Encapsulation.
- Dynamic Binding.
- Message Passing.
- Inheritance.
- Polymorphism.

OOP is the object oriented programming approach to any programming language. By using these features you can easily reuse your program or part of your program.OOP provides you a power to do programming in a very effective manner. Learn OOP concept and implement it in your daily programs.

C is not an 'object-oriented' language. It has no concept of classes, objects, polymorphism, inheritance. The confusion may be that C can be used to implement object oriented concepts like polymorphism, encapsulation, etc. which may lead your friend to believe that C is object oriented.

The major difference between C and C++ is that C is a procedural programming language and does not support classes and objects, while C++ is a combination of both procedural and object oriented programming language; therefore C++ can be called a hybrid language.

Four Pillars of Object Oriented Programming

- 1. Abstraction. Abstraction is a process of exposing essential feature of an entity while hiding other irrelevant detail.
- 2. Encapsulation. We have to take in consideration that Encapsulation is somehow related to Data Hiding.
- 3. Inheritance.
- Polymorphism.

Data/Information Hiding -

Information hiding is the principle of segregation of the design decisions in a computer program that are most likely to change, thus protecting other parts of the program from extensive modification if the design decision is changed.

Information hiding is written another way, information hiding is the ability to prevent certain aspects of a class or software component from being accessible to its clients, using either programming language features (like private variables) or an explicit exporting policy.

In programming, the process of hiding details of an object or function. Information hiding is a powerful programming technique because it reduces complexity. One of the chief mechanisms for hiding information is encapsulation -- combining elements to create a larger entity.

Data hiding is a software development technique specifically used in object-oriented programming (OOP) to hide internal object details (data members). Data hiding ensures exclusive data access to class members and protects object integrity by preventing unintended or intended changes.

Data hiding and encapsulation both are the important concept of object oriented programming. Encapsulation means wrapping the implementation of data member and methods inside a class. Data Hiding means protecting the members of a class from an illegal or unauthorized access.

In computer science, information hiding is the principle of segregation of the design decisions in a computer program that are most likely to change, thus protecting other parts of the program from extensive modification if the design decision is changed.

How can a Class enforce data- hiding, abstraction and encapsulation. And a Class groups its members into three sections: as private, protected and public. The members like private and protected remains hidden from the outside world. Thus through these private and protected members a class enforces data hiding.

Data hiding is a software development technique specifically used in object-oriented programming (OOP) to hide internal object details (data members). Data hiding ensures exclusive data access to class members and protects object integrity by preventing unintended or intended changes.

Encapsulation is most often achieved through information hiding (not just data hiding), which is the process of hiding all the secrets of an object that do not contribute to its essential characteristics; typically, the structure of an object is hidden, as well as the implementation of its methods.

Encapsulation - The key advantage of using an Object Oriented Programming language like Java is that it provides your code - security, flexibility and its easy maintainability through encapsulation. ... Encapsulation is also useful in hiding the data(instance variables) of a class from an illegal direct access.

Lecture: 23

Reuse –

A definition of software reuse is the process of creating software systems from predefined software components. The advantage of software reuse: The systematic development of reusable components. The systematic reuse of these components as building blocks to create new systems.

Reuse-oriented model or reuse-oriented development (ROD) - The reuse-oriented model can reduce the overall cost of software development compared with more tedious manual methods. It can also save time because each phase of the process builds on the previous phase which has already been refined.

In software engineering, reusability is the use of existing assets in some form within the software product development process; these assets are products and by-products of the software development life cycle and include code, software components, test suites, designs and documentation.

Reusable components are simply pre-built pieces of programming code designed to perform a specific function. Beans are "capsules" of code, each designed for a specific purpose.

Software evolution is the term used in software engineering (specifically software maintenance) to refer to the process of developing software initially, then repeatedly updating it for various reasons.

Code reuse, also called software reuse, is the use of existing software, or software knowledge, to build new software, following the reusability principles.

There are some fundamental activities that are common to all software processes:

- Software specification. In this activity the functionality of the software and constraints on its operation must be defined.
- Software design and implementation.
- Software validation.
- Software evolution.

Concept of reusability in C++ Using Inheritance. C++ strongly supports the concept of reusability. The C++ classes can be reused in several ways. The old class is referred to as the base class and the new one is called the derived class or subclass.

System Documentation

System documentation: The collection of documents that describes the requirements, capabilities, limitations, design, operation, and maintenance of a system, such as a communications, computing, or information processing system.

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

In computer hardware and software product development, documentation is the information that describes the product to its users. It consists of the product technical manuals and online information (including online versions of the technical manuals and help facility descriptions).

The Importance of Documentation in Software Development - The presence of documentation helps keep track of all aspects of an application and it improves on the quality of a software product. Its main focuses are development, maintenance and knowledge transfer to other developers.

Documentation is the written and retained record of employment events. Documentation is made up of government and legally mandated elements, documents required by company policy and practice, documents suggested by best Human Resources practices, and formal and informal recordkeeping about employment events.

Documentation is a set of documents provided on paper, or online, or on digital or analog media, such as audio tape or CDs. Examples are user guides, white papers, on-line help, quick-reference guides. It is becoming less common to see paper (hard-copy) documentation.

Lecture: 24

Testing

Levels of Testing

There are generally four recognized levels of testing: Unit/component testing, integration testing, system testing, and acceptance testing. Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test.

Functional testing types include:

- Unit testing.
- Integration testing.
- System testing.
- Sanity testing.
- Smoke testing.
- Interface testing.
- Regression testing.
- Beta/Acceptance testing.

Below are the phases of STLC:

- Requirements phase.
- Planning Phase.
- Analysis phase.
- Design Phase.
- Implementation Phase.
- Execution Phase.
- Conclusion Phase.
- Closure Phase.

A level of the software testing process where individual units of software are tested. The purpose is to validate that each unit of the software performs as designed. A level of the software testing process where a complete, integrated system is tested.

The Four Levels of Software Testing

- Unit Testing. During this first round of testing, the program is submitted to assessments that focus on specific units or components of the software to determine whether each one is fully functional.

- Integration Testing.
- System Testing.
- Acceptance Testing.

The Requirements Traceability Matrix (RTM) is a document that links requirements throughout the validation process. The purpose of the Requirements Traceability Matrix is to ensure that all requirements defined for a system are tested in the test protocols.

The following examples are different types of testing that should be considered during System testing:

- Graphical user interface testing.
- Usability testing.
- Software performance testing.
- Compatibility testing.
- Exception handling.
- Load testing.
- Volume testing.
- Stress testing.

There are three main stages of testing.

- The drugs are tested using computer models and human cells grown in the laboratory.
- Drugs that pass the first stage are tested on animals.
- Drugs that have passed animal tests are used in clinical trials.

Software Testing Life Cycle (STLC) is the testing process which is executed in systematic and planned manner. In STLC process, different activities are carried out to improve the quality of the product. Let's quickly see what all stages are involved in typical Software Testing Life Cycle (STLC)

SDLC is Software Development Life Cycle; it is a systematic approach to develop software. The process of testing software in a well planned and systematic way is known as software testing life cycle (STLC). Requirements Analysis is done is this phase, software requirements are reviewed by test team.

In the types of functional testing following testing types should be cover:

- Unit Testing.
- Smoke testing.
- Sanity testing.
- Integration Testing.
- Interface Testing.
- System Testing.

- Regression Testing.
- UAT.

White box testing is the software testing method in which internal structure is being known to tester who is going to test the software. This type of testing is carried out by testers. ... Programming Knowledge is not required to carry out Black Box Testing. Programming Knowledge is required to carry out White Box Testing.

A level of the software testing process where individual units of a software are tested. The purpose is to validate that each unit of the software performs as designed. A level of the software testing process where a complete, integrated system is tested.

Different types of tests (GUI testing, Functional testing, Regression testing, Smoke testing, load testing, stress testing, security testing, stress testing, ad-hoc testing etc.,) are carried out to complete system testing.

SMOKE TESTING, also known as “Build Verification Testing”, is a type of software testing that comprises of a non-exhaustive set of tests that aim at ensuring that the most important functions work. The result of this testing is used to decide if a build is stable enough to proceed with further testing.

Lecture: 25

Integration Testing

INTEGRATION TESTING is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

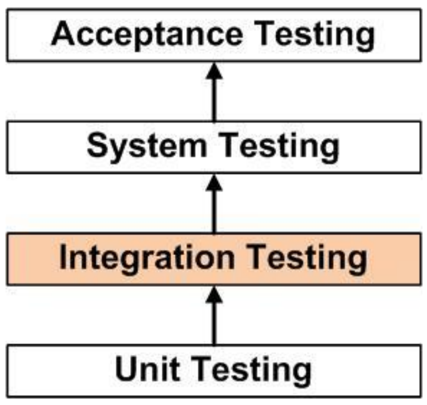


Fig 5.1. Integration Testing

System Testing

Software system integration testing. For software SIT is part of the software testing life cycle for collaborative projects. Usually, a round of SIT precedes the user acceptance test (UAT) round. Software providers usually run a pre-SIT round of tests before consumers run their SIT test cases. Integration testing is a testing in which one or two modules which are unit tested are integrated to test and verification is done to verify if the integrated modules work as expected or not.

System Integration Testing (SIT) & User Acceptance Testing (UAT) both are part of Functional Testing. System Integration Testing (SIT) ensures that individual modules, product set-ups, batch operations, basic reporting functionalities and key interfaces of an application-under-test (AUT) work well.

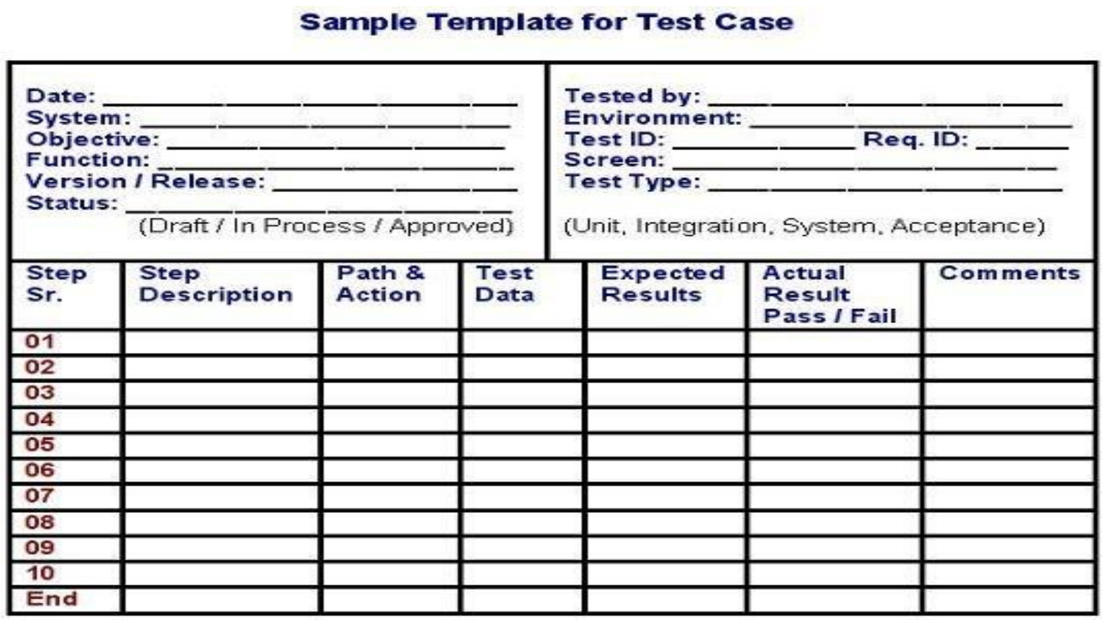
Lecture: 26

Test Cases

A TEST CASE is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly. The process of developing test cases can also help find problems in the requirements or design of an application.

Different types of test cases:

- Functionality Test Cases.
- User Interface Test Cases.
- Performance Test Cases.
- Integration Test Cases.
- Usability Test Cases.
- Database Test Cases.
- Security Test Cases.
- User Acceptance Test Cases.



Sample Template for Test Case

Date: ______ System: ______ Objective: ______ Function: ______ Version / Release: ______ Status: ______ (Draft / In Process / Approved)	Tested by: ______ Environment: ______ Test ID: ______ Req. ID: ______ Screen: ______ Test Type: ______ (Unit, Integration, System, Acceptance)

Step Sr.	Step Description	Path & Action	Test Data	Expected Results	Actual Result Pass / Fail	Comments
01						
02						
03						
04						
05						
06						
07						
08						
09						
10						
End						

Fig 5.2 Template for Test Case

Difference between Test case and Test scenario: Test case consist of a set of input values, execution precondition, expected results and executed post condition, developed to cover certain test condition. A Test Scenarios has one or many relations with Test Case, meaning a scenario can have multiple test cases.

“A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements and works correctly.” Typically, test cases should be small, isolated and atomic. Test cases should be easy to understand and steps should be executed fast.

While a test case is only designed to test a particular scenario, a test plan is a comprehensive document that lays out all major activities associated with a particular testing project. A test plan includes: Scope of the project Objectives.

White Box and Black Testing

Black box testing is the Software testing method which is used to test the software without knowing the internal structure of code or program. White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.

So normally white box test cases are written after black box test cases are written. Black Box Test Cases do not require system understanding but white Box Testing needs more structural understanding. And structural understanding is clearer i00n the later part of project, i.e., while executing or designing.

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.

In computer hardware, a white box is a personal computer or server without a well-known brand name. For instance, the term applies to systems assembled by small system integrators and to home-built computer systems assembled by end users from parts purchased separately at retail.

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing).

Typical black-box test design techniques include:

- Decision table testing.
- All-pairs testing.
- Equivalence partitioning.
- Boundary value analysis.
- Cause–effect graph.
- Error guessing.
- State transition testing.
- Use case testing.

Lecture: 27

Software Quality

The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations. software quality: The totality of functionality and features of a software product that bear on its ability to satisfy stated or implied needs.

A quality plan is a document, or several documents, that together specify quality standards, practices, resources, specifications, and the sequence of activities relevant to a particular product, service, project, or contract.

The 11 factors are grouped into three categories – product operation, product revision, and product transition factors. Product operation factors − Correctness, Reliability, Efficiency, Integrity, Usability.

Quality Assurance

Quality assurance (QA) is a way of preventing mistakes and defects in manufactured products and avoiding problems when delivering solutions or services to customers; which ISO 9000 defines as "part of quality management focused on providing confidence that quality requirements will be fulfilled".

The difference is that QA is process oriented and QC is product oriented. Testing, therefore is product oriented and thus is in the QC domain. Testing for quality isn't assuring quality, it's controlling it. Quality Assurance makes sure you are doing the right things, the right way.

QA/QC is the combination of quality assurance, the process or set of processes used to measure and assure the quality of a product, and quality control, the process of ensuring products and services meet consumer expectations.

Quality assurance managers play a crucial role in business by ensuring that products meet certain thresholds of acceptability. They plan, direct or coordinate quality assurance programs and formulate quality control policies. They also work to improve an organization's efficiency and profitability by reducing waste.

Quality Assurance Activities. Quality Assurance Activities or QAA is designed for product evaluation and process monitoring. They also assure that the product development and associated processes are correctly carried out as per the process control plan.

The difference between quality assurance (QA) testing and user acceptance testing (UAT) varies across organizations and often within organizations. In general, UAT is usually the final testing process prior to deployment and is performed by a business tester who ultimately signs off on the product.

The seven tools are:

- Cause-and-effect diagram (also known as the "fishbone" or Ishikawa diagram)
- Check sheet.
- Control chart.
- Histogram.
- Pareto chart.
- Scatter diagram.
- Stratification (alternately, flow chart or run chart)

Quality assurance and audit functions. Auditing is part of the quality assurance function. It is important to ensure quality because it is used to compare actual conditions with requirements and to report those results to management.

Quality assurance (QA) is the systematic process of determining whether products meet customers' expectations. ASQ's quality assurance training courses can help teach you how to avoid problems when delivering solutions or services to customers.

Importance of a QA team. According to Wikipedia quality assurance definition, it is 'a way of preventing mistakes or defects in manufactured products and avoiding problems when delivering solutions or services to customers'. It helps meet clients' demands and expectations most fully.

Key skills for quality assurance managers

- Confidence.
- Excellent technical skills.
- Good numerical skills and an understanding of statistics.
- Leadership skills.
- Planning and organisation skills.
- Communication and interpersonal skills.
- Problem-solving skills.
- Team working skills.

At the very root of what they do, QA analysts are testers and problem solvers. Job duties include testing websites or software for problems, documenting any issues and ensuring errors are corrected. They are a crucial component to any software development process.

Lecture: 28

Software Maintenance

Software maintenance in software engineering is the modification of a software product after delivery to correct faults, to improve performance or other attributes. A common perception of maintenance is that it merely involves fixing defects.

Types of Software Maintenance

- There are four types of maintenance, namely, corrective, adaptive, perfective, and preventive. ...
- Corrective maintenance deals with the repair of faults or defects found in day-today system functions.
- In the event of a system failure due to an error, actions are taken to restore the operation of the software system.

Why Software Maintenance Is Necessary? Maintaining a system is equally important as Web Application Development. It keeps solutions healthy to deal with changing technical and business environment. It introduces technical advancements almost every day that improve solution efficiency to streamline business operations.

Definition of 'Software Maintenance' Definition: Software maintenance is a part of Software Development Life Cycle. Its main purpose is to modify and update software application after delivery to correct faults and to improve performance. Software is a model of the real world.

Top 10 things you should be doing to maintain your computer

1. Back up your data. When was the last time that you backed up your data?
2. Clean dust from your computer.
3. Clean up your cabling and everything else too.
4. Organize your installation disks.
5. Run antivirus and spyware scans regularly.
6. Clean up your software.
7. Clean up your OS.
8. Update everything.

The Five Types of Systems Software

- The five types of system software work closely with computer hardware.
- Windows 8.1 graphical desktop is a component of Windows operating system type.
- Intel driver page.

- BIOS chip in a desktop motherboard.
- BIOS setup utility.
- UEFI setup utility.

Software Maintenance Cost Defined. Software maintenance cost is derived from the changes made to software after it has been delivered to the end user. ... Perfective maintenance – costs due to improving or enhancing a software solution to improve overall performance (generally 5% of software maintenance costs)

Software maintenance in software engineering is the modification of a software product after delivery to correct faults, to improve performance or other attributes. A common perception of maintenance is that it merely involves fixing defects.

Systems Maintenance. is also called as : IT Maintenance, System Maintenance, Support, Maintenance Management Software, Information Technology Maintenance. DEFINITION: The modification of a system to correct faults, to improve performance, or to adapt the system to a changed environment or changed requirements.

The Maintenance Phase occurs once the system is operational. It includes implementation of changes that software might undergo over a period of time, or implementation of new requirements after the software is deployed at the customer location.

Lecture: 29

Software configuration Management

In software engineering, software configuration management (SCM or S/W CM) is the task of tracking and controlling changes in the software, part of the larger cross-disciplinary field of configuration management. SCM practices include revision control and the establishment of baselines.

Configuration management (CM) is a systems engineering process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life.

Configuration management (CM) focuses on establishing and maintaining consistency of a product's performance, and its functional and physical attributes with its requirements, design, and operational information throughout its life.

The SCM activities are management and planning of the SCM process, software configuration identification, software configuration control, software configuration status accounting, software configuration auditing, and software release management and delivery.

List of Software configuration management tools available are:

- VSS – Visual source safe.
- CVS- Concurrent version system.
- Rational Clear Case.
- SVN- Subversion.
- Perforce.
- TortoiseSVN.
- IBM Rational team concert.
- IBM Configuration management version management.

Configuration Management Tools. Resources about configuration management tools, which fall into two categories: automation-based tools that can automatically configure software, and CMDB or similar tools that help organizations manage and track software configurations.

Goals of a configuration management tool. A configuration management system helps development teams control the process of modifying source code, developing documentation, and managing products.

Configuration Management helps organizations to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle. The primary goal of the SCM process is to increase productivity with minimal mistakes

In software engineering, software configuration management (SCM or S/W CM)is the task of tracking and controlling changes in the software, part of the larger cross-disciplinary field of configuration management. SCM practices include revision control and the establishment of baselines.

Software configuration management (SCM) is a software engineering discipline consisting of standard processes and techniques often used by organizations to manage the changes introduced to its software products. SCM is also known as software control management.

Supply Chain Management (SCM) is the management and oversight of materials, information, and finances as they travel through the supply chain from supplier to manufacturer, wholesaler, retailer, and customer.

The term configuration item (CI) refers to the fundamental structural unit of a configuration management system. Examples of CIs include individual requirements documents, software, models, and plans.

Configuration management (CM) is the ongoing process of identifying and managing changes to deliverables and other work products. The configuration management plan (CMP) is developed to define, document, control, implement, account for, and audit changes to the various components of this project

Software Architecture

Software architecture refers to the high level structures of a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations

So, with that in mind, here are the latest advancements in architectural rendering software:

- Revit. Building Information Modeling (BIM) is all the rage these days and Autodesk Revit is specifically built to address this need. ...
- SketchUp.

- Rhino3D.
- Archicad.
- Maya.
- AutoCAD Architecture.
- 3ds Max.
- Vectorworks Architect.

A representation of a system, including a mapping of functionality onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and human interaction with these components.

Software Architect Job Description. Software Architects design and develop software systems and applications. They act as high-level decision makers in the process, determining everything from design choices to technical standards, such as platforms and coding standards.

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.

These programs are widely used in design firms all over the nation, but can also be used by everyday consumers, as well.

- Microstation.
- Archicad.
- Chief Architect.
- SketchUp.
- Autodesk Products, such as AutoCad, AutoCAD LT, Revit, VIZ, 3ds Max, AutoSketch, Maya, and other plugins and add-ons.

Here's a list of the top 5 interior design software tools employers expect their interior designers to be well versed with.

- SketchUp.
- Autodesk 3Ds Max.
- Autodesk AutoCad.
- Autodesk Revit.
- Autodesk Home-style.

Software architecture represents a common abstraction of a system that most if not all of the system's stakeholders can use as a basis for mutual understanding, negotiation, consensus, and communication. It is also the earliest point at which design decisions governing the system to be built can be analyzed.

MODULE: V

CODING AND DOCUMENTATION

REFERENCES

Web/Video Links -

1. http://www.softwaretestinggenius.com
2. https://www.techwalla.com/articles/what-is-cohesion-in-software-engineering
3. https://courses.cs.washington.edu/courses/cse403/96sp/coupling-cohesion.html
4. https://continuousdelivery.com/foundations/configuration-management/
5. https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/
6. https://www.quora.com/What-are-the-different-types-of-coupling-in-software-engineering
7. https://www.techopedia.com/definition/14738/data-hiding
8. https://www.quora.com/What-is-data-hiding-in-a-program
9. https://documentation.microfocus.com/help/topic/com.borland.silktest.workbench.doc/SILKTEST-FDF71E3D-VTTUTORIALMODULEOVERVIEW-CON.html
10. https://www.tutorialspoint.com/software_engineering/software_maintenance_overview.htm
11. https://www.owlgen.com/question/what-is-software-maintenance-explain-maintenance-process
12. https://www.youtube.com/watch?v=TYLpChzXTBc
13. https://www.youtube.com/watch?v=W49zokvnL2g
14. https://www.youtube.com/watch?v=i2E1VDjmrXo
15. https://www.youtube.com/watch?v=9VaZLEW0aTI
16. https://www.youtube.com/watch?v=lt_USS_B-jg
17. https://www.youtube.com/watch?v=YjvSbg3Te0k
18. https://www.youtube.com/watch?v=pTB0EiLXUC8
19. https://www.youtube.com/watch?v=YcbcfkLzgvs

MODULE: V

CODING AND DOCUMENTATION

MULTIPLE CHOICE QUESTIONS

1. …………………. is the process, which controls the changes made to a system and manages the different versions of the software project.
   A) Software maintenance
   B) Configuration management
   C) Software re-engineering
   D) Software refactoring

2. The process of changing a system after it has been delivered and is in use is called …………..
A) Software maintenance
B) Configuration management
C) Software re-engineering
D) Software refactoring

3. ………………. is concerned with taking existing legacy systems and re-implementing them to make it more maintainable.
   A) Software maintenance
   B) Configuration management
   C) Software re-engineering
   D) Software refactoring

4. T or F? Coupling and cohesion are closely linked in that as one increases, so does the other.
A. True
B. False

5. When an expected result is not specified in test case template then ___________.

A. We cannot run the test.
B. It may be difficult to repeat the test.
C. It may be difficult to determine if the test has passed or failed.
D. We cannot automate the user inputs.

6) A test technique that involves testing with various ranges of valid and invalid inputs of a particular module or component functionality extensively is ___________.

A. Gorilla Testing
B. Monkey Testing
C. Agile Testing
D. Baseline Testing

7) Which Testing is performed first?

A. Black box testing
B. White box testing
C. Dynamic testing
D. Static testing

8) Testing beyond normal operational capacity is ___________.

A. Load testing
B. Performance testing
C. Stress testing
D. All of these.

9) Classification of software maintenance include/includes :
A A. corrective maintenance
B B. adaptive maintenance
C C. perfective maintenance
D D. All of these

10) Software maintenance takes inputs from
A A. SRS
B B. Customer feedback reports
C C. SDD
D D. All of the above

11. What is the typical relationship between coupling and cohesion?
A. There is no relationship between coupling and cohesion.
B. As cohesion increases, coupling increases.
C. As cohesion increases, coupling decreases.

MODULE: V

CODING AND DOCUMENTATION

SHORT ANSWER TYPE QUESTIONS

1. What is cohesion and coupling?
2. What is data coupling?
3. Is Data Hiding the same as encapsulation?
4. How does a class enforce information hiding?
5. What is quality control and quality assurance?
6. What are the 7 QC tools?
7. What is system architecture in software engineering?
8. Why maintenance is required for software?
9. What are the four types of software?
10. What are the features of OOP?

MODULE: VI

SOFTWARE PROJECT MANAGEMENT

Lecture: 30

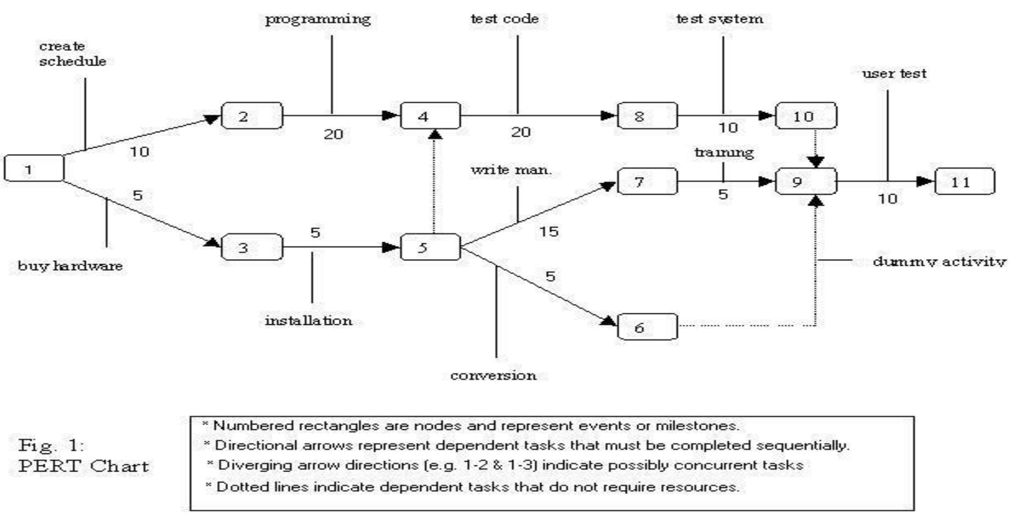
Project Scheduling

The project schedule is the tool that communicates what work needs to be performed, which resources of the organization will perform the work and the timeframes in which that work needs to be performed.

In project management, a schedule is a listing of a project's milestones, activities, and deliverables, usually with intended start and finish dates. A schedule is commonly used in the project planning and project portfolio management parts of project management.

Effective project scheduling plays a crucial role in ensuring project success. To keep projects on track, set realistic time frames, assign resources appropriately and manage quality to decrease product errors. This typically results in reduced costs and increased customer satisfaction.

Scheduling techniques in Project Management- Techniques such as PERT (Program Evaluation and Review Technique), CPM (Critical Path Method) and GANTT are the most used to plan into details a project, prevent uncertainties and avoid risk.



* Numbered rectangles are nodes and represent events or milestones.
* Directional arrows represent dependent tasks that must be completed sequentially.
* Diverging arrow directions (e.g. 1-2 & 1-3) indicate possibly concurrent tasks
* Dotted lines indicate dependent tasks that do not require resources.

Fig: 6.1 PERT Charts

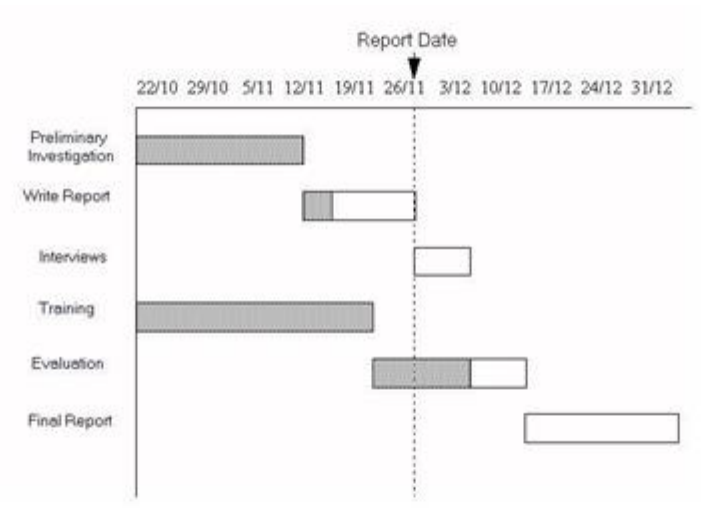


Fig: 6.2 Gnatt Chart

The top free project management software options

- Stream time: Collaborative work management tool for creative agencies.
- Trello: Simple and easy-to-use Kanban board.
- Wrike: Strong security for very small teams.
- GanttProject: Established project planning tool.
- Open Project: Robust project management solution

Here's our quick guide to effective project management scheduling.

1. Step 1: Write Down Your Tasks. First, you're going to work out what it is that you have to do
2. Step 2: Establish the Order of Tasks.
3. Step 3: Create Some Milestones.
4. Step 4: Calculate the Timescale.
5. Step 5: Allocate People to Tasks.
6. Step 6: Review Regularly.

Level 1 Schedule Executive Summary, also called a Project Master Schedule(PMS). This is a major milestone type of schedule; usually only one page, it highlights major project activities, milestones, and key deliverables for the whole project.

The purpose of schedules is to accomplish recurrent tasks on a regular interval in order to free the Coveo administrator from having to perform these basic operations manually. In other words, schedules maintain the index in its optimal state and let the Coveo administrator focus on more important duties.

Project management is important because it ensures a project's progress is tracked and reported properly. This data is invaluable not only for tracking progress but helps clients gain the trust of other stakeholders in their organization, giving them easy oversight of a project's progress.

Scheduling staff creates an order and a flow to your business. Everyone knows when they're supposed to work, which allows them to focus on their job. Proper scheduling ensures the important tasks are covered at appropriate times.

A tool that provides schedule component names, definitions, structural relationships and formats that support the application of a scheduling method is referred to as scheduling tool. Scheduling tool is one of the factors that influence the sequence activities process

Let's look at a few of the best Gantt chart software available so you can see which one is best for your team.

- Workzone. Workzone integrates Gantt chart software into all of its project views.
- Easy Projects.
- Team Gantt.
- Ganttpro.
- Celoxis.
- ProjectLibre.
- Project Insight.
- Wrike.

The difference between Tasks and Projects: A task is a single thing that you can do in one session. A task might be labeling your products or making a single item. Aproject is bigger and includes multiple tasks.

Lecture: 31

Project Staffing-

Project staff means a group of persons involved in a project and belong to the project team. A project group always consists of the project manager (leader) and persons that are charged by the project manager for the task implementation in a project.

The core project team is thereby distinguished. It consists of the project manager and the project group that are responsible for the project execution. The project group is composed of the core team and other persons that are charged only for certain tasks or phases in a project. There are permanent staff, who always works on a project and belongs to the project team, and temporary staff, the specialists that are not engaged for the whole project duration. The participants of the core team can also decide who can be a speaker compared to a supervisor of the project.

The project group should be as small as possible. The project manager determines the project members for the duration of the project. It is very important to note, that all areas are represented in a project team. They are also disciplinary led by a project manager during the project.

Project staff means a group of persons involved in a project and belong to the project team. There are permanent staff, who always works on a project and belongs to the project team, and temporary staff, the specialists that are not engaged for the whole project duration.

Staff planning is a systematic process to ensure that an organization has the right number of people with the right skills to fulfil business needs. You must take into account internal and external changes and must integrate HR planning with the company's business plan.

Staffing Management Plan. A staffing management plan or process is ultimately a document that explains the various human resources requirements that will be met for both staff management and employees alike.

A Responsibility Assignment Matrix (RAM), also known as RACI matrix or Linear Responsibility Chart (LRC), describes the participation by various roles in completing tasks or deliverables for a project or business process.

A resource histogram is one of the tools given to us by the companies that produce project management software to help the allocation of resources in project plans. Automatic resource leveling does not usually give us the best solution to aresource allocation problem.

The function of strategic staffing is to recruit and retain employees to perform jobs in line with your company's overall goals. By identifying needs before recruiting, developing and keeping employees, your organization will be prepared to maintain status quo or weather growth or decline.

Staffing Process in Management. Staffing means manning of an organization. Staffing process involves determination of manpower requirements, recruitment, selection, placement, training, development, job transfer and appraisal of personnel to fill the various positions in an organization.

Staffing is the function of employee recruitment, screening and selection performed within an organization or business to fill job openings. Some related terms and departments include human resources, personnel management and hiring.

It is also concerned with employing the right type of people and developing their skills through training. The staffing function focuses on maintaining and improving the manpower in an organization. Human resource is a very important factor of organization because other resources depend upon in.

Staffing function is normally the sub function of the organizing function. All the five functions of the management viz. planning, organizing, directing, coordinating, and controlling depend upon the employees of the organization which are made available through the staffing function. Staffing is a pervasive activity.

Meaning of Staffing: The term 'Staffing' relates to the recruitment, selection, development, training and compensation of the managerial personnel. Staffing, like all other managerial functions, is the duty which the apex management performs at all times. In a newly created enterprise, the staffing would come as a.

RACI is an acronym that stands for responsible, accountable, consulted and informed. A RACI chart is a matrix of all the activities or decision making authorities undertaken in an organisation set against all the people or roles.

Lecture: 32

Project Monitoring

Project Monitoring refers to the process of keeping track of all project-related metrics including team performance and task duration, identifying potential problems and taking corrective actions necessary to ensure that the project is within scope, on budget and meets the specified deadlines.

Following diagram shows the project monitoring cycle to be followed at regular period of intervals of the project duration.

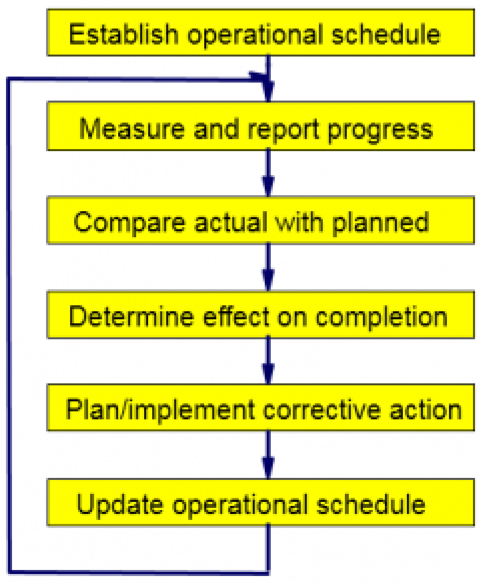

Establish operational schedule
Measure and report progress
Compare actual with planned
Determine effect on completion
Plan/implement corrective action
Update operational schedule

Fig 6.3 Project Monitoring cycle

A Monitoring and Evaluation (M&E) Plan is a guide as to what you should evaluate, what information you need, and who you are evaluating for. ... Retro- fitting an M&E plan to an existing project may just mean that you may be constrained in some of the data that you can collect.

The Monitoring and Controlling process oversees all the tasks and metrics necessary to ensure that the approved and authorized project is within scope, on time, and on budget so that the project proceeds with minimal risk. ... Monitoring and Controlling process is continuously performed throughout the life of the project.

Monitoring and Evaluation (M&E) is a process that helps improve performance and achieve results. Its goal is to improve current and future management of outputs, outcomes and impact.

There are several types of monitoring in M&E and they include process monitoring, technical monitoring, assumption monitoring, financial monitoring and impact monitoring.

- Process monitoring/ physical progress monitoring. ...
- Technical monitoring. ...
- Assumption monitoring. ...
- Financial Monitoring. ...

- Impact Monitoring

Monitoring is the regular observation and recording of activities taking place in a project or programme. It is a process of routinely gathering information on all aspects of the project. To monitor is to check on how project activities are progressing. It is observation; ─ systematic and purposeful observation.

Why is monitoring and evaluation important? ... At the programme level, the purpose of monitoring and evaluation is to track implementation and outputs systematically, and measure the effectiveness of programmes. It helps determine exactly when a programme is on track and when changes may be needed.

Lecture: 33

Risk Management

Risk management is an ongoing process that is implemented as part of the initial project planning activities and utilized throughout all of the phases of the software development lifecycle. Risk management requires a fear-free environment where risks can be identified and discussed openly.

Risk is an expectation of loss, a potential problem that may or may not occur in the future. It is generally caused due to lack of information, control or time. A possibility of suffering from loss in software development process is called a software risk. Risk management is carried out to: Identify the risk.

Software development is activity connected with advanced technology and high level of knowledge. Risks on software development projects must be successfully mitigated to produce successful software systems. Lack of a defined approach to risk management is one of the common causes for project failures.

In software testing, risk analysis is the process of identifying risks in applications and prioritizing them to test. A risk is a potential for loss or damage to an organization from materialized threats. Items with lower risk value can be tested later, or not at all. It can also be used with defects

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.

Together these 5 risk management process steps combine to deliver a simple and effective risk management process.

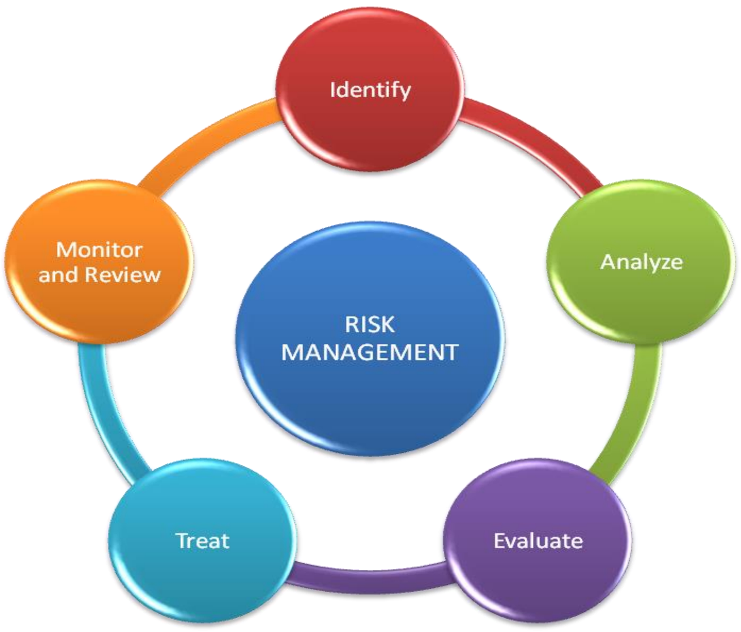


Fig:6.4 Risk Management Process steps

- Step 1: Identify the Risk.
- Step 2: Analyze the risk.
- Step 3: Evaluate or Rank the Risk.
- Step 4: Treat the Risk.
- Step 5: Monitor and Review the risk.

In business, risk management is defined as the process of identifying, monitoring and managing potential risks in order to minimize the negative impact they may have on an organization. Examples of potential risks include security breaches, data loss, cyber attacks, system failures and natural disasters.

In software engineering, software configuration management (SCM or S/W CM) is the task of tracking and controlling changes in the software, part of the larger cross-disciplinary field of configuration management. SCM practices include revision control and the establishment of baselines.

How to Develop a Risk Management Plan

1. Understand how Risk Management works.
2. Define your project.
3. Get input from others.
4. Identify the consequences of each risk.
5. Eliminate irrelevant issues.
6. List all identified risk elements.
7. Assign probability.
8. Assign impact.

Risk identification is the process of determining risks that could potentially prevent the program, enterprise, or investment from achieving its objectives. It includes documenting and communicating the concern. Keywords: risk, risk identification, risk management.

Five steps to risk assessment can be followed to ensure that your risk assessment is carried out correctly, these five steps are:

1. Identify the hazards.
2. Decide who might be harmed and how.
3. Evaluate the risks and decide on control measures.
4. Record your findings and implement them.

Software Risk Management: Since there could be various risks associated with the software development projects, the key to identify and manage those risks is to know about the concepts of software risk management. Basically, risk analysis is used to identify the high risk elements of a project in software engineering.

In software testing, risk analysis is the process of identifying risks in applications and prioritizing them to test. A risk is a potential for loss or damage to an organization from materialized threats. Risk Analysis attempts to identify all the risks and then quantify the severity of the risks.

What Is Software Risk And Software Risk Management? Risk is an expectation of loss, a potential problem that may or may not occur in the future. It is generally caused due to lack of information, control or time. A possibility of suffering from loss in software development process is called a software risk.

Risk is the possibility of suffering loss. Risk Management is a software engineering practice with processes, methods, and tools for managing risks in a project. It provides a disciplined environment for proactive decision-making to: assess continuously what can go wrong (risks identification and assessment).

Risk management is a four-step process for controlling exposure to health and safety risks associated with hazards in the workplace.

- Step 1: Identify hazards. Examples of common hazards which can lead to musculoskeletal disorders (MSD)
- Step 2: Assess the risk.
- Step 3: Control the risk.
- Step 4: Review risk control.

Some of the techniques of quantitatively determining probability and impact of a risk include:

- Interviewing.
- Cost and time estimating.
- Delphi technique.

- Historical Records.
- Expert judgment.
- Expected monetary value analysis.
- Monte Carlo Analysis.
- Decision tree.

The most common types of risk management techniques include avoidance, mitigation, transfer and acceptance.

- Avoidance of Risk. The easiest way for a business to manage its identified risk is to avoid it altogether.
- Risk Mitigation.
- Transfer of Risk.
- Risk Acceptance.

- Step 1: Hazard identification. This is the process of examining each work area and work task for the purpose of identifying all the hazards which are “inherent in the job”.
- Step 2: Risk identification.
- Step 3: Risk assessment.
- Step 4: Risk control.
- Step 5: Documenting the process.
- Step 6: Monitoring and reviewing.

The Main Types of Business Risk

- Strategic Risk. Everyone knows that a successful business needs a comprehensive, well-thought-out business plan.
- Compliance Risk. Are you complying with all the necessary laws and regulations that apply to your business?
- Operational Risk.
- Financial Risk.
- Reputational Risk.

Once risks have been identified and assessed, all techniques to manage the risk fall into one or more of these four major categories:

- Avoidance (eliminate, withdraw from or not become involved)
- Reduction (optimize – mitigate)
- Sharing (transfer – outsource or insure)
- Retention (accept and budget)

Reactive vs. Proactive Risk

Proactive Strategies are interventions which are used on an ongoing basis in an attempt to reduce the likelihood of occurrence of the challenging behavior. ...Reactive Strategies are interventions which are used only once the behavior occurs. They are consequences (or reactions) to the behavior.

A proactive approach focuses on eliminating problems before they have a chance to appear and a reactive approach is based on responding to events after they have happened. The difference between these two approaches is the perspective each one provides in assessing actions and events.

The Future of Risk Management: A Proactive Approach. ... To achieve effective enterprise risk management, organizations must focus on being proactive, rather than merely reactive, and use risk management to both drive competitive advantage and sustain future profitability and growth.

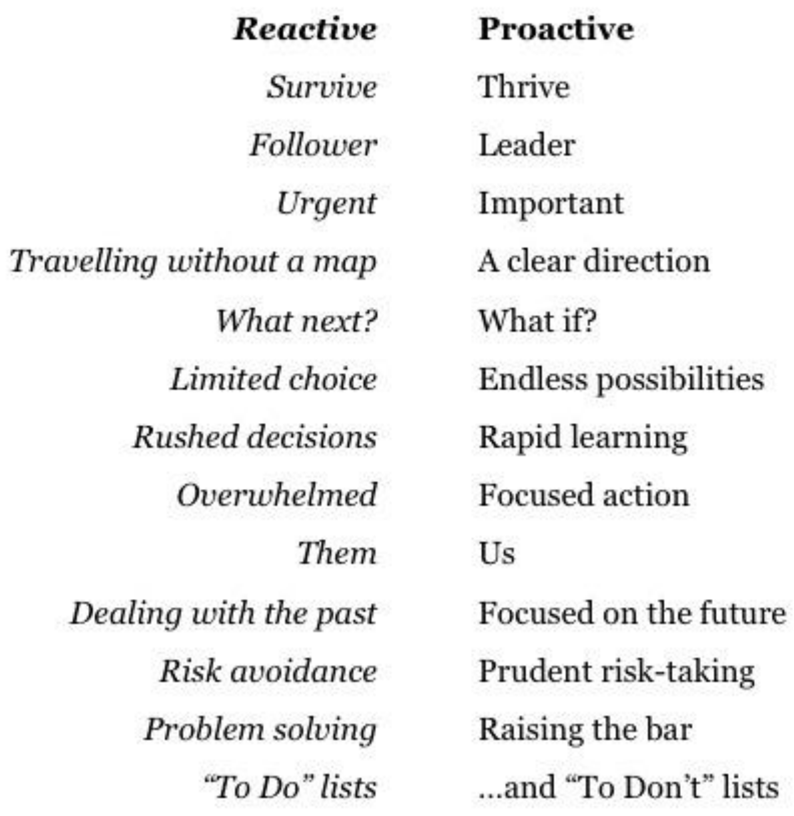
Risk is an expectation of loss, a potential problem that may or may not occur in the future. It is generally caused due to lack of information, control or time. A possibility of suffering from loss in software development process is called a software risk. ... Reduce the probability or likelihood of risk.

A proactive approach focuses on eliminating problems before they have a chance to appear and a reactive approach is based on responding to events after they have happened. The difference between these two approaches is the perspective each one provides in assessing actions and events.

Reactive change is change initiated in an organization because it is made necessary by outside forces. For instance, introduction of a new employee benefit scheme is proactive as the management strongly believes that it enhances the satisfaction and motivation of employees.

Being willing to take action when a problem or issue presents itself is part of proactive behavior -- neither reacting impulsively nor allowing a situation to play out on its own.

Problem management can be reactive or proactive. Reactive problem management aims to find and eliminate the root cause of known incidents, while proactive problem management aims to identify and prevent future incidents from recurring by identifying and eliminating the root cause



Reactive	Proactive
Survive	Thrive
Follower	Leader
Urgent	Important
Travelling without a map	A clear direction
What next?	What if?
Limited choice	Endless possibilities
Rushed decisions	Rapid learning
Overwhelmed	Focused action
Them	Us
Dealing with the past	Focused on the future
Risk avoidance	Prudent risk-taking
Problem solving	Raising the bar
"To Do" lists	...and "To Don't" lists

Table 6.1 Reactive vs. Proactive

Software Risks

Risk is an expectation of loss, a potential problem that may or may not occur in the future. It is generally caused due to lack of information, control or time. A possibility of suffering from loss in software development process is called a software risk. ... Reduce the probability or likelihood of risk. Risk monitoring.

Various Kinds of Risks Associated with Software Project...

- Schedule / Time-Related / Delivery Related Planning Risks.
- Budget / Financial Risks.
- Operational / Procedural Risks.
- Technical / Functional / Performance Risks.
- Other Unavoidable Risks.

In software testing Risks are the possible problems that might endanger the objectives of the project stakeholders. It is the possibility of a negative or undesirable outcome. A risk is something that has not happened yet and it may never happen; it is a potential problem.

What is Software Risk And Software Risk Management? Risk is an expectation of loss, a potential problem that may or may not occur in the future. It is generally caused due to lack of information, control or time. A possibility of suffering from loss in software development process is called a software risk.

Lecture: 34

Risk Identification

The risk identification process on a project is typically one of brainstorming, and the usual rules of brainstorming apply:

1. The full project team should be actively involved.
2. Potential risks should be identified by all members of the project team.
3. No criticism of any suggestion is permitted.

Some of the techniques of quantitatively determining probability and impact of a risk include:

1. Interviewing.
2. Cost and time estimating.
3. Delphi technique.
4. Historical Records.
5. Expert judgment.
6. Expected monetary value analysis.
7. Monte Carlo Analysis.
8. Decision tree.

Five steps to risk assessment can be followed to ensure that your risk assessment is carried out correctly, these five steps are:

1. Identify the hazards.
2. Decide who might be harmed and how.
3. Evaluate the risks and decide on control measures.
4. Record your findings and implement them
5. Review your assessment and update if necessary

Risk Projection

The consequences of the problems associated with the risk, should it occur. Project planner, along with other managers and technical staff, performs four risk projection activities: (1) establish a measure that reflects the perceived likelihood of a risk.

Risk projection, also called risk estimation, attempts to rate each risk in two ways – the likelihood or the probability that the risk is real and the consequences of the problems associated with the risk, should it occur.

The project planner, along with other managers and technical staff, performs four risk projection activities – (1) establish a scale that reflects the perceived likelihood of a risk, (2) delineate the consequences of the risk, (3) estimate the impact of the risk on the project and the product and (4) note the overall accuracy of the risk projection so that there will be no misunderstandings.

MODULE: VI

SOFTWARE PROJECT MANAGEMENT

REFERENCES

1. https://www.opsfolio.com/responsibilitycenter/proactive-vs-reactive-risk-management/
2. https://www.differencebetween.com/difference-between-proactive-and-vs-reactive-risk-management/
3. https://slideplayer.com/slide/6417787/
4. https://www.projectmanager.com/blog/what-is-project-scheduling
5. http://csit.merospark.com/sixth-semester/project-management-staffing-planning-planning-process-software-engineering/
6. https://www.test-institute.org/What_Is_Software_Risk_And_Software_Risk_Management.php
7. https://www.castsoftware.com/research-labs/risk-management-in-software-development-and-software-engineering-projects
8. https://www.tutorialride.com/software-engineering/risk-management-in-software-engineering.htm

MODULE: VI

SOFTWARE PROJECT MANAGEMENT

MULTIPLE CHOICE QUESTIONS

1) A ____ is a set of activities which are networked in an order and aimed towards achieving the goals of a project.
(A) Project
(B) Process
(C) Project management
(D) Project cycle

2) Resources refers to
(A) Manpower
(B) Machinery
(C) Materials
(D) All of the above

3) Developing a technology is an example of
(A) Process
(B) Project
(C) Scope
(D) All of the above

4) The project life cycle consists of
(A) Understanding the scope of the project
(B) Objectives of the project
(C) Formulation and planning various activities
(D) All of the above

5) Following is(are) the responsibility(ies) of the project manager.
(A) Budgeting and cost control
(B) Allocating resources
(C) Tracking project expenditure
(D) All of the above

6) Project team becomes aware of a recent innovation that allows a

A. Faster solution
B. Cheaper solution
C. Both A and B
D. Expensive Solution

7) Getting many more projects than management expected, is a symptom of a misaligned

A. Strategy
B. Vision
C. Portfolio

D. Mission

8) Staffing is the function of …

A. employee recruitment
B. screening
C. Selection
D. All of the above

9) RACI is an acronym that stands for

A. responsible, assigned, consulted and informed
B. responsible, assigned, consulted and instructed
C. responsible, accountable, consulted and informed
D. None of the above

10) In software testing, risk analysis is the process of …

A. identifying risks in applications and prioritizing them to test
B. only identifying risks in applications
C. only prioritizing risks
D. None of the above

MODULE: VI

SOFTWARE PROJECT MANAGEMENT

SHORT ANSWER TYPE QUESTIONS

1. What is proactive and reactive strategy?
2. Why is scheduling important in project management?
3. What is proactive and reactive problem management?
4. What is a staffing plan?
5. What is a scheduling tool?
6. What is HR staffing?
7. What is software engineering risk?
8. What does RACI mean in project management?
9. What is the best project scheduling software?
10. What is a proactive decision?