[CSc. 351 Software Engineering]

Lecturer: ***Hiranya Bastakoti***

Amrit Science Campus

1.1.1 Software:

- ✓ Software is computer programs and associated documentation. This definition clearly states that, the software is not a collection of programs, but includes all associated documentation.
- ✓ Software system usually consists of a number of separate programs, configuration files: which are used to set up these programs, System documentation: which describes the structure of the system, and user documentation: which explains how to use the system and web sites for users to download recent product information.
- ✓ Software products may be developed for a particular customer or may be developed for a general market

Software products may be:

- **Generic** – These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them. (i.e. developed to be sold to a range of different customers). e.g. Databases, Office packages, Drawing Packages etc.
- **Bespoke (custom)** – These are the systems which are commissioned by a particular customer. A software contractor develops the software especially for that customer. (i.e. developed for a single customer according to their specification). e.g.: Control system for electronic device, software to support particular business process.

1.1.2 Software Engineering:

- ✓ Software engineering is strategy for producing quality software.
- ✓ It is the establishment and use of sound engineering principles in order to obtain economically software is reliable and works efficiently on real machines.
- ✓ *Software engineering is an engineering discipline which is concerned with all aspects of software production*
- ✓ Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available
- ✓ Software engineering as a discipline provides us with structured technical means of developing and maintaining software. It provides methods to perform the tasks that the making of any software requires, analyzing the requirements, designing the system to meet these requirements, constructing the programs, maintaining the system, etc. Software engineering tools are used to support the tasks by automating the tasks or parts of the tasks.

The advantages of using software engineering for developing software are:

- Improved quality
- Improved requirement specification
- Improved cost and schedule estimates
- Better use of automated tools and techniques.
- Better maintenance of delivered software
- Well defined process
- Improved reliability
- Improved productivity
- Less defects in final processes

Why software Engineering?

- ✓ The economies of ALL developed nations are dependent on software
- ✓ More and more systems are software controlled
- ✓ Software engineering is concerned with theories, methods and tools for professional software development
- ✓ Software engineering expenditure represents a significant fraction of GNP in all developed countries

What is the difference between software engineering and computer science?

- ✓ Computer science is concerned with theories and methods that underline computer software system.; software engineering is concerned with the practicalities of developing and delivering useful software
- ✓ Some knowledge of computer science is essential for software engineers
- ✓ Computer science theories are currently insufficient to act as a complete underpinning for software engineering

What is a software process?

- ✓ It is a set of activities and associated results that produce a software product.
- ✓ Generic activities in all software processes are:

• **Specification** –where customers and engineers define the software to be produced and the constraints on its operation. (i.e. what the system should do and its development constraints)

- **Development** – Where the software is designed and programmed. (i.e. production of the software system)
- **Validation** – where the software is checked to ensure that it is what the customer requires. (i.e. checking that the software is what the customer wants)
- **Evolution** – Where the software is modified to adapt it to changing customer and market requirements. (i.e. changing the software in response to changing demands)

What is a software process model?

- ✓ It is a simplified representation of a software process, presented from a specific perspective. Software process model may include activities that are part of the software process, software products and the role of people involved in software engineering.
- ✓ **software process models are:**
  - **Workflow model** – (sequence of activities)

    This model shows the sequence of activities in the process along with their inputs, outputs and dependencies. The activities in this model represent human actions.
  - **Data-flow or Activity model** – (information flow)

    This model represents the process as a set of activities each of which carries out some data transformation. It shows how the input to the process, such as a specification, is transformed to an output, such as a design. The activities here may represent transformations carried out by people /computer
  - **Role/action Model** – (who does what)

    This model represents the roles of people involved in the process and the activities for which they are responsible.
- ✓ **Generic process models**
  - Waterfall
  - Evolutionary development
  - Formal transformation
  - Integration from reusable components-CBSE (computer -based software engineering)

Software costs

The distribution of costs across the different activities in the software process depends on the process used and the type of software that is being developed

- ✓ Software costs often dominate system costs. The costs of software on a PC are often greater than the hardware cost

- ✓ Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs
- ✓ Software engineering is concerned with cost-effective software development
- ✓ Roughly 60% of costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs
- ✓ Costs vary depending on the type of system being developed and the requirements of system attributes such as performance and system reliability
- ✓ Distribution of costs depends on the development model that is used

What are software engineering methods?

It is a structured approach to software development which includes system models, notations, rules, design advice and process guidance whose aim to facilitate the production of high-quality software in cost-effective way.

- ✓ **Model descriptions**
  - Descriptions of graphical models which should be developed and the notation used to define these models.eg. Object model, Data Flow Model
- ✓ **Rules**
  - Constraints applied to system models e.g. Every entity in a system model must be unique name
- ✓ **Recommendations-**
  - Heuristic which characterise good design practice in this method. Following these recommendations should lead to a well-organized system model (i.e. Advice on good design practice).e.g. No object should have more than seven sub-objects associated with it.
- ✓ **Process guidance**
  - Descriptions of the activities which may be followed to develop the system models and the organization of these activities. e.g. Object attributes should be documented before designing the operations associated with an object.

What is CASE (Computer –Aided Software Engineering)?

It covers a wide range of different types of programs that are used to support software process activities such as requirement analysis, system modelling, debugging and testing

Computer-Aided Software Engineering (CASE) tools are software programs that automate or support the drawing and analysis of system models and provide for the translation of system models into application programs. Some CASE tools also provide prototyping and code generation capabilities.

What are the key challenges facing software engineering?

- Coping with legacy systems, coping with increasing diversity and coping with demands for reduced delivery times are challenges of software engineering.

The key challenges are:

- **Legacy systems**
    - Old, valuable systems must be maintained and updated
- **Heterogeneity**
    - Systems are distributed and include a mix of hardware and software
- **Delivery**
    - There is increasing pressure for faster delivery of software

1.2 Software characteristics

✓ Like all engineering, software engineering is not just about producing product but involves producing products in a cost effective way. Given unlimited resources, the majority of software problems can probably be solved. The challenged for the software engineers is to produce high quality software with a finite amount of resources and to a predicted schedule.

✓ The software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable

Maintainability: (Software must evolve to meet changing needs)

Software should be written in such a way that it may evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable consequence of a changing environment.

Dependability: (Software must be trustworthy)

Software dependability has a range of characteristics, including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure**.**

Efficiency: (Software should not make wasteful use of system resources)

System should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, and memory utilization.

Usability: (Software must be usable by the users for which it was designed)

Software must be usable, without undue effort, by the type of user for whom it is designed. This means that it should have an appropriate user interface and adequate documentation.

Professional and ethical responsibility

- ✓ Software engineering involves wider responsibilities than simply the application of technical skills
- ✓ Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals
- ✓ Ethical behaviour is more than simply upholding the law.

Issues of professional responsibility

- ✓ **Confidentiality**
    - Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- ✓ **Competence**
    - Engineers should not misrepresent their level of competence. They should not knowingly accept work which is out with their competence.
- ✓ **Intellectual property rights**
    - Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
- ✓ **Computer misuse**
    - Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

[CSc. 351 Software Engineering]

Lecturer: ***Hiranya Bastakoti***

Amrit Science Campus

System Engineering: Chapter-Two Software Engineering

What is a system?

Definition: System is a purposeful collection of inter-related components working together towards some common objective.

OR

It is an interrelated set of components, with an identifiable boundary, working together for some purpose.

- ✓ A system may include software, mechanical, electrical and electronic hardware and be operated by people.
- ✓ System components are dependent on other system components
- ✓ Every system interacts with the environment through exchange of information.
- ✓ Every system must fit into its operating environment.
- ✓ All systems exhibit predictable behavior.
- ✓ All systems come in hierarchies. Therefore, system exists to maintain its higher level systems.

Characteristics of a System:

Components: An irreducible part or aggregation of parts that make up a system, also called subsystem

Interrelated Components: Dependence of one subsystem on one or more subsystems.

Boundary: The line that marks the inside and outside of a system and that sets off the system from its environment.

Purpose: The overall goal or function of a system.

Environment: Everything external/internal to a system that interacts with the system.

Interfaces: Point of contact where a system meets its environment or where subsystems meet each other

Input: Whatever a system takes from its environment in order to fulfill its purpose.

Output: Whatever a system returns to its environment in order to fulfill its purpose.

Constraints: A limit to what a system can accomplish

Note:

- ✓ ***Open system:*** *A system that interacts freely with its environment, taking input and returning output.*
- ✓ ***Closed System:*** *A system that is cut off from its environment and does not interact with* ***it***

A computer-based system makes use of a variety of system elements which includes:
Software: Computer programs, data structures, and related documentation that serve to effect the logical method, procedure, or control that is required.

Hardware: Electronic devices that provide computing capability, the interconnectivity devices (e.g., network switches, telecommunications devices) that enable the flow of data, and electromechanical devices (e.g., sensors, motors, pumps) that provide external world function.

People: Users and operators of hardware and software.

Database: A large, organized collection of information that is accessed via software.

Documentation: Descriptive information (e.g., hardcopy manuals, on-line help files, Web sites) that portrays the use and/or operation of the system.

Procedures: The steps that define the specific use of each system element or the procedural context in which the system resides.

Emergent system properties:

- ✓ Properties of the system as a whole rather than properties that can be derived from the properties of components of a system
- ✓ Emergent properties are a consequence of the relationships between system components
- ✓ They can therefore only be assessed and measured once the components have been integrated into a system

Types of emergent property

- **Functional properties**
  - ✓ These appear when all the parts of a system work together to achieve some objective. For example, a bicycle has the functional property of being a transportation device once it has been assembled from its components.
- **Non-functional emergent properties**
  - ✓ Examples are reliability, performance, safety, and security. These relate to the behaviour of the system in its operational environment. They are often critical for computer-based systems as failure to achieve some minimal defined level in these properties may make the system unusable

Examples of emergent properties

- ✓ **The volume of the system**
  - The volume (total space occupied) varies depending on how the component assemblies are arranged and connected.
- ✓ **The reliability of the system**
  - This depends on the reliability of system components and the relationships between the components but unexpected interactions can cause new types of failure and affect the reliability of the system.
- ✓ **The Security of the system**
  - The ability to resist attack means security. It is a complex property that cannot be easily measured.
- ✓ **The Reparability of the system**
  - This property reflects how easy it is to fix a problem with the system once it has been discovered. It depends on being able to diagnose the problem, access components that are faulty and modify or replace these components.
- ✓ ***The usability of a system***
  - This property reflects how easy it is to use the system which is not simply dependent on the system hardware and software but also depends on the system operators and the environment where it is used.

Influences on reliability

- ✓ ***Hardware reliability***
  - What is the probability of a hardware component failing and how long does it take to repair that component?
- ✓ ***Software reliability***
  - How likely is it that a software component will produce an incorrect output? Software failure is usually distinct from hardware failure in that software does not wear out.
- ✓ ***Operator reliability***
  - How likely is it that the operator of a system will make an error?

Reliability relationships

- ✓ Hardware failure can generate spurious signals that are outside the range of inputs expected by the software

- ✓ Software errors can cause alarms to be activated which cause operator stress and lead to operator errors
- ✓ The environment in which a system is installed can affect its reliability

The 'shall-not' properties

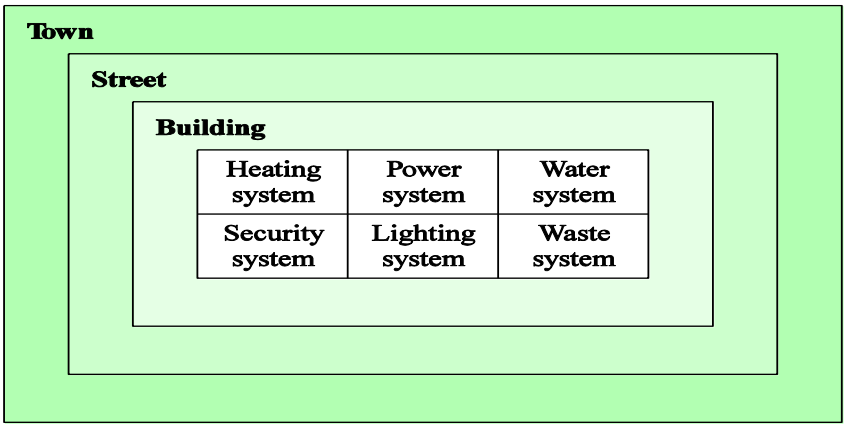
- ✓ Properties such as performance and reliability can be measured
- ✓ However, some properties are properties that the system should not exhibit
    - o Safety - the system should not behave in an unsafe way
    - o Security - the system should not permit unauthorised use
- ✓ Measuring or assessing these properties is very hard

- ✓ Properties such as performance and reliability can be measured
- ✓ However, some properties are properties that the system should not exhibit
    - o Safety - the system should not behave in an unsafe way
    - o Security - the system should not permit unauthorised use
- ✓ Measuring or assessing these properties is very hard

Systems and their environment

- ✓ Systems are not independent but exist in an environment
- ✓ System's function may be to change its environment
- ✓ Environment affects the functioning of the system e.g. system may require electrical supply from its environment
- ✓ The organizational as well as the physical environment may be important

System hierarchies



Human and organizational factors in Organizations

- ✓ ***Process changes***
  - Does the system require changes to the work processes in the environment?
- ✓ ***Job changes***
  - Does the system de-skill the users in an environment or cause them to change the way they work?
- ✓ ***Organisational changes***
- ✓ **Does the system change the political power structure in an organisation?**

Systems Engineering:

Systems engineering is the activity of specifying, designing, implementing, validating, deploying and maintaining socio-technical systems.

Discipline involved in System engineering:

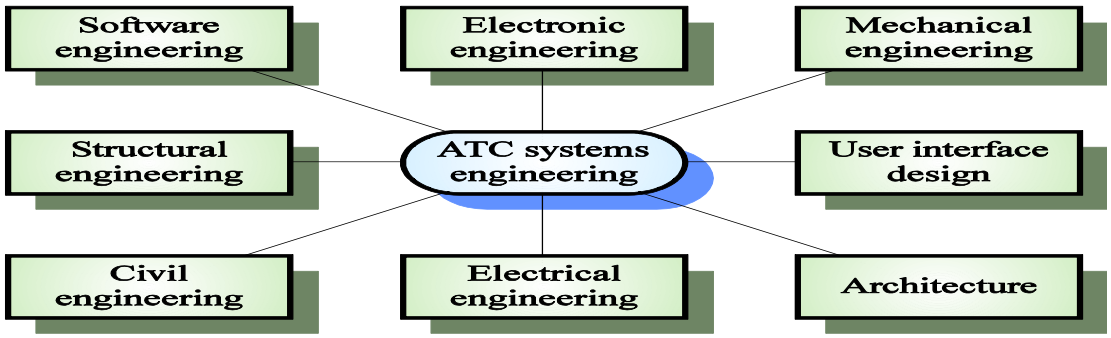


Fig: Discipline involved in System Engineering (Air Traffic Control)

Software and systems engineering

- ✓ The proportion of software in systems is increasing. Software-driven general purpose electronics is replacing special-purpose systems
- ✓ Problems of systems engineering are similar to problems of software engineering
- ✓ Software is (unfortunately) seen as a problem in systems engineering. Many large system projects have been delayed because of software problems

Problems of systems engineering

- ✓ Large systems are usually designed to solve 'wicked' problems
- ✓ Systems engineering requires a great deal of co-ordination across disciplines

- Almost infinite possibilities for design trade-offs across components
- Mutual distrust and lack of understanding across engineering disciplines

✓ Systems must be designed to last many years in a changing environment

✓ Missed schedule

✓ Improper integration of subsystems

✓ Maintenance problems

✓ Unmanageable systems.

The system engineering process

✓ A set of activities whose goal is the development or evolution of software

✓ Usually follows a 'waterfall' model because of the need for parallel development of different parts of the system

- Little scope for iteration between phases because hardware changes are very expensive. Software may have to compensate for hardware problems

✓ Inevitably involves engineers from different disciplines who must work together

- Much scope for misunderstanding here. Different disciplines use a different vocabulary and much negotiation is required. Engineers may have personal agendas to fulfil

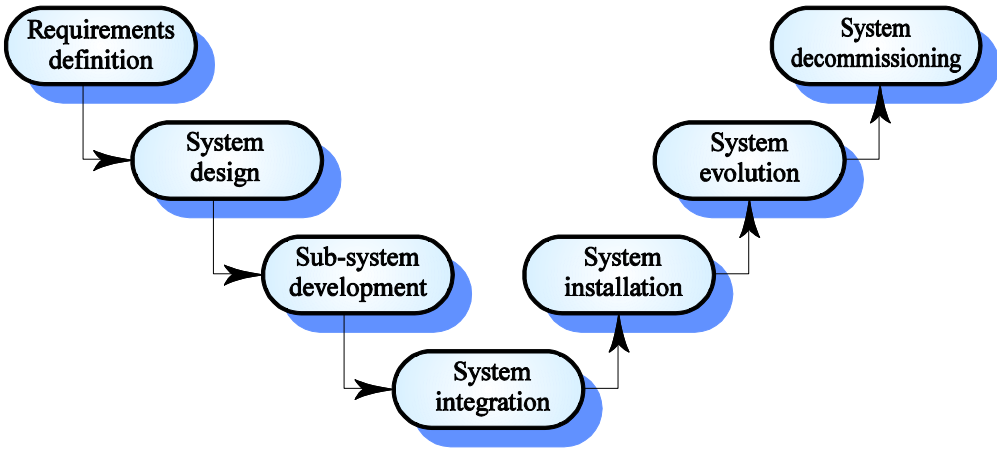


Fig: Phases of the system engineering process

1. System requirements definition

System requirements definitions specify what the system should do (its functions) and its essential and desirable system properties. Creating system requirements definitions involves consultations with system customers and end-users.

This phase concentrates on three types of requirements:

- ✓ **Abstract functional requirements**: The basic function that the system must provide are defined at an abstract level. More detailed functional requirements specification takes place at the sub-system level.
- ✓ **System properties**: These are non-functional emergent system properties such as availability, performance and safety. These properties affect the requirements for all sub-systems.
- ✓ **Characteristics that the system must not exhibit:** It is sometimes as important to specify what the system must not do as it is to specify what the system should do.

  An important part of the requirements definition phase is to establish a set of overall objectives that the system should meet**.** These should not necessarily be expressed in terms of the system's functionality but should define why the system is being procured for a particular environment.

System requirements problems:

- ✓ Changing as the system is being specified
- ✓ Must anticipate hardware/communications developments over the lifetime of the system
- ✓ Hard to define non-functional requirements (particularly) without an impression of component structure of the system

2. The system design process

System design is concerned with how the system functionality is to be provided by the components of the systems. The activities involved in this process are:

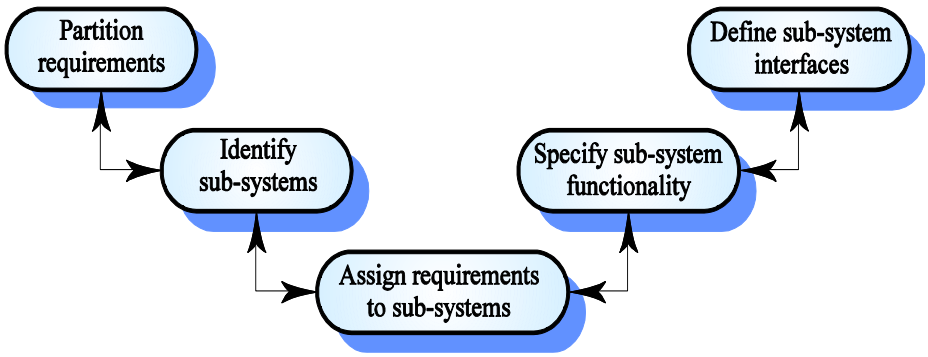


Fig: The system design process

- ✓ **Partition requirements**
  - Organise requirements into related groups
- ✓ **Identify sub-systems**
  - Identify a set of sub-systems which collectively can meet the system requirements

- ✓ **Assign requirements to sub-systems**
    - Causes particular problems when COTS (Commercial Off-the-Shelf)are integrated
- ✓ **Specify sub-system functionality**
    - Specify the specific function provided by each sub-system and also try to identify relationships between subsystems.
- ✓ **Define sub-system interfaces**
    - Critical activity for parallel sub-system development

System design problems

- ✓ Requirements partitioning to hardware, software and human components may involve a lot of negotiation
- ✓ Difficult design problems are often assumed to be readily solved using software
- ✓ Hardware platforms may be inappropriate for software requirements so software must compensate for this

3. Sub-system development

- ✓ Typically, parallel projects developing the hardware, software and communications
- ✓ May involve some COTS (Commercial Off-the-Shelf) systems procurement
- ✓ Lack of communication across implementation teams
- ✓ Bureaucratic and slow mechanism for proposing system changes means that the development schedule may be extended because of the need for rework

4. System integration

- ✓ It is the process of putting hardware, software and people together to make a system
- ✓ It should be tackled incrementally so that sub-systems are integrated one at a time
- ✓ Interface problems between sub-systems are usually found at this stage
- ✓ May be problems with uncoordinated deliveries of system components

5. System installation

- ✓ The organizational process of changing over from the current system to a new one

Four approaches of installation

- **Direct Installation**
  Changing over from the old system to a new one by turning off the old system when the new one is turned on
- **Parallel Installation**

Running the old system and the new one at the same time until management decides the old system can be turned off

- **Single location installation**
  Trying out a system at one site and using the experience to decide if and how the new system should be deployed throughout the organization
- **Phased Installation**
  Changing from the old system to the new one incrementally, starting with one or a few functional components and then gradually extending the installation to cover the whole new system

Problems in installation:

- ✓ Environmental assumptions may be incorrect
- ✓ May be human resistance to the introduction of a new system
- ✓ System may have to coexist with alternative systems for some time
- ✓ May be physical installation problems (e.g. cabling problems)
- ✓ Operator training has to be identified

6. System evolution

- ✓ Large, complex systems have a very long lifetime. They must evolve to meet changing requirements i.e. during their life, they are changed to correct errors in the original system requirements and to implement new requirements that have emerged.
- ✓ System Evolution is inherently costly, like software evolution for several reasons:
  - *Changes must be analysed from a technical and business perspective*
  - *Sub-systems interact so unanticipated problems can arise*
  - *There is rarely a rationale for original design decisions*
  - *System structure is corrupted as changes are made to it*
- ✓ Existing systems which must be maintained are sometimes called **legacy systems**

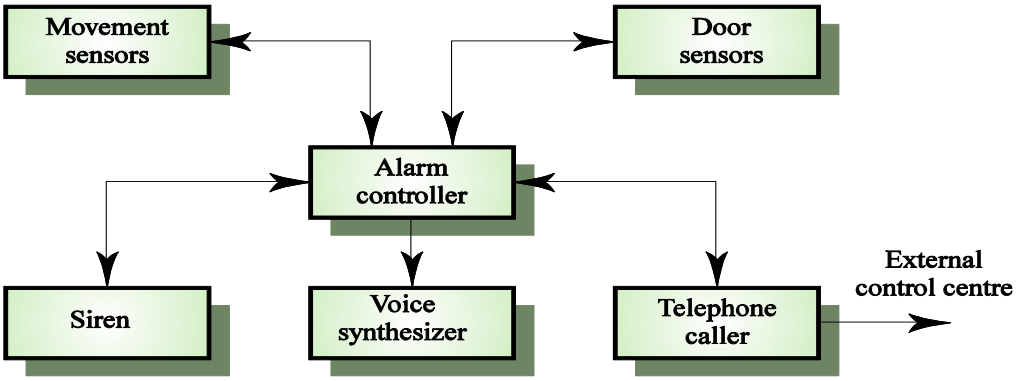
7. System decommissioning

- ✓ It means taking the system out of service after its useful lifetime
- ✓ May require data to be restructured and converted to be used in some other system
- ✓ If the data in the system that is being decommissioned is still valuable to your organization or you may have to convert it for use by other systems.

System modelling:

- ✓ During the system requirements and design activity, systems may be modelled as a set of components and relationships between these components. These are normally illustrated graphically in a system architecture model that gives the reader/user an overview of the system organization.
- ✓ An architectural model presents an abstract view of the sub-systems making up a system
- ✓ May include major information flows between sub-systems
- ✓ Usually presented as a block diagram showing the major sub-systems and interconnections between these subsystems.
- ✓ May identify different types of functional component in the model

e.g. Intruder alarm system



Component types in alarm system

- **Movement Sensor:** Detects movement in the rooms monitored by the system
- **Door Sensors:** detects door opening in the external doors of the building
- **Alarm Controller:** Controls the operation of the system
- **Siren:** Emits an audible warning when an intruder is suspected
- **Voice synthesizer:** Synthesises a voice message giving the location of the suspected intruder
- **Telephone caller:** makes external calls to notify security, the police, etc.

Classes of Information Systems:

- **Transaction processing systems**
  Transaction processing systems are information system applications that capture and process data about business transactions.

- **Management information systems**
  A management information system (MIS) is an information system application that provides for management-oriented reporting. These reports are usually generated on a predetermined schedule and appear in a prearranged format

- **Decision support systems**
  A decision support system (DSS) is an information system application that provides its users with decision-oriented information whenever a decision-making situation arises. When applied to executive managers, these systems are sometimes called executive information systems (EIS).

- **Expert systems**
  An expert system is a programmed decision-making information system that captures and reproduces the knowledge and expertise of an expert problem solver or decision maker and then simulates the "thinking" or "actions" of that expert**.**

- **Office automation systems**
  Office automation (OA) systems support the wide range of business office activities that provide for improved work flow and communications between workers, regardless of whether or not those workers are located in the same office.

System Analyst:

A systems analyst studies the problems and needs of an organization to determine how people, data, processes, communications, and information technology can best accomplish improvements for the business. When information technology is used, the analyst is responsible for:

- ✓ The efficient capture of data from its business source,
- ✓ The flow of that data to the computer,
- ✓ The processing and storage of that data by the computer, and
- ✓ The flow of useful and timely information back to the business and its people.

Task of System Analyst:

1. Identify the problem.

2. Analyze and understand the problem.

3. Identify solution requirements or expectations.

4. Identify alternative solutions and decide a course of action.

5. Design and implement the “best” solution.

6. Evaluate the results. If the problem is not solved, return to step 1 or 2 as appropriate.

Attributes of System analyst:

1. Working Knowledge of Information Technology

- ✓ The systems analyst is an *agent of change*.
- ✓ The systems analyst is responsible for showing end-users and management how new technologies can benefit their business and its operations.
- ✓ The systems analyst must be aware of both existing and emerging information technologies and techniques.

2. Programming Experience and Expertise

- ✓ A systems analyst must know how to program because they are the principle link between business users and computer programmers.
- ✓ It is wrong to assume that a good programmer will become a good analyst or that a bad programmer could not become a good analyst.
- ✓ Most systems analysts need to be proficient in one or more high-level programming languages.

3. Computer Programming Experience and Expertise

- ✓ Historically, the language of choice has been COBOL for business applications, but many organizations are shifting to **visual programming languages** or to **object-oriented programming languages**.
- ✓ The reasons for the shift are as follows:
  - The transition to graphical user interfaces.
  - The desire to downsize applications from the mainframe to networks of PCs.
  - The pressures to improve productivity in applications development through rapid, iterative prototyping and the reuse of programming modules called *objects* and *components*.

Visual and object-oriented programming requires a completely different style of program design, construction, and testing.

4. General Business Knowledge

- ✓ The systems analysts are expected to immerse themselves in the business and be able to specify and defend technical solutions that address the bottom-line value returned to the business.
- ✓ Systems analysts should be able to communicate with business experts to gain knowledge of problems and needs.
- ✓ It is not uncommon for systems analysts to develop so much expertise over time they move out of information systems and into the user community.

5. Problem-Solving Skills

- ✓ The systems analyst must have the ability to take a large business problem, break that problem down into its component parts, analyze the various aspects of the problem, and then assemble an improved system to solve the problem.
- ✓ The systems analyst must learn to analyze problems in terms of causes and effects rather than in terms of simple remedies.

- ✓ The systems analyst must be well organized.
- ✓ System analysts must be able to creatively define alternative solutions to problems and needs.

6. Communications Skills

- ✓ The systems analyst must be able to communicate effectively, both orally and in writing.
- ✓ The systems analyst should have a good command of the English language.
- ✓ Almost without exception, communications skills, not technical skills, prove to be the single biggest factor in career success or failure.

7. Interpersonal Relations Skills

- ✓ Systems work is people-oriented and systems analysts must be extroverted or people-oriented.
- ✓ Interpersonal skills help systems analysts work effectively with people.
- ✓ Interpersonal skills are also important because of the political nature of the systems analyst's job.
- ✓ The systems analyst's first responsibility is to the business, its management, and its workers.
- ✓ The systems analyst must mediate problems between team problems and achieve benefits for the business as a whole.

8. Flexibility and Adaptability

- ✓ No two systems development projects encountered by a systems analyst are identical.
- ✓ There is no single, magical approach or solution applicable to systems development.
- ✓ Successful systems analysts learn to be flexible and adapt to special challenges or situations presented by specific systems development projects.
- ✓ The systems analyst must be able to recognize when variations upon (or single-instance exceptions to) development standards are necessary and beneficial to a particular project.
- ✓ The systems analyst must be aware of the implications of not following the standards.

9. Character and Ethics

- ✓ The nature of the systems analyst's job requires a strong character and sense of ethics.
- ✓ **Ethics** is a personal character trait in which an individual(s) understands the difference between ‘right’ and ‘wrong’ and acts accordingly.

[CSc. 351 Software Engineering]

Lecturer: ***Hiranya Bastakoti***

Amrit Science Campus

3.1 The software process:

- ✓ The software process is a structured set of activities whose goal is development or evolution of software.
- ✓ These activities may involve the development of software from a standard programming language like C, C++or Java. Increasingly, however, new software is developed by extending and modifying existing systems and by configuring and integrating off-the-shelf software or system components.
- ✓ Software processes are complex and, like all intellectual and creative processes, rely on people making decisions and judgements. Because of the need for judgement and creativity, attempts to automatic processes have met with limited success. CASE tools can support some process activities.

Some Fundamental activities which are common to all software process.

- **Software Specification**: The functionality of software and constraints on its operation must be defined
- **Software Design and Implementation**: The software to meet the specifications must be produced.
- **Software Validation**: The software must be validated to ensure that it does what the customer wants
- **Software Evolution**: The software must evolve to meet changing customer needs.

The software Process Model:

- ✓ A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective
- ✓ To solve actual problems in an industry setting, a software engineer or a team of engineers must incorporate a development strategy that encompasses the process, methods, and tools and the generic phases. This strategy is often referred to as a *process model* or a *software engineering paradigm.*
- ✓ A process model for software engineering is chosen based on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required.

Generic software process models

- **The waterfall(Linear) model:** Separate and distinct phases of specification and development
- ✓ **Evolutionary development:** Specification and development are interleaved
- ✓ **Formal systems development:** A mathematical system model is formally transformed to an implementation

- ✓ **Reuse-based development:** The system is assembled from existing components

3.2 Sequential (Waterfall) model:

- ✓ The first published model of the software development process was Waterfall model. It was proposed by Royce in 1970s.
- ✓ Because of the cascade from one phase to another, this model is known as waterfall/software life cycle.
- ✓ This model takes the fundamental process activities of specification, development, validation and evolution and represents them as separate phases such as requirements specification, software **design,** implementation, testing and maintenance**.**

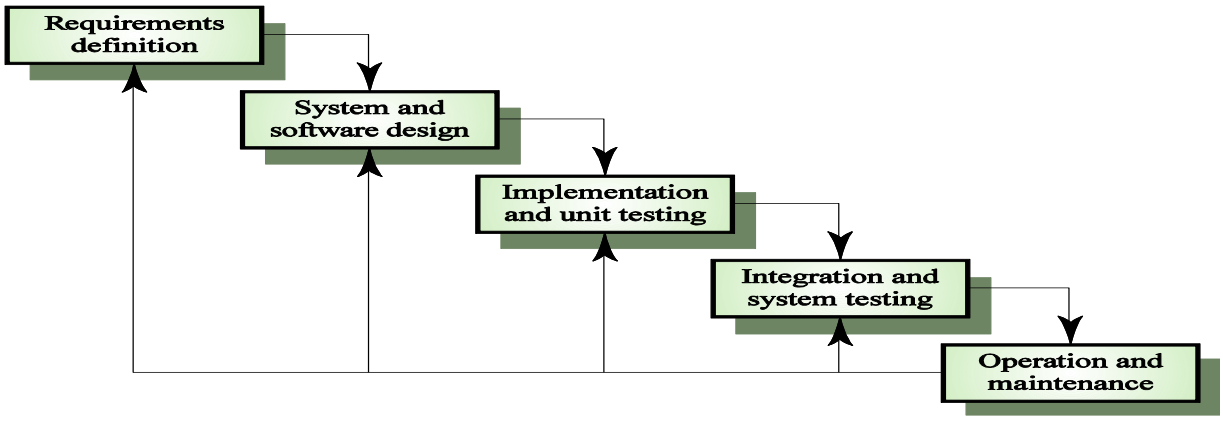


Fig: Waterfall (Sequential) Model

Waterfall model phases are:

- **Requirements analysis and definition:** The system's service, constraints and goals are established by consultation with system users. They are often defined in detail and serve as a system specification.
- **System and software design:** The system design process partitions the requirements to either hardware or software systems. It establishes overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationship
- **Implementation and unit testing:** During this stage, the software design is realised as asset of programs and program units. Unit testing involves verifying that each unit meets its specification.
- **Integration and system testing:** The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer**.**

- **Operation and maintenance:** This is the longest life-cycle phase**.** The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered earlier stages of the life-cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

Advantages:

- ✓ Easy to understand even by non-technical person, i.e customers.
- ✓ Each phase has well defined inputs and outputs.
- ✓ Easy to use as software development proceeds,
- ✓ Each stage has well defined deliverables.
- ✓ Helps the project manager in proper planning of the project.

Waterfall model problems

- ✓ The drawback of the waterfall model is the difficulty of accommodating change after the process is underway because of sequential nature
- ✓ Inflexible partitioning of the project into distinct stages
- ✓ This makes it difficult to respond to changing customer requirements

This model is only appropriate when the requirements are well-understood

3.3 Prototyping Model

- ✓ **Prototyping:** An iterative process of software development in which requirements are converted to a working system that is continually revised through close work between developer and user.

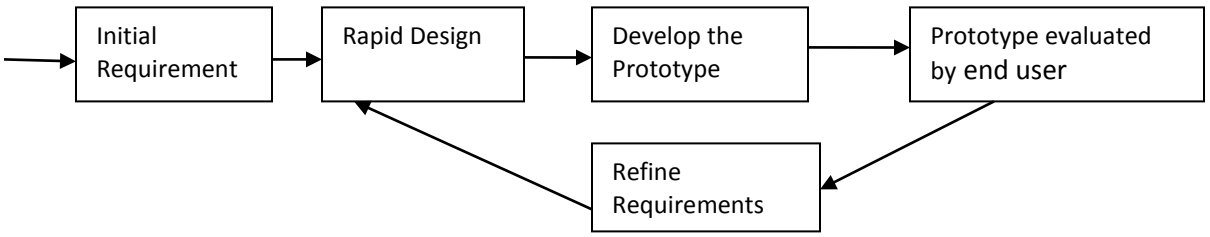


Fig: The Prototype Model

- ✓ Often, a customer defines a set of general objectives for software but does not identify detailed input, processing, or output requirements. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human/machine interaction should take. In these, and many other situations, a *prototyping model may* offer the best approach.
- ✓ The prototyping model begins with requirements gathering. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory. A "quick design" then occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the

customer/user (e.g., input approaches and output formats). The quick design leads to the construction of a prototype. The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.

- ✓ Prototyping model should be used when requirements of the system are not clearly understood or are unstable. It can also be used if requirements are changing quickly. This model can be successfully used for developing user interfaces, high technology software intensive systems, and systems with complex algorithms and interfaces. It is also a very good choice to demonstrate technical feasibility of the product.

Advantages:

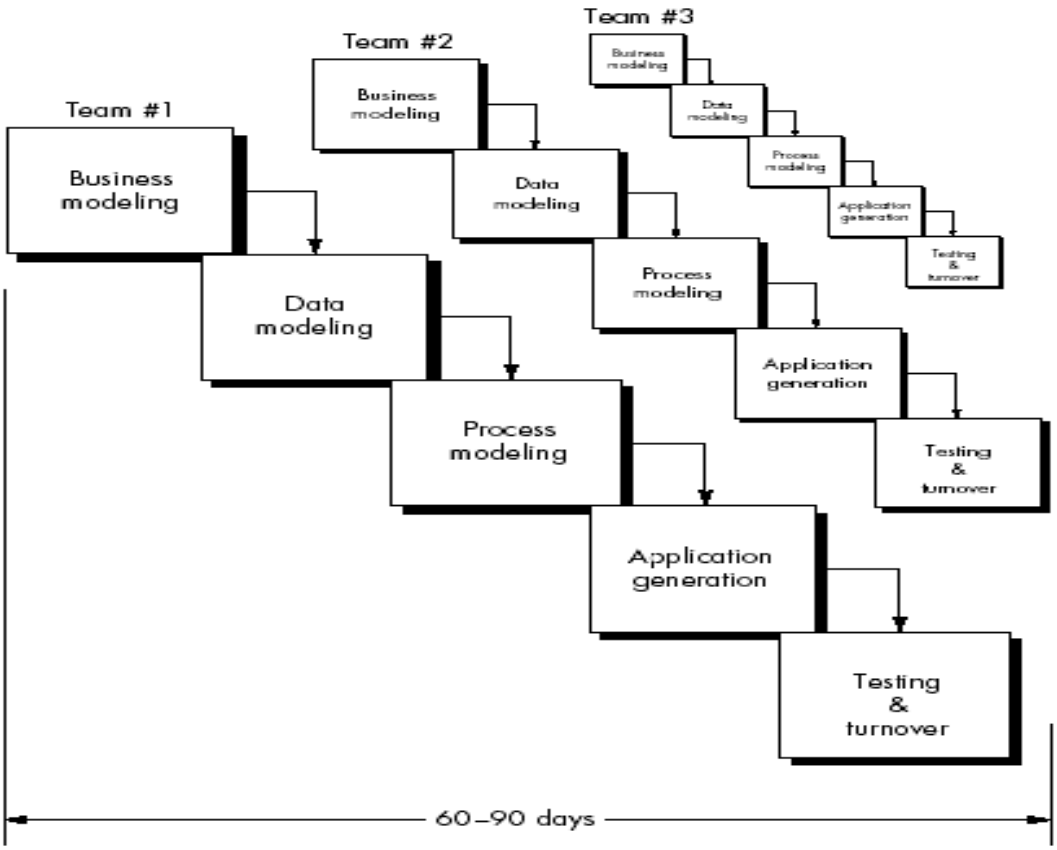
- ✓ A partial product is built in the initial stages. Therefore customers get a chance to see the product early in the life cycle and thus give necessary feedback.
- ✓ New requirements can be easily added
- ✓ Requirements become more clear resulting into an accurate product.
- ✓ As user is involved from the starting of the project, he/she tends to be more secure, comfortable and satisfied.
- ✓ Flexibility in design and development is also supported by the model.

Disadvantages:

- ✓ After seeing an early prototype end users demand the actual system to be delivered.
- ✓ Developers in a hurry to build prototypes may end up with sub-optional solutions.
- ✓ If not managed properly, the iterative process of prototype demonstration and refinement can continue for long duration.
- ✓ If end user is not satisfied with initial prototype, he/she may loose interest in the project.
- ✓ Poor documentation

3.4 Rapid Application Development Model

Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle. The RAD model is a “high-speed” adaptation of the linear sequential model in which rapid development is achieved by using component-based construction. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a “fully functional system” within very short time periods (e.g., 60 to 90 days)



It is used primarily for information systems applications, the RAD approach encompasses the following phases:

Business modeling: The information flow among business functions is modeled in a way that answers the following questions: What information drives the business process? What information is generated? Who generates it? Where does the information go? Who processes it?

Data modeling: The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business. The characteristics (called *attributes*) of each object are identified and the relationships between these objects defined.

Process modeling: The data objects defined in the data modeling phase are transformed to achieve the information flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

Application generation: RAD assumes the use of fourth generation techniques). Rather than creating software using conventional third generation programming languages the RAD process works to reuse existing program components (when possible) or create reusable components (when necessary). In all cases, automated tools are used to facilitate construction of the software.

Testing and turnover: Since the RAD process emphasizes reuse, many of the program components have already been tested. This reduces overall testing time. However, new components must be tested and all interfaces must be fully exercised. Obviously, the time constraints imposed on a RAD project demand

"scalable scope". If a business application can be modularized in a way that enables each major function to be completed in less than three months, it is a candidate for RAD. Each major function can be addressed by a separate RAD team and then integrated to form a whole.

Advantages

- ✓ As customer is involved at all stages of development. It leads to product achieving customer satisfaction**.**
- ✓ Usage of powerful development tools results into reduced software development cycle time.
- ✓ Makes use of reusable components, to decrease the cycle time

Drawbacks of RAD

- ✓ For large but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
- ✓ RAD requires developers and customers who are committed to the rapid-fire activities necessary to get a system complete in a much abbreviated time frame. If commitment is lacking from either constituency, RAD projects will fail.
- ✓ Not all types of applications are appropriate for RAD. If a system cannot be properly modularized, building the components necessary for RAD will be problematic. If high performance is an issue and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work.
- ✓ RAD is not appropriate when technical risks are high. This occurs when a new application makes heavy use of new technology or when the new software requires a high degree of interoperability with existing computer programs.

3.5 Evolutionary Software Process Model

- ✓ The linear sequential model is designed for straight-line development. In essence, this waterfall approach assumes that a complete system will be delivered after the linear sequence is completed.
- ✓ The prototyping model is designed to assist the customer (or developer) in understanding requirements. In general, it is not designed to deliver a production system.
- ✓ **The evolutionary** nature of software is not considered in either of these classic software engineering paradigms.
- ✓ **Evolutionary model** is based on the idea of developing an initial implementation, exposing to this to user comment and refining it through many versions until an adequate system has been developed.
- ✓ Specification, development and validation activities are interleaved rather than separate, with rapid feedback across activities.
- ✓ Evolutionary models are iterative. They are characterized in a manner that enables software engineers to develop increasingly more complete versions of the software.

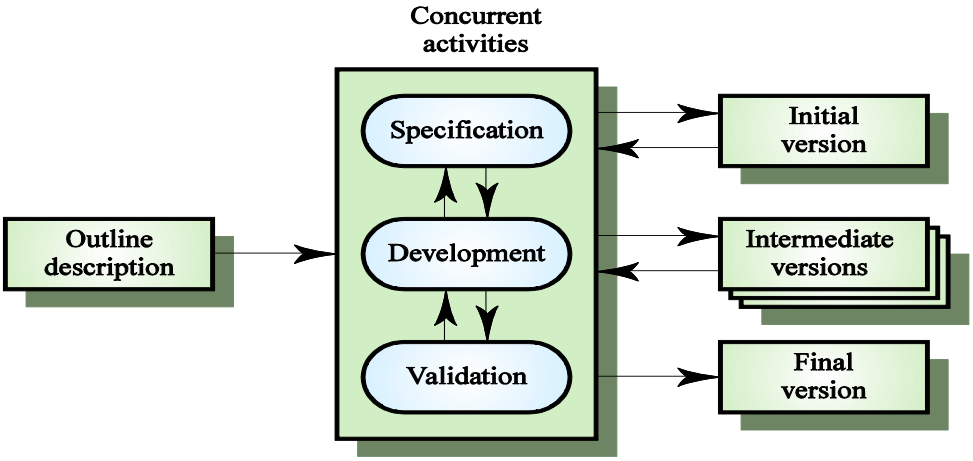


Fig: Evolutionary Development

There are two fundamental types of evolutionary development

- ✓ **Exploratory development**
  - Objective is to work with customers and to evolve a final system from an initial outline specification. Should start with well-understood requirements
- ✓ **Throw-away prototyping**
  - Objective is to understand the system requirements. Should start with poorly understood requirements

Advantages:

- ➢ *An evolutionary approach is often more effective than waterfall approaches in producing systems that meet the immediate needs of customers.*
- ➢ *The advantage of software process that is based on an evolutionary approach is that the specification can be developed incrementally.*
- ➢ *As users develop a better understanding of their problem, this can be reflected in the software system.*

Problems towards engineering and management perspective:

- ➢ ***The process is not visible:*** *Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.*
- ➢ ***Systems are often poorly structured:*** *Continual change tends to corrupt the software structure. Incorporating software changes becomes increasingly difficult and costly.*

3.5.1 Incremental Model

- ✓ In an incremental model, customers identify, in outline, the services to be provided by the system. They identify which of the services are most important and which are least important to them. A number of delivery increments are then defined, with each increment providing a sub-set of the system functionality. The allocation of services to increments depends on the service priority with the highest priority services delivered first.
- ✓ Once the system increments have been identified, the requirements for the services to be delivered in the first increment are defined in detail, and that increment is developed. During development, further requirements analysis for later increments can take place, but requirements changes for the current increment are not accepted.
- ✓ Once increment is completed and delivered, customers can put it into services>they can experiment with the system that helps to clarify their requirements for later increments and for later versions of the current increment. As new increments are completed, they are integrated with existing increments so that the system functionality improves each delivered increment. The common services may be implemented early in the process or may be implemented incrementally as functionality is required by an increment.
- ✓ (In incremental model, rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality
- ✓ User requirements are prioritised and the highest priority requirements are included in early increments
- ✓ Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve)

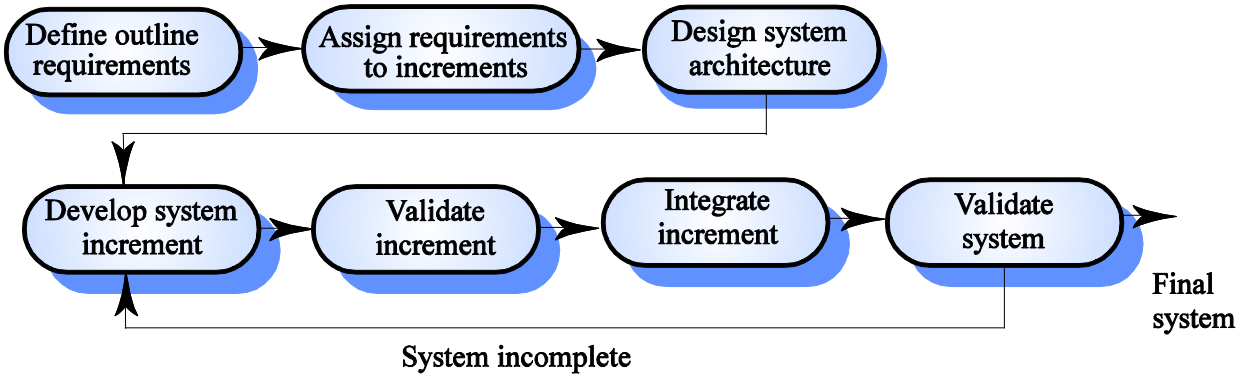


Fig: Incremental delivery

Incremental development advantages

- ✓ Customer value can be delivered with each increment so system functionality is available earlier
- ✓ Early increments act as a prototype to help elicit requirements for later increments
- ✓ Lower risk of overall project failure

- ✓ The highest priority system services tend to receive the most testing

Variant in this approach called Extreme programming:

- ✓ New approach to development based on the development and delivery of very small increments of functionality
- ✓ Relies on constant code improvement, user involvement in the development team and pairwise programming

Disadvantages:

- ✓ As product is delivered in parts, total development cost is higher.
- ✓ Well defined interfaces are required to connect modules developed with each phase.
- ✓ Testing of modules also results into overhead and increased cost.

3.5.2 Spiral development

- ✓ The spiral model was originally proposed by Boehm in 1988.
- ✓ In this model, process is represented as a spiral rather than as a sequence of activities with backtracking from one activity to another.
- ✓ Each loop in the spiral represents a phase in the process.
- ✓ No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required
- ✓ Risks are explicitly assessed and resolved throughout the process

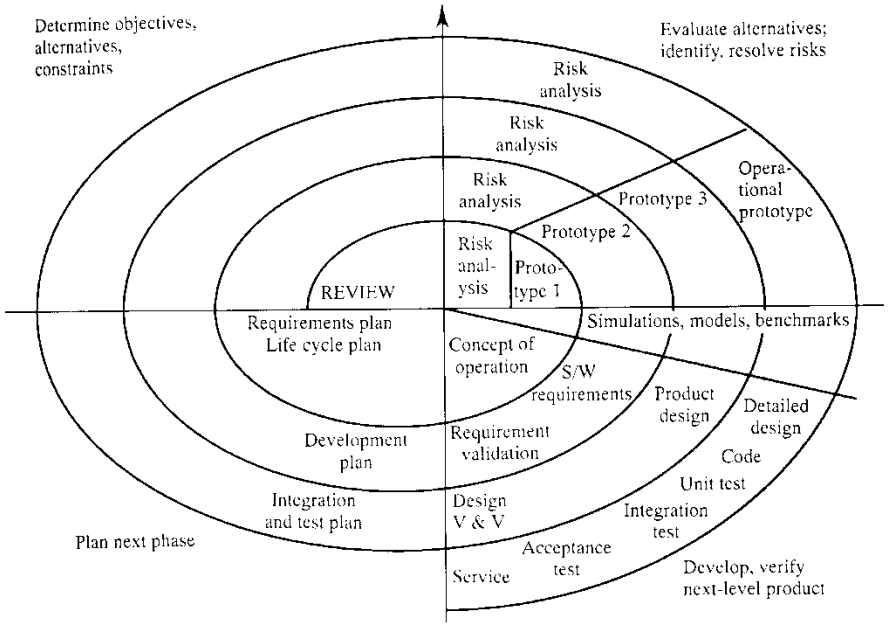


Fig:Spiral model

Spiral model sectors are

- ✓ **Objective setting: (**Specific objectives for the phase are identified)
  - In this sector, specific objectives are defined. Constraints on the process and product are identified and a detailed management plan is drawn up. Project risks are identified. Alternative strategies, depending on these risks, may be planned.
- ✓ **Risk assessment and reduction:** Risks are assessed and activities put in place to reduce the key risks)
  - For each of the identified project risks, a detailed analysis is carried out. Steps are taken to reduce the risk. For example, if there is risk that requirement are inappropriate, a prototype system may be developed.
- ✓ **Development and validation:** A development model for the system is chosen which can be any of the generic models
  - After risk evaluation, a development model for the system is chosen which can be any generic model. For example. the waterfall model may be the most appropriate development model if the main identified risk is sub-system integration
- ✓ **Planning: (**The project is reviewed and the next phase of the spiral is planned)
  - The project is reviewed and a decision made whether to continue with a further loop of the spiral. If it is decided to continue, plans are drawn up for the next phase of the project.

Advantages:

- ✓ The model tries to resolve all possible risks involved in the project starting with the highest risk.
- ✓ End users get a chance to see the product early in life cycle.
- ✓ With each phase as product is refined after customer feedback, the model ensures a good quality product.
- ✓ The model makes use of techniques like reuse, prototyping and component based design

Disadvantages:

- ✓ The model requires expertise in risk management and excellent management skills.
- ✓ This model is not suitable for small projects as cost of risk analysis may exceed the actual cost of the project.
- ✓ Different persons involved in the project may find it complex to use.

Computer-Aided Software Engineering (CASE):

Computer-aided software engineering (CASE) tools are software programs that automate or support the drawing and analysis of system models and provide for the translation of system models into application programs. Some CASE tools also provide prototyping and code generation capabilities.

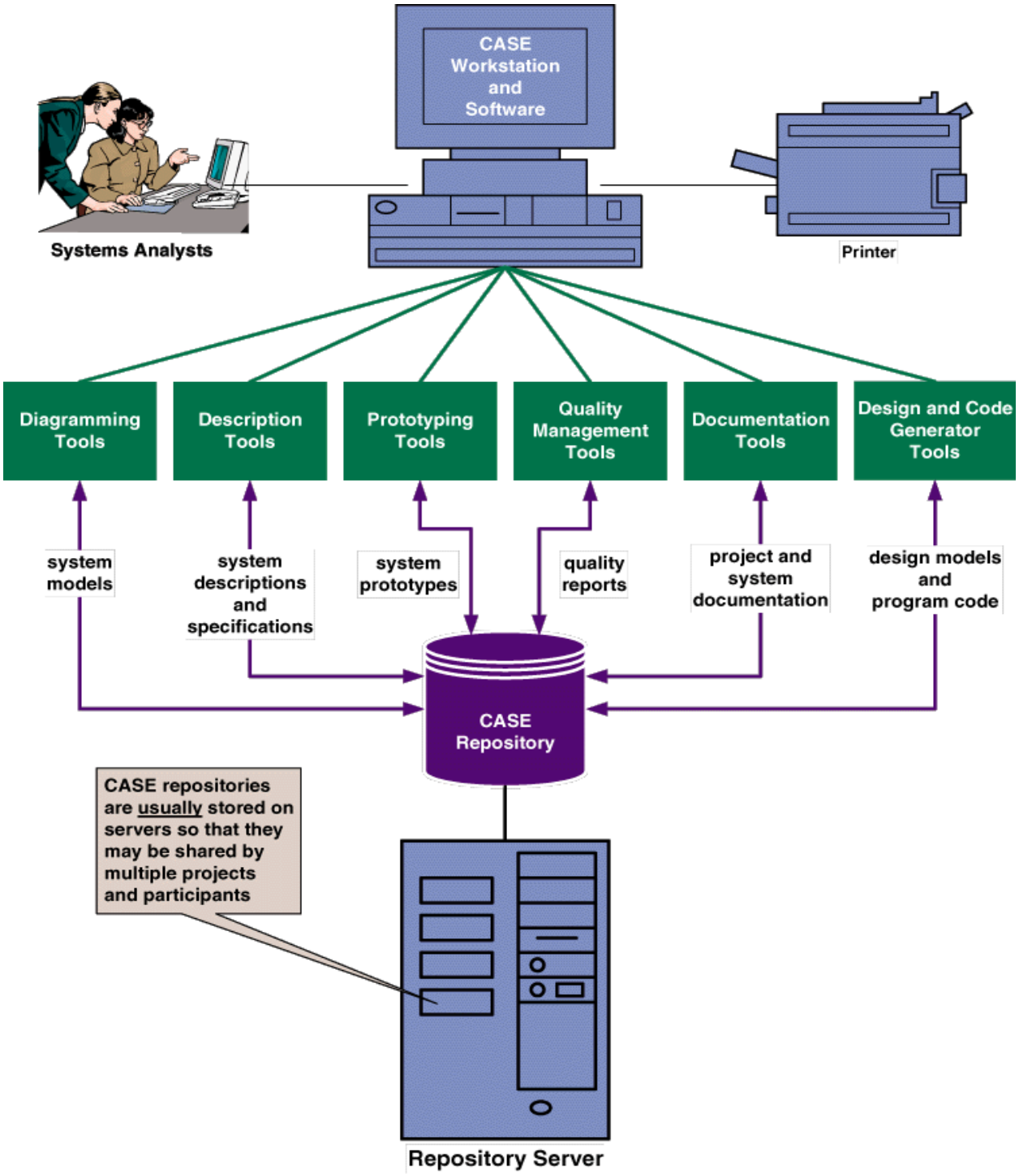
A **CASE repository** is a system developers' database. It is a place where developers can store system models, detailed descriptions and specifications, and other products of system development. Synonyms include **dictionary** and **encyclopedia**.

CASE Facilities: To use the repository, the CASE tools provide some combination of the following facilities.

- ✓ **Diagramming tools**: are use to draw the system models required or recommended in most system development methodologies. Usually, the shapes on one system model can be linked to other models and to detailed descriptions.
- ✓ **Dictionary tools**: are used to record, delete, edit, and output detailed documentation and specifications. The descriptions can be associated with shapes appearing on system models that were drawn with diagramming tools.
- ✓ **Design tools**: can be used to develop system components such as inputs and outputs.
- ✓ **Quality management tools**: analyze system models, descriptions and specifications, and designs for completeness, consistency, and conformance to accepted rules of the methodologies.
- ✓ **Documentation tools:** are used to assemble, organize, and report on system models, descriptions and specifications, and prototype that can be reviewed by system owners, users, designers, and builders.
- ✓ **Design and code generator tools:** automatically generate database designs and application programs or significant portions of those programs.

Forward engineering requires the systems analyst to draw system models, either from scratch or from templates. The resulting models are subsequently transformed into program code

Reverse engineering allows a CASE tool to read existing program code and transform that code into a representative system model that can be edited and refined by the systems analyst.


CASE Workstation and Software
Systems Analysts
Printer
Diagramming Tools
Description Tools
Prototyping Tools
Quality Management Tools
Documentation Tools
Design and Code Generator Tools
system models
system descriptions and specifications
system prototypes
quality reports
project and system documentation
design models and program code
CASE Repository
CASE repositories are usually stored on servers so that they may be shared by multiple projects and participants
Repository Server

[CSc. 351 Software Engineering]

Lecturer: ***Hiranya Bastakoti***

Amrit Science Campus

Introduction:

Project:

- ✓ A project is a temporary endeavor undertaken to provide a unique product or service
- ✓ A project is a (temporary) sequence of unique complex and connected activities that have one goal or purpose and that must be completed by a specific time, within budget and according to specification.

OR

- ✓ A project is a sequence of activities that must be completed on time, within budget and according to specification.

Project management

- ✓ Project management is the process of scoping, planning, staffing, organizing, directing and controlling the development of an acceptable system at a minimum cost within a specified time frame.

A project is considered as a success if:

- ✓ The resulting information system is acceptable to the customer.
- ✓ The system is delivered "on time".
- ✓ The system is delivered "within budget".
- ✓ The system development process had minimal impact ongoing business operations.

A project is said to be failed if:

- ✓ Failure to establish upper management commitment to the project.
- ✓ Lack of organization's commitment to the system development method.
- ✓ Taking shortcuts through or a around the system development method
- ✓ The project gets behind schedule.
- ✓ The project is over budget.
- ✓ The team is not trained or skilled.
- ✓ Poor planning
- ✓ Lack of quality standards.
- ✓ Lack of communication between end users and developers.
- ✓ Changing requirements.

Software project management:

- ✓ It is concerned with activities involved in ensuring that software is delivered on time and on schedule and in accordance with the requirements of the organisations developing and procuring the software.
- ✓ Project management is needed because software development is always subject to budget and schedule constraints that are set by the organisation developing the software.

Software management distinctions:

- ✓ *Software engineering is different from other types of engineering in a number of ways. These distinctions make software management particularly difficult. some of differences are:*
  - ➢ The product is intangible: it cannot be seen or touched. Software mangers cannot see progress.
  - ➢ The product is uniquely flexible.

- Software engineering is not recognized as an engineering discipline with the same status as mechanical, electrical engineering, etc.
- The software development process is not standardised: process may vary dramatically from one organization to another.
- Many software projects are 'one-off' projects: rapid technological changes in computers and communications can make a manager's experience obsolete (out dated). Lessons learned from previous projects may not be transferable to new projects.

Software Manager:

- ✓ Software managers are responsible for planning and scheduling project development. They supervise the work to ensure that it is carried out to the required standards and monitor progress to check that the development is on time and within budget.
- ✓ The software project manager's job is to ensure that the software project meets constraints (Budget and schedule constraints) and delivers software that contributes to the goals of the company developing the software.

Common activities software managers include:

- Management
- Leadership
- Technical problem solving
- Problem solving
- Conflict management
- Customer relations
- Team management
- Risk and change management

Management activities:

- ✓ Proposal writing:
- ✓ Project planning and scheduling.
- ✓ Project costing.
- ✓ Project monitoring and reviews.
- ✓ Personnel selection and evaluation.
- ✓ Report writing and presentations.

Management commonalities

- ✓ These activities are not peculiar to software management.
- ✓ Many techniques of engineering project management are equally applicable to software project management.
- ✓ Technically complex engineering systems tend to suffer from the same problems as software systems.

Project staffing

- ✓ May not be possible to appoint the ideal people to work on a project
  - Project budget may not allow for the use of highly-paid staff;
  - Staff with the appropriate experience may not be available;
  - An organisation may wish to develop employee skills on a software project.
- ✓ Managers have to work within these constraints especially when there are shortages of trained staff.

Project planning

- ✓ It involves making detailed plan to achieve the objectives.
- ✓ Probably the most time-consuming project management activity.

- ✓ Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
- ✓ Various different types of plan may be developed to support the main software project plan that is concerned with schedule and budget.

Types of plan:

- **Quality Plan:** It describes quality procedures and standards that will be used in a project**.**
- **Validation Plan:** It describes the approach, resources and schedules used for system validation.
- **Configuration management plan:** it describes the configuration management procedures and structures to be used.
- **Maintenance plan:** It predicts the maintenance requirements of the system maintenance costs and effort required.
- **Staff development plan:** It describes how the skills and experience of the project team members will be developed.

Project planning process

```
Establish the project constraints
Make initial assessments of the project parameters
Define project milestones and deliverables
While project has not been completed or cancelled loop
        Draw up project schedule
        Initiate activities according to schedule
        Wait (for a while)
        Review project progress
        Revise estimates of project parameters
        Update the project schedule
        Re-negotiate project constraints and deliverables
        if (problems arise) then
                Initiate technical review and possible revision
        end if
end loop
```

The project plan

- ✓ *The project plan sets out:*
  - The resources available to the project;
  - The work breakdown;
  - A schedule for the work.
- ✓ The details of the project plan vary depending on the type of project and organization. However, most plans should include the following sections:

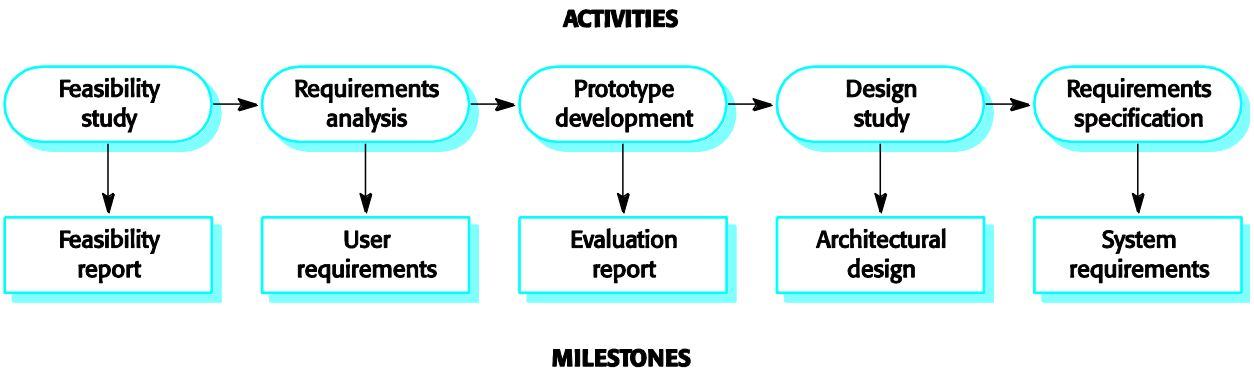
1. **Introduction:** This briefly describes the objectives of the project and sets out the constraints (e.g. budget, time etc.) that affect the project management
2. **Project organization:** This describes the way in which the development team is organized, the people involved and their roles in the team.

3. **Risk analysis:** This describes possible project risks, the likelihood of these risks arising and the risk reduction strategies that proposed**.**
4. **Hardware and software resource requirements:** This specifies the hardware and the support software required to carry out the development. If hardware has to be bought, estimate of the prices and delivery schedule may be included.
5. **Work breakdown:** This sets out the breakdown of the project into activities and identifies the milestones and deliverables associated with each activity.
6. **Project schedule:** This shows the dependencies between activities, the estimated time required to reach each milestone and the allocation of people to activities.
7. **Monitoring and reporting mechanisms:** This defines the management reports that should be produced, when these should be produced and the project monitoring mechanisms used.

Milestones and Deliverables:

- Activities in a project should be organised to produce tangible outputs for management to judge progress.
- *Milestones* are the end-point of a process activity.
- *A project milestone is a predictable state where a formal report of progress is presented to management.*
- *Deliverables are project results delivered to customers. It is usually delivered at the end of some major phase such as specification or design. Deliverables are usually milestones, but milestones need not be deliverables.*
- *Milestones may be internal project results that are used by the project manager to check project delivered to the customer*
- The waterfall process allows for the straightforward definition of progress milestones.
- To establish milestones, the software process must be broken down into basis activities with associated outputs. The fig (below) shows possible activities involved in requirement specification when prototyping is used to help validate requirements. The milestones in this case are completion of the outputs for each activity. The project deliverables, which are delivered to the customer, are the requirements definition and the requirements specification.

Milestones in the RE process



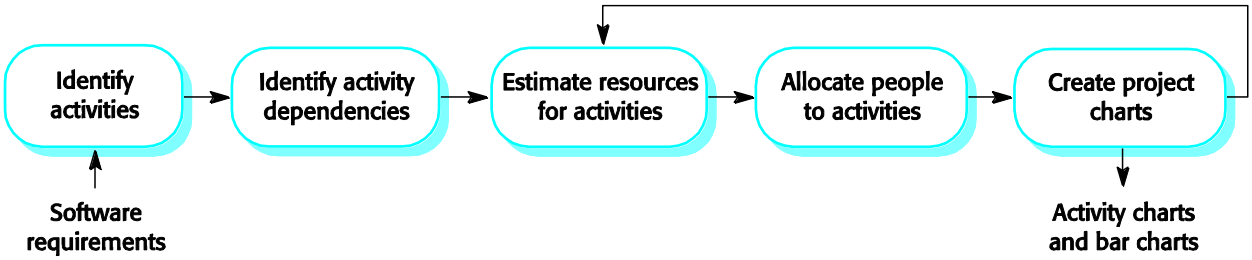
Project scheduling

- It is the one of the most difficult job for a project manager. managers estimate time and resources required to complete activities and organize them into coherent sequence.

It involves:

- Split project into tasks and estimate time and resources required to complete each task.
- Organize tasks concurrently to make optimal use of workforce.
- Minimize task dependencies to avoid delays caused by one task waiting for another to complete.
  It's Dependent on project managers intuition and experience.
- ✓ *Project scheduling involves preparing various graphical representations showing project activities, their durations and staffing.*

The project scheduling process



Scheduling problems

- Estimating the difficulty of problems and hence the cost of developing a solution is hard.
- Productivity is not proportional to the number of people working on a task.
- Adding people to a late project makes it later because of communication overheads.
- The unexpected always happens. Always allow contingency in planning.

Bar charts and activity networks

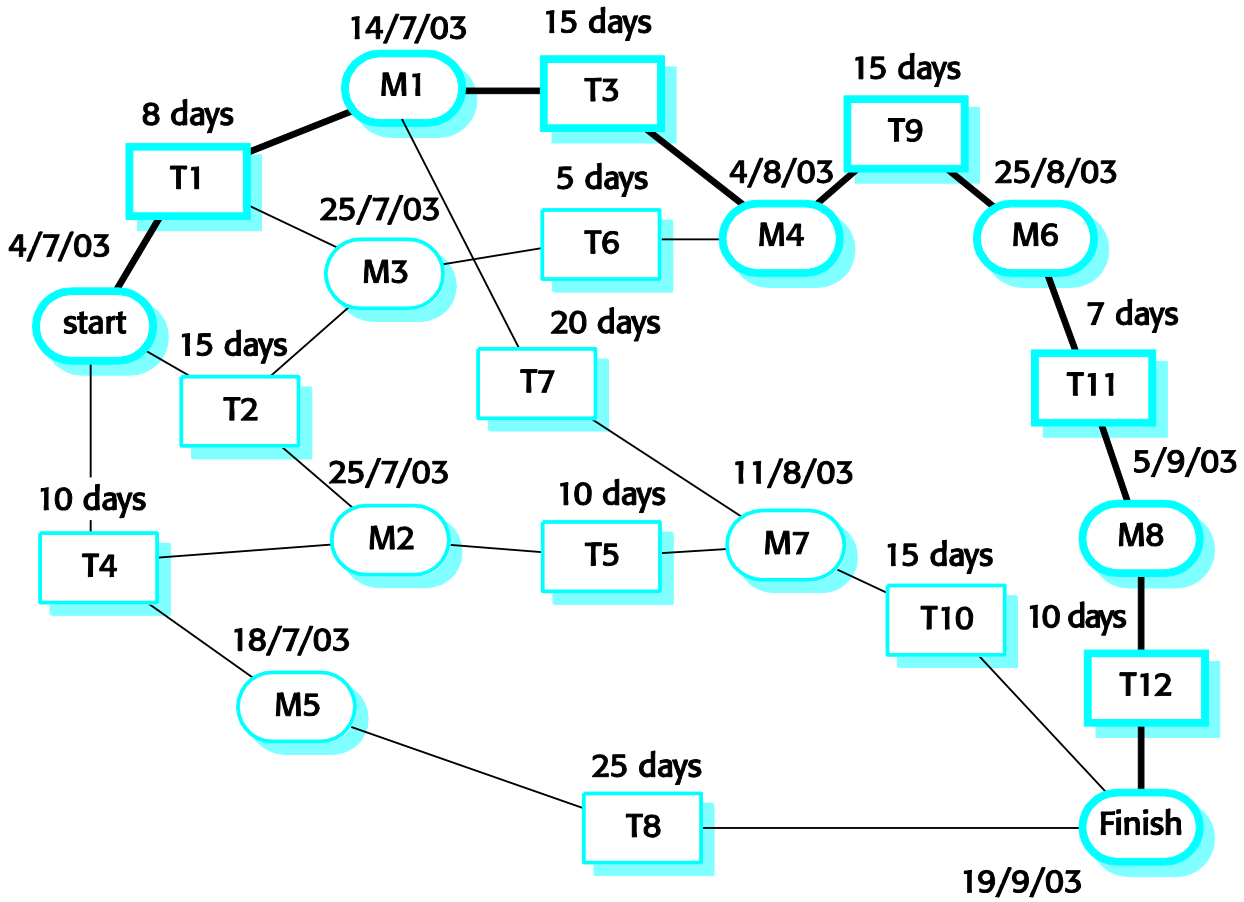
- Graphical notations used to illustrate the project schedule.
- Show project breakdown into tasks. Tasks should not be too small. They should take about a week or two.
- Activity charts show task dependencies and the critical path.
- Bar charts show schedule against calendar time.

Task durations and dependencies

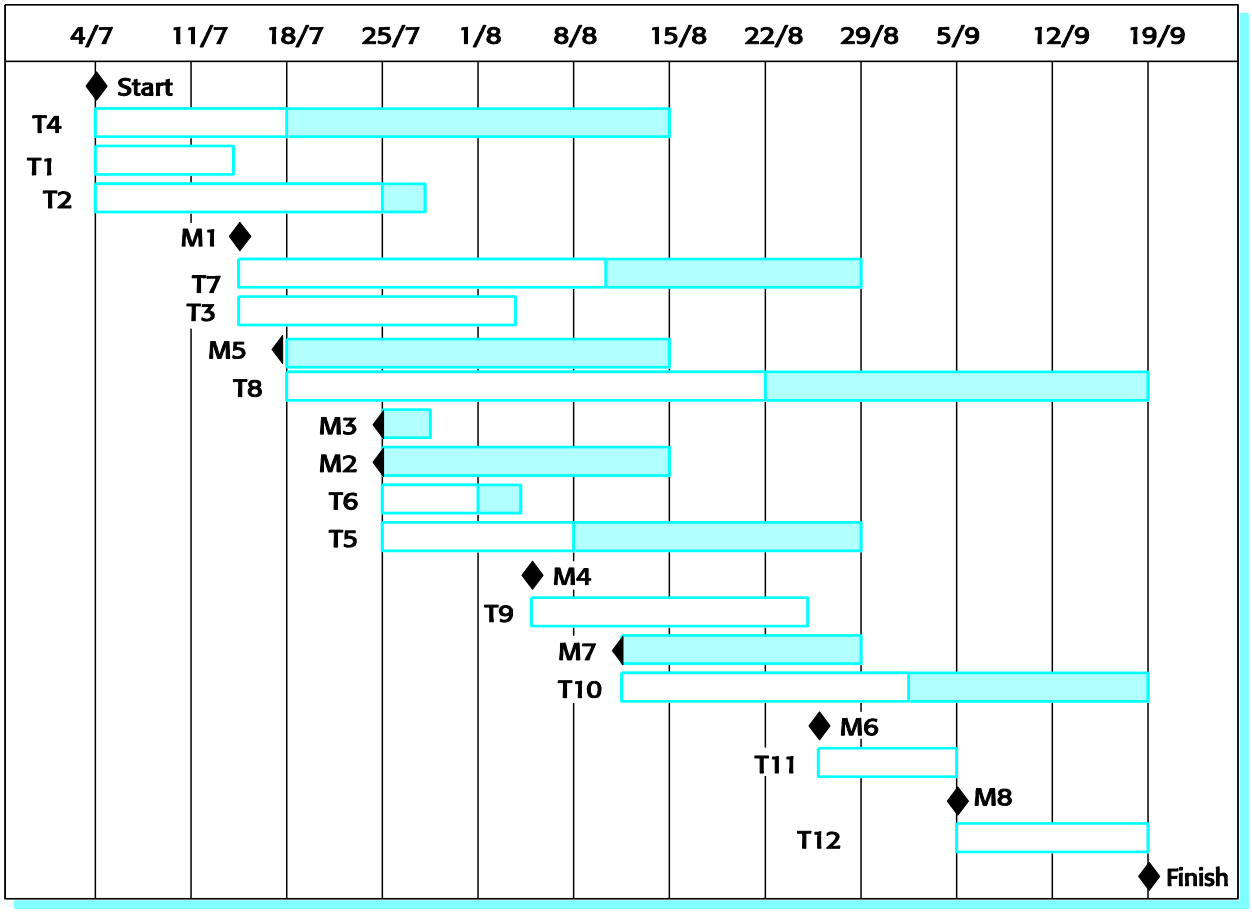
Activity	Duration (days)	Dependencies
T1	8	
T2	15	
T3	15	T1 (M1)
T4	10	
T5	10	T2, T4 (M2)
T6	5	T1, T2 (M3)
T7	20	T1 (M1)
T8	25	T4 (M5)
T9	15	T3, T6 (M4)
T10	15	T5, T7 (M7)
T11	7	T9 (M6)
T12	10	T11 (M8)

Note-milestone

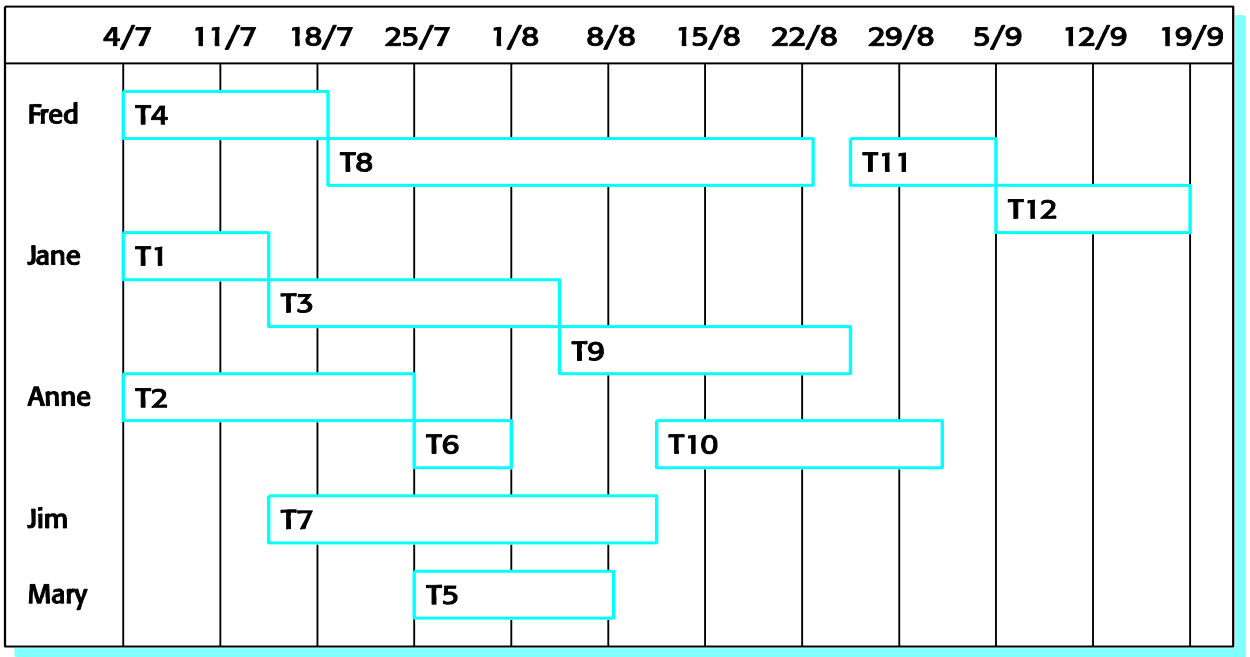
An Activity Network



Activity bar chart(Gantt chart)



Staff allocation vs.Timechart



Risk management

- Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.
- Risk management is concerned with identifying risks which may affect the project and planning to ensure that these risks do not develop into major threats
- A risk is a probability that some adverse circumstance will occur
    - **Project risks** affect schedule or resources.eg.loss of experienced designer
    - **Product risks** affect the quality or performance of the software being developed. e.g. failure of purchased component to perform an expected.
    - **Business risks** affect the organisation developing or procuring the software.e.g,a competitor introducing a new product is a business risk.

Software risks:

The risks may be affect a project depend on the project and the organizational environment where software is being developed.

Risk	Affects	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organisational management with different priorities.
Hardware unavailability	Project	Hardware that is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule
Size underestimate	Project and product	The size of the system has been underestimated.
CASE tool under-performance	Product	CASE tools which support the project do not perform as anticipated
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

The risk management process:

- **Risk identification**
    - Identify project, product and business risks;
- **Risk analysis**
    - Assess the likelihood and consequences of these risks;
- **Risk planning**
    - Draw up plans to avoid or minimise the effects of the risk;
- **Risk monitoring**
    - Monitor the risks throughout the project;

The risk management process, like other project planning is an iterative process which continues throughout the project. Once an initial set of plans are drawn up, the situation is monitored. As more

information about the risks becomes available, the risks have to be reanalyzed and new priorities established. The risk avoidance and contingency plans may be modified as new risk information emerges.

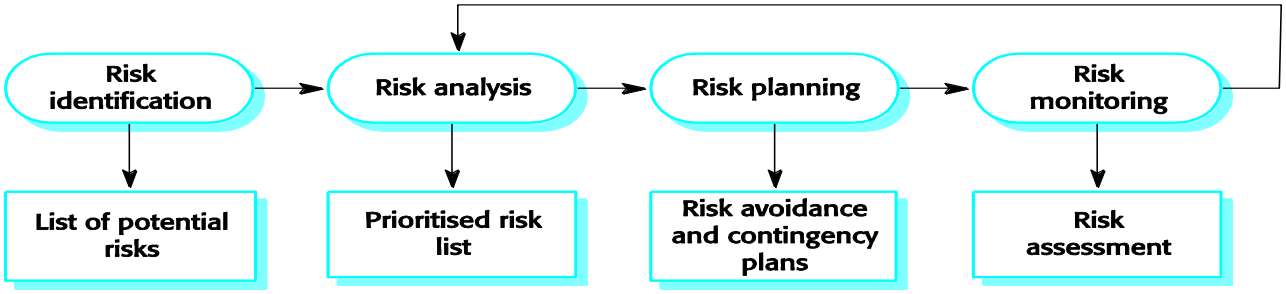


Fig: **The risk management process**

Risk identification:

It is the first stage of risk management. It is concerned with discovering possible risks to the project. Risk identification may be carried out as a team process using a brainstorming approach or may simply be based on experience. There are si x types of risk that can arise:

- **Technology risks.**
- **People risks.**
- **Organisational risks.**
- **Requirements risks.**
- **Estimation risks.**

Risk type	Possible risks
Technology	The database used in the system cannot process as many transactions per second as expected. Software components that should be reused contain defects that limit their functionality.
People	It is impossible to recruit staff with the skills required. Key staff are ill and unava ilable at critical times. Required training for staff is not available.
Organisational	The organisation is restructured so that different management are responsible for the project. Organisational financial problems force reductions in the project budget.
Tools	The code generated by CASE tools is inefficient. CASE tools cannot be integrated.
Requirements	Changes to requirements that require major design rework are proposed. Customers fail to understand the impact of requirements change s.
Estimation	The time required to deve lop the software is underestimated. The rate of defect repair is underestimated. The size of the software is underestimated.

Risk analysis

- ✓ During risk analysis process, assess probability and seriousness of each risk.
- ✓ Probability may be very low, low, moderate, high or very high.
- ✓ Risk effects might be catastrophic, serious, tolerable or insignificant.

Risk	Probability	Effects
Organisational financial problems force reductions in the project budget	Low	Catastrophic
It is impossible to recruit staff with the skills required for the project.	High	Catastrophic
Key staff are ill at critical times in the project.	Moderate	Serious
Software components that should be reused contain defects which limit their functionality.	Moderate	Serious
Changes to requirements that require major design rework are proposed.	Moderate	Serious
The organisation is restructured so that different management are responsible for the project.	High	Serious

Risk	Probability	Effects
Organisational financial problems force reductions in the project budget	Low	Catastrophic
It is impossible to recruit staff with the skills required for the project.	High	Catastrophic
Key staff are ill at critical times in the project.	Moderate	Serious
Software components that should be reused contain defects which limit their functionality.	Moderate	Serious
Changes to requirements that require major design rework are proposed.	Moderate	Serious
The organisation is restructured so that different management are responsible for the project.	High	Serious

Risk planning

- The risk planning process considers each of the key risks that have been identified and identifies strategies to manage the risk
- Consider each risk and develop a strategy to manage that risk.
- **Avoidance strategies**
  - The probability that the risk will arise is reduced. e.g.: Defective components
- **Minimisation strategies**
  - The impact of the risk on the project or product will be reduced**.** e.g.: Staff illness

- **Contingency plans**
    - If the risk arises, contingency plans are plans to deal with that risk. Organizational financial problems

Risk management strategies:

Risk	Strategy
Organisational financial problems	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Recruitment problems	Alert customer of potential difficulties and the possibility of delays, investigate buying-in components.
Staff illness	Reorganise team so that there is more overlap of work and people therefore understand each other's jobs.
Defective components	Replace potentially defective components with bought-in components of known reliability.

Risk	Strategy
Requirements changes	Derive traceability information to assess requirements change impact, maximise information hiding in the design.
Organisational restructuring	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Database performance	Investigate the possibility of buying a higher-performance database.
Underestimated development time	Investigate buying in components, investigate use of a program generator

Risk monitoring

- It involves regular assess each identified risks regularly to decide whether or not it is becoming less or more probable.
- Also assess whether the effects of the risk have changed.
- Each key risk should be discussed at management progress meetings.
- Risk monitoring should be a continuous process, and, at every management progress review, we should consider and discuss each of the key risks separately

Risk indicators

Risk type	Potential indicators
Technology	Late delivery of hardware or support software, many reported technology problems
People	Poor staff morale, poor relationships amongst team member, job availability
Organisational	Organisational gossip, lack of action by senior management
Tools	Reluctance by team members to use tools, complaints about CASE tools, demands for higher-powered workstations
Requirements	Many requirements change requests, customer complaints
Estimation	Failure to meet agreed schedule, failure to clear reported defects

Key points

- ✓ Good project management is essential for project success.
- ✓ The intangible nature of software causes problems for management.
- ✓ Managers have diverse roles but their most significant activities are planning, estimating and scheduling.
- ✓ Planning and estimating are iterative processes which continue throughout the course of a project.
- ✓ A project milestone is a predictable state where a formal report of progress is presented to management.
- ✓ Project scheduling involves preparing various graphical representations showing project activities, their durations and staffing.
- ✓ Risk management is concerned with identifying risks which may affect the project and planning to ensure that these risks do not develop into major threats

Planning

A plan is a predetermined course of action. It represents goals ad the activities to achieve these goals.

Planning is an ongoing organizational function that provides the framework for operational activities and decision making. The organizational mission is translated into operational objectivities through an organizational hierarchy of planning activities.

Planning focuses the energies and activities of the organization on achievement of its objectives, to reconcile differences in objectives and plans of subareas and individuals within the organization and to remove ambiguities about what the organization should do. The formal plan not only guides activities. It provides a basis for evaluation results. The planning process can result in significant motivation for individual and organizational achievement but there are also individual and organizational forces opposed to planning.

Reason for planning system.

1. To offset uncertainty:
   Aside from the uncertainty of business operations and the resulting need for better forecasting information, the special need for a system plan is evident because of advancing computer technology and its widespread effect on business operations. Both s/w and h/w have become so complete that the job of selection and utilization is much more difficult. As a result, the majority of organizations have fallen for short of their potential to use computers for processing the information necessary to manage the company effectively.
   A master plan may not remove the uncertainty, but it will almost surely place the firm in a better position to deal with the unknowns and to take advantage of development as they occur.
2. To improve economy of operations:
   Planning the overall approach to an integrated system is also economical when one job or function is automated; the need for design and automation of contiguous functions frequently becomes obvious. Money can be saved and performance improved by an effective linking together of these neighboring functions through a good plan for integrated system design.
3. To focus on objectives:
   A good plan for system development also serves to focus on company and system objectives. Planning cannot proceed in any area of endeavor until adequate objectives have been first set. It follows that development of a master system plan forces examination and definition of objectives.
4. To provide a device for control of operations:
   Control: control is the activity which measures deviation from planned performance and initiates corrective action. System development, implementation and operations are among the most difficult of activities within the company to control. A major advantage of the development of system effort under a predetermined plan is that the plan provides a means for subsequent plan.

Project plans activities:

The following items should certainly be included in every project plan:

1. Negotiate scopes:

Scope defines the boundaries of a project and included in the statement of work, a narrative description of the work to be performed as part of a project.

All parties must agree to the project scope before any attempt is made to identify and schedule tasks or to assign resources (people) to those tasks.

2. Identity tasks:

Tasks identify the work to be done. Typically, this work is defined in a top-down, outline manner. A work breakdown structure (WBS) is a hierarchical decomposition of the project into tasks and sub-tasks. Some tasks represent the completion of milestones or the completion of major deliverables during a project.

3. Estimate task duration:

Duration of any tasks is a random variable subject to factors such as the size of team, number of users, availability of users, aptitudes of users, complexity of the business system, information technology architecture, experiences of team personal, time committed to other projects, and experiences with other projects.

4. Given the duration estimates for all tasks, we can begin to develop a project schedule. The project schedule depends not only on task duration but also on intertask dependencies.

The start or completion of individual tasks may be dependent on the start or completion of other tasks. These dependencies impact the completion of any project.

There are four types of intertask dependencies:

Finish-to-start (FS) - the finish of one task triggers the start of another task.

Start-to-start (SS) - the start of one task triggers the start of another task.

Finish-to-finish (FF) – two tasks must finish at the same time.

Start-to-finish (SF) – the start of one task signifies the finish of another task.

5. Assign resources:

The following resources may impact a project schedule:

People, services, facilities and equipment, supplies and materials and money.

6. One of the most important dimensions of directing the team effort is the supervision of people.
7. Assess project result and experiences:

This final activity involves soliciting feedback from project team members (including customers) concerning their project experiences and suggestions aimed at improving the project (and process) management of the organization.

Planning techniques:

1. Work breakdown structure:

A fundamental concept in project management is the work breakdown structure (WBS). A work breakdown structure is a hierarchical decomposition of the project into phases, activities, and tasks.

Or

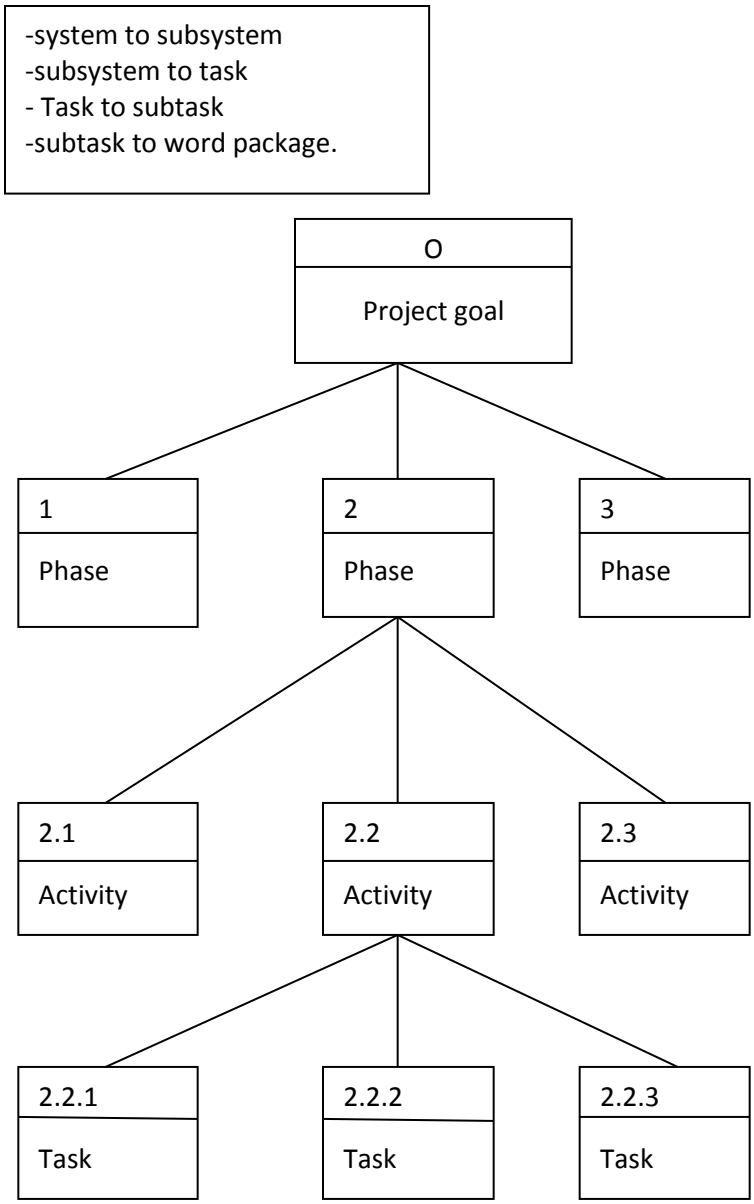


FIG: A graphical work breakdown structure.

Or

1 Phase 1 of the project

1.1 Activity 1 of the phase 1

1.1.1 Task 1 of the activity 1 in the phase 1

1.1.2 Task 2 of activity 1 in the phase 1.

1.2 Task 2 of phase 1.

2 phase 2 of the project.

Gantt chart:

A Gantt chart depicts an overall picture of events and schedule of each task. It lists activities vertically downwards on the left most column while the time-periods in months, weeks or days, appear horizontally in the topmost row.

I D	Task Name	may	june	july	aug	sept	octo	nov	dece
1	Problem analysis								
2	Requirement analysis								
3	Logical design								
4	Decision analysis								
5	Physical design								
6	Construction & testing								
7	Implementation & testing								

Today

Legends
Complete task
Incomplete task

Fig: Gantt chart

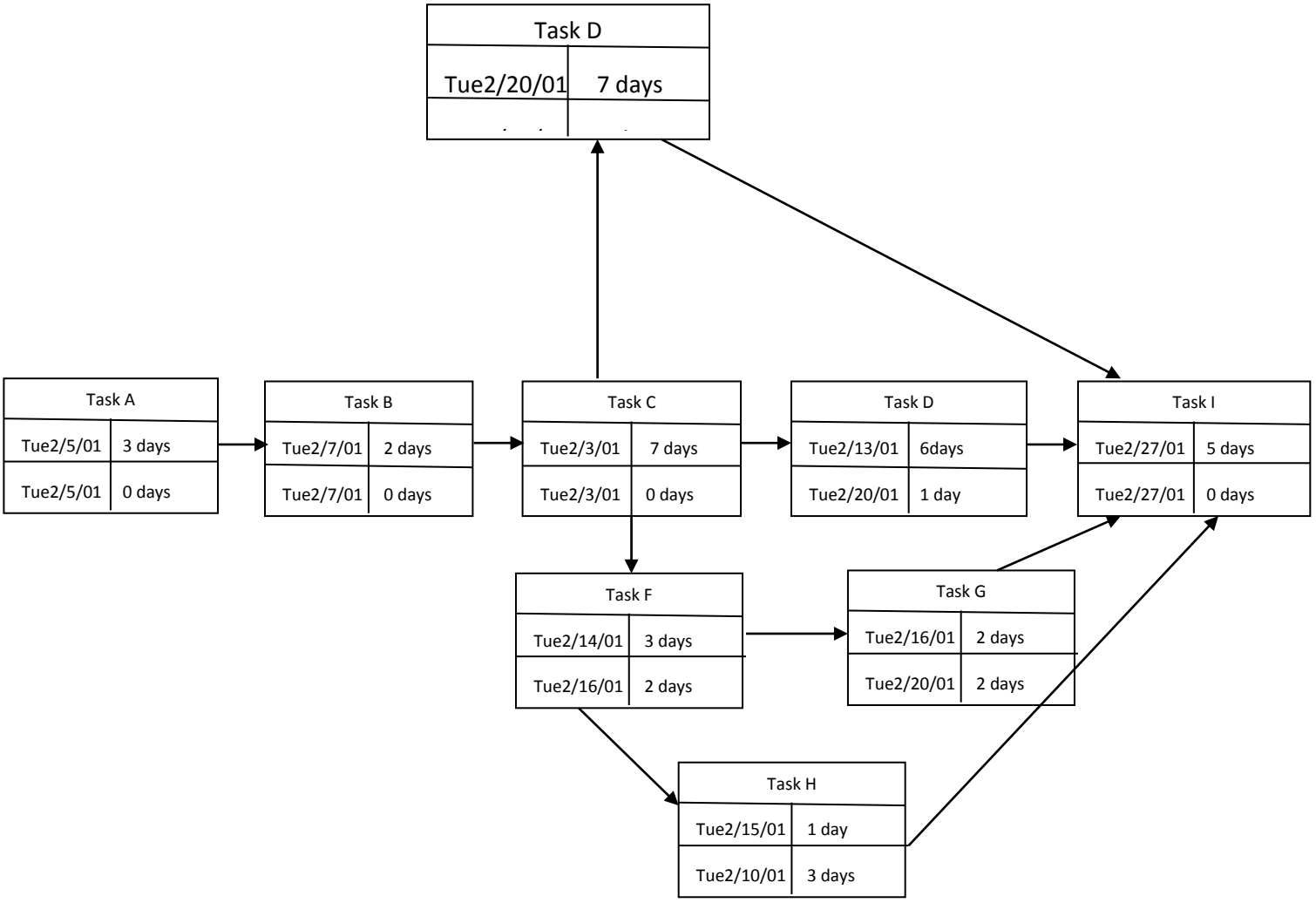


Fig: critical analysis path.

Suppose a project consists of the nine primitive tasks as shown in fig. The most likely duration (in days) for each task is recorded. There are four distinct sequences of tasks in a project. They are:

PATH 1: A →B →C →D →I
PATH 2: A→ B→C →E→I
PATH 3: A→B →C →F →G→I
PATH 4: A→B→ C→ F→ H→ I

The total of likely duration times for each path is calculated as follows:

PATH 1: 3+2+2+7+5=19
PATH 2: 3+2+2+6+5=18
PATH 3: 3+2+2+3+2+5=17
PATH 4: 3+2+2+3+1+5=16

[CSc. 351 Software Engineering]

Lecturer: ***Hiranya Bastakoti***

Amrit Science Campus

What is a requirement?

- ✓ In a very simple language, a requirement can be defined as a feature which end user needs in the existing or the new system
- ✓ According to IEEE standard 729: A requirement is defined as (i) a condition or capacity needed by a user to solve a problem or to achieve an objective;(ii) a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed document;(iii) a document representation of condition or capacity as in(i) and (ii).
- ✓ Requirement may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification
- ✓ Requirement for a software system set out what the system should do and define constraints on its operation and implementation.
- ✓ This is inevitable as requirements may serve a dual function
    - May be the basis for a bid for a contract - therefore must be open to interpretation
    - May be the basis for the contract itself - therefore must be defined in detail
    - Both these statements may be called requirements

Types of requirement

1. **User requirements**
    - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers
2. **System requirements**
    - A structured document setting out detailed descriptions of the system services. Written as a contract between client and contractor
3. **Software specification**
    - A detailed software description which can serve as a basis for a design or implementation. Written for developers

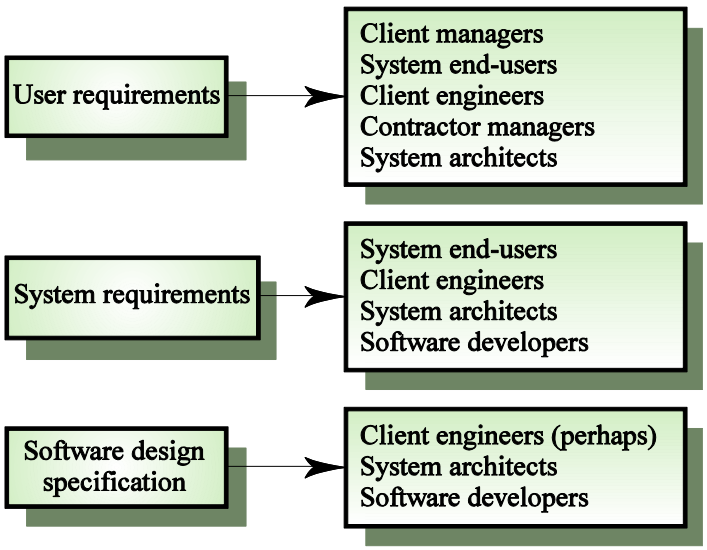


Fig: Types of readers for the user and system requirements

Functional and non-functional requirements

1. **Functional requirements**

- Statements of services the system should provide how the system should react to particular inputs and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should no do.

2. **Non-functional requirements**

- These are constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc. Non-functional requirements often apply to the system as a whole. They do not usually just apply to individual system features or services.

3. **Domain requirements**

- Requirements that come from the application domain of the system and that reflect characteristics and constraints of that domain. They may be functional or non-functional

Functional requirements

- ✓ The functional requirements for a system describe what the system should do. These requirements depend on the type of software being developed, the expected users of the software and the general approach taken by the organization when writing requirements. When expressed as user requirements, the requirements described in a fairly abstract way. However, functional system requirements describe the system function in detail, its inputs and outputs, expectations, and so on.
- ✓ Functional requirements for the software system may be expressed in a number of ways.

- ✓ E.g.: Functional requirements for a library system called LIBSYS, used by students and faculty to order books and documents from other libraries.
    - The user shall be able to search either all of the initial set of databases or select a subset from it.
    - The system shall provide appropriate viewers for the user to read documents in the document store.
    - Every order shall be allocated a unique identifier (ORDER_ID) which the user shall be able to copy to the account's permanent storage area

Requirements imprecision

- ✓ Imprecision in the requirements specification is the cause of many software engineering problems.
- ✓ Problems arise when requirements are not precisely stated
- ✓ Ambiguous requirements may be interpreted in different ways by developers and users
- ✓ Consider the term 'appropriate viewers'
    - User intention - special purpose viewer for each different document type
    - Developer interpretation - Provide a text viewer that shows the contents of the document

Requirements completeness and consistency

- ✓ In principle requirements should be both complete and consistent
- ✓ **Complete**
    - They should include descriptions of all facilities required
- ✓ **Consistent**
    - There should be no conflicts or contradictions in the descriptions of the system facilities
- ✓ In practice, it is impossible to produce a complete and consistent requirements document

Non-functional requirements

- ✓ These functions are not directly concerned with the specific functions delivered by the system.
- ✓ Define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.

- ✓ They may be specifying system performance, security, availability, and other emergent properties. This means they are often more critical than individual functional requirements.
- ✓ Non-functional requirements are not just concerned with the software system to be developed. Some non-functional requirements may constrain (restrict) the process that should be used to develop the system.
- ✓ Process requirements may also be specified mandating a particular CASE system, programming language or development method
- ✓ Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless

Non-functional classifications

Product requirements

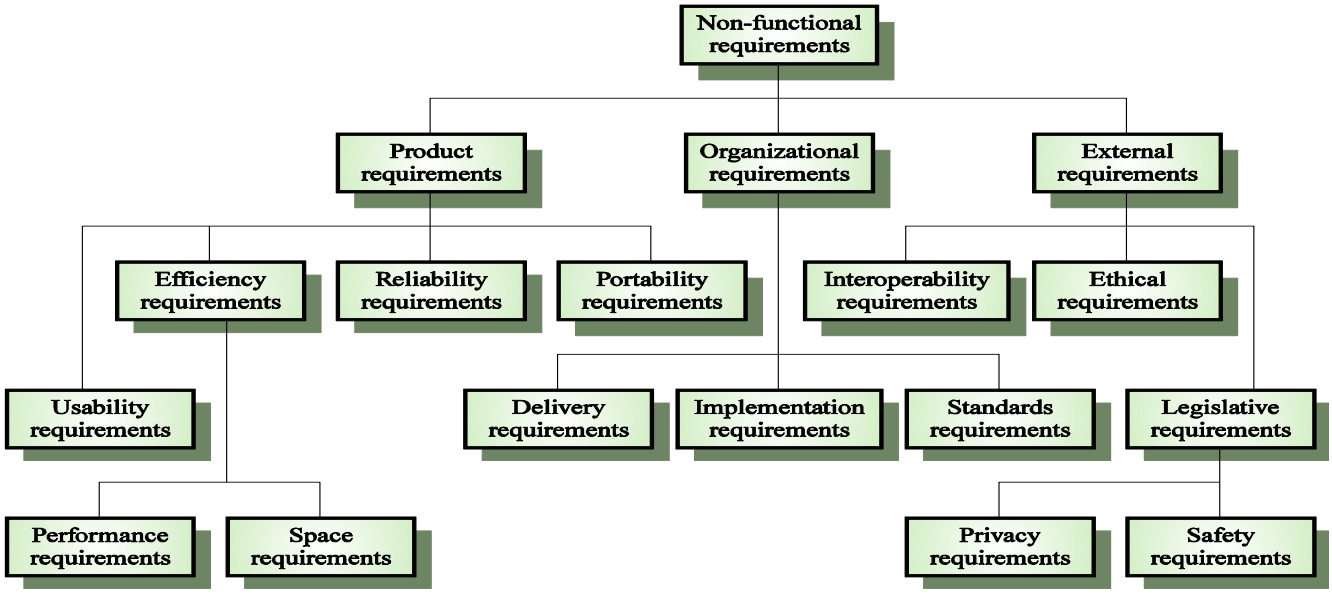
- Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, portability etc.

Organisational requirements

- Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc**.**

External requirements

- Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc**.**



Non-functional requirement types

Non-functional requirements examples

Product requirement

- The user interface for LIBSYS shall be implemented as simple HTML without frames or java applets.

Organisational requirement

- The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95

External requirement

- The system shall not disclose any personal information about customers apart from their name and library reference number to the operators of the system

Goals and requirements

✓ Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.

Goal

- A general intention of the user such as ease of use
- Goals are helpful to developers as they convey the intentions of the system users

Examples:

A system goal

- The system should be easy to use by experienced controllers and should be organised in such a way that user errors are minimised.

A verifiable non-functional requirement

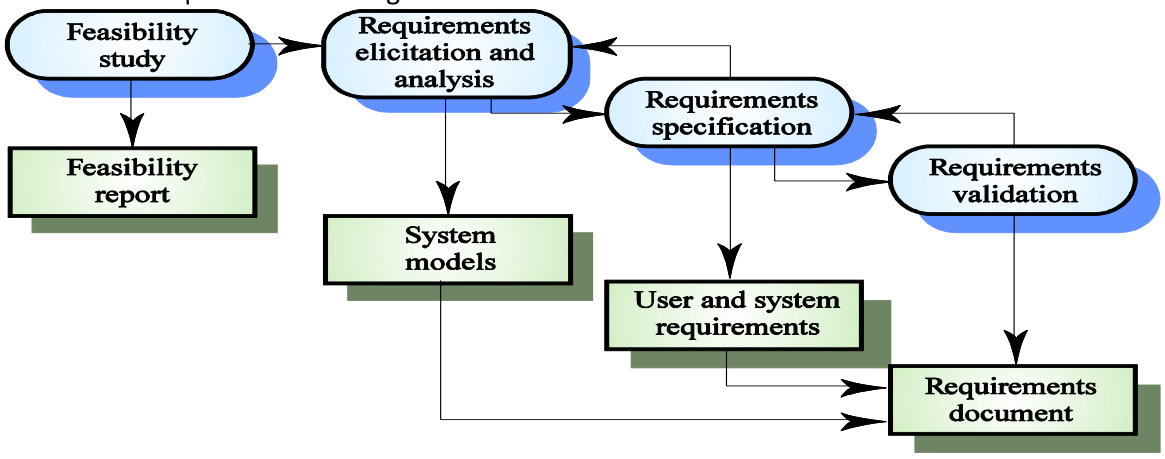
- Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.

Non-functional Requirements measures specifications:

Property	**Measure**
Speed	Processed transactions/second User/Event response time
Size	K Bytes Number of RAM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

6.1 Requirements Engineering Processes

- It is the systematic of documenting requirements through an interactive co-operative process of analyzing the problems, documenting the resulting observations in a variety of representation format, and checking the accuracy of the understanding gained.
- Requirement Engineering is the systematic use of proven principles, techniques, tools and languages for the cost effective analysis, documentation and ongoing evolution of user needs and the specification of the external behavior of the system in order to satisfy these needs.
- An input to this activity of requirements engineering are requirements which are informal and fuzzy (unclear) and output is clear, well defined, complete and consistent.
- The requirements engineering activity is the most crucial and most important in the software development life cycle because the errors introduced at this stage are the most expensive errors requiring lot of rework if detected late in software life cycle.
- The processes used for Requirement Engineering vary widely depending on the application domain, the people involved and the organisation developing the requirements
  - However, there are a number of generic activities common to all processes
    - Feasibility study
    - Requirements elicitation
    - Requirements analysis
    - Requirements validation
    - Requirements management



Feasibility study:

- Feasibility study is a set of preliminary business requirements, an outline description of the system and how the system is intended to support business process. The results of the feasibility study should be are part that recommends whether or not it is worth carrying on with the requirements engineering and system development process.
- Feasibility study involves information assessment (what is required), information collection and report writing.

A feasibility study is a short, focused study that aims to answer a number of questions:

i. Does the system contribute to the overall objectives of the organization?
ii. Can the system be implemented using current technology and within given cost and schedule constraints?
iii. Can the system be integrated with other systems which are already in place?

Feasibility Analysis:

Feasibility: The measure of how beneficial or practical an information system will be to an organization.

Feasibility analysis: A feasibility analysis is the process by which feasibility is measured.

Feasibility should be measured throughout the life cycle.

Four tests for feasibility:

1. **Operational feasibility**: this is the measure of how well the solution will work in an organization. It is also measure of how people feel about the system/project. There are two aspects of operational feasibility to be considered:

a. Is the problem worth solving, or will the solution to the problem?

PIECES can be used as the basis for analyzing the urgency of a problem or the effectiveness of the solution.

P - Performance.
I - Information.
E - Economy.
C - Control.
E - Efficiency.
S - Services.

b. How does the end users and management feel about the problem (solution)?

- Does management support the system?
- How do the end users feel about their role in the new system?
- How will the working environment of the end users change?

2. **Technical Feasibility**: technical feasibility is a measure of the practically of a specific technical solution and the availability of technical resources and expertise.

Technical feasibility addresses three major issues.

Is the proposed technology or solution practical?
Do we currently posses the necessary technology?
Do we posses the necessary technical expertise?

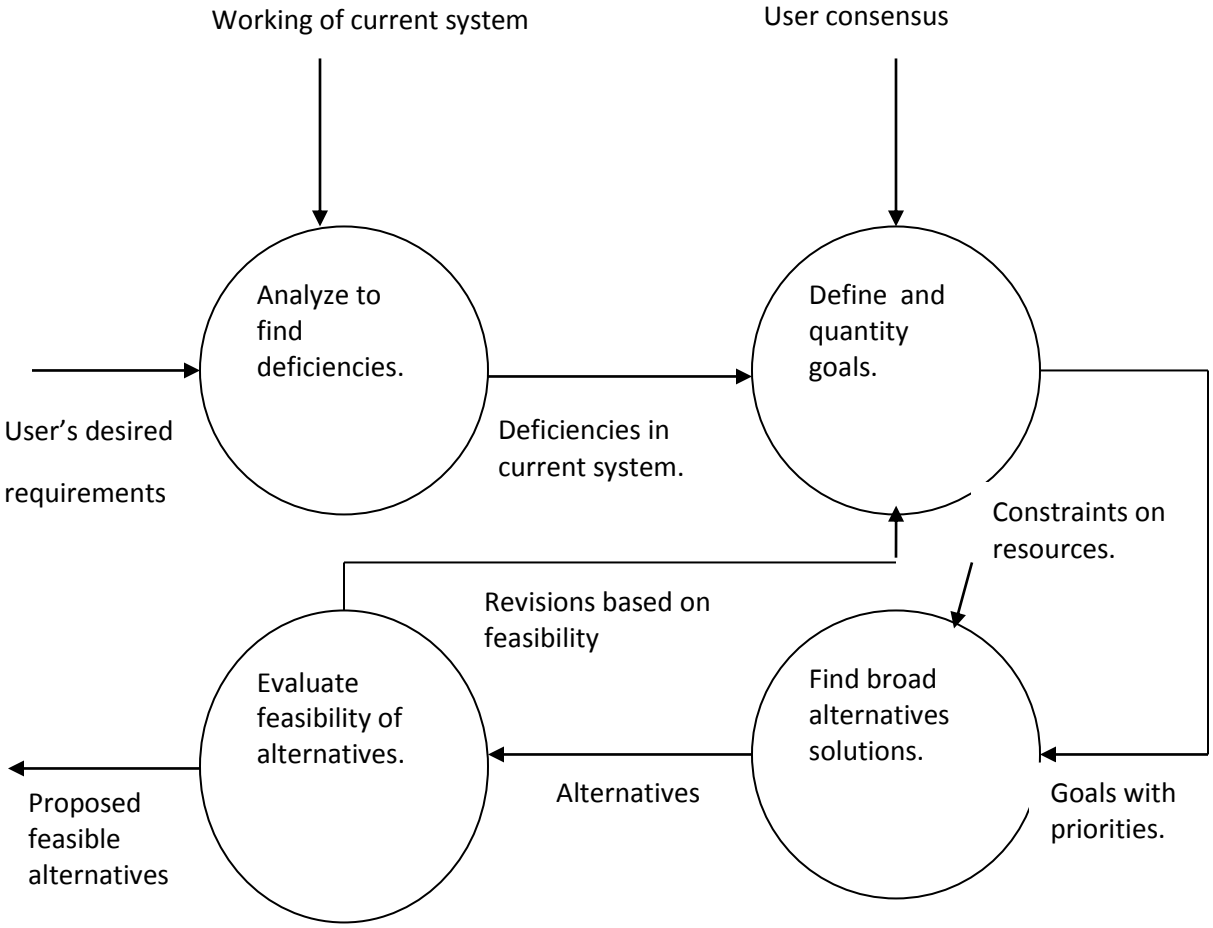
3. **Schedule feasibility**: schedule feasibility is a measure of how reasonable project time table is?
4. **Economic feasibility**: Economic feasibility is the measure of the cost-effectiveness of a project or Solution.

Steps in feasibility analysis:

1. Identify deficiency by pinpointing
   - Missing functions.
   - Unsatisfactory performance and
   - Excessive cost of operations.
2 Set goals to overcome these deficiencies.
3. Goals must be
   - quantified
   - Realizable within the constraints of an organization.
   - Broken down into sub-goals and

- Agreeable to all concerned

4 .Set goals not only to remove deficiencies but also to effectively meet composition. For instance, goals may be based on want competitors do.



Questions for people in the organisation

- What if the system wasn't implemented?
- What are current process problems?
- How will the proposed system help?
- What will be the integration problems?
- Is new technology needed? What skills?
- What facilities must be supported by the proposed system?

Elicitation and analysis

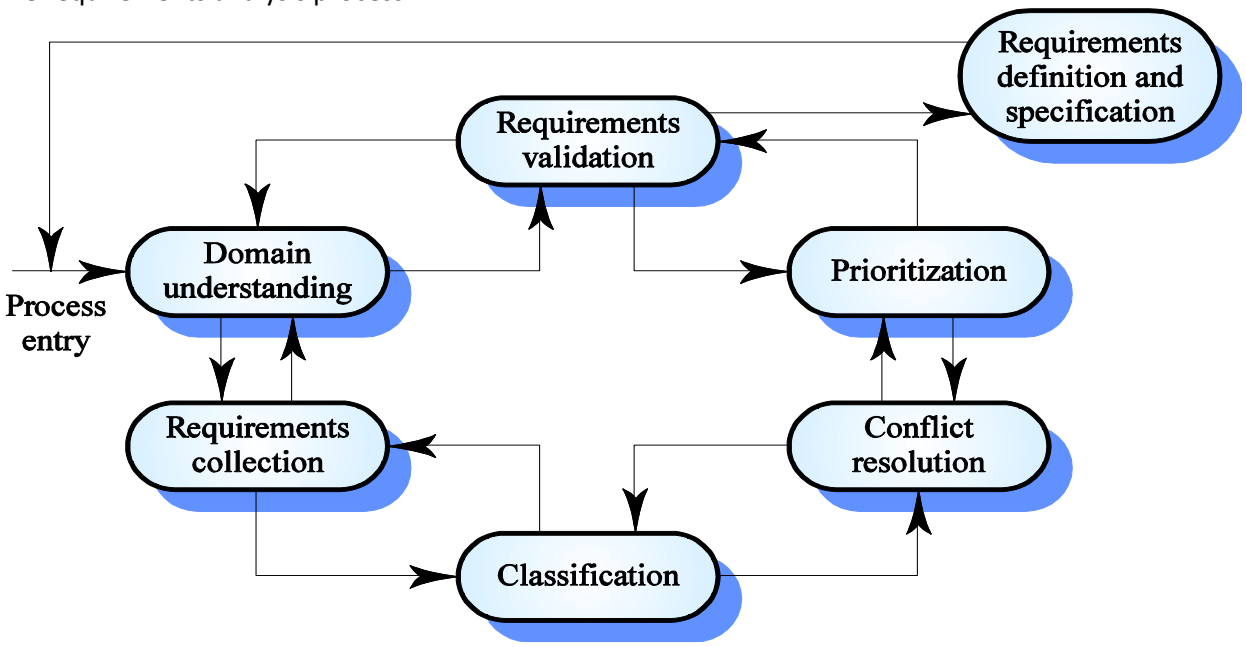
- Sometimes called requirements elicitation or requirements discovery
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders*

Problems of requirements analysis

- Stakeholders don't know what they really want
- Stakeholders express requirements in their own terms

- Different stakeholders may have conflicting requirements
- Organisational and political factors may influence the system requirements
- The requirements change during the analysis process. New stakeholders may emerge and the business environment change

The requirements analysis process



Process activities

- Domain understanding
- Requirements collection
- Classification
- Conflict resolution
- Prioritisation
- Requirements checking

Viewpoint-oriented elicitation

- Stakeholders represent different ways of looking at a problem or problem viewpoints
- This multi-perspective analysis is important as there is no single correct way to analyse system requirements

Banking ATM system

- The example used here is an auto-teller system which provides some automated banking services
- I use a very simplified system which offers some services to customers of the bank who own the system and a narrower range of services to other customers
- Services include cash withdrawal, message passing (send a message to request a service), ordering a statement and transferring funds

Autoteller viewpoints

- Bank customers
- Representatives of other banks
- Hardware and software maintenance engineers
- Marketing department
- Bank managers and counter staff

- Database administrators and security staff
- Communications engineers
- Personnel department

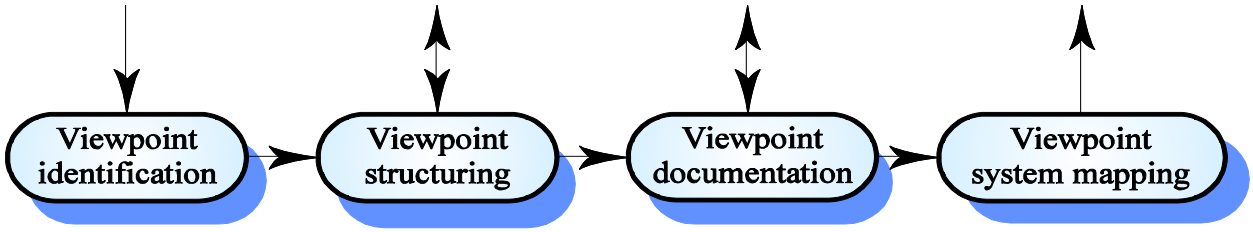
Types of viewpoint

- Data sources or sinks
  - Viewpoints are responsible for producing or consuming data. Analysis involves checking that data is produced and consumed and that assumptions about the source and sink of data are valid
- Representation frameworks
  - Viewpoints represent particular types of system model. These may be compared to discover requirements that would be missed using a single representation. Particularly suitable for real-time systems
- Receivers of services
  - Viewpoints are external to the system and receive services from it. Most suited to interactive systems

External viewpoints

- Natural to think of end-users as receivers of system services
- Viewpoints are a natural way to structure requirements elicitation
- It is relatively easy to decide if a viewpoint is valid
- Viewpoints and services may be sued to structure non-functional requirements

The VORD method



VORD process model

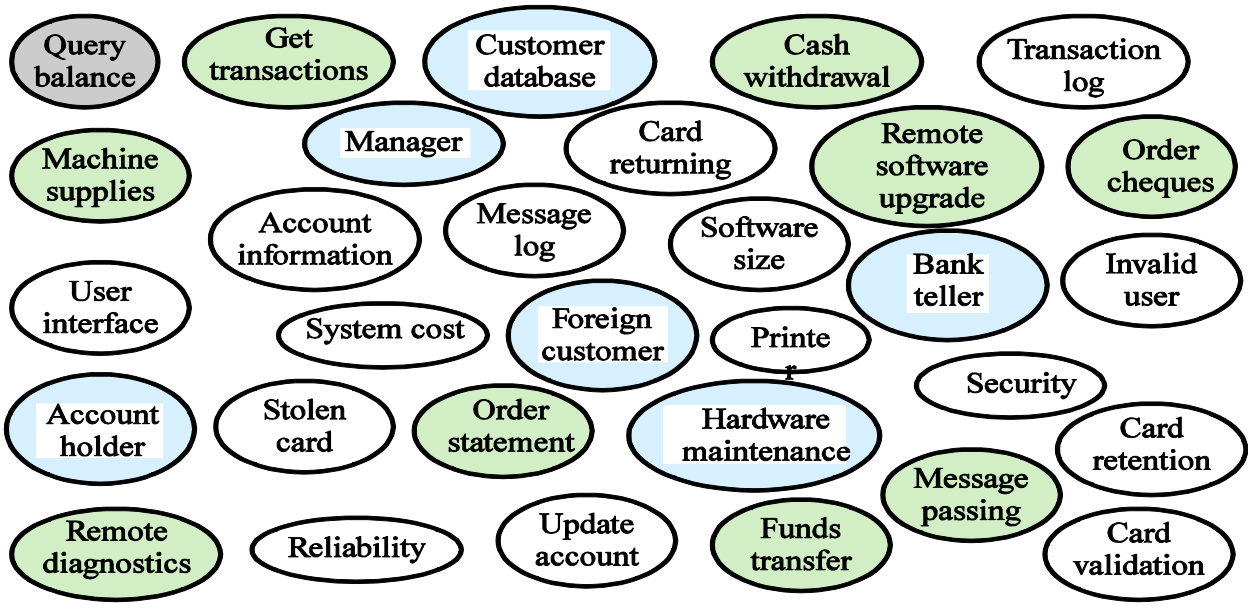
- Viewpoint identification
  - Discover viewpoints which receive system services and identify the services provided to each viewpoint
- Viewpoint structuring
  - Group related viewpoints into a hierarchy. Common services are provided at higher-levels in the hierarchy
- Viewpoint documentation
  - Refine the description of the identified viewpoints and services
- Viewpoint-system mapping
  - Transform the analysis to an object-oriented design

VORD standard forms

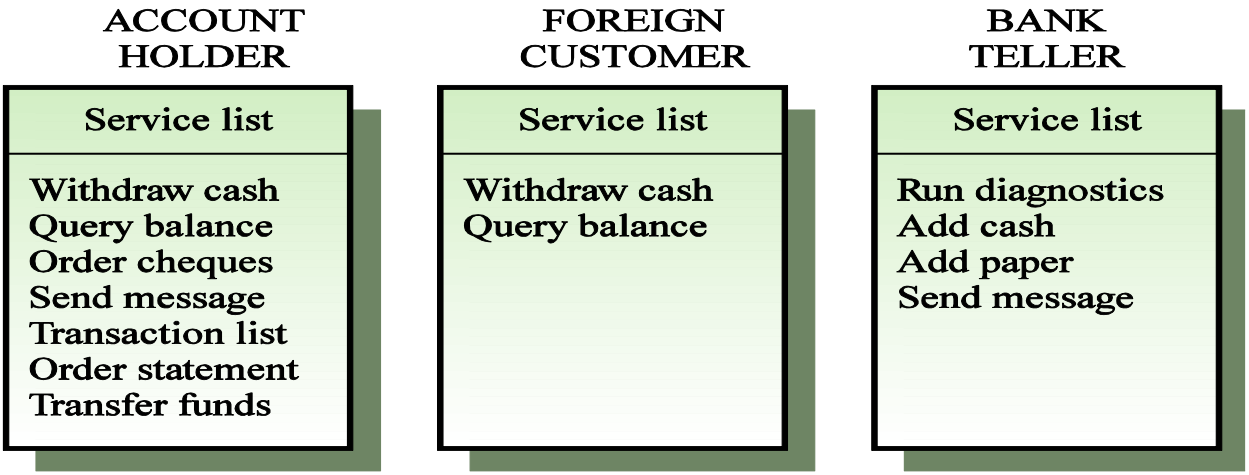
Viewpoint template	
Reference:	The viewpoint name.
Attributes:	Attributes providing viewpoint information.
Events:	A reference to a set of event scenarios describing how the system reacts to viewpoint events.
Services	A reference to a set of service descriptions.
Sub-VPs:	The names of sub-viewpoints.

Service template	
Reference:	The service name.
Rationale:	Reason why the service is provided.
Specification:	Reference to a list of service specifications. These may be expressed in different notations.
Viewpoints:	List of viewpoint names receiving the service.
Non-functional requirements:	Reference to a set of non-functional requirements which constrain the service.
Provider:	Reference to a list of system objects which provide the service.

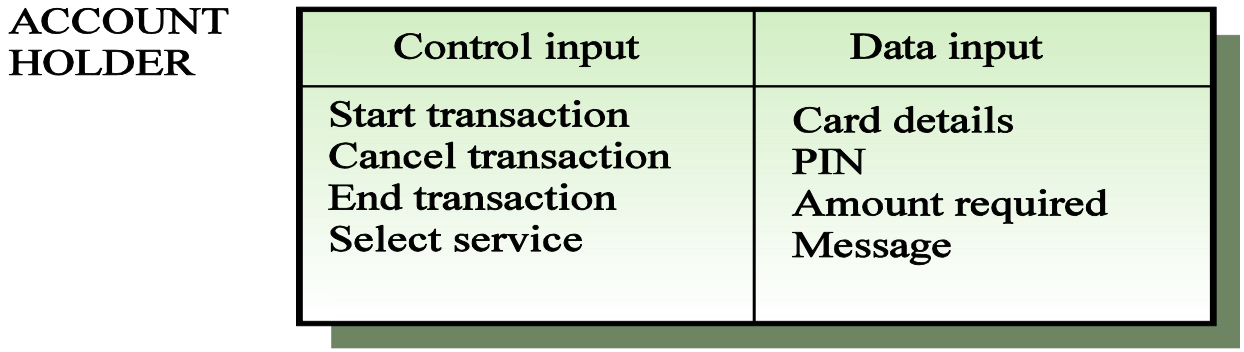
Viewpoint identification



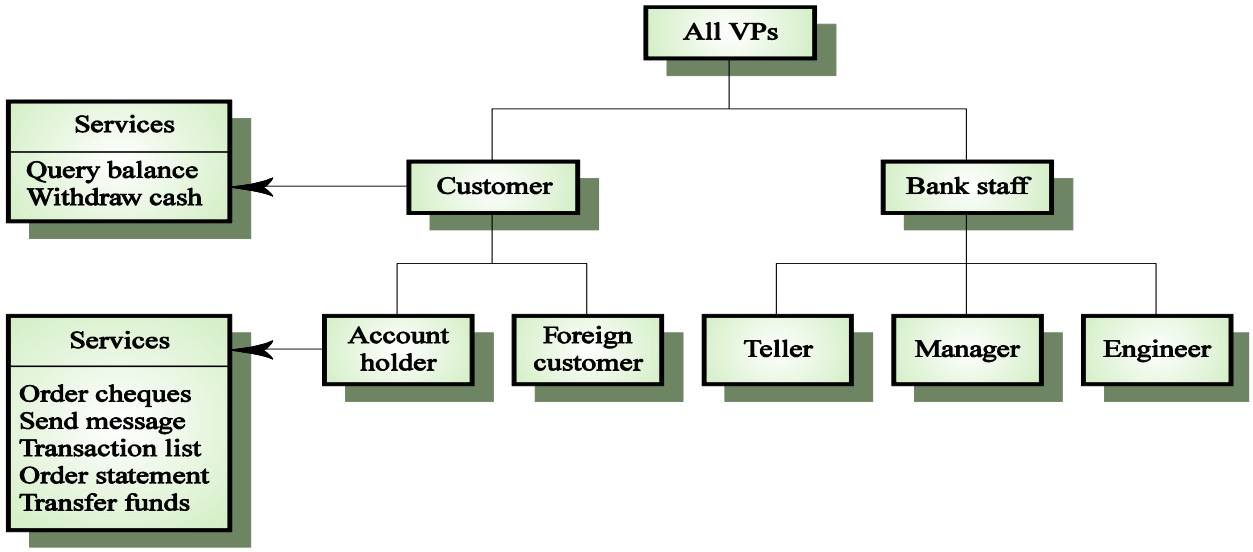
Viewpoint service information



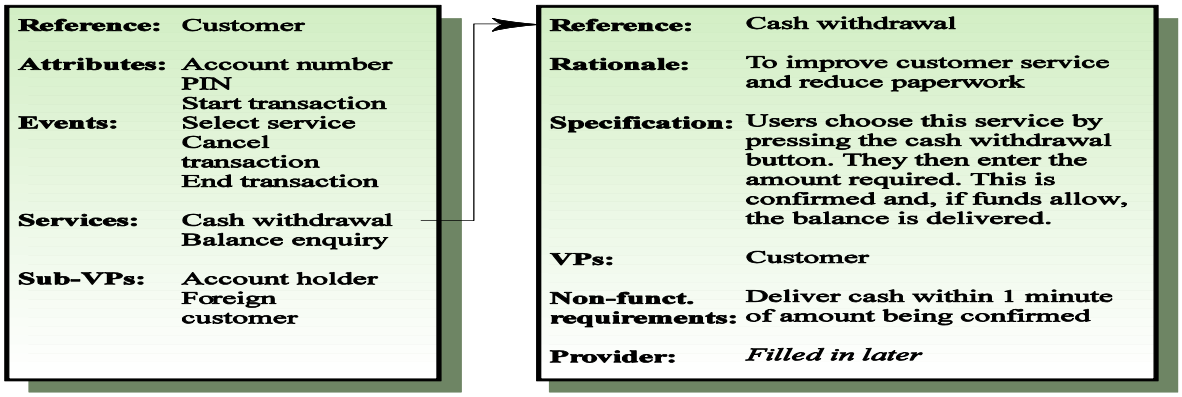
Viewpoint data/control



Viewpoint hierarchy



Customer/cash withdrawal templates



[CSc. 351 Software Engineering]

Lecturer: ***Hiranya Bastakoti***

Amrit Science Campus

4.2 Testing and Types Of Testing: Software Engineering

Introduction:

Software testing is a process used to identify the correctness, completeness and quality of developed computer software

- Testing is the process of executing a program with the intent of finding errors.
- Testing is the process of uncovering errors in a program makes it a feasible task

Testing software is operating the software under controlled conditions, to (1) verify that it behaves "as specified"; (2) to detect errors, and (3) to validate that what has been specified is what the user actually wanted.

1. **Verification** is the checking or testing of items, including software, for conformance and consistency by evaluating the results against pre-specified requirements. [*Verification: Are we building the system right?*]

2. **Error Detection**: Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should.

3. **Validation** looks at the system correctness – i.e. is the process of checking that what has been specified is what the user actually wanted. [*Validation: Are we building the right system?*]

In other words, validation checks to see if we are building what the customer wants/needs, and verification checks to see if we are building that system correctly. Both verification and validation are necessary, but different components of any testing activity.

Testing is that testing is the process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Testing objectives:

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has a high probability of finding an as-yet undiscovered error.
- A successful test is one that uncovers an as-yet-undiscovered error.

Testing Principles:

- All tests should be traceable to customer requirements.
- Tests should be planned long before testing begins.
- Testing should begin "in the small" and progress toward testing "in the large."
- Exhaustive testing is not possible.
- To be most effective, testing should be conducted by an independent third party.

The testing process

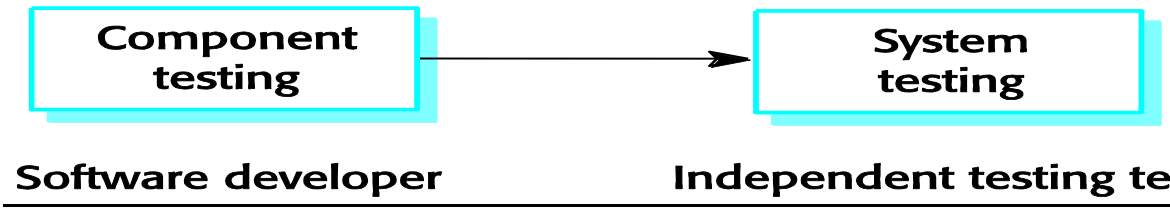
Component testing

- Testing of individual program components;
- Usually the responsibility of the component developer (except sometimes for critical systems);
- Tests are derived from the developer's experience.

System testing

- Testing of groups of components integrated to create a system or sub-system
- The responsibility of an independent testing team
- Tests are based on a system specification.
- Tests are derived from the developer's experience

Testing phases



Testing process goals:

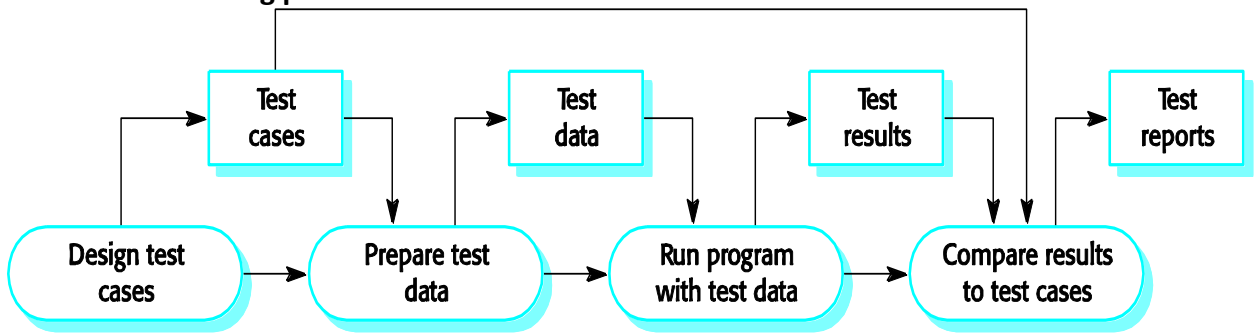
Defect testing

- The goal of defect testing is to discover defects in programs
- A successful defect test is a test which causes a program to behave in an anomalous way
- Tests show the presence not the absence of defect

Validation testing

- To demonstrate to the developer and the system customer that the software meets its requirements;
- A successful test shows that the system operates as intended.

The software testing process:



- Test data Inputs which have been devised to test the system

- Test cases Inputs to test the system and the predicted outputs from these inputs if the system operates according to its specification

Testing policies:

- Only exhaustive testing can show a program is free from defects. However, exhaustive testing is impossible,
  - Testing policies define the approach to be used in selecting system tests:
    - All functions accessed through menus should be tested;
    - Combinations of functions accessed through the same menu should be tested;
    - Where user input is required, all functions must be tested with correct and incorrect input.

System testing

- Involves integrating components to create a system or sub-system.
- May involve testing an increment to be delivered to the customer.

Two phases:

a. Integration testing - the test team have access to the system source code. The system is tested as components are integrated.

- Involves building a system from its components and testing it for problems that arise from component interactions.

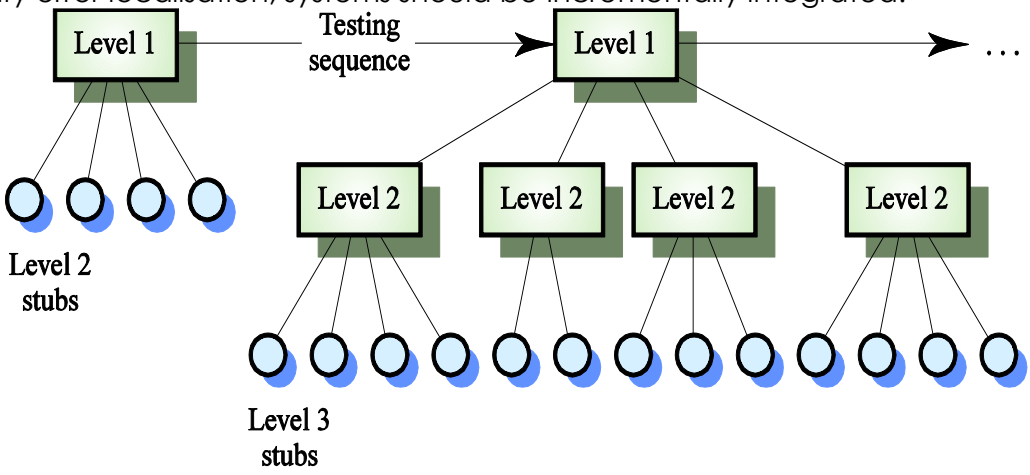
Top-down integration

- Develop the skeleton of the system and populate it with components.
- Start with high-level system and integrate from the top-down replacing individual components by stubs where appropriate

Bottom-up integration

- Integrate infrastructure components then add functional components.
- Integrate individual components in levels until the complete system is created

To simplify error localisation, systems should be incrementally integrated.



.

Fig: Top-down testing

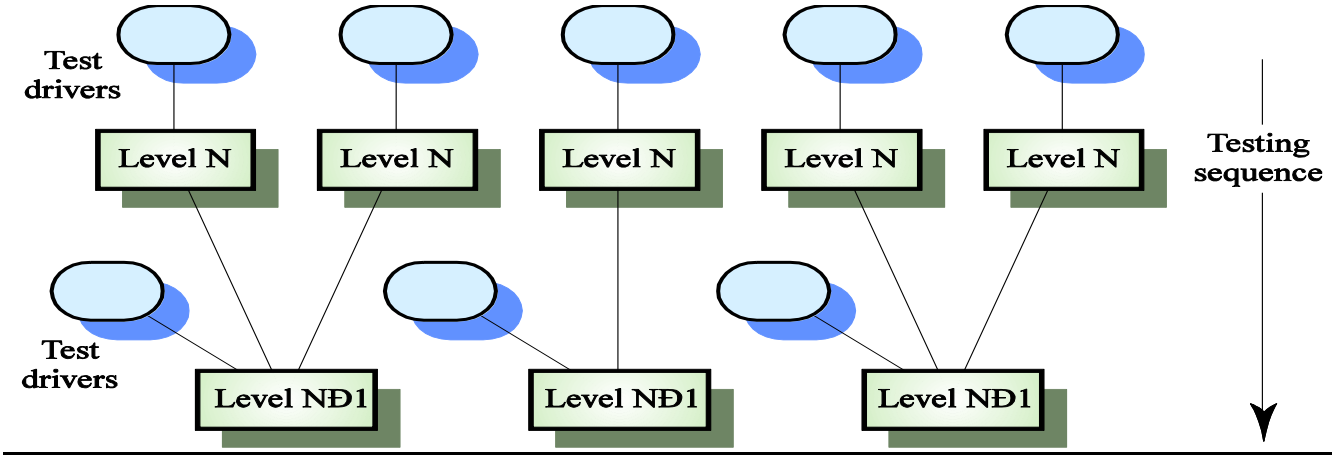
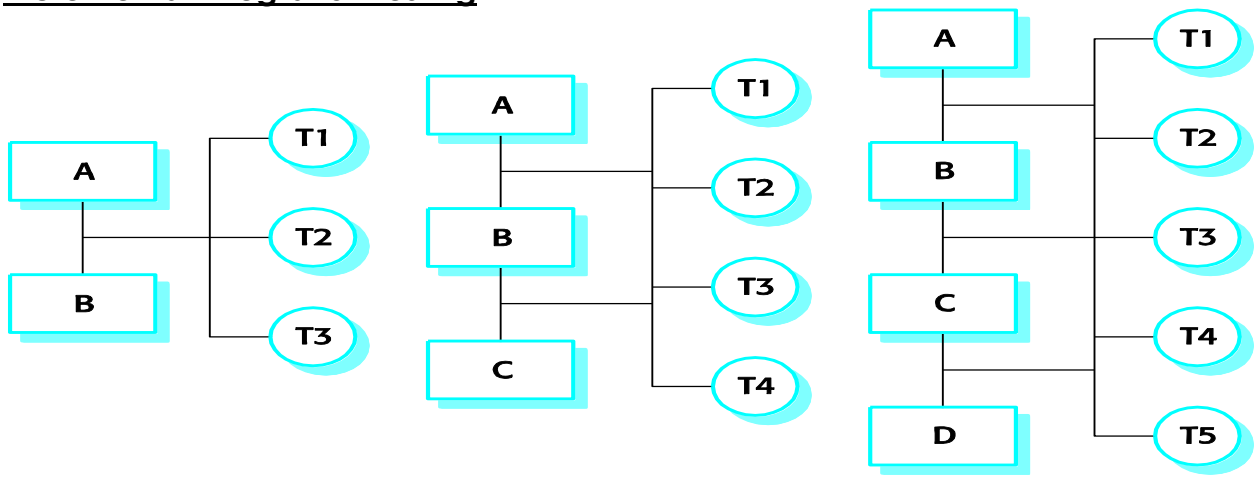


Fig: Bottom-up testing

Incremental integration testing



Test sequence 1 **Test sequence 2** **Test sequence 3**

Testing approaches

Architectural validation

- Top-down integration testing is better at discovering errors in the system architecture.

System demonstration

- Top-down integration testing allows a limited demonstration at an early stage in the development.

Test implementation

- Often easier with bottom-up integration testing.

Test observation

- Problems with both approaches. Extra code may be required to observe tests.

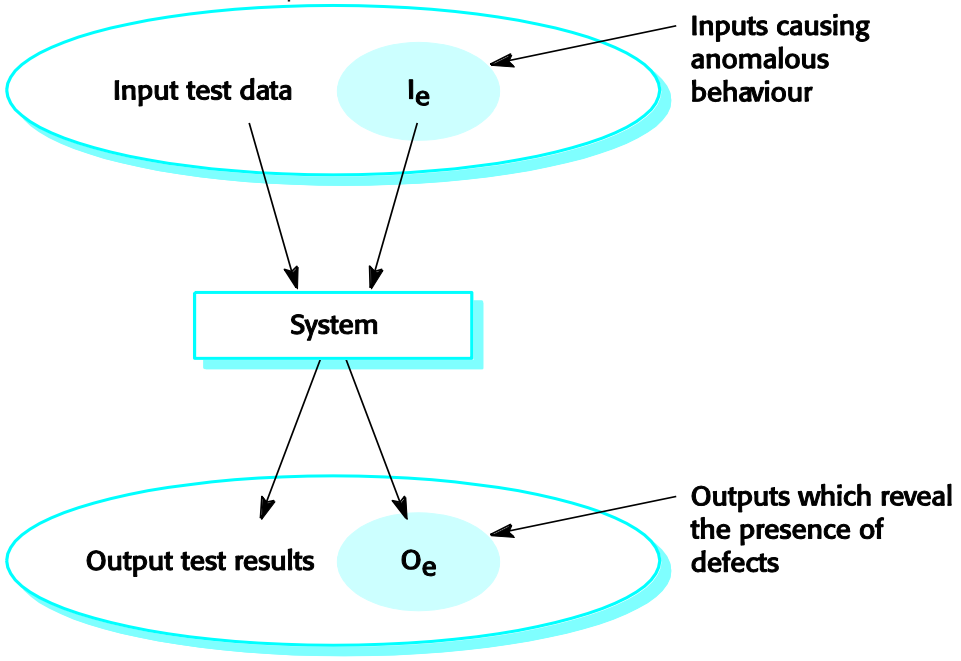
b. Release testing - the test team test the complete system to be delivered as a black-box.

- The process of testing a release of a system that will be distributed to customers.
- Primary goal is to increase the supplier's confidence that the system meets its requirements.
- Release testing is usually black-box or functional testing
  - Based on the system specification only;

Testers do not have knowledge of the system implementation

Black-box testing

This approach tests all possible combinations of end-user actions. Black box testing assumes no knowledge of code and is intended to simulate the end-user experience. You can use sample applications to integrate and test the application block for black box testing. You can begin planning for black box testing immediately after the requirements and the functional specifications are available..



- An approach to testing where the program is considered as a 'black-box'
- The program test cases are based on the system specification
- Test planning can begin early in the software process

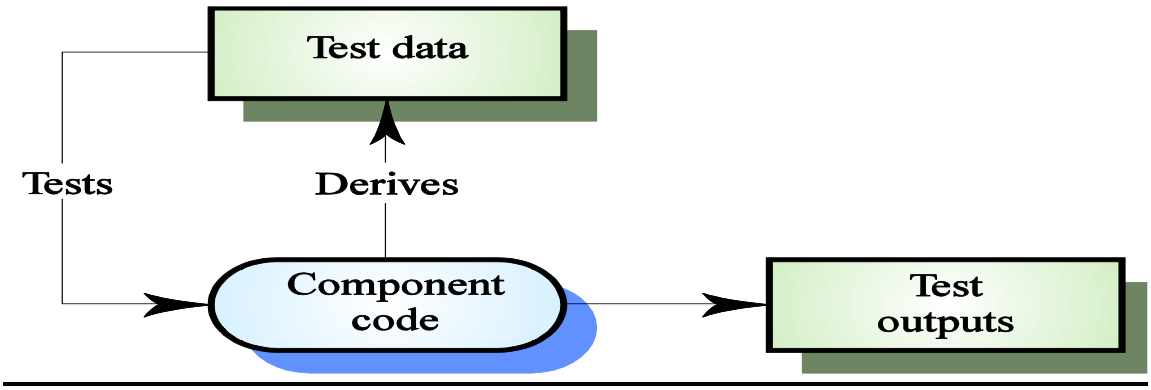
Structural testing

- Sometime called white-box testing
- Derivation of test cases according to program structure. Knowledge of the program is used to identify additional test cases
- Objective is to exercise all program statements (not all path combinations)

White-box testing

(This is also known as glass box, clear box, and open box testing.) In white box testing, you create test cases by looking at the code to detect any potential failure scenarios. You determine the suitable input data for testing various APIs and the special code paths that

need to be tested by analyzing the source code for the application block. Therefore, the test plans need to be updated before starting white box testing and only after a stable build of the code is available.



Black box testing is *testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions*.

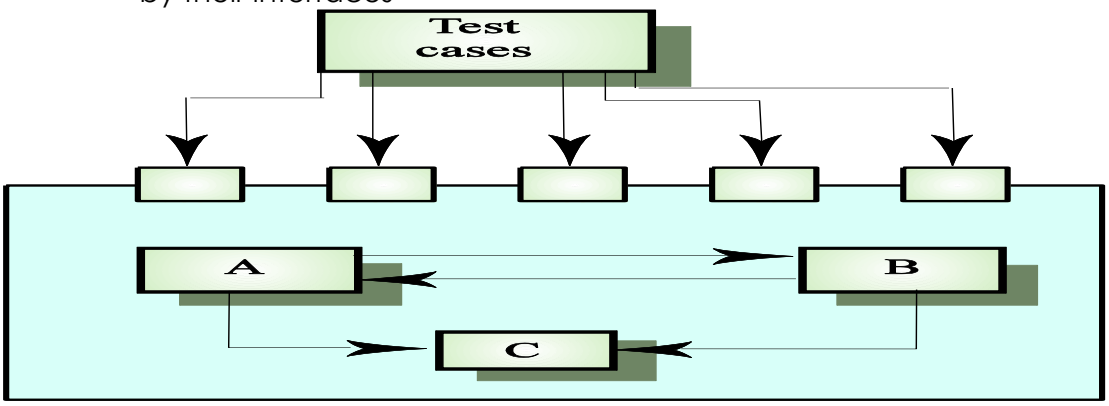
White box testing is *testing that takes into account the internal mechanism of a system or component*.

Difference Between Black Box and white Box testing

Criteria	**Black Box Testing**	**White Box Testing**
Definition	Black Box Testing is a software testing method in which the internal structure/design/implementation of the item being tested is NOT known to the tester	White Box Testing is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester.
Levels Applicable	Mainly applicable to higher levels of testing: Acceptance Testing, System Testing	Mainly applicable to lower levels of testing: Unit Testing, Integration Testing
Responsibility	Generally, independent Software Testers	Generally, Software Developers
Programming Knowledge	Not Required	Required
Implementation Knowledge	Not Required	Required
Basis for Test Cases	Requirement Specifications	Detail Design

Interface testing

- Takes place when modules or sub-systems are integrated to create larger systems
- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces
- Particularly important for object-oriented development as objects are defined by their interfaces



Interfaces types

Parameter interfaces

- Data passed from one procedure to another

Shared memory interfaces

- Block of memory is shared between procedures

Procedural interfaces

- Sub-system encapsulates a set of procedures to be called by other sub-systems

Message passing interfaces

- Sub-systems request services from other sub-systems

Interface errors

Interface misuse

- A calling component calls another component and makes an error in its use of its interface e.g. parameters in the wrong order

Interface misunderstanding

- A calling component embeds assumptions about the behaviour of the called component which are incorrect

Timing errors

- The called and the calling component operate at different speeds and out-of-date information is accessed

Interface testing guidelines

- Design tests so that parameters to a called procedure are at the extreme ends of their ranges
- Always test pointer parameters with null pointers
- Design tests which cause the component to fail
- Use stress testing in message passing systems
- In shared memory systems, vary the order in which components are activated

Stress testing

- Exercises the system beyond its maximum design load. Stressing the system often causes defects to come to light
- Stressing the system test failure behaviour.. Systems should not fail catastrophically. Stress testing checks for unacceptable loss of service or data
- Particularly relevant to distributed systems which can exhibit severe degradation as a network becomes overloaded

Object-oriented testing

- The components to be tested are object classes that are instantiated as objects
- Larger grain than individual functions so approaches to white-box testing have to be extended
- No obvious 'top' to the system for top-down integration and testing

Testing levels

- Testing operations associated with objects
- Testing object classes
- Testing clusters of cooperating objects
- Testing the complete OO system

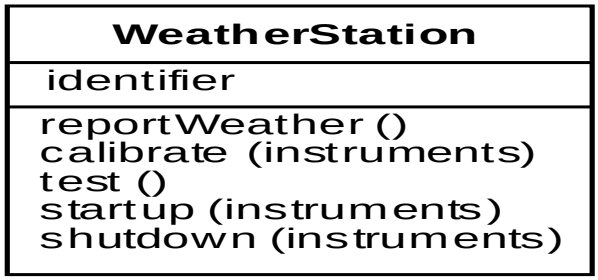
Object class testing

Complete test coverage of a class involves

- Testing all operations associated with an object
- Setting and interrogating all object attributes
- Exercising the object in all possible states

Inheritance makes it more difficult to design object class tests as the information to be tested is not localised

Weather station object interface



Test cases are needed for all operations
Use a state model to identify state transitions for testing
Examples of testing sequences

- Shutdown — Waiting— Shutdown
- Waiting — Calibrating —Testing —Transmitting —Waiting
- Waiting — Collecting —Waiting — Summarising — Transmitting — Waiting

Object integration

- Levels of integration are less distinct in object-oriented systems

- Cluster testing is concerned with integrating and testing clusters of cooperating objects
- Identify clusters using knowledge of the operation of objects and the system features that are implemented by these clusters

Approaches to cluster testing

Use-case or scenario testing

- Testing is based on a user interactions with the system
- Has the advantage that it tests system features as experienced by users

Thread testing

- Tests the systems response to events as processing threads through the system

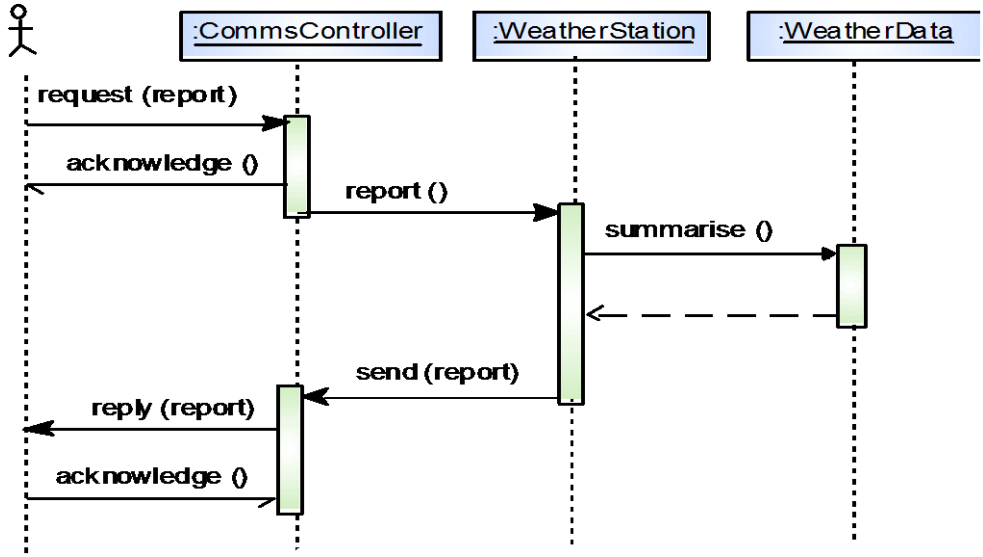
Object interaction testing

- Tests sequences of object interactions that stop when an object operation does not call on services from another object

Scenario-based testing

- Identify scenarios from use-cases and supplement these with interaction diagrams that show the objects involved in the scenario
- Consider the scenario in the weather station system where a report is generated

Collect weather data



Weather station testing

Thread of methods executed

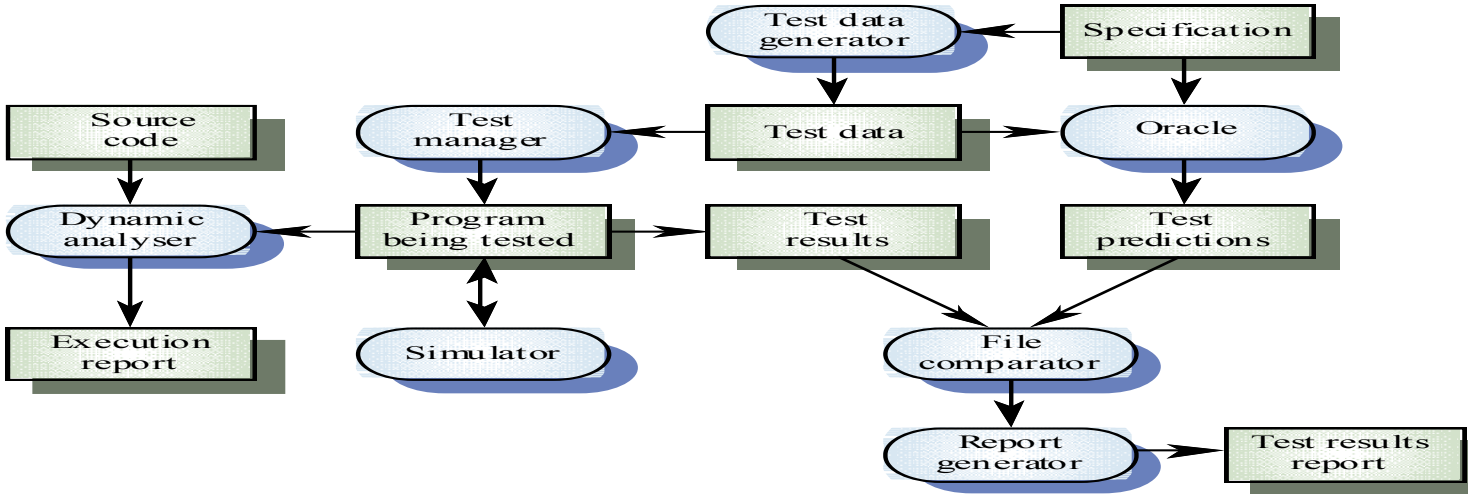
- CommsController :request --- WeatherStation :report ---WeatherData::summarise

Inputs and outputs

- Input of report request with associated acknowledge and a final output of a report
- Can be tested by creating raw data and ensuring that it is summarised properly
- Use the same raw data to test the WeatherData object

Testing workbenches

- Testing is an expensive process phase. Testing workbenches provide a range of tools to reduce the time required and total testing costs
- Most testing workbenches are open systems because testing needs are organisation-specific
- Difficult to integrate with closed design and analysis workbenches



Testing workbench has four components:

1. **Input:** The entrance criteria or deliverables needed to perform work,
2. **Procedures to do**: The tasks or processes that will transform the input into the output,
3. **Procedures to check**: The processes that determine that the output meets the standards,
4. **Output**: The exit criteria or deliverables produced from the workbench.

The tools used to automate the testing process in a test bench perform the following functions:

1. **Test manager**: manages the running of program tests; keeps track of test data, expected results and program facilities tested.
2. **Test data generator**: generates test data for the program to be tested.
3. **Oracle**: generates predictions of the expected test results; the oracle may be either previous program versions or prototype systems.
4. **File comparator:** compares the results of the program tests with previous test results and records any differences in a document.
5. **Report generator**: provides report definition and generation facilities for the test results.
6. **Dynamic analyzer**: adds code to a program to count the number of times each statement has been executed. It generates an execution profile for the statements to show the number of times they are executed in the program run.
7. **Simulator**: simulates the testing environment where the software product is to be used.