PART I: INTRODUCTION TO SOFTWARE ENGINEERING

CHAPTER I: INTRODUCTION

Software Engineering History

The term **software engineering** was suggested at conferences organized by NATO in 1968 and 1969 to discuss the **software crisis**. The software crisis was the name given to the difficulties encountered in developing large, complex systems in the 1960s. It was proposed that the adoption of an engineering approach to software development would reduce the costs of software development and lead to more reliable software.

1.1. Professional Software development

- Lots of people write programs. People in business write spreadsheet programs to simplify their jobs, scientists and engineers write programs to process their experimental data. The vast majority of software development is a professional activity where software is developed for a specific task. The economies of ALL developed nations are dependent on software .

Software engineering is intended to support professional software development, rather than individual programming. Many people think that software is another word for computer programs. However, when we are talking about software engineering ,software is not just the programs themselves but also all associated documentation and configuration data that is required to make these programs operate correctly. A professionally developed software is more than a single program. The following questions are formulated in order to help you to get a broad view of what software engineering is about.

- What is a system?
- What is software?
- What are the attributes of good software?
- What is Engineering?
- What is software engineering?
- What are the fundamental software engineering activities?
- What is the difference between software engineering and computer science?
- What is the difference between software engineering and system engineering?
- What is a software process model?
- What are the costs of software engineering?
- What are software engineering methods?

What is a System?

A system is a set of elements(actors, processes and their interactions) that work together as a whole to achieve a given objective.

What is software?

A software is computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market. There are two kinds of software products:

1. Generic products – These are stand-alone that are produced by a development organization and sold on the open market to any customer who is able to buy them.
2. Customized (or bespoke)products –These are systems that are commissioned by a particular customer.

   The difference between these types of software is that, in generic products, the organization that develops the software controls the software specification. For custom products, the specification is usually developed and controlled by the organization that is buying the software.

What are the attributes of good software?

Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.

What is Engineering?

Engineering is defined as a discipline that aim is:

> "Creating cost-effective solutions ...... to practical problems ...... by applying scientific knowledge" [Shaw90] related to a given discipline.

What is software engineering?

Software engineering is an engineering discipline which is concerned with all aspects of software production. In this definition there are two key phrases:
1. **Engineering discipline** Engineers make things to work. They apply theories, methods, and tools where these are appropriate. They use them selectively and always try to discover solutions to problems even when there are no applicable theories and methods. Engineers also recognize that they must work to organizational and financial constraints so they look for solutions within these constraints.

2.All aspects of software production Software engineering is not just concerned with the technical processes of software development. It also includes activities such as software project management and the development of tools, methods, and theories to support software development. Engineering is about getting results of the required quality within the schedule and budget.

Software engineering is important for two reasons:

1. More and more, individuals and society rely on advanced software systems.

2. Software is cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project.

What are the fundamental software engineering activities?

Software specification, Software development, Software validation and Software evolution.
Specification - what the system should do and its development constraints
Development - production of the software system
Validation - checking that the software is what the customer wants
Evolution - changing the software in response to changing demands(**if you want to modify something**).

What is the difference between software engineering and computer science?

Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.

What is the difference between software engineering and system engineering?

System engineering is concerned with all aspects of computer-based systems development including hardware, software, and process engineering. Software engineering is a part of this more general process. System engineers are involved in system specification, architectural design, integration and deployment.

What is a software process model?

A simplified representation of a software process, presented from a specific perspective
Examples of process perspectives are
• Workflow perspective - sequence of activities
• Data-flow perspective - information flow
• Role/action perspective - who does what
Generic process models
• Waterfall
• Evolutionary development
• Formal transformation
• Integration from reusable components

What are the costs of software engineering?

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

What are software engineering methods?

Structured approaches to software development which include system models, notations, rules, design advice and process guidance

- **Model descriptions**

Descriptions of graphical models which should be produced

- **Rules**

Constraints applied to system models

- **Recommendations**

Advice on good design practice

- **Process guidance**

What activities to follow

1.2. Software engineering diversity

Software engineering is a systematic approach to the production of software that takes into account practical cost, schedule, and dependability issues, as well as the needs of software customers and producers. Perhaps the most significant factor in determining which software engineering methods and techniques are most important is the type of application that is being developed. There are different types of application including:

Stand-alone applications

These are application systems that run on a local computer such as a PC. They include all necessary functionality and do not need to be connected to a network.
Example: Microsoft office

Interactive transaction-based applications

These are applications that execute on a remote computer and that are accessed by users from their own PCs or terminals.
Example: web application such as e-commerce application.

Embedded control systems

These are software control systems that control and manage hardware devices.
Example: software in a cell phone, software that controls anti-lock braking in a car

Batch processing systems

These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.
Example: salary payment system

Entertainment systems

These are systems that are primarily designed for personal use and which are intended to entertain the user. Most of these systems are games of one kind or another.

System for modelling and simulation

These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interactive objects.

Data collection systems

These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.

Systems of systems

These are systems that are composed of a number of other software systems.

1.3. Software engineering and the web

Instead of writing software and deploying it on user's PCs, the software was deployed on a web server. This made it much cheaper to change and to upgrade the software, as there was no need to install the software on every PC. It also reduced costs.

1.4. Software engineering ethics

Like other engineering disciplines, software engineering is carried out within a social and legal framework that limits the freedom of people working in that area. As a software engineer, you should not use your skills and abilities to behave in a dishonest way or in a way that will bring disrepute to the software engineering profession. However, there are areas where standards of acceptable behaviour are not bound by laws but by the more tenuous notion of professional responsibility. Some of these are:

Confidentiality you should normally respect the confidentiality of your employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.

Competence you should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.

Intellectual property rights you should be aware of local laws governing the use of intellectual property such as patents and copyright.

Computer misuse you should not use your technical skills to misuse other people's computer.

Software engineering Code of Ethics and Professional Practice

Preamble

The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC

   Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER

   Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT

Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

4. JUDGMENT

Software engineers shall maintain integrity and independence in their professional judgment.

5. MANAGEMENT

Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

6. PROFESSION

Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

7. COLLEAGUES

Software engineers shall be fair to and supportive of their colleagues.

8. SELF

Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

CHAPTER II: SOFTWARE PROCESS MODEL

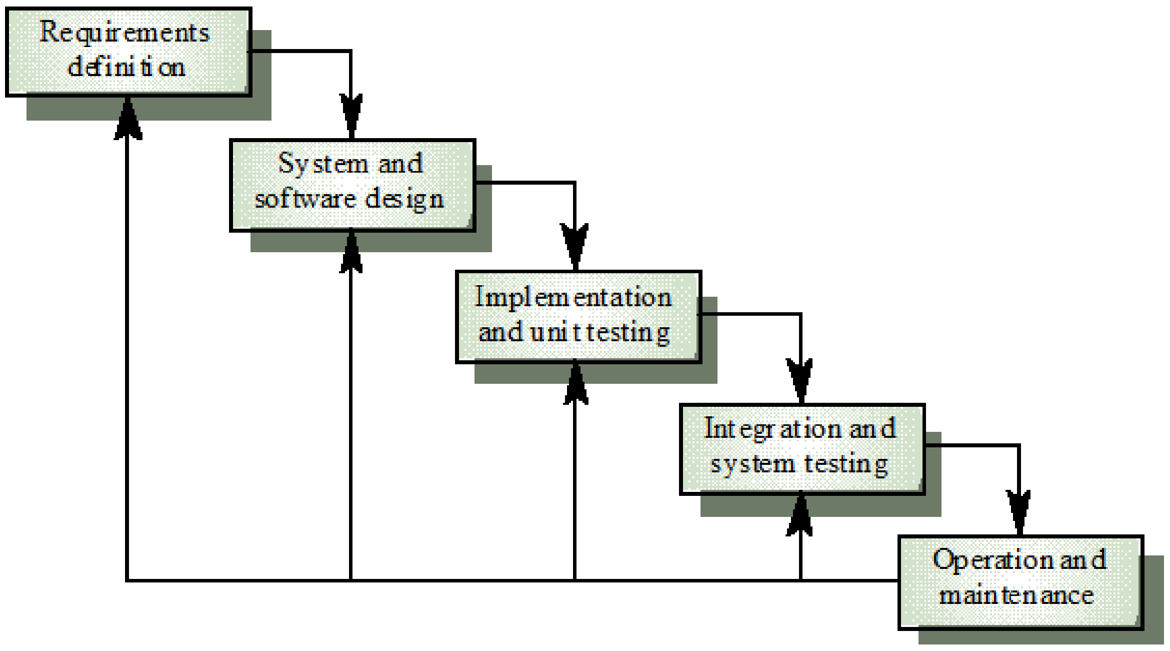
2.1. Process model

A software process is a simplified representation of a software process presented from a specific perspective. It describes the workflow and how the process elements interact together.

Process models help in the software development and guide the software team through a set of framework activities. Process Models may be linear, incremental or evolutionary. There different process models such as the following: waterfall model, evolutionary process models, Incremental process model, Unified process models,…

2.1.1. The Waterfall Model

Waterfall model consists of a number of dependent phases that are executed in a sequential order. The waterfall model maintains that one should move to a phase only when its preceding phase is completed and perfected. The complete solution is not released until the final phase.


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Requirement Analysis & Definition: This phase is focused on possible requirements of the system for the development are captured. Requirements are gathered subsequent to the end user consultation.

System & Software Design: Prior to beginning the actual coding, it is inevitable to understand what actions are to be taken and what they should like. The requirement specifications are studied in detail in this phase and the design of the system is prepared. The design specifications are the base for the implementation and unit testing model phase.

Implementation & Unit Testing: Subsequent to receiving the system design documents, the work is shared into various modules and the real coding is commenced. The system is developed into small coding units. These units are later integrated in the subsequent phase. Every unit is tested for its functionality.

Integration & System Testing: The modules that are divided into units are integrated into a complete system and tested for proper coordination among modules and system behaves as per the specifications. Once the testing is completed, the software product is delivered to the customer.

Operations & Maintenance: It is a never ending phase. Once the system is running in production environment, problems come up. The issues that are related to the system are solved only after deployment of the system. The problems arise from time to time and need to be solved; hence this phase is referred as maintenance.

Problems in Waterfall Model

Appropriate when the requirements are well-understood from the beginning, which is often difficult.
Inflexible partitioning of the project into distinct stages (Real projects rarely follow the

sequential flow). All delivery is at the end: no available working model early in the project time plan.

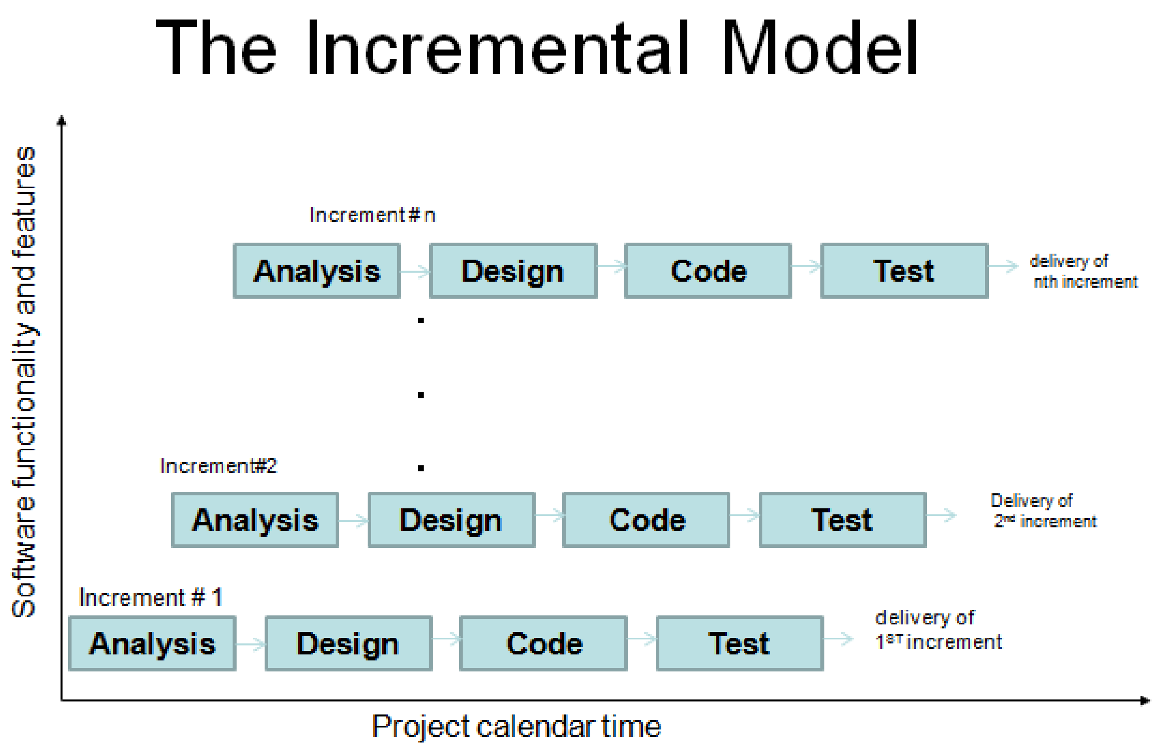
2.1.2. The Incremental Process Model

Used when initial requirements are reasonably well-defined
Software is developed in increments: development and delivery is broken down into increments: each increment delivers part of the required functionality
Lower risk of overall project failure
User requirements are prioritised and the highest priority requirements are included in early increments

Advantages

Incremental process model has 3 important benefits compared to waterfall:

- The cost of accommodating changing customer requirements is reduced- analysis and documentation is less
- It is easier to get customer's feedback on the development work done
- More rapid delivery and deployment of useful deployment of useful software is possible even if all functionality is not included

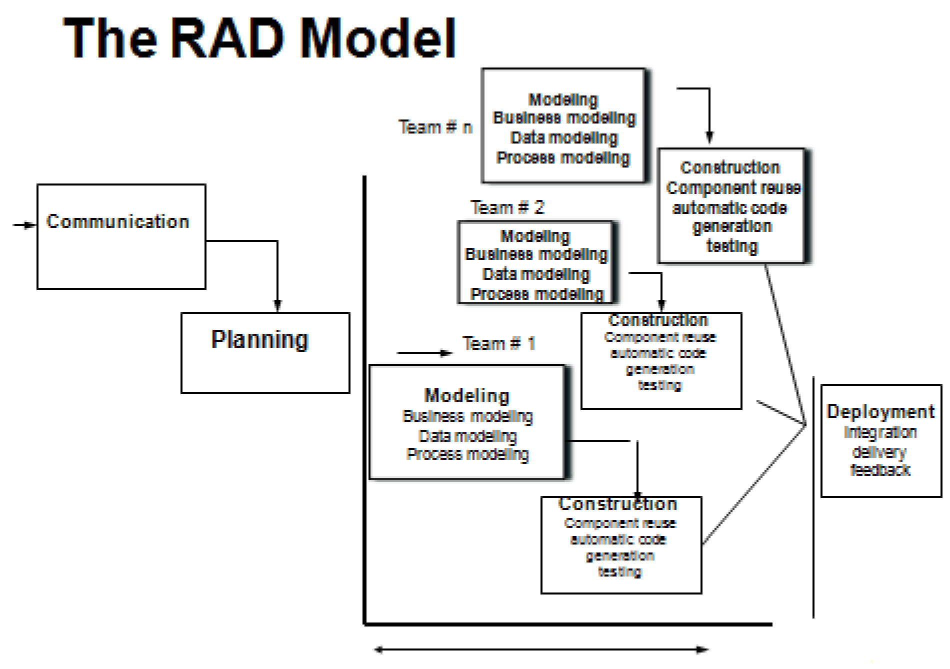


Problems with Incremental Model

- Due to bureaucratic procedures in large organizations that evolve overtime there may be mismatch between these procedures and a more informal iterative process
- Sometimes procedures may be for good reasons for e.g. to ensure software implements external regulations. Changing procedures may not be possible so process conflicts may be unavoidable.

2.1.3. The Rapid Application Development (RAD) Model

- An incremental software process model
- Short development cycle
- High-speed adoption of the waterfall model using a component based construction approach
- Creates a fully functional system within a very short span time of 60 to 90 days
- Useful where requirements are well understood and scope is limited
- Construction uses reusable components, automatic code generation and testing



Issues in RAD Model

- Requires many RAD teams
- Requires commitment from both developer and customer for rapid-fire completion of activities
- Not suitable when technical risks are high

2.1.4. Iterative Model

Each release of Iterative Model is developed in a specific, fixed time period called iteration. Each iteration focuses on a certain set of requirements. Each cycle ends with a usable system i.e. each iteration results in an executable release. The iterative model approach is to iterate on steps as the project progresses with requirements. Iterative model iterates Requirements, Design, Build and test phases again and again for each requirement and builds up a system iteratively till the complete system is built. The advantage is that iterative model can accommodate changes in requirements which are very common in most of the projects. It also provides an opportunity to identify and build upon any major requirement or design flaws throughout the process because of its iterative nature.

2.1.5. EVOLUTIONARY PROCESS MODEL

Software evolves over a period of time
Business and product requirements often change as development proceeds making a straight-line path to an end product unrealistic
Evolutionary models are iterative and as such are applicable to modern day applications
Types of evolutionary models:

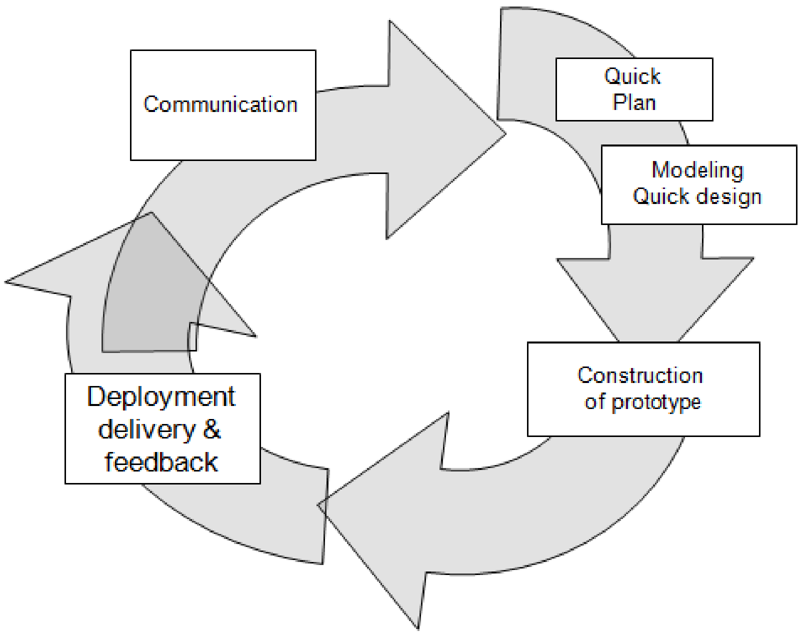
Prototyping

Spiral model

PROTOTYPING

- Mock up or model (throw away version) of a software product
- Used when customer defines a set of objective but does not identify input, output, or processing requirements
- A prototype is built to understand the requirements
- Use existing program fragments, program generators to throw together working version
- Prototype evaluation leads to redefining requirements
- process is iterated until customer and developer are satisfied

Evolutionary Models: Prototype



Steps in Prototyping

- Requirement gathering: identify whatever requirements are known
- Outline areas where further definition is mandatory
- A quick design
- Construction of prototype
- Prototype is evaluated by the customer
- Requirements are refined
- Prototype is tuned to satisfy the needs of customer

Limitation in prototyping

- In a rush to get it working, overall software quality or long term maintainability are generally overlooked
- May use inappropriate Operating Systems, Programming languages and inefficient algorithms

Issues of prototyping

- Client is likely to recommend to do a few changes on the prototype instead of developing the initial software.
- Developer may get a working prototype quickly and forget why they are inappropriate.

The Spiral Model

- An evolutionary model which combines the best feature of the classical life cycle and the iterative nature of prototype model
- Include new element : Risk element
- Starts in middle and continually visits the basic tasks of communication, planning, modeling, construction and deployment
- Each cycle produces something to be evaluated but not necessarily a usable system
- Spiral model is simplified form of Waterfall model plus risk analysis

Each phase begins by:

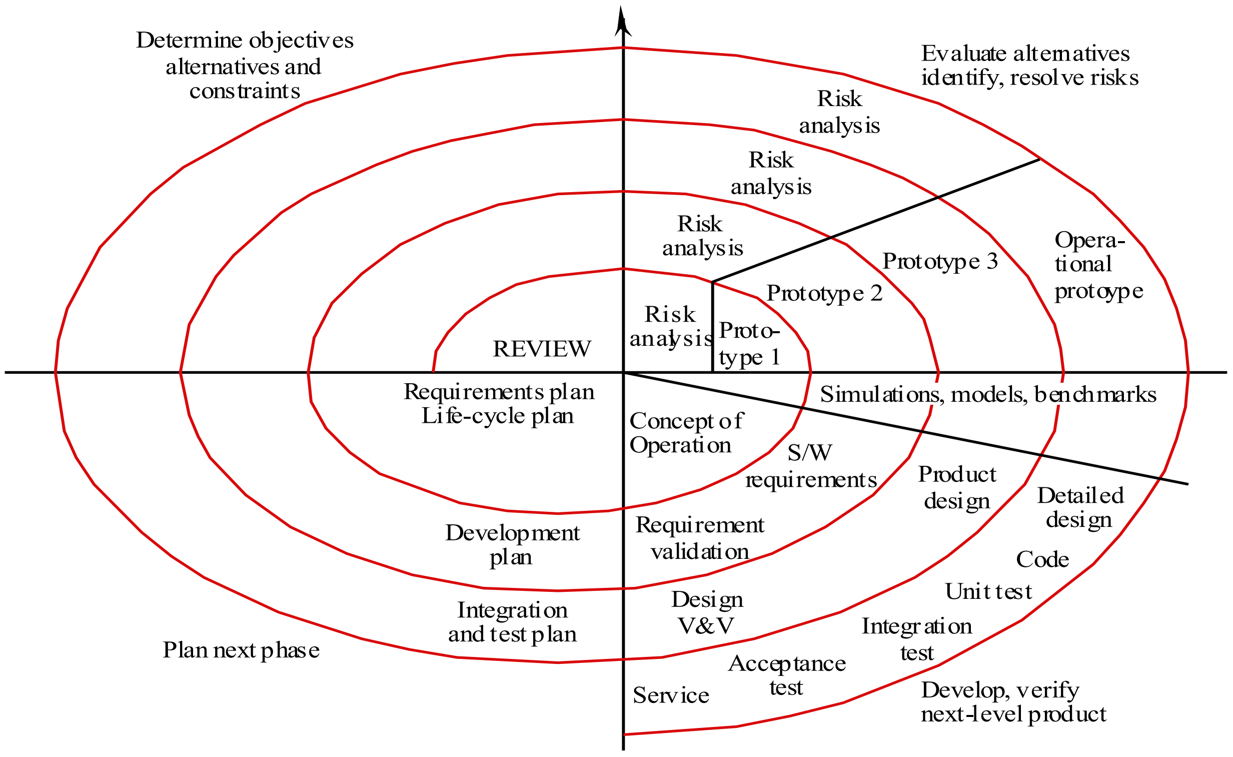
Alternatives

Risk analysis

Each phase is followed by:

Evaluation

Planning of next phase



- Realistic approach to the development of large scale system and software
- Software evolves as process progresses
- Better understanding between developer and customer

- The first circuit might result in the development of a product specification
- Subsequent circuits develop a prototype
- And then sophisticated version of software

There are also **Specialized Process Models** such as the following**: Component-Based Development,** **Formal Method model** which uses mathematical notations to construct programs.

2.2. Process activities

There are four basic activities in different development processes such are the following: specification, development, validation and evolution.

2.2.1. Software specification

Software specification or requirement engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development. There are 4 main activities in the requirements engineering process such as follows: feasibility study, requirements elicitation and analysis, requirements specification and requirements validation.

CHAPTER III. SOFTWARE REQUIREMENTS

3.1. Definition

Software requirement is a condition or capability that must be met or possessed by a system or a system component to satisfy contract, standard, specification or formally imposed document. Software requirements can be classified as follows:

- **user requirements and system requirements**
- **Functional requirements and non functional requirements**

3.1.1. User requirements and system requirements

- **User requirements** are statements, in natural language plus diagrams, of what services the system is expected to provide and the constraints under which it must operate.
- **System Requirements** are more detailed description of the software system's function, services and operational conditions.
  The system requirements document (also called a functional specification) should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.

Examples of user requirement definition and System requirements Specification for a mental health care patient management system.

User requirement definition

The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month

System requirements
1.1. On the last working day of each month, a summary of drugs prescribed, their cost , and the prescribing cl
1.2. The system shall automatically generate the report for printing after 17h30 on the last working day of th
1.3. A report shall be created for each clinic, listing the individual drugs names, the total number of prescripti
and the total cost of prescribed drugs.
1.4 If drugs available in different dose units (e.g. 10mg, 20mg) separate reports shall be created for each dose
1.5. Access to all cost reports shall be restricted to authorized users listed on a management access control lis

3.1.2. Functional requirements and non functional requirements

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.

Non-functional requirements

- constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.

Domain requirements

- Requirements that come from the application domain of the system and that reflect characteristics of that domain.

Functional requirements

- They are statements of services the system should provide
- Describe how the system should react to particular inputs
- Describe how the system should behave in particular situations
- Sometimes, may also explicitly state what the system should not do.

Example of functional requirements for the MHC -PMS, used to maintain information about patients

S.N.	**Software Requirements**
1.	A user shall be able to search the appointments lists for all clinics

2. The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.

3. Each staff member using the system shall be uniquely identified by his or her eight-digit employee number.

Functional Requirements should be both complete and consistent.

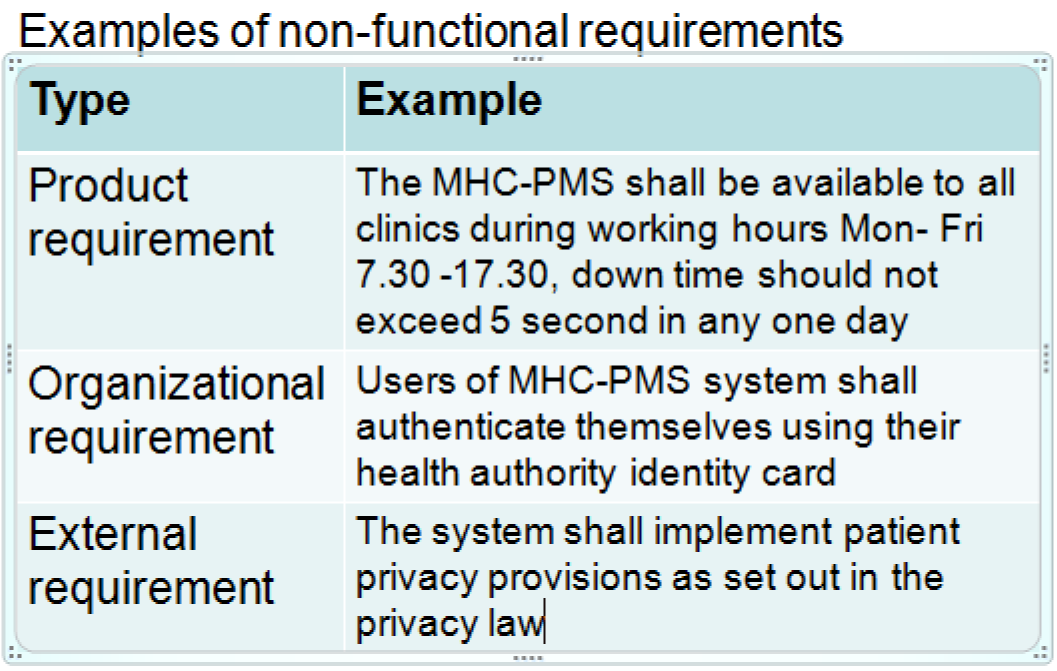
- **Completeness** means that all services required by the user should be defined
- **Consistency** means that requirements should not have contradictory definitions.

Difficult to achieve completeness and consistency for large and complex systems

Non-functional requirements:

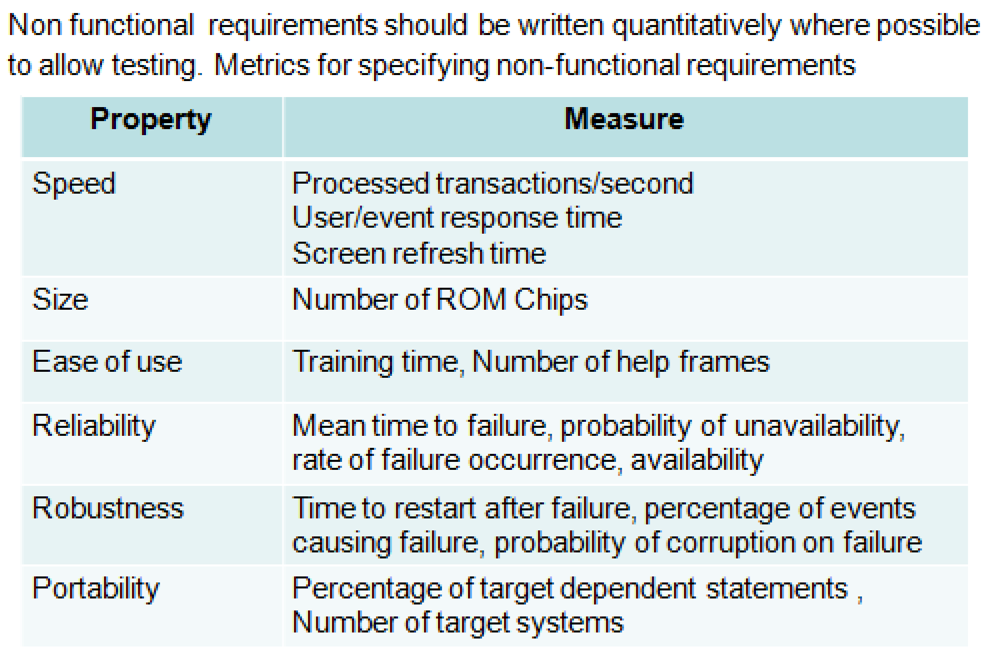
- Not directly concerned with specific services to be delivered by the system
- Are constraints on the services or functions offered by the system
- Include timing constraints, constraints on the development process and standards
- Apply to system as a whole and are more critical than a functional requirement

E.g. An aircraft system reliability requirements. An aircraft **must** be certified as safe for operation.



Examples of non-functional requirements

Type	Example
Product requirement	The MHC-PMS shall be available to all clinics during working hours Mon- Fri 7.30 -17.30, down time should not exceed 5 second in any one day
Organizational requirement	Users of MHC-PMS system shall authenticate themselves using their health authority identity card
External requirement	The system shall implement patient privacy provisions as set out in the privacy law



Non functional requirements should be written quantitatively where possible to allow testing. Metrics for specifying non-functional requirements

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Number of ROM 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 corruption on failure
Portability	Percentage of target dependent statements , Number of target systems

3.2. Software requirements document

Also called Software requirement specification (SRS) is an official statement of what is required of the system developers. It includes user requirements for a system and detailed specification of the system requirements. At times user and system requirements are integrated into a single description other times user requirements are put in the introduction of SRS. It is NOT a design document. It sets WHAT the system should do not HOW it should do it.
Users of requirements document are: system customers, Managers, system engineers, system test engineers, system maintenance engineers.
There are 6 conditions that requirements document should satisfy. It should:

- specify only external system behavior
- specify constraints on the implementation.
- Be easy to change
- Serve as reference tool for system maintainers
- Record forethought about the life cycle of the system.
- Characterize acceptable responses to undesired events.

Purpose of SRS :

- Communication between the Customer, Analyst, System Developers, Maintenance Teams, etc
- Firm foundation for the design phase
- Support system testing activities
- Support project management and control
- Controlling the evolution of the system

The structure of a requirements document

- Preface
- Introduction
- Glossary
- User requirements definition
- System architecture
- System requirements specification
- System models
- System evolution
- Appendices
- Index

3.3. Requirements engineering processes

Requirements engineering processes include four high-level activities.

- **Feasibility study**: usefulness of the system
- **Elicitation and analysis**: requirements
- **Specification**: requirements aligned to standards
- **Validation**: requirements meet the user needs

Activities of this process are organized as in iterative process around a spiral, with the output being a system requirements document.

Feasibility study

A feasibility study is a short, focused study that should take place early in RE process.

It provides answers to 3 main questions:

- Does the system contribute to the overall objective of the institution?
- Can the system be implemented within time, and budget using current technology?
- Can the system be integrated with other systems in use?

Requirements Elicitation and analysis

Software engineers work with customers and end-users to find out information about the application domain, services of the system, performance hardware constraints, etc

Activities involved are:

Requirements discovery: the process of interacting with stakeholders of the system to know their requirements

Requirement classification and organization: this activity takes the unstructured collection of requirements, groups related requirements and organizes them into coherent clusters.

Requirements prioritization and negotiation: conflicts may arise from different stakeholders' requirements, this activity prioritizes requirements and find and resolve requirements conflicts through negotiation.

Requirements specification: requirements are well documented for future use.

Requirements specification

Requirement specification is the process of writing down the user and system requirements in a requirement document. The requirements should be clear, unambiguous, easy to understand, complete and consistent.

Ways of writing system requirements specification:

- Natural language sentences
- Structured natural language
- Design description languages
- Graphical notation
- Mathematical specifications

Requirements validation

- This is an activity of checking that requirements actually define the system that the customer wants.
- It has to solve problems with the requirements. Errors in requirements lead errors in design and errors in coding
- Cost of fixing requirements problem by making system change is greater than repairing design or coding errors.

During validation, different checks are carried out on the requirements in the requirements document. They include:

1. Validity checks
2. Consistency checks: no conflicts
3. Completeness checks: all functions
4. Realism checks: for implementation
5. Verifiability: system meets the needs

There are a number of requirement validation techniques that can be used individually or in conjunction with one another:

- **Requirement reviews:** team of reviewers check requirements for errors and inconsistencies.
- **Prototyping:** an executable model of the system is shown to the end-users to experiment if it meets their real needs.
- **Test-case generation:** requirements are tested, they should be testable. If not, the requirement will be difficult to implement. So it has to be revised. Developing tests form user requirements is an integral part of extreme programming.

3.4. Requirements management

Definition

Requirement management is a process of understanding and controlling changes to system requirements.

Requirements management planning
Planning is an essential stage of requirement management process

- Planning establishes level of requirements management detail that is required.
- Decision to be taken in planning are related to:
  - Requirement identification,
  - Change management process
  - Traceability policies (define the relationships between each requirement and between requirements and the system design.
  - and tool support
- Requirement management needs automated support and the software tools for this should be chosen during the planning phase:
- Tools needed are to support:
  - Requirements storage
  - Change management
  - Traceability management

Requirements change management

- Requirement change management should be applied to all proposed changes to a system's requirements after the requirement document has been approved.
- Change management is essential to decide if the benefits of implementing the new requirements are justified by the cost of implementation.

There are 3 principles to a change management process:

1. Problem Analysis and Change Specification
2. Change Analysis and Costing
3. Change Implementation

1. Problem Analysis and Change Specification

➢ Identified problem requirement with change proposal
➢ Validity of problem and proposal are analysed
➢ Analysis id fed back to requestor who may provide more
➢ specification or withdraw the request

2. Change analysis and costing: Assess traceability

➢ information the impact of the change in the
➢ requirement document, design and implementation and

- ➢ decide if it worth making the requirement change

3. Change Implementation

Modification of the requirement document, and where necessary, design and implementation.

CHAPTER IV. SYSTEM MODELING

4.1. Definition

System modeling is the process of developing abstract representation (model) of a system, with each model representing a different view or perspective of that system. It is the representation of the system using some kind of graphical notation mainly based on the Unified Modeling Language (UML). Software modeling helps the engineer to understand the functionality of the system. Models are used for communication among stakeholders:

- During the requirements engineering process to help derive the requirements of a system;
- During the design process to describe the system to engineers implementing the system
- After implementation to document the systems structure and operation.

Models may be developed for both existing system and the system to develop.

4.2. The Unified Modeling Language(UML)

Has been developed by the developers of object-oriented analysis and design methods;

Is a set of 13 different diagram types that may be used to model software systems.

Is universally accepted as the standard approach for developing models of software systems.

The UML diagram types:

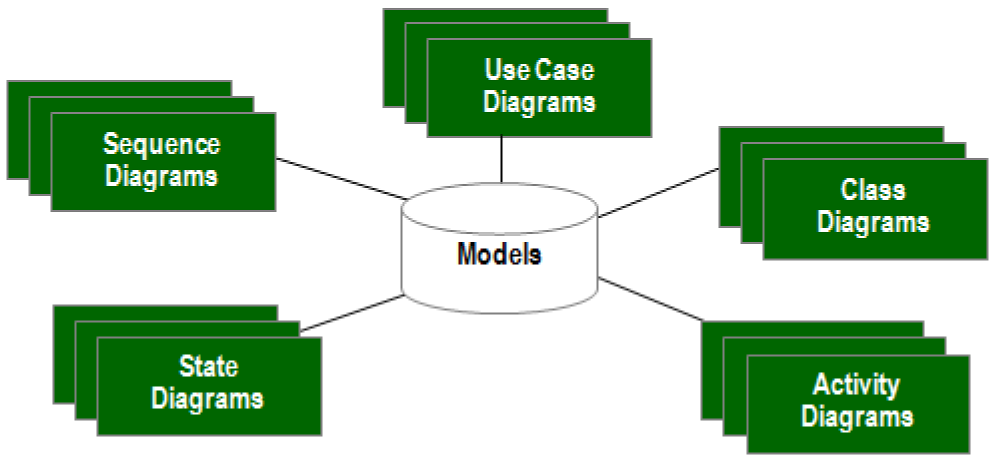
Activity diagrams show the activities involved in a process or in data processing

Use case diagrams show the interactions between a system and its environment

Sequence diagrams show interactions between actors and the system and between the system components

Class diagrams show object classes in the system and the association between these classes

State diagrams show how the system reacts to internal and external events.



Some of UML symbols

- Filled circle: start of a process
- Filled circle inside another circle: end of process
- Rectangles with round corners : activity

(sub-activity to be carried out)

- Arrows: flow of work from activity to another
- Solid bar: 1. activities leading to the bar must be completed before progress is possible 2. Solid bar leads to a number of activities they may be executed in parallel.
- Arrows annotations: indicates the condition when the flow is taken.

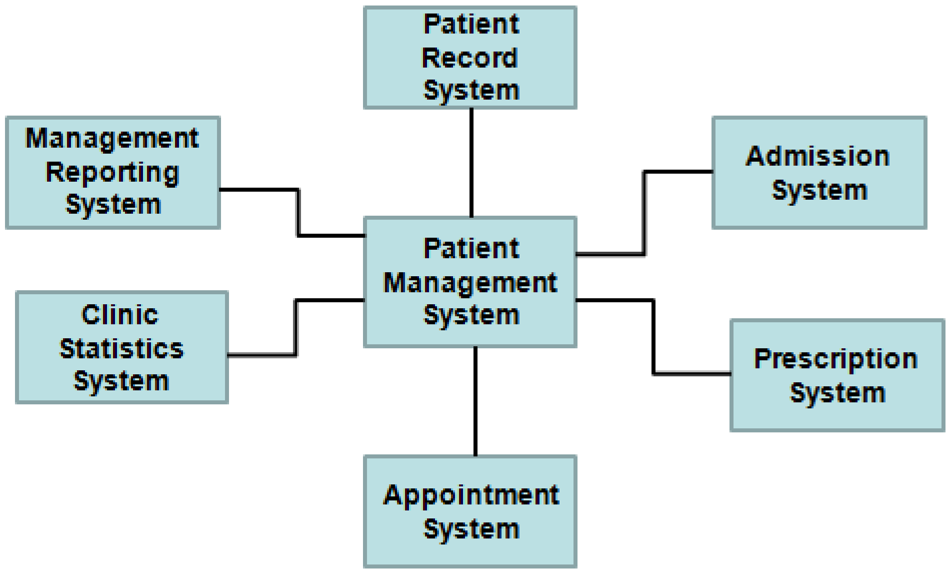
4.3. Context models

Context models show other systems that are related in a way or another to the system to develop.

External systems might:

- produce for or consume data from the system
- share data with the system
- be connected directly or not connected
- be physically located in the same place or in different buildings

Figure: the Context of a Patient Management System



4.4. Interaction models

All system involve interaction:

- User interaction involving user inputs and outputs: helps to identify user requirements
- Interaction between the system being developed and other systems: highlights communication problems that may arise
- Interaction between the components of the system: checks if a proposed system structure is likely to deliver the required system performance and dependability.
- **Use case modeling** is mostly used to model interaction between a system and external actors (users or other systems)
- **Sequence diagrams** are used to model interactions between system components

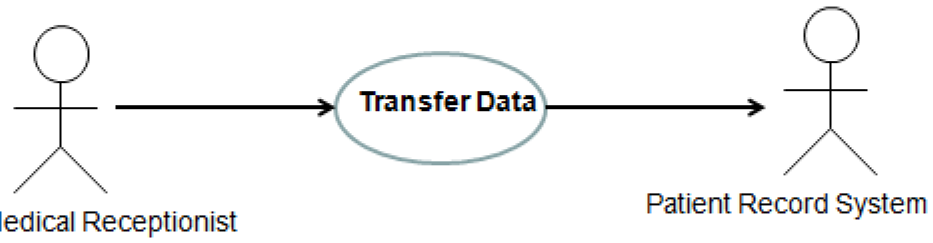
Use case models and Sequence diagrams present interaction at different levels of detail, so they may be used together.

4.4.1. Use case modelling

Use case modeling is mostly used to model interaction between a system and external actors (users or other systems)

- Used to model interaction between a system and external actors
- Widely used to support requirement elicitation
- A simple scenario that describes what the user expects from a system
- Each use case represents a distinct task that involves external interaction with a system

Figure: Transfer data use case



Note: Use case model is represented with an ellipse
the actors represented as stick figures.
Use of arrows, in UML they indicate direction

4.4.2. Sequence diagrams

- Used to model interactions between the actors and the objects in a system and between the object themselves.
- Show sequence of interactions that take place during a particular use case.

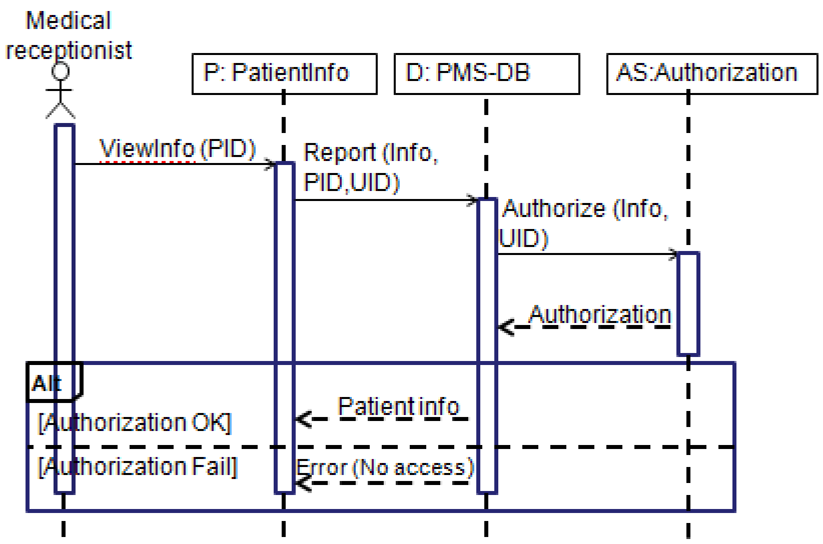


Figure: Sequence diagram for view patient information

Interactions between objects are indicated by annotated arrows, the rectangle on the dotted lines show the lifeline of the object concerned; annotations indicate call to the object, their parameters and return values. The box **Alt** denote alternatives used with condition in brackets

4.5. Structural models

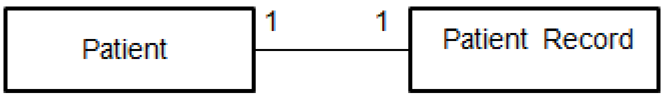
Structural models Display the organization of a system in terms of the components that make up that system and their relationships. Structural models are created when discussing and designing the system architecture.

4.5.1. Class diagrams

- Class diagrams are used when developing an object-oriented system model to show classes in a system and the association between these classes.
- In the early stages of S.E. process, objects represents something in real world (patient, a prescription, a doctor,…)

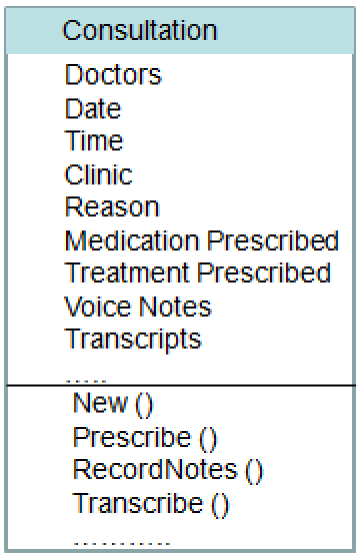
Class diagrams cont.

Figure: UML Classes and association



- Class name in a box
- An association is shown by a line between classes.
- This is a 1:1 relationship
- Multiplicities are possible as per the next example . Multiplicities are indicated either by a number or by * for indefinite number of objects.

Figure the consultation class:



In the UML attributes and operations are shown by extending the simple rectangle that represents a class

1. Name of class in the top section
2. Class attributes in the middle
3. Operations (methods in OO programming languages) in the lower section

CHAPTER V. ARCHITECTURAL DESIGN

5.1. Definition

Architectural design is concerned with understanding how a system should be organized and designing the overall structure of that system.

The architectural design:

- Is an early stage of the system design process.
- Represents the link between specification and design processes.
- Often carried out in parallel with some specification activities.
- It involves identifying major system components and their communications.

Advantages of explicit architecture:

- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- Large-scale reuse
  - The architecture may be reusable across a range of systems.

5.2. Architectural design decisions

- Architectural design should usefully be thought of a series of decisions to be made rather than a sequence of activities.
- Architectural design is a creative process so the process differs depending on the type of system being developed.
- However, a number of common decisions span all design processes.

In order to decide architectural design which can be used, you can answer the following questions:

- Is there a generic application architecture that can be used?
- How will the system be distributed across a number of cores or processors?
- What architectural styles are appropriate?
- What approach will be used to structure the system?
- How will the system be decomposed into modules?
- What strategy should be used to control the operation of the components in the system?
- What architectural organization is best for delivering the non-functional requirements of the system ?
- How will the architectural design be evaluated?
- How should the architecture be documented?

5.3. Architectural views

- Architectural model of a software can be used to focus discussions about the software requirement, design, or to document a design.
  - What views or perspectives are useful when designing and documenting a system's architecture?
  - What notations should be used for describing architectural models?
- The 4+1 view model has four fundamental architectural views related using use cases or scenarios. Those views are:

A logical view

shows the key abstractions in the system as objects or object classes. In this view, you can relate the system requirements with entities.

A process view

show how, at run-time, the system is composed of interacting processes. Useful for making judgement about non-functional requirements.

A development view

shows how software is decomposed for development, into components developed by a developer or development team. Useful for software managers or programmers

A physical view

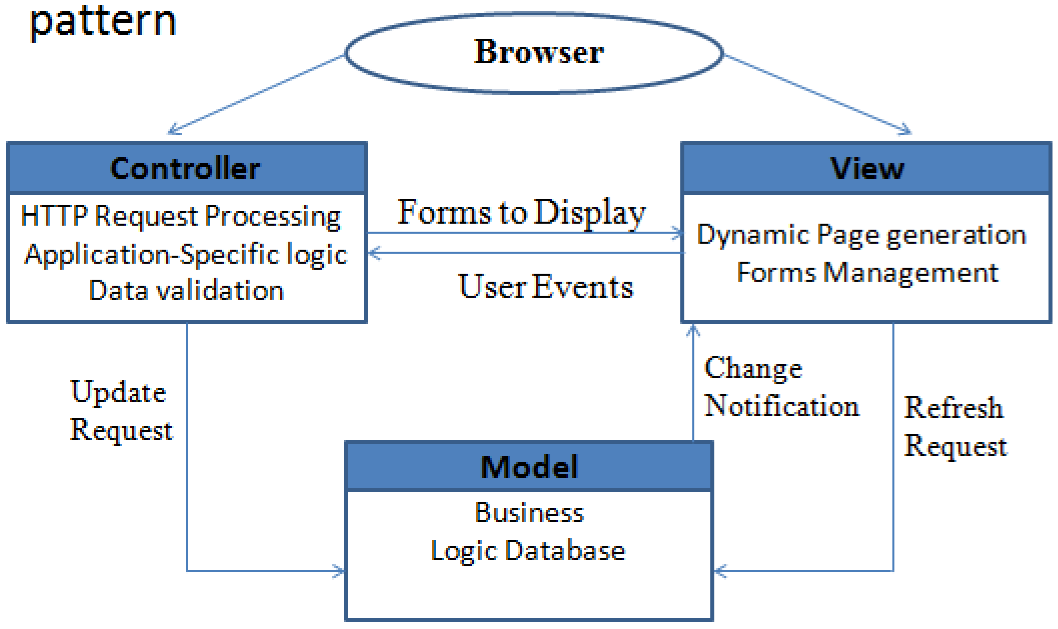
shows the system hardware and how software components are distributed in the system. Useful for system engineers for planning a system deployment

5.4. Architectural patterns

Architectural pattern is a stylized, abstract description of good practice, which has been tried and tested in different systems and environment. It describes a system organization that has been successful in previous systems. It should have the information of when it is and is not appropriate to use that pattern.

Architectural patterns cont.

Figure: web application architecture using MVC pattern



5.5. System organization

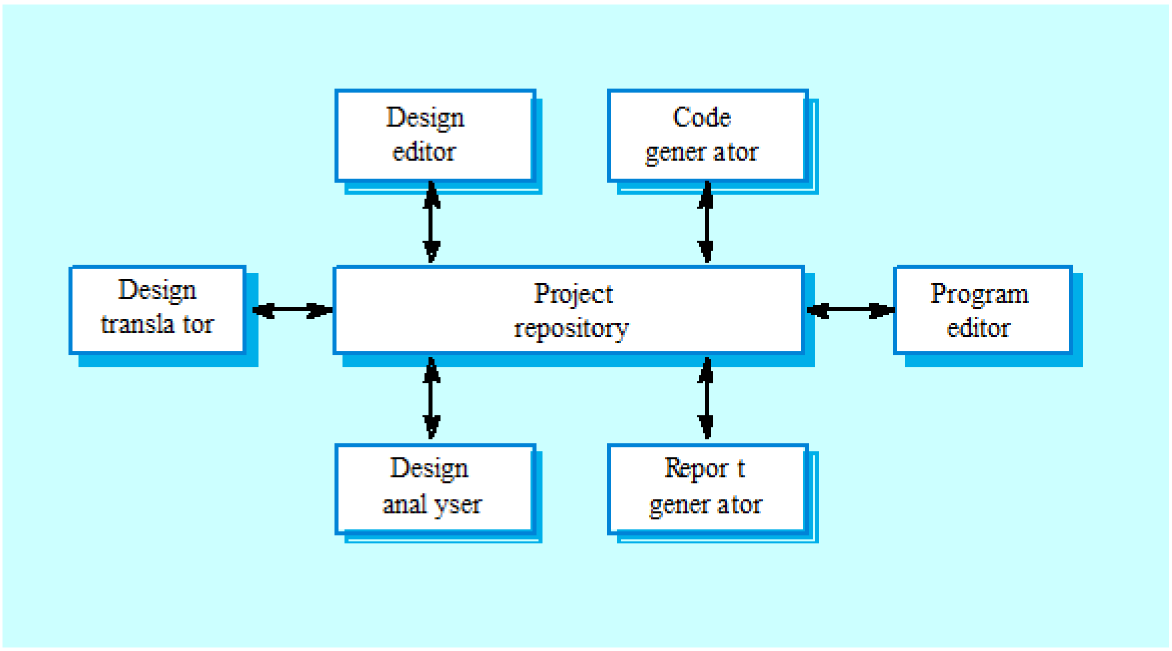
- Reflects the basic strategy that is used to structure a system.
- Four organizational styles are widely used:
  - Repository architecture
  - Client-server architecture
  - Layered architecture

– Pipe and filter architecture

5.5.1. The repository architecture

- Sub-systems must exchange data. This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems;
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems.
- When large amounts of data are to be shared, the repository model of sharing is most commonly used.
- The diagram shows an IDE that includes different tools to support model-driven development.
- Organizing tools around a repository is an efficient way to share large amounts of data.
- Components must operate on an agreed repository data model.
- In practice it is difficult to distribute the repository over a number of machines. Although it is possible to distribute a logically centralized repository, there may be problems with data redundancy and inconsistency.
- In the next example, the repository is passive and control is the responsibility of the components using the repository.

Figure: A repository architecture for an Interactive Development Environment



- **Advantages**
  - Efficient way to share large amounts of data;
  - Sub-systems need not be concerned with how data is produced and its centralised management e.g. backup, security, etc.
  - Sharing model is published as the repository schema.
- **Disadvantages**
  - Sub-systems must agree on a repository data model. Inevitably a compromise;
  - Data evolution is difficult and expensive;
  - No scope for specific management policies;
  - Difficult to distribute efficiently.

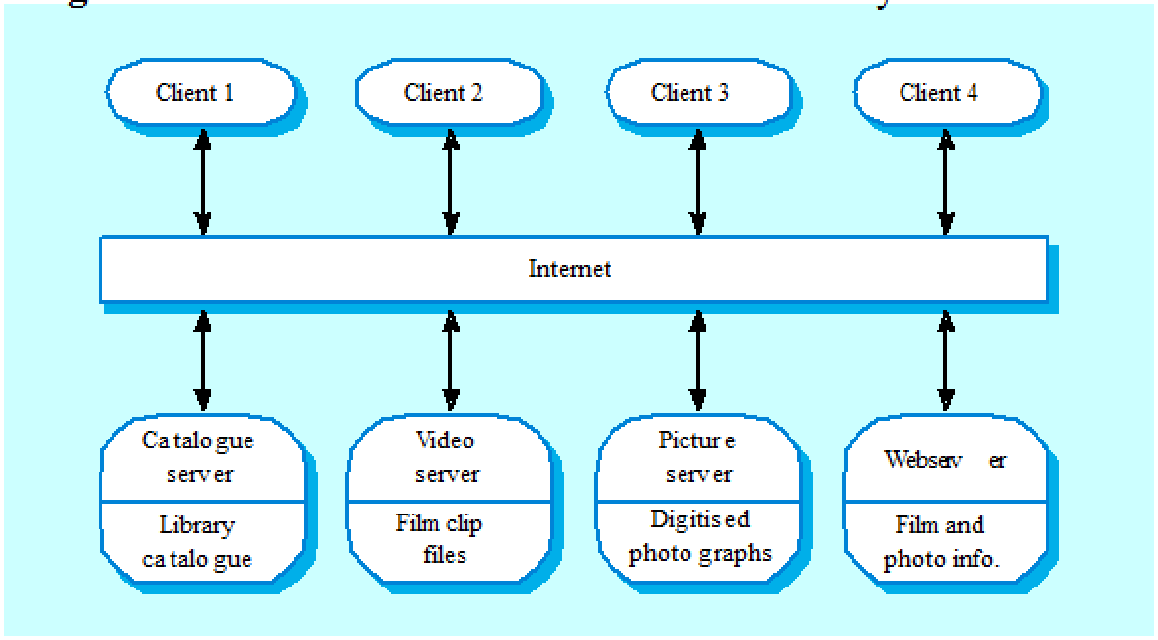
5.5.2. Client-server architecture

- Distributed system model which shows how data and processing is distributed across a range of components.

Major components of this architecture are:

1. Set of servers which provide specific services such as printing, data management, etc.
2. Set of clients which call on these services.
3. Network which allows clients to access servers.

Figure: a client-server architecture for a film library



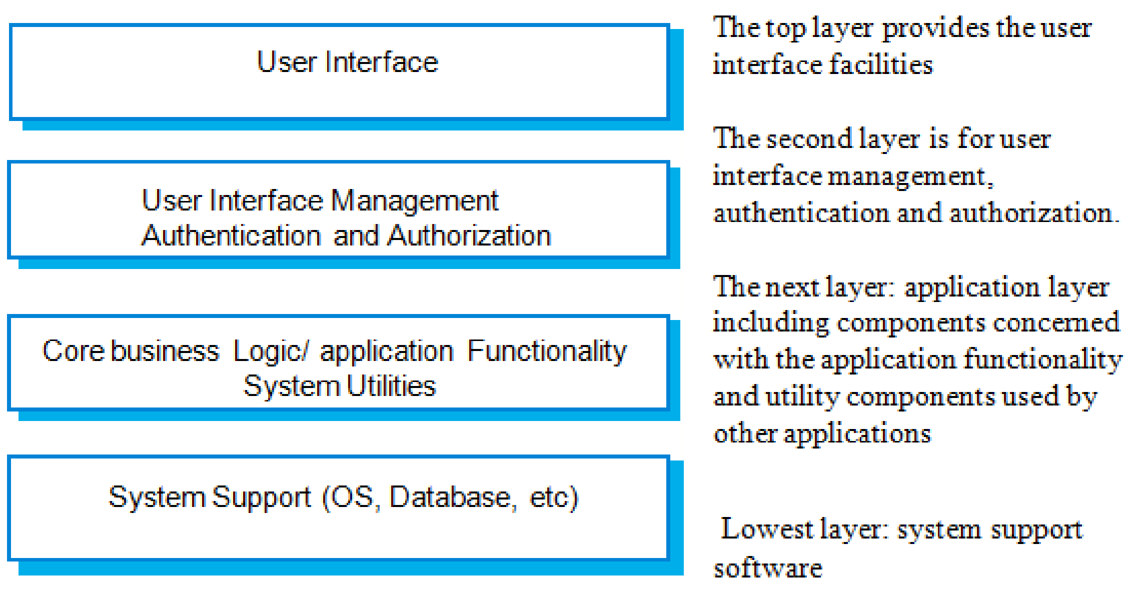
- **Advantages**
  - Distribution of data is straightforward;

  - Makes effective use of networked systems. May require cheaper hardware;
  - Separation and independence: Easy to add new servers or upgrade existing servers.
  - Clients may have to know the names of available servers and the services they provide; but the servers do not need to know the identity of clients or how many clients access their services.
- **Disadvantages**
  - No shared data model so sub-systems use different data organisation. Data interchange may be inefficient;
  - Redundant management in each server;
  - No central register of names and services - it may be hard to find out what servers and services are available.

5.5.3. Layered architecture

- The layered architecture pattern is another way of achieving separation and independence. Used to model the interfacing of sub-systems.
- Organises the system into a set of layers each of which provide a set of services.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- This architecture is changeable and portable.

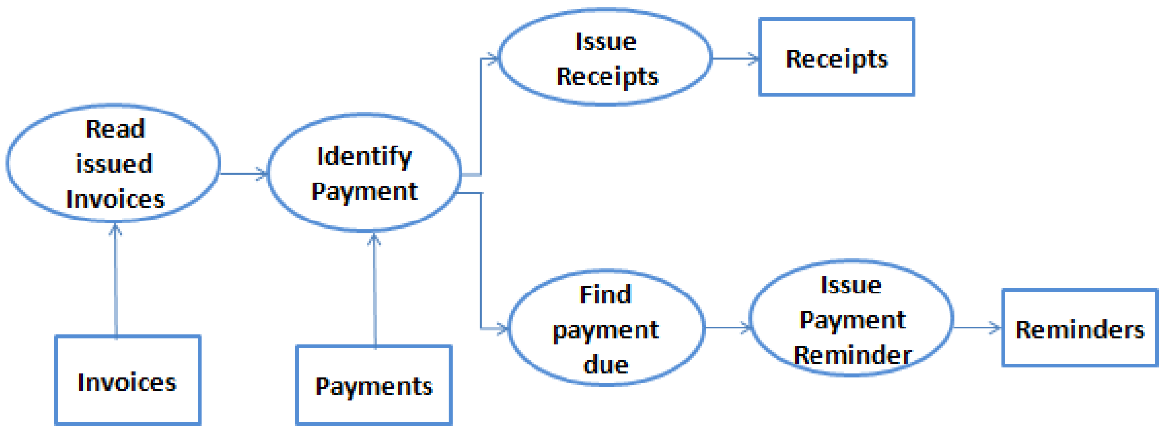
Figure: A generic layered architecture


User Interface
User Interface Management
Authentication and Authorization
Core business Logic/ application Functionality
System Utilities
System Support (OS, Database, etc)
The top layer provides the user interface facilities
The second layer is for user interface management, authentication and authorization.
The next layer: application layer including components concerned with the application functionality and utility components used by other applications
Lowest layer: system support software

5.5.4. Pipe and Filter Architecture

- This is a model of the run-time organisation of a system where functional transformations process their inputs and produce outputs. Data flows from a point to another and is transformed as it moves through the sequence.
- Each processing is executed as a transform; and transformations can be executed in parallel or sequentially.

Figure: an example of the pipe and filter architecture



5.6. Application Architectures

- The application architecture may be re-implemented when developing new systems but for many business application reuse is also possible without re-implementation

There are two types of applications:

- Transaction processing applications
- Language processing systems

5.6.1. Transaction processing applications

- These are database-centred applications that process user requests for information and update the information in a database.
- These are the most common type of interactive business systems.
- User actions cannot interfere with each other and integrity of database is maintained

e.g.: interactive banking systems, e-commerce systems, information systems, and booking systems

5.6.2. Language processing systems

- Are systems in which user's intentions are expressed in a formal language such us Java.
- The language processing system processes this language into an internal format and then interprets this internal representation.

e.g. compilers, which translate high level language programs into machine code. However, language processing systems are also used to interpret command languages for databases and information systems, and markup language such as XML (Harold and Means, 2002; Hunter et al., 2007)

5.6.3. Transaction processing systems

- Transaction processing (TP) systems are designed to process user requests for information from a database, or requests to update a database (Lewis et al. 2003)
- Technically, a database transaction is a sequence of operations that is treated as single unit (an atomic unit) . All of the operations in a transaction have to be completed before database changes are made permanent. This avoids inconsistencies in a database.
- From a user perspective, a transaction is any coherent sequence of operations that satisfies a goal, such as 'find times of buses from Kigali to Huye'.
- If the user does not require the database to be changed then it may not be necessary to package this as a technical database transaction.

Transaction systems may be organized as a ‘pipe and filter’ architecture with system components responsible for input, processing and output. For example the use of ATM to query clients accounts and withdraw cash.

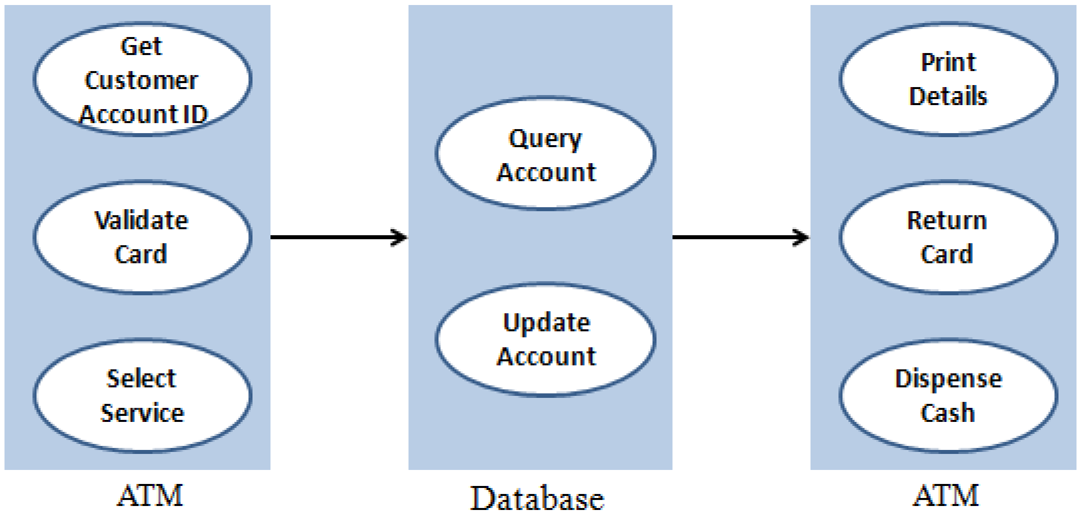


Figure: The software architecture of an ATM system

Information Systems

- All systems that involve interaction with shared database can be considered to be transaction-based information systems.
- An information system (IS) allows controlled access to a large base of information, such as library catalogue, a bus timetable, a record of patient in an hospital, or a record of students in a school.
- Increasingly, IS are web-based systems that are accessed through a web browser.

This is a general model of an IS. The system is modelled using a
layered approach; top layer supports user interface and
bottom layer is the system database.

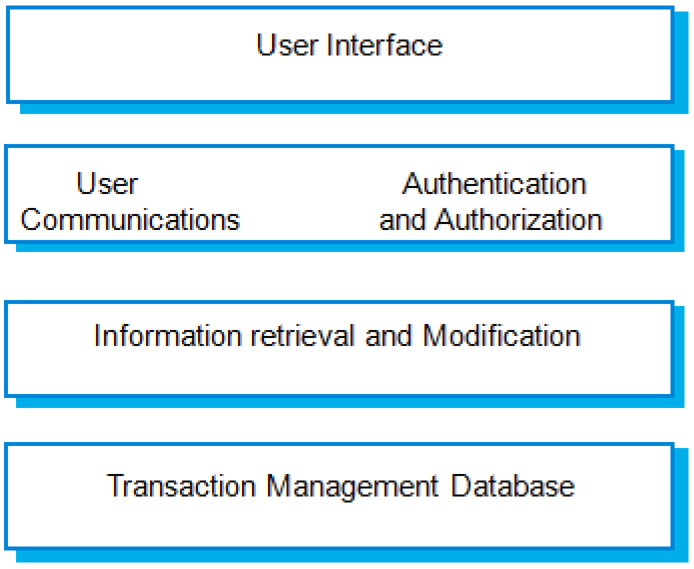

User Interface
User Communications
Authentication and Authorization
Information retrieval and Modification
Transaction Management Database

Figure: Layered information system architecture

CHAP VI: DESIGN AND IMPLEMENTATION

Software design is an activity in which you identify software components and their relationships based on customer's requirements.

Implementation is the process of realizing the design as a program. Sometimes there is a separate design stage, which is modeled and documented. Other times the design is in the programmer's head or roughly sketched on a board or a paper.

6.1. Object oriented design using UML

Object-oriented design processes involve designing object classes and the relationships between these classes. These classes define objects in the in the system and their interactions.

OO systems are easier to change

Objects include both data and operations to manipulate that data i.e. they may be understood and modified as stand-alone entities.

Changing the implementation of an object or adding services should not affect other system objects. This improves the understanding, and hence the maintainability of the design.

To develop a system design from concept to detailed OO design, there several things you need to do:

1. Understand and define the context and external interactions with the system
2. Design the system architecture
3. Identify principal objects in a system
4. Develop design models
5. Specify interfaces

Design is not a clear-cut sequential process, activities are related and influence each other.

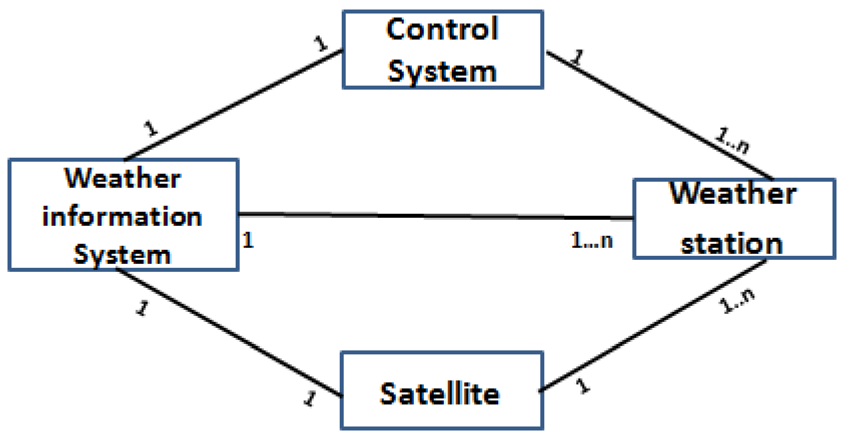
6.1.1. System context and interactions

The 1st step in any software design process is to develop an understanding of relationships between the software being developed and its external environment.
This helps setting boundaries and thus allows you to decide what features are to be implemented and the features of associated systems.
System context models and interaction models present complementary views of the relationships between a system and its environment :

1. A system model is a structural model that shows other systems in the environment of the system being developed.
2. An interaction model is a dynamic model that shows how the system being developed interacts with its environment as it is used.

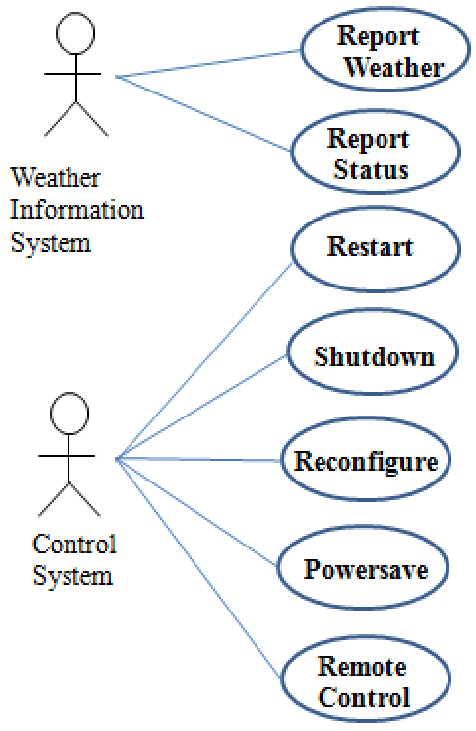
The context model of a system may be represented using associations. Associations simply show that there some relationships between the entities involved in the association. So, you may document the environment of a system using a block diagram showing entities and their associations.

Figure: System context for the weather station



The systems in the environment of each weather station are a weather IS, a satellite system and a control system. The number information on the link shows that there is 1 control system but many weather stations, 1 satellite, and 1 weather IS.

Figure: Weather station use cases



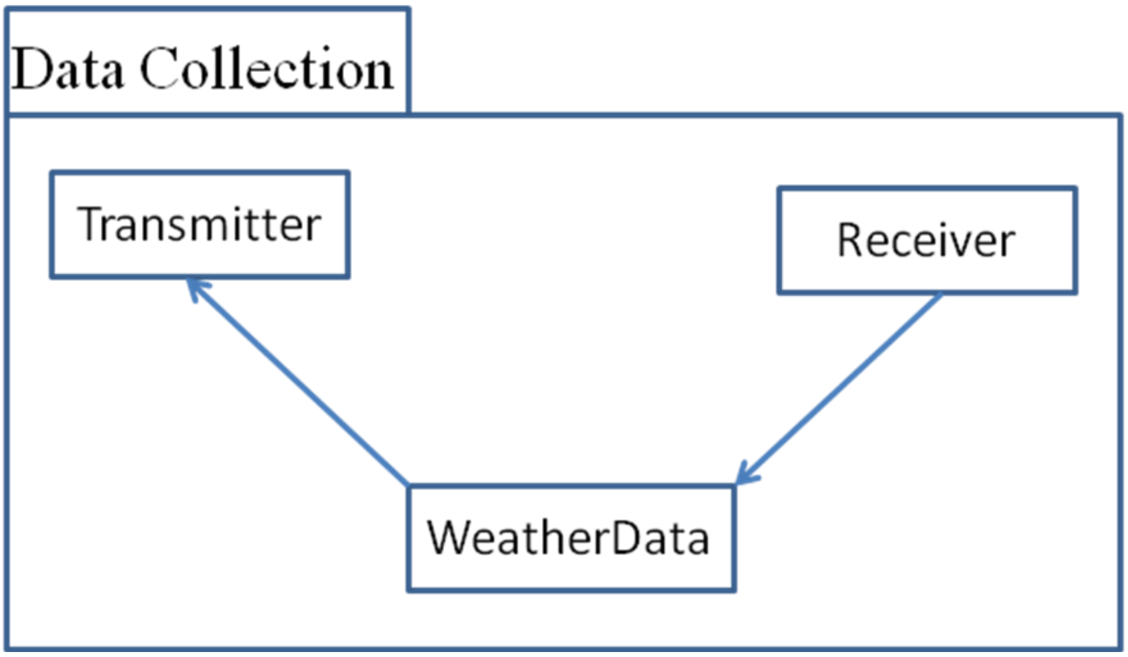
Weather station interacts with weather IS to report weather data and status of the weather station hardware.

Other interactions are with a control system that can issue specific weather station control commands

6.1.2. Architectural design

When the interaction between the Software system and the system's environment have been defined, you use this information as a basis for designing the system architecture. You need to combine it with the general knowledge of the principals of architectural design and with domain knowledge. You identify the major components of the system and their interactions and then may organize components using architectural pattern such as a layered or client-server model.

Figure: Architecture of Data collection system



Data collection subsystem is included in the weather station architecture. The transmitter and Receiver objects are concerned with managing communication and WeatherData encapsulates the information that is collected from the instruments and transmitted to the weather information system.

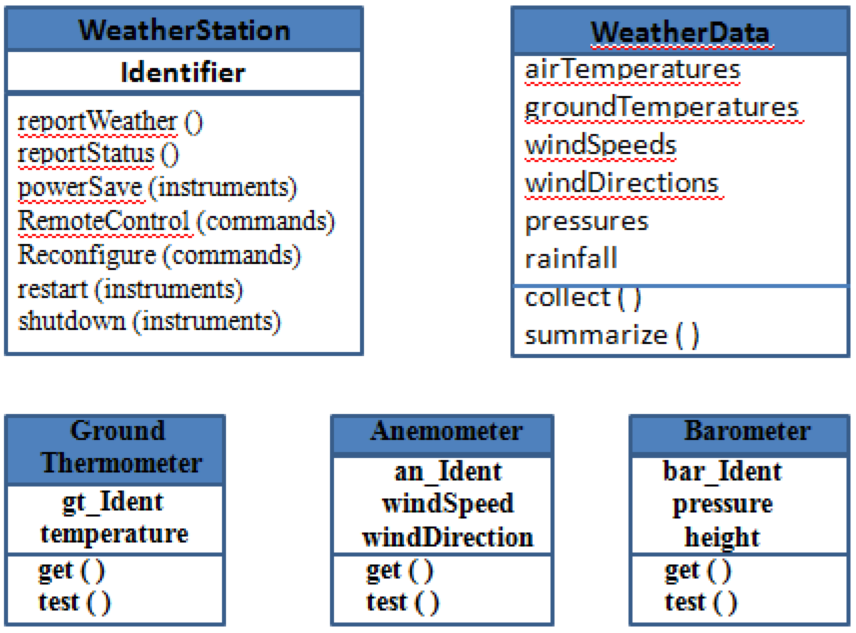
6.1.3. Object class identification

There have been many proposals on how to identify object classes in OO Systems:

1. Use a grammatical analysis of a natural language description of the system to be developed. Objects and attributes are nouns; operations and services are verbs. (Abbott,1983)
2. Use tangible entities (things) in the application domain like aircraft, roles like manager or doctor, events like requests, interactions like meetings, locations like offices, organization units like companies, etc.. (Coad and Yourdon, 1990; Shlaer and Mellor, 1988; Wirfs-Brock at al.,1990)
3. Use a scenario-based analysis where various scenarios of system are identified and analyzed in turn. As the analysis takes place, the analysts should identify required objects, attributes and operations (Beck and Cunningham, 1989).

Figure: Weather station objects

(objects identification is based on the tangible hardware in the system)



6.1.4. Design models

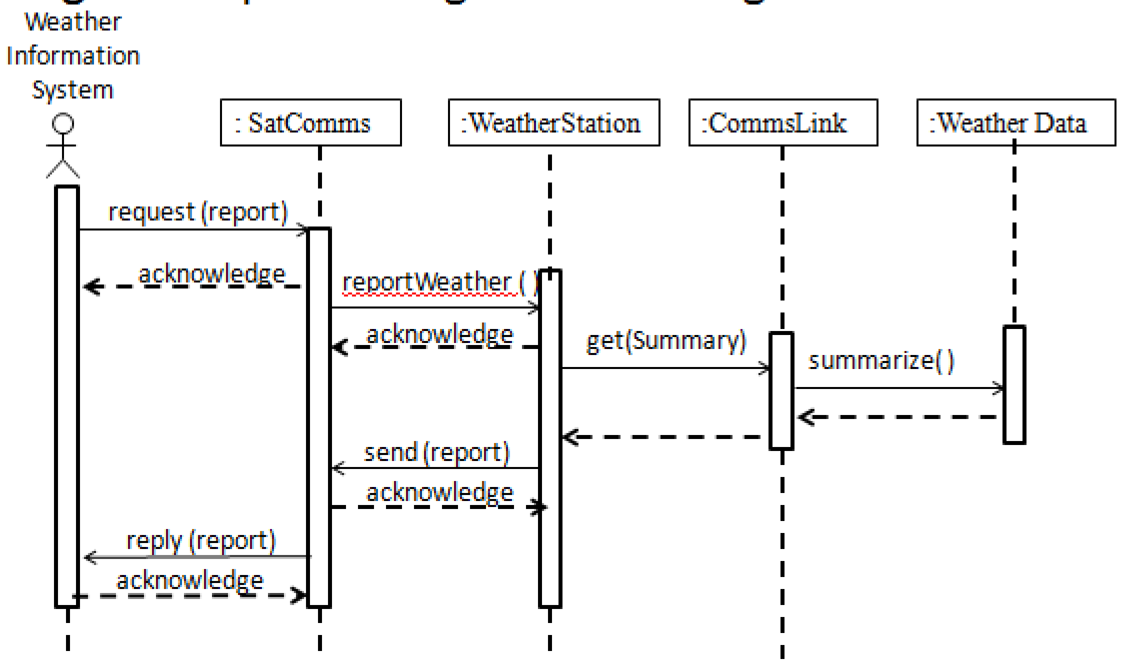
Design and system models show the objects or object classes in a system, the associations and relationships between those entities. The models are the link between system requirement and the implementation of a system. Though models have to be abstract, they should have enough details for programmers to take implementation decisions. An important step in the design process is to decide on the design models you need and their level of details.

When you use UML to develop a design, you will develop 2 types of design model.

(Static) Structural Models: showing static structure of the system, using object classes and their relationships.

Dynamic Models showing the dynamic structure of the system showing the interactions between the system objects.

Figure: Sequence diagram describing data collection



6.1.5. Interface specification

An important part of any design process is the specification of the interface between the components in the design. Interface design is concerned with specifying the detail of an interface to an object or to a group objects. This means defining the signatures and semantics of the services provided by the object or by a group of objects. In UML interfaces are specified using the same notation as a class diagram though there is no attribute section and the UML stereotype <<interface>> should be included in the name part. Operations to access or update data should be included. There is not a 1:1 relationship between objects and interfaces. One object may have several interfaces and a group of objects may be accessed through a single interface.

6.2. Implementation

Aspects of implementation that are particularly important to software engineering.

1. **Reuse:** modern software is constructed by reusing existing components or systems; you should make as much use as possible of existing code
2. **Configuration management:** while in the development process you should keep track of versions of components created to avoid errors of including wrong versions in the system.

3. **Host-target development:** development of software is done on one computer (the host system) and the execution on a separate computer (target system). Host and target systems may be the same or different.

6.2.1. Reuse

Reuse existing software saves time and money, and reduces development risks. But they are costs related to reuse:

1. Cost related to time spent looking for the suitable software to reuse.
2. Costs of buying the reusable software, where applicable. Large COTS system may be expensive.
3. Costs of adapting and configuring the reusable software component/systems to meet the requirement of the system you are developing.
4. The costs of integrating reusable software elements with each other and with your own code. They may be having conflicting assumptions.

6.2.2. Configuration management

Configuration management is the process of managing a changing software system. This makes available the up-to-date version of the code to developers and document in a controlled way.

There are 3 essential configuration management activities:

1. Version management: where support is provided to keep track of the different versions of software components.
2. System integration: here the support is provided to help developers define what versions of components are used to create each version of the system. The system is then built automatically by compiling and linking the required components
3. Problem tracking: here support allows users to report bugs and other problems, and allow users to know who is working on solving those problems and when they are solved.

6.2.3. Host target development

- Sometimes the development and execution platforms are the same and it is possible to develop the software and test it on the same machine. Other times when those platforms are different you either move the developed software to the execution platform for testing or run a simulation on the development machine.
- A software development platform should provide a range of **tools to support software engineering processes**. They include:

1. An integrated compiler and syntax-directed system that allows you to create, edit and compile code
2. A language debugging system
3. Graphical editing tools such as tools to edit UML models
4. Testing tools that can automatically run a set of tests on a new version of a program
5. Project support tools that help to organize the code for different development projects.

6.3. Open Source development

Open source development is an approach to software development in which the source code of a software system is published and volunteers are invited to participate in the development process (Raymond, 2001).

Example of open source products: Linux OS used as a server system and as a desktop environment, Java, and mySQL.

Advantages of open source products:

- Low cost; may need to pay for documentation and support only
- Mature open source systems are usually very reliable: bugs are discovered and repaired quickly.

Open source issues:

- Should the product being developed use open source components?
- Should an open source approach be used for the software development?

Answers will depend on type of software to develop and on the availability of open source products in the domain. Some companies use open source to develop systems targeting to sell their support; small companies use open source for customers maintenance assurance; while others don't want to reveal their confidential knowledge.

Open Source licensing

Though the principle of open source development is that source code should be freely available, it does not mean that anyone use that code anyhow. Legally, the developer of the code still owns it . They can place restriction on how it is used by including legally binding conditions in an open source software license (St Laurent, 2004).

Most of open source licenses are delivered from one of the 3 general model:

1. **The GNU General Public License (GPL)** also called ‘reciprocal’ license means that if you use open source software licensed under GPL license, then you must make that software open source.
2. **The GNU Lesser General Public License (LGPL):** here you can write components that link to open source code without having to publish the source of these components. But if you change the licensed component, then you must publish this as open source.
3. **The Berkley Standard Distribution (BSD) license:** is a non-reciprocal license. You are not obliged to publish changes made to open source code, you can use the code in systems to sell. If you use open source components, you must acknowledge the original creator of the code.

CHAPTER VIII. SOFTWARE TESTING AND SOFTWARE EVOLUTION

Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put in use. Testing is a part of broader process of software verification and validation. Verification and validation are no the same thing, although they are often confused.

Validation: are we building the right product?

Verification: are we building the product right?

The aim of verification is to check that the software meets its stated functional and non functional requirements. The aim of validation is to ensure that the software meets the customer’s expectations. It goes beyond simply checking conformance with specification to demonstrating that the software does what the customer expects it to do.

8.1. Development testing

Development testing include all testing activities that are carried out by the team developing the system. Testing may be carried out at 3 levels.

Unity testing: the process of testing program components, such as methods or object classes. Test are calls to routine with input parameters.

(Choosing unit test cases: testing is expensive; you should choose effective unit test cases to see if components do what is expected; if there are errors they are revealed.)

Component testing: this test shows if the component interface behaves according to its specification.

System testing: this checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.

8.2. Test-driven development

Test-driven development (TDD) is an approach where you develop the code incrementally, along with a test for that increment. You can only move to the next increment only if the previous increment passes its test.

Benefits of the test driven development are:

Code coverage: defects are discovered early in the development process.

Regression testing: regression tests are used to check that changes to the program have not introduced new bugs

Simplified debugging: when test fails, it is obvious that the problem lies in the newly written code.

System documentation: testing act as a documentation that describe what the code should be doing.

8.3. Release testing

This is the process of testing a particular release of a system that is intended for use outside of the development team.

Requirement based testing: this checks if the requirement has been satisfied. This validates proper implementation of the requirement

Scenario testing: this is a way of testing a scenario (a story that describes a way in which the system might be used)

Performance testing: once the system is complete, some properties like performance and reliability are tested. Performance testing involves running a series of tests where you can increase the load until the performance becomes unacceptable.

8.4. User/customer testing

Is the testing process where users or customers provide input and advice on system testing. It is an experiment to check if the system does what is needed.

There are 3 different types of user testing:

Alpha testing: users and developer test the software at the developer's site.

Beta testing: a release of software is given to user to experiment it and raise discovered problems with developers.

Acceptance testing: customers test if the system is acceptable and can be deployed in the customer’s environment.

8.5. SOFTWARE EVOLUTION

Software development does not stop when a system is delivered but it continues through the life time of the system. After deployment, if the system is to remain useful, it then has to change. Business changes and changes to user expectations generate new requirements for the existing software.

Software evolution is important because of the investments done by organizations in their software system; because organizations depend on these systems. Organizations need to have return on their investments.

Evolution processes

Software evolution processes vary depending on the type of software being maintained, the development processes used in an organization and the skills of the people involved.

Software maintenance

Software maintenance is the general process of changing a system after it has been delivered. The changes made to the software may be simple changes to correct coding errors, more extensive changes to correct specification errors.

PART II: SOFTWARE PROJECT MANAGEMENT

CHAPTER I: INTRODUCTION

Project and Project Management

A project is a [temporary] sequence of unique, complex, and connected activities having one goal or purpose and that must be completed by specific time, within budget, and according to specification.
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.

Process management is an ongoing activity that documents, manages the use of, and improves an organization's chosen methodology (the "process") for system development. Process management is concerned with the activities, deliverables, and quality standards to be applied to all projects.

I.1. Why do software projects fail?

- ✓ **People begin programming before they understand the problem**
  - o Everyone likes to feel that they're making progress
  - o When the team starts to code as soon as the project begins, they see immediate gains
  - o When problems become more complex (as they always do!), the work gets bogged down
  - o In the best case, a team that begins programming too soon will end up writing good software that solves the wrong problem
- ✓ **The team has an unrealistic idea about how much work is involved.**
- o Teams can commit to impossible deadlines by being overly optimistic and not thinking through the work
- ✓ **Defects are injected early but discovered late.**

    - o Projects can address the wrong needs
    - o Requirements can specify incorrect behavior
    - o Design, architecture and code can be technically flawed
    - o Test plans can miss functionality
- o The later these problems are found, the more likely they are to cause the project to fail
- ✓ **Programmers have poor habits – and they don't feel accountable for their work.**
    - o Programmers don't have good control of their source code
    - o Code written by one person is often difficult for another person to understand
    - o Programmers don't test their code, which makes diagnosing and fixing bugs more expensive
- ✓ **Managers try to test quality into the software.**
    - o Everyone assumes that the testers will catch all of the defects that were injected throughout the project.
    - o When testers look for defects, managers tell them they are wasting time.
    - o When testers find defects, programmers are antagonized because they feel that they are being personally criticized.
    - o When testers miss defects, everyone blames them for not being perfect.

I.2. Measures of Project Success

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

I.3. Who needs software?

Most software is built in organizations for people with specific needs.

- ▷ A *stakeholder* is a anyone who has an interest (or stake) in the software being completed
- ▷ A *user* is someone who will need to use the software to perform tasks.
- ▷ Sometimes stakeholders will be users; but often the stakeholder will not use the software.

Perspectives or Stakeholders

System owners pay for the system to be built and maintained.

System users use the system to perform or support the work to be completed.

System designers design the system to meet the users' requirements.

System builders construct, test, and deliver the system into operation.

Systems analysts facilitate the development of information systems and computer applications by bridging the communications gap that exists between nontechnical system owners and users and technical system designers and builders.

IT vendors and consultants sell hardware, software, and services to businesses for incorporation into their information systems.

I.4. Who builds software?

Software is typically built by a team of software engineers, which includes:

- ▷ *Business analysts* or *requirements analysts* who talk to users and stakeholders, plan the behavior of software and write software requirements
- ▷ *Designers and architects* who plan the technical solution
- ▷ *Programmers* who write the code
- ▷ *Testers* who verify that the software meets its requirements and behaves as expected

I.5. Project Manager

The project manager plans and guides the software project

- ▷ The project manager is responsible for identifying the users and stakeholders and determining their needs
- ▷ The project manager coordinates the team, ensuring that each task has an appropriate software engineer assigned and that each engineer has sufficient knowledge to perform it
- ▷ To do this well, the project manager must be familiar with every aspect of software engineering

Project Manager Competencies

- Business awareness
- Business partner orientation
- Commitment to quality
- Initiative
- Information gathering
- Analytical thinking
- Conceptual thinking
- Interpersonal awareness
- Organizational awareness
- Anticipation of impact
- Resourceful use of influence
- Motivating others
- Communication skills
- Developing others
- Monitoring and controlling
- Self-confidence
- Stress management
- Concern for credibility
- Flexibility

- ✓ **The project manager drives the scope of the project.**
  - ▷ The project manager should identify and talk to the main stakeholder
  - ▷ The effective way to show stakeholders that their needs are understood and that those specific needs will be addressed is with a *vision and scope document*

A typical vision and scope document follows an outline like this one:

Problem Statement

Project background

Stakeholders

Users

Risks

Assumptions

Vision of the Solution

Vision statement

List of features

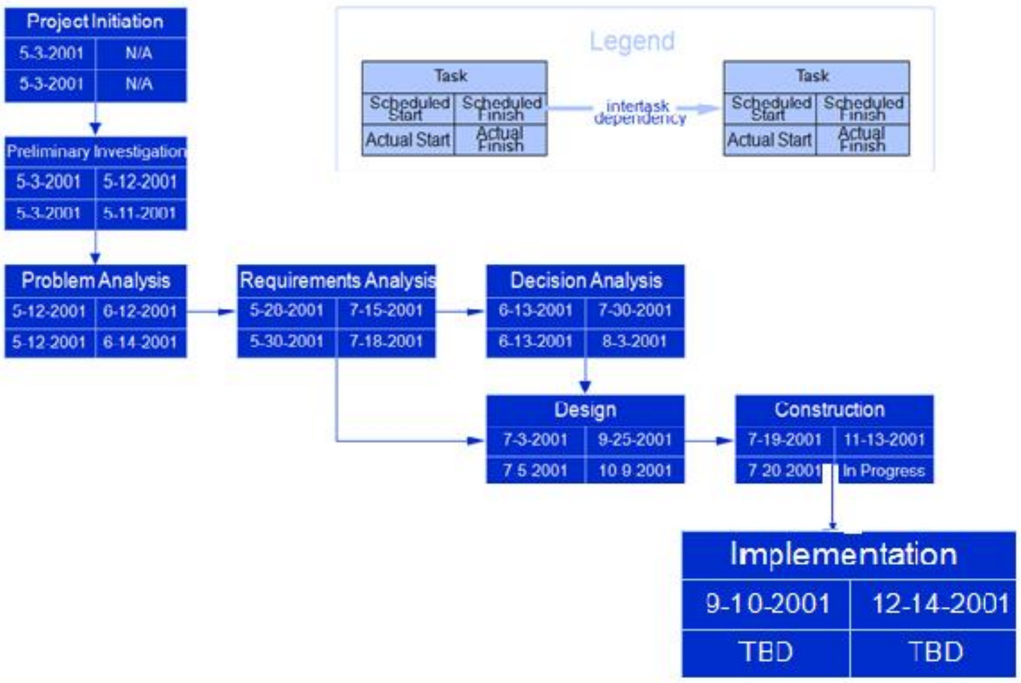
Scope of phased release *(optional)*

Features that will not be developed

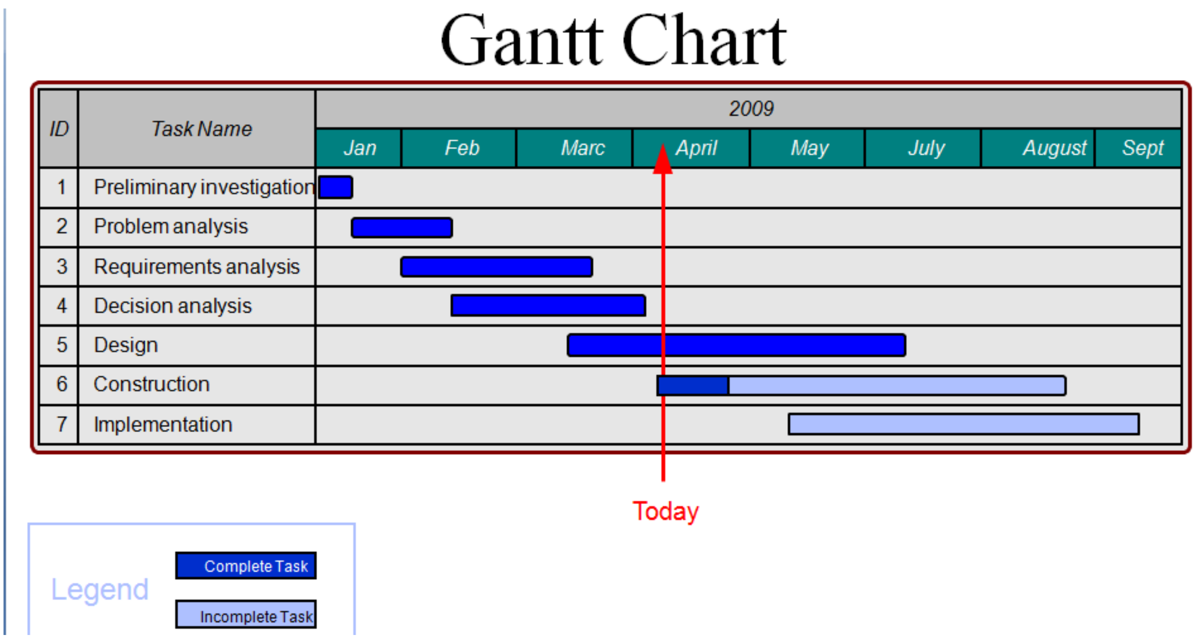
Joint project planning (JPP) is a strategy wherein all stakeholders in a project (meaning system owners, users, analysts, designers, and builders) participate in a one-to-three day project management workshop, the result of which is consensus agreement on project scope, schedule, resources, and budget. (Of course, subsequent workshops or meetings may be required to adjust scope, budget, and schedule.)

I.6. Project Management Tools & Techniques

A PERT chart is a graphical network model that depicts a project’s tasks and the relationships between those tasks. The *Program Evaluation and Review Technique* (PERT) is a network model that allows for randomness in activity completion times. PERT was developed in the late 1950's for the U.S. Navy's Polaris project having thousands of contractors. It has the potential to reduce both the time and cost required to complete a project.



A Gantt chart is a simple horizontal bar chart that depicts project tasks against a calendar. Each bar represents a named project task. The tasks are listed vertically in the left-hand column. The horizontal axis is a calendar timeline.



CHAPTER II. RISK MANAGEMENT

2.1. What is risk?

- It is the potential for realizing unwanted negative consequences of an event.
- A risk is a probability that some adverse circumstance will occur .
  - Not inherently bad, Essential to progress!.
  - The challenge is to manage the amount of risk.
- Risk is a function of: Threats, vulnerabilities, and impact

Assets, Threats, Vulnerabilities and impacts

- Assets: Anything of value such as people, information, hardware, software, facilities, reputation, activities, and operations
- Threat: An indication of a potential undesirable event.
- Examples: Hackers, Insiders, ..etc
- Vulnerability: Any weakness (in a system, control, or countermeasure) that can be exploited by an adversary.
- Examples: Networks, operating Systems, Application, People, or Processes.
- An impact: Refers to the degree of likely harm incurred should a vulnerability be exploited such as strategic, financial, or legal consequences.

...the risks will actively attack you –*Tom Gilb*

2.2. Risk management

- ➢ Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.
- ➢ Systematic and analytical process by which an organization identifies, reduces, and controls its potential risks and losses.
- ➢ Project risks affect schedule or resources;
- ➢ Product risks affect the quality or performance of the software being developed;
- ➢ Business risks affect the organisation developing or procuring the software.
- ➢ General idea:
    - o Identify your project's risks
    - o Assess their impact and likelihood
    - o Prepare plans to mitigate or avert them
    - o Monitor the risks and their corresponding plans

The risk management process

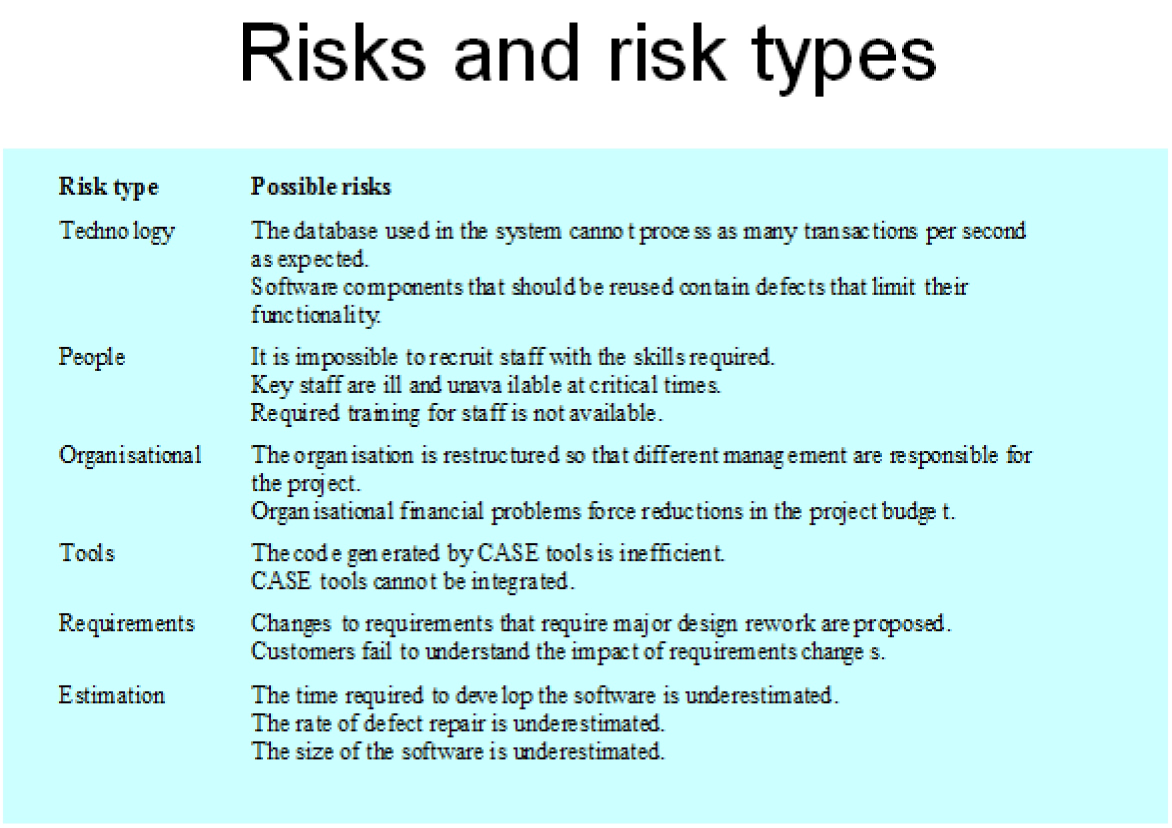
- Risk identification
    - – Identify project, product and business risks;
- Risk analysis
    - – Assess the likelihood and consequences of these risks;
- Risk planning
    - – Draw up plans to avoid or minimise the effects of the risk;
- Risk monitoring
    - – Monitor the risks throughout the project;

Risk identification

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

RISK ANALYSIS

During the risk analysis process, you have to consider each identified risk and make a judgement about the probability and seriousness of that risk. You have to rely on your own judgement and experience of previous projects and problems that arose in them.



Risks and risk types

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 unavailable 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 changes.
Estimation	The time required to develop the software is underestimated. The rate of defect repair is underestimated. The size of the software is underestimated.

RISK PLANNING

The risk planning process considers each of the key risks that have been identified and develops strategies to manage these risks. For each of the risks, you have to think of actions that might take to minimize the disruption to the project if the problem identified in the risk occurs.

RISK MONITORING

Risk monitoring is the process that assumptions about the product, process, and business risks have not changed. You should regularly assess each of the identified risks to decide whether or not that risk is becoming more or less probable. You should monitor risks regularly at all stages in a project.

CHAPTER III. PROJECT PLAN

The *project plan* defines the work that will be done on the project and who will do it. It consists of:

- ▷ A statement of work (SOW) that describes all work products that will be produced and a list of people who will perform that work
- ▷ A resource list that contains a list of all resources that will be needed for the product and their availability
- ▷ A work breakdown structure and a set of estimates
- ▷ A project schedule
- ▷ A risk plan that identifies any risks that might be encountered and indicates how those risks would be handled should they occur.

Project Management Functions

- **Scoping**
- **Planning**
- **Estimating**
- **Scheduling**
- **Organizing**
- **Directing**
- **Controlling**
- **Closing**

Activity 1: Negotiate Scope

Scope defines the boundaries of a project—What part of the business is to be studied, analyzed, designed, constructed, implemented, and ultimately improved?

- Product
- Quality
- Time
- Cost
- Resources

A statement of work is a narrative description of the work to be performed as part of a project. Common synonyms include *scope statement*, *project definition*, *project overview*, and *document of understanding*.

Activity 2: Identify Tasks

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

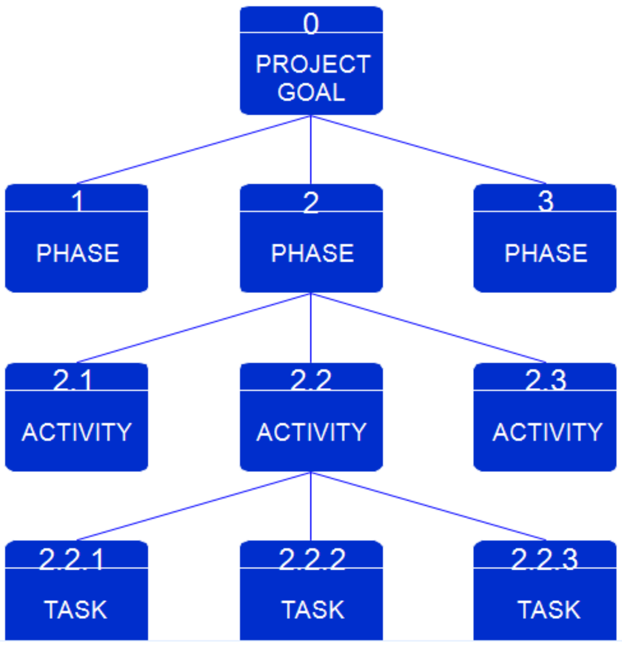
Milestones are events that signify the accomplishment or completion of major deliverables during a project.

Work Breakdown Structures

1 Phase 1 of the project …
2 Phase 2 of the project …
- **2.1** Activity 1 of Phase 2 …
- **2.2** Activity 2 of Phase 2
  - **2.2.1** Task 1 of Activity 2.2 in Phase 2
  - **2.2.2** Task 2 of Activity 2.2 in Phase 2
  - **2.2.3** Task 3 of Activity 2.2 in Phase 2
- **2.3** Activity 3 of Phase 2 …

3 Phase 3 of the project …

=



Activity 3: Estimation

What is estimation?

The project manager must set expectations about the time required to complete the software among the stakeholders, the team, and the organization's management.

Elements of a Sound Estimate

To generate a sound estimate, a project manager must have:

- ▷ A work breakdown structure (WBS), or a list of tasks which, if completed, will produce the final product
- ▷ An effort estimate for each task
- ▷ A list of assumptions which were necessary for making the estimate
- ▷ Consensus among the project team that the estimate is accurate

Assumptions Make Estimates More Accurate

Team members make *assumptions* about the work to be done in order to deal with incomplete information

- ▷ Any time an estimate must be based on a decision that has not yet been made, team members can assume the answer for the sake of the estimate
- ▷ Assumptions must be written down so that if they prove to be incorrect and cause the estimate to be inaccurate, everyone understands what happened
- ▷ Assumptions bring the team together very early on in the project so they can make progress on important decisions that will affect development

Wideband Delphi

Wideband Delphi is a process that a team can use to generate an estimate

- ▷ The project manager chooses an estimation team, and gains consensus among that team on the results
- ▷ Wideband Delphi is a *repeatable* estimation process because it consists of a straightforward set of steps that can be performed the same way each time

The Wideband Delphi Process

Step 1: Choose the team

- ▷ The project manager selects the estimation team and a moderator. The team should consist of 3 to 7 project team members.
  - The moderator should be familiar with the Delphi process, but should not have a stake in the outcome of the session if possible.
  - If possible, the project manager should not be the moderator because he should ideally be part of the estimation team.

Step 2: Kickoff Meeting

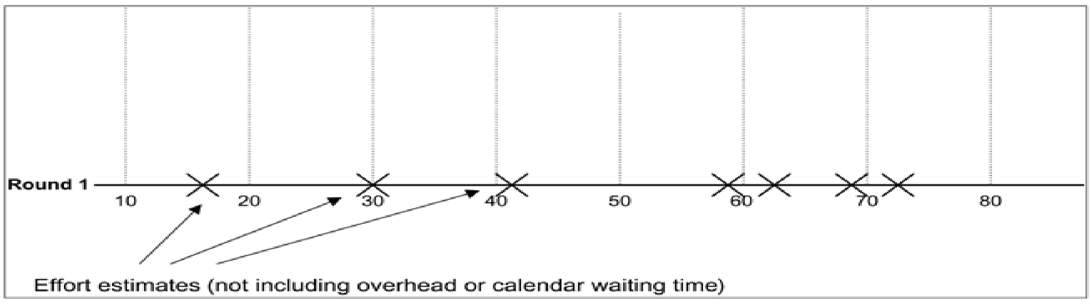
- ▷ The project manager must make sure that each team member understands the Delphi process, has read the vision and scope document and any other documentation, and is familiar with the project background and needs.
- ▷ The team brainstorms and writes down assumptions.
- ▷ The team generates a WBS with 10-20 tasks.
- ▷ The team agrees on a unit of estimation.

Step 3: Individual Preparation

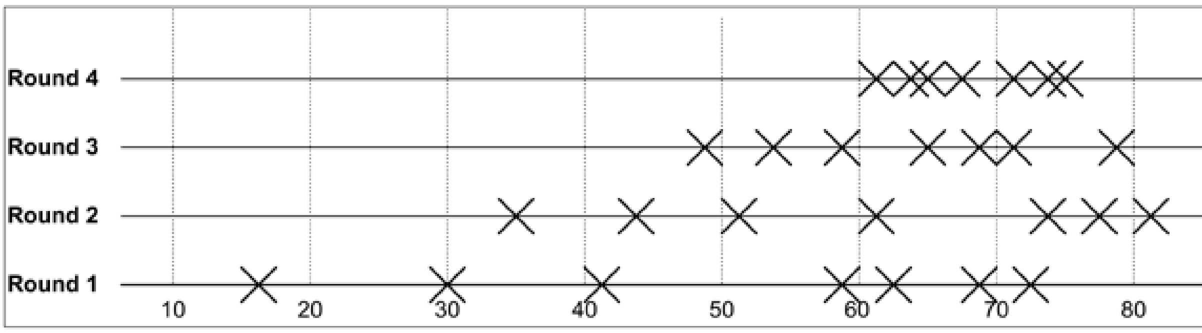
- ▷ Each team member independently generates a set of preparation results.
- ▷ For each task, the team member writes down an estimate for the effort required to complete the task, and any additional assumptions he needed to make in order to generate the estimate.

Step 4: Estimation Session

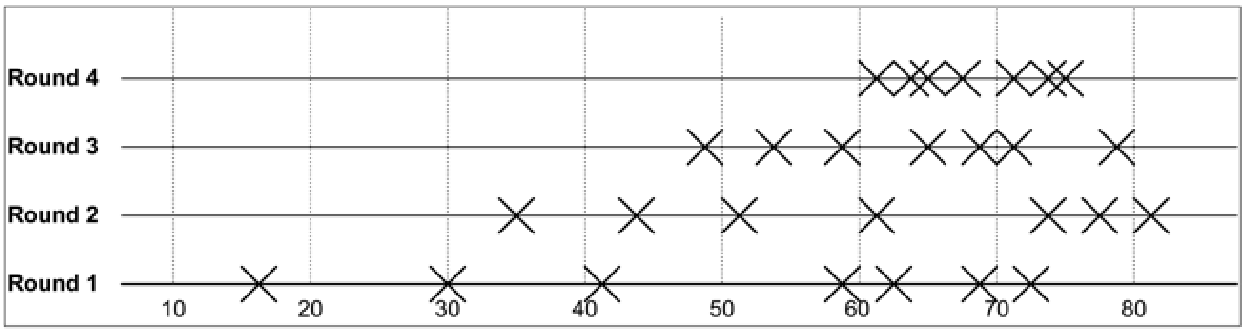
- ▷ During the estimation session, the team comes to a consensus on the effort required for each task in the WBS.
- ▷ Each team member fills out an estimation form which contains his estimates.
- ▷ The rest of the estimation session is divided into rounds during which each estimation team member revises her estimates based on a group discussion. Individual numbers are not dicsussed.
- ▷ The moderator collects the estimation forms and plots the sum of the effort from each form on a line:



- ▷ The team resolves any issues or disagreements that are brought up.
- ▷ Individual estimate times are not discussed. These disagreements are usually about the tasks themselves. Disagreements are often resolved by adding assumptions.
- ▷ The estimators all revise their individual estimates. The moderator updates the plot with the new total:



▷ The team resolves any issues or disagreements that are brought up. Individual estimate times are not discussed. These disagreements are usually about the tasks themselves. Disagreements are often resolved by adding assumptions. The estimators all revise their individual estimates. The moderator updates the plot with the new total:



▷ The moderator leads the team through several rounds of estimates to gain consensus on the estimates. The estimation session continues until the estimates converge or the team is unwilling to revise estimates.

Step 5: Assemble Tasks

▷ The project manager works with the team to collect the estimates from the team members at the end of the meeting and compiles the final task list, estimates and assumptions.

Step 6: Review Results

▷ The project manager reviews the final task list with the estimation team.

Other Estimation Techniques

PROBE, or Proxy Based Estimating

- ▷ PROBE is based on the idea that if an engineer is building a component similar to one he built previously, then it will take about the same effort as it did in the past.
- ▷ Individual engineers use a database to maintain a history of the effort they have put into their past projects.
- ▷ A formula based on linear regression is used to calculate the estimate for each task from this history.

COCOMO II

- ▷ In Constructive Cost Model, or COCOMO, projects are summarized using a set of variables that must be provided as input for a model that is based on the results of a large number of projects across the industry.
- ▷ The output of the model is a set of size and effort estimates that can be developed into a project schedule.

The Planning Game

- ▷ The Planning Game is the software project planning method from Extreme Programming (XP), a lightweight development methodology developed by Kent Beck in the 1990s at Chrysler.
- ▷ It is a full planning process that combines estimation with identifying the scope of the project and the tasks required to complete the software.
- ▷ The Planning Game is highly iterative. The scope is established by having Development and Business work together to interactively write “user stories” written on index cards to describe the scope. Each story is given an estimate of 1, 2 or 3 weeks. This process is repeated continuously throughout the project.

Estimate Task Durations

1. Estimate the minimum amount of time it would take to perform the task. We'll call this the optimistic duration (OD).

2. Estimate the maximum amount of time it would take to perform the task. We'll call this the pessimistic duration (PD).

3. Estimate the expected duration (ED) that will be needed to perform the task.

4. Calculate the most likely duration (D) as follows:

$$D = \frac{(1 \text{ x OD}) + (4 \text{ x ED}) + (1 \text{ x PD})}{6}$$

Activity 4: Project Schedules

What is a project schedule?

The *project schedule* is a calendar that links the tasks to be done with the resources that will do them.

- ▷ Before a project schedule can be created, the project manager must have a work breakdown structure (WBS) and estimates.
- ▷ The schedule is part of the project plan.

Scheduling concepts:
Effort vs. Duration

Effort represents the work required to perform a task.

- ▷ Effort is measured in person-hours (or person-days, person-weeks, etc.)
- ▷ It represents the total number of hours that each person spent working on the task.

Duration is amount of time that elapses between the time the task is started and the time it is completed.

- ▷ Duration is measured in hours (or days, weeks, etc.)
- ▷ It does not take into account the number of people performing the task.

Building the project schedule

Allocate resources

- ▷ For each task in the WBS, one or more resources must be assigned
- ▷ Choose person or people for each task based on qualifications, familiarity and availability
- ▷ Take overhead into account when calculating the duration of each task

Identify dependencies

- ▷ A task has a dependency if it involves an activity, resource or work product which is subsequently required by another task
- ▷ Tasks may have dependencies because they require the same resource
- ▷ Every dependency has a *predecessor*, or a task that must be begun, in progress, or completed, for another task to begin
- ▷ Identify the type of predecessor for each dependency
- ▷ Create the schedule
- ▷ Most project schedules are represented using a Gantt chart
- ▷ The Gantt chart shows tasks, dependencies and milestones using different shapes

Add review meetings to the schedule

- ▷ Progress reviews are meetings held regularly to check the progress of a project versus it's scheduled progress.
- ▷ Milestone reviews are meetings which the project manager schedules in advance to coincide with project events.
- ▷ The most common way for project managers to handle milestone reviews is to schedule them to occur after the last task in a project phase (such as the end of design or programming).

Step 4: Optimize the schedule

- ▷ The *critical path* is the sequence of tasks that represent the minimum time required to complete the project.

- ✓ If a task is only on the critical path when delaying that task will delay the project.
- ✓ Allocating resources to tasks on the critical path will reduce the project schedule; allocating them to other tasks will have less effect.

Critical Path Analysis (and Slack Time)

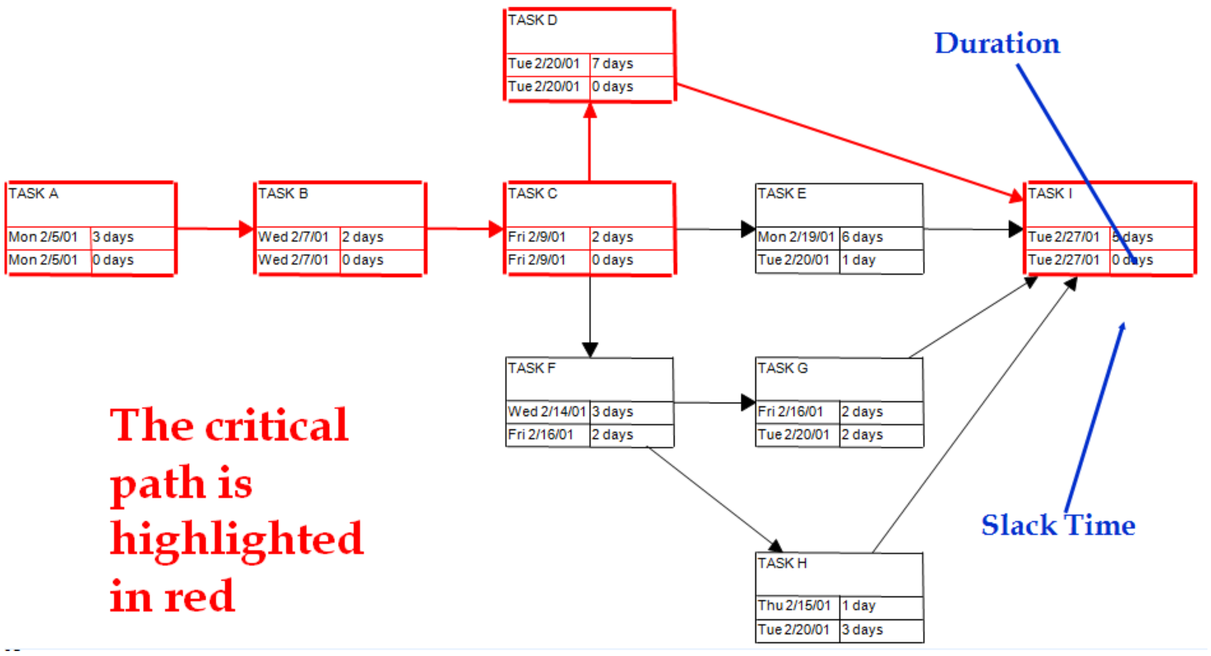
The critical path for a project is that sequence of dependent tasks that have the largest sum of most likely durations. The critical path determines the earliest possible completion date of the project.

- – Tasks that are on the critical path cannot be delayed without delaying the entire project schedule. To achieve resource leveling, critical tasks can only be split.

The slack time available for any noncritical task is the amount of delay that can be tolerated between the starting time and completion time of a task without causing a delay in the completion date of the entire project.

- – Tasks that have slack time can be delayed to achieve resource leveling

1. Using intertask dependencies, determine every possible path through the project.
2. For each path, sum the durations of all tasks in the path.
3. The path with the longest total duration is the critical path.
   - – The critical path for a project is that sequence of dependent tasks that have the largest sum of *most likely durations*. The critical path determines the earliest completion date of the project.
   - – The slack time available for any noncritical task is the amount of delay that can be tolerated between the starting time and completion time of a task without causing a delay in the completion date of the entire project.



Scheduling Strategies

Forward scheduling establishes a project start date and then schedules forward from that date. Based on the planned duration of required tasks, their interdependencies, and the allocation of resources to complete those tasks, a projected project completion date is calculated.

Reverse scheduling establishes a project deadline and then schedules backward from that date. Essentially, tasks, their duration, interdependencies, and resources must be considered to ensure that the project can be completed by the deadline.

Activity 5: Assign Resources

A *resource* is a person, hardware, room or anything else that is necessary for the project but limited in its availability

The resource list should give each resource a name, a brief one-line description, and list the availability and cost (if applicable) of the resource

People—inclusive of all the system owners, users, analysts, designers, builders, external agents, and clerical help that will be involved in the project in any way, shape, or form.

Services—a service such as a quality review that may be charged on a per use basis.

Facilities and equipment—including all rooms and technology that will be needed to complete the project.

Supplies and materials— everything from pencils, paper, notebooks, toner cartridges, etc.

Money—A translation of all of the above into the language of accounting—budgeted dollars!

Resource Leveling

Resource leveling is a strategy used to correct resource overallocations by some combination of *delaying* or *splitting tasks*.

There are two techniques for resource leveling:

- ***task delaying***
- ***task splitting***

Activity 6: Direct the Team Effort

Supervision resources

- The DEADLINE – A Novel About Project Management
- The One Minute Manager
- The Care and Feeding of Monkeys

Stages of Team Maturity

Activity 7: Monitor and Control Progress

- Progress reporting
- Change management
- Expectations management
- Schedule adjustments—critical path analysis (CPA)

CHAPTER III. REVIEWS

When are reviews needed?

A *review* is any activity in which a work product is distributed to reviewers who examine it and give feedback.

- ▷ Reviews are useful not only for finding and eliminating defects, but also for gaining consensus among the project team, securing approval from stakeholders, and aiding in professional development for team members.
- ▷ Reviews help teams find defects soon after they are injected making them cost less to fix than they would cost if they were found in test.
- ▷ All work products in a software project should be either reviewed or tested.
    - Software requirements specifications, schedules, design documents, code, test plans, test cases, and defect reports should all be reviewed.

Types of Review:
There are many types of reviews such as follows: Inspections, Deskchecks, Walkthroughs, Code Review and *Pair programming.*

Inspections

Inspections are moderated meetings in which reviewers list all issues and defects they have found in the document and log them so that they can be addressed by the author.

The goal of the inspection is to repair all of the defects so that everyone on the inspection team can approve the work product.

- ▷ Commonly inspected work products include software requirements specifications and test plans.

- ■ Running an inspection meeting:
    - ▷ A work product is selected for review and a team is gathered for an inspection meeting to review the work product.
    - ▷ A moderator is chosen to moderate the meeting.
    - ▷ Each inspector prepares for the meeting by reading the work product and noting each defect.
    - ▷ In an inspection, a defect is any part of the work product that will keep an inspector from approving it.
    - ▷ Discussion is focused on each defect, and coming up with a specific resolution.
        - ■ It's the job of the inspection team to do more than just identify the problems; they must also come up with the solutions.
    - ▷ The moderator compiles all of the defect resolutions into an *inspection log*

Deskchecks

- **A *deskcheck*** is a simple review in which the author of a work product distributes it to one or more reviewers.
  - The author sends a copy of the work product to selected project team members. The team members read it, and then write up defects and comments to send back to the author.

- Unlike an inspection, a deskcheck does not produce written logs which can be archived with the document for later reference.

- Deskchecks can be used as predecessors to inspections.
  - In many cases, having an author of a work product pass his work to a peer for an informal review will significantly reduce the amount of effort involved in the inspection.

Walkthroughs

- A *walkthrough* is an informal way of presenting a technical document in a meeting.
  - Unlike other kinds of reviews, the author runs the walkthrough: calling the meeting, inviting the reviewers, soliciting comments and ensuring that everyone present understands the work product.
  - Walkthroughs are used when the author of a work product needs to take into account the perspective of someone who does not have the technical expertise to review the document.
  - After the meeting, the author should follow up with individual attendees who may have had additional information or insights. The document should then be corrected to reflect any issues that were raised.

Code Review

- A *code review* is a special kind of inspection in which the team examines a sample of code and fixes any defects in it.
  - In a code review, a defect is a block of code which does not properly implement its requirements, which does not function as the programmer intended, or which is not incorrect but could be improved
    - For example, it could be made more readable or its performance could be improved
  - It’s important to review the code which is most likely to have defects. This will generally be the most complex, tricky or involved code.
  - Good candidates for code review include:
    - A portion of the software that only one person has the expertise to maintain
    - Code that implements a highly abstract or tricky algorithm
    - An object, library or API that is particularly difficult to work with

- Code written by someone who is inexperienced or has not written that kind of code before, or written in an unfamiliar language
- Code which employs a new programming technique
- An area of the code that will be especially catastrophic if there are defects

Pair Programming

- ***Pair programming*** is a technique in which two programmers work simultaneously at a single computer and continuously review each others' work.

- Although many programmers were introduced to pair programming as a part of Extreme Programming, it is a practice that can be valuable in any development environment.

- **Pair programming** improves the organization by ensuring that at least two programmers are able to maintain any piece of the software.

- In pair programming, two programmers sit at one computer to write code. Generally, one programmer will take control and write code, while the other watches and advises.
  - Some teams have found that pair programming works best for them if the pairs are constantly rotated; this helps diffuse the shared knowledge throughout the organization. Others prefer to pair a more junior person with a more senior for knowledge sharing.

- The project manager should not try to force pair programming on the team; it helps to introduce the change slowly, and where it will meet the least resistance.
  - It is difficult to implement pair programming in an organization where the programmers do not share the same nine-to-five (or ten-to-six) work schedule.
  - Some people do not work well in pairs, and some pairs do not work well together.

CHAPTER IV. FEASIBILITY ANALYSIS AND THE SYSTEM PROPOSAL

Feasibility is the measure of how beneficial or practical the development of an information system will be to an organization.

Feasibility analysis is the process by which feasibility is measured.

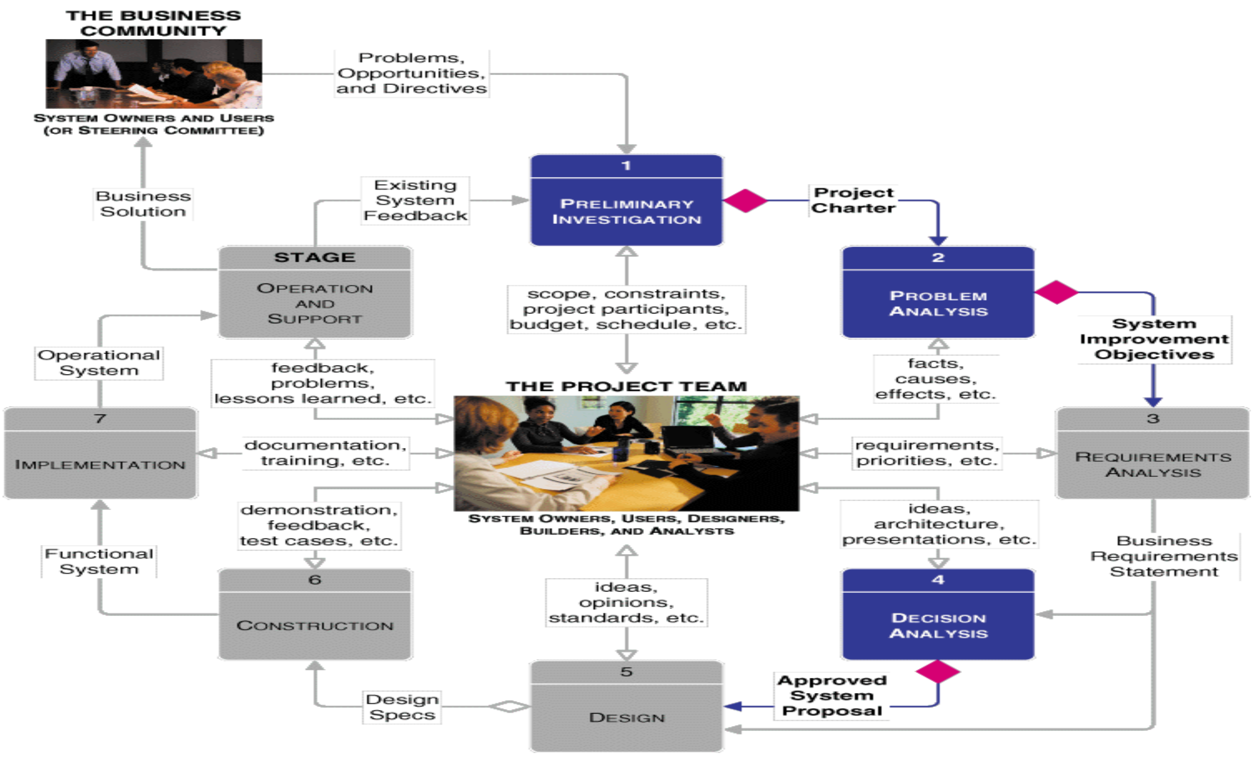
Creeping Commitment approach to feasibility proposes that feasibility should be measured throughout the life cycle.

Feasibility Checkpoints

Systems Analysis — Preliminary Investigation

Systems Analysis — Problem Analysis

Systems Design — Decision Analysis


THE BUSINESS COMMUNITY
SYSTEM OWNERS AND USERS (OR STEERING COMMITTEE)
Problems, Opportunities, and Directives
1 PRELIMINARY INVESTIGATION
Project Charter
2 PROBLEM ANALYSIS
System Improvement Objectives
3 REQUIREMENTS ANALYSIS
Business Requirements Statement
4 DECISION ANALYSIS
Approved System Proposal
5 DESIGN
Design Specs
6 CONSTRUCTION
Functional System
7 IMPLEMENTATION
Operational System
STAGE OPERATION AND SUPPORT
Business Solution
Existing System Feedback
scope, constraints, project participants, budget, schedule, etc.
facts, causes, effects, etc.
requirements, priorities, etc.
ideas, architecture, presentations, etc.
ideas, opinions, standards, etc.
demonstration, feedback, test cases, etc.
documentation, training, etc.
feedback, problems, lessons learned, etc.
THE PROJECT TEAM
SYSTEM OWNERS, USERS, DESIGNERS, BUILDERS, AND ANALYSTS

Four Tests For Feasibility

- **Operational feasibility** is a measure of how well the solution will work in the organization. It is also a measure of how people feel about the system/project.
- **Technical feasibility** is a measure of the practicality of a specific technical solution and the availability of technical resources and expertise.
- **Schedule feasibility** is a measure of how reasonable the project timetable is.
- **Economic feasibility** is a measure of the cost-effectiveness of a project or solution.

Three Popular Techniques to Assess Economic Feasibility

- Payback Analysis
- Return On Investment
- Net Present Value

The **Time Value of Money** is a concept that should be applied to each technique. The time value of money recognizes that a dollar today is worth more than a dollar one year from now.

Payback analysis is a simple and popular method for determining if and when an investment will pay for itself.
Payback period is the period of time that will lapse before accrued benefits overtake accrued and continuing costs.

Present Value Formula

$$\mathbf{PV}_{n} = \mathbf{1/(1 + }\boldsymbol{i}\mathbf{)}^{n}$$

Where *n* is the number of years and *i* is the discount rate.

Return-on-Investment Analysis (ROI)

Return-on-Investment compares the lifetime profitability of alternative solutions or projects.

The ROI for a solution or project is a percentage rate that measures the relationship between the amount the business gets back from an investment and the amount invested.