Chapter 3

SOFTWARE REQUIREMENT AND SCHEDULING

REQUIREMENTS ENGINEERING

- The broad spectrum of tasks and techniques that lead to an understanding of requirements is called requirements engineering.
- From a software process perspective, requirements engineering is a major software engineering action that begins during the communication activity and continues into the modelling activity.
- Requirements engineering HELPS for understanding what the customer wants, analysing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system.

It encompasses distinct tasks

Inception.

- But in general, most projects begin when a business need is identified or a potential new market or service is discovered.
- Stakeholders from the business community define a business case for the idea, try to identify the breadth and depth of the market,
- All of this information is subject to change, but it is sufficient to precipitate discussions with the software engineering organization
- It establishes a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired, and the effectiveness of preliminary communication and collaboration between the other stakeholders and the software team.

Elicitation.

- It tells the objectives for the system or product are, what is to be accomplished,
- It shows how the system or product fits into the needs of the business, and finally, how the system or product is to be used on a day-to-day basis.
- problems that are encountered as elicitation occurs.
  - Problems of scope.
  - Problems of understanding.
  - Problems of volatility. (The requirements change over time)

Elaboration.

- The information obtained from the customer during inception and elicitation is expanded and refined during elaboration.

- It identifies various aspects of software function, behaviour, and information.
- Elaboration is driven by the creation and refinement of user scenarios
- It describes how the end user (and other actors) will interact with the system.

Negotiation.

- It isn't unusual for customers and users to ask for more than can be achieved, given limited business resources.
- Its need to reconcile these conflicts through a process of negotiation.
- Customers, users, and other stakeholders are asked to rank requirements and then discuss conflicts in priority.
- Using an iterative approach that prioritizes requirements, assesses their cost and risk, and addresses internal conflicts, requirements are eliminated,

Specification.

- A specification can be a written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype, or any combination of these.
- a "standard template" should be developed and used for a specification,
- However, it is sometimes necessary to remain flexible when a specification is to be developed.
- For large systems, a written document, combining natural language descriptions and graphical models may be the best approach

Validation.

- The work products produced as a requirement engineering are assessed for quality during a validation step.
- Requirements validation examines the specification to ensure that all software requirements have been stated unambiguously; that inconsistencies, omissions, and errors have been detected and corrected;
- The primary requirements validation mechanism is the technical review
- The review team that validates requirements examine the specification looking for errors in content or interpretation,

Requirements management.

- Requirements for computer-based systems change, and the desire to change requirements persists throughout the life of the system.
- Requirements management is a set of activities that help the project team identify, control, and track requirements and changes to requirements at any time as the project proceeds.
- Many of these activities are identical to the software configuration management (SCM) techniques.

SRS (SOFTWARE REQUIREMENTS SPECIFICATION)

- Is a document which is used as a communication medium between the customer and the supplier.
- When the software requirement specification is completed and is accepted by all parties, the end of the requirements engineering phase has been reached.
- After the acceptance phase, any of the requirements cannot be changed, but the changes must be tightly controlled.
- The software requirement specification should be edited by both the customer and the supplier,

NEED OF SRS

- company publishing a software requirement specification to companies for competitive tendering,
- company writing their own software requirement specification in response to a user requirement document.
- SRS allow s number of different suppliers to propose solutions,
- Use to identify any constraints which must be applied.
- SRS is used to capture the user's requirements and
- SRS highlights inconsistencies and conflicting requirements and define system and acceptance testing activities.

CHARACTERISTICS OF A GOOD SRS

Complete

- Description of all major requirements relating to functionality, performance, design constraints and external interfaces.
- Definition of the response of the software system to all reasonable situations.
- Conformity to any software standards, detailing any sections which are not appropriate.
- Have full labelling and references of all tables and references, definitions of all terms and units of measure.
- Be fully defined, if there are sections in the software requirements specification still to be defined, the software

Consistent

types of confliction:

- Multiple descriptors - This is where two or more words are used to reference the same item, i.e. where the term cue and prompt are used interchangeably.
- Opposing physical requirements - This is where the description of real-world objects clash, e.g. one requirement states that the warning indicator is orange, and another states that the indicator is red.
- Opposing functional requirements - This is where functional characteristics conflict, e.g. perform function X after both A and B has occurred, or perform function X after A or B has occurred

Traceable

A software requirement specification is traceable if both the origins and the references of the requirements are available. Traceability of the origin or a requirement can help understand who asked for the requirement and also what modifications have been made to the requirement to bring the requirement to its current state. Traceability of references are used to aid the modification of future documents by stating where a requirement has been referenced. By having foreword traceability, consistency can be more easily contained

Unambiguous

As the Oxford English dictionary states the word unambiguous means "not having two or more possible meanings". This means that each requirement can have one and only one interpretation. If it is unavoidable to use an ambiguous term in the requirements specification, then there should be clarification text describing the context of the term. One way of removing ambiguity is to use a formal requirements specification language. The advantage to using a formal language is the relative ease of detecting errors by using lexical syntactic analysers to detect ambiguity. The disadvantage of using a formal requirements specification language is the learning time and loss of understanding of the system by the client.

Verifiable

A software requirement specification is verifiable if all of the requirements contained within the specification are verifiable. A requirement is verifiable if there exists a finite cost-effective method by which a person or machine can check that the software product meets the requirement. Non-verifiable requirements include "The system should have a good user interface" or "the software must work well under most conditions" because the performance words of good, well and most are subjective and open to interpretation. If a method cannot be devised to determine whether the software meets a requirement, then the requirement should be removed or revised

COMPONENTS OF THE SRS

Given below are the system properties that an SRS should specify. The basic issues, an SRS must address are:

1. Functional requirements

2. Non-functional requirements
3. Performance requirements
4. Design constraints
5. External interface requirements

Conceptually, any SRS should have these components. Now we will discuss them one by one.

1. Functional Requirements

- Functional requirements specify what output should be produced from the given inputs. So, they basically describe the connectivity between the input and output of the system. For each functional requirement:
- A detailed description of all the data inputs and their sources, the units of measure, and the range of valid inputs be specified:
- All the operations to be performed on the input data obtain the output should be specified, and
- Care must be taken not to specify any algorithms that are not parts of the system but that may be needed to implement the system.
- It must clearly state what the system should do if system behaves abnormally when any invalid input is given or due to some error during computation. Specifically, it should specify the behaviour of the system for invalid inputs and invalid outputs.

2.Non-functional Requirements

Characteristics of the system which cannot be expressed as functions:

- Maintainability,
- Portability,
- Usability,
- Security,
- Safety, etc.

3. Performance Requirements (Speed Requirements)

This part of an SRS specifies the performance constraints on the software system. All the requirements related to the performance characteristics of the system must be clearly specified. Performance requirements are typically expressed as processed transaction per second or response time from the system for a user event or screen refresh time or a combination of these. It is a good idea to pin down performance requirements for the most used or critical transactions, user events and screens.

4. Design Constraints

The client environment may restrict the designer to include some design constraints that must be followed. The various design constraints are standard compliance, resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

Standard Compliance: It specifies the requirements for the standard the system must follow. The standards may include the report format and according procedures.

Hardware Limitations: The software needs some existing or predetermined hardware to operate, thus imposing restrictions on the design. Hardware limitations can include the types of machines to be used operating system availability memory space etc.

Fault Tolerance: Fault tolerance requirements can place a major constraint on how the system is to be designed. Fault tolerance requirements often make the system more complex and expensive, so they should be minimized.

Security: Currently security requirements have become essential and major for all types of systems. Security requirements place restriction s on the use of certain commands control access to database, provide different kinds of access, requirements for different people, require the use of passwords and cryptography techniques, and maintain a log of activities in the system.

5. External Interface Requirements

For each external interface requirements:

- All the possible interactions of the software with people hardware and other software should be clearly specified,
- The characteristics of each user interface of the software product should be specified and
- The SRS should specify the logical characteristics of each interface between the software product and the hardware components for hardware interfacing.

STRUCTURE OF SRS

A software requirements specification is a document which is used as a communication medium between the customer and the supplier. When the software requirement specification is completed and is accepted by all parties, the end of the requirements engineering phase has been reached.
The requirements document is devised in a manner that is easier to write, review, and maintain. It is organized into independent sections and each section is organized into modules or units. Note that the level of detail to be included in the SRS depends on the type of the system to be developed and the process model chosen for its development.
Document comprises the following sections.

1. Introduction: This provides an overview of the entire information described in SRS. This involves purpose and the scope of SRS, which states the functions to be performed by the system. In addition, it describes definitions, abbreviations, and the acronyms used. The references used in SRS provide a list of documents that is referenced in the document.

2. Overall description: It determines the factors which affect the requirements of the system. It provides a brief description of the requirements to be defined in the next section called 'specific requirement'. It comprises the following sub-sections.

3. Product perspective: It determines whether the product is an independent product or an integral part of the larger product. It determines the interface with hardware, software, system, and communication. It also defines memory constraints and operations utilized by the user.

4. Product functions: It provides a summary of the functions to be performed by the software. The functions are organized in a list so that they are easily understandable by the user:

5. User characteristics: It determines general characteristics of the users.

6. Constraints: It provides the genera1 description of the constraints such as regulatory policies, audit functions, reliability requirements, and so on.

7. Assumption and dependency: It provides a list of assumptions and factors that affect the requirements as stated in this document.

8. Apportioning of requirements: It determines the requirements that can be delayed until release of future versions of the system.

9. Specific requirements: These determine all requirements in detail so that the designers can design the system in accordance with them. The requirements include description of every input and output of the system and functions performed in response to the input provided.

10. External interface: It determines the interface of the software with other systems, which can include interface with operating system and so on. External interface also specifies the interaction of the software with users, hardware, or other software. The characteristics of each user interface of the software product are specified in SRS. For the hardware interface, SRS specifies the logical characteristics of each interface among the software and hardware components. If the software is to be executed on the existing hardware, then characteristics such as memory restrictions are also specified.

11. Functions: It determines the functional capabilities of the system. For each functional requirement, the accepting and processing of inputs in order to generate outputs are specified. This includes validity checks on inputs, exact sequence of operations, relationship of inputs to output, and so on.

12. Performance requirements: It determines the performance constraints of the software system. Performance requirement is of two types: static requirements and dynamic requirements. Static requirements (also known as capacity requirements) do not impose constraints on the execution characteristics of the system. These include requirements like number of terminals and users to be supported. Dynamic requirements determine the constraints on the execution of the behaviour of the system, which includes response time (the time between the start and ending of an operation under specified conditions) and throughput (total amount of work done in a given time).

13. Logical database of requirements: It determines logical requirements to be stored in the database. This includes type of information used, frequency of usage, data entities and relationships among them, and so on.

14. Design constraint: It determines all design constraints that are imposed by standards, hardware limitations, and so on. Standard compliance determines requirements for the system, which are in compliance with the specified standards. These standards can include accounting procedures and report format. Hardware limitations implies when the software can operate on existing hardware or some pre-determined hardware. This can impose restrictions while developing the software design. Hardware limitations include hardware configuration of the machine and operating system to be used.

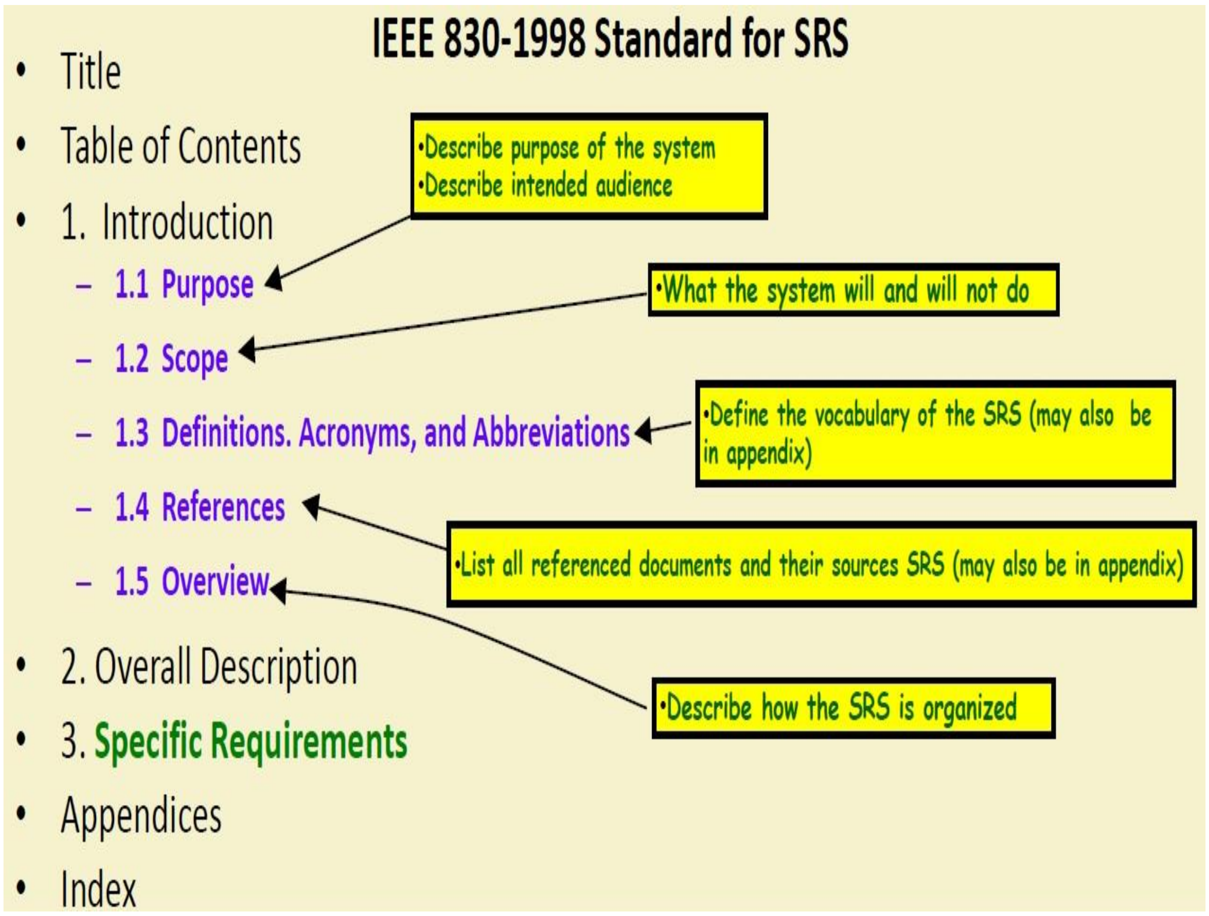
15. Software system attributes: It provide attributes such as reliability, availability, maintainability and portability. It is essential to describe all these attributes to verify that they are achieved in the final system.

16. Organizing Specific Requirements: It determines the requirements so that they can be properly organized for optimal understanding. The requirements can be organized on the basis of mode of operation, user classes, objects, feature, response, and functional hierarchy.

17. Change management process: It determines the change management process in order to identify, evaluate, and update SRS to reflect changes in the project scope and requirements.

18. Document approvals: These provide information about the approvers of the SRS document with the details such as approver's name, signature, date, and so on.

19. Supporting information: It provides information such as table of contents, index, and so on. This is necessary especially when SRS is prepared for large and complex projects.



IEEE 830-1998 Standard for SRS

- Title
- Table of Contents
- 1. Introduction
  - 1.1 Purpose
  - 1.2 Scope
  - 1.3 Definitions. Acronyms, and Abbreviations
  - 1.4 References
  - 1.5 Overview
- 2. Overall Description
- 3. **Specific Requirements**
- Appendices
- Index

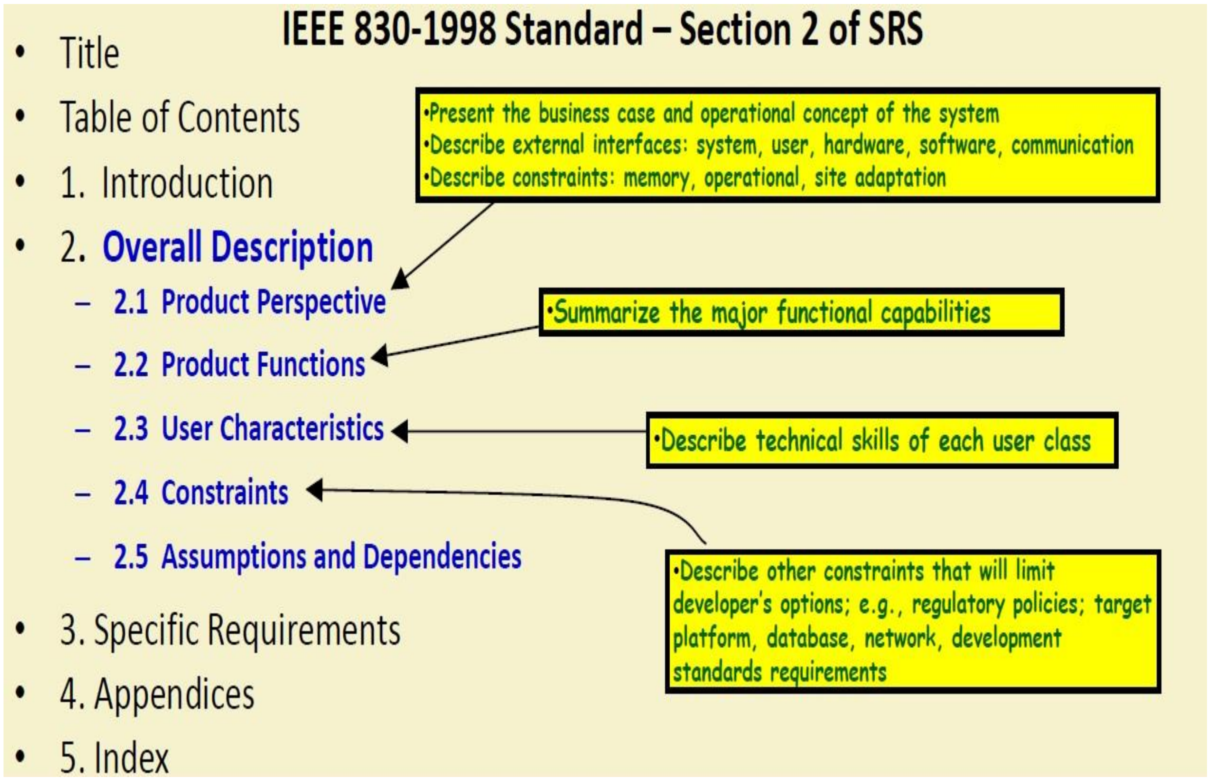
•Describe purpose of the system
•Describe intended audience

•What the system will and will not do

•Define the vocabulary of the SRS (may also be in appendix)

•List all referenced documents and their sources SRS (may also be in appendix)

•Describe how the SRS is organized



IEEE 830-1998 Standard – Section 2 of SRS

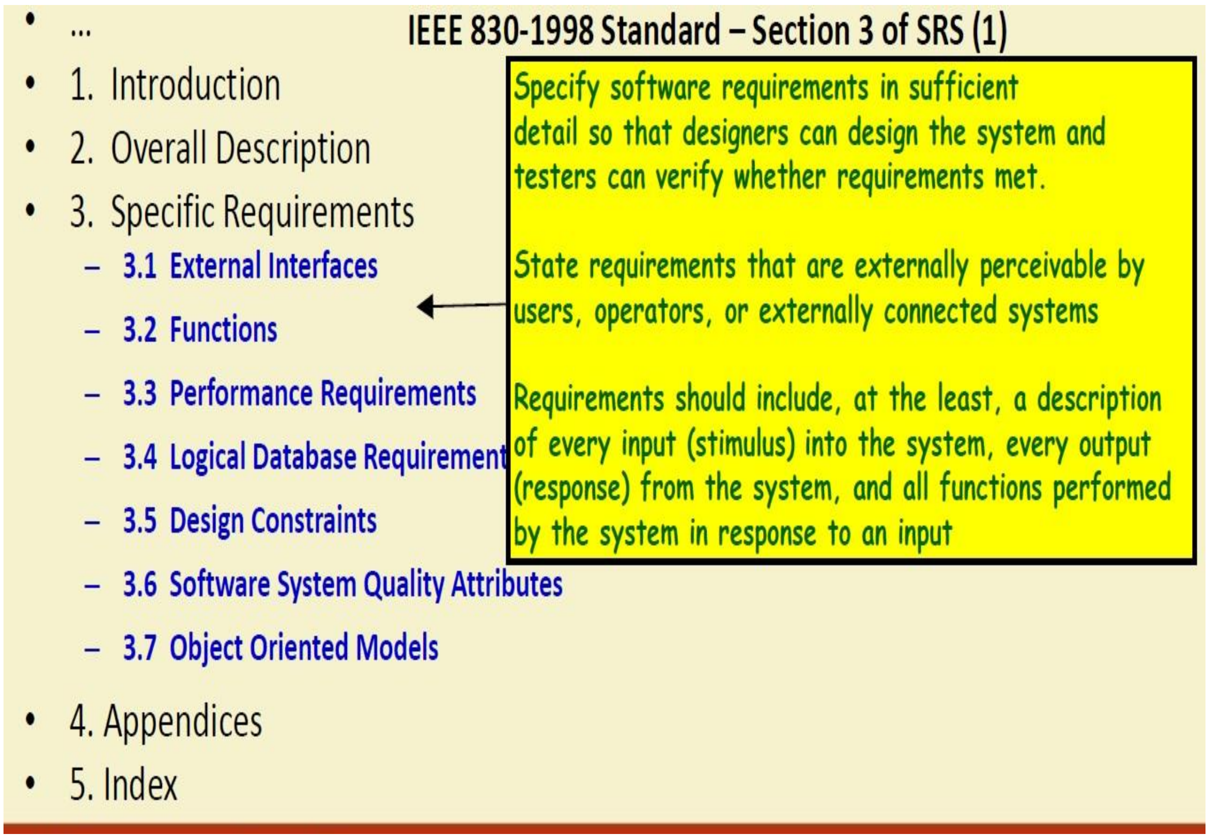
- Title
- Table of Contents
- 1. Introduction
- 2. **Overall Description**
  - 2.1 Product Perspective
  - 2.2 Product Functions
  - 2.3 User Characteristics
  - 2.4 Constraints
  - 2.5 Assumptions and Dependencies
- 3. Specific Requirements
- 4. Appendices
- 5. Index

•Present the business case and operational concept of the system
•Describe external interfaces: system, user, hardware, software, communication
•Describe constraints: memory, operational, site adaptation

•Summarize the major functional capabilities

•Describe technical skills of each user class

•Describe other constraints that will limit developer's options; e.g., regulatory policies; target platform, database, network, development standards requirements



IEEE 830-1998 Standard – Section 3 of SRS (1)

- …
- 1. Introduction
- 2. Overall Description
- 3. Specific Requirements
  - 3.1 External Interfaces
  - 3.2 Functions
  - 3.3 Performance Requirements
  - 3.4 Logical Database Requirement
  - 3.5 Design Constraints
  - 3.6 Software System Quality Attributes
  - 3.7 Object Oriented Models
- 4. Appendices
- 5. Index

Specify software requirements in sufficient detail so that designers can design the system and testers can verify whether requirements met.

State requirements that are externally perceivable by users, operators, or externally connected systems

Requirements should include, at the least, a description of every input (stimulus) into the system, every output (response) from the system, and all functions performed by the system in response to an input

WHAT ARE THE VALIDATION METHODS PERFORMED?

There are three methods of validation performed:

1. **Domain-validated certificates:** Only the verified owner of the domain name can purchase an SSL certificate for the domain. Validation is done via email sent to the domain owner. Domain validated SSL certificates can be issued very quickly - often in minutes.

2. **Organization-validated certificates:** When corporate identity validation is important, an SSL Certificate for the organization assures customers that the website is trustworthy and secure. Only verified representatives of the organization may purchase these certificates and business licences or other proof is required. The Certificate Authority will verify through phone call to ensure that the certificate request is legitimate.

3. **Extended Validation (EV) certificates:** With Extended Validation, as well as displaying the certificate seal, the address bar is displayed in green, providing customers with an extra level of confidence. The green address bar is a strong visual indication that the site has an Extended Validation Certificate. The Security Status bar displays the organization name and the name of the Certificate Authority (CA).

   In order to be approved for an Extended Validation certificate, the certificate authority will actively check the Organization and the individual applying for the certificate.

This is to verify that the Organization is positively the Organization they claim to be, and the individual requesting the certificate is someone who is authorized to request a digital certificate. Extended Validation may take as long as one week to complete.

ESTABLISHING GROUNDWORK

- The requirements engineering is simply a matter of conducting meaningful conversations with colleagues who are well-known members of the team.
- The steps required to establish the groundwork for an understanding of software requirements—

Identifying Stakeholders

- Identify benefits in a direct or indirect way from the system, also identify beneficiaries which is being developed.
- At inception, you should create a list of people who will contribute input as requirements.

Recognizing Multiple Viewpoints

- As many different stakeholders exist, the requirements of the system will be explored from many different points of view.
- Also, be consider views nontechnical stakeholders
- Support engineers may focus on the maintainability of the
Software, each of these constituencies will contribute information to the requirements engineering process. As information from multiple viewpoints is collected,
- Emerging requirements may be inconsistent or may conflict with one another.
- It is Important to take consider all stakeholders view

Working toward Collaboration

- If five stakeholders are involved in a software project, we may have five different opinions about the proper set of requirements. Consider and collaborate all views
- customers and other stakeholders must collaborate among themselves and with software engineering practitioners its job of a requirements engineer to identify areas of commonality
- Also identify and areas of conflict or inconsistency (i.e., requirements that are desired by one stakeholder but conflict with the needs of another stakeholder).

Asking the Questions

- Questions asked at the inception of the project the first set of context-free questions focuses on the customer and other stakeholders, the overall project goals and benefits.

- For example, we might ask:
  - Who is behind the request for this work?
  - Who will use the solution?
  - What will be the economic benefit of a successful solution?
  - Is there another source for the solution that you need?
- These questions help to identify all stakeholders who will have interest in the software to be built.
- The questions identify the measurable benefit of a successful implementation and possible alternatives to custom software development

ELICITING REQUIREMENTS

- Requirements elicitation also called requirements gathering, combines elements of problem solving, elaboration, negotiation, and specification.
- In team-oriented approach to requirements gathering, stakeholders work together to identify the problem, propose solution, negotiate etc

Collaborative Requirements Gathering

- Possibility that Each stakeholder makes use of a slightly different scenario, but all apply some variation
- For this
  - Meetings are conducted and attended by both software engineers and other stakeholders.
  - Rules for preparation and participation are established.
  - An agenda enough to cover all important points

Quality Function Deployment

- Quality function deployment (QFD) is a quality management technique that translates the needs of the customer into technical requirements for software.
- It concentrates on maximizing customer satisfaction from the software engineering process"
- QFD emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the engineering process.
- QFD identifies three types of requirements
  <u>Normal requirements</u>. (graphical displays, specific system functions,)
  <u>Expected requirements</u>. (ease of human/machine interaction,
  correctness and reliability, and ease of software installation.)
  <u>Exciting requirements</u>. (e.g., multitouch screen, visual voice mail)

Usage Scenarios

- It is difficult to understand how functions and features will be used by different classes of end users.
- The developers and users can create a set of scenarios that identify a thread of usage for the system to be constructed.

Elicitation Work Products

work products include

- A statement of need and feasibility.
- A bounded statement of scope for the system or product.
- A list of customers, users, and other stakeholders who participated in requirements elicitation.
- A description of the system's technical environment.

DEVELOPING USE CASES

- use case depicts the software or system from the end user's point of view.
- The first step in writing a use case is to define the set of "actors" that will be involved in the story. Actors are the different people (or devices) that use the system or product within the context of the function and behaviour that is to be described.
- Actors represent the roles that people (or devices) play as the system operates.
- Defined somewhat more formally, an actor is anything that communicates with the system or product and that is external to the system itself.
- Every actor has one or more goals when using the system.
- It is important to note that an actor and an end user are not necessarily the same thing.
- A typical user may play a number of different roles when using a system, whereas an actor represents a class of external entities (often, but not always, people) that play just one role in the context of the use case.
- As an example, consider a machine operator (a user) who interacts with the control computer for a manufacturing cell that contains a number of robots and numerically controlled machines.
- After careful review of requirements, the software for the control computer requires four different modes (roles) for interaction: programming mode, test mode, monitoring mode, and troubleshooting mode.
- Therefore, four actors can be defined: programmer, tester, monitor, and trouble-shooter. In some cases, the machine operator can play all of these roles. In others, different people may play the role of each actor.
- Because requirements elicitation is an evolutionary activity, not all actors are identified during the first iteration.
- It is possible to identify primary actors [Jac92] during the first iteration and secondary actors as more is learned about the system.

- Primary actors interact to achieve required system function and derive the intended benefit from the system.
- They work directly and frequently with the software. Secondary actors support the system so that primary actors can do their work.
- Once actors have been identified, use cases can be developed. Jacobson [Jac92] suggests a number of questions that should be answered by a use case:

  1. Who is the primary actor, the secondary actor(s)?
  2. What are the actor's goals?
  3. What preconditions should exist before the story begins?
  4. What main tasks or functions are performed by the actor?
  5. What exceptions might be considered as the story is described?
  6. What variations in the actor's interaction are possible?
  7. What system information will the actor acquire, produce, or change?
  8. Will the actor have to inform the system about changes in the external environment?
  9. What information does the actor desire from the system?
  10. Does the actor wish to be informed about unexpected changes?

VALIDATING REQUIREMENTS

The requirements represented by the model are prioritized by the stakeholders and requirements will be implemented as software increments.

A review of the requirements model addresses the following questions before validation:

• Is each requirement consistent with the overall objectives for the system/product?
• Have all requirements been specified at the proper level of abstraction?
• Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
• Is each requirement bounded and unambiguous?
• Does each requirement have attribution? That is, is a source noted for each requirement?
• Do any requirements conflict with other requirements?
• Is each requirement achievable in the technical environment that will house the system or product?
• Is each requirement testable, once implemented?
• Does the requirement model properly reflect the information, function, and behaviour of the system to be built?
• Has the requirements provides progressively more detailed information about the system?
• Have requirements patterns been used to simplify the requirements model?

Project Scheduling

Basic Concept

Software project scheduling is an action that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks

Reasons for software delivered late:

• An unrealistic deadline established

• Changing customer requirements that are not reflected in schedule changes.

• An honest underestimate of the amount of effort and/or the number of resources.

• Predictable and/or unpredictable risks that were not considered.

• Technical difficulties that could not have been foreseen in advance.

• Human difficulties that could not have been foreseen in advance.

• Miscommunication among project staff that results in delays, etc.

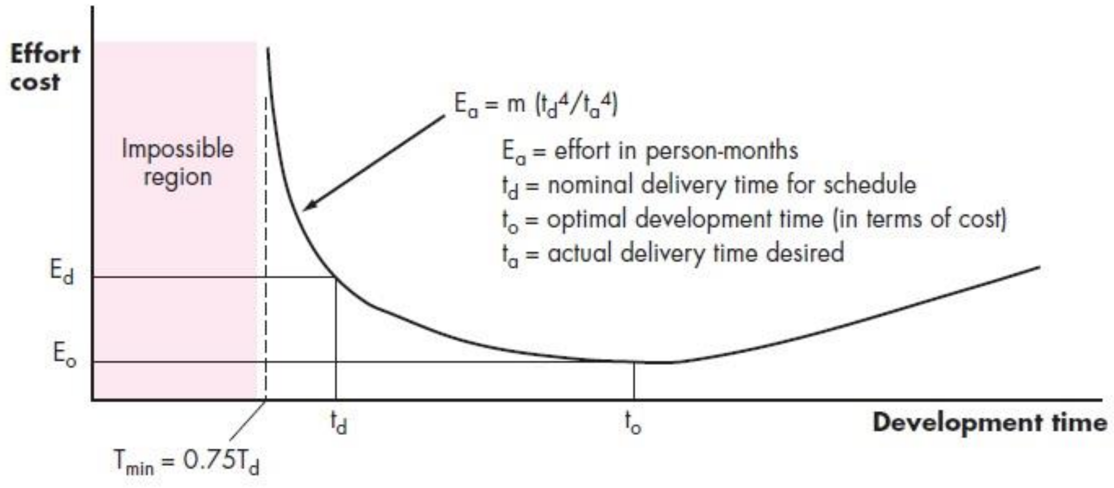
Basic Principles Software project scheduling

- **Compartmentalization**. The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both the product and the process are refined.
- **Interdependency.** The interdependency of each compartmentalized activity or task must be determined. Some tasks must occur in sequence, while others can occur in parallel. Some activities cannot commence until the work product produced by another is available. Other activities can occur independently.
- **Time allocation.** Each task to be scheduled must be allocated some number of work units (e.g., person-days of effort). In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies and whether work will be conducted on a full-time or part-time basis.
- **Effort validation.** Every project has a defined number of people on the software team. As time allocation occurs, you must ensure that no more than the allocated number of people has been scheduled at any given time.
- **Defined responsibilities.** Every task that is scheduled should be assigned to a specific team member.
- **Defined outcomes.** Every task that is scheduled should have a defined outcome. For software projects, the outcome is normally a work product (e.g., the design of a component) or a part of a work product. Work products are often combined in deliverables.

- **Defined milestones.** Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality.

People and Effort

- In a small software development of project, a single person can analyse requirements, perform design, generate code, and conduct tests. As the size of a project increases, more people must become involved.
- There is a common **myth** that is still believed by many managers who are responsible for software development projects: **"If we fall behind schedule, we can always add more programmers and catch up later in the project."** Unfortunately, adding people late in a project often has a disruptive effect on the project, causing schedules to slip even further.
- The people who **are added must learn the system**, and the people who teach them are the same people who were doing the work. While teaching, no work is done, and the project falls further behind.
- Over the years, eit founds that **project schedules are elastic**. That is, it is possible to compress a desired project completion date by adding additional resources to some extent.
- It is also possible to extend a completion date (by reducing the number of resources). The Putnam-Norden-Rayleigh (PNR) Curve5 provides an indication of the relationship between effort applied and delivery time for a software project.



A curve, representing project effort as a function of delivery time, is shown in Figure

The curve indicates a minimum value to that indicates the least cost for delivery

we assume that a project team has estimated a level of effort **Ed** will be required to achieve a nominal delivery time **td** that is optimal in terms of schedule and available resources. Although it is possible to accelerate delivery, the curve rises very sharply to the left of td. In fact, the PNR curve indicates that the project delivery time cannot be compressed much beyond 0.75td. If we attempt further compression, the project moves into "the impossible region" and risk of failure becomes very high.

The PNR curve also indicates that the lowest cost delivery option,
to _ 2td. The implication here is that delaying project delivery can reduce costs significantly. Of course, this must be weighed against the business cost associated with the delay.

Effort Distribution

- Each of the software project estimation, leads to estimates of work unit person-months required to complete software development.
- A recommended distribution of effort across the software process is often referred to as the **40–20–40** rule.
- **Forty percent of all effort is allocated to frontend analysis and design. A similar percentage is applied to back-end testing**. And coding (20 percent of effort) is deemphasized.
- This effort distribution should be used as a guideline only. The characteristics of each project dictate the distribution of effort.
- **Work expended on project planning rarely accounts for more than** 2 to 3 percent of effort, unless the plan commits an organization to large expenditures with high risk.
- Customer communication and requirements analysis may comprise 10 to 25 percent of project effort.
- A range of 20 to 25 percent of effort is normally applied to software design.
- The criticality of the software often dictates the amount of testing that is required. If software is human rated, even higher percentages are typical

Task Set

- A task set is a collection of software engineering work tasks, milestones, work products, and quality assurance filters that must be accomplished to complete a particular project
- Task Set enable you to define, develop, and support computer software
- Process model that is chosen, or the work that a software team performs is achieved through a set of tasks that
- No single task set is appropriate for all projects. The set of tasks that would be requires
- An effective software process should define a collection of task sets, each designed to meet the needs of different types of projects.

Example of Task Set

Concept development projects are approached by applying the following actions (Concept development projects that are initiated to explore some new business concept or application of some new technology.)

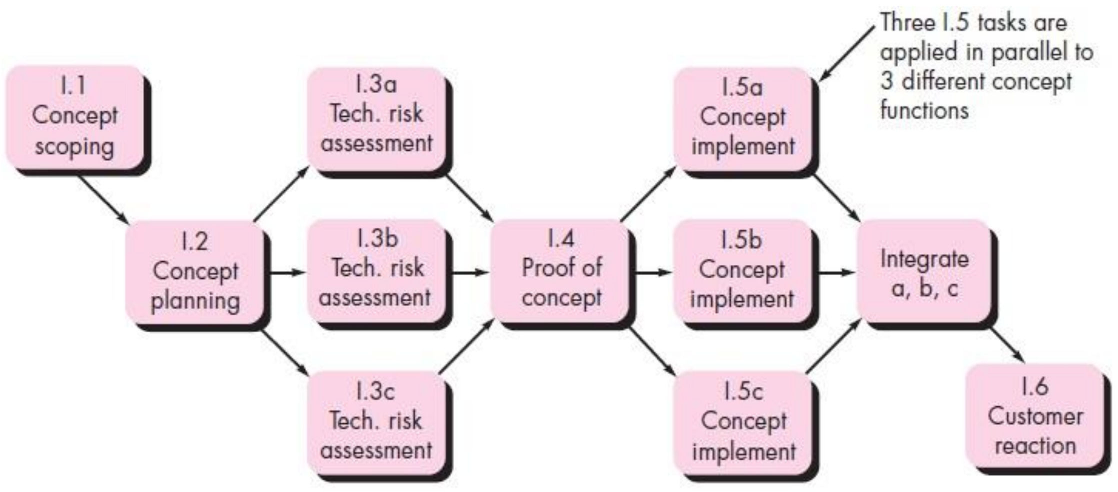
- Concept **scoping determines** the overall scope of the project.
- Preliminary concept planning establishes the organization's ability to undertake the work implied by the project scope.
- **Technology risk assessment** evaluates the risk associated with the technology to be implemented as part of the project scope.
- Proof of concept demonstrates the viability of a **new technology in the software context**.

Concept implementation implements the concept representation in a manner that can be reviewed by a customer and is used for "marketing" purposes when a concept must be sold to other customers or management.

Customer reaction to the concept solicits feedback on a new technology concept and targets specific customer applications.

Task Network

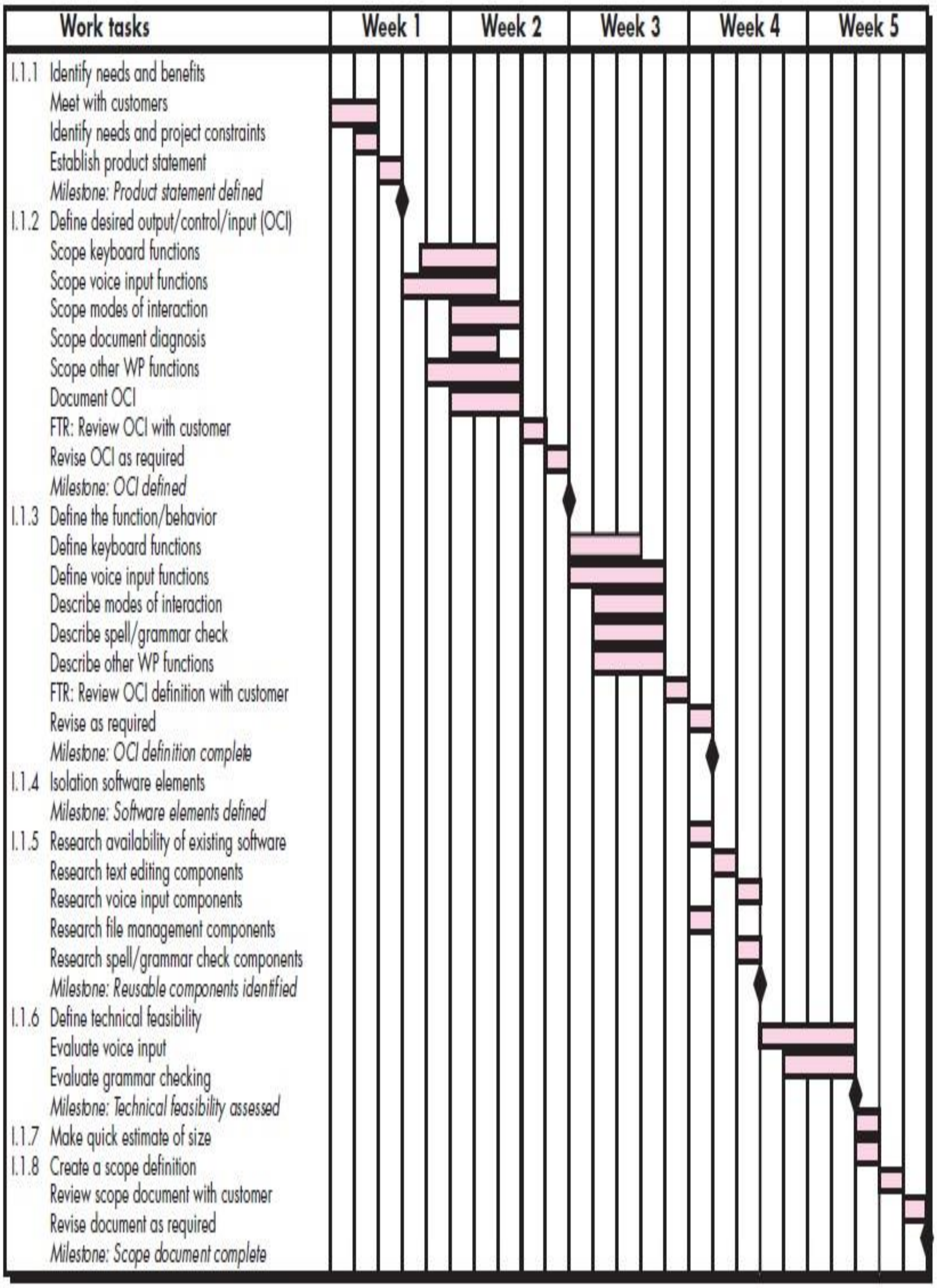
- **Individual tasks and subtasks** have interdependencies based on their sequence.
- when more than one person is involved in a software engineering project,development activities and tasks will be performed in parallel, concurrent tasks must be coordinated
- A task network, also called an activity network, is a graphic representation of the task flow for a project.
- It is sometimes used as the mechanism through which task sequence and dependencies are input to an automated project scheduling tool.
- the task network depicts major software engineering actions.

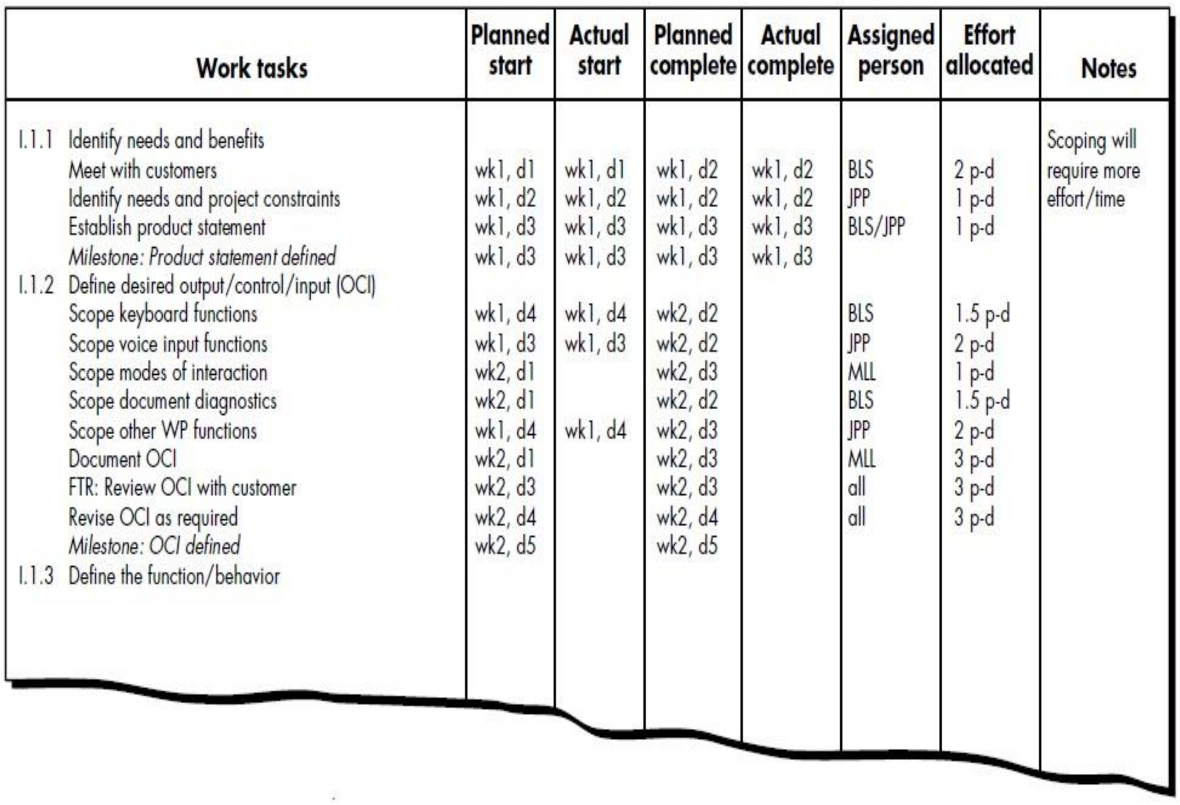


- The concurrent nature of software engineering actions leads to a number of important scheduling requirements.

GANTT CHART

- Time-line chart, also called a Gantt chart,A time-line chart can be developed for the entire project.
- Alternatively, separate charts can be developed for each project function or for each individual working on the project.
- Figure illustrates the format of a time-line chart. It depicts a part of a software project schedule
- All project tasks (for concept scoping) are listed in the lefthand column.
- Thhe horizontal bars indicate the duration of each task.
- When multiple bars occur at the same time on the calendar, task concurrency is implied.
- The diamonds indicate milestones.


Work tasks
Week 1
Week 2
Week 3
Week 4
Week 5
I.1.1 Identify needs and benefits
Meet with customers
Identify needs and project constraints
Establish product statement
Milestone: Product statement defined
I.1.2 Define desired output/control/input (OCI)
Scope keyboard functions
Scope voice input functions
Scope modes of interaction
Scope document diagnosis
Scope other WP functions
Document OCI
FTR: Review OCI with customer
Revise OCI as required
Milestone: OCI defined
I.1.3 Define the function/behavior
Define keyboard functions
Define voice input functions
Describe modes of interaction
Describe spell/grammar check
Describe other WP functions
FTR: Review OCI definition with customer
Revise as required
Milestone: OCI definition complete
I.1.4 Isolation software elements
Milestone: Software elements defined
I.1.5 Research availability of existing software
Research text editing components
Research voice input components
Research file management components
Research spell/grammar check components
Milestone: Reusable components identified
I.1.6 Define technical feasibility
Evaluate voice input
Evaluate grammar checking
Milestone: Technical feasibility assessed
I.1.7 Make quick estimate of size
I.1.8 Create a scope definition
Review scope document with customer
Revise document as required
Milestone: Scope document complete

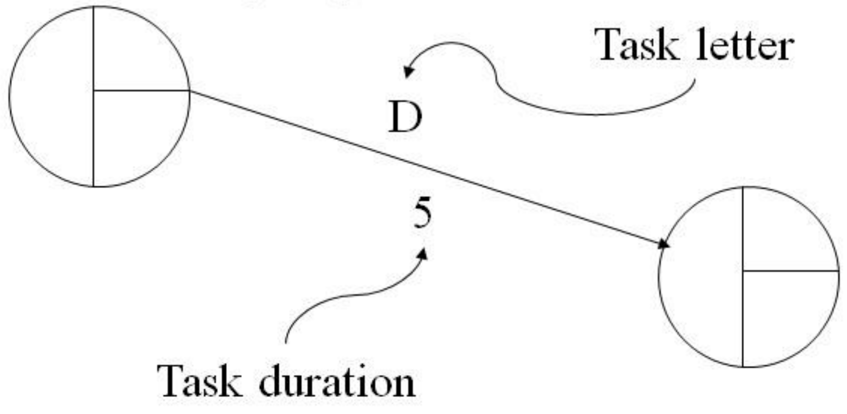


Work tasks	Planned start	Actual start	Planned complete	Actual complete	Assigned person	Effort allocated	Notes
I.1.1 Identify needs and benefits							Scoping will require more effort/time
Meet with customers	wk1, d1	wk1, d1	wk1, d2	wk1, d2	BLS	2 p-d	
Identify needs and project constraints	wk1, d2	wk1, d2	wk1, d2	wk1, d2	JPP	1 p-d	
Establish product statement	wk1, d3	wk1, d3	wk1, d3	wk1, d3	BLS/JPP	1 p-d	
Milestone: Product statement defined	wk1, d3	wk1, d3	wk1, d3	wk1, d3			
I.1.2 Define desired output/control/input (OCI)							
Scope keyboard functions	wk1, d4	wk1, d4	wk2, d2		BLS	1.5 p-d	
Scope voice input functions	wk1, d3	wk1, d3	wk2, d2		JPP	2 p-d	
Scope modes of interaction	wk2, d1		wk2, d3		MLL	1 p-d	
Scope document diagnostics	wk2, d1		wk2, d2		BLS	1.5 p-d	
Scope other WP functions	wk1, d4	wk1, d4	wk2, d3		JPP	2 p-d	
Document OCI	wk2, d1		wk2, d3		MLL	3 p-d	
FTR: Review OCI with customer	wk2, d3		wk2, d3		all	3 p-d	
Revise OCI as required	wk2, d4		wk2, d4		all	3 p-d	
Milestone: OCI defined	wk2, d5		wk2, d5				
I.1.3 Define the function/behavior							

PERT (PROJECT EVALUATION REVIEW TECHNIQUE)

TASK ID	Task Description	Preceed ID	Succ. ID	Duration
A	Specification	1	2	3
B	High Level Design	2	3	2
C	Detailed Design	3	4	2
D	Code/Test Main	4	5	7
E	Code/Test DB	4	6	6
F	Code/Test UI	4	7	3
G	Write test plan	4	8	2
	Dummy Task	5	8	
	Dummy Task	6	8	
	Dummy Task	7	8	
H	Integrate/System Test	8	9	5
I	Write User Manual	8	10	2
J	Typeset User Manual	10	9	1

•**Parts of a task/activity edge**

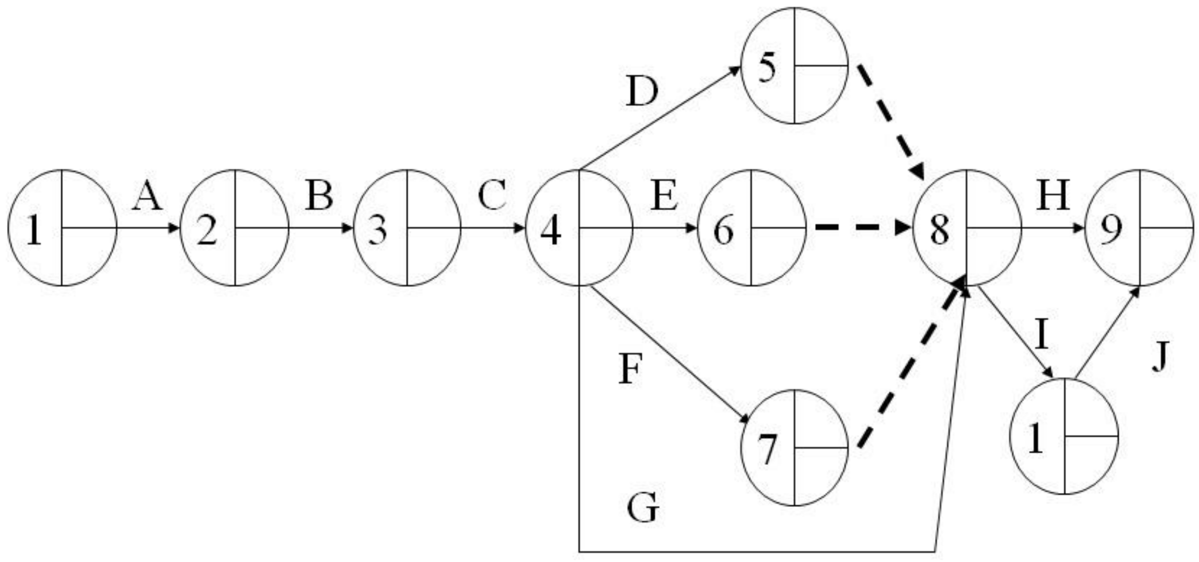


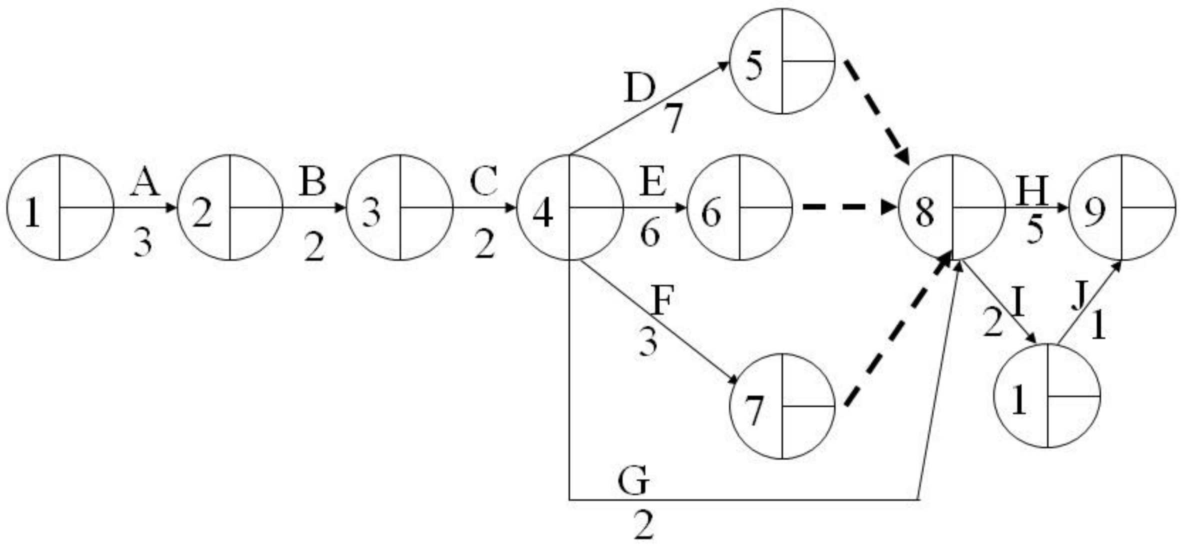
•**Task letter:**

- Often keyed to a legend to tell which task it represents

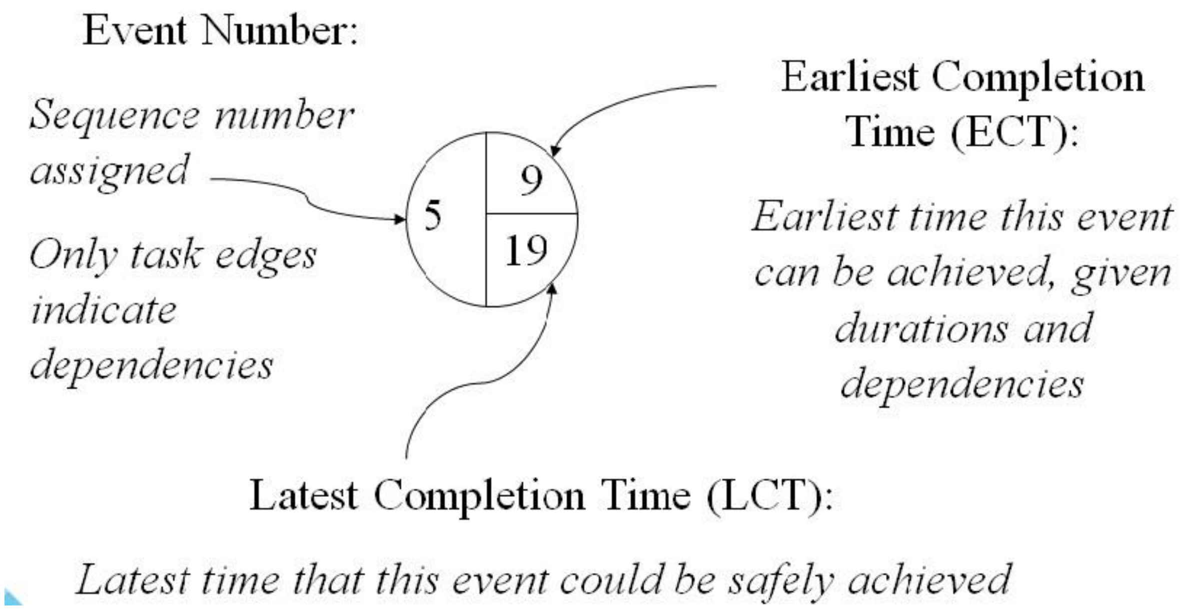
•**Task duration = how long (e.g. days, hours) task will take**

<u>Construct a project network</u>





Calculate ECT and LCT


Event Number:

Sequence number assigned

Only task edges indicate dependencies

Earliest Completion Time (ECT):

Earliest time this event can be achieved, given durations and dependencies

Latest Completion Time (LCT):

Latest time that this event could be safely achieved

ECT Earliest completion Time

ECT = earliest time event can be completed

To calculate:

- For an event not depending on others: ECT = 0

  Usually this is the first event
- For an event E depending on one or more others:

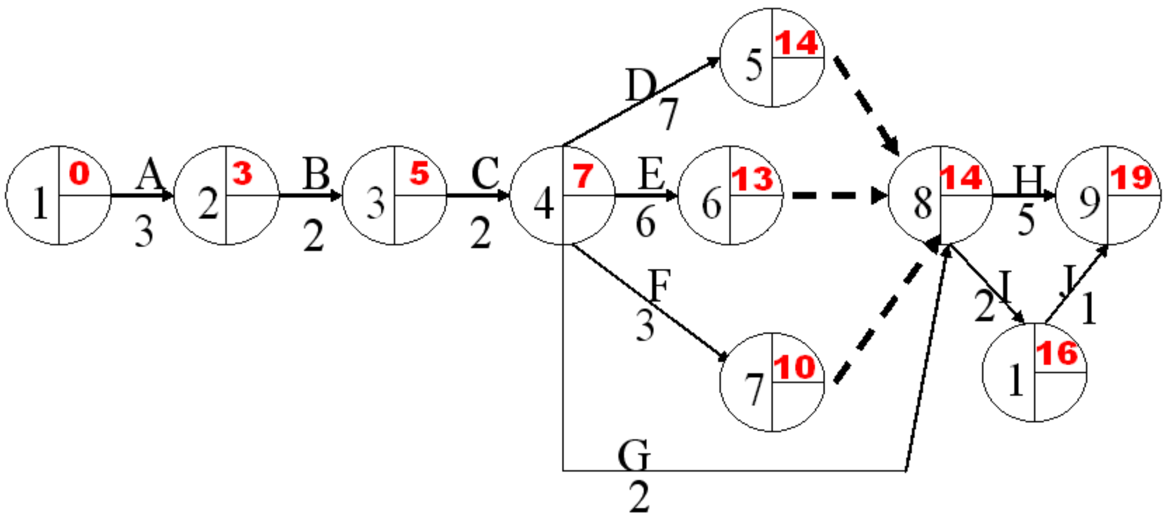
  Calculate ECTs of event(s) that E depends on

  Add duration(s) of task(s) leading to E

  If E depends on more than one event, take MAX

Proceed left to right (→) through the chart

Exercise: calculate the ECT for our example.



LCT Latest Completion Time

LCT = latest time event can be completed, while still finishing last ask at indicated time

To calculate:

- For an event which no other events depend on: LCT = ECT

  Generally there will only be one such event
- For an event E which one or more others depend on:

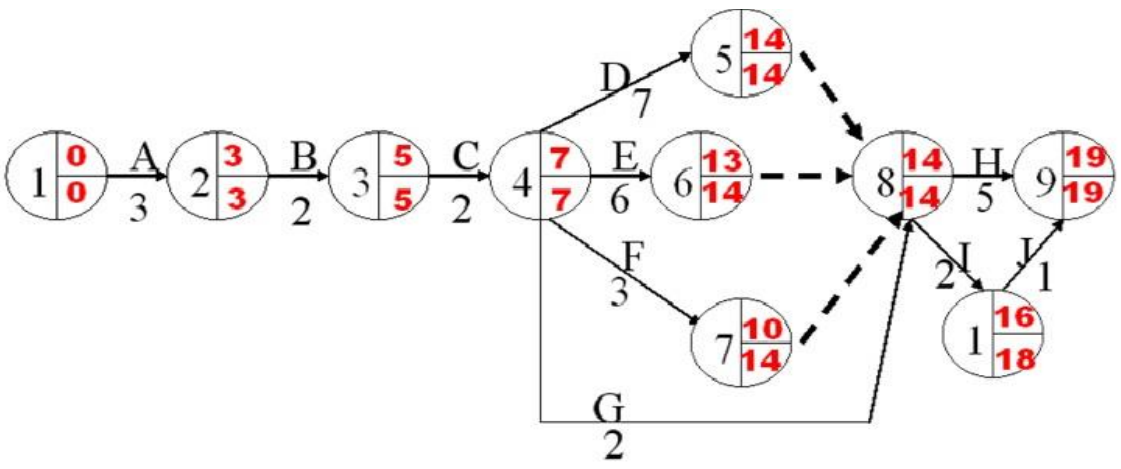
  Calculate LCTs of event(s) that depend on E

  Subtract duration(s) of task(s) leading from E

  If more than one event depends on E, take **MINIMUM**

Proceed right to left (←) through PERT chart

Exercise: calculate LCT for our example



Find Critical Path

Critical Path: **Path through chart such that if any deadline slips, the final deadline slips (where all events have ECT = LCT (usually there is only one)**

In software example:

- Task I is not on the critical path: even if we don't finish it until time 18, we're still okay
- Task D is on the critical path: if we don't finish it until for example, time 16, then:

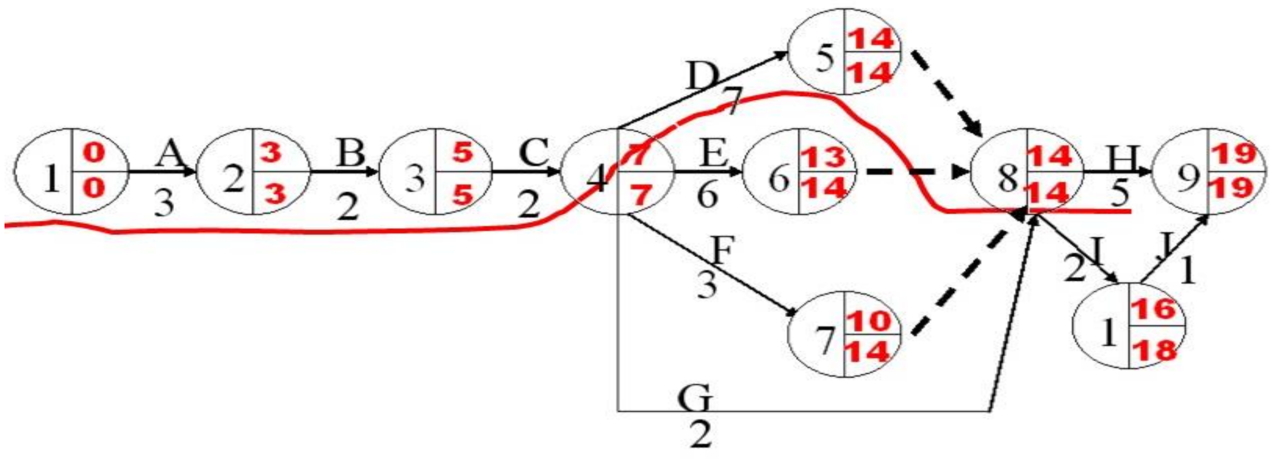
We can't start task H (duration 3) until time 16

So we can't complete task H until time 21

We can use PERT charts for

- Identifying the critical path
- Reallocating resources, e.g. from non-critical to critical tasks.

Conditions :1. Select nodes where ECT=LCT



CPM= 1-2-3-4-5-8-9

=3+2+2+7+5=19

Ex-

Task	**Prec Tasks**	**Description**	**Time(hrs)**
A	none	decide on date for party	1
B	A	book bouncy castle	1
C	A	send invitations	4
D	C	receive replies	7
E	D	buy toys and balloons	1
F	D	buy food	3
G	E	blow up balloons	2
H	F	make food	1
I	H, G	decorate	1
J	B	get bouncy castle	1
K	J, I	have party	1
L	K	clean up	4
M	K	send back bouncy castle	1
N	L	send thank you letters	3
O	M	donate unwanted gifts	3

For given data of PERT,

1.Construct system network

2.Find values of ECT and LCTs

3.Find the CPM