UNIT-II

Planning and Managing the Project: Tracking Progress, Project Personnel, Effort Estimation, Risk Management, the Project Plan, Process Models and Project Management, Information Systems Example, Real-time Example.

Requirement Engineering: A bridge to design and construction, Requirement Engineering tasks, Initiating Requirement Engineering Process, Eliciting Requirement, Developing Uses cases, Building the Analysis Model, Negotiating Requirements, Validating Requirements.

PART - I

Tracking Progress

The following questions will be raised during this process:

- Do we understand customer's needs?
- Can we design a system to solve customer's problems or satisfy customer's needs?
- How long will it take to develop the system?
- How much will it cost to develop the system?

Project Schedule:

- Describes the software-development cycle for a particular project by
  - enumerating the phases or stages of the project
  - breaking each phase into discrete tasks or activities to be completed
- Portrays the interactions among the activities and estimates the times that each task or activity will take

Project Schedule: Approach:

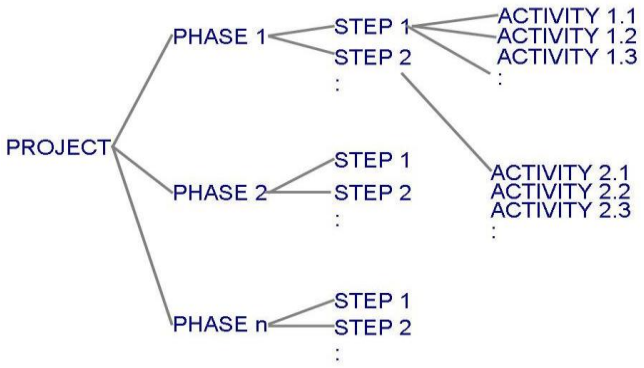
- Understanding customer's needs by listing all project deliverables
  - Documents
  - Demonstrations of function
  - Demonstrations of subsystems
  - Demonstrations of accuracy
  - Demonstrations of reliability, performance or security
- Determining milestones and activities to produce the deliverables

Milestones and activities:

- **Activity**: takes place over a period of time

- **Milestone**: completion of an activity -- a particular point in time
- **Precursor**: event or set of events that must occur in order for an activity to start
- **Duration**: length of time needed to complete an activity
- **Due date**: date by which an activity must be completed

- Project development can be separated into a succession of phases which are composed of steps, which are composed of activities



- Table 3.1 shows the phases, steps and activities to build a house
  - landscaping phase
  - building the house phase
- Table 3.2 lists milestones for building the house phase

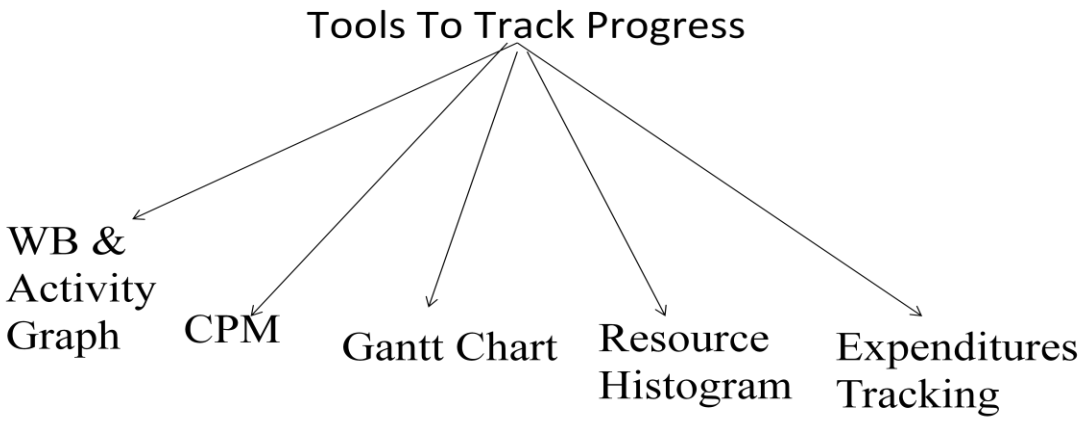
Phases, Steps, and Activities in Building a House

Table 3.1

Phase 1: Landscaping the lot	**Phase 2: Building the house**
Step 1.1: Clearing and grubbing	*Step 2.1: Prepare the site*
Activity 1.1.1: Remove trees	Activity 2.1.1: Survey the land
Activity 1.1.2: Remove stumps	Activity 2.1.2: Request permits
Step 1.2: Seeding the turf	Activity 2.1.3: Excavate for the foundation
Activity 1.2.1: Aerate the soil	Activity 2.1.4: Buy materials
Activity 1.2.2: Disperse the seeds	*Step 2.2: Building the exterior*
Activity 1.2.3: Water and weed	Activity 2.2.1: Lay the foundation
Step 1.3: Planting shrubs and trees	Activity 2.2.2: Build the outside walls
Activity 1.3.1: Obtain shrubs and trees	Activity 2.2.3: Install exterior plumbing
Activity 1.3.2: Dig holes	Activity 2.2.4: Exterior electrical work
Activity 1.3.3: Plant shrubs and trees	Activity 2.2.5: Exterior siding
Activity 1.3.4: Anchor the trees and mulch around them	Activity 2.2.6: Paint the exterior
	Activity 2.2.7: Install doors and fixtures
	Activity 2.2.8: Install roof
	Step 2.3: Finishing the interior
	Activity 2.3.1: Install the interior plumbing
	Activity 2.3.2: Install interior electrical work
	Activity 2.3.3: Install wallboard
	Activity 2.3.4: Paint the interior
	Activity 2.3.5: Install floor covering
	Activity 2.3.6: Install doors and fixtures

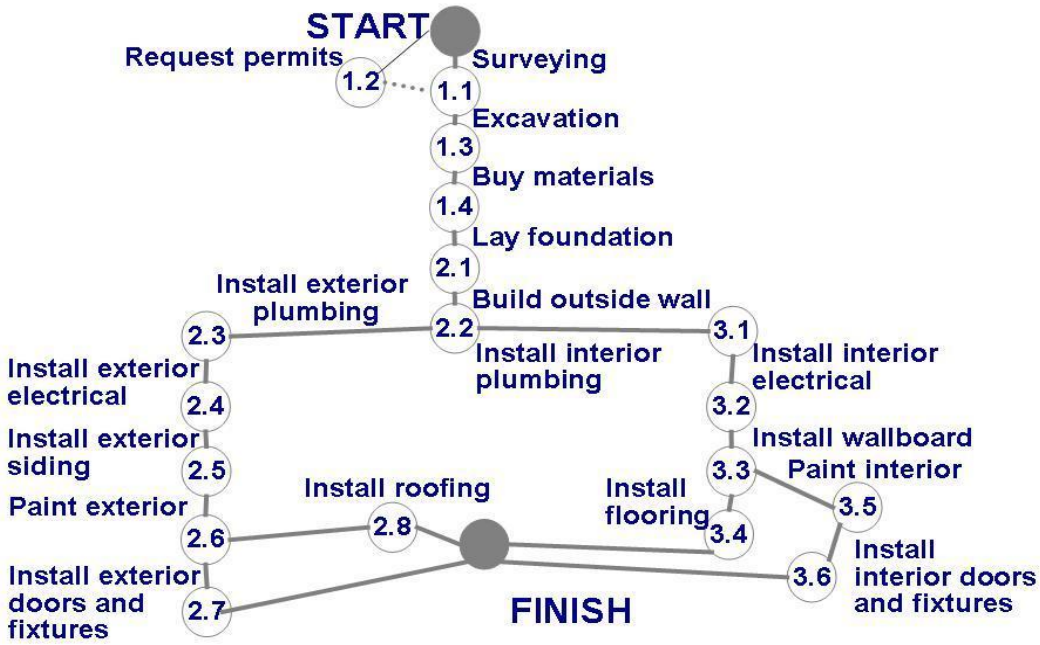
Table 3.2 Milestones in Building a House

1.1. Survey complete
1.2. Permits issued
1.3. Excavation complete
1.4. Materials on hand
2.1. Foundation laid
2.2. Outside walls complete
2.3. Exterior plumbing complete
2.4. Exterior electrical work complete
2.5. Exterior siding complete
2.6. Exterior painting complete
2.7. Doors and fixtures mounted
2.8. Roof complete
3.1. Interior plumbing complete
3.2. Interior electrical work complete
3.3. Wallboard in place
3.4. Interior painting complete
3.5. Floor covering laid
3.6. Doors and fixtures mounted



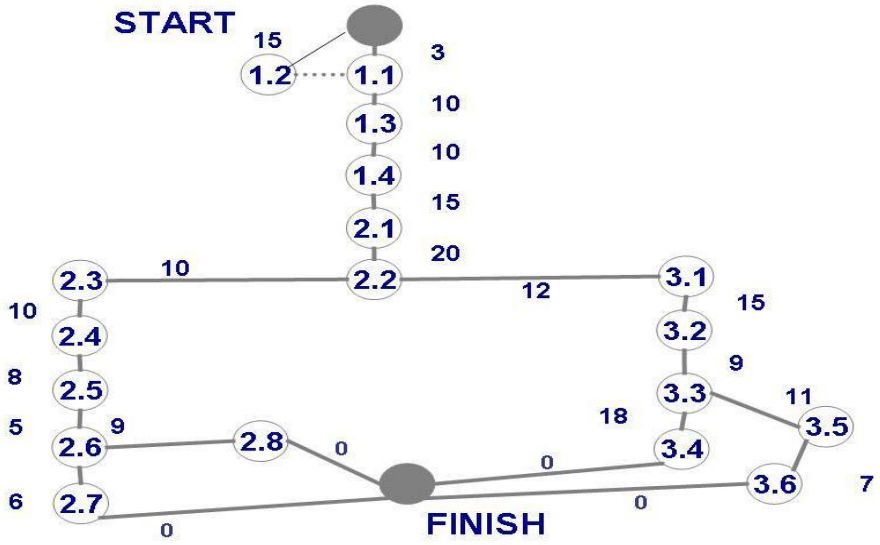
Work Breakdown and Activity Graphs:

- Work breakdown structure depicts the project as a set of discrete pieces of work
- Activity graphs depict the dependencies among activities
  - *Nodes*: project milestones
  - *Lines*: activities involved
- Activity graph for building a house



Estimating Completion:

- Adding estimated time in activity graph of each activity to be completed tells us more about the project's schedule



Critical Path Method (CPM)

- Minimum amount of time it will take to complete a project
  - Reveals those activities that are most critical to completing the project on time

- **Real time (actual time):** estimated amount of time required for the activity to be completed
- **Available time:** amount of time available in the schedule for the activity's completion
- **Slack time:** the difference between the available time and the real time for that activity
- **Critical path**: the slack at every node is zero
- can be more than one in a project schedule
- **Slack time** = **available time – real time** = **latest start time – earliest start time**

Activity	Earliest start time	Latest start time	Slack
1.1	1	13	12
1.2	1	1	0
1.3	16	16	0
1.4	26	26	0
2.1	36	36	0
2.2	51	51	0
2.3	71	83	12
2.4	81	93	12
2.5	91	103	12
2.6	99	111	12
2.7	104	119	15
2.8	104	116	12
3.1	71	71	0
3.2	83	83	0
3.3	98	98	0
3.4	107	107	0
3.5	107	107	0
3.6	118	118	0
Finish	124	124	0

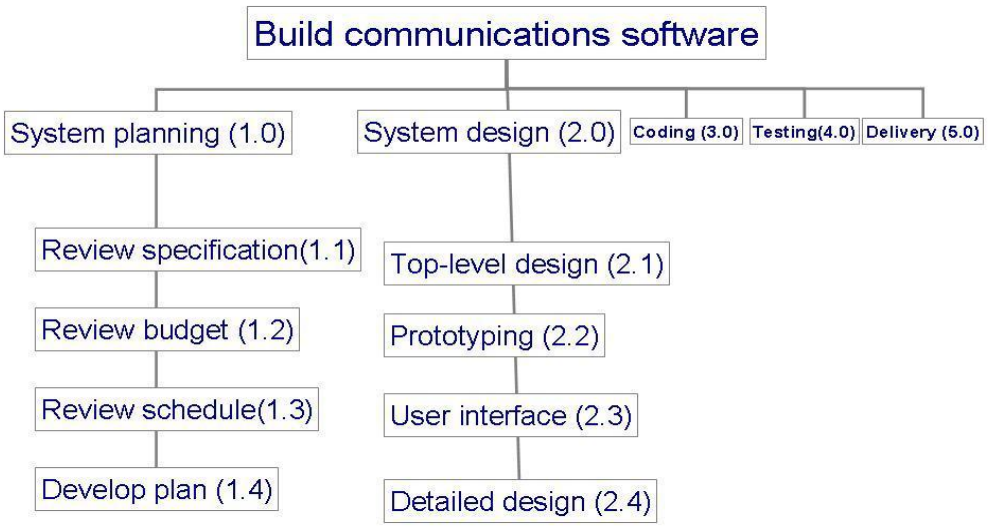
CPM Bar Chart

- Including information about the early and late start dates
- Asterisks indicate the critical path

Description	Early Date	Late Date	Jan 1	Jan 8	Jan 15	Jan 22	Jan 29	Feb 5	Feb 12	Feb 17	Feb 24
Test of phase 1	1 Jan 98	5 Feb 98	**********	**********							
Define test cases	1 Jan 98	8 Jan 98	****								
Write test plan	9 Jan 98	22 Jan 98			******						
Inspect test plan	9 Jan 98	22 Jan 98			******						
Integration testing	23 Jan 98	1 Feb 98				******					
Interface testing	23 Jan 98	1 Feb 98				--FFFFF					
Document results	23 Jan 98	1 Feb 98				-----FFF					
System testing	2 Feb 98	17 Feb 98						******* ***			
Performance tests	2 Feb 98	17 Feb 98						---- FFFFF			
Configuration tests	2 Feb 98	17 Feb 98						---- FFFFF			
Document results	17 Feb 98	24 Feb 98									****

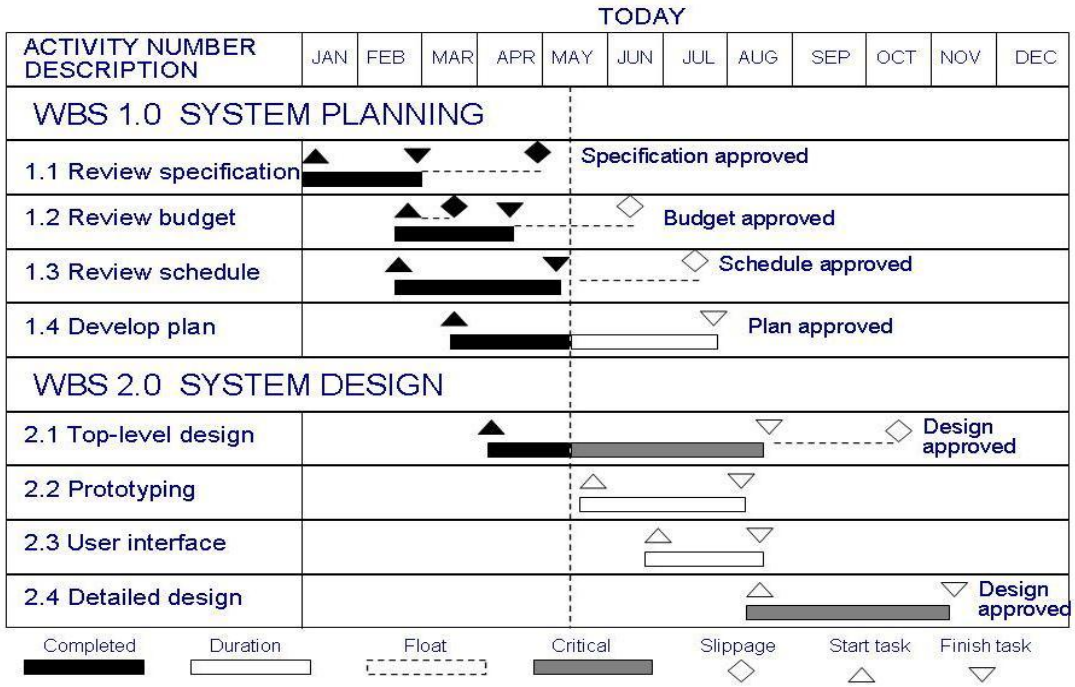
Tools to Track Progress

- Example: to track progress of building a communication software



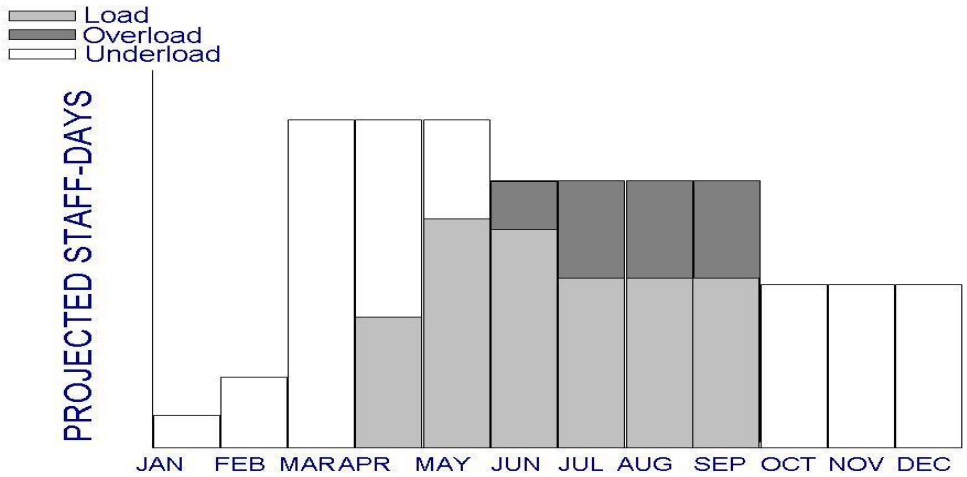
Tools to Track Progress: Gantt Chart

- Activities shown in parallel
  - helps understand which activities can be performed concurrently



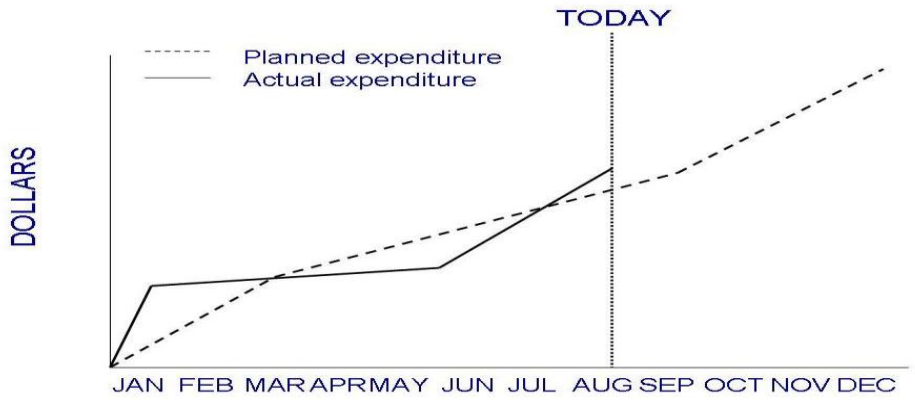
Tools to Track Progress: Resource Histogram

- Shows people assigned to the project and those needed for each stage of development



Tools to Track Progress: Expenditures Tracking

- An example of how expenditures can be monitored



Project Personnel

- Key activities requiring personnel
  - requirements analysis
  - system design
  - program design
  - program implementation
  - testing
  - training
  - maintenance
  - quality assurance
- There is great advantage in assigning different responsibilities to different people

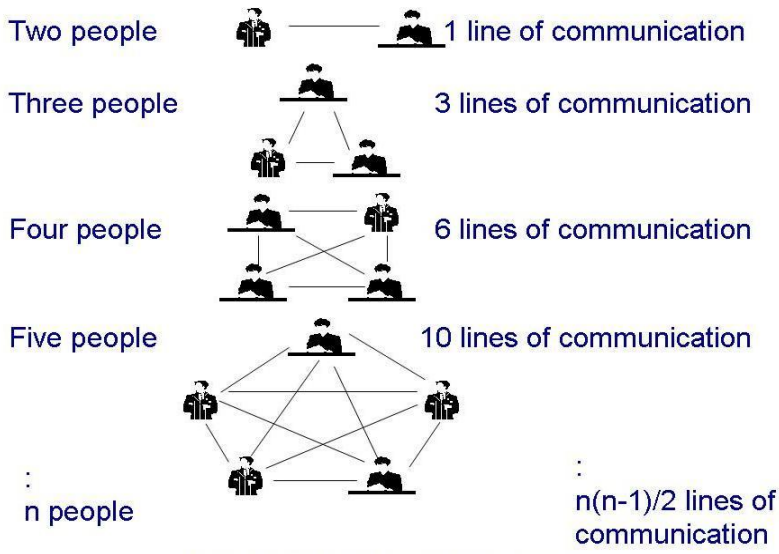
Choosing Personnel:

- Ability to perform work
- Interest in work
- Experience with
  - similar applications
  - similar tools, languages, or techniques
  - similar development environments
- Training

- Ability to communicate with others
- Ability to share responsibility
- Management skills

Communication:

- A project's progress is affected by
  - degree of communication
  - ability of individuals to communicate their ideas
- Software failures can result from breakdown in communication and understanding
- Line of communication can grow quickly
- If there is *n* worker in project, then there are *n(n-1)/2* pairs of communication



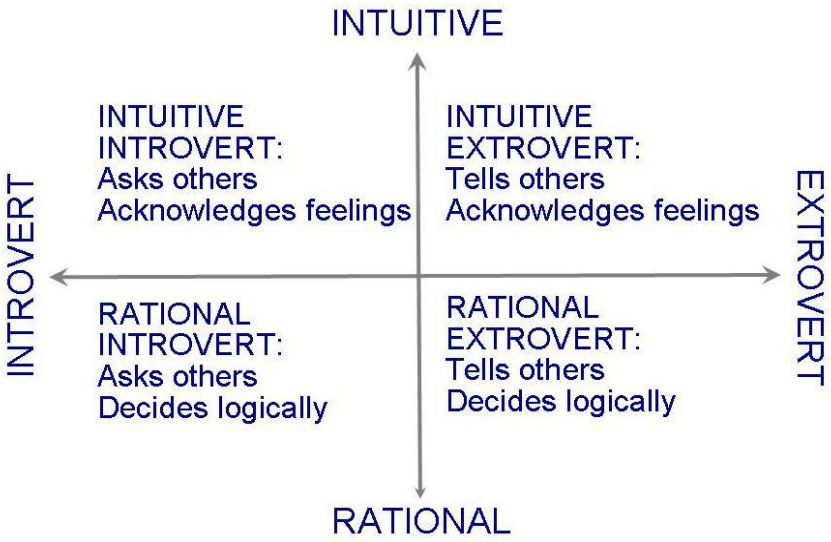
Make Meeting Enhance Project Progress:

- Common complains about meeting
  - the purpose is unclear
  - the attendees are unprepared
  - essential people are late or absent
  - the conversation veers away from its purpose
  - participants do not discuss, instead argue
  - decisions are never enacted afterward
- Ways to ensure a productive meeting

- clearly decide who should be in the meeting
- develop an agenda
- have someone who tracks the discussion
- have someone who ensures follow-up actions

Work Styles:

- Extroverts: tell their thoughts
- Introverts: ask for suggestions
- Intuitives: base decisions on feelings
- Rationals: base decisions on facts, options
- Horizontal axis: communication styles
- Vertical axis: decision styles



- Work styles determine communication styles
- Understanding workstyles
  - help to be flexible
  - give information based on other's priorities
- Impacts interaction among customers, developers and users

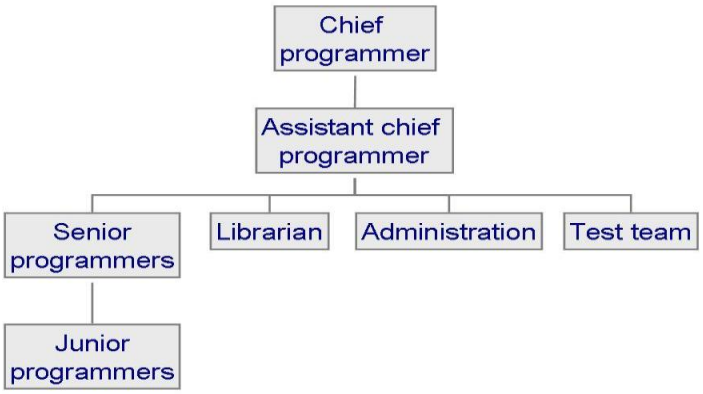
Project Organization:

- Depends on

  - backgrounds and work styles of team members
  - number of people on team
  - management styles of customers and developers
- Examples:
  - *Chief programmer team*: one person totally responsible for a system's design and development
  - *Egoless approach: hold everyone equally responsible*

Project Organization: Chief Programmer Team:

- Each team member must communicate often with chief, but not necessarily with other team members



- Characteristics of projects and the suggested organizational structure to address them

Highly structured	Loosely structured
High certainty	Uncertainty
Repetition	New techniques or technology
Large projects	Small projects

Structure vs. Creativity:

- Experiment by Sally Phillip examining two groups building a hotel
  - structured team: clearly defined responsibilities
  - unstructured team: no directions

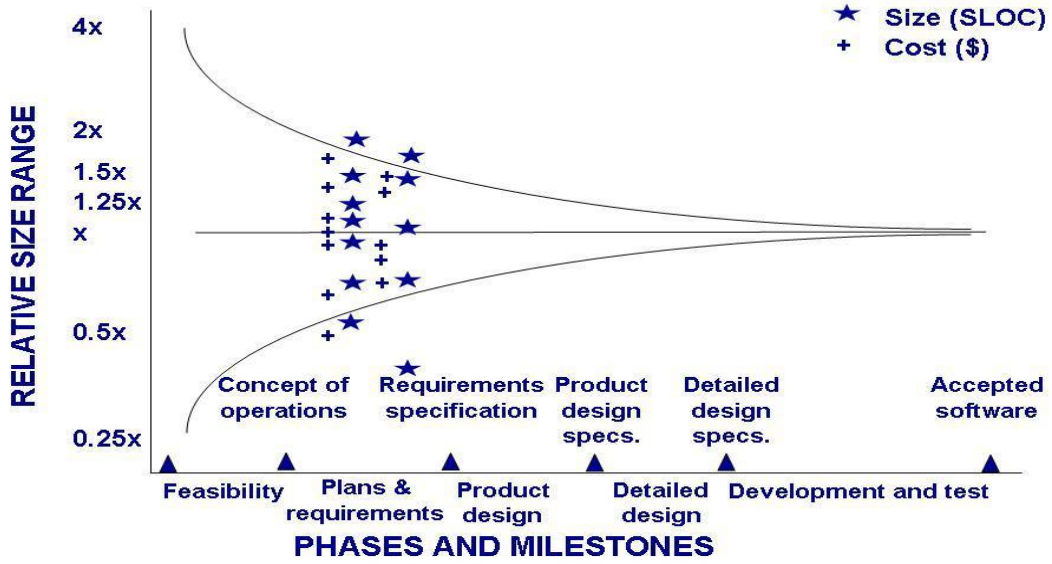
- The results are always the same
  - Structured teams *finish* a functional Days Inn
  - Unstructured teams build a creative, multistoried Taj Mahal and never complete
- Good project management means finding a balance between structure and creativity

Effort Estimation

- Estimating project costs is one of the crucial aspects of project planning and management
- Estimating cost has to be done as early as possible during the project life cycle
- Type of costs
  - facilities: hardware, space, furniture, telephone, etc
  - software tools for designing software
  - staff (effort): the biggest component of cost

Estimation Should be Done Repeatedly:

- Uncertainty early in the project can affect the accuracy of cost and size estimations



Causes of Inaccurate Estimates:

- **Key causes**
  - Frequent request for change by users

- Overlooked tasks
- User's lack of understanding of the requirements
- Insufficient analysis when developing estimates
- Lack of coordination of system development, technical services, operations, data administration, and other functions during development
- Lack of an adequate method or guidelines for estimating

- **Key influences**
  - Complexity of the proposed application system
  - Required integration with existing system
  - Complexity of the program in the system
  - Size of the system expressed as number of functions or programs
  - Capabilities of the project team members
  - Project team's experience with the application, the programming language, and hardware
  - Capabilities of the project team members
  - Database management system
  - Number of project team member
  - Extent of programming and documentation standards

Type of Estimation Methods:

- Expert judgment
  - Top-down or bottom-up
  - Analogy: pessimistic *(x),* optimistic *(y),* most likely *(z);* estimate as *(x + 4y + z)/6*
  - Delphi technique: based on the average of “secret” expert judgments
- Algorithmic methods: $E = (a + bS^c)\, m(\mathbf{X})$
  - Walston and Felix model: $E = 5.25\, S^{0.91}$
  - Bailey and Basili model: $E = 5.5 + 0.73\, S^{1.16}$

Expert Judgement: Wolverton Model:

- Two factors that affect difficulty
  - whether problem is old (O) or new (N)
  - whether it is easy (E) or moderate (M)

	Difficulty					
Type of software	*OE*	*OM*	*OH*	*NE*	*NM*	*NH*
Control	21	27	30	33	40	49
Input/output	17	24	27	28	35	43
Pre/post processor	16	23	26	28	34	42
Algorithm	15	20	22	25	30	35
Data management	24	31	35	37	46	57
Time-critical	75	75	75	75	75	75

Algorithmic Method: Watson and Felix Model:

- A productivity index is inlcuded in the equation
- There are 29 factors that can affect productivity
  - 1 if increase the productivity
  - 0 if decrease the productivity

Watson and Felix Model Productivity Factors:

1. Customer interface complexity	16. Use of design and code inspections
2. User participation in requirements definition	17. Use of top-down development
3. Customer-originated program design changes	18. Use of a chief programmer team
4. Customer experience with the application area	19. Overall complexity of code
5. Overall personnel experience	20. Complexity of application processing
6. Percentage of development programmers who participated in the design of functional specifications	21. Complexity of program flow
7. Previous experience with the operational computer	22. Overall constraints on program's design
8. Previous experience with the programming language	23. Design constraints on the program's main storage
9. Previous experience with applications of similar size and complexity	24. Design constraints on the program's timing
10. Ratio of average staff size to project duration (people per month)	25. Code for real-time or interactive operation or for execution under severe time constraints
11. Hardware under concurrent development	26. Percentage of code for delivery
12. Access to development computer open under special request	27. Code classified as nonmathematical application and input/output formatting programs
13. Access to development computer closed	28. Number of classes of items in the database per 1000 lines of code
14. Classified security environment for computer and at least 25% of programs and data	29. Number of pages of delivered documentation per 1000 lines of code
15. Use of structured programming	

Software Cost Components

- Machine and tool costs
- Training and travel
- Efforts
  - Engineer salaries, benefits, and overhead
    - Insurances
    - Office, phone and networking, shared facilities and supporting

Estimation Techniques

- Professional or expert judgment
  - By experts with experiences in developing software in the application domain
- Past project experiences
  - Comparing to similar project in cost datsbase
- Algorithmic cost modeling
  - Function point analysis and object point
  - COCOMO and COCOMO 2

Expert Judgement: Wolverton Model

- Two factors that affect difficulty
  - whether problem is old (O) or new (N)
  - whether it is easy (E) or moderate (M)

	Difficulty					
Type of software	*OE*	*OM*	*OH*	*NE*	*NM*	*NH*
Control	21	27	30	33	40	49
Input/output	17	24	27	28	35	43
Pre/post processor	16	23	26	28	34	42
Algorithm	15	20	22	25	30	35
Data management	24	31	35	37	46	57
Time-critical	75	75	75	75	75	75

Effort Estimation

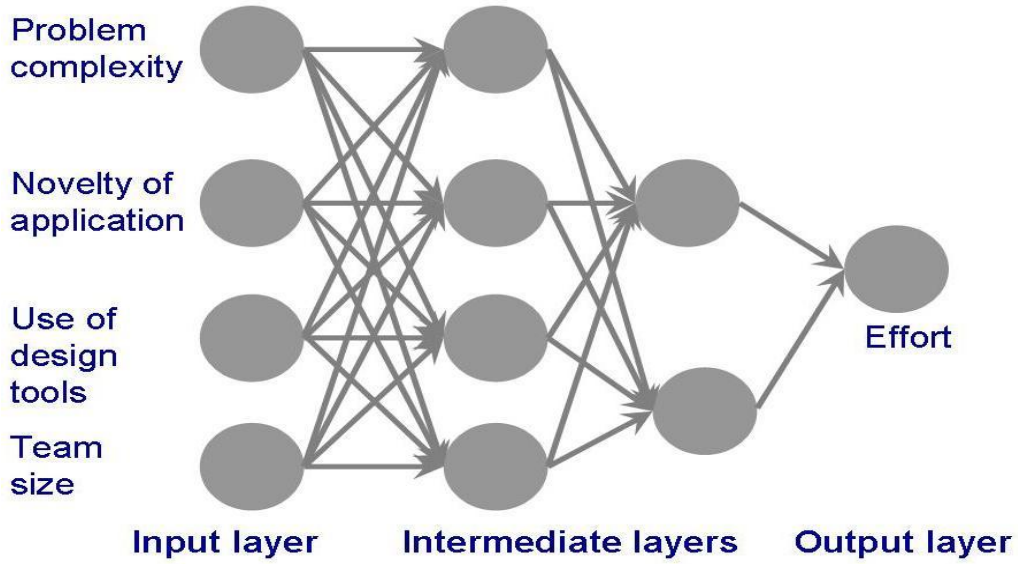
Machine Learning Techniques

- Example: case-based reasoning (CBR)
  - user identifies new problem as a case
  - system retrieves similar cases from repository

  - system reuses knowledge from previous cases
  - system suggests solution for new case

- Example: neural network
  - cause-effect network “trained” with data from past history

Machine learning techniques: Neural Network



Machine Learning Techniques: CBR

- Involves four steps
  - the user identifies a new problem as a case
  - the system retrieves similar case from a respository of historical information
  - the system reuses knowledge from previous case
  - the system suggests a solution for the new case
- Two big hurdles in creating successful CBR system
  - characterizing cases
  - determining similarity

FPA – Function Point Analysis

- Analyze the project documentation to obtain a list of logical files
- Identify the boundary between the application and its external environment, and classify logical files into
  - Internal logical files
  - External interface files
- Count number of data element types, record element types, and file types
- Compute the function points of all logical files and add them to produce unadjusted function points

Function Point Calculation

- Based on a combination of program size and technical complexity
  - Internal logical files
  - external interface files
  - External inputs
  - External outputs
  - External inquiries
- A weight is associated with each characteristics
  - Low (e.g., 7)
  - Average (e.g., 10)
  - High (e.g., 15)
- The unadjusted function point (UFP) is computed by multiplying each raw count with the weight and summing all values
- The UFP is multiplied by a complexity adjustment factor
  - The adjustment factor is between 0.65 and 1.35

  - Computed from general system characteristics (GSC) to assess the environment and processing complexity

- Function point is computed in two steps.
- The first step is to compute the unadjusted function point (UFP).
- **UFP = (Number of inputs)*4 + (Number of outputs)*5 + (Number of inquiries)*4 + (Number of files)*10 + (Number of interfaces)*10**

Number of inputs: Each data item input by the user is counted.

Number of outputs: The outputs considered refer to reports printed, screen outputs, error messages produced, etc.

Number of inquiries: Number of inquiries is the number of distinct interactive queries which can be made by the users i.e User Commands

Number of files: Each logical file is counted. A logical file means groups of logically related data. Thus, logical files can be data structures or physical files.

Number of interfaces: Here the interfaces considered are the interfaces used to exchange information with other syste

Function Point Calculation

Functional type	Count	Scale	Weight	Sum
Internal logical files	3	Average	10	30
External interface files	0	Average	7	0
External inputs	2	Average	4	8
External output	2	Average	5	10
External inquiries	5	Average	4	20
Count total				68
Complexity				0.77
Functional point				52

Usage of Function Point

- FPs can be used to estimate LOC (Lines of Code) or efforts (in man-month)
  - Language dependent: different languages have different LOC or man-month per FP
  - LOC = AVC * number of function points
  - AVC is a language-dependent factor varying from 200-300 for assemble language to 2-40 for a 4GL
- FPs are very subjective and dependent on the estimator
  - Difficult to have automatic function-point counting
- Work-effort = 0.2 * TDET + 2.71 * TGRE + 79
  - Work-effort: person-days
  - TDET: number of data elements
    - Unique user identifiable field on an internal logical file or external interface file (e.g., var or field in structure)
  - TGRE: number of data element groups
    - Unique user identifiable subgroup of data elements within an internal logical file or external interface file (e.g., structure, union, or other record type)

Object Point

- Object points are an alternative to function point when 4GLs or object-oriented languages are used for development
- Object points are not number of object classes
- The number of object points in a program is a weighted estimate of
  - The number of separate screens that are displayed
  - The number of reports that are produced by the system
  - The number of code modules that must be developed
- Object points are easier to estimate from a specification than function points as they are simply concerned with screens, reports and code modules

- They can be estimated at an early point in the development process.
- Can be used for web-based development

Constructive Cost Model

- An empirical model based on database of different projects
- History
  - COCOMO (original COCOMO model)
    - Classify project into three classes
    - Basic, intermediate and advanced stages
    - Estimate project efforts, duration and staff from lines of code
  - COCOMO 2
    - Three levels: different development stage
    - Takes into account on development approaches, component reuse, etc
    - More complicated

Project Classes in COCOMO

- **Organic mode**:
  - Small teams, in familiar environment, well-understood applications, no difficult non-functional requirements
  - Examples: most simple data processing
- **Semi-detached mode**
  - Project team may have experience mixture, system may have more significant non-functional constraints, organization may have less familiarity with application
  - Examples: transaction processing, distributed data collection and monitoring
- **Embedded**

- Hardware/software systems, tight constraints, highly distributed and networking requirements
- Examples: real-time systems, embedded software

COCOMO Stages

- **Basic**
  - Gives an estimate based on product attributes
- **Intermediate**
  - Modifies basic estimate using project and process attributes
- **Advanced**
  - Estimates project phases and parts separately

COCOMO Estimates

- **Primary parameter**
  - Lines of code
  - KDSI (thousands of delivered source instructions)
- **Estimates**
  - Project efforts: in terms of man-month
  - Project duration
  - Staffing requirements

Basic COCOMO Formula

- **Organic mode**
  - **Man-month (PM) = 2.4 (KDSI) $^{1.05}$**
- **Semi-detached mode**
  - **Man-month (PM) = 3 (KDSI) $^{1.12}$**
- **Embedded mode**
  - **Man-month (PM) = 3.6 (KDSI) $^{1.2}$**

Project Duration and Staffing

- Organic
  - Calendar time (TDEV) = 2.5 (PM) $^{0.38}$
- Semi-detached
  - Calendar time (TDEV) = 2.5 (PM) $^{0.35}$
- Embedded mode
  - Calendar time (TDEV) = 2.5 (PM) $^{0.32}$
- Personnel requirement: N = PM/TDEV

Example:

Organic mode project, 44 KLOC

- PM = 2.4 (44) $^{1.05}$ = 128 person months
- TDEV = 2.5 (128) $^{0.38}$ = 16 months
- N = 128/16 = 8 people

- **Embedded mode project, 44 KLOC**
  - PM = 3.6 $(44)^{1.2}$ = 338 person-months
  - TDEV = 2.5 $(338)^{0.32}$ = 16 months
  - N = 338/16 = 21 people

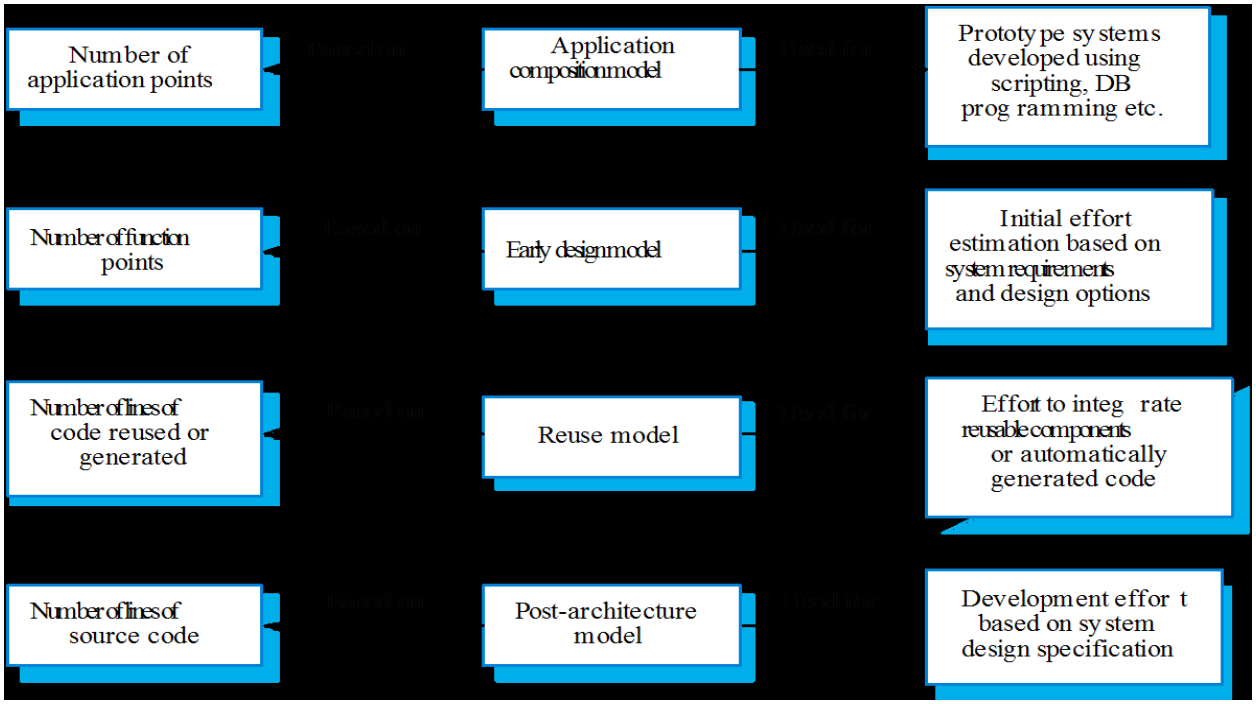
COCOMO 2

- COCOMO 2 consists of three models that allow increasingly detailed estimates to be made as development progresses
  - Application composition (or early prototyping) model
    - Used during the early stages of a project
    - Estimates based on object points and a simple formula is used for effort estimation
  - Early design model

- Used when requirements are stabilized
- Estimates based on function points that are translated to LOC

– Post-architecture model

- Used during development stages
- Estimates based on lines of source code



Early Prototyping Model:

- Used during the early stages of a project
- To estimate the efforts for prototyping or for projects that is composed of components
- Based on weighted object points
- Formula

– PM = (NOP ´ (1 - % of reuse/100)) / PROD

- PM is the effort in person-months
- NOP is the number of object points
- PROD is the productivity

Early Design Model

- Estimates is made after the requirements have been stabilized
- Based on standard algorithmic models
  - $PM = A \times Size^{B} \times M + PM_m$
    - $M = PERS \times RCPX \times RUSE \times PDIF \times PREX \times FCIL \times SCED$
    - $PM_m = (ASLOC \times (AT/100)) / ATPROD$
    - A = 2.5 in initial calibration
    - Size in KLOC
    - B ranges from 1.1 to 1.24 depending on novelty of the project, development flexibility, risk management approaches and the process maturity

Multipliers

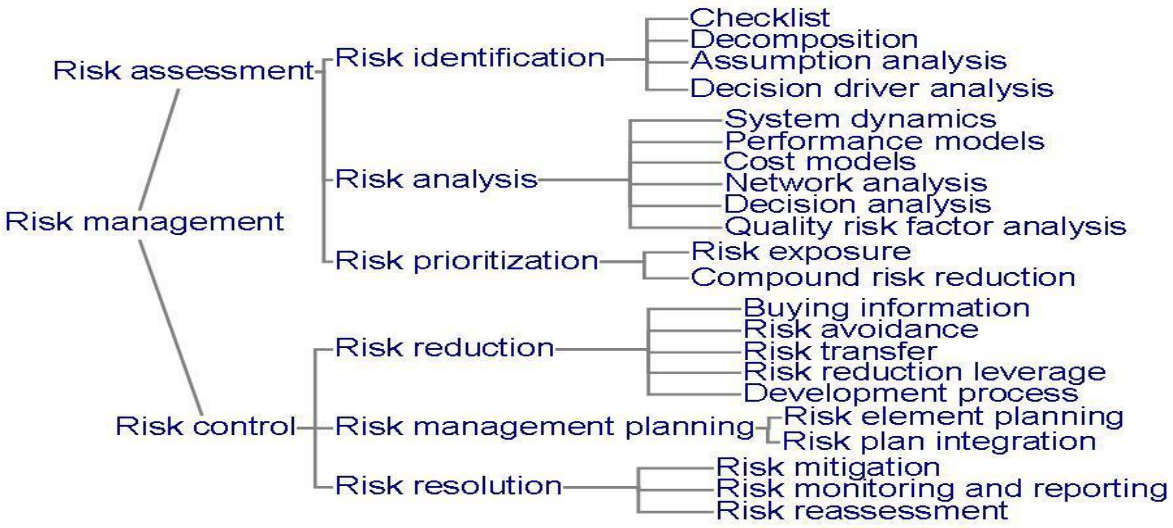
- M represents the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.
  - RCPX - product reliability and complexity
  - RUSE – required reusability
  - PDIF - platform difficulty
  - PREX - personnel experience
  - PERS - personnel capability
  - SCED – required development schedule
  - FCIL - the team support facilities
- PM_m reflects the amount of automatically generated code
  - ASLOC: number of automatically generated lines of source code
  - ASPROD: productivity level for this type of code production
  - AT: percentage of total system code that is automatically generated

Risk Management

What is a Risk?

- Risk is an unwanted event that has negative consequences
- Distinguish risks from other project events
  - ***Risk impact****:* the loss associated with the event
  - ***Risk probability***: the likelihood that the event will occur
- Quantify the effect of risks
  - *Risk exposure* = (risk probability) x (risk impact)
- Risk sources: generic and project-specific

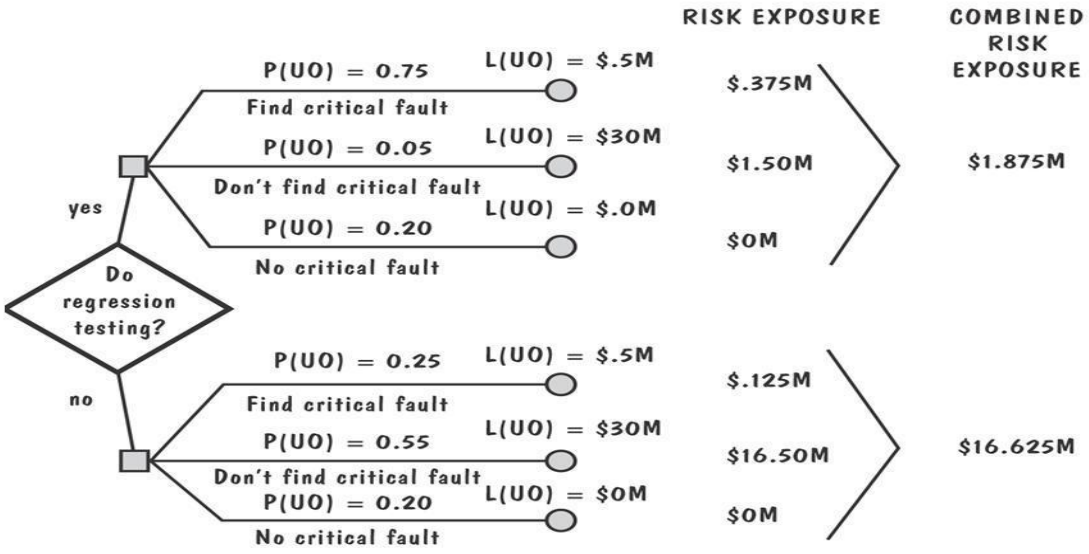
Risk Management Activities:



- Example of risk exposure calculation

PU: prob. of unwanted outcome

LU: lost assoc with unwanted outcome



- Three strategies for risk reduction
  - *Avoiding the risk*: change requirements for performance or functionality
  - *Transferring the risk*: transfer to other system, or buy insurance
  - *Assuming the risk*: accept and control it
- Cost of reducing risk
  - *Risk leverage* = (risk exposure before reduction – (risk exposure after reduction) / (cost of risk reduction)

Boehm's Top Ten Risk Items:

- Personnel shortfalls
- Unrealistic schedules and budgets
- Developing the wrong functions
- Developing the wrong user interfaces
- Gold-plating
- Continuing stream of requirements changes
- Shortfalls in externally-performed tasks
- Shortfalls in externally-furnished components
- Real-time performance shortfalls
- Straining computer science capabilities

Project Plan

Project Plan Contents:

- Project scope
- Project schedule
- Project team organization
- Technical description of system
- Project standards and procedures
- Quality assurance plan
- Configuration management plan
- Documentation plan
- Data management plan
- Resource management plan
- Test plan
- Training plan
- Security plan
- Risk management plan
- Maintenance plan

Project Plan Lists:

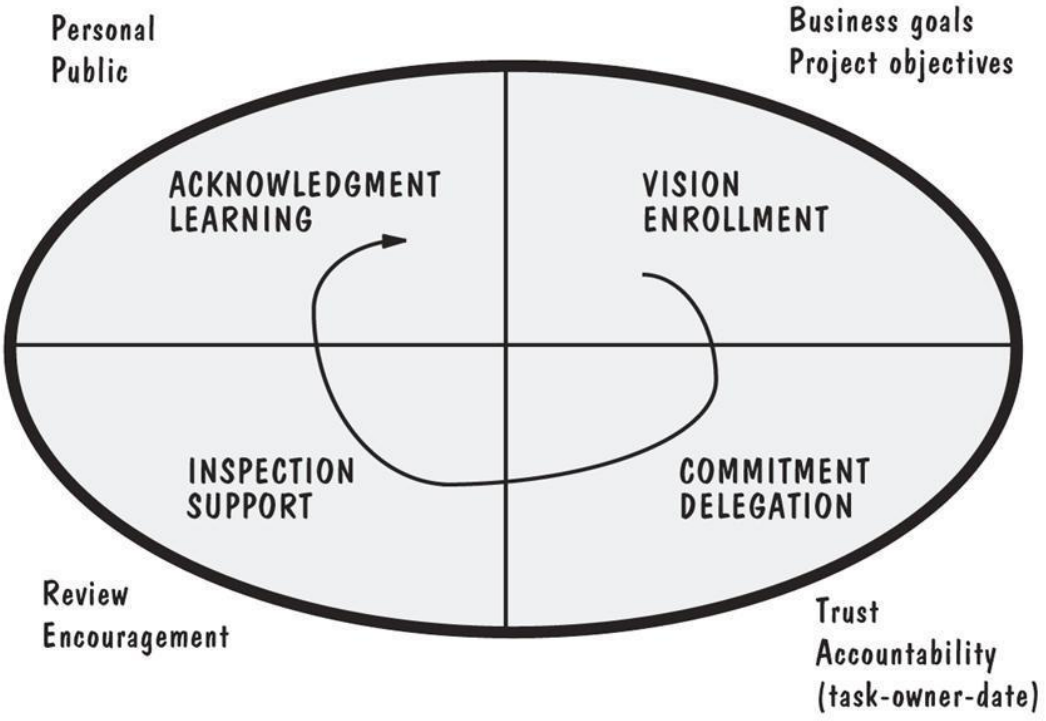
- List of the people in development team
- List of hardware and software
- Standards and methods, such as
  - algorithms
  - tools
  - review or inspection techniques

- design language or representaions
- coding languages
- testing techniques

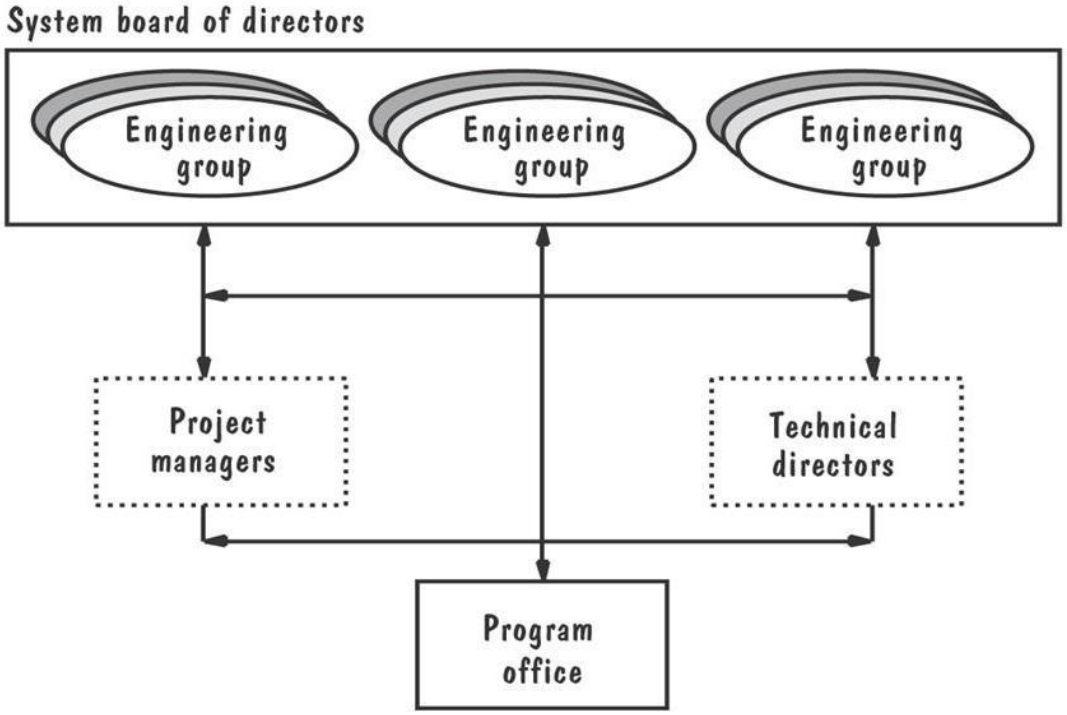
Process Models and Project Management

Enrollment Management Model: Digital Alpha AXP:

- Establish an appropriately large shared vision
- Delegate completely and elicit specific commitments from participants
- Inspect vigorously and provide supportive feedback
- Acknowledge every advance and learn as the program progresses
- Vision: to “enroll” the related programs, so they all shared common goals

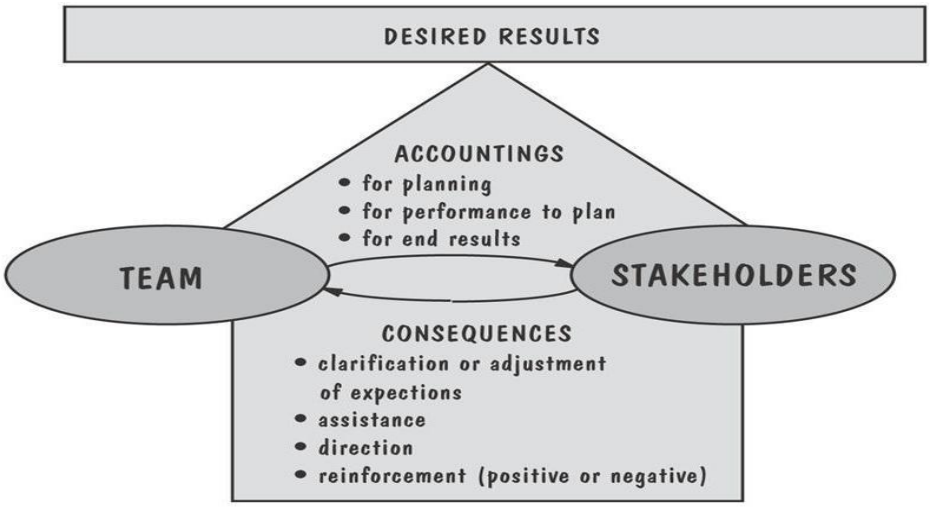


- An organization that allowed technical focus and project focus to contribute to the overall program

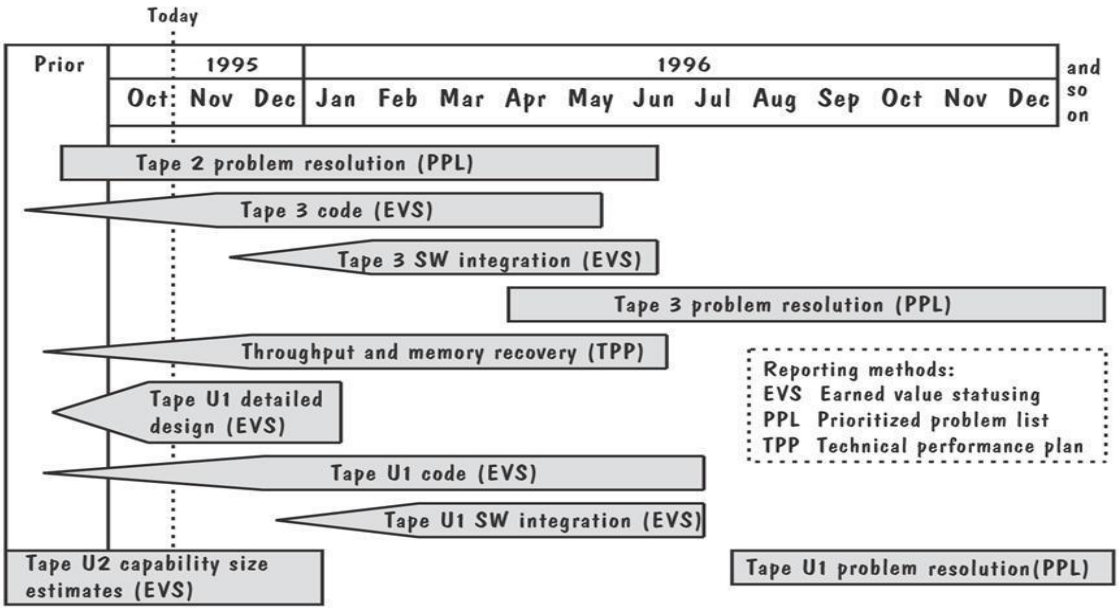


Accountability modeling: Lockheed Martin:

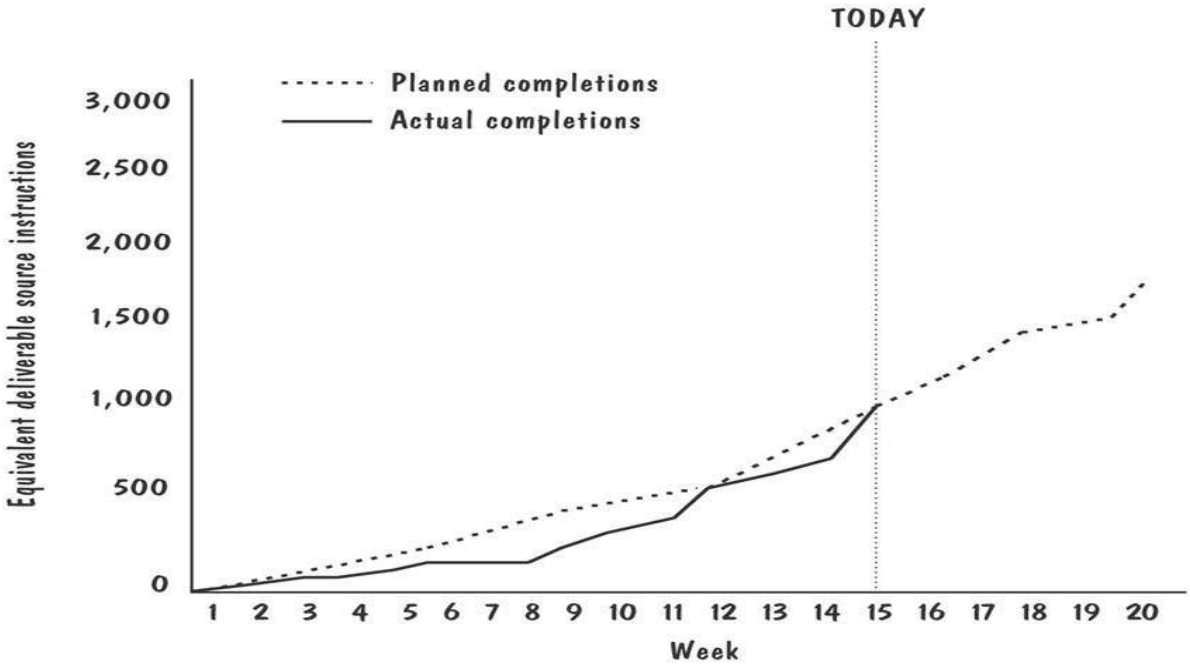
- Matrix organization
  - Each engineer belongs to a functional unit based on type of skill
- Integrated product development team
  - Combines people from different functional units into interdisciplinary work unit
- Each activity tracked using cost estimation, critical path analysis, schedule tracking
  - Earned value a common measure for progress
- Accountability model used in F-16 Project



- Teams had multiple, overlapping activities
- An activity map used to illustrate progress on each activity

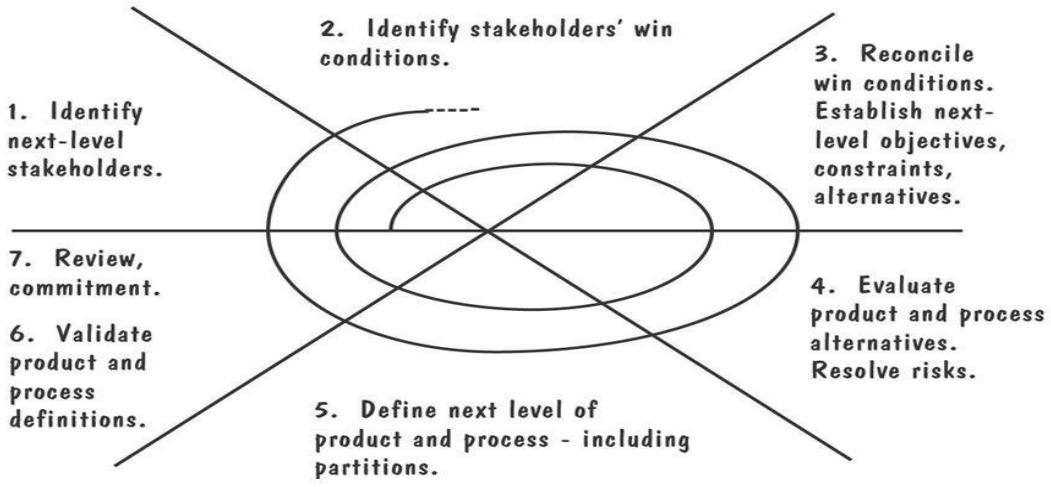


- Each activitiy's progress was tracked using earned value chart



Anchoring (Common) Milestones:

- Life cycle objectives
    - **Objectives**: Why is the system being developed?
    - **Milestones and schedules**: What will be done by when?
    - **Responsibilities**: Who is responsible for a function?
    - **Approach**: How will the job be done, technically and managerially?
    - **Resources**: How much of each resource is needed?
    - **Feasibility**: Can this be done, and is there a good business reason for doing it?
- Life-cycle architecture: define the system and software architectures and address architectural choices and risks
- Initial operational capability: readiness of software, deployment site, user training
- The Win-Win spiral model suggested by Boehm is used as supplement to the milestones



Information System Example

Piccadilly System:

- Using COCOMO II

- Three screens and one report
  - Booking screen: complexity simple, weight 1
  - Ratecard screen: complexity simple, weigth 1
  - Availability screen: complexity medium, weight 2
  - Sales report: complexity medium, weight 5
- Estimated effort = 182 person-month

Ariane-5 System:

- The Ariane-5 destruction might have been prevented had the project managers developed a risk management plan
  - Risk identification: possible problem with reuse of the Ariane-4)
  - Risk exposure: prioritization would have identified if the inertial reference system (SRI) did not work as planned
  - Risk control: assesment of the risk using reuse software
  - Risk avoidance: using SRI with two different designs

PART - II

The Problems with our Requirements Practices

- We have trouble understanding the requirements that we do acquire from the customer
- We often record requirements in a disorganized manner
- We spend far too little time verifying what we do record
- We allow change to control us, rather than establishing mechanisms to control change
- Most importantly, we fail to establish a solid foundation for the system or software that the user wants built
- Many software developers argue that
  - Building software is so compelling that we want to jump right in (before having a clear understanding of what is needed)

    - Things will become clear as we build the software
    - Project stakeholders will be able to better understand what they need only after examining early iterations of the software
    - Things change so rapidly that requirements engineering is a waste of time
    - The bottom line is producing a working program and that all else is secondary
- All of these arguments contain some truth, especially for small projects that take less than one month to complete
- However, as software grows in size and complexity, these arguments begin to break down and can lead to a failed software project

A Solution: Requirements Engineering

- Begins during the communication activity and continues into the modeling activity
- Builds a bridge from the system requirements into software design and construction
- Allows the requirements engineer to examine
    - the context of the software work to be performed
    - the specific needs that design and construction must address
    - the priorities that guide the order in which work is to be completed
    - the information, function, and behavior that will have a profound impact on the resultant design

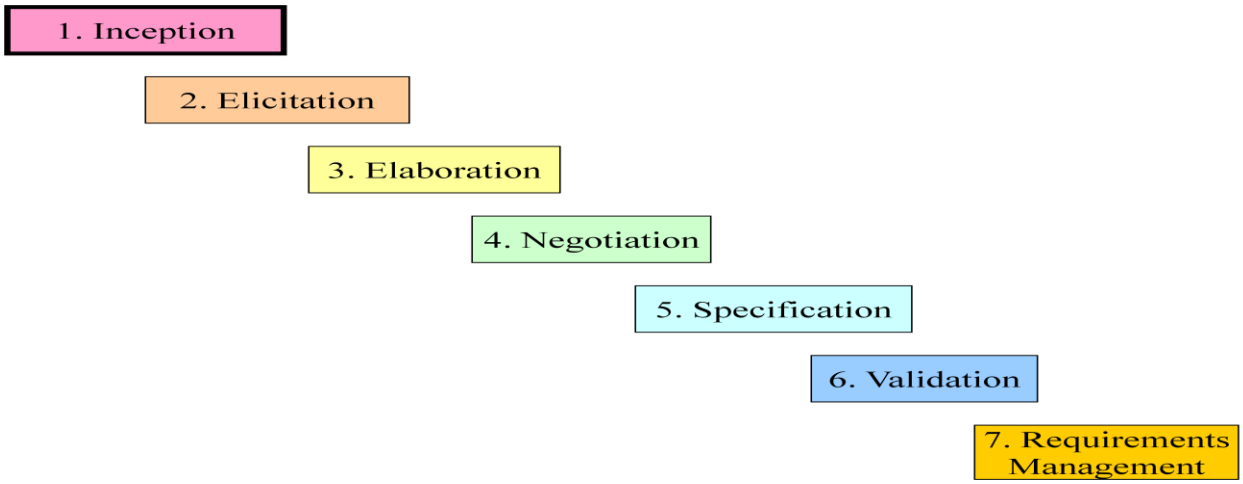
Requirements engineering tasks

Seven distinct tasks

- **Inception**
- **Elicitation**
- **Elaboration**
- **Negotiation**
- **Specification**
- **Validation**
- **Requirements management**

- Some of these tasks may occur in parallel and all are adapted to the needs of the project

- All strive to define what the customer wants
- All serve to establish a solid foundation for the design and construction of the software



1. Inception Task

- During inception, the requirements engineer asks a set of context free questions to establish…
    a. A basic understanding of the problem
    b. The people who want a solution
    c. The nature of the solution that is desired
    d. The effectiveness of preliminary communication and collaboration between the customer and the developer
- Through these questions, the requirements engineer needs to…
    e. Identify the stakeholders
    f. Recognize multiple viewpoints
    g. Work toward collaboration
    h. Break the ice and initiate the communication

➢ **The First Set of Questions**

These questions focus on the customer, other stakeholders, the overall goals, and the benefits

- 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?

➢ **The Next Set of Questions**

These questions enable the requirements engineer to gain a better understanding of the problem and allow the customer to voice his or her perceptions about a solution

- How would you characterize "good" output that would be generated by a successful solution?
- What problem(s) will this solution address?
- Can you show me (or describe) the business environment in which the solution will be used?
- Will special performance issues or constraints affect the way the solution is approached?

➢ **The Final Set of Questions**

These questions focus on the effectiveness of the communication activity itself

- Are you the right person to answer these questions? Are your answers "official"?
- Are my questions relevant to the problem that you have?
- Am I asking too many questions?
- Can anyone else provide additional information?
- Should I be asking you anything else?

2. Elicitation Task

- Eliciting requirements is difficult because of
  - Problems of scope in identifying the boundaries of the system or specifying too much technical detail rather than overall system objectives
  - Problems of understanding what is wanted, what the problem domain is, and what the computing environment can handle (Information that is believed to be "obvious" is often omitted)
  - Problems of volatility because the requirements change over time
- **Elicitation may be accomplished through two activities**

- **Collaborative requirements gathering**
- **Quality function deployment**

➢ Basic Guidelines of Collaborative Requirements Gathering

A collaborative team-oriented approach to requirements gathering encourage a team of stakeholders and developers work together to identify the problem, propose elements of the solution, negotiate different approaches and specify set of solutions.

Different approaches to collaborative RG:

- Meetings are conducted and attended by both software engineers, customers, and other interested stakeholders
- Rules for preparation and participation are established
- An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas
- A "facilitator" (customer, developer, or outsider) controls the meeting
- A "definition mechanism" is used such as work sheets, flip charts, wall stickers, electronic bulletin board, chat room, or some other virtual forum
- The goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements

➢ Quality Function Deployment

- This is a technique that translates the needs of the customer into technical requirements for software
- It emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the engineering process through functions, information, and tasks
- It identifies three types of requirements
    - Normal requirements: These requirements are the objectives and goals stated for a product or system during meetings with the customer

- Expected requirements: These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them
- Exciting requirements: These requirements are for features that go beyond the customer's expectations and prove to be very satisfying when present

Elicitation Work Products

The work products will vary depending on the system, but should include one or more of the following items

- 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
- A list of requirements (organized by function) and the domain constraints that apply to each
- A set of preliminary usage scenarios (in the form of use cases) that provide insight into the use of the system or product under different operating conditions
- Any prototypes developed to better define requirements

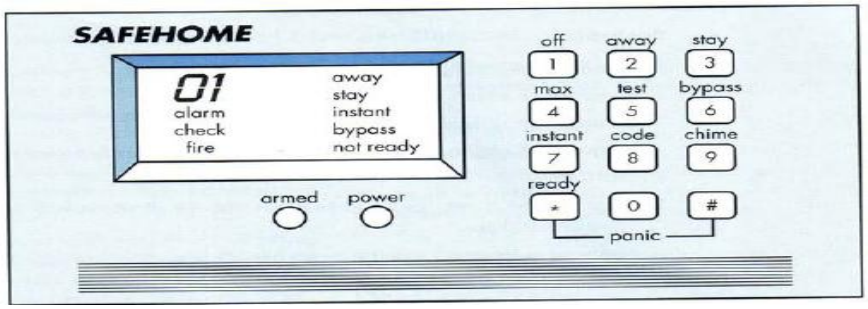
3. Elaboration Task

- During elaboration, the software engineer takes the information obtained during inception and elicitation and begins to expand and refine it
- Elaboration focuses on developing a refined technical model of software functions, features, and constraints
- It is an analysis modeling task
  - Use cases are developed
  - Domain classes are identified along with their attributes and relationships
  - State machine diagrams are used to capture the life on an object
- The end result is an analysis model that defines the functional, informational, and behavioral domains of the problem

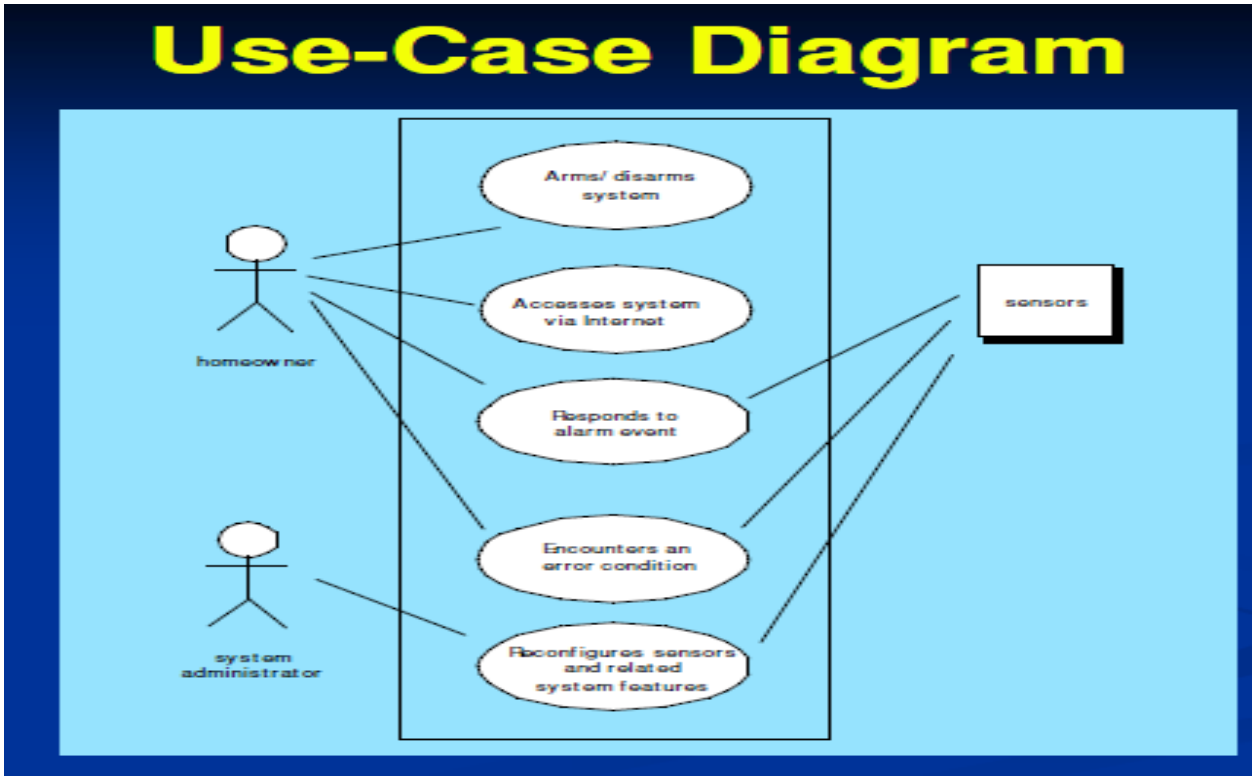
➢ **Developing Use Cases**

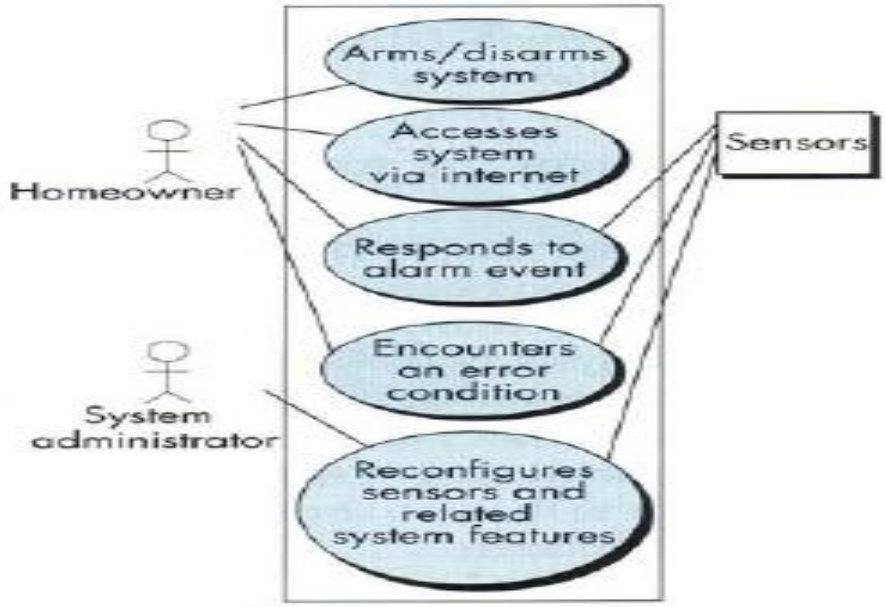
- Step One – Define the set of actors that will be involved in the story
    - Actors are people, devices, or other systems that use the system or product within the context of the function and behavior that is to be described
    - Actors are anything that communicate with the system or product and that are external to the system itself
- Step Two – Develop use cases, where each one answers a set of questions

Eg: Safehome Control Panel



1. The homeowner observes the *SafeHome* control panel (Figure 7.2) to determine if the system is ready for input. If the system is not ready a *not ready* message is displayed on the LCD display, and the homeowner must physically close windows/doors so that the *not ready* message disappears. (A *not ready* message implies that a sensor is open; i.e., that a door or window is open.)
2. The homeowner uses the keypad to key in a four-digit password. The password is compared with the valid password stored in the system. If the password is incorrect, the control panel will beep once and reset itself for additional input. If the password is correct, the control panel awaits further action.
3. The homeowner selects and keys in *stay* or *away* (see Figure 7.2) to activate the system. Stay activates only perimeter sensors (inside motion detecting sensors are deactivated). Away activates all sensors.
4. When activation occurs, a red alarm light can be observed by the homeowner.





Questions Commonly Answered by a Use Case

- Who is the primary actor(s), the secondary actor(s)?
- What are the actor's goals?
- What preconditions should exist before the scenario begins?

- What main tasks or functions are performed by the actor?
- What exceptions might be considered as the scenario is described?
- What variations in the actor's interaction are possible?
- What system information will the actor acquire, produce, or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

Elements of the Analysis Model

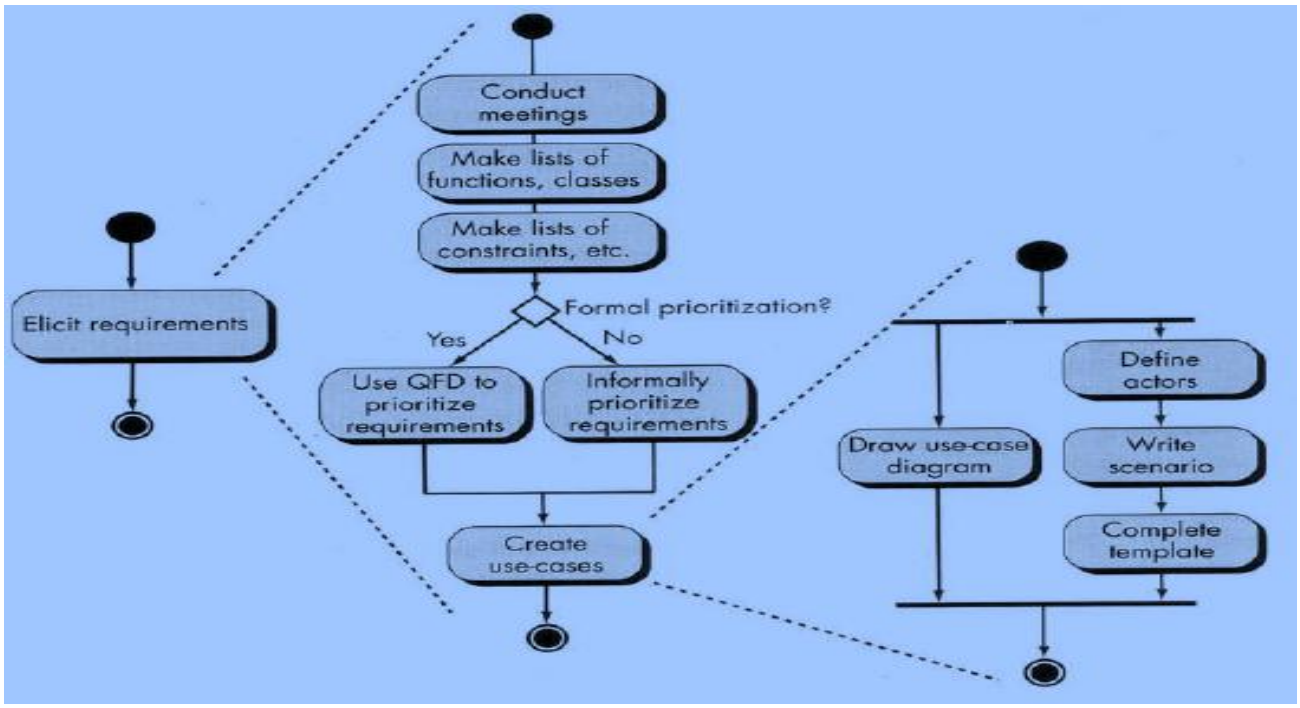
- Scenario-based elements
  - Functional—processing narratives for software functions
  - Use-case—descriptions of the interaction between an "actor" and the system
- Class-based elements
  - Implied by scenarios
- Behavioral elements
  - State diagram
- Flow-oriented elements
  - Data flow diagram

Scenario-based elements

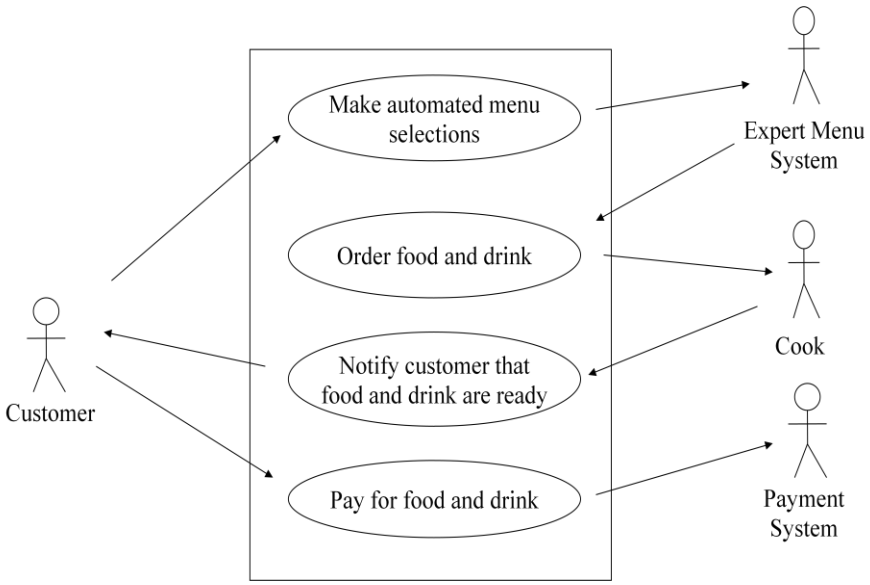
- Describe the system from the user's point of view using scenarios that are depicted in use cases and **Activity diagrams**

Activity diagrams

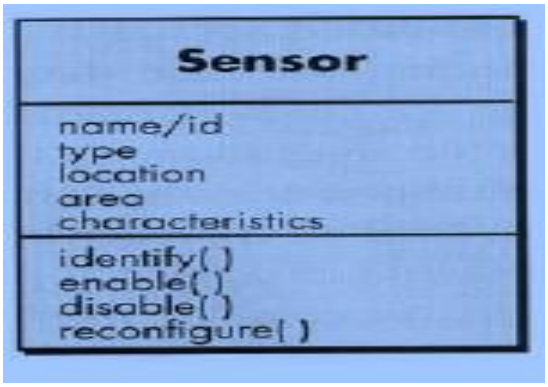
- Supplements the use case by providing a graphical representation of the flow of interaction within a specific scenario
- Uses flowchart-like symbols
  - **Rounded rectangle** - represent a specific system function/action
  - **Arrow** - represents the flow of control from one function/action to another
  - **Diamond** - represents a branching decision
  - **Solid bar –** represents the fork and join of parallel activities



Use-case



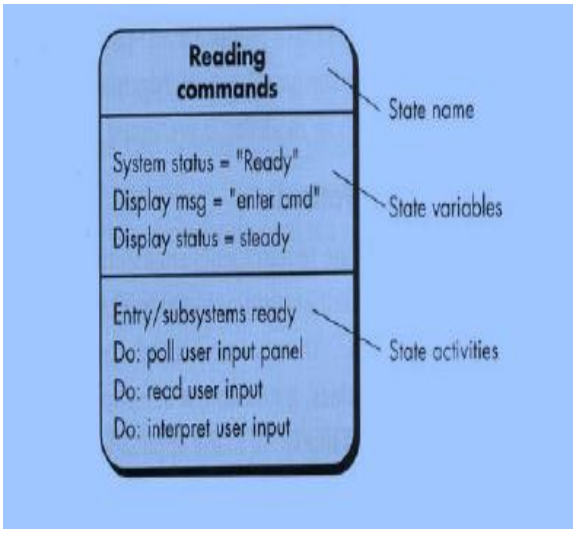
- Class-based elements
- Identify the domain classes for the objects manipulated by the actors, the attributes of these classes, and how they interact with one another; they utilize class diagrams to do this



Behavioral elements

- Use **state diagrams** to represent the state of the system, the events that cause the system to change state, and the actions that are taken as a result of a particular event; can also be applied to each class in the system

State Diagram:



- Flow-oriented elements
  - Use data flow diagrams to show the input data that comes into a system, what functions are applied to that data to do transformations, and what resulting output data are produced

4. **Negotiation Task**

- During negotiation, the software engineer reconciles the conflicts between what the customer wants and what can be achieved given limited business resources
- Requirements are ranked (i.e., prioritized) by the customers, users, and other stakeholders
- Risks associated with each requirement are identified and analyzed

- Rough guesses of development effort are made and used to assess the impact of each requirement on project cost and delivery time
- Using an iterative approach, requirements are eliminated, combined and/or modified so that each party achieves some measure of satisfaction

The Art of Negotiation

- Recognize that it is not competition
- Map out a strategy
- Listen actively
- Focus on the other party's interests
- Don't let it get personal
- Be creative
- Be ready to commit

5. Specification Task

- A specification is the final work product produced by the requirements engineer
- It is normally in the form of a software requirements specification
- It serves as the foundation for subsequent software engineering activities
- It describes the function and performance of a computer-based system and the constraints that will govern its development
- It formalizes the informational, functional, and behavioral requirements of the proposed software in both a graphical and textual format

Typical Contents of a Software Requirements Specification

- Requirements
  - Required states and modes
  - Software requirements grouped by capabilities (i.e., functions, objects)
  - Software external interface requirements
  - Software internal interface requirements
  - Software internal data requirements
  - Other software requirements (safety, security, privacy, environment, hardware, software, communications, quality, personnel, training, logistics, etc.)

  - Design and implementation constraints
- Qualification provisions to ensure each requirement has been met
  - Demonstration, test, analysis, inspection, etc.
- Requirements traceability
  - Trace back to the system or subsystem where each requirement applies

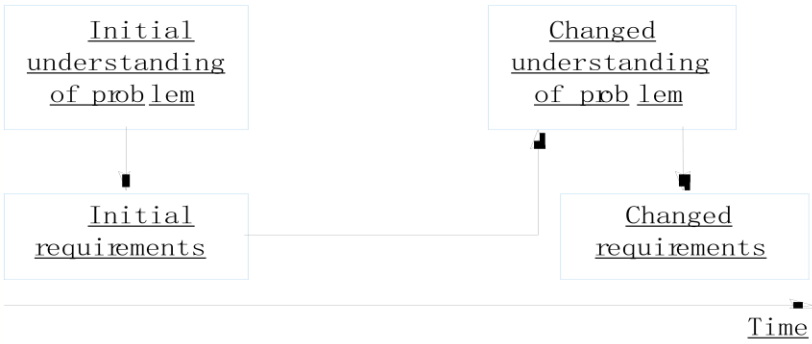
6. Validation Task

- During validation, the work products produced as a result of requirements engineering are assessed for quality
- The specification is examined to ensure that
  - all software requirements have been stated unambiguously
  - inconsistencies, omissions, and errors have been detected and corrected
  - the work products conform to the standards established for the process, the project, and the product
- The formal technical review serves as the primary requirements validation mechanism
  - Members include software engineers, customers, users, and other stakeholders

7. Requirements management

- Requirements are likely to change for large software systems and as such requirements management process is required to handle changes.
- Reasons for requirements changes
- (a) Diverse Users community where users have different requirements and priorities
- (b) System customers and end users are different
- (c) Change in the business and technical environment after installation
- Two classes of requirements
- (a) Enduring requirements: Relatively stable requirements
- (b) Volatile requirements: Likely to change during system development process or during operation

Requirements evolution



Requirements management planning

An essential first stage in requirement management process

- Planning process consists of the following

  1.Requirements identification

  -- Each requirement must have unique tag for cross reference and traceability

  2.Change management process

  -- Set of activities that assess the impact and cost of changes

  3.Traceability policy

  -- A matrix showing links between requirements and other elements of software development

  4.CASE tool support

  --Automatic tool to improve efficiency of change management process. Automated tools are required for requirements storage, change management and traceability management

Requirements Management Task

- During requirements management, the project team performs a set of activities to identify, control, and track requirements and changes to the requirements at any time as the project proceeds
- Each requirement is assigned a unique identifier
- The requirements are then placed into one or more traceability tables
- These tables may be stored in a database that relate features, sources, dependencies, subsystems, and interfaces to the requirements

- A requirements traceability table is also placed at the end of the software requirements specification

Traceability

- Maintains three types of traceability information.

1.Source traceability

--Links the requirements to the stakeholders

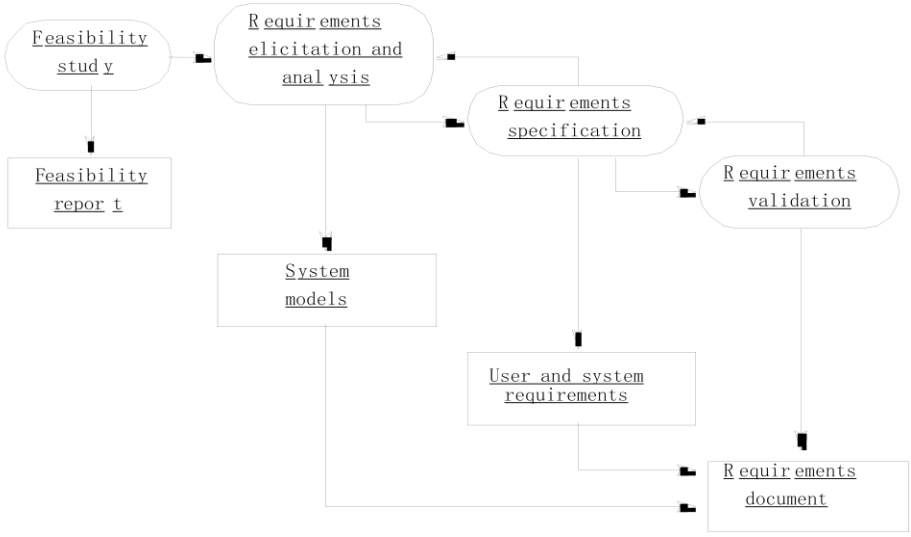
2. Requirements traceability

--Links dependent requirements within the requirements document

3. Design traceability

-- Links from the requirements to the design module

The requirements engineering process



FUNCTIONAL REQUIREMENTS

"A requirement specifies a function that a system or component must be able to perform." Should be both complete and consistent

- Completeness

  -- All services required by the user should be defined

- Consistent

  -- Requirements should not have contradictory definition

- Difficult to achieve completeness and consistency for large system

Typical functional requirements are:

- Business Rules
- Transaction corrections, adjustments, cancellations
- Administrative functions
- Authentication
- Authorization –functions user is delegated to perform
- Audit Tracking
- External Interfaces
- Certification Requirements
- Reporting Requirements
- Historical Data
- Legal or Regulatory Requirements

NON-FUNCTIONAL REQUIREMENTS

"A non-functional requirement is a statement of how a system must behave, it is a constraint upon the systems behavior."

Types of Non-functional Requirements

1.Product Requirements

-Specify product behavior

-Include the following

- Usability
- Efficiency
- Reliability
- Portability

2.Organisational Requirements

--Derived from policies and procedures

--Include the following:

- Delivery
- Implementation
- Standard

3.External Requirements

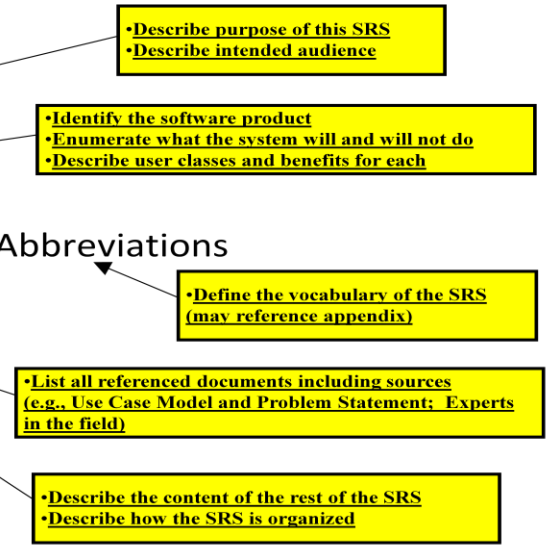
-- Derived from factors external to the system and its development process

--Includes the following

- ❖ Interoperability
- ❖ Ethical
- ❖ Legislative

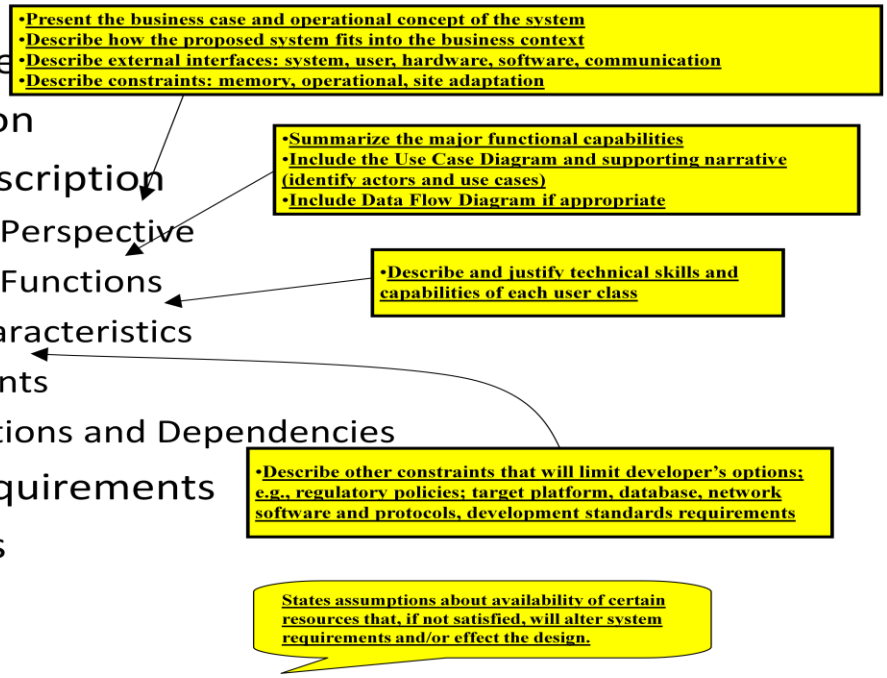
IEEE 830-1998 Standard – Section 1 of SRS

- 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



IEEE 830-1998 Standard – Section 2 of SRS

- Title
- Table of Conte
- 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



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 Requireme
  - 3.4 Logical Database Require
  - 3.5 Design Constraints
  - 3.6 Software System Quality
  - 3.7 Object Oriented Models
- 4. Appendices
- 5. Index

Specify software requirements in sufficient detail to enable designers to design a system to satisfy those requirements and testers to verify requirements

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

Requirements should include, at a minimum, 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 or in support of an output

(a) Requirements should have characteristics of high quality requirements
(b) Requirements should be cross-referenced to their source.
(c) Requirements should be uniquely identifiable
(d) Requirements should be organized to maximize readability