Software Engineering (Handouts by – T.Sujanavan, Asst. Prof., MVSREC)

Unit 2

Software Engineering Practice *(not included in current syllabus)*

- Consists of a collection of concepts, principles, methods, and tools that a software engineer calls upon on a daily basis
- Equips managers to manage software projects and software engineers to build computer programs
- Provides necessary technical and management how to's in getting the job done
- Transforms a haphazard unfocused approach into something that is more organized, more effective, and more likely to achieve success

The Essence of Problem Solving *(not included in current syllabus)*

1) Understand the problem (communication and analysis)
    - Who has a stake in the solution to the problem?
    - What are the unknowns (data, function, behavior)?
    - Can the problem be compartmentalized?
    - Can the problem be represented graphically?
2) Plan a solution (planning, modeling and software design)
    - Have you seen similar problems like this before?
    - Has a similar problem been solved and is the solution reusable?
    - Can subproblems be defined and are solutions available for the subproblems?
3) Carry out the plan (construction; code generation)
    - Does the solution conform to the plan? Is the source code traceable back to the design?
    - Is each component of the solution correct? Has the design and code been reviewed?
4) Examine the results for accuracy (testing and quality assurance)
    - Is it possible to test each component of the solution?
    - Does the solution produce results that conform to the data, function, and behavior that are required?

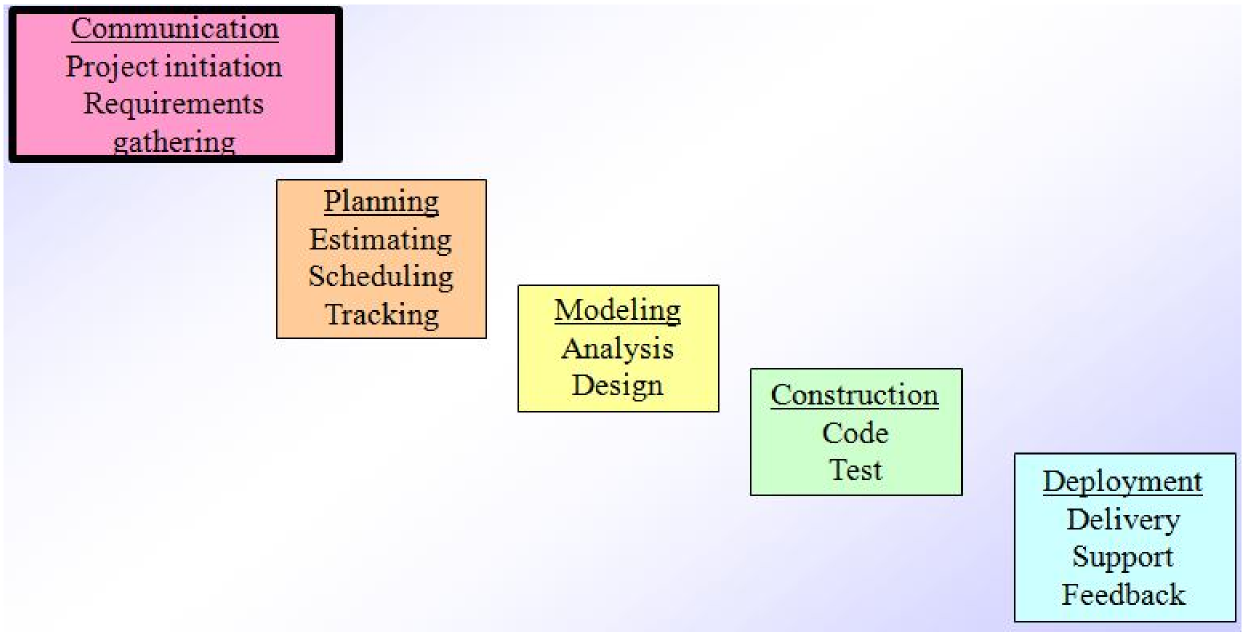
Seven Core Principles for Software Engineering *(not included in current syllabus)*

1) Remember the reason that the software exists
    - The software should provide value to its users and satisfy the requirements
2) Keep it simple, stupid (KISS)
    - All design and implementation should be as simple as possible
3) Maintain the vision of the project
    - A clear vision is essential to the project's success
4) Others will consume what you produce

- Always specify, design, and implement knowing that someone else will later have to understand and modify what you did

5) Be open to the future
   - Never design yourself into a corner; build software that can be easily changed and adapted
6) Plan ahead for software reuse
   - Reuse of software reduces the long-term cost and increases the value of the program and the reusable components
7) Think, then act
   - Placing clear, complete thought before action will almost always produce better results

Practices -



Communication Practice Principles *(not included in current syllabus)*

1) Listen to the speaker and concentrate on what is being said
2) Prepare before you meet by researching and understanding the problem
3) Someone should facility the meeting and have an agenda
4) Face-to-face communication is best, but also have a document or presentation to focus the discussion
5) Take notes and document decisions
6) Strive for collaboration and consensus
7) Stay focused on a topic; modularize your discussion
8) If something is unclear, draw a picture
9) Move on to the next topic a) after you agree to something, b) if you cannot agree to something, or c) if a feature or function is unclear and cannot be clarified at the moment
10) Negotiation is not a contest or a game; it works best when both parties win

Planning Practice principles *(not included in current syllabus)*

1) Understand the scope of the project
2) Involve the customer in the planning activity
3) Recognize that planning is iterative; things will change
4) Estimate based only on what you know
5) Consider risk as you define the plan
6) Be realistic on how much can be done each day by each person and how well
7) Adjust granularity as you define the plan
8) Define how you intend to ensure quality
9) Describe how you intend to accommodate change
10) Track the plan frequently and make adjustments as required

Barry Boehm's W^{5}HH Principle

- Why is the system being developed?
- What will be done?
- When will it be accomplished?
- Who is responsible for each function?
- Where are they organizationally located?
- How will the job be done technically and managerially?
- How much of each resource is needed?

The answers to these questions lead to a definition of key project characteristics and the resultant project plan

Modelling Practices *(not included in current syllabus)*

Analysis Modeling Principles

1) The information domain of a problem (the data that flows in and out of a system) must be represented and understood
2) The functions that the software performs must be defined
3) The behavior of the software (as a consequence of external events) must be represented
4) The models that depict information, function, and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion
5) The analysis task should move from essential information toward implementation detail

Design Modeling Principles

1) The design should be traceable to the analysis model
2) Always consider the software architecture of the system to be built
3) Design of data is as important as design of processing functions

4) Interfaces (both internal and external) must be designed with care
5) User interface design should be tuned to the needs of the end-user and should stress ease of use
6) Component-level design should be functionally independent (high cohesion)
7) Components should be loosely coupled to one another and to the external environment
8) Design representations (models) should be easily understandable
9) The design should be developed iteratively; with each iteration, the designer should strive for greater simplicity

Construction Practices *(not included in current syllabus)*

Coding principles -

Preparation before coding

1) Understand the problem you are trying to solve
2) Understand basic design principles and concepts
3) Pick a programming language that meets the needs of the software to be built and the environment in which it will operate
4) Select a programming environment that provides tools that will make your work easier
5) Create a set of unit tests that will be applied once the component you code is completed

As you begin coding

1) Constrain your algorithms by following structured programming practices
2) Select data structures that will meet the needs of the design
3) Understand the software architecture and create interfaces that are consistent with it
4) Keep conditional logic as simple as possible
5) Create nested loops in a way that makes them easily testable
6) Select meaningful variable names and follow other local coding standards
7) Write code that is self-documenting
8) Create a visual layout (e.g., indentation and blank lines) that aids code understanding

After completing the first round of code

1) Conduct a code walkthrough
2) Perform unit tests (black-box and white-box) and correct errors you have uncovered
3) Re-factor the code

Testing Principles -

1) All tests should be traceable to the software requirements
2) Tests should be planned long before testing begins

3) The Pareto principle applies to software testing
    - 80% of the uncovered errors are in 20% of the code
4) Testing should begin “in the small” and progress toward testing “in the large”
    - Unit testing --> integration testing --> validation testing --> system testing
5) Exhaustive testing is not possible

Test Objectives

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

Deployment Practices principles *(not included in current syllabus)*

1) Customer expectations for the software must be managed
    - Be careful not to promise too much or to mislead the user
2) A complete delivery package should be assembled and tested
3) A support regime must be established <u>before</u> the software is delivered
4) Appropriate instructional materials must be provided to end users
5) Buggy software should be fixed first, delivered later

SYSTEM ENGINEERING *(not included in current syllabus)*

Introduction

- Software engineering occurs as a consequence of system engineering
- System engineering may take on <u>two</u> different forms depending on the application domain
    - <u>“Business process” engineering</u> – conducted when the context of the work focuses on a business enterprise
    - <u>Product engineering</u> – conducted when the context of the work focuses on a product that is to be built
- Both forms bring order to the development of computer-based systems
- Both forms work to allocate a role for computer software and to establish the links that tie software to other elements of a computer-based system

System

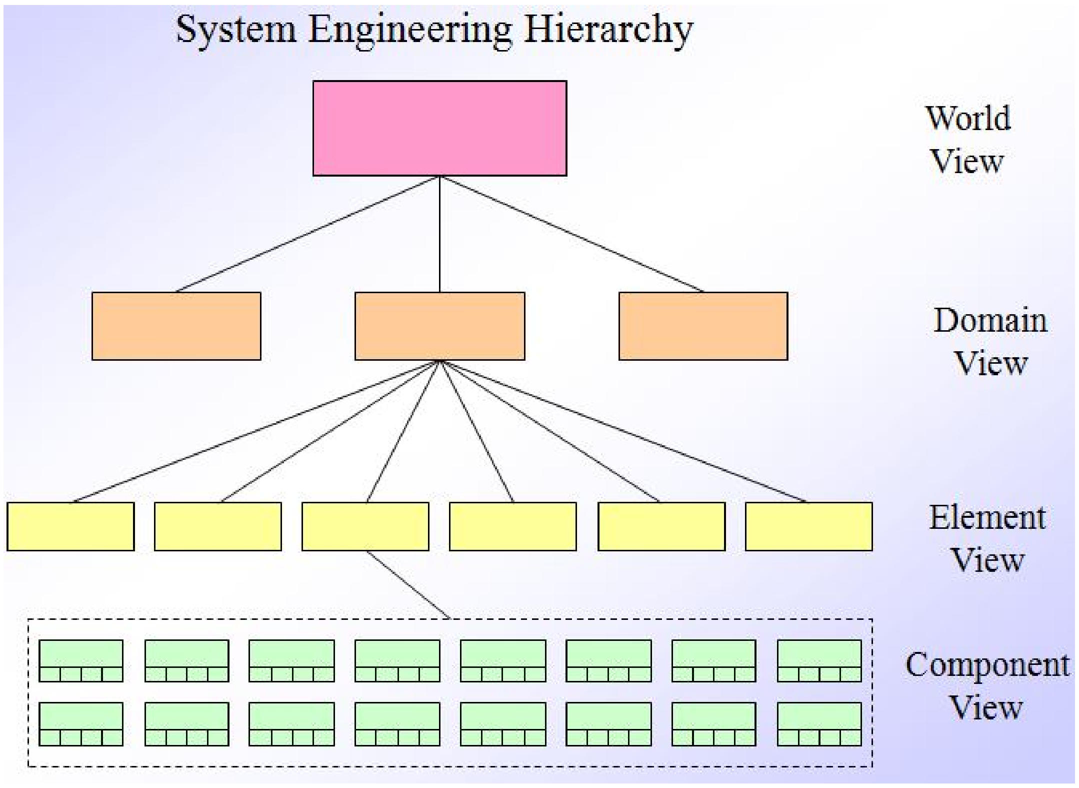
- A set or arrangement of things so related as to form a unity or organic whole
- A set of facts, principles, rules. etc., … to show a logical plan linking the various parts
- A method or plan of classification or arrangement
- An established way of doing something such as a method or procedure

Computer-based System

- Defined: A set or arrangement of elements that are organized to accomplish some predefined goal by processing information
- The goal may be to support some business function or to develop a product that can be sold to generate business revenue
- A computer-based system makes use of system elements
- Elements constituting one system may represent one macro element of a still larger system
- Example
  - A factory automation system may consist of a numerical control machine, robots, and data entry devices; each can be its own system
  - At the next lower hierarchical level, a manufacturing cell is its own computer-based system that may integrate other macro elements
- The role of the system engineer is to define the elements of a specific computer-based system in the context of the overall hierarchy of systems
- A computer-based system makes use of the following four system elements that combine in a variety of ways to transform information
  - **Software**: computer programs, data structures, and related work products that serve to effect the logical method, procedure, or control that is required
  - **Hardware**: electronic devices that provide computing capability, interconnectivity devices that enable flow of data, and electromechanical devices that provide external functions
  - **People**: Users and operators of hardware and software
  - **Database**: A large, organized collection of information that is accessed via software and persists over time
- The uses of these elements are described in the following:
  - **Documentation**: Descriptive information that portrays the use and operation of the system
  - **Procedures**: The steps that define the specific use of each system element or the procedural context in which the system resides

System Engineering Process *(not included in current syllabus)*

- The system engineering process begins with a world view; the business or product domain is examined to ensure that the proper business or technology context can be established
- The world view is refined to focus on a specific domain of interest
- Within a specific domain, the need for targeted system elements is analyzed
- Finally, the analysis, design, and construction of a targeted system element are initiated
- At the world view level, a very broad context is established
- At the bottom level, detailed technical activities are conducted by the relevant engineering discipline (e.g., software engineering)



at each view level

- Defines the <u>processes</u> (e.g., domain classes in OO terminology) that serve the needs of the view under consideration
- Represents the <u>behavior</u> of the processes and the assumptions on which the behavior is based
- Explicitly defines intra-level and inter-level <u>input</u> that form <u>links</u> between entities in the model
- Represents all linkages (including output) that will enable the engineer to better understand the view
- May result in models that call for one of the following
  - Completely automated solution
  - A semi-automated solution
  - A non-automated (i.e., manual) approach

Factors to Consider when Constructing a Model

- Assumptions
  - These reduce the number of possible variations, thus enabling a model to reflect the problem in a reasonable manner
- Simplifications
  - These enable the model to be created in a timely manner
- Limitations
  - These help to bound the maximum and minimum values of the system
- Constraints

  - These guide the manner in which the model is created and the approach taken when the model is implemented

- Preferences
  - These indicate the preferred solution for all data, functions, and behavior
  - They are driven by customer requirements

System Modeling with UML

- The Uniform Modeling Language (UML) provides diagrams for analysis and design at both the system and software levels
- Examples
  - Use case diagrams
  - Activity diagrams
  - Class diagrams
  - State diagrams

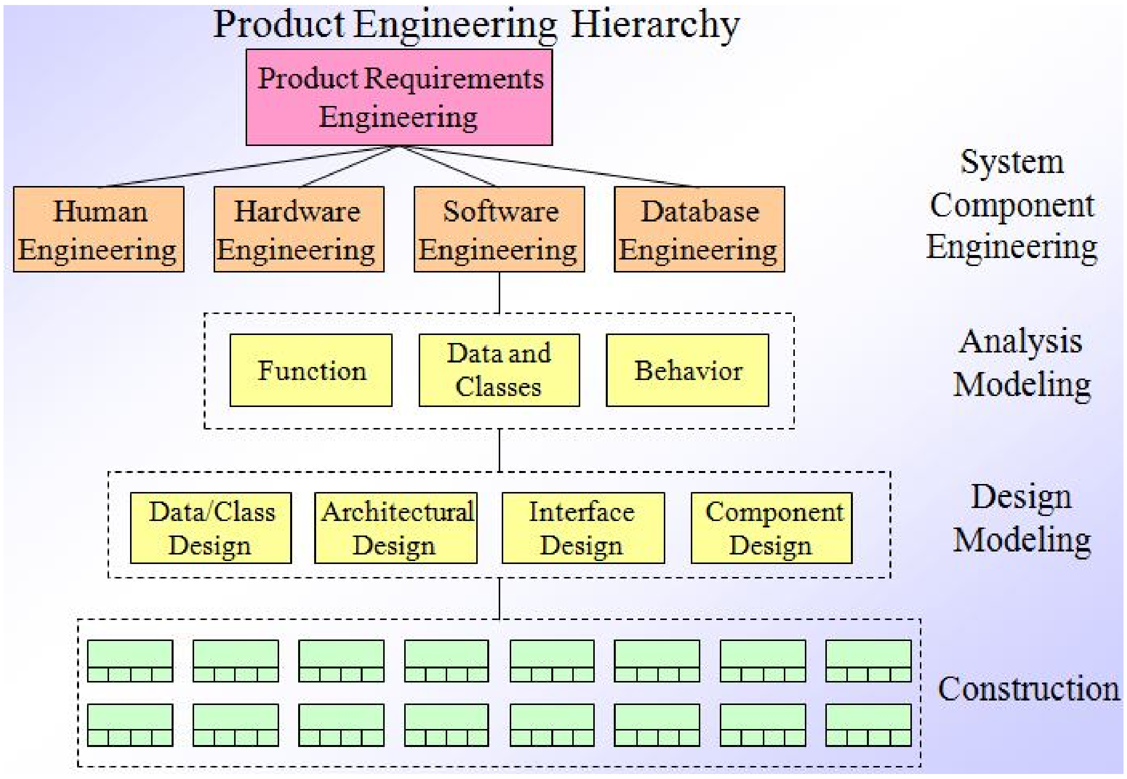
"Business Process" Engineering

- "Business process" engineering defines architectures that will enable a business to use information effectively
- It involves the <u>specification</u> of the appropriate computing architecture and the <u>development</u> of the software architecture for the organization's computing resources
- <u>Three</u> different architectures must be analyzed and designed within the context of business objectives and goals
  - The <u>data architecture</u> provides a framework for the information needs of a business (e.g., ERD)
  - The <u>application architecture</u> encompasses those elements of a system that transform objects within the data architecture for some business purpose
  - The <u>technology infrastructure</u> provides the foundation for the data and application architectures
    - It includes the hardware and software that are used to support the applications and data

Product Engineering

- Product engineering translates the customer's desire for a set of defined capabilities into a working product
- It achieves this goal by establishing a <u>product architecture</u> and a <u>support infrastructure</u>
  - Product architecture components consist of people, hardware, software, and data
  - Support infrastructure includes the technology required to tie the components together and the information to support the components
- <u>Requirements engineering</u> elicits the requirements from the customer and allocates function and behavior to each of the four components

- System component engineering happens next as a set of concurrent activities that address each of the components separately
  - Each component takes a domain-specific view but maintains communication with the other domains
  - The actual activities of the engineering discipline takes on an element view
- Analysis modeling allocates requirements into function, data, and behavior
- Design modeling maps the analysis model into data/class, architectural, interface, and component design



Planning and Managing the Project *(ref. – Soft. Engg. – Theory and practise 4th Ed. - Shari L.P. & J.M. Atlee)*

Objectives

- Tracking project progress
- Project personnel and organization
- Effort and schedule estimation
- Risk management
- Using process modelling with project planning

Tracking Progress

- 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

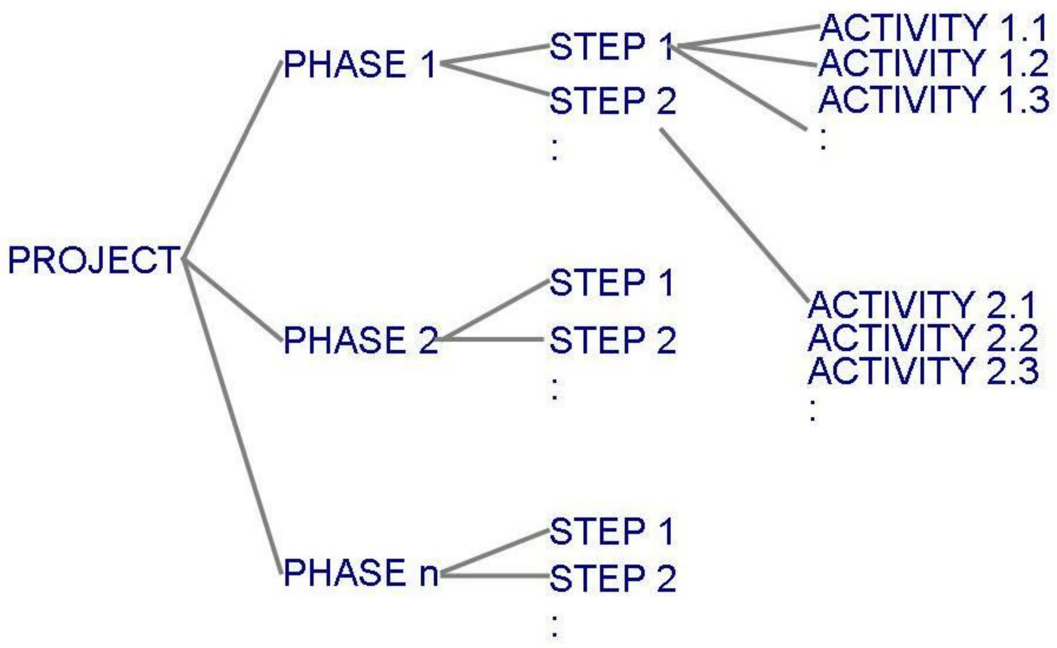


Fig : Project development can be separated into a succession of phases which are composed of steps, which are composed of activities

Example –

- Table 1 shows the phases, steps and activities to build a house
  - landscaping phase
  - building the house phase
- Table 2 lists milestones for building the house phase

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 1 - Phases, Steps, and Activities 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

Table 2 - Milestones in Building a House

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

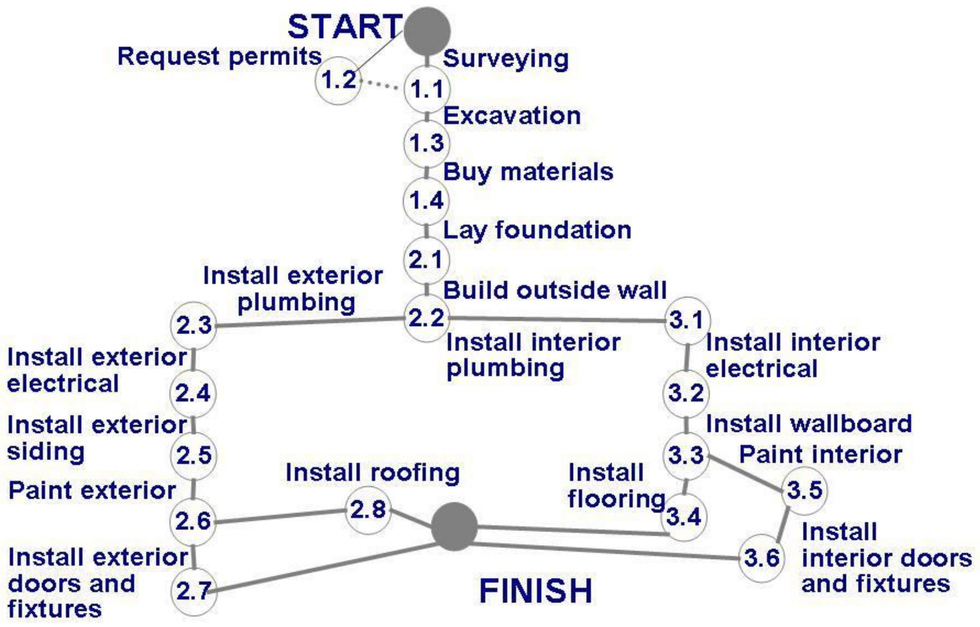
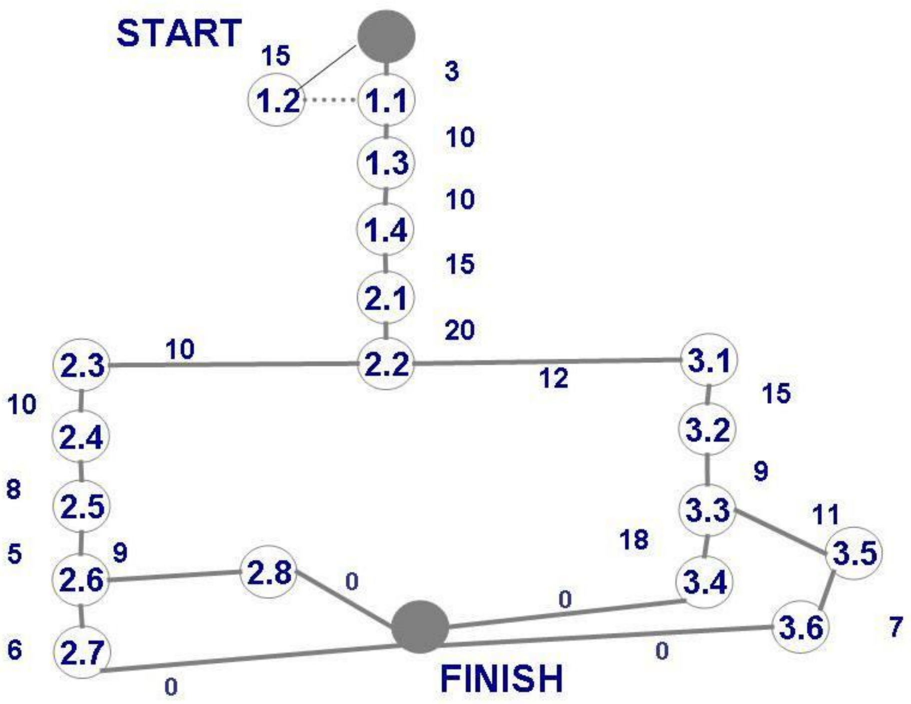


Fig : Activity graph for building a house

Estimating Completion



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

Estimating Completion for Building a House

Activity	**Time estimate (in days)**
Step 1: Prepare the site	
Activity 1.1: Survey the land	3
Activity 1.2: Request permits	15
Activity 1.3: Excavate for the foundation	10
Activity 1.4: Buy materials	10
Step 2: Building the exterior	
Activity 2.1: Lay the foundation	15
Activity 2.2: Build the outside walls	20
Activity 2.3: Install exterior plumbing	10
Activity 2.4: Exterior electrical work	10
Activity 2.5: Exterior siding	8
Activity 2.6: Paint the exterior	5
Activity 2.7: Install doors and fixtures	6
Activity 2.8: Install roof	9
Step 3: Finishing the interior	
Activity 3.1: Install the interior plumbing	12
Activity 3.2: Install interior electrical work	15
Activity 3.3: Install wallboard	9
Activity 3.4: Paint the interior	18
Activity 3.5: Install floor covering	11
Activity 3.6: Install doors and fixtures	7

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

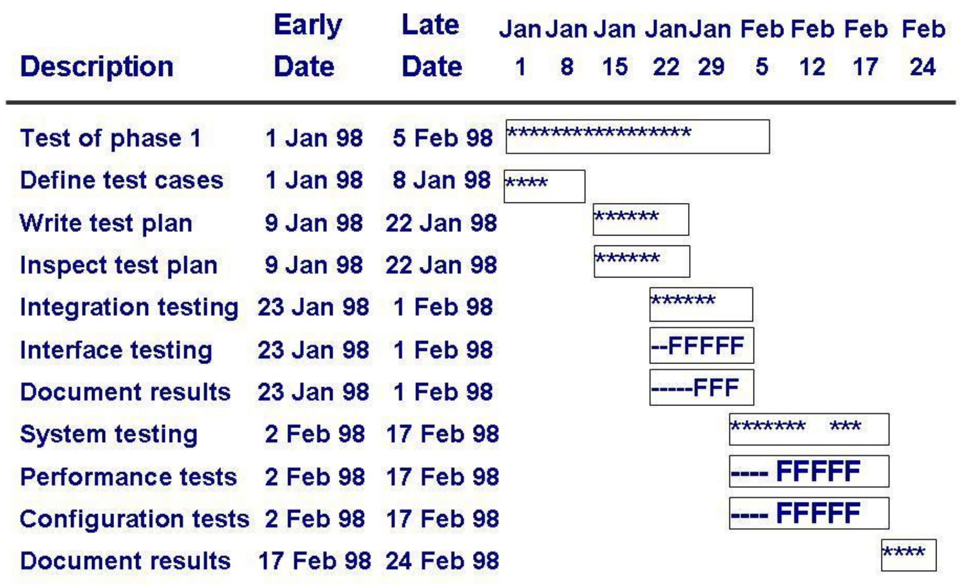
- **Real time (actual time):** estimated amount of time required for the activity to be completed
- **Available time:** amount of time available in the schedule for the activity's completion
- **Slack time:** the difference between the available time and the real time for that activity
- **Critical path**: 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

Slack Time for Activities of Building a House -

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

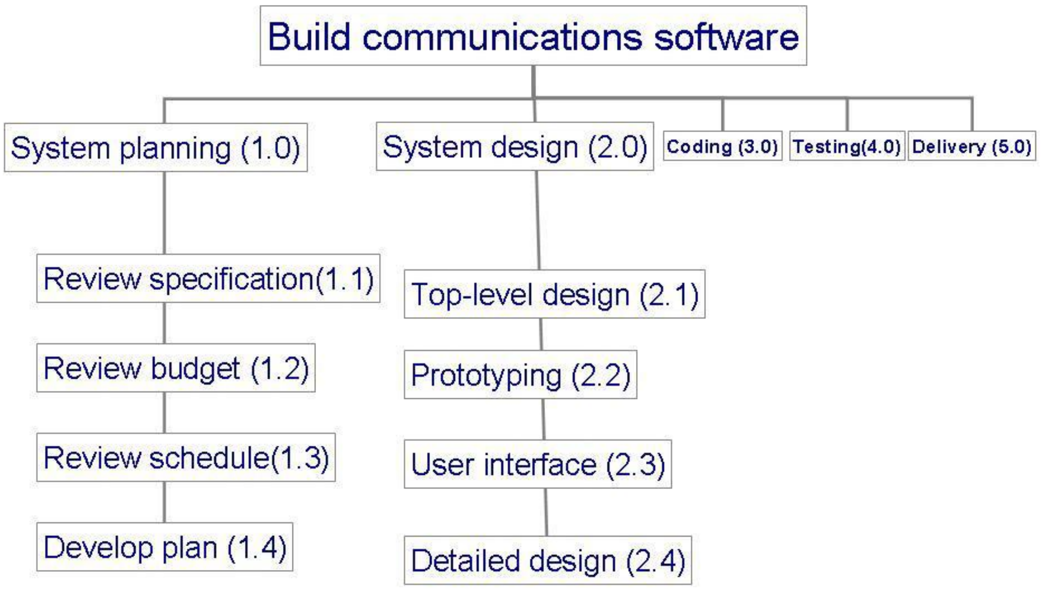


Description	Early Date	Late Date	Jan 1 · Jan 8 · Jan 15 · Jan 22 · Jan 29 · Feb 5 · Feb 12 · Feb 17 · Feb 24
Test of phase 1	1 Jan 98	5 Feb 98	*****************
Define test cases	1 Jan 98	8 Jan 98	****
Write test plan	9 Jan 98	22 Jan 98	******
Inspect test plan	9 Jan 98	22 Jan 98	******
Integration testing	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	****

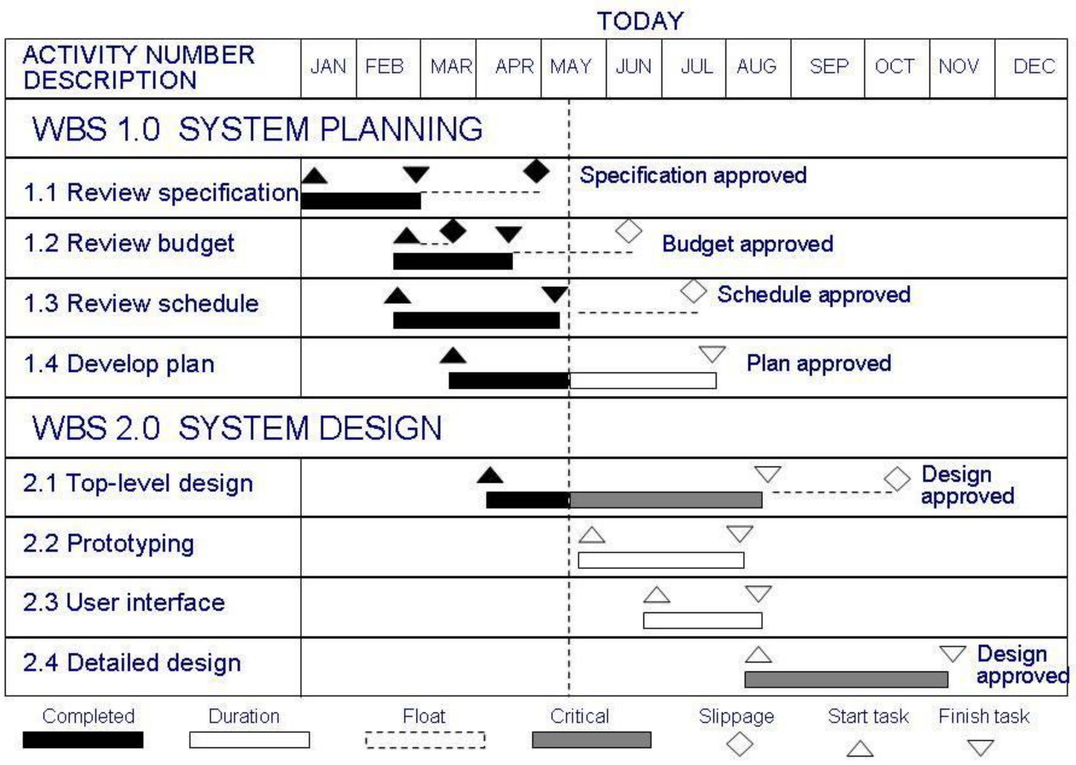
- Including information about the early and late start dates
- Asterisks indicate the critical path

Tools to Track Progress

- Example: to track progress of building a communication software

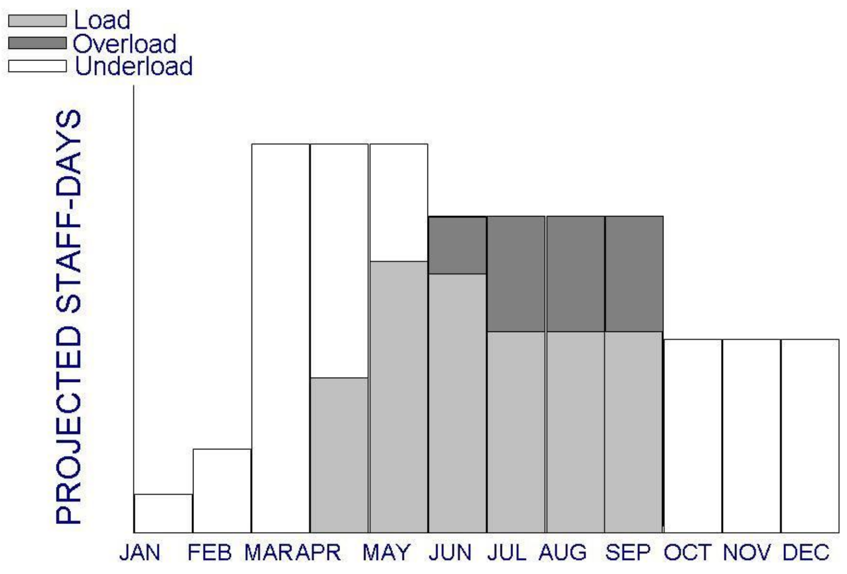


Gantt Chart



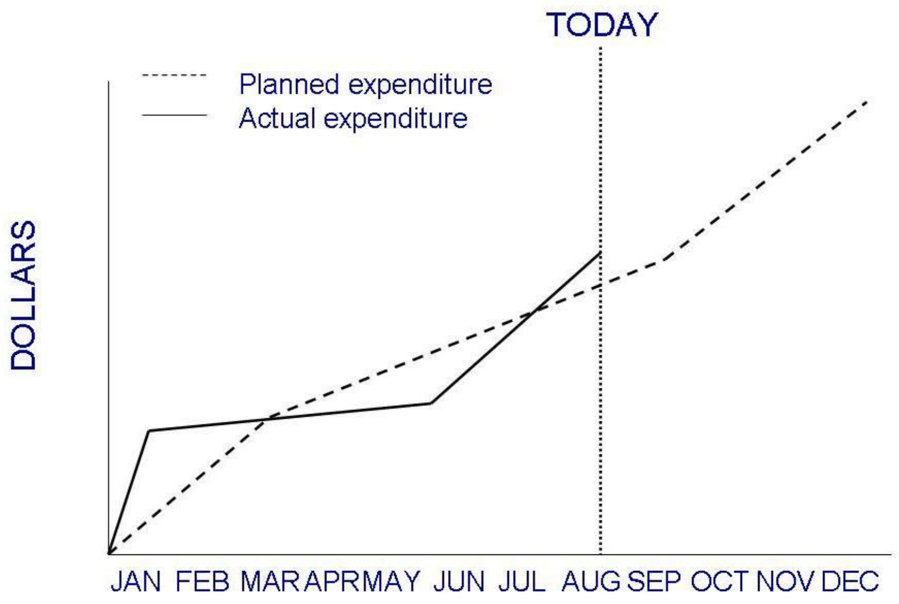
- Activities shown in parallel
  - helps understand which activities can be performed concurrently

Resource Histogram



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

Expenditures Tracking



- An example of how expenditures can be monitored

Project Personnel

- Key activities requiring personnel
  - requirements analysis
  - system design
  - program design
  - program implementation

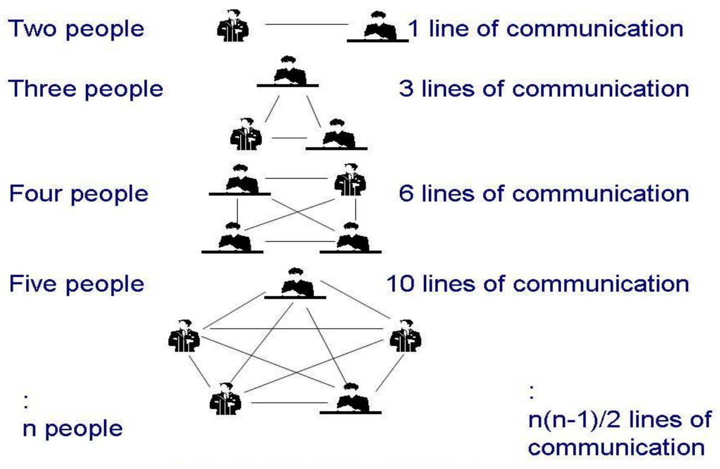
    - testing
    - training
    - maintenance
    - quality assurance
- There is great advantage in assigning different responsibilities to different people

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

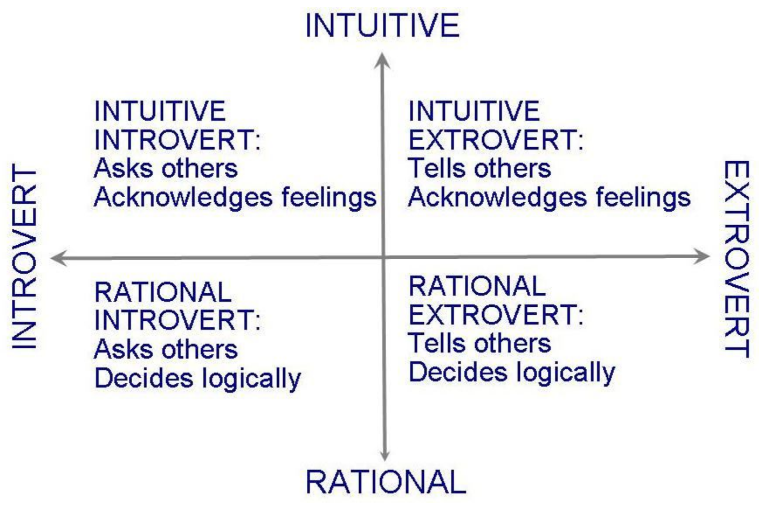

INTUITIVE
INTUITIVE INTROVERT: Asks others Acknowledges feelings
INTUITIVE EXTROVERT: Tells others Acknowledges feelings
INTROVERT
EXTROVERT
RATIONAL INTROVERT: Asks others Decides logically
RATIONAL EXTROVERT: Tells others Decides logically
RATIONAL

Fig : 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*

Chief Programmer Team

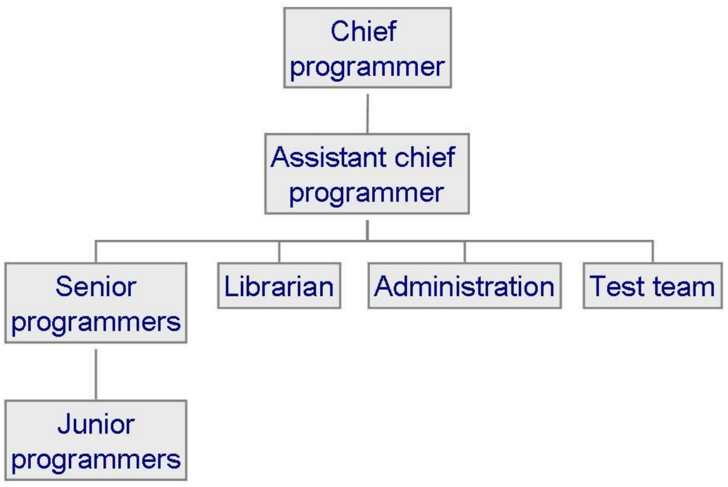


Fig : Each team member must communicate often with chief, but not necessarily with other team members

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, multi-storeyed 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

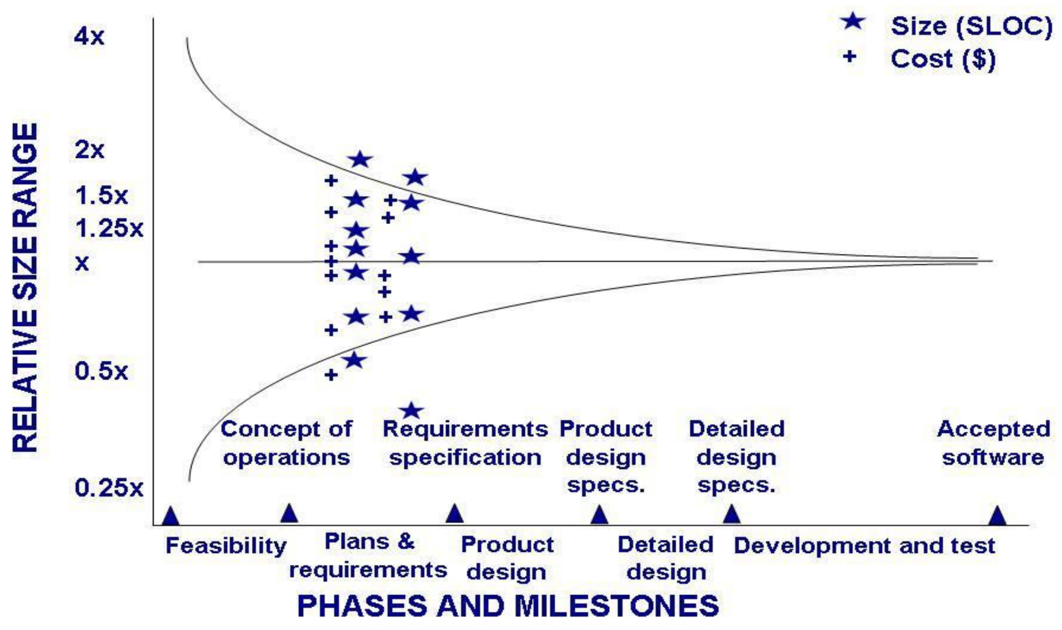


Fig : Uncertainty early in the project can affect the accuracy of cost and size estimations

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 included 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	

Agorithmic Method: Bailey-Basili technique

- Minimize standard error estimate to produce an equation such as

$E = 5.5 + 0.73S^{1.16}$

- Adjust initial estimate based on the difference ratio
  - If R is the ratio between the actual effort, E, and the predicted effort, E', then the effort adjustment is defined as
  - $ER_{adj} = R - 1$ if $R > 1$ $\quad = 1 - 1/R$ if $R < 1$
- Adjust the initial effort estimate E_{adj}
  - $E_{adj} = (1 + ER_{adj})E$ if $R > 1$ $\quad = E/(1 + ER_{adj})$ if $R < 1$

Bailey-Basily Modifier

	Cumulative complexity (CPLX)	*Cumulative experience (EXP)*
Tree charts	Customer interface complexity	Programmer qualifications
Top-down design	Application complexity	Programmer machine experience
Formal documentation	Program flow complexity	Programmer language experience
Chief programmer teams	Internal communication complexity	Programmer application experience
Formal training	Database complexity	Team experience
Formal test plans	External communication complexity	
Design formalisms	Customer-initiated program design changes	
Code reading		
Unit development folders		

COCOMO model ***

- Introduced by Boehm
- COCOMO II
  - updated version
  - include models of reuse
- The basic models
  - $E = bS^c m(X)$
  - where
    - bS^c is the initial size-based estimate
    - $m(X)$ is the vector of cost driver information

COCOMO II: Stages of Development

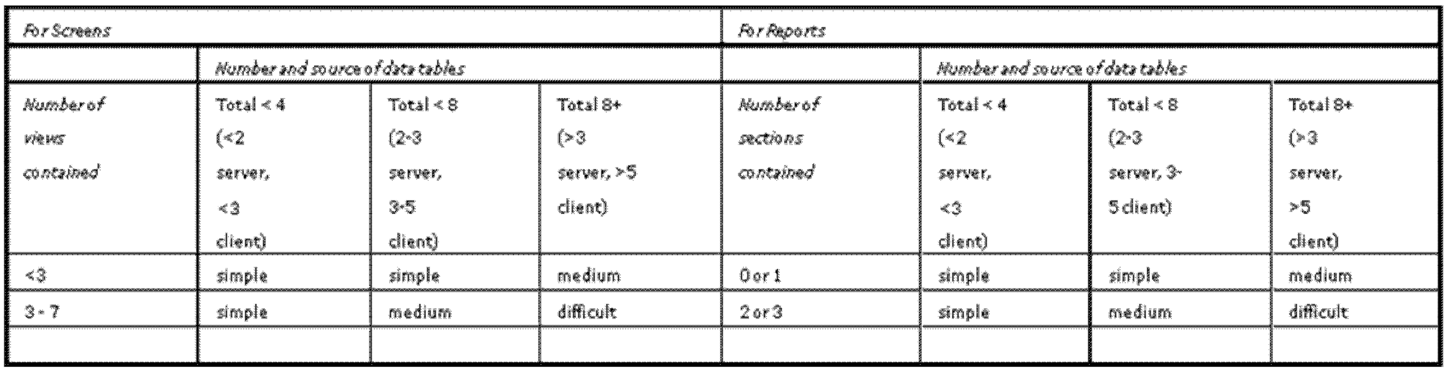
- Application composition
  - prototyping to resolve high-risk user interface issues
  - size estimates in object points
- Early design
  - to explore alternative architectures and concepts
  - size estimates in function points
- Post architecture
  - development has begun
  - size estimates in lines of code

Three Stages of COCOMO II

Model Aspect	Stage 1: Application Composition	Stage 2: Early Design	Stage 3: Post-architecture
Size	Application points	Function points (FP) and language	FP and language or source lines of code (SLOC)
Reuse	Implicit in model	Equivalent SLOC as function of other variables	Equivalent SLOC as function of other variables
Requirements change	Implicit in model	% change expressed as a cost factor	% change expressed as a cost factor
Maintenance	Application Point Annual Change Traffic	Function of ACT, software understanding, unfamiliarity	Function of ACT, software understanding, unfamiliarity
Scale (c) in nominal effort equation	1.0	0.91 to 1.23, depending on precedentedness, conformity, early architecture, risk resolution, team cohesion, and SEI process maturity	0.91 to 1.23, depending on precedentedness, conformity, early architecture, risk resolution, team cohesion, and SEI process maturity
Product cost drivers	None	Complexity, required reusability	Reliability, database size, documentation needs, required reuse, and product complexity
Platform cost drivers	None	Platform difficulty	Execution time constraints, main storage constraints, and virtual machine volatility
Personnel cost drivers	None	Personnel capability and experience	Analyst capability, applications experience, programmer capability,
programmer experience, experience, and personnel continuity			language and tool
Project cost drivers	None	Required development schedule, development	Use of software tools, required development schedule, and
	environment	multisite development	

COCOMO II: Estimate Application Points

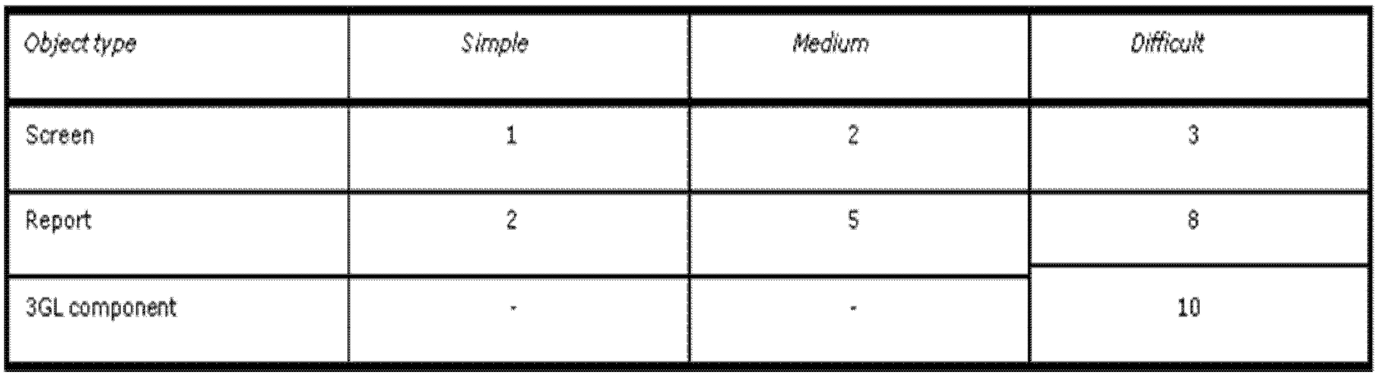
- To compute application points, first we need to count the number of screens, reports and programming language used to determine the complexity level



For Screens				*For Reports*			
	Number and source of data tables				*Number and source of data tables*		
Number of views contained	Total < 4 (<2 server, <3 client)	Total < 8 (2-3 server, 3-5 client)	Total 8+ (>3 server, >5 client)	*Number of sections contained*	Total < 4 (<2 server, <3 client)	Total < 8 (2-3 server, 3-5 client)	Total 8+ (>3 server, >5 client)
<3	simple	simple	medium	0 or 1	simple	simple	medium
3 - 7	simple	medium	difficult	2 or 3	simple	medium	difficult

- Determine the relative effort required to implement a report or screen simple, medium or difficult
- Calculate the productivity factor based on developer experience and capability
- Determine the adjustment factors expressed as multipliers based on rating of the project

Complexity Weights for Application Points



Object type	*Simple*	*Medium*	*Difficult*
Screen	1	2	3
Report	2	5	8
3GL component	-	-	10

Productivity Estimate Calculation

Developers' experience and capability	Very low	Low	Nominal	High	Very high
CASE maturity and capability	Very low	Low	Nominal	High	Very high
Productivity factor	4	7	13	25	50

Tool Use Categories

Category	*Meaning*
Very low	Edit, code, debug
Low	Simple front-end, back-end CASE, little integration
Nominal	Basic life-cycle tools, moderately integrated
High	Strong, mature life-cycle tools, moderately integrated
Very high	Strong, mature, proactive life-cycle tools, well-integrated with processes, methods, reuse

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

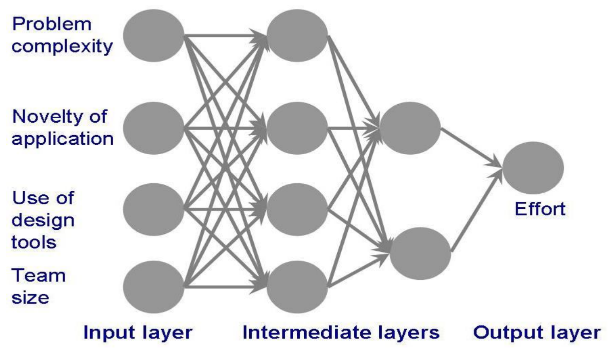


Fig : Neural network used by Shepperd to produce effort estimation

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

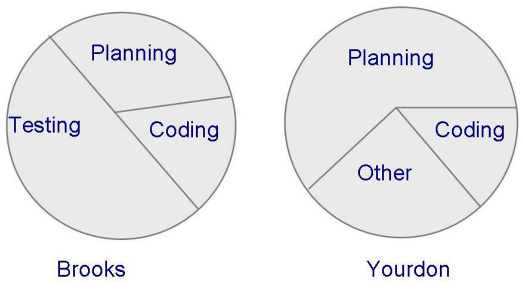
Finding the Model for Your Situation

- Mean magnitude of relative error (MMRE)
  - absolute value of mean of [(actual - estimate)/actual]
  - goal: should be .25 or less
- Pred(x/100): percentage of projects for which estimate is within *x*% of the actual
  - goal: should be .75 or greater for *x* = .25
  - 75% project estimates are within 25% of actual

Evaluating Models

Model	*PRED(0.25)*	*MMRE*
Walston-Felix	0.30	0.48
Basic COCOMO	0.27	0.60
Intermediate COCOMO	0.63	0.22
Intermediate COCOMO (variation)	0.76	0.19
Bailey-Basili	0.78	0.18
Pfleeger	0.50	0.29
SLIM	0.06-0.24	0.78-1.04
Jensen	0.06-0.33	0.70-1.01
COPMO	0.38-0.63	0.23-5.7
General COPMO	0.78	0.25

- No model appears to have captured the essential characteristics and their relationships for all types of development



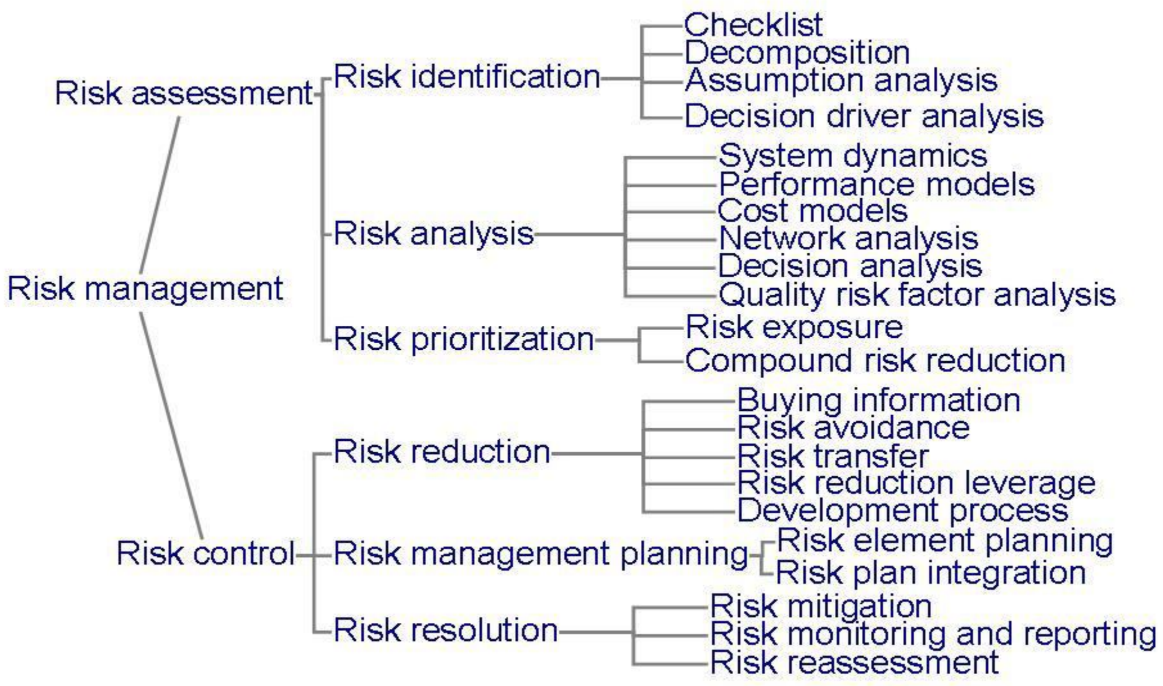
- It is important to understand which types of effort are needed during development even when we have reasonably accurate estimate
  - Categorize and save the results
- Two different reports of effort distribution from different researchers

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

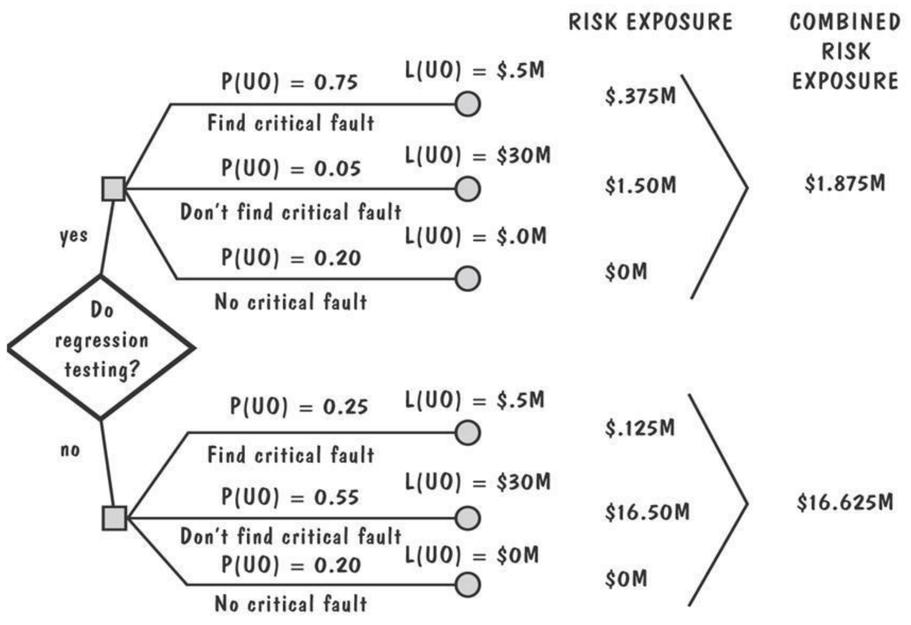


Fig : PU: prob. of unwanted outcome ; LU: lost assoc with unwanted outcome

- 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

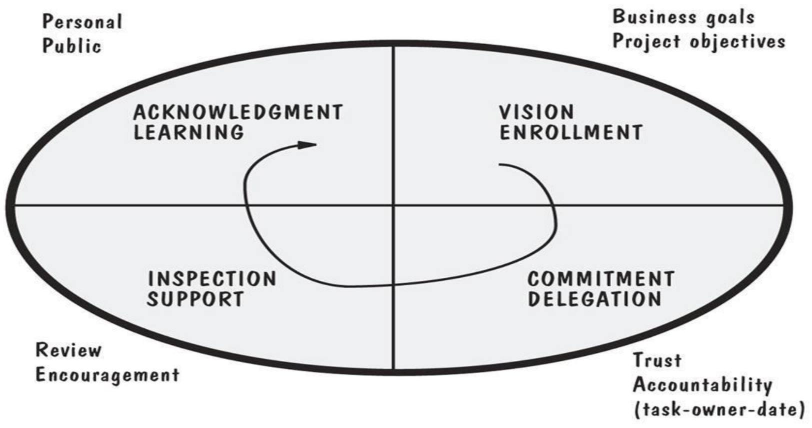
- Project scope
- Project schedule
- Project team organization
- Technical description of system
- Project standards and procedures
- Quality assurance plan
- Configuration management plan
- Documentation plan
- Data management plan
- Resource management plan
- Test plan
- Training plan
- Security plan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

Project Plan Lists

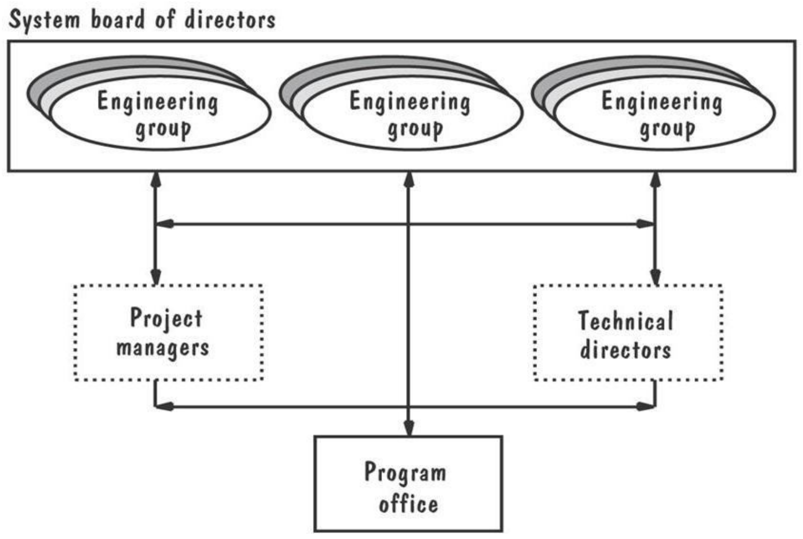
- List of the people in development team
- List of hardware and software
- Standards and methods, such as
  - algorithms
  - tools
  - review or inspection techniques
  - design language or representations
  - coding languages
  - testing techniques

Process Models and Project Management Enrolment 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 "enrol" the related programs, so they all shared common goals



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

Process Models and Project Management Accountability modelling: **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

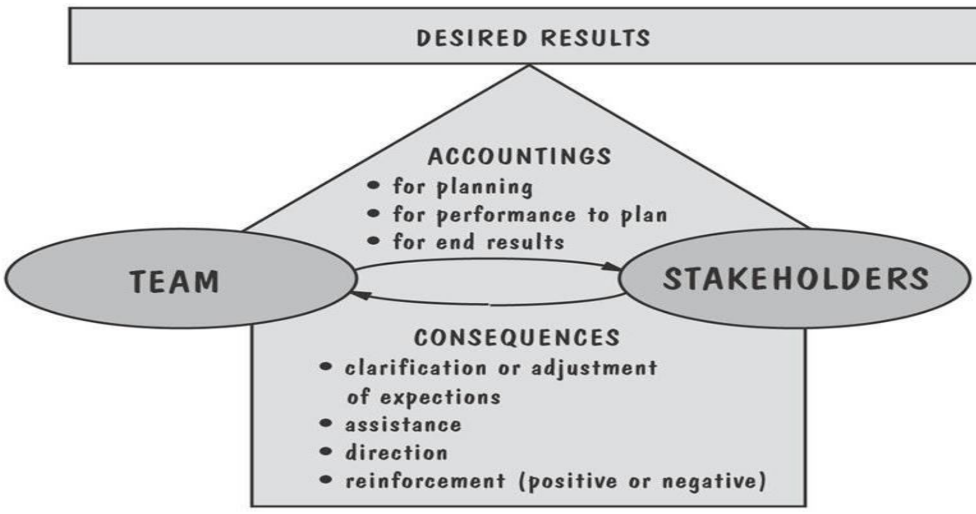
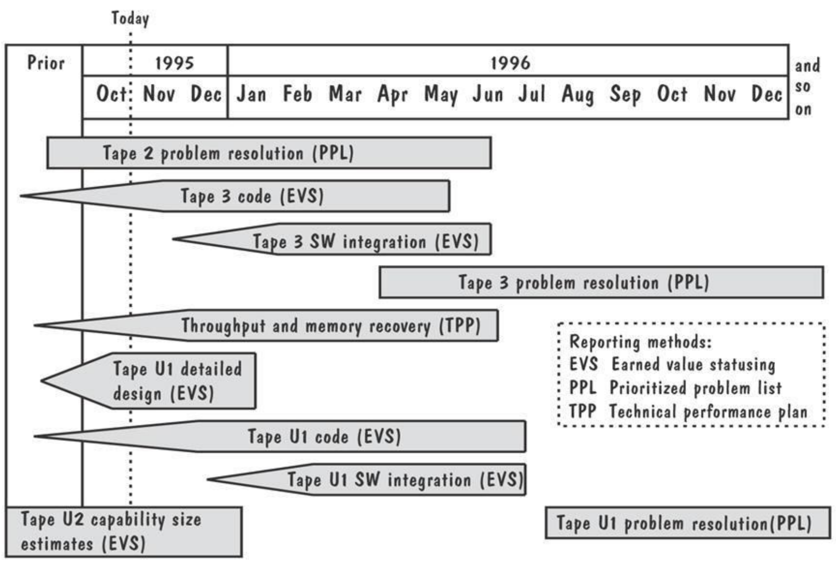
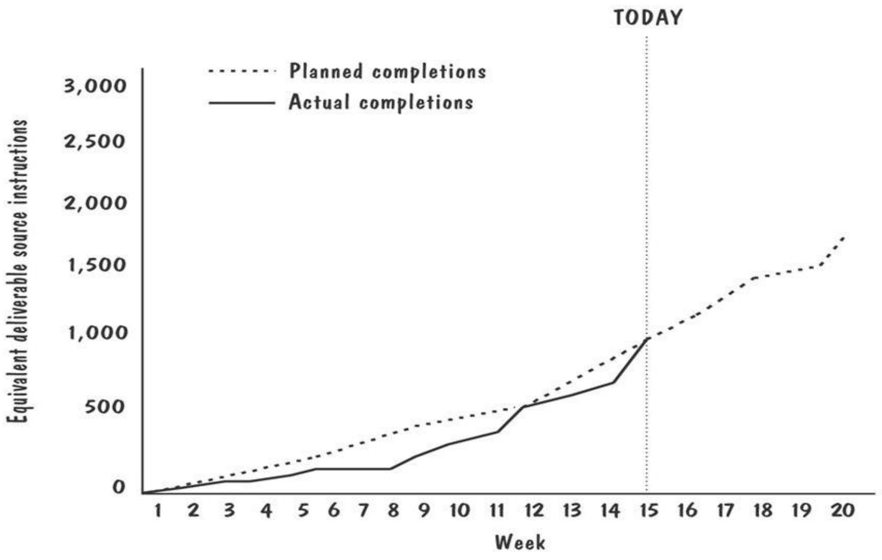


Fig : Accountability model used in F-16 Project


Today
Prior
1995
1996
Oct Nov Dec Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
and so on
Tape 2 problem resolution (PPL)
Tape 3 code (EVS)
Tape 3 SW integration (EVS)
Tape 3 problem resolution (PPL)
Throughput and memory recovery (TPP)
Tape U1 detailed design (EVS)
Tape U1 code (EVS)
Tape U1 SW integration (EVS)
Tape U2 capability size estimates (EVS)
Tape U1 problem resolution(PPL)
Reporting methods:
EVS Earned value statusing
PPL Prioritized problem list
TPP Technical performance plan

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

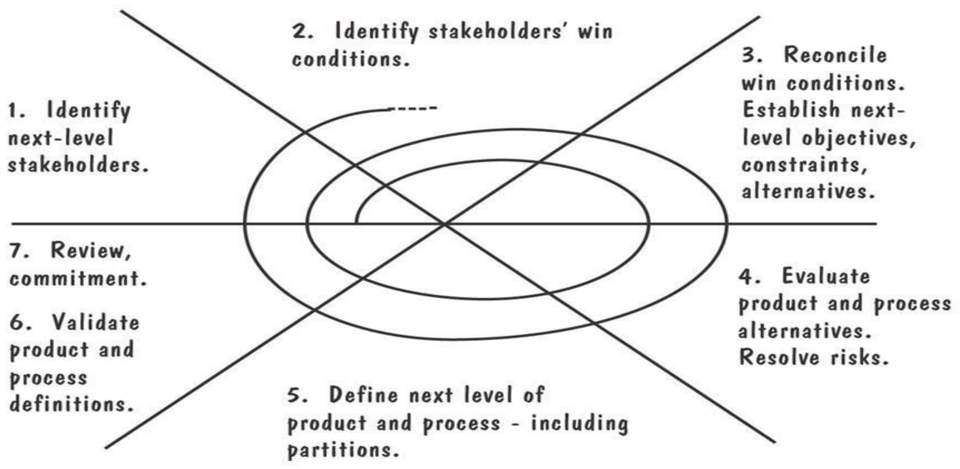


- Each activity's progress was tracked using earned value chart

Process Models and Project Management 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



1. Identify next-level stakeholders.
2. Identify stakeholders' win conditions.
3. Reconcile win conditions. Establish next-level objectives, constraints, alternatives.
4. Evaluate product and process alternatives. Resolve risks.
5. Define next level of product and process - including partitions.
6. Validate product and process definitions.
7. Review, commitment.

- 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

Real Time Example - 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: assessment of the risk using reuse software
  - Risk avoidance: using SRI with two different designs

Requirements Engineering

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

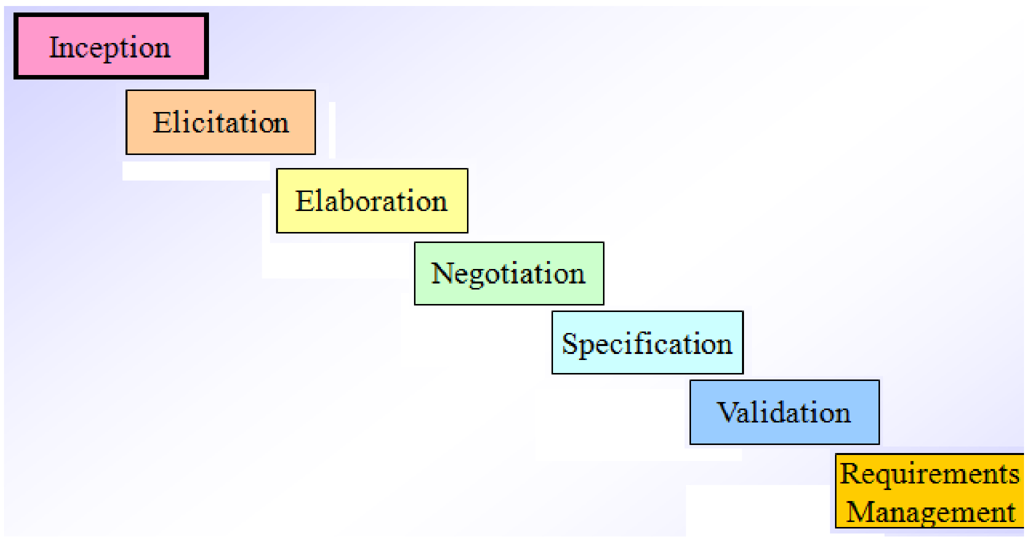
- Begins during the communication activity and continues into the modeling activity
- Builds a bridge from the system requirements into software design and construction
- Allows the requirements engineer to examine
    - the context of the software work to be performed
    - the specific needs that design 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



Inception Task

- During inception, the requirements engineer asks a set of questions to establish...
  - A basic understanding of the problem
  - The people who want a solution
  - The nature of the solution that is desired
  - The effectiveness of preliminary communication and collaboration between the customer and the developer
- Through these questions, the requirements engineer needs to...
  - Identify the stakeholders
  - Recognize multiple viewpoints
  - Work toward collaboration
  - 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?

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

- 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

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

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

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

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

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

Questions to ask when Validating Requirements

- Is each requirement consistent with the overall objective for the system/product?
- Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
- 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 (generally, a specific individual) 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?
  - Approaches: Demonstration, actual test, analysis, or inspection
- Does the requirements model properly reflect the information, function, and behavior of the system to be built?
- Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system?

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