

Software Engineering(SE)

CSC 601



Subject Incharge

Varsha Nagpurkar

Assistant Professor

Room No. 407

email: varshanagpurkar@sfit.ac.in

Module-1

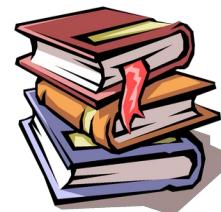
Module No.	Unit No.	Topics
1.0		Introduction To Software Engineering and Process Models
	1.1	Nature of Software, Software Engineering, Software Process, Capability Maturity Model (CMM)
	1.2	Generic Process Model, Prescriptive Process Models: The Waterfall Model, V-model, Incremental Process Models, Evolutionary Process Models, Concurrent Models, Agile process, Agility Principles, Extreme Programming (XP), Scrum, Kanban model

What is Software

- Software is: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately manipulate information, and (3) descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.

What is software?

- Computer programs and associated documentation

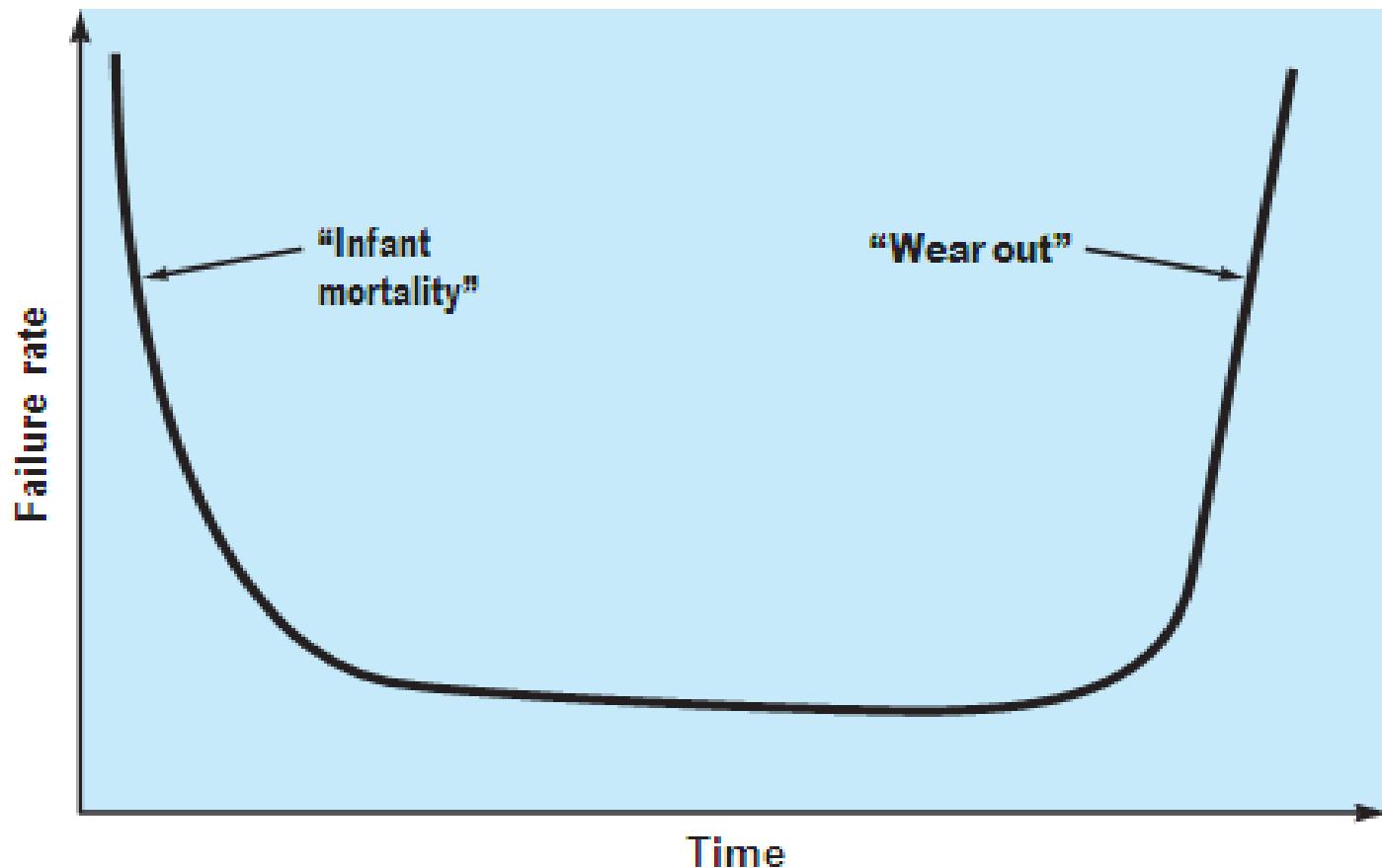


- Software products may be developed for a particular customer or may be developed for a general market
- Software products may be
 - Generic - developed to be sold to a range of different customers
 - Bespoke (custom) - developed for a single customer according to their specification

Software Characteristic

- Software is a logical rather than a physical system element.
- Software has one fundamental characteristic that makes it considerably different from hardware: *Software doesn't "wear out.-exhaust"*

Failure curve for Hardware



Failure curve for Hardware

- The relationship, often called the “bathtub curve,” indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects);
- defects are corrected and the failure rate drops to a steady-state level (hopefully, quite low) for some period of time.

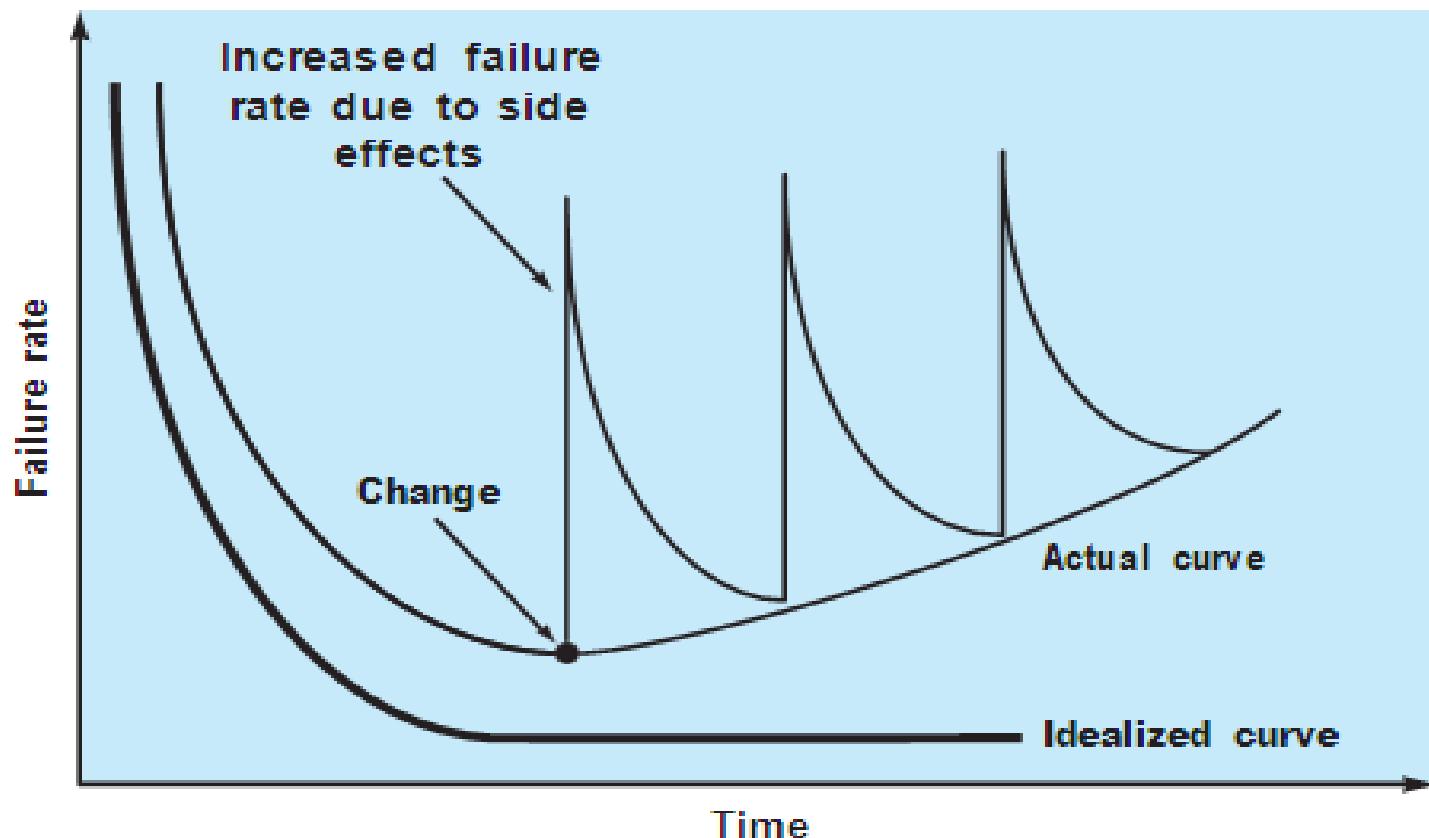
Failure curve for Hardware

- As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to *wear out*.

Failure curve for Software

- Software is not susceptible to the environmental maladies that cause hardware to wear out.
- In theory, therefore, the failure rate curve for software should take the form of the “idealized curve” shown in Figure
- Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected and the curve flattens as shown. However, the implication is clear—software doesn’t wear out. But it does *deteriorate!*

Failure curve for Software



Failure curve for Software

- During its life, software will undergo change. As changes are made, it is likely that errors will be introduced, causing the failure rate curve to spike as shown in the figure .
- Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again.
- Slowly, the minimum failure rate level begins to rise—the software is deteriorating due to change.

Software Application Domains

- **System Software**
- **Application Software**
- **Engineering/Scientific Software**
- **Embedded Software**
- **Product-line software**
- **Web/Mobile applications**
- **Artificial intelligence software**

Software Application

Domains

- **System software—**
- A collection of programs written to service other programs.
- It is characterized by its heavy interaction with computer hardware.
- Some system software examples (e.g., compilers, editors, and file management utilities) operating systems, drivers, networking software

Software Application Domains

- **Application software—**
- It consists of stand-alone programs that solve a specific business need.
- Examples of application software
 - Hospital management system
 - Bank management system
 - A payroll system

Software Application Domains

- **Engineering/scientific software**
—
- **In this type application areas are:**
 - Astronomy-Theory of space
 - Volcanology-Science about volcano
 - Molecular biology
 - Computer-aided design(e.g.AutoCAD software)

Software Application Domains

- **Embedded software—**
- It resides within a product or system and is used to implement and control features and functions for the end user and for the system itself.
- Embedded software can perform limited and esoteric functions (e.g., key pad control for a microwave oven)

Software Application Domains

- **Product-line software—** designed to provide a specific capability for use by many different customers.
- The personal computer software comes in this category

Examples of product-line software

- Inventory control(A software developed for some supermarket keeping track of inventory database and related manipulations.
- Word processing
- Spreadsheets
- Multimedia(Audio,Video)
- Entertainment(e.g.animation and other movies)

Software Application Domains

- **Web/Mobile applications—**
- This network-centric software category spans a wide range of applications and includes both browser-based apps and software that resides on mobile devices.
- Website related applications with databases, hyperlinks and graphics capabilities
- E-commerce web applications

Software Application Domains

- **Artificial intelligence software**—makes use of nonnumerical algorithms to solve complex problems that are not amenable to computation or straight-forward analysis.
- Applications within this area include robotics, expert systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing.

What is software engineering?

Software engineering is an engineering discipline which is concerned with all aspects of software production

Software engineers should

- adopt a systematic and organised approach to their work
- use appropriate tools and techniques depending on
 - **the problem to be solved,**
 - **the development constraints and**
 - **the resources available**



Software Engineering

- Software Engineering : (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software
- Software engineering is a layered technology. Referring to Figure, any engineering approach (including software engineering) must rest on an organizational commitment to quality.

Software Engineering Layers



Quality Focus

- Quality is nothing but ‘degree of goodness’.
- Good quality software has following characteristics:
 - Correctness is degree to which software performs its required function
 - Maintainability is an ease with which software is maintained
 - Integrity is a security provided so that unauthorized user can not access data or information.e.g.password authentication
 - Usability is the efforts required to use or operate the software.

Software Engineering Layers- Process Layer

- The foundation for software engineering is the *process* layer. The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software
- A *process* is a collection of activities, actions, and tasks that are performed when some work product is to be created.

Software Engineering Layers- Process Layer

- Software process defines following ‘what’ activities, actions and tasks for software development:
 - What activities are to be carried out?
 - What actions will be taken?
 - What tasks are to be carried out in a given action?

Software Engineering Layers- Process Layer

- An *activity* refers to achieve a broad objective (e.g., communication with stakeholders)
- An *action* (e.g., architectural design) encompasses a set of tasks that produce a major work product (e.g., an architectural model).
- A *task* focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

Software Engineering Layers-Methods

- Software engineering *methods* provide the technical way to implement the software.
- Methods consist of collection of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support.

Software Engineering Layers-Tools

- Software engineering tools provide automated or semi-automated support for the process and the methods.
- For example
 - Microsoft front page or Microsoft Publisher can be used as web designing tool.
 - Rational Rose can be used as object oriented analysis and design tool.

The Process Framework-A Generic Process Model

- A generic process framework for software engineering encompasses five activities:
 - Communication
 - Planning
 - Modeling
 - Construction
 - Deployment

A software process framework

Software process

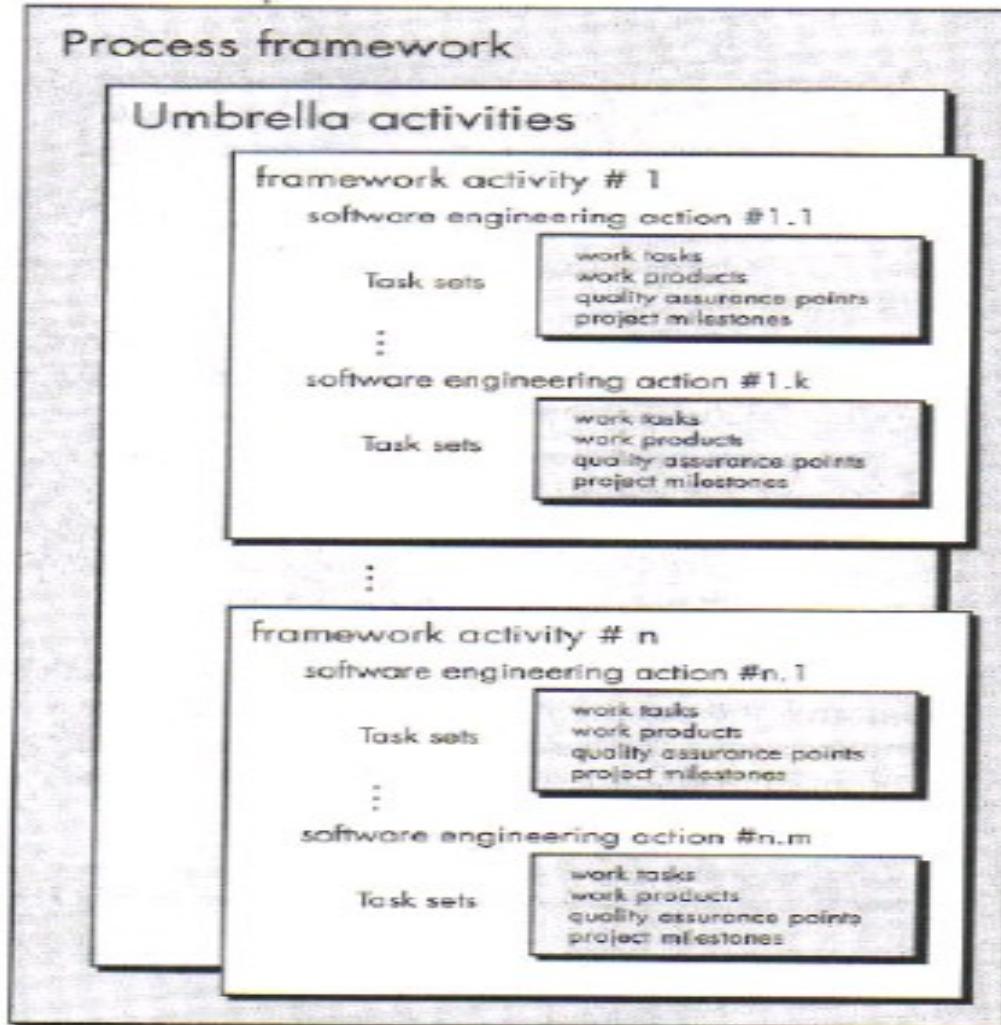


Fig. Software process framework

A software process framework

- Each framework activity is populated by a set of software engineering actions-a collection of related tasks that produces a major software engineering work product(e.g.,design is a software engineering action)
- Each action is populated with individual work tasks that accomplish some part of the work implied by the action.

Umbrella Activities

- Software engineering process framework activities are complemented by a number of *umbrella activities*.
- Umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk. Typical umbrella activities include:

Software Activities

- **Software project tracking and control**—allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.
- **Risk management**—assesses risks that may affect the outcome of the project or the quality of the product.
- **Software quality assurance**—defines and conducts the activities required to ensure software quality.
- **Technical reviews**—assess software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.

Umbrella Activities

- **Measurement**—defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs
- **Software configuration management**—manages the effects of change throughout the software process.
- **Reusability management**—defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.
- **Work product preparation and production**—includes the activities required to create work products such as models, documents, logs, forms, and lists.

Prescriptive Process Models

- All software organizations should encompass the following set of framework activities in their process model
 - Communication
 - Planning
 - Modelling
 - Construction
 - Deployment

Prescriptive Process Models

- The name ‘prescriptive’ is given since the model prescribes set of activities, actions, tasks, quality assurance and change control mechanism for every project
- It also prescribes a workflow
- Workflow is the flow of process elements and the manner in which they are interrelated
- Each prescriptive model put different emphasis to generic framework activities and give different workflow.

Prescriptive Process Models

- The waterfall model
- Incremental process models
- Evolutionary process models
- The specialized process models

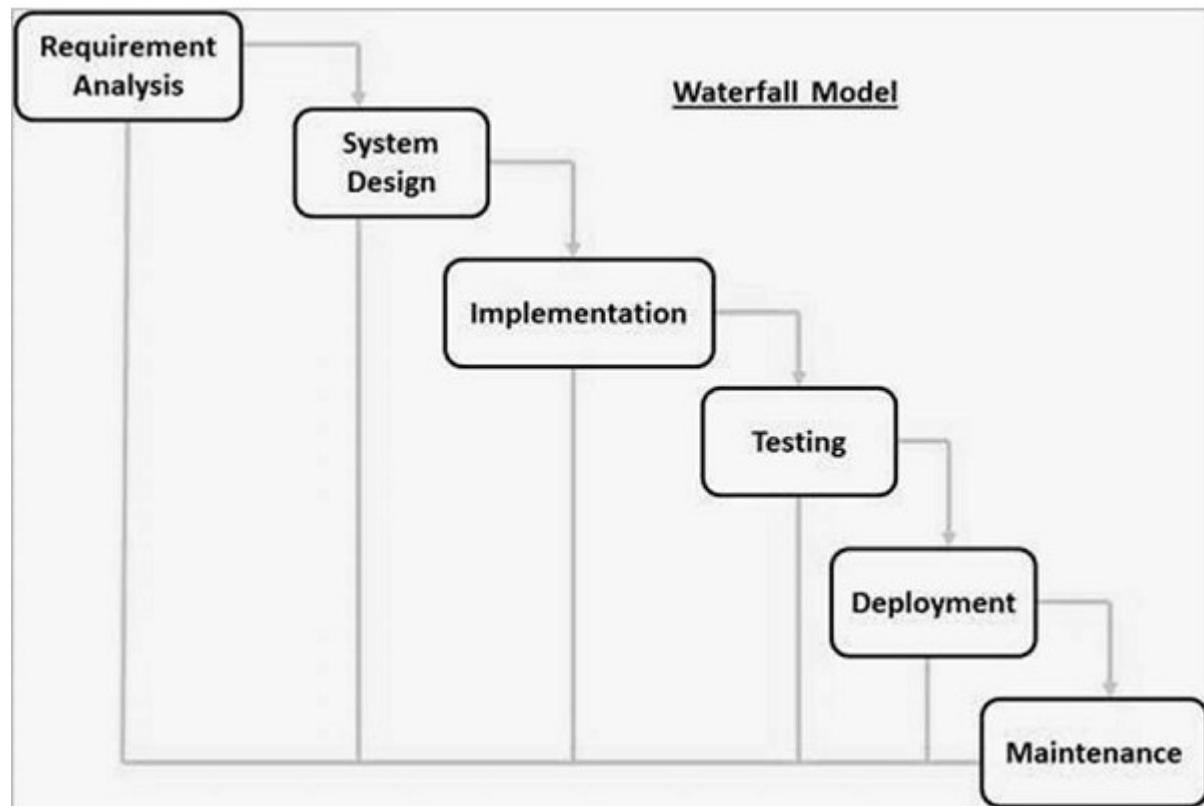
Prescriptive Process Models

- The waterfall model
- Incremental process models
 - The Incremental Model
 - The RAD Model
- Evolutionary process models
 - Prototyping
 - The Spiral Model
 - The Concurrent Development Model
- The specialized process models
 - Component-Based Development
 - The Formal Methods Model
 - Aspect-Oriented Software Development

The *waterfall model*

- The *waterfall model*, sometimes called the *Classic Life Cycle* or *Linear Sequential Model*, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment,

The *waterfall model*



The *waterfall model*

- Communication:-
 - It starts with communication between customer and developer.
 - According to this model customer must state all requirements at the beginning of project.
- Planning:-
 - It includes complete estimation(e.g cost estimation of project) and scheduling(complete timeline chart for project development) and tracking.

The *waterfall model*

- Modelling:-
 - It includes detail requirement analysis and project design(algorithm,flowchart etc).
 - Flowchart shows complete pictorial flow of program whereas algorithm is step-by-step solution of problem.
- Construction:-
 - It includes coding and testing phases
 - Coding-Design details are implemented using appropriate programming language
 - Testing-It is carried out to check whether flow of coding is correct,to check out the errors of program

The *waterfall model*

- Deployment-
 - It includes software delivery, support and feedback from customer

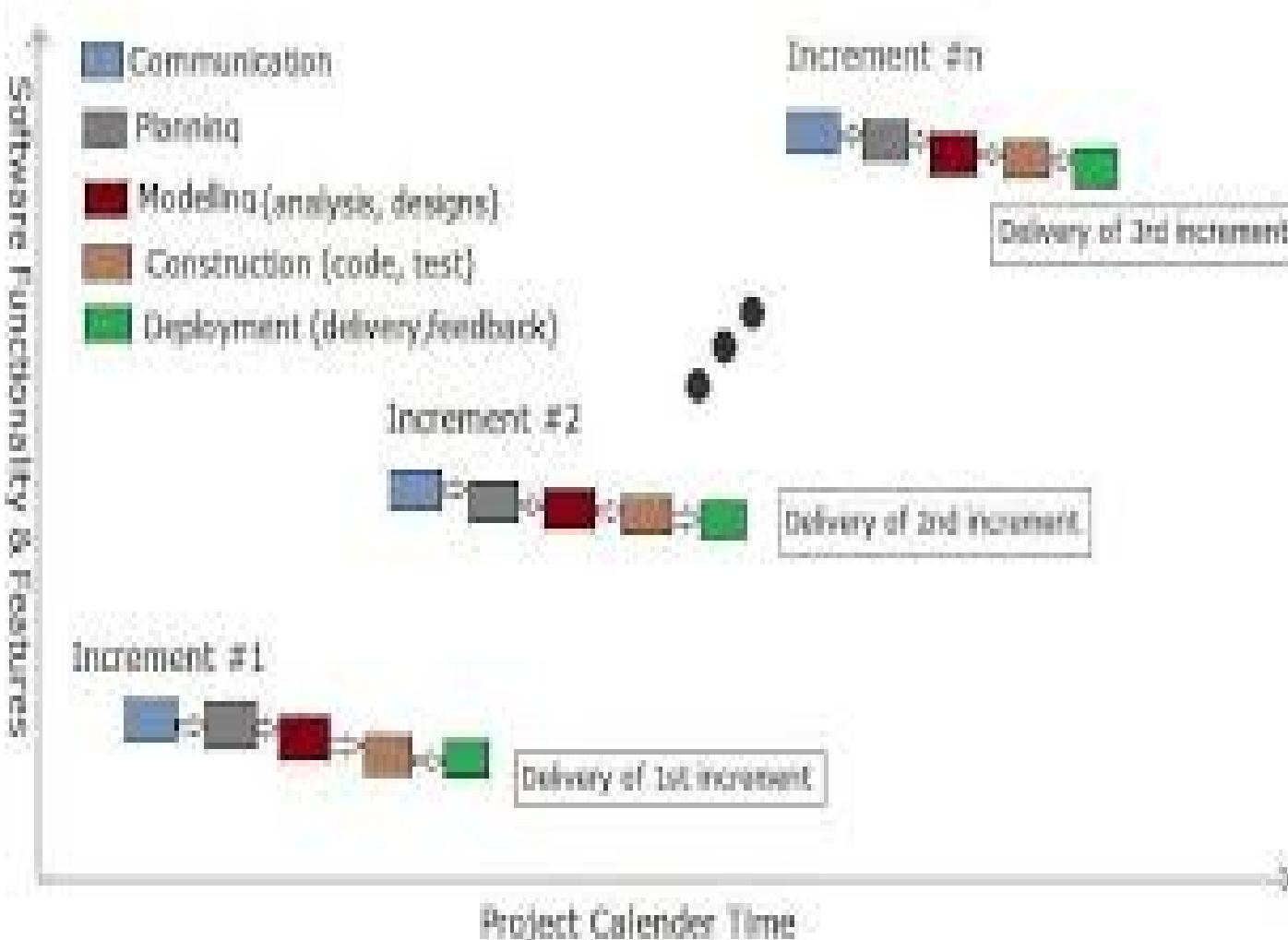
The incremental Model

- It combines the elements of the waterfall model applied in an interactive fashion
- Each linear sequence produces deliverable “increments” of the software
- For example, word processing software developed using incremental paradigm might deliver basic file management, editing and document production functions in the first increment; more sophisticated editing, and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment

The incremental Model

- In this model, the first increment is generally a core product
- That is, basic requirements are addressed, but many supplementary features (some known, others unknown) remain undelivered.
- The core product is used by the customer (or undergoes detailed evaluation)
- As a result of use and/or evaluation, a plan is developed for the next increment
- The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality
- This process is repeated following the delivery of each increment, until the complete product is produced

The incremental Model



RAD Model

- Rapid Application Developed(RAD) is an incremental software process model that emphasizes a short development cycle.
- The RAD model is a high-speed adaptation of the waterfall model,in which rapid development is achieved by using a component based construction approach
- The RAD process enables a development team to create a “fully functional system” within a very short time period(e.g.,60 to 90 days)
- Like other process models,the RAD approach maps into the generic framework activities

RAD Model

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

What does each team carries out the following steps

- Communication- It works to understand the business problem and the information characteristics that the software must accommodate
- Planning-It is essential because multiple software teams work in parallel on different system functions
- Modeling-It consists of 3 major phases-business modeling,data modeling and process modeling

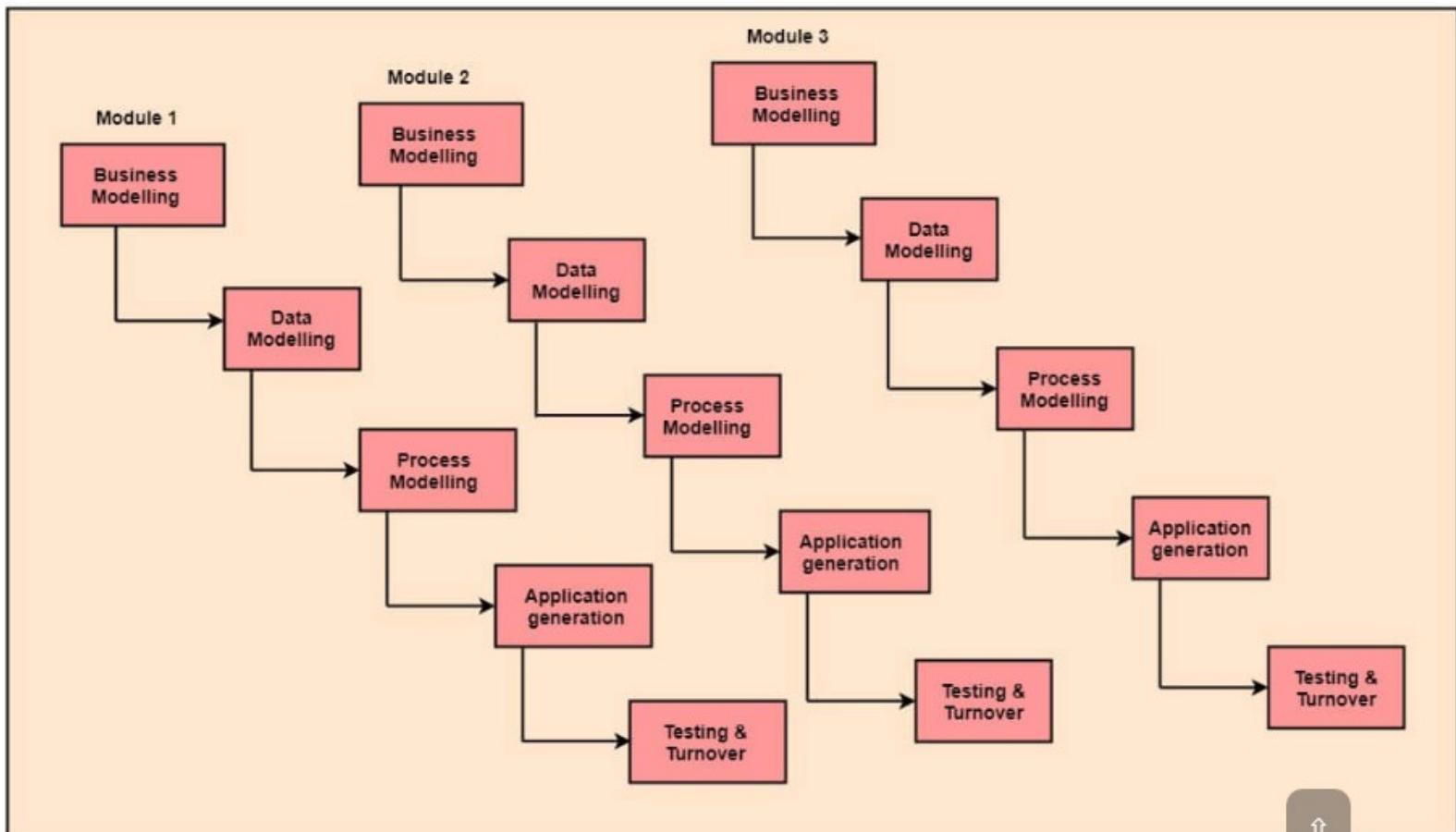
The RAD Model

- Modeling
- Modeling establishes design representations that serve as the basis for RAD's construction activity
 - Business Modeling-It includes information flow among different functions in the project e.g.,what information will be produced by each function,which functions handles that information etc.
 - Data Modeling-It includes different data objects used in software and relationship among different objects
 - Process Modeling-During process modeling,process descriptions are created e.g add,modify,delete etc

RAD Model

- Construction- It emphasizes the use of preexisting software components and the application of automatic code generation
- Deployment- It establishes a basis for subsequent iterations, if required

The RAD Model



Merits/Advantages

- Using the RAD approach,a product can be developed within very short period of time
- It supports increased reusability of the components
- It results in minimal code writing as it supports automatic code generation
- It encourages the customer feedback
- In this approach,quick initial reviews are possible
- Module integration is done from the beginning thus resolving number of integration issues

Drawbacks

- For large, but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams
- If developers and customers are not committed to the rapid-fire activities necessary to complete the system in a much abbreviated time frame, RAD projects will fail
- If a system cannot be properly modularized, building the components necessary for RAD will be problematic
- RAD may not be appropriate when technical risks are high(e.g., when a new application makes a heavy use of new technology)

Evolutionary process models

- Prototyping
- The Spiral Model
- The Concurrent Development Model

The Prototyping Paradigm

- It assists the software engineer and the customer to better understand what is to be built when requirements are fuzzy-not clear
- The prototyping paradigm begins with communication
- The software engineer and customer meet and define the overall objectives for the software, identify whatever requirements are known
- A prototyping iteration is planned quickly and modeling(in the form of a “quick design”)occurs
- A quick design focuses on a representation of those aspects of the software that will be visible to the customer/end-user(e.g human interface layout or output display formats)
- The quick design leads to the construction of a prototype

The Prototyping Paradigm

- The prototype is deployed and then evaluated by the customer/user
- Feedback is used to refine requirements for the software
- Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done
- Ideally, the prototype serves as a mechanism for identifying software requirements

The Prototyping Paradigm

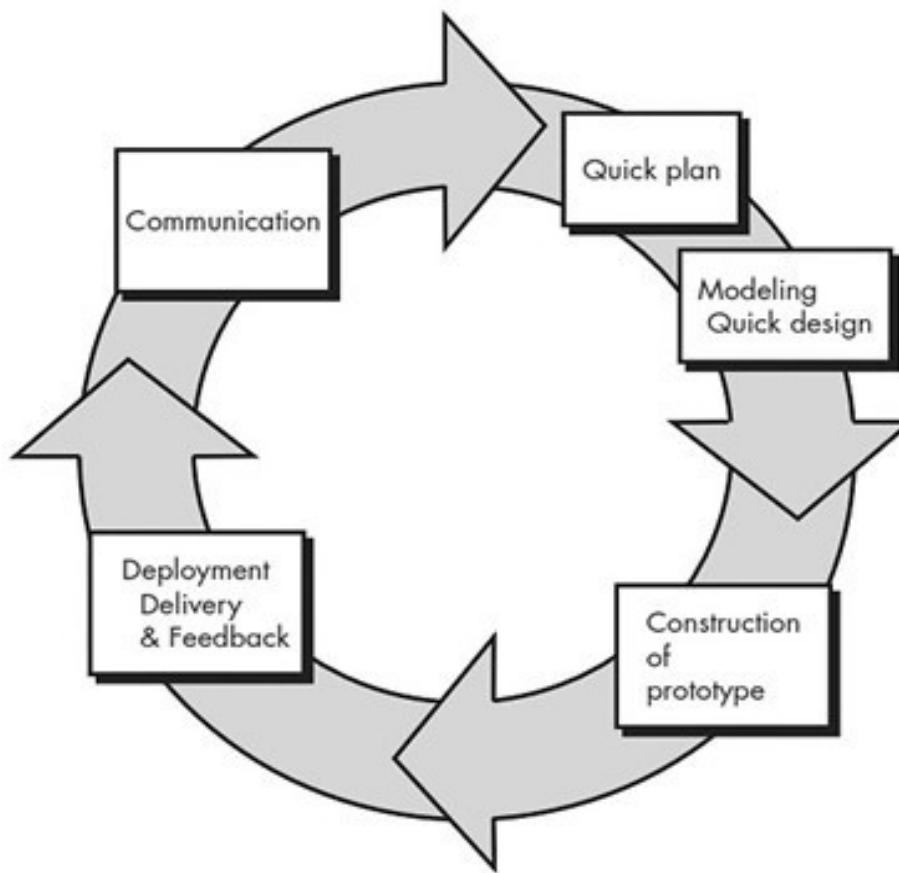
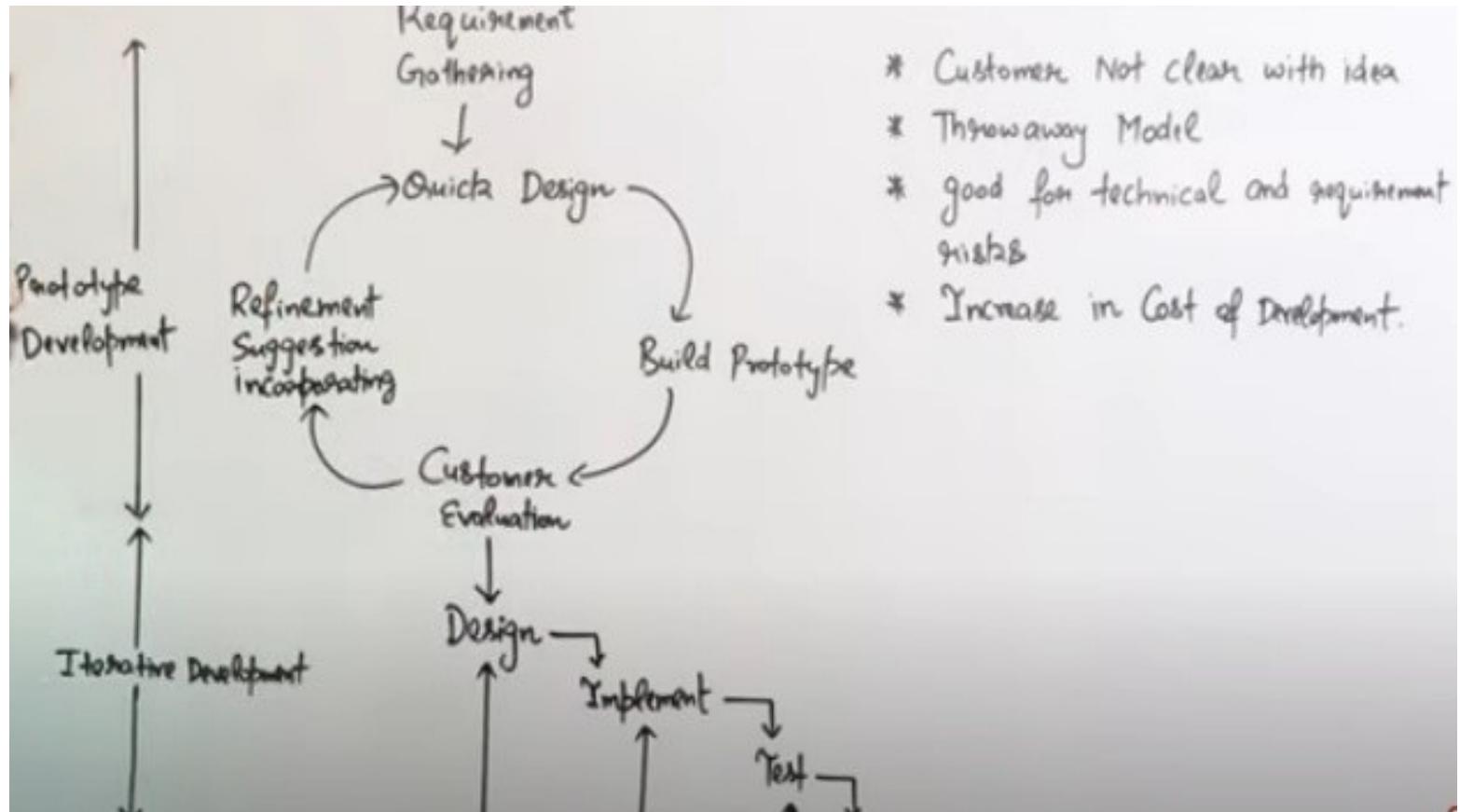
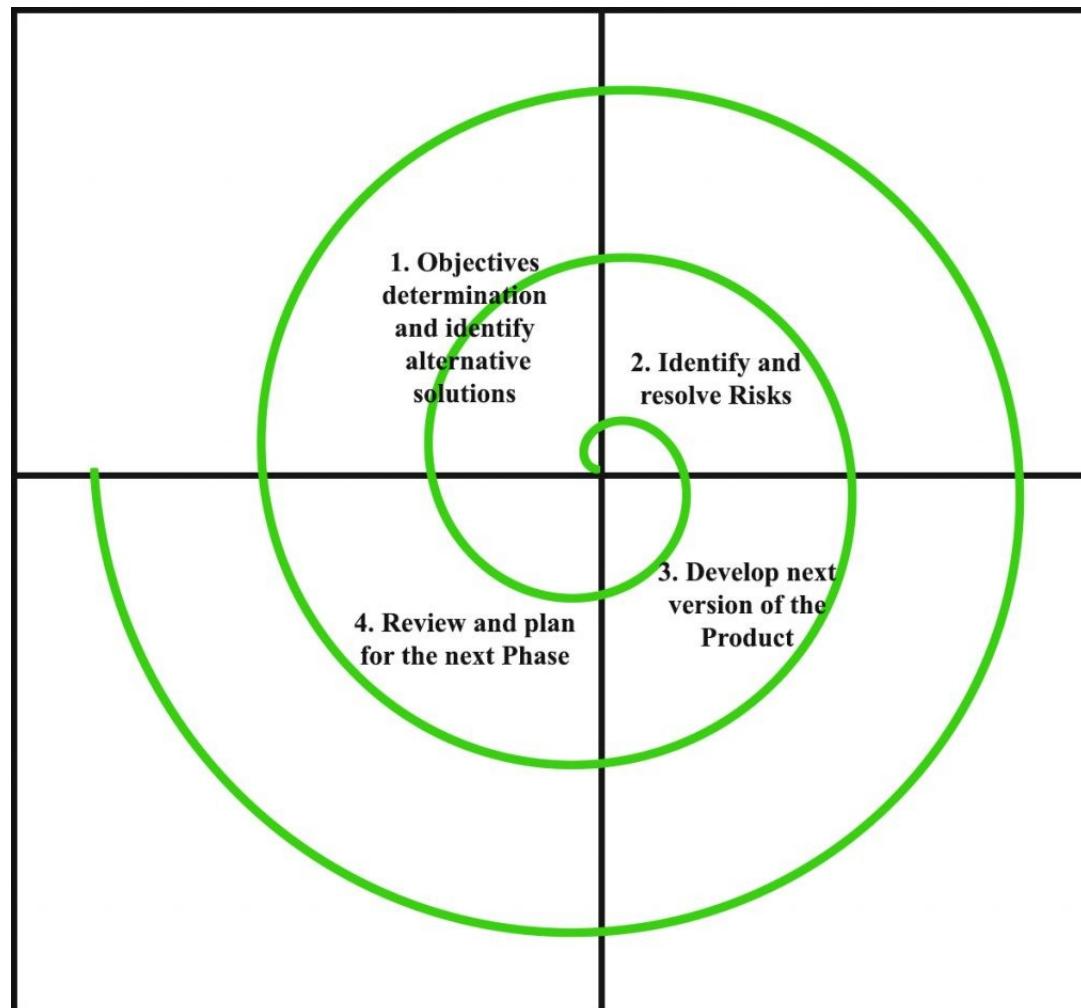


Figure: Prototype Model

The Prototyping Paradigm



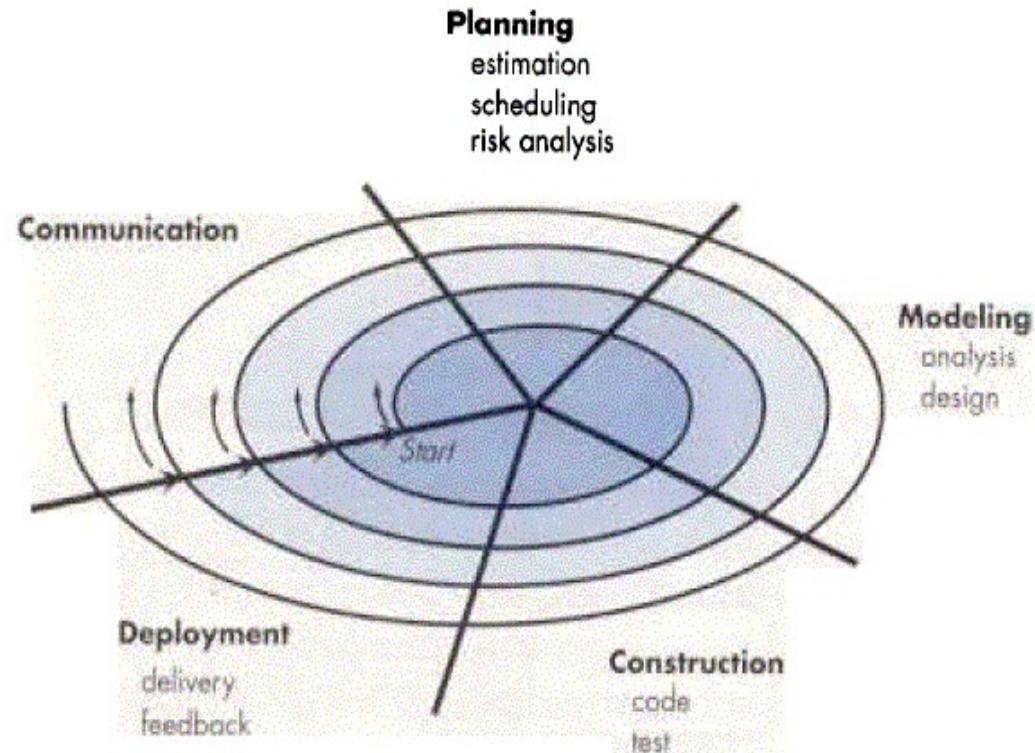
Spiral Model



Spiral Model

FIGURE 3.5

A typical
spiral model



Spiral Model

- The spiral model, originally proposed by Boehm, is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model
- It provides the potential for rapid development of increasingly more complete versions of the software
- The software is developed as series of evolutionary releases
- During the initial releases, it may be just paperwork or prototype. But during later releases the version goes towards more completed stage

Spiral Model

- A spiral model is divided into a set of the framework activities defined by the software engineering team
- Each of the framework activities represent one segment of the spiral path as shown in fig.
- As this evolutionary process begins, the software team performs activities that are implied by a circuit around the spiral in a clockwise direction, beginning at the center

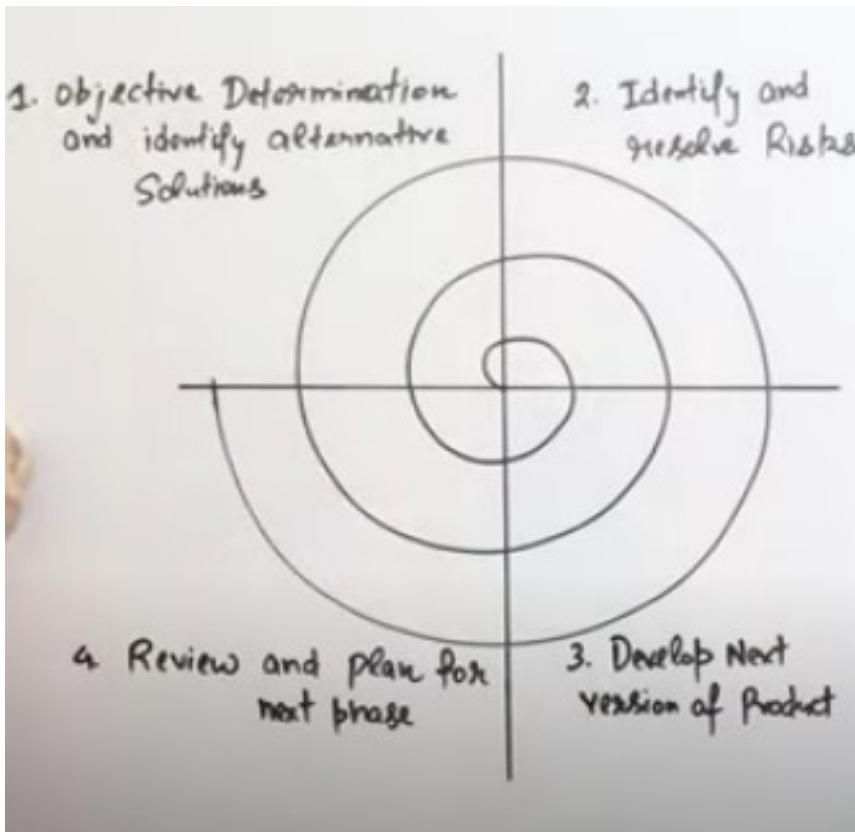
Spiral Model

- The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software

Spiral Model

- The spiral model is a realistic approach to the development of large-scale systems and software
- It uses prototyping as a risk reduction mechanism
- It maintains the systematic stepwise approach suggested by the classic life cycle but incorporates it into an iterative framework that more realistically reflects the real world

Spiral Model



- * Risk Handling
- * Radius of spiral = Cost
- * Angular Dimension = Progress
- * Meta model

Advantages

- 1) Risk Handling
- 2) Large Projects
- 3) Flexible
- 4) Customer satisfaction

Disadvantages

- 1) Complex
- 2) Expensive
- 3) Too much Risk Analysis
- 4) Time



Difference between Waterfall Model and Spiral Model

- Waterfall model works in sequential method.
- In waterfall model errors or risks are identified and rectified after the completion of stages.
- Waterfall model is applicable for small project.
- In waterfall model requirements and early stage planning is necessary.
- Flexibility to change in waterfall model is Difficult.
- While spiral model works in evolutionary method.
- In spiral model errors or risks are identified and rectified earlier.
- While Spiral model is used for large project.
- While in spiral model requirements and early stage planning is necessary if required.
- Flexibility to change in spiral model is not Difficult.

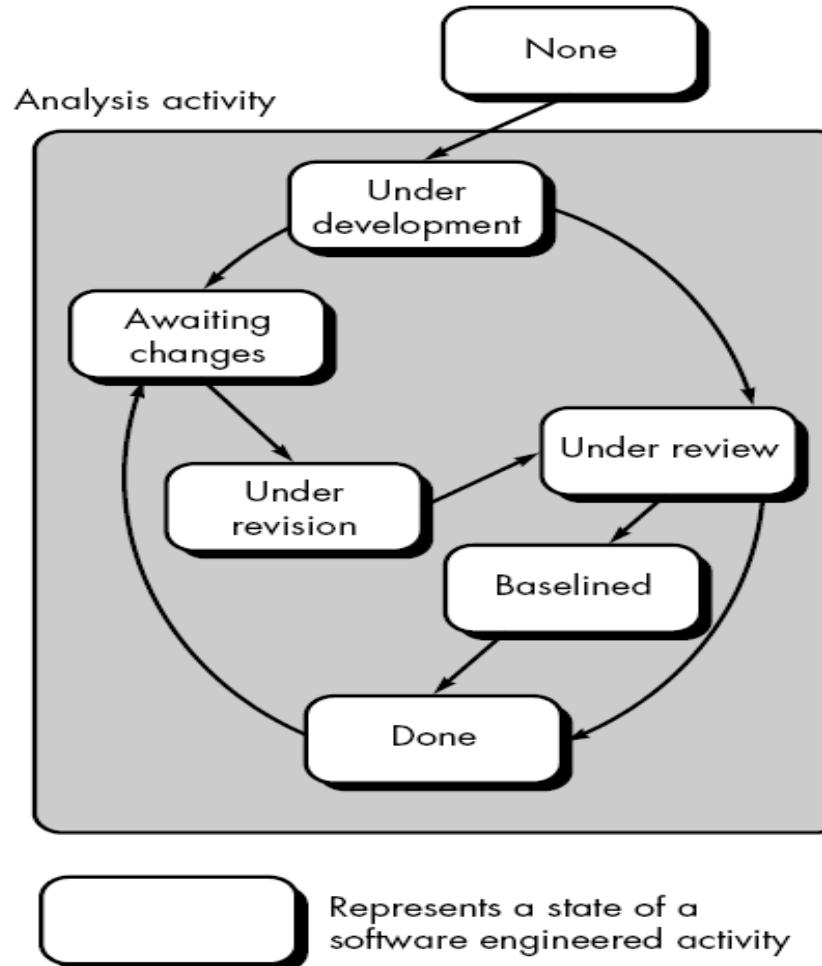
Difference between Waterfall Model and Spiral Model

- There is high amount risk in waterfall model.
- Waterfall model is comparatively inexpensive.
- There is low amount risk in spiral model.
- While cost of spiral model is very expensive.

The Concurrent Development Model

- The concurrent development model,sometimes called concurrent engineering,can be represented as a series of framework activities,software engineering actions and tasks, and their associated states
- This model is more appropriate for system engineering projects where different engineering teams are involved

Fig shows one element of the concurrent process model i.e. modeling activity



The Concurrent Development Model

- The modeling activity is in one of the states and other activities like communication or construction can also be represented in an analogous manner
- Let the communication activity has completed its first iteration and available in awaiting changes state
- The modeling activity has completed its initial communication and ready to move to under development state from none state

The Concurrent Development Model

- During these transitions, if customer indicates some modification in the requirement, then the modelling activity transits to awaiting changes state from under development state
- It defines network of activity instead of considering activities, actions and tasks as sequence of events
- This model is applicable to all types of software development and provides an accurate picture of the current state of a project

The Concurrent Development Model

- A baseline is a fixed point of reference that is used for comparison purposes. In business, the success of a project or product is often measured against a baseline number for costs, sales, or any number of other variables. A project may exceed a baseline number or fail to meet it.

Concurrent Development Model: Contd...

Merits:

- Applicable to all types of S/W development & provides an accurate picture of the current state of the project.**

Demerits:

- Problem to Project planning. How many No of iterations are to be planned? Uncertainty...**
- Process may fall in chaos if the evolutions occurs too fast without a period of relaxation. On the other hand if the speed is too slow productivity could be affected.**
- S/W processes are focussed on flexibility & extendability, rather than on high quality.**

Specialized Process Models:

far,

Specialized process models use many of the characteristics of one or more of the conventional models presented so far, however they tend to be applied when a narrowly defined software engineering approach is chosen. They include,

- Components based development
- The Formal Methods Model
- Aspect oriented software development

Components Based Development :

In this approach, Commercial Off-The-Shelf (COTS) S/w components, developed by vendors who offer them as products are used in the development of software.

These components provide targeted functionality with well-defined interfaces that enable the components to be integrated into the software

The component-based development model incorporated many of the characteristics of the spiral model.

Components Based Development :

- This model incorporates the following steps
 - Available component-based products are researched and evaluated for the application domain
 - Component integration issues are considered
 - A software architecture is designed to accommodate the components
 - Components are integrated into the architecture
 - Comprehensive testing is conducted to ensure proper functionality

Components Based Development :

- **Merits:**

- Leads to software reuse, which provides number of benefits
 - 70% reduction in development cycle time
 - 84 % reduction in project cost
 - Productivity index goes up to 26.2 (Norm : 16.9)

- **Demerits:**

- Component Library must be robust.
- Performance may degrade

The Formal Methods Model:

- The formal methods model encompasses a set of activities that lead to formal mathematical specification of computer software.
- Formal methods enable a software engineer to specify, develop, and verify a computer based system by applying a rigorous, mathematical notation
- The main focus is on defect prevention rather than defect removal
- Example, Clean Room S/W Engineering (CRSE)

The Formal Methods Model:

- **Merits:**
 - Removes many of the problems that are difficult to remove using other S/W Engg. Paradigms.
 - Ambiguity, Incompleteness & Inconsistency can be discovered & corrected easily by using formal methods of mathematical analysis.
- **Demerits:**
 - Development is time consuming & expensive
 - Extensive training is required
 - Difficult to use with technically unsophisticated customers

Aspect Oriented Software Development (AOSD):

- A set of localized features, functions & information contents are used while building complex software.
- These localized s/w characteristics are modeled as components (e.g. OO classes) & then constructed within the context of a system architecture.
- Certain “concerns” (Customer required properties or areas of technical interest) span the entire architecture i.e. Cross cutting concerns like system security, fault tolerance etc.

Merits:

- It is similar to component based development for aspects

Demerits:

- Component Library must be robust.
- Performance may degrade

What is “Agility”?

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed

Yielding ...

- Rapid, incremental delivery of software

An Agile Process

- Is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers multiple ‘software increments’
- Adapts as changes occur

An Agile Process

- These models satisfy customer through early and continuous delivery
- It can accommodate changing requirements
- In this development process customer,business people and developer must work together daily
 - The messages are conveyed orally i.e.face-to-face.Conversation is essential
 - Technical excellence and good design is achieved

Principles of Agility

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome/Accommodate changing requirements, even late in development.
- Deliver working software frequently,in shorter time span.
- The customer,Business people and developers must work together daily throughout the project.

Principles of Agility

- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Principles of Agility

- Continuous attention to technical excellence and good design enhances agility.
- There must be simplicity in development
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Models

- Crystal
- Atern
- Feature-driven development
- Scrum
- Extreme programming (XP)
- Lean development
- Unified process
- Kanban

Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck
- XP Planning
 - Begins with the creation of a set of stories(also called user stories)that describe required features and functionality for software to be built
 - Each story is written by the customer and is placed on an index card
 - The customer assigns a value(i.e. a priority) to the story based on the overall business value of the feature or function
 - Members of the XP team then assess each story and assign a cost-measured in development weeks-to it
 - If the story will require more than 3 development weeks,the customer is asked to split the story into smaller stories ,and assignment of value and cost occurs again
 - Stories are grouped to for a deliverable increment
 - A commitmentt is made on delivery date
 - After the first increment project velocity is used to help define subsequent delivery dates for other increments
 - Project velocity is the number of customer stories implemented during the first release

Extreme Programming (XP)

- XP Design

- Follows the KIS(Keep it simple) principle
- A simple design is always preferred over a more complex representation
- Design provides implementation guidance for a story as it is written
- Encourage the use of CRC(class-responsibility collaborator) cards
- CRC cards is an effective mechanism for thinking about the software in an object-oriented context
- CRC cards identify and organize the object-oriented classes that are relevant to the current software increment
- For difficult design problems, suggests the creation of spike solutions — a design prototype-the immediate creation of an operational prototype of that portion of the design
- Encourages refactoring — an iterative refinement of the internal program design
- Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves the internal structure

Extreme Programming (XP)

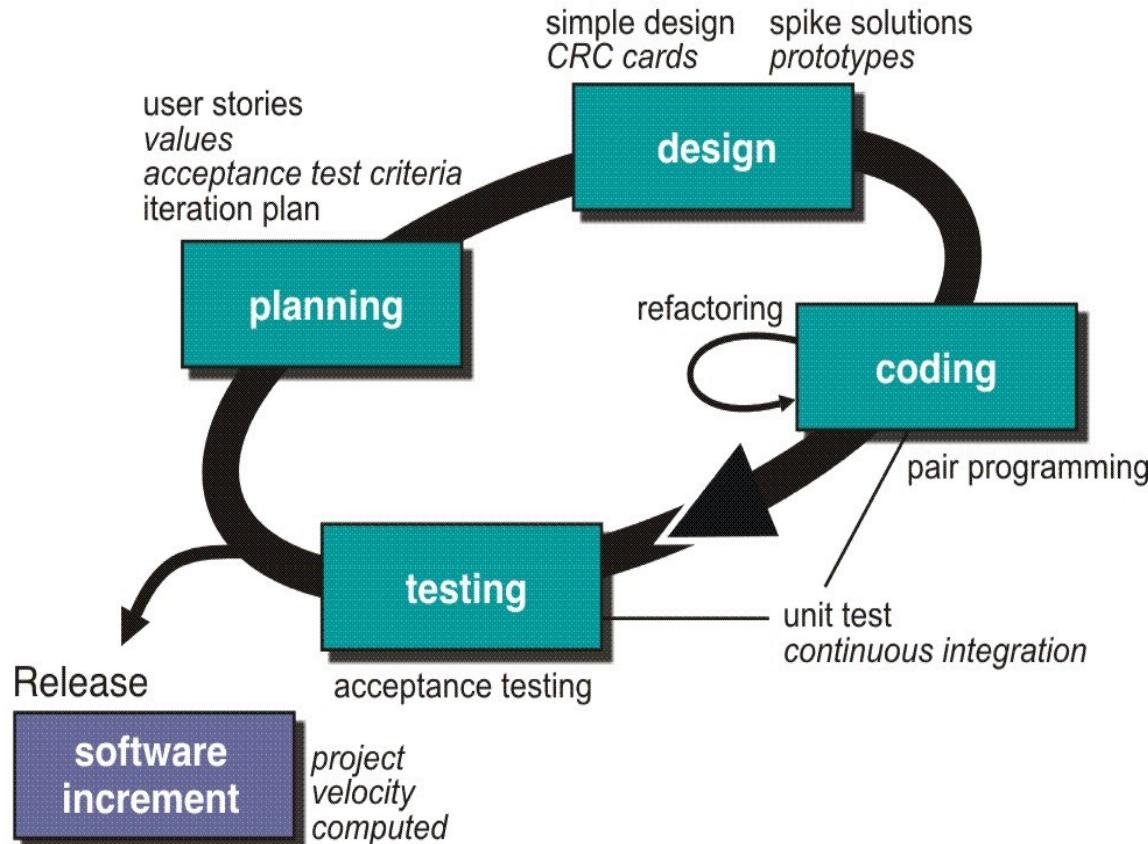
- XP Coding

- Recommends the construction of a unit test for a story *before* coding commences
- Once the unit test has been created, the developer is better able to focus on what must be implemented to pass the unit test
- Once the unit test has been created, the developer is better able to focus on what must be implemented to pass the unit test
- Encourages pair programming – It recommends that two people work together at one computer workstation to create code for a story
- For example, one person might think about the coding details of a particular portion of the design while the other ensures that coding standards are being followed

Extreme Programming (XP)

- XP Testing
 - All unit tests are executed daily
 - Acceptance tests, are called customer tests, defined by the customer and focus on overall system features and functionality that are visible and reviewable by the customer

Extreme Programming (XP)



Scrum

- Scrum is an agile process model developed by Jeff Sutherland and his team in the early 1990s
- Scrum principles
 - Small working teams are organized to “maximize communication,minimize overhead, and maximize sharing of tacit,informal knowledge.”
 - The process must be adaptable to both technical and business changes “to ensure the best possible product is produced.”
 - The process yields frequent software increments “that can be inspected,adjusted,tested,documented, and built on.”

Scrum

- Development work and the people who perform it are partitioned “into clean, low coupling partitions, or packets.”
- Constant testing and documentation is performed as the product is built
- It provides the “ability to declare a product ‘done’ whenever required”

Scrum

- Scrum process pattern development activities
 - Backlog-a prioritized list of project requirements or features that provide business value for the customer.Items can be added to the backlog at any time(this is how changes are introduced.The product manager assesses the backlog and updates priorities as required
 - Sprints-consist of work units that are required to achieve a requirement defined in the backlog that must fit into a predefined time-box(typically 30 days).During the sprint,the backlog items that the sprint work units address are frozen(i.e.,changes are not introduced during the sprint).

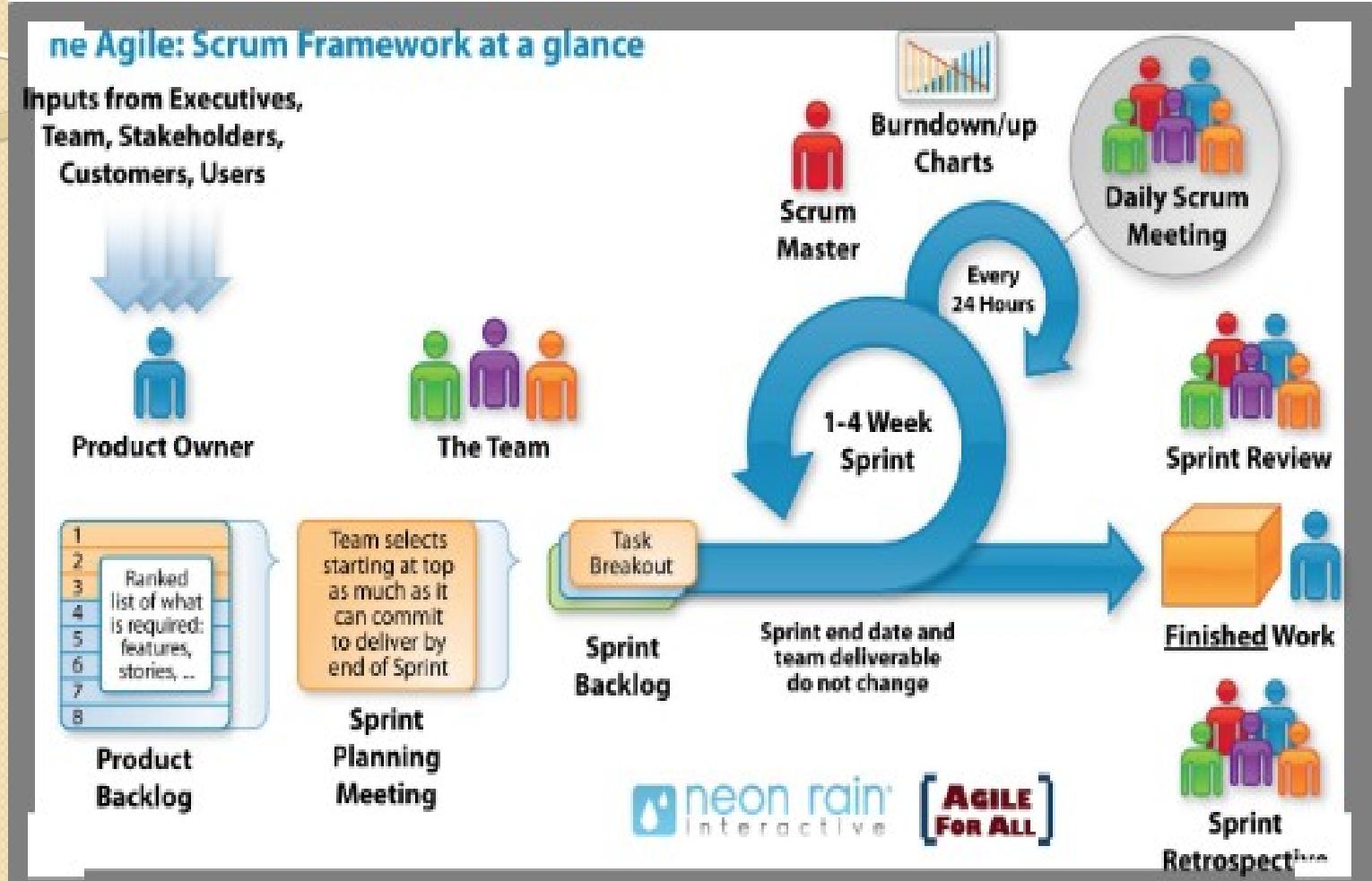
Scrum

- Scrum meetings-are short(typically15 minutes)meetings held daily by the scrum team
- 3 key questions are asked and answered by all team members
 - What did you do since the last team meeting?
 - What obstacles are you encountering?
 - What do you plan to accomplish by the next team meeting?
- A team leader,called a “Scrum master”,leads the meeting and assesses the responses from each person

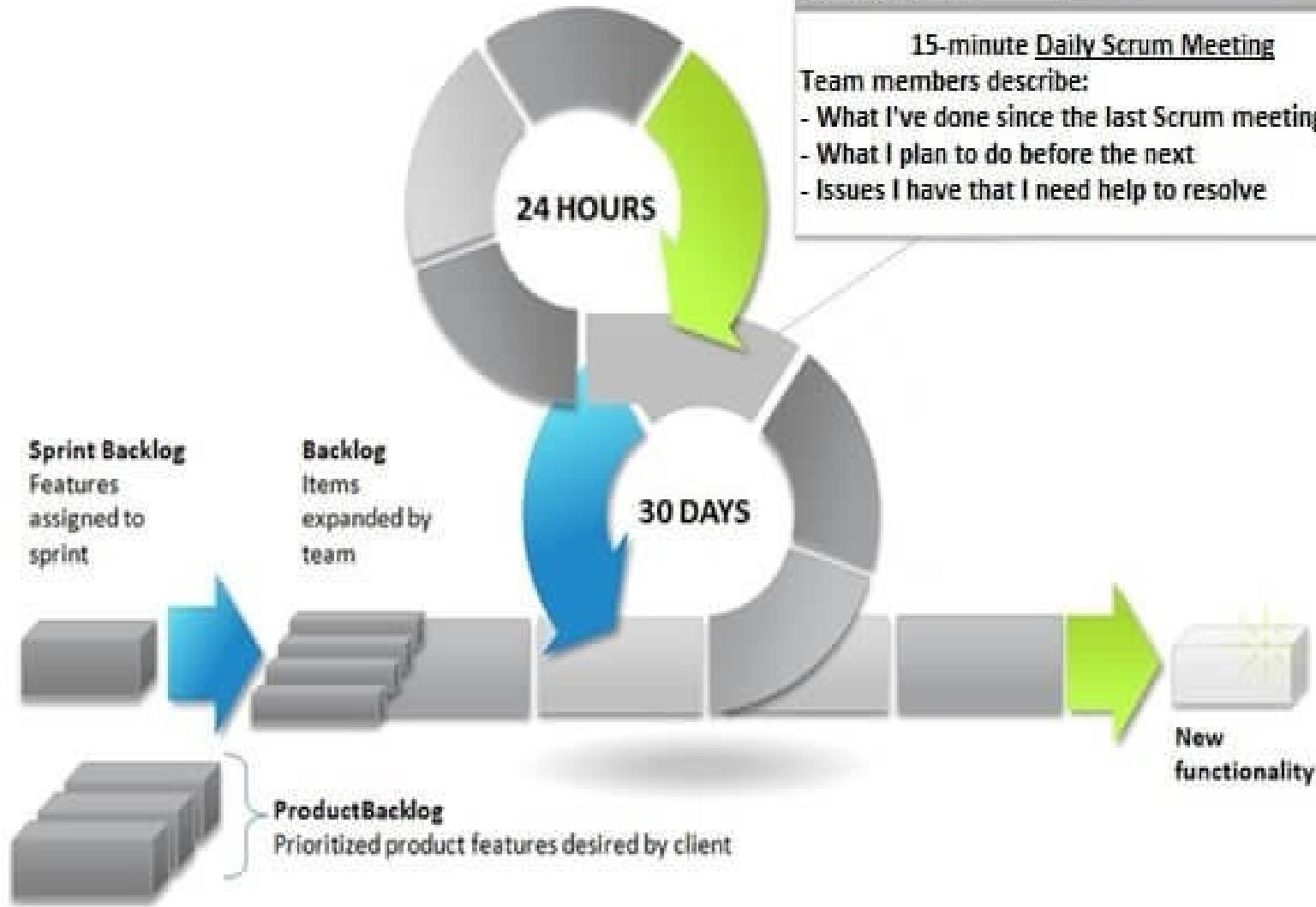
Scrum

- Demos-deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer

Scrum

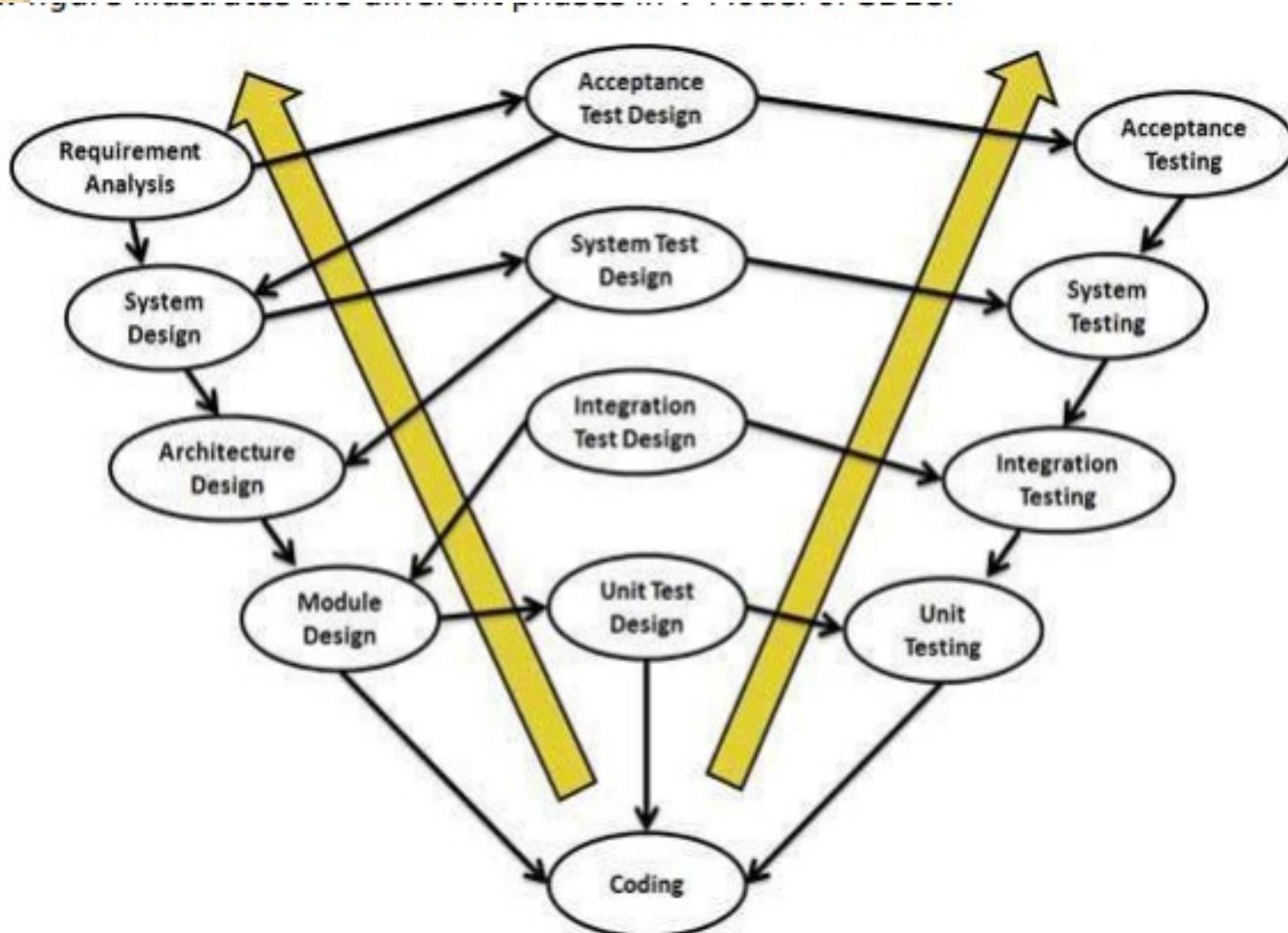


SCRUM PROCESS



The V - model

- The V - model is SDLC model where execution of processes happens in a sequential manner in V shape. It is also known as **Verification** and **Validation** model.
- V - Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase.
- This is a highly disciplined model and next phase starts only after completion of the previous phase.



- Verification phases on one side of the .V. and Validation phases on the other side

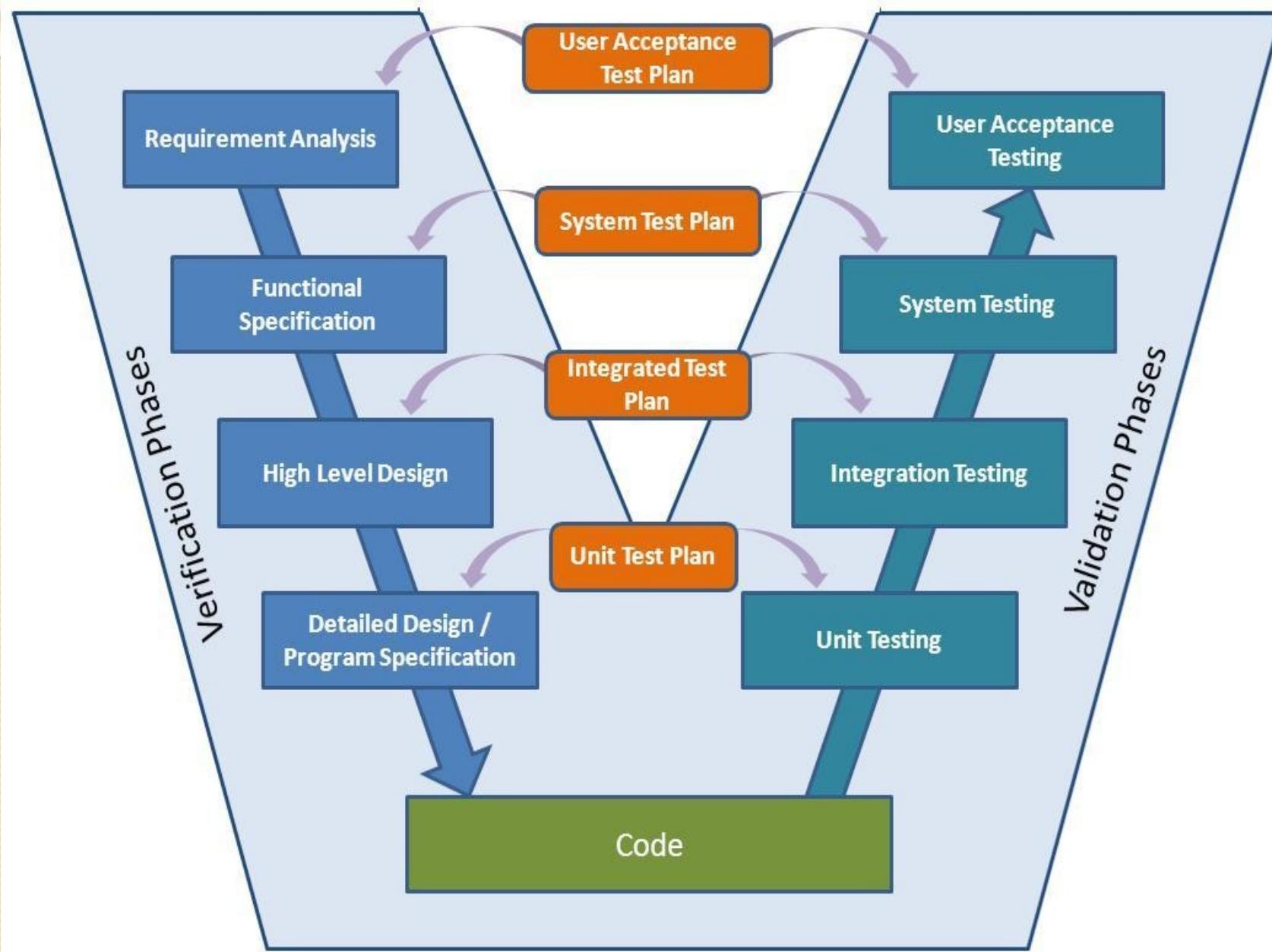
Verification Phases

- **Business Requirement Analysis:** This phase involves detailed communication with the customer to understand his expectations and exact requirement.
- **System Design :** System design would comprise of understanding and detailing the complete hardware and communication setup for the product under development.
- **Architectural Design(Detail Design) :** The data transfer and communication between the internal modules and with the outside world other systems is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

Verification Phases

- **Module Design :** In this phase the detailed internal design for all the system modules is specified, referred to as Low Level Design LLD.

V- Model



Advantages of the V-Model

- This is a highly-disciplined model and Phases are completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Simple and easy to understand and use.
- Each phase has specific deliverables and a review process.

Disadvantages of the V-Model

- High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.
- Once an application is in the testing stage, it is difficult to go back and change a functionality.
- No working software is produced until late during the life cycle.

Kanban Model

- “Kanban” is a word of Japanese origin. It consists of two words: “Kan” – visible; and “Ban” – board.
- The Kanban board is a way of your project representation that allows tracking the current status.
- The main purpose of representing work as a card on the kanban board is to allow team members to track the progress of work through its workflow in a highly visual manner

Kanban Model

- Agile Kanban is Agile Software Development with Kanban approach.
- In Agile Kanban, the Kanban board is used to visualize the workflow. The Kanban board is normally put up on a wall in the project room.
- The status and progress of the story development tasks is tracked visually on the Kanban board with flowing Kanban cards.

Kanban Board

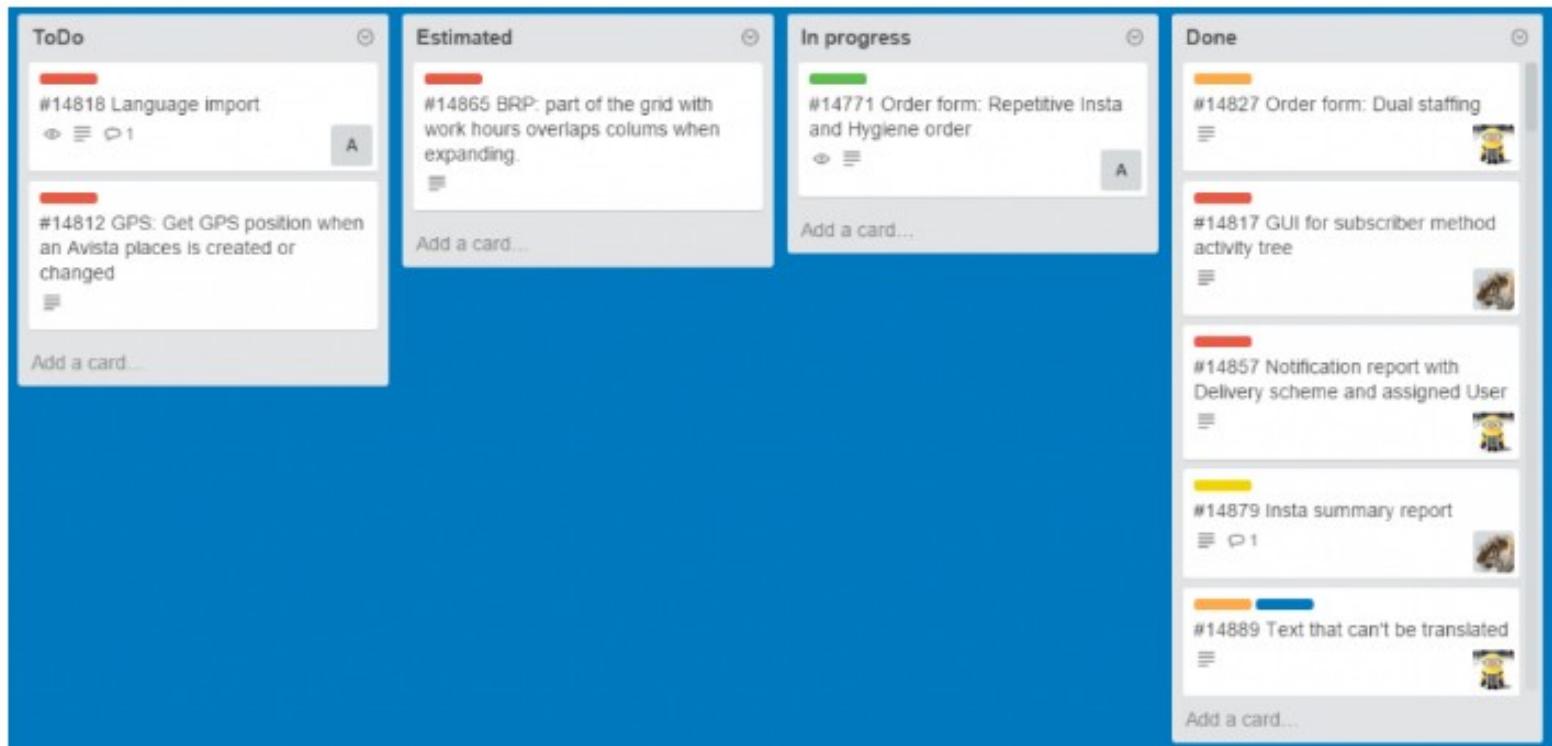
- Kanban board is used to depict the flow of tasks across the value stream.
- The Kanban board –
- Provides easy access to everyone involved in the project.
- Facilitates communication as and when necessary.
- Progress of the tasks are visually displayed.
- Bottlenecks are visible as soon as they occur.

Advantages of Kanban board

- The major advantages of using a Kanban board are –
- **Empowerment of Team** – This means –
 - Team is allowed to take decisions as and when required.
 - Team collaboratively resolves the bottlenecks.
 - Team has access to the relevant information.
 - Team continually communicates with customer.
- **Continuous Delivery** – This means –
 - Focus on work completion.
 - Limited requirements at any point of time.
 - Focus on delivering value to the customer.
 - Emphasis on whole project.

Kanban Board Example

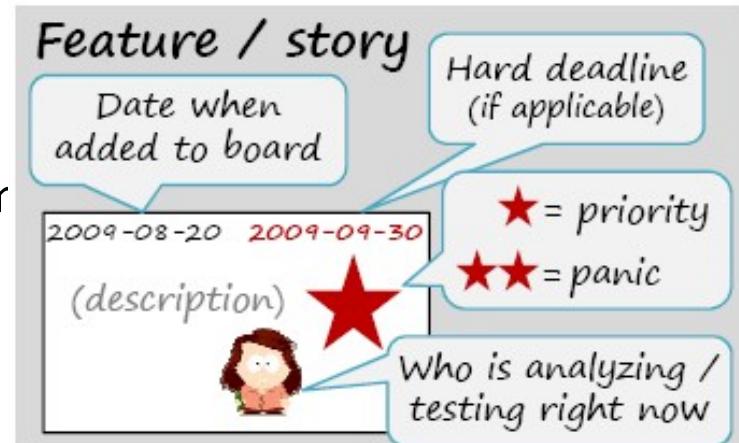
Kanban Board Example



Kanban (contd...)

Visualize the workflow

- Split the work into pieces, write each item on a card and put on the wall
- Use named columns to illustrate where each item is in the workflow



Limit WIP (work in progress)

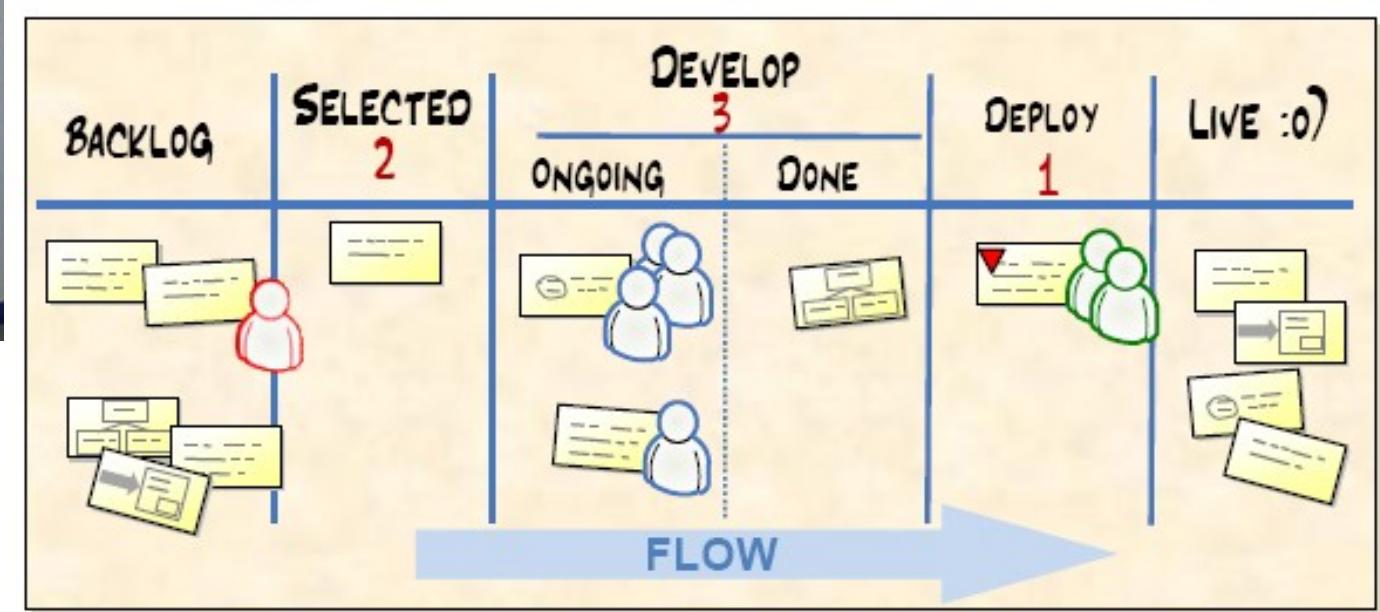
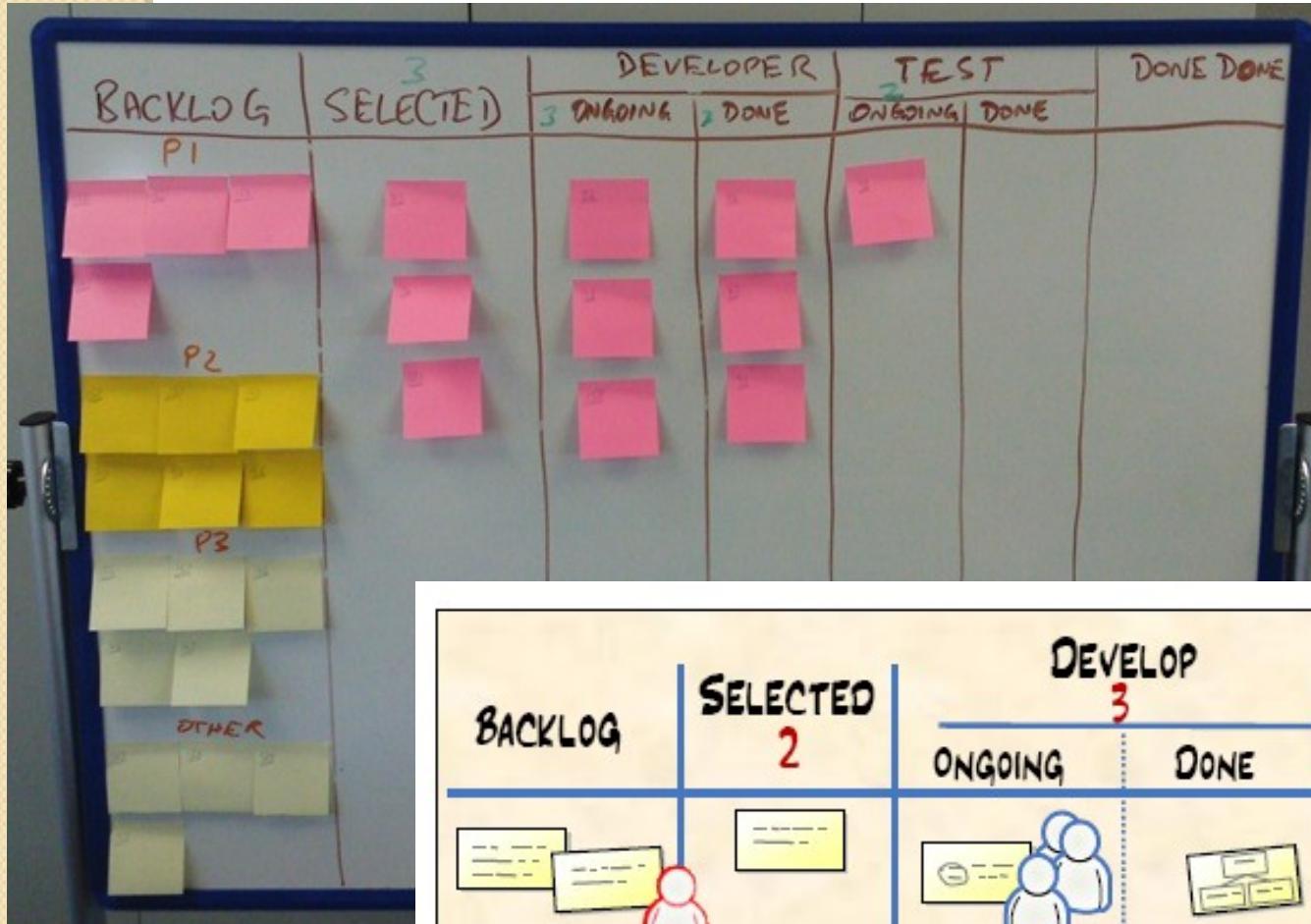
- Assign explicit limits to how many items may be in progress at each stage



Measure the lead time (average time to complete one item, sometimes called “cycle time”)

- Optimize the process to make lead time as small and predictable as possible

Kanban Board Illustration - I



Kanban Board

- every task has its card. In this case, the board was divided into four parts:
- ToDo section contains tasks that were received from the customer and required to be analyzed. Each task is marked with color according to its priority.
- When tasks from the first section have been analyzed and estimated by the Team, they are moved to the Estimated section.
- When the developer takes a task to be developed, he moves it from Estimated to In Progress section and marks it with his tag to show who handles each task.
- When a task is done, it's moved to the Done section.

Kanban Model

- WIP Limit
- The label in the Doing/In-process column also contains a number, which represents the maximum number of tasks that can be in that column at any point of time. i.e., the number associated with the **Doing** column is the WIP (Work-In-Progress) Limit.
- Pull Approach
- Pull approach is used as and when a task is completed in the Doing column. Another card is pulled from the To Do column.

CMM Models

- CMM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.
- It is not a software process model. It is a framework which is used to analyse the approach and techniques followed by any organization to develop software products.
- It also provides guidelines to further enhance the maturity of the process used to develop those software products.

CMM Models

- It is based on profound feedback and development practices adopted by the most successful organizations worldwide.
- This model describes a strategy for software process improvement that should be followed by moving through 5 different levels.
- Each level of maturity shows a process capability level. All the levels except level-1 are further described by Key Process Areas (KPA's).

What is CMM?

- CMM: Capability Maturity Model
- Also called as SEI-CMM
- Developed by the Software Engineering Institute (SEI) of the Carnegie Mellon University
- Framework that describes the key elements of an effective software process.

What is CMM?

- Describes an evolutionary improvement path for software organizations from an ad hoc, immature process to a mature, disciplined one.
- Provides guidance on how to gain control of processes for developing and maintaining software and how to evolve toward a culture of software engineering and management excellence.

Process Maturity

Concepts

- Software Process
 - set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, user manuals)
- Software Process Capability
 - describes the range of expected results that can be achieved by following a software process
 - means of predicting the most likely outcomes to be expected from the next software project the organization undertakes

Process Maturity Concepts

- Software Process Performance
 - actual results achieved by following a software process
- Software Process Maturity
 - extent to which a specific process is explicitly defined, managed, measured, controlled and effective
 - implies potential growth in capability
 - indicates richness of process and consistency with which it is applied in projects throughout the organization

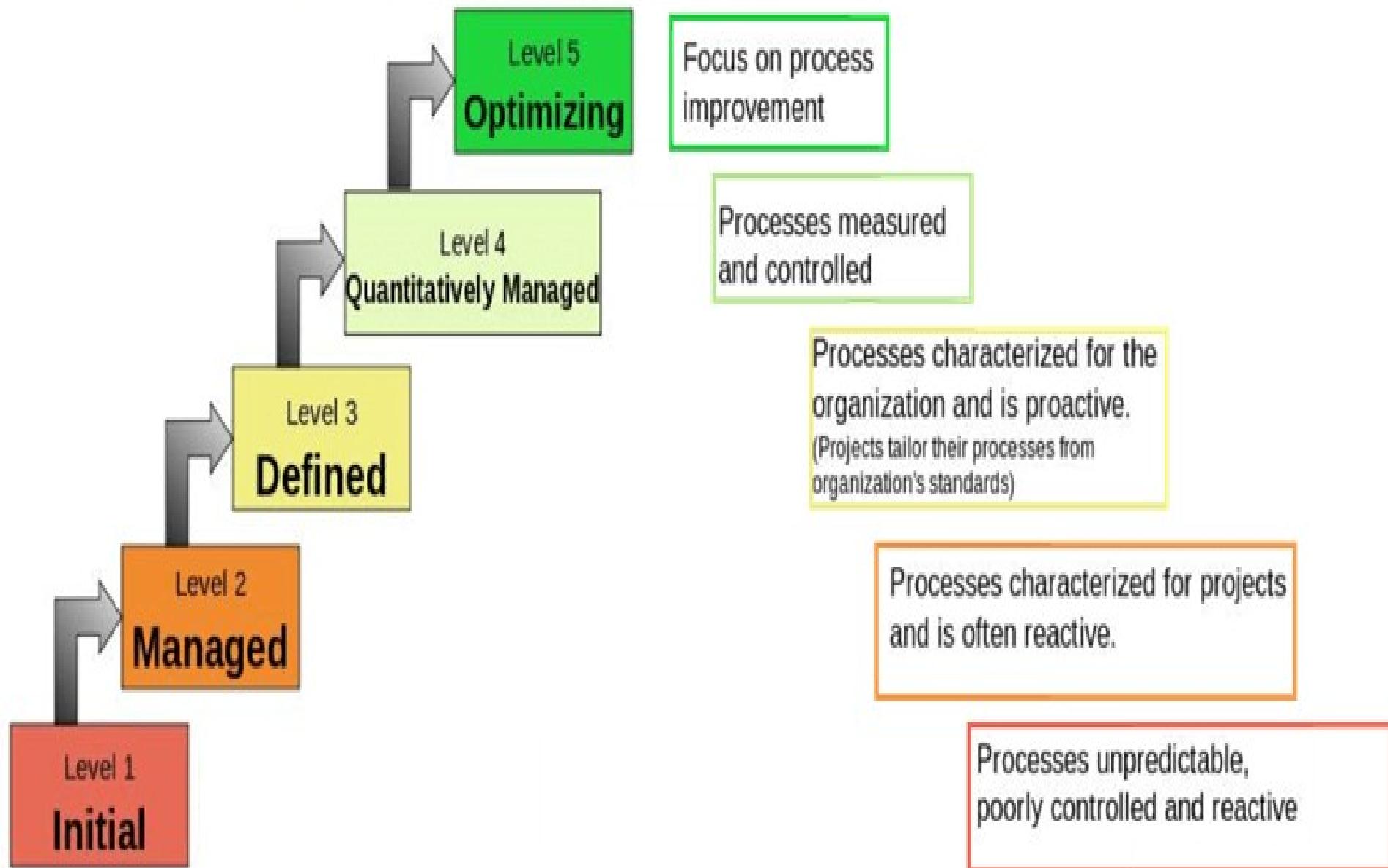
What are the CMM Levels?

(The five levels of software process maturity)

Maturity level indicates level of process capability:

- Initial
- Repeatable
- Defined
- Managed
- Optimizing

Levels of CMM



Software Project Maturity Model				
		Level	Capability	Result
Maturity Level	Process Focus	1 Initial	Design Code Compile Test	Risk & Waste
		2 Repeatable	Requirements Management Software Project Planning Software Project Tracking & Oversight Software Subcontract Management Software Quality Assurance Software Configuration Management	
		3 Defined	Organizational Process Focus Organizational Process Definition Training Program Integrated Software Management Software Product Engineering Intergroup Coordination Peer Reviews	
		4 Managed	Quantitative Process Management Software Quality Management	
		5 Optimizing	Continuous Improvement Defect Prevention Technology Change Management Process Change Management	Productivity & Quality

The 5 levels of CMM are as follows:

- **Level-1: Initial -**
- No KPA's defined
- Processes followed are adhoc and immature and are not well defined
- Unstable environment for software development
- No basis for predicting product quality, time for completion, etc.

Level-2: Repeatable -

- Focuses on establishing basic project management policies
- Experience with earlier projects is used for managing new similar natured projects

Level-3: Defined -

- At this level, documentation of the standard guidelines and procedures takes place.
- It is a well defined integrated set of project specific software engineering and management processes.

Level-4: Managed -

- At this stage, quantitative quality goals are set for the organization for software products as well as software processes.
- The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.

Level-5: Optimizing -

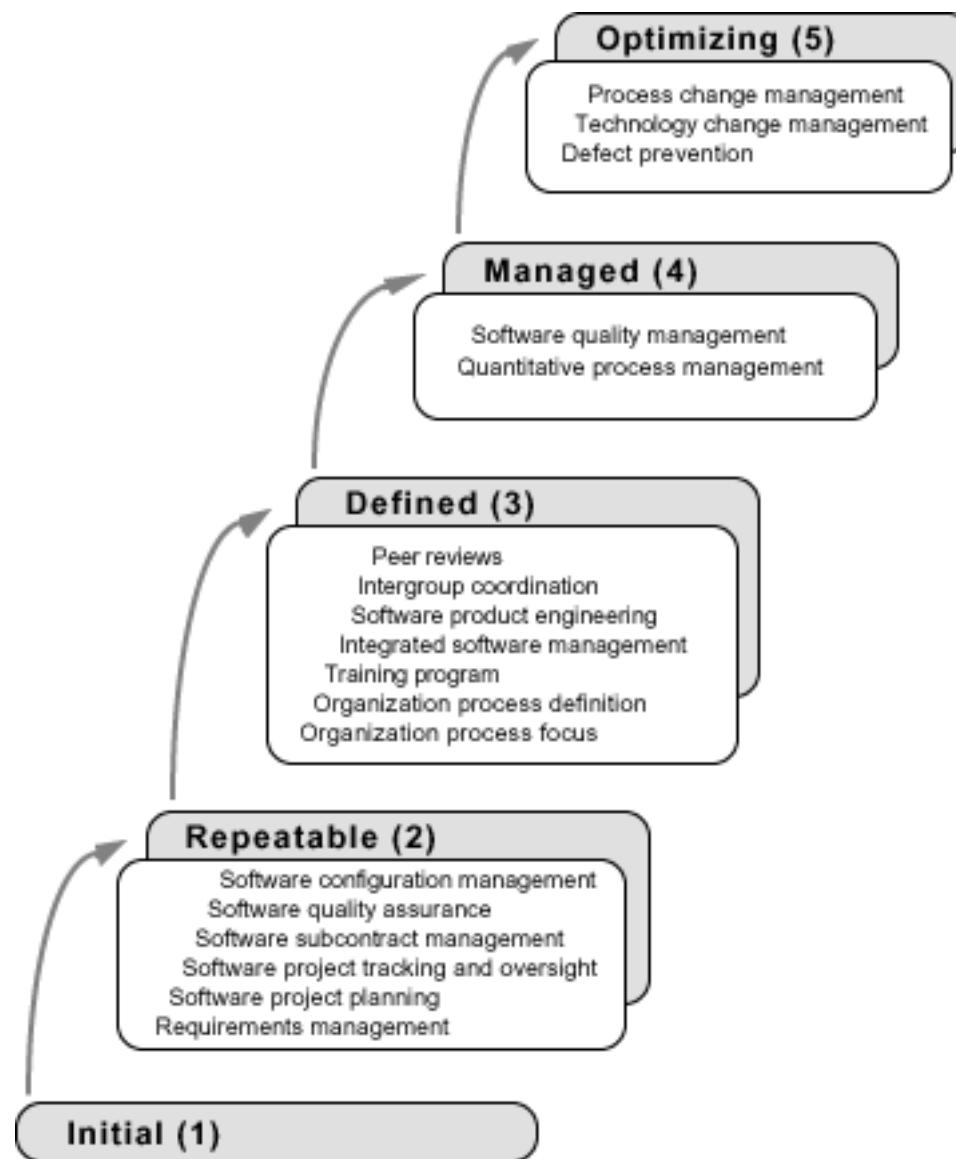
- This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.
- Use of new tools, techniques and evaluation of software processes is done to prevent recurrence of known defects.

Internal Structure to Maturity Levels

- Except for level 1, each level is decomposed into key process areas (KPA)
- Each KPA identifies a cluster of related activities that, when performed collectively, achieve a set of goals considered important for enhancing software capability.
 - commitment
 - ability
 - activity
 - measurement
 - verification

Key Process Areas (KPA's):

- Each of these KPA's defines the basic requirements that should be met by a software process in order to satisfy the KPA and achieve that level of maturity.
- Conceptually, key process areas form the basis for management control of the software project and establish a context in which technical methods are applied, work products like models, documents, data, reports, etc. are produced, milestones are established, quality is ensured and change is properly managed.



The Key Process Areas by Maturity Level

Repeatable

- L
e
v
e
I
2
K
P
As
- Requirement Management
- Software Project Planning
- Software project tracking & Oversight
- Software Subcontract management
- Software Quality Assurance
- Software Configuration Management

Level 2

KPAs

- Requirements Management
 - Establish common understanding of customer requirements between the customer and the software project
 - Requirements is basis for planning and managing the software project
 - Not working backwards from a given release date!
- Software Project Planning
 - Establish reasonable plans for performing the software engineering activities and for managing the software project

Level 2

KPAs

- Software Project Tracking and Oversight
 - Establish adequate visibility into actual progress
 - Take effective actions when project's performance deviates significantly from planned
- Software Subcontract Management
 - Manage projects outsourced to subcontractors
- Software Quality Assurance
- **Software quality assurance** (SQA) is a means and practice of monitoring the **software** engineering processes and methods used in a project to ensure proper **quality** of the **software**.
 - Provide management with appropriate visibility into
 - process being used by the software projects
 - work products

Level 2 KPA_s

- Software Configuration Management
- **software configuration management** (SCM or S/W CM) is the task of tracking and controlling changes in the **software**,
 - Establish and maintain the integrity of work products
 - Product baseline
 - Baseline authority

Defined

- L e v e l 3 K P As
- Organization Process Focus
 - Organization Process Definition
 - Training Program
 - Integrated Software Management
 - Software Product Engineering
 - Intergroup Coordination
 - Peer Reviews

Level 3

KPAs

- Organization Process Focus
 - Establish organizational responsibility for software process activities that improve the organization's overall software process capability
- Organization Process Definition
 - Develop and maintain a usable set of software process assets
 - stable foundation that can be institutionalized
 - basis for defining meaningful data for quantitative process management

Level 3

KPAs

- Training Program
 - Develop skills and knowledge so that individual can perform their roles effectively and efficiently
 - Organizational responsibility
 - Needs identified by project
- Integrated Software Management
 - Integrated engineering and management activities
 - Engineering and management processes are tailored from the organizational standard processes
 - Tailoring based on business environment and project needs

Level 3

KPAs

- Software Product Engineering
 - technical activities of the project are well defined (SDLC)
 - correct, consistent work products
- Intergroup Coordination
 - Software engineering groups participate actively with other groups
- Peer Reviews
 - early defect detection and removal
 - better understanding of the products
 - implemented with inspections, walkthroughs, etc

Managed

L

e
v
e
l

4

K

P

A

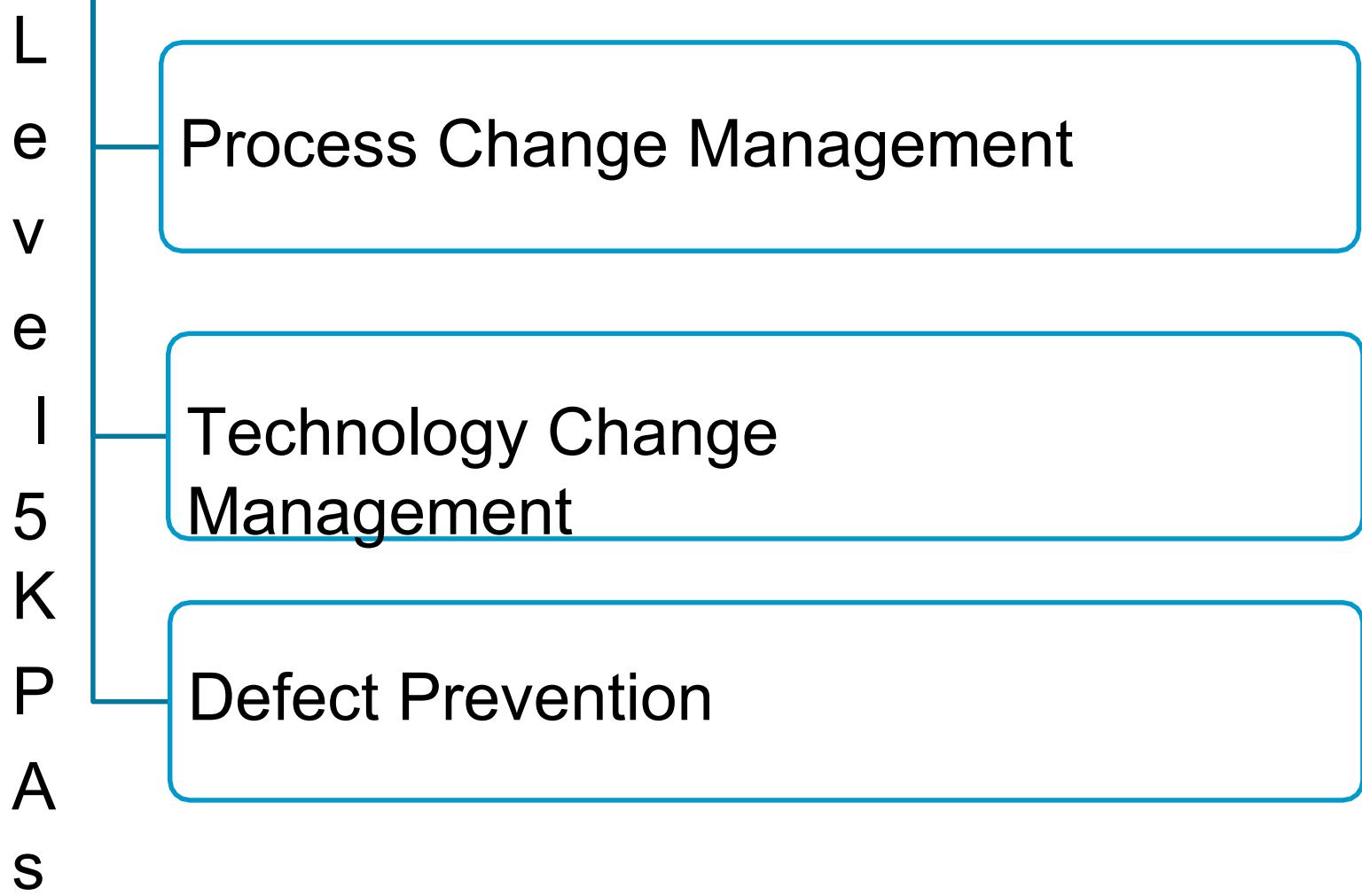
Quantitative Process Management

Software Quality Management

Level 4 KPAs

- Quantitative Process Management
 - control process performance quantitatively
 - actual results from following a software process
 - focus on identifying and correcting special causes of variation with respect to a baseline process
- Software Quality Management
 - quantitative understanding of software quality
 - products
 - process

Optimizing



Level 5

KPAs

- Process Change Management
 - continuous process improvement to improve quality, increase productivity, decrease cycle time
- Technology Change Management
 - identify and transfer beneficial new technologies
 - tools
 - methods
 - processes
- Defect Prevention
 - causal analysis of defects to prevent recurrence

Software Engineering(SE)

CSC 601



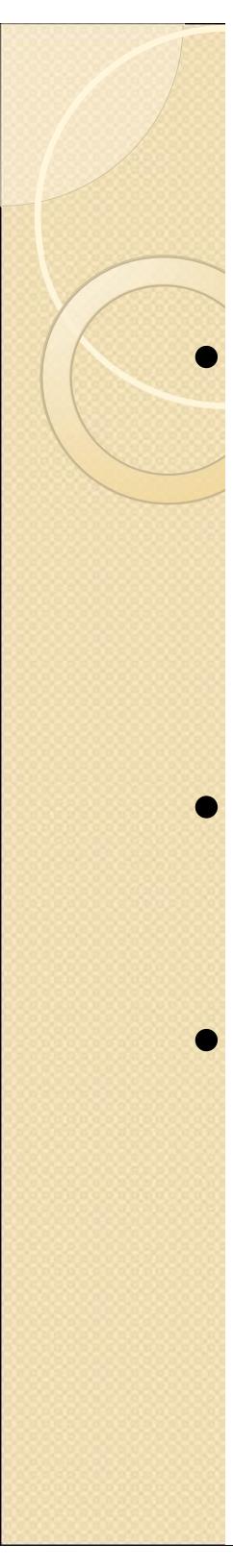
Subject Incharge

Varsha Nagpurkar
Assistant Professor
Room No. 407

email: varshanagpurkar@sfit.ac.in

Module-2 Syllabus

2.0	Requirements Analysis and Modelling	08
2.1	Requirement Elicitation, Software requirement specification (SRS), Developing Use Cases (UML)	
2.2	Requirement Model – Scenario-based model, Class-based model, Behavioural model.	



Key Ideas

- The goal of the **analysis phase** is to truly understand the requirements of the new system and develop a system that addresses them.
- The first challenge is **collecting and integrating the information**
- The second challenge is finding the **right people** to participate.

Elicitation & Analysis

- The requirements process consists of two activities:
 - Requirements **Elicitation**:
 - Definition of the system in terms understood by the **customer** (“Problem Description”)
 - Requirements **Analysis**:
 - *Technical* specification of the system in terms understood by the **developer** (“Problem Specification”)



Analysis Phase

- This phase takes the general ideas in the system request and
 - refines them into a detailed requirements definition
 - functional models,
 - structural models and
 - behavioral models
- **Final deliverable:** system proposal
 - Includes revised project management deliverables,
 - feasibility analysis and
 - work plan

Requirement Specification

- a statement of **what**
 - the system must do or
 - characteristics it must have
 - Written from business person perspective (**what the system does?**)
 - Later requirements become more technical (**how the system will be implemented?**)

Functional vs. Nonfunctional

- A *functional requirement* relates directly to a process the system has to perform or information it needs to contain (**functions the system needs to have**).
- *Nonfunctional requirements* refer to behavioral properties that the system must have, such as performance and usability.

Functional Requirements

Interface requirements

- Field 1 accepts numeric data entry.
- Field 2 only accepts dates before the current date.

Business Requirements

- Data must be entered before a request can be approved.
- Clicking the Approve button moves the request to the Approval Workflow.

Nonfunctional Requirements

D. Nonfunctional Requirements

1. Operational Requirements

- 1.1. The system will operate in Windows and Macintosh environments
- 1.2. The system will be able to read and write Word documents, RTF, and HTML
- 1.3. The system will be able to import Gif, Jpeg, and BMP graphics files

2. Performance Requirements

- 2.1. Response times must be less than 7 seconds
- 2.2. The Inventory database must be updated in real time

3. Security Requirements

- 3.1. No special security requirements are anticipated

4. Cultural and Political Requirements

- 4.1. No special cultural and political requirements are anticipated

Nonfunctional Requirements

- Performance – for example Response Time, Throughput, Utilization, Static Volumetric
- Scalability ,Capacity, Availability, Reliability, Recoverability, Maintainability, Serviceability
- Security ,Regulatory, Manageability, Environmental, Data Integrity, Usability ,Interoperability

One of the most common mistakes made by new analysts is to confuse functional and non-functional requirements. Pretend that you received the following list of requirements for a sales system:

Requirements for Proposed System:

The system should...

1. be accessible to Web users.
2. include the company standard logo and color scheme.
3. restrict access to profitability information.
4. include actual and budgeted cost information.
5. provide management reports.
6. include sales information that is updated at least daily.

7. have 2-second maximum response time for predefined queries and 10-minute maximum response time for ad hoc queries.
8. include information from all company subsidiaries.
9. print subsidiary reports in the primary language of the subsidiary.
10. provide monthly rankings of salesperson performance.

QUESTIONS:

1. Which requirements are functional business requirements? Provide two additional examples.
2. Which requirements are nonfunctional business requirements? What kind of nonfunctional requirements are they? Provide two additional examples.

Purpose Of requirement analysis

- *Scope of project*
- *Establish the user's expectations for system*



Requirements Gathering

Stakeholder Analysis

- Ensure that all stakeholder will considered
- Identifies all users and stakeholders who will influence system

Brain Storming

- It is used in identifying all possible solutions
- It will give good number of ideas from group

One to One Interview

- Sit with clients and ask questions
- In depth ideas from stakeholders

Group Interview

- Interview with more persons
- Can get hidden requirements

Document Analysis

- Prepare documentation
- Can help in As Is process design analysis

Prototyping

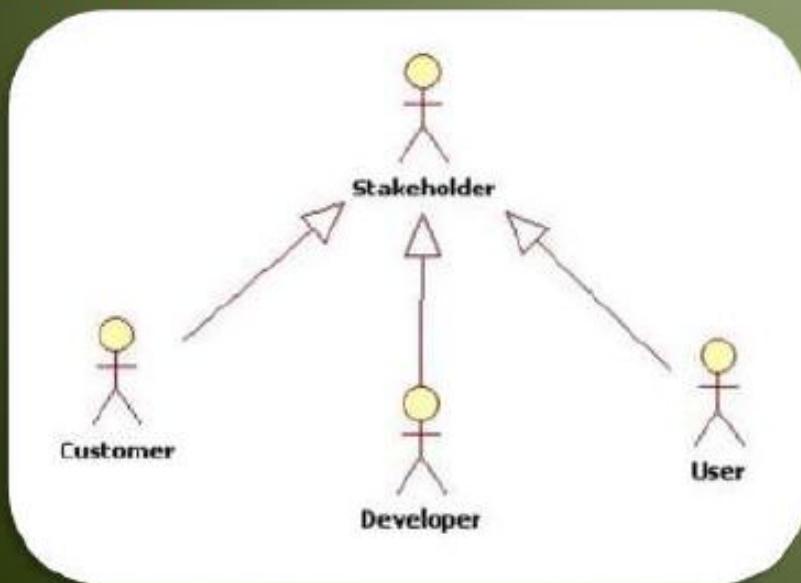
- Modern technique for gathering requirements
- Build the prototype ,its more practical and reduce risk

Joint Application Development

- Arrange workshops for gathering requirements
- Contain two analyst facilitator and Scriber

Stakeholder Analysis

Stakeholder analysis identifies all the users and stakeholders who may influence or be impacted by the system. This helps ensure that the needs of all those involved are taken into account



Benefits

1. Ensures that all relevant stakeholders are considered
 1. All important stakeholders are captured, and yet that irrelevant actors are not included

Drawbacks

There is a danger that too much time is spent on identifying roles and relationships, and the team is swamped with data.

Brainstorming

It is utilized in requirements elicitation to gather good number of ideas from a group of people. Usually brainstorming is used in identifying all possible solutions to problems and simplifies the detail of opportunities. It casts a broad net, determining various discreet possibilities.



Basic Rules

1. Start out by clearly stating the objective of the brainstorming session.
2. Generate as many ideas as possible.
3. Let your imagination soar.
4. Do not allow criticism or debate while you are gathering information.
5. Once information is gathered, reshape and combine ideas.

Brainstorming

Benefits

- 1. Generate a variety of ideas in a short time**

- 2. Produce new and creative ideas**



Risks

- 1. The risk of having a bad session**

- 2. Making staff scared to voice their ideas because they were criticized in the session**

- 3. Problems normally occur if you only use traditional brainstorming techniques.**

One On One Interview

The most common technique for gathering requirements is to sit down with the clients and ask them what they need. The discussion should be planned out ahead of time based on the type of requirements you're looking for



Benefits

- Privacy of everyone
- in-depth a stakeholder's thoughts and get his or her perspective

Risks & Drawbacks

- Time Consuming
- Misunderstandings

Group Interview

If there are more than one person during interview usually 2 or 4 these people must be on some level must be on some level less time required



Benefits

- we can get hidden requirements
- uncover a richer set of requirements in a shorter period of time
- Uncover ambiguities

Risks & Drawbacks

- Not relaxed environment
- Conflicts
- The allotted time have been exhausted

Requirements Elicitation Process

- Elicitation is a task that helps the customer to define what is required
- Requirement Elicitation Methods
 - Collaborative Requirements Gathering
 - Quality function deployment
 - User scenarios
 - Elicitation Work Products



Collaborative Requirements Gathering

- In this collaborative, team-oriented approach to requirements gathering, a team of stakeholders and developers work together
 - To identify the problem
 - Propose elements of the solution
 - Negotiate different approaches
 - Specify a preliminary set of solution requirements
- The meeting for collaborative requirements gathering is conducted to discuss all above issues

5/25/2021

Guidelines for conducting a collaborative requirements gathering meeting are

- Meetings are conducted and attended by both software engineer and customers(along with 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” (can be a customer, a developer, or an outsider) controls the meeting.
- A “definition mechanism” (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room, or virtual forum) is used.

Quality Function Deployment

- *Quality function deployment (QFD) is a quality management technique that translates the needs of the customer into technical requirements for software. QFD “concentrates on maximizing customer satisfaction from the software engineering process”*

3 types of QFD requirements

- **Normal requirements:-***Normal requirements* identify the objectives and goals that are stated for a product or system during meetings with the customer. If these requirements are present, the customer is satisfied.
- Examples of normal requirements might be requested types of graphical displays, specific system functions, and defined levels of performance

Expected requirements

- These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them
- Their absence will be a cause for significant dissatisfaction
- Examples of expected requirements are ease of human/machine interaction,overall operational correctness and reliability, and ease of software installation

Exciting requirements

- These requirements reflect features that go beyond the customer's expectations and prove to be very satisfying when present
- For example, word processing software is requested with standard features
- The delivered product contains a number of page layout capabilities that are quite pleasing and unexpected

User Scenarios

- It is difficult to move into more technical software engineering activities until the software team understands how these functions and features will be used by different classes of end users
- To accomplish this, developers and users can create a set of scenarios that identify a thread of usage for the system to be constructed
- The scenarios, often called use-cases

Elicitation Work product

- The work product produced by requirement elicitation depend upon the size of the system or the system to be built
- The information produced as a consequence of requirements gathering includes
 - 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

Elicitation Work product

- The information produced as a consequence of requirements gathering includes
 - a list of requirements (preferably organized by function) and the domain constraints that applies to each
 - a set of usage scenarios that provide insight into the use of the system or product under different operating conditions,
 - any prototypes developed to better define requirements

Software Requirements Specification(SRS)

- SRS is the official statement of what the system developers should implement.
- SRS is a complete description of the behaviour of the system to be developed
- SRS should include both a definition of user requirements and a specification of the system requirements.
- The SRS fully describes what the software will do and how it will be expected to perform.

Purpose of SRS

- The SRS precisely defines the software product that will be built.
- SRS used to know all the requirements for the software development and thus that will help in designing the software.
- It provides feedback to the customer.

SRS Format

- All the requirements for the system have to be included in a document that is clear and concise
- For this, it is necessary to organize the requirements document as sections and subsection.
- Here, we discuss the organization proposed in the IEEE guide to software requirements specifications [IEEE87, IEEE94].

SRS Format

- The details which are included in SRS depend on the type of system that is being developed and the type of the development process
- A number of large organizations, such as IEEE have defined standards for software requirements document
- The most widely used standard is IEEE/ANSI 830-1998

SRS Format

1. Introduction

- 1.1 Purpose
- 1.2 Scope
- 1.3 Definitions, Acronyms and Abbreviations
- 1.4 Reference
- 1.5 Overview

2. Overall Description

- 2.1 Product Perspective
- 2.2 Product Functions
- 2.3 User Characteristics
- 2.4 General Constraints
- 2.5 Assumption and dependencies

SRS Format

3. Specific Requirements

3.1 External interface Requirements.

3.1.1 User interfaces.

3.1.2 Hardware interfaces.

3.1.3 Software interfaces.

3.1.4 Communications interfaces.

3.2 Functional Requirements

3.3 Performance Requirements

3.4 Design Constraints

3.5 Attributes

3.6 Other Requirements

Product Perspective

- As an external view on the **product**, the **product perspective** defines how the **product** contributes to fulfilling stakeholder needs and adjacent systems assumptions.
- They define the **product's** functional behavior, qualities, and constraints in response to stakeholder needs or adjacent systems' assumptions.

Types of reader for requirement specification

System Customers

Specify the requirements and read them to check that they meet their needs. Customer specify changes to the requirements .

Managers

Use the Requirements Document to plane a bid for the system and to plan system development process.

System Engineers

Use the requirements to understand what system is to be deployed.

Types of Reader for Requirement Specification

System test engineers

Use the requirements to develop validation tests for the system

System Maintains engineers

Use the requirements to understand the system and the relationship between its parts

Characteristics of an SRS

- Complete-All the project functionalities should be recorded by SRS
- Consistent-SRS should be consistent
- Accurate-SRS defines systems functionality in real world.we need to record requirement very carefully
- Modifiable-Logical and hierarchical modification should be allowed in SRS

Characteristics of an SRS

- Ranked-Requirement should be ranked using different factors like stability, security, ease or difficulty
- Testable-The requirement should be realistic and able to implement
- Traceable-Every requirement we must be able to uniquely identify
- Unambiguous-Statement in the SRS document should have only one meaning
- Valid-All the requirements should be valid



Unified Modeling Language

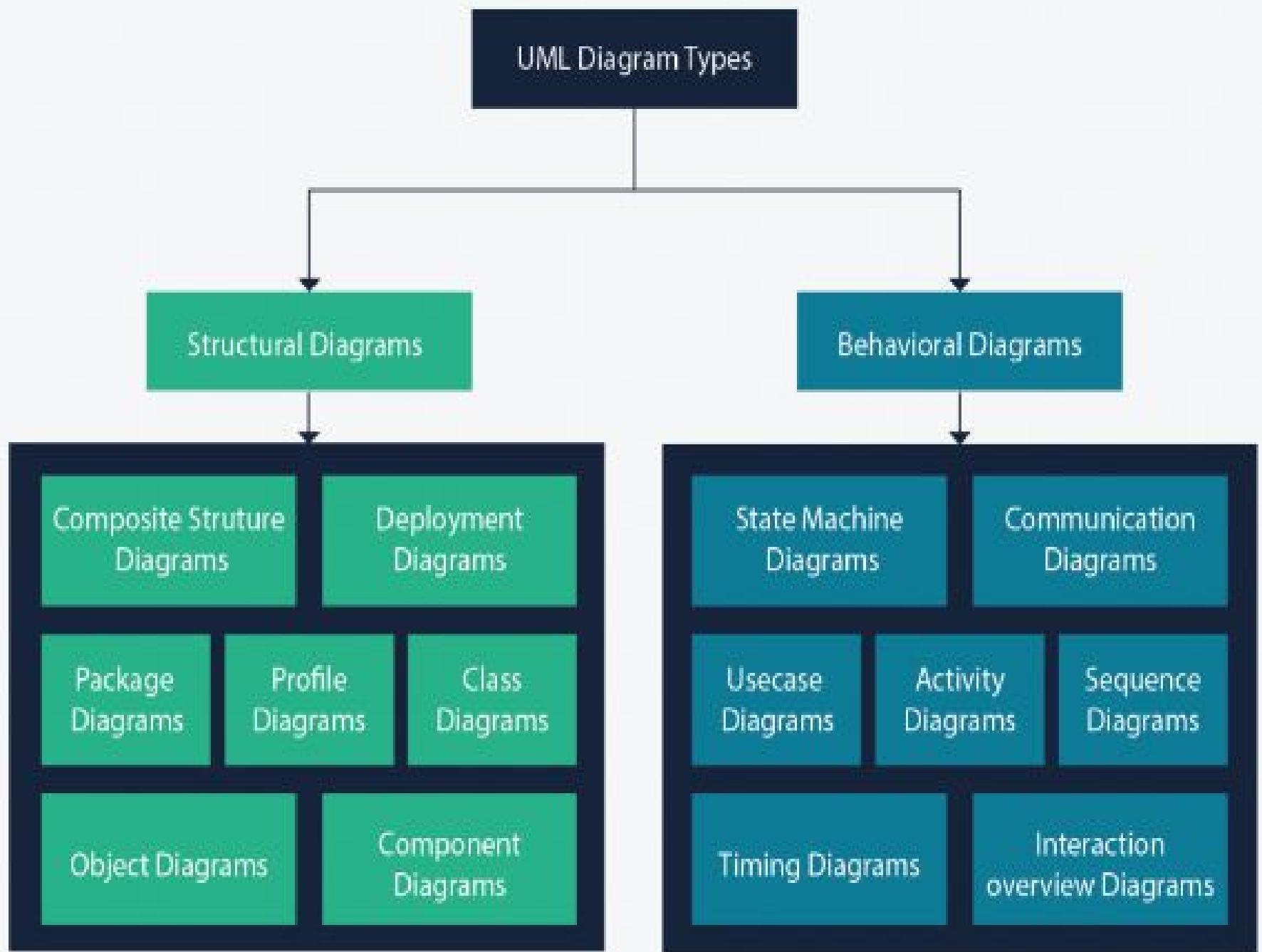


UML

- UML stands for **Unified Modeling Language**.
- It's a rich language to model software solutions, application structures, system behavior and business processes.

UML Diagram Types

- There are two main categories;
 - **Structure diagrams**
These diagrams show different objects in a system
 - **Behavioral diagrams.**
 - These diagrams show what should happen in a system.
 - They describe how the objects interact with each other to create a functioning system.





Use Case Diagram

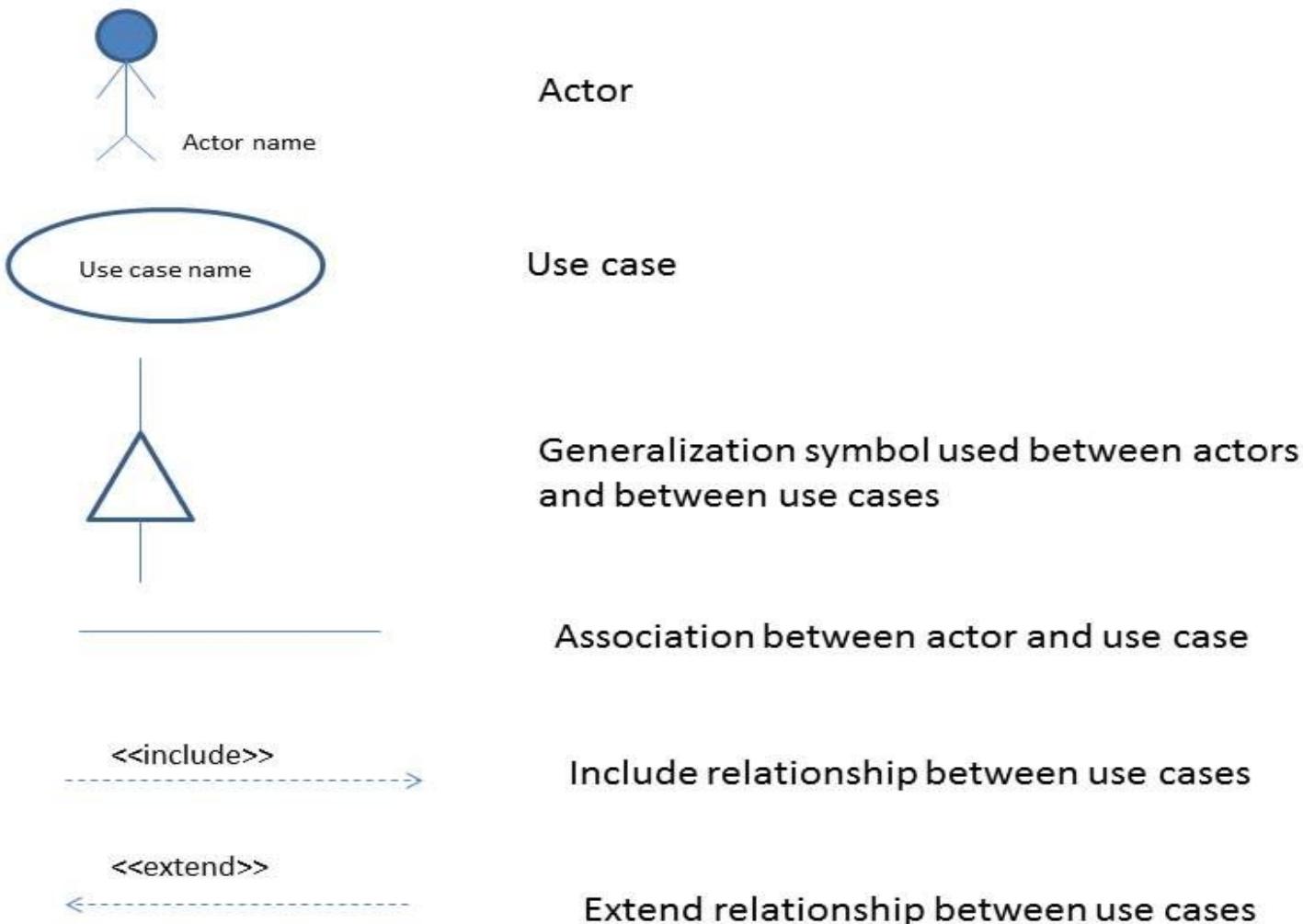
- **Use Case Diagram** captures the system's functionality and requirements by using actors and use cases.
- Use Cases model the services, tasks, function that a system needs to perform.
- Use cases represent high-level functionalities and how a user will handle the system.



Use Case Diagram

- Use case diagrams capture the dynamic behaviour of a live system.
- It models how an external entity interacts with the system to make it work.
- It's a great starting point for any project discussion because you can easily identify the main actors involved and the main processes/functionalities of the system

Symbols used in Use Case Diagram



Symbols in a use case diagram

Use Cases

- What is a Use Case

- A formal way of representing how a business system interacts with its environment

- Illustrates the activities that are performed by the users of the system

- A **scenario-based** technique in the UML

Scenario is A sequence of actions a system performs that yields a valuable result for a particular actor.

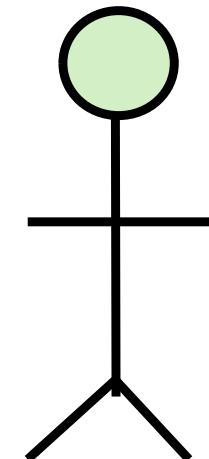


Actor

- What is an Actor?
 - A **user** or **outside** system that interacts with the system being designed in order to obtain some value from that interaction
- Use Cases describe **scenarios** that describe the interaction between users of the system (the actor) and the system itself.

Actors

- An Actor is outside or external the system.
- It can be a:
 - Human
 - Peripheral device (hardware)
 - External system or subsystem
 - Time or time-based event
- Represented by stick figure





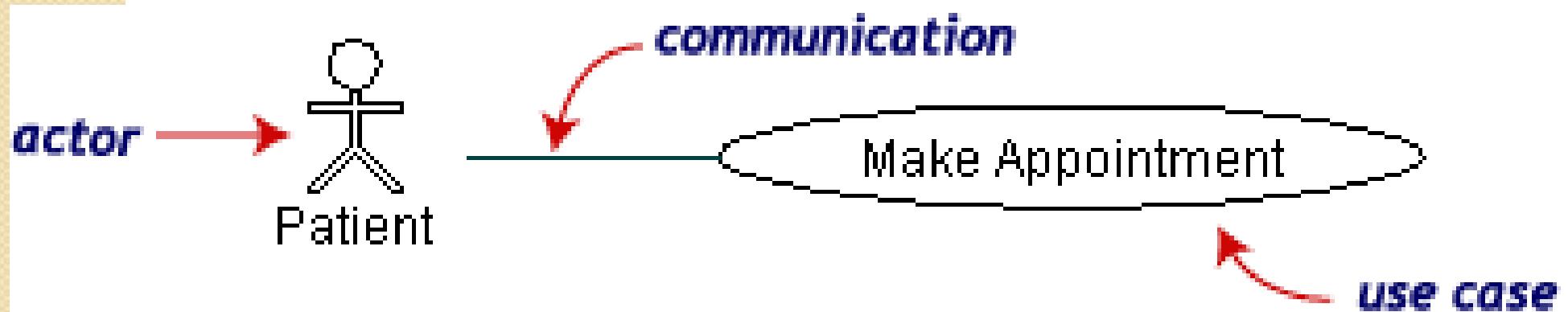
Use Case Diagram

- Use case diagrams describe what a system does from the standpoint of an external observer. **The emphasis is on *what* a system does rather than *how*.**
- Use case diagrams are closely connected to scenarios. A **scenario** is an example of what happens when someone interacts with the system.

Use Cases

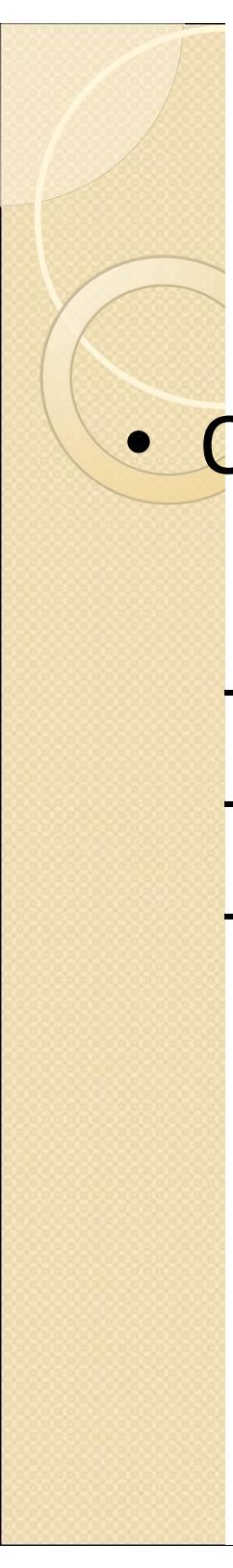
- The picture below is a **Make Appointment** use case for the medical clinic.
- The actor is a **Patient**. The connection between actor and use case is a **communication association** (or **communication** for short).

**Actors are stick figures. Use cases are ovals.
Communications are lines that link actors to use cases.**



Use Case Relationships

Relations	Symbol	Meaning
Communicates		An actor is connected to a use case using a line with no arrowheads.
Includes		A use case contains a behavior that is common to more than one other use case. The arrow points to the common use case.
Extends		A different use case handles exceptions from the basic use case. The arrow points from the extended to the basic use case.
Generalizes		One UML "thing" is more general than another "thing." The arrow points to the general "thing."



Use Case Diagram

- Other Types of Relationships for Use Cases
 - Generalization
 - Include
 - Extend

Use Case Diagram for student management system:

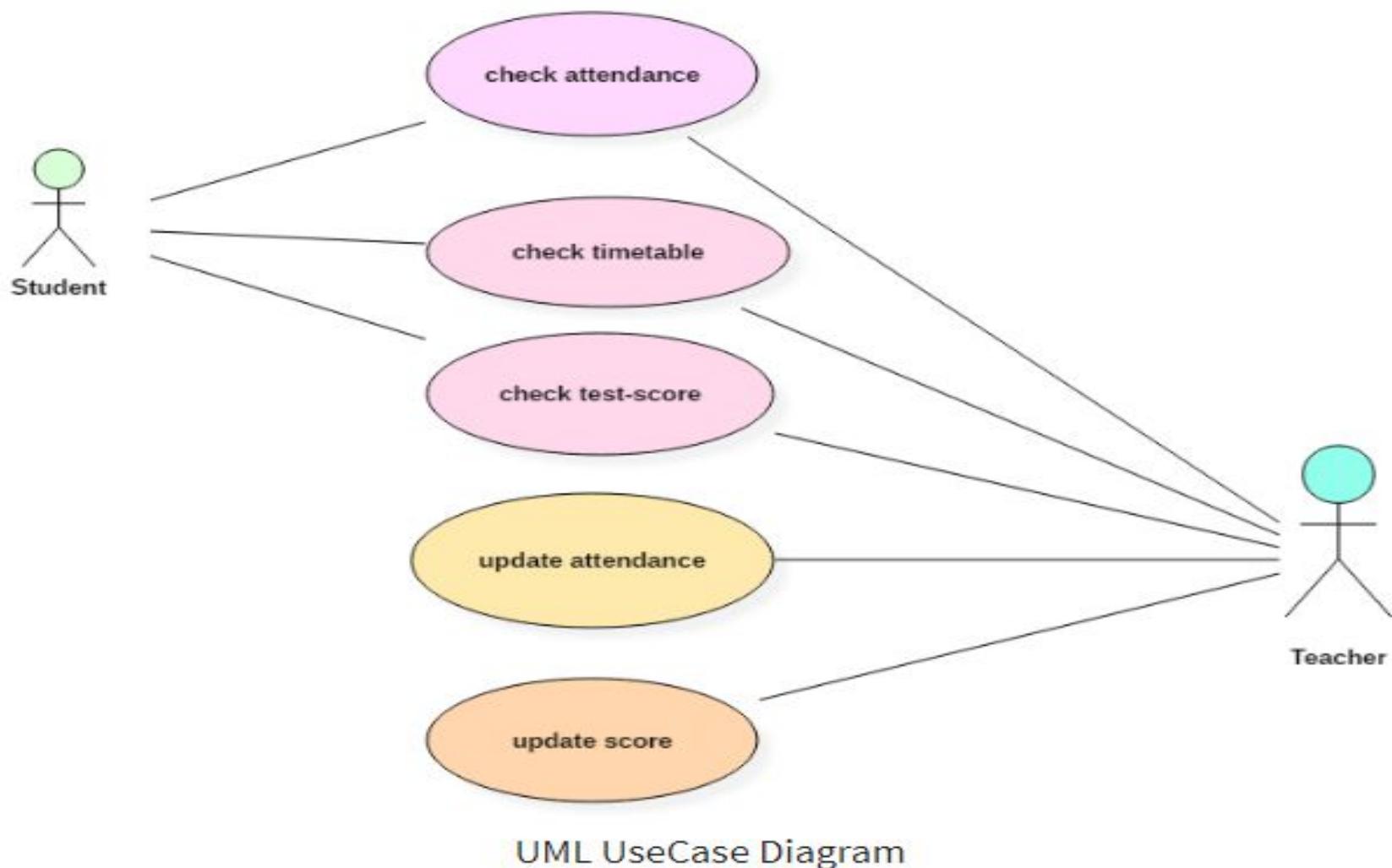


Fig.1 Use Case Diagram for Student Management System

Use Case Diagram for Bank ATM System

- Step- I

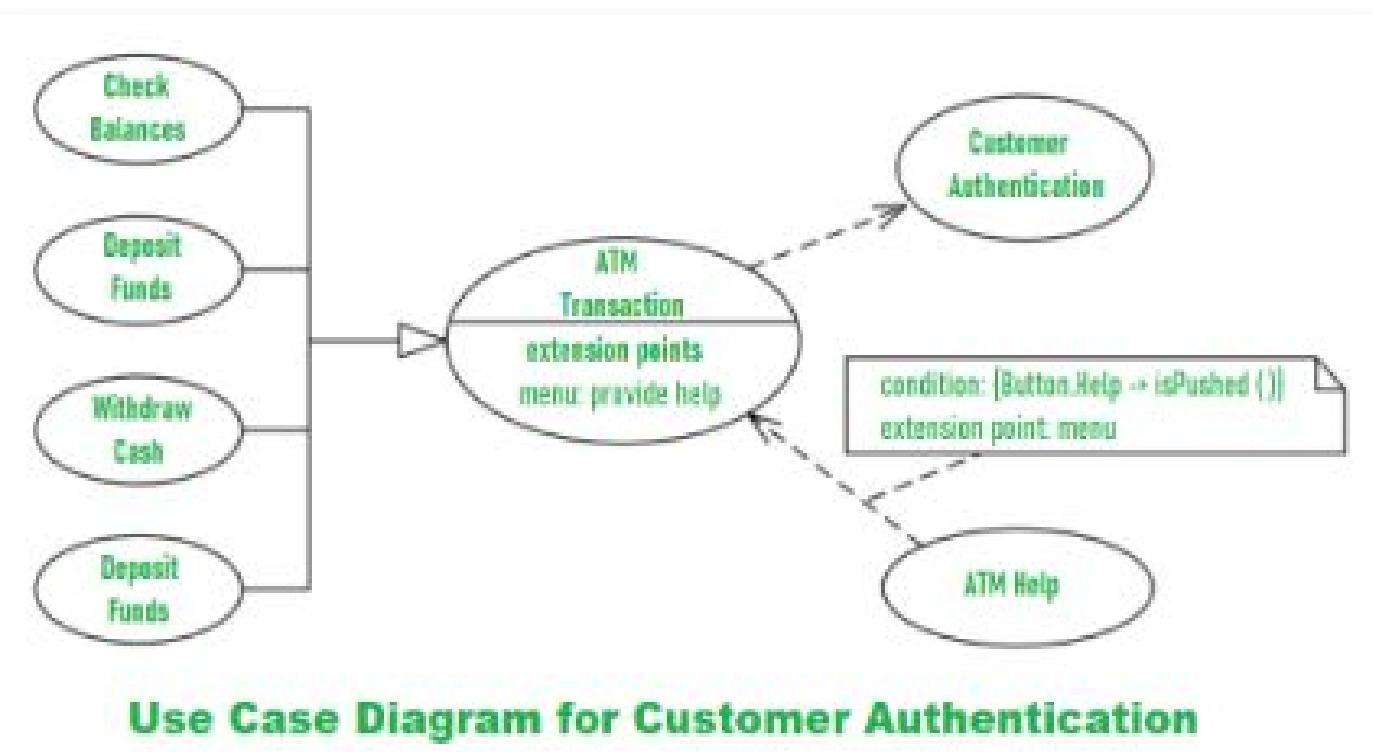
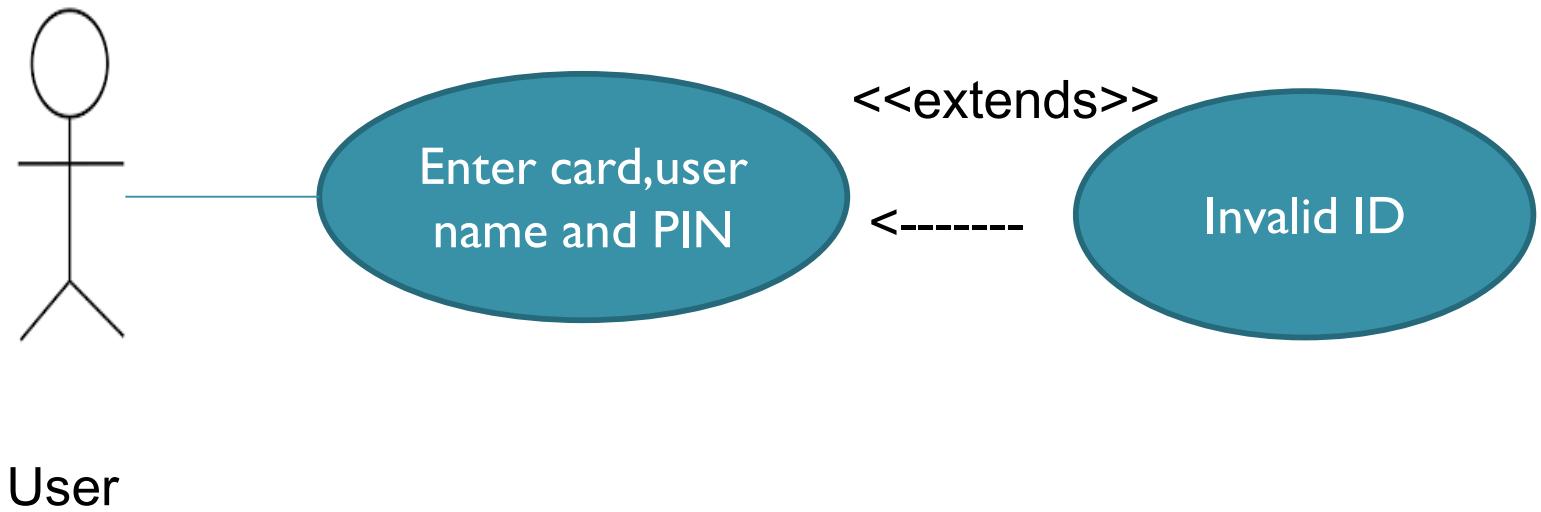


Fig.2 Use Case Diagram for Customer Authentication

Step- I

- The user is authenticated when enters the plastic ATM card in a Bank ATM. Then enters the user name and PIN (Personal Identification Number).
- For every ATM transaction, a Customer Authentication use case is required and essential. So, it is shown as include relationship.
- Example of use case diagram for Customer Authentication is shown in Fig.No-2

Extends Relationship



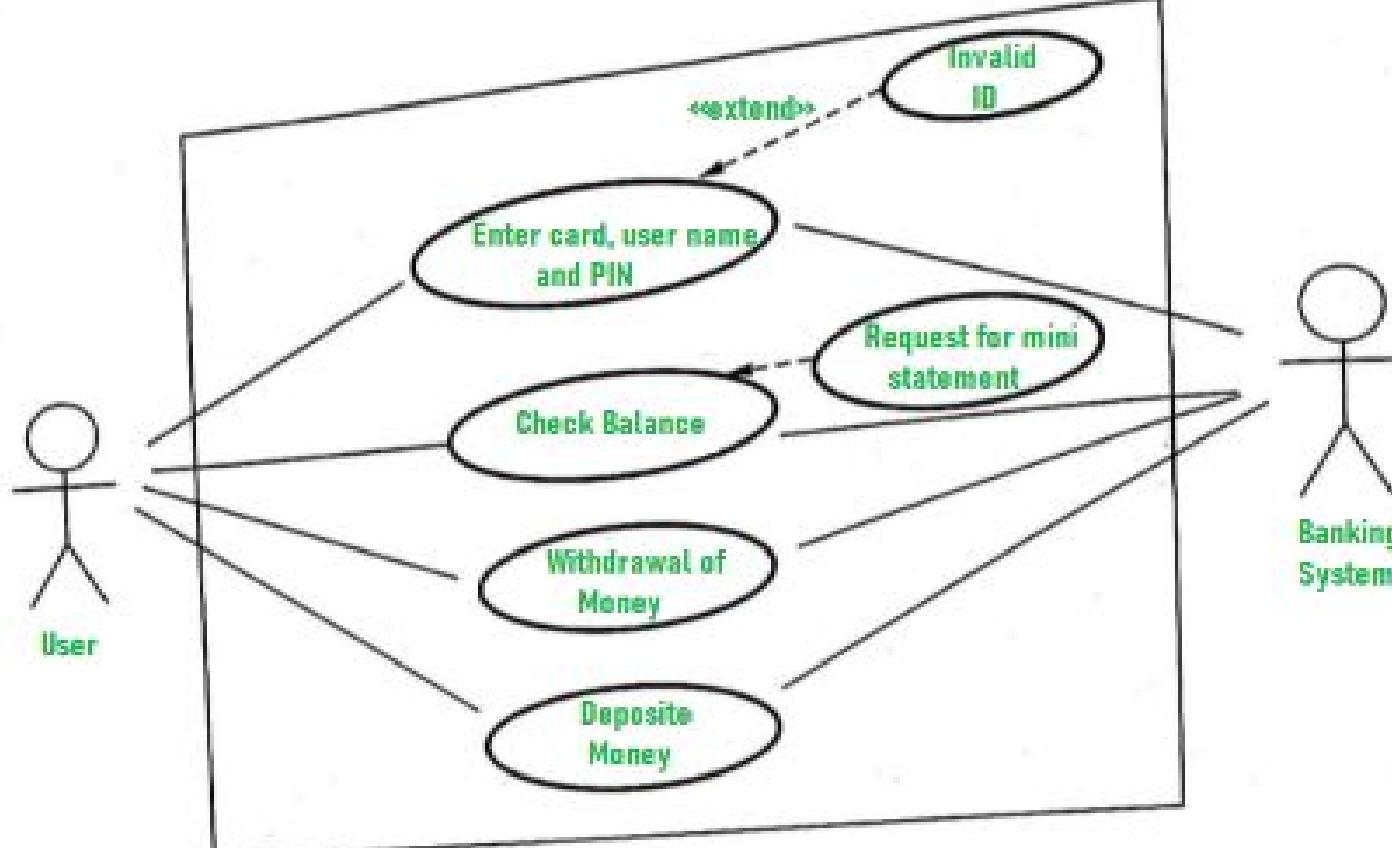


Extends Relationship

- The extends relationship describes the situation in which one use case allows the new use case to handle an exception or variation from the basic use case.
- The arrow goes from the extended to the basic use case.

Use Case Diagram for Bank ATM System

- Step-2



Use Case Diagram for Bank ATM System

Fig.3 Use Case Diagram for Bank ATM System

Step-2

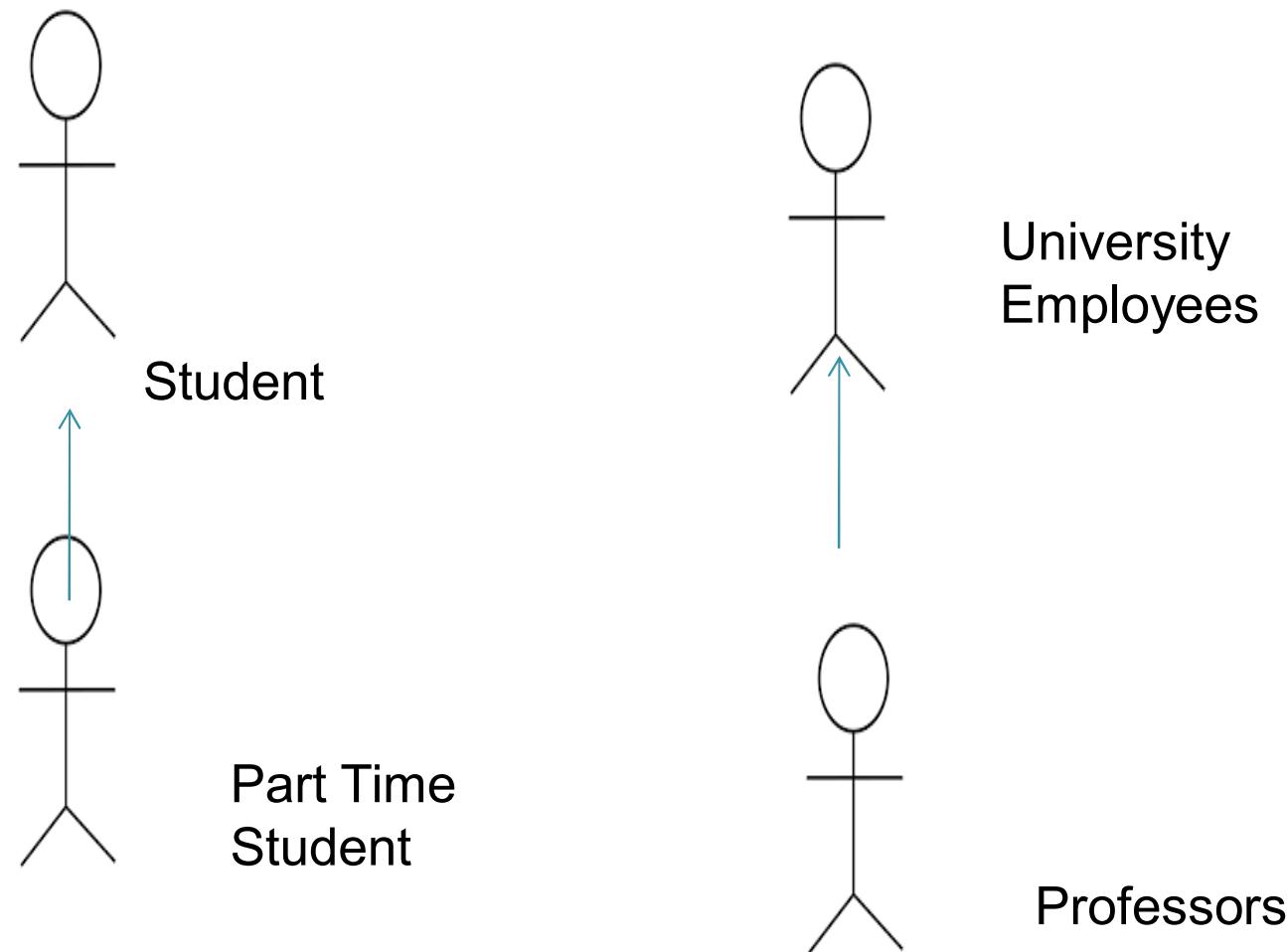
- User checks the bank balance as well as also demands the mini statement about the bank balance if they want.
- Then the user withdraws the money as per their need. If they want to deposit some money, they can do it. After complete action, the user closes the session. Example of the use case diagram for Bank ATM system is shown in fig. no-3:



Generalization Relationship

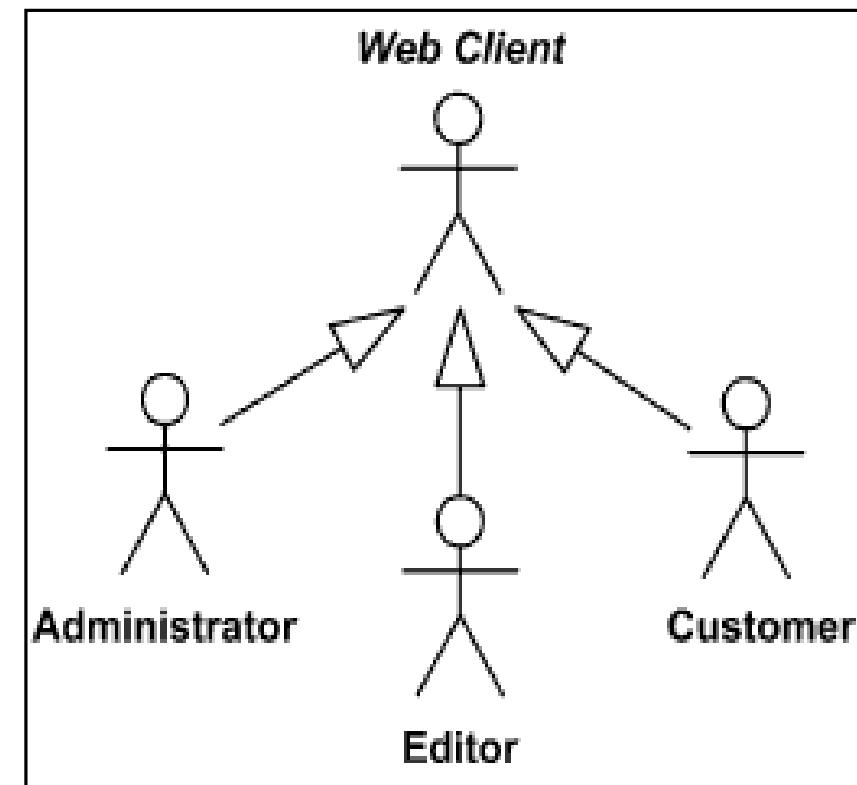
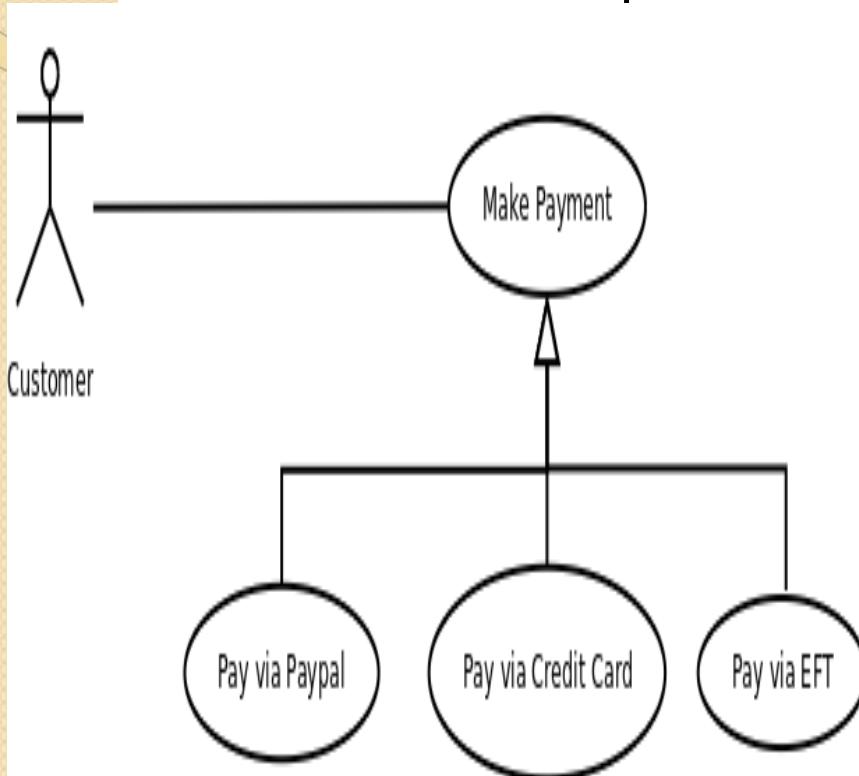
- It implies that one thing is more typical than the other thing.
- The relationship may exist between 2 actors or 2 use cases.
- For example,
 - A part time student generalizes a student.
 - Some of the university employees are professors.
- The arrow points to the general thing.

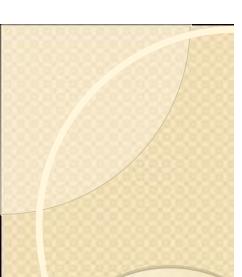
Generalization Example



Components of Use Case Diagram

- Generalization Relationship
 - Represented by a line and a hollow arrow
 - From child to parent





Registration

**Generalization
Example 1**

Under-graduate
registration

Graduate
registration

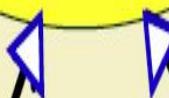


Factoring Use Cases Using Generalization

Pay membership fee

Pay through credit card

Pay through library pay card



Use Case Diagram for Bank ATM System

- Step-3

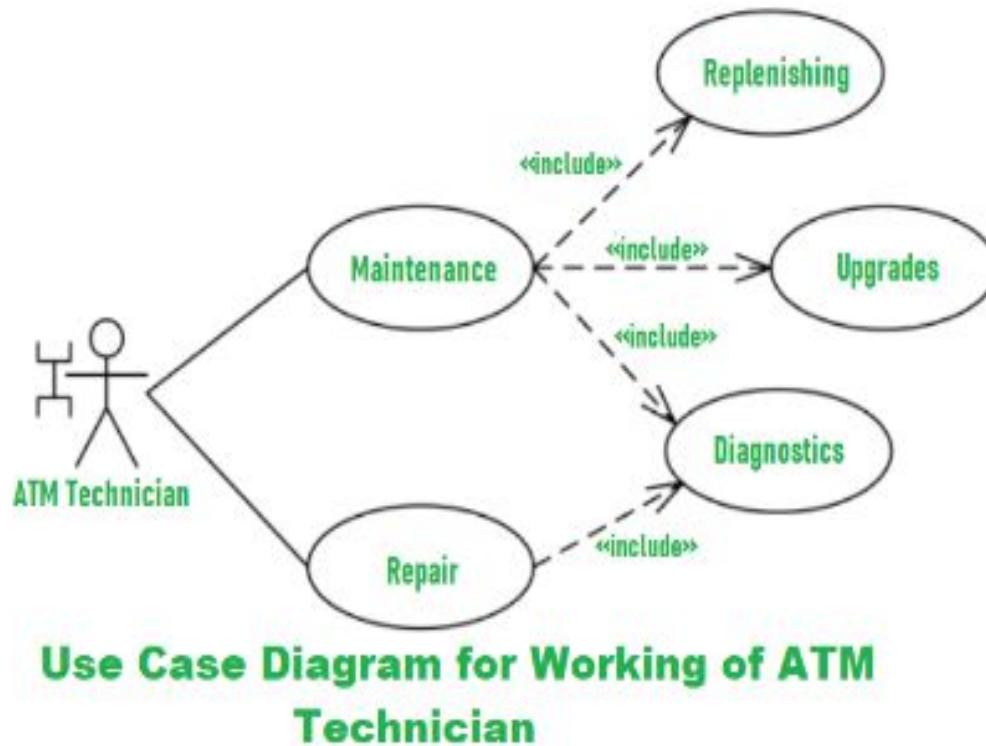
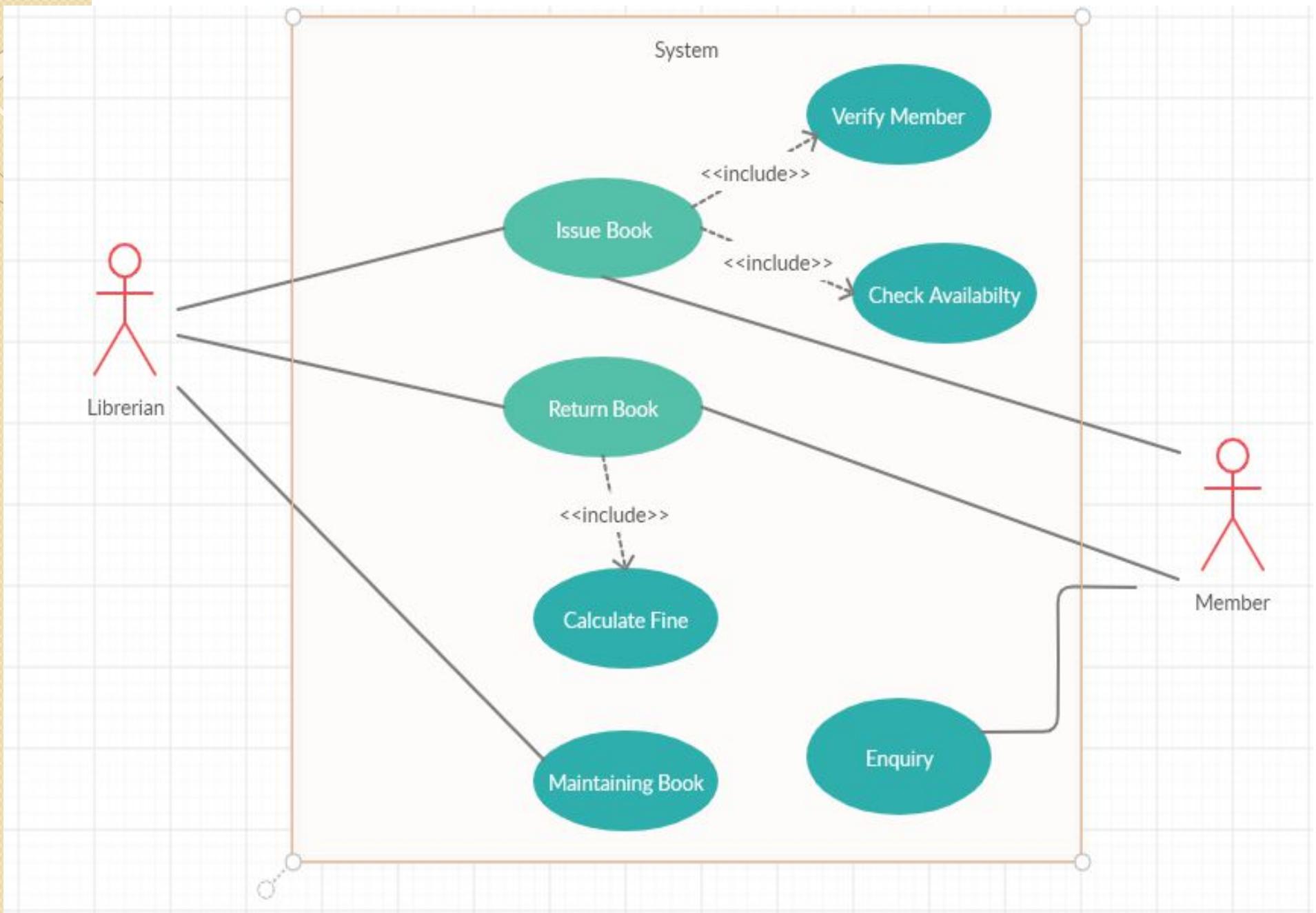


Fig.4 Use Case Diagram for Working of ATM Technician

Step-3:

- If there is any error or repair needed in Bank ATM, it is done by an ATM technician.
- ATM technician is responsible for the maintenance of the Bank ATM, upgrades for hardware, firmware or software, and on-site diagnosis.
- Example of use case diagram for working of ATM technician is shown in fig no-4

Library Management System





Scenario Based Model : Use Cases,Activity Diagrams and Swim lane Diagrams

Activity Diagram

- The UML activity diagram supplements the use case by providing a graphical representation of the flow of interaction within a specific scenario
- Used to document workflow of business process activities for each use case or scenario
- Similar to the flowchart, an activity diagram uses rounded rectangles to imply a specific system function, arrows to represent flow through the system, decision diamonds to depict a branching decision and solid horizontal lines to indicate parallel activities

Activity Diagrams Symbols

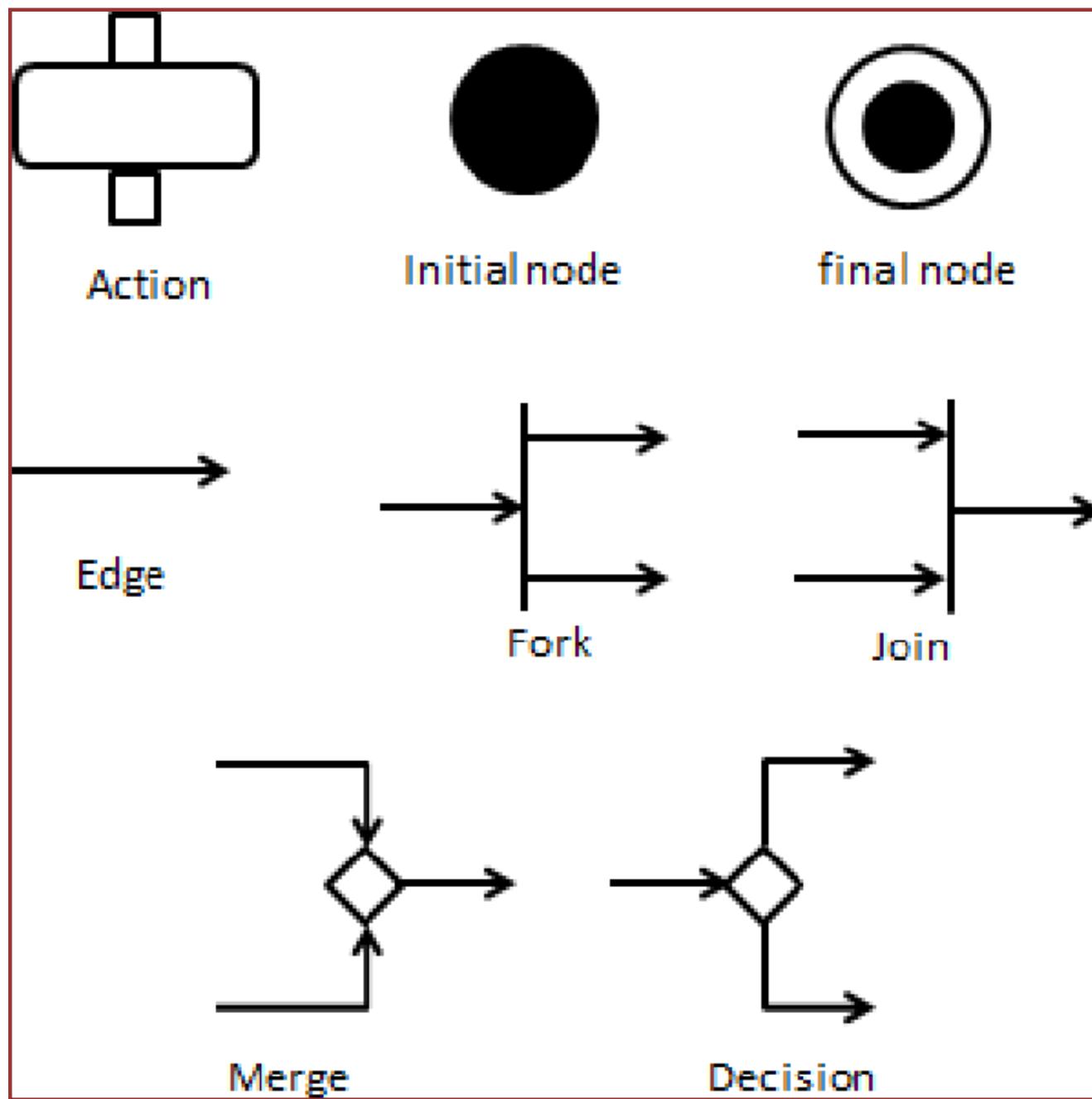
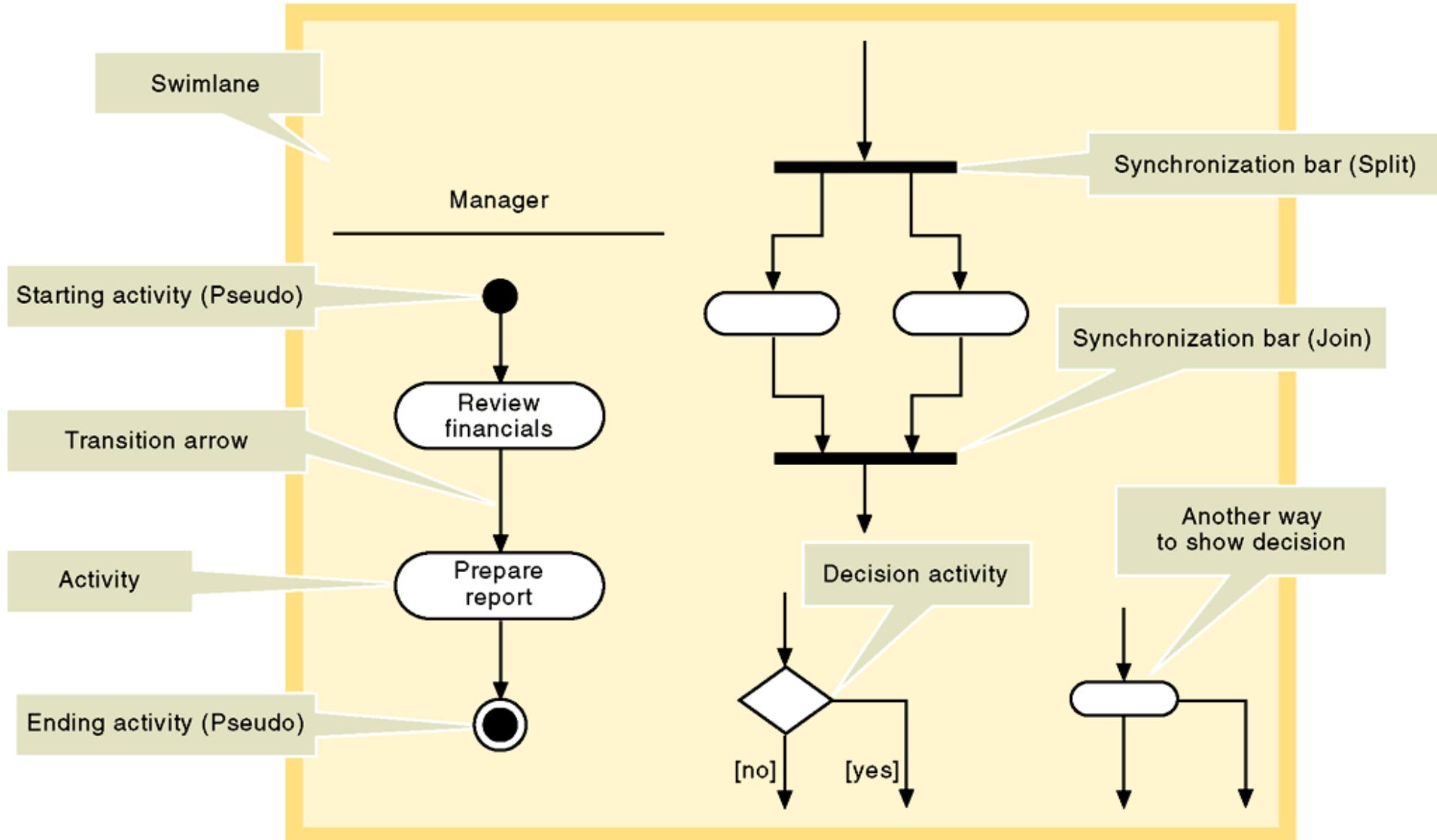


FIGURE 4-11

Activity diagram symbols.

Activity Diagram Symbols

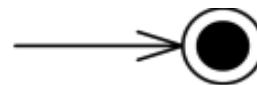


Basic Components in an Activity Diagram

- **Initial node**
 - The filled circle is the starting point of the diagram
- **Final node**
 - The filled circle with a border is the ending point. An activity diagram can have zero or more activity final state.
- **Activity**
 - The rectangle with rounded ends represents activities that occur. An activity is not necessarily a program, it may be a manual thing also
- **Flow/ edge**
 - The arrows in the diagram. No label is necessary



Activity initial node.



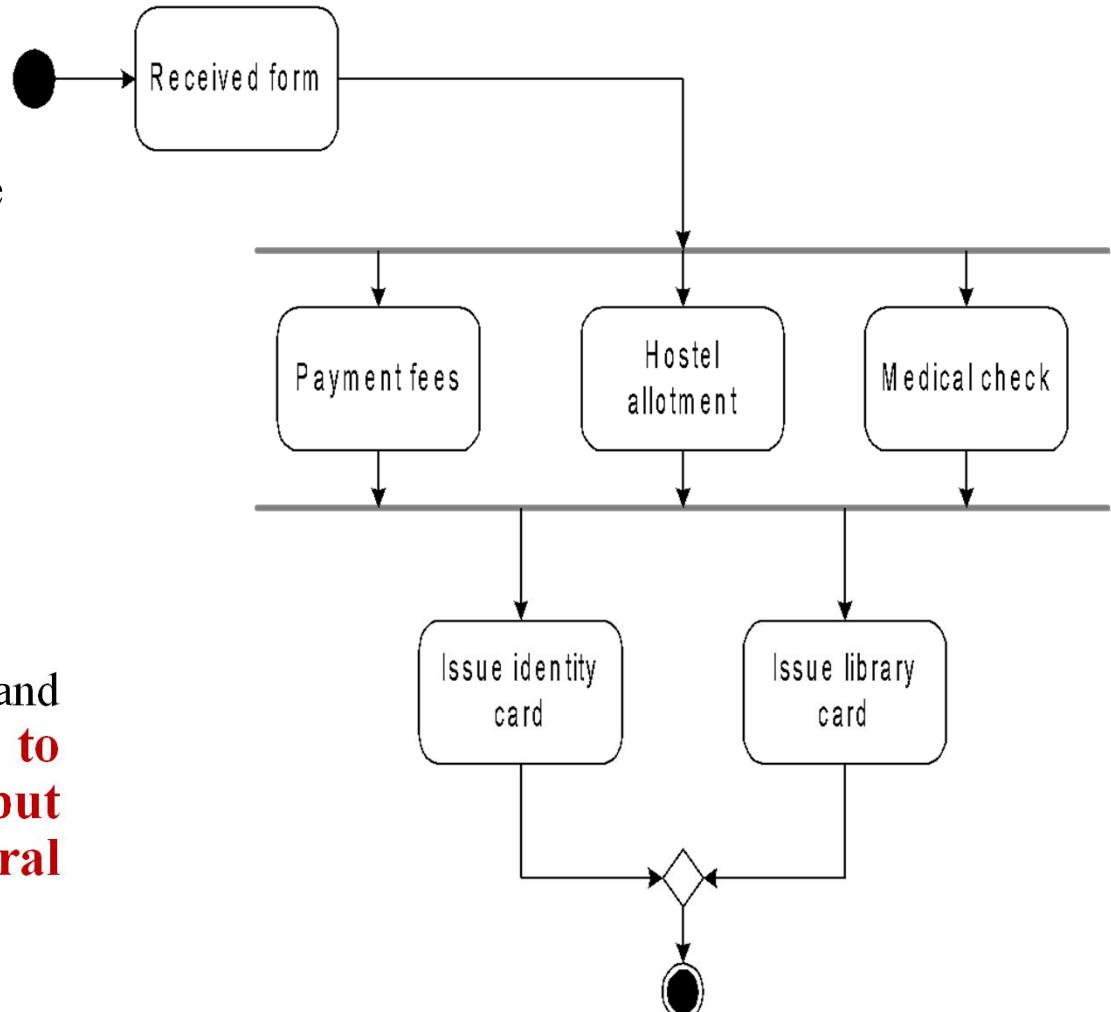
Activity final node.



Basic Components in an Activity Diagram

- **Fork**

- A black bar (horizontal/vertical) with one flow going into it and several leaving it. This denotes the **beginning of parallel activities**



- **Join**

- A block bar with several flows entering it and one leaving it. **this denotes the end of parallel activities**

- **Merge**

A diamond with several flows entering and one leaving. **It is not used to synchronize concurrent flows but to accept one among several alternate flows.**

How to Draw an Activity Diagram

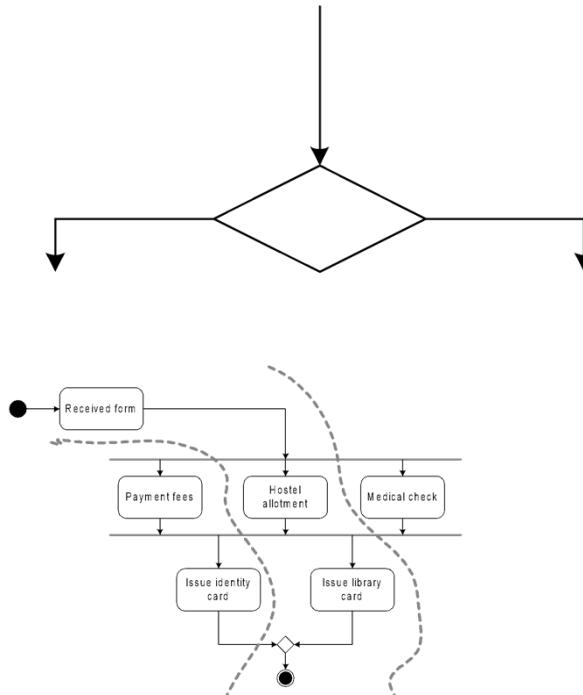
- Diagrams are read from top to bottom and have branches and forks to describe conditions and parallel activities.
 - A fork is used when multiple activities are occurring at the same time.
 - A branch describes what activities will take place based on a set of conditions.
 - All branches at some point are followed by a merge to indicate the end of the conditional behavior started by that branch.
 - After the merge all of the parallel activities must be combined by a join before transitioning into the final activity state.

Basic Components in an Activity Diagram

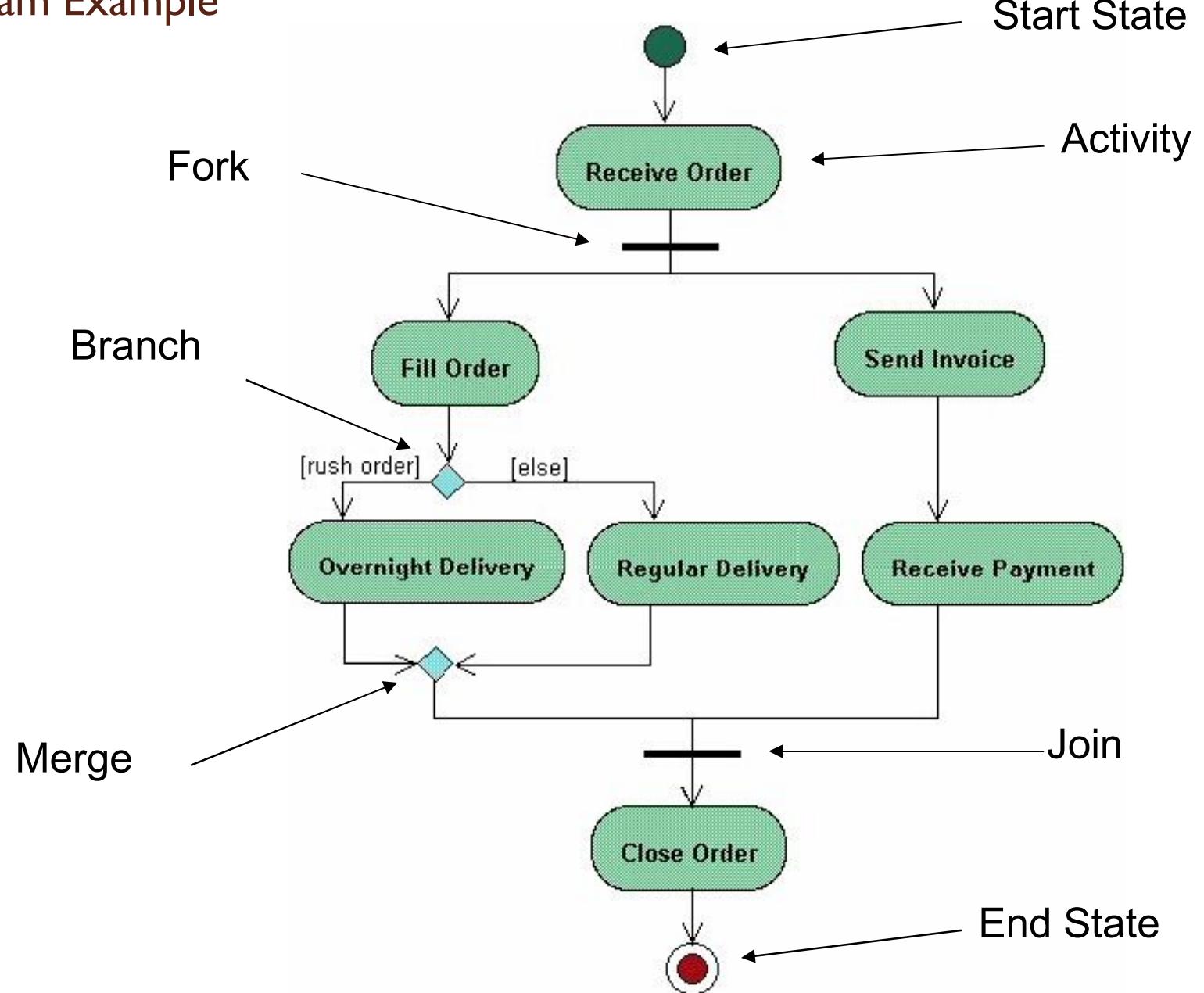
- Difference between Join and Merge
 - A join is different from a merge in that the join synchronizes two inflows and produces a single outflow. The outflow from a join cannot execute until all inflows have been received
 - A merge passes any control flows straight through it. If two or more inflows are received by a merge symbol, the action pointed to by its outflow is executed two or more times

Basic Components in an Activity Diagram

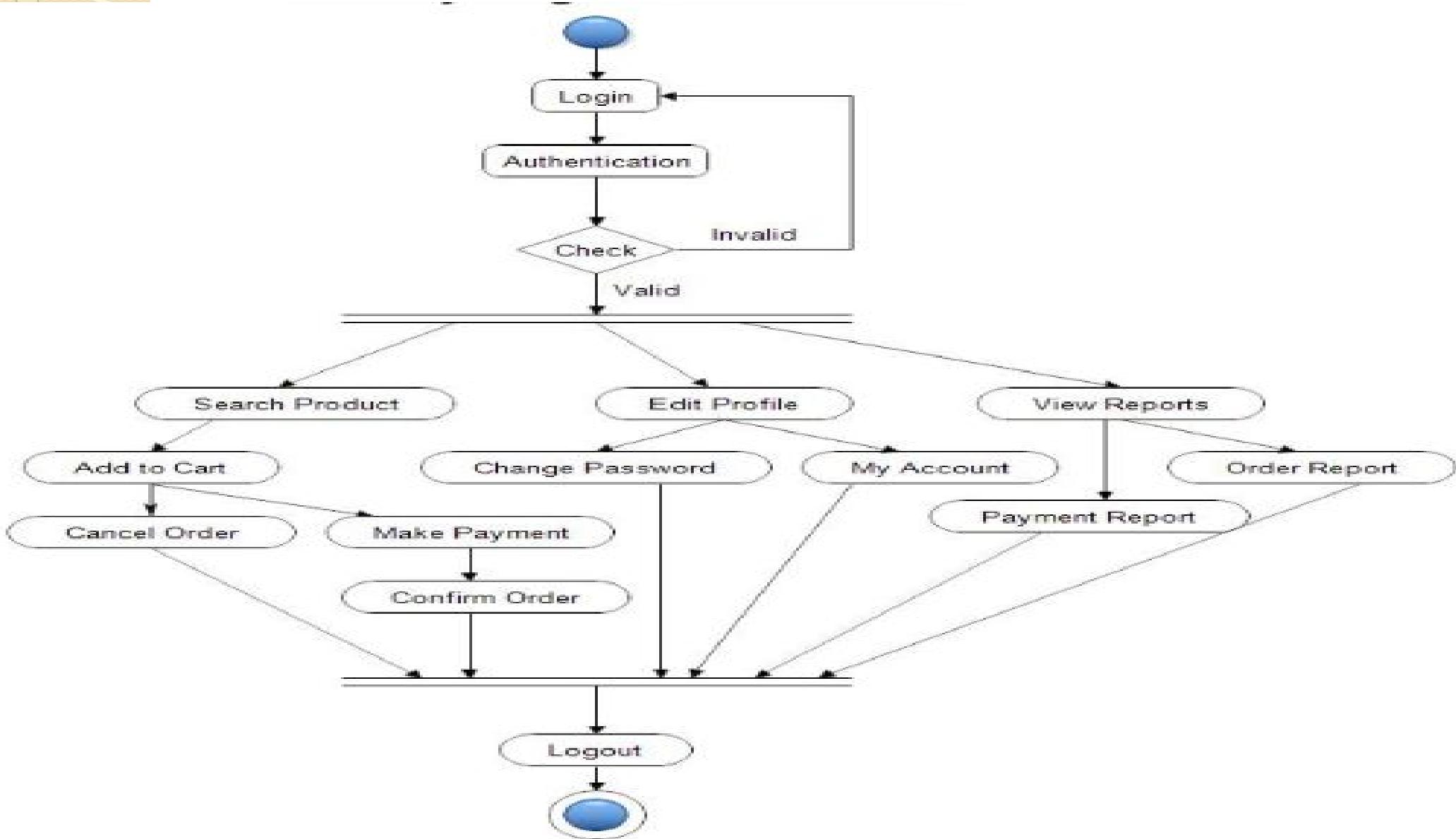
- **Decision**
 - A diamond with one flow entering and several leaving. The flow leaving includes conditions as yes/ no state
- **Swim lane**
 - A partition in activity diagram by means of line, called swim lane. This swim lane may be horizontal or vertical



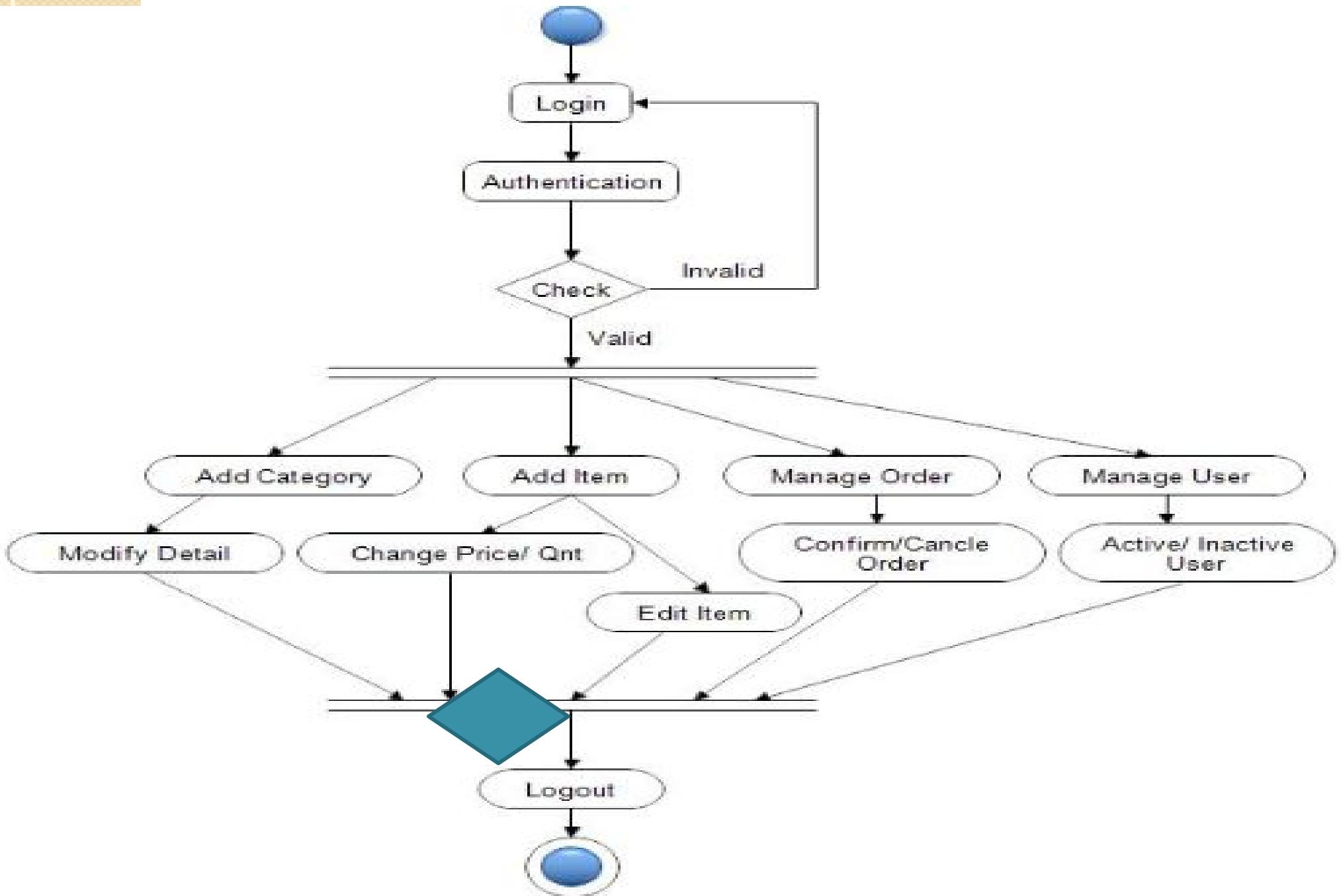
Activity Diagram Example



Activity Diagram for Online Shopping Website-Activity Diagram for User Side



Activity Diagram for Admin Side



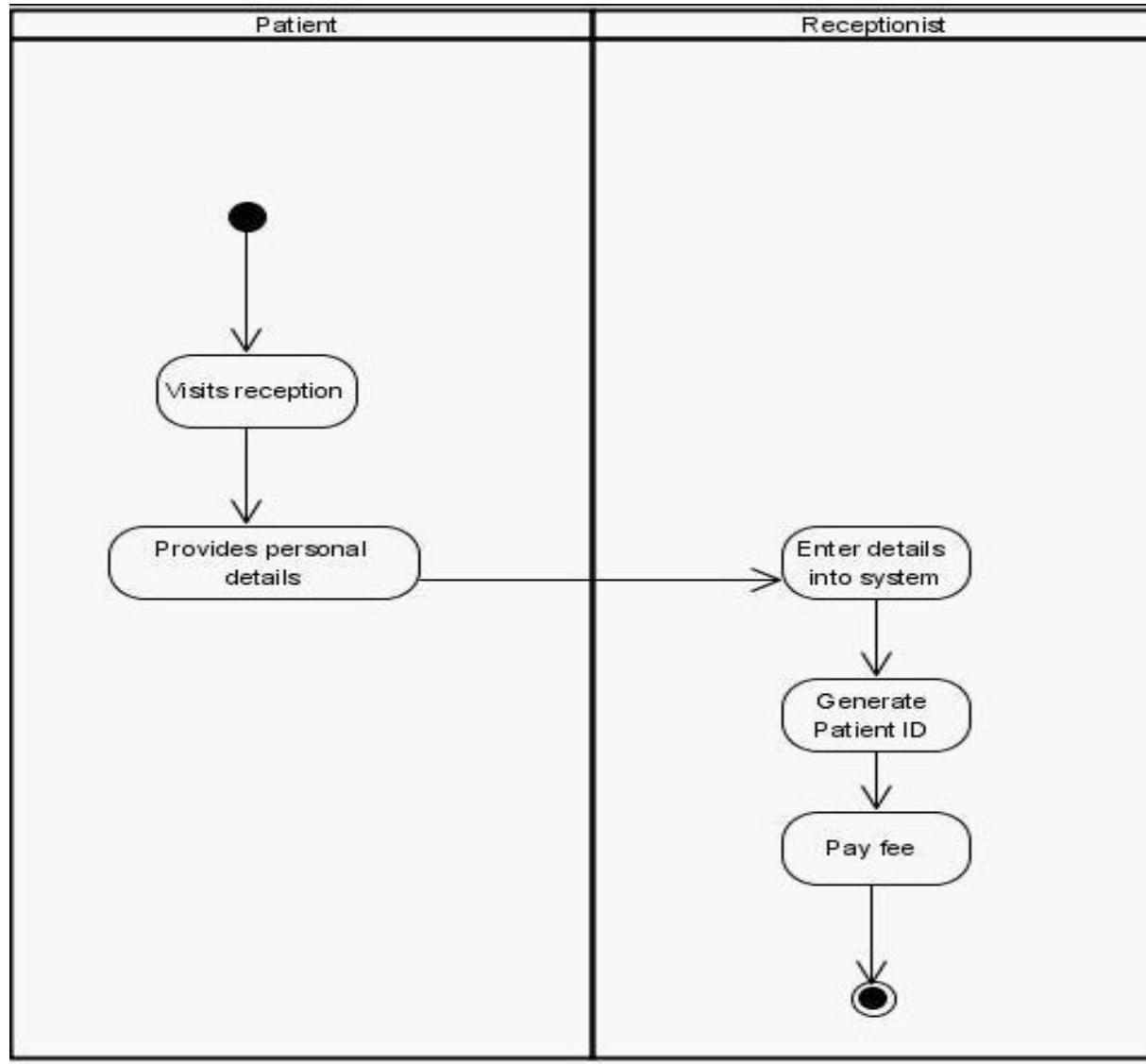
Swim lane Diagram

- Swim lane diagram is a useful variation of the activity diagram
- It allows the modeler to represent the flow of activities described by the use case
- It also indicate which actor(if there are multiple actors involved in a specific function) or analysis class has responsibility for the action described by an activity rectangle
- Here,the interaction between different activities shown in different lanes
- To show parallel activities,diagram is divided vertically into lanes,similar to lanes of swimming pool.

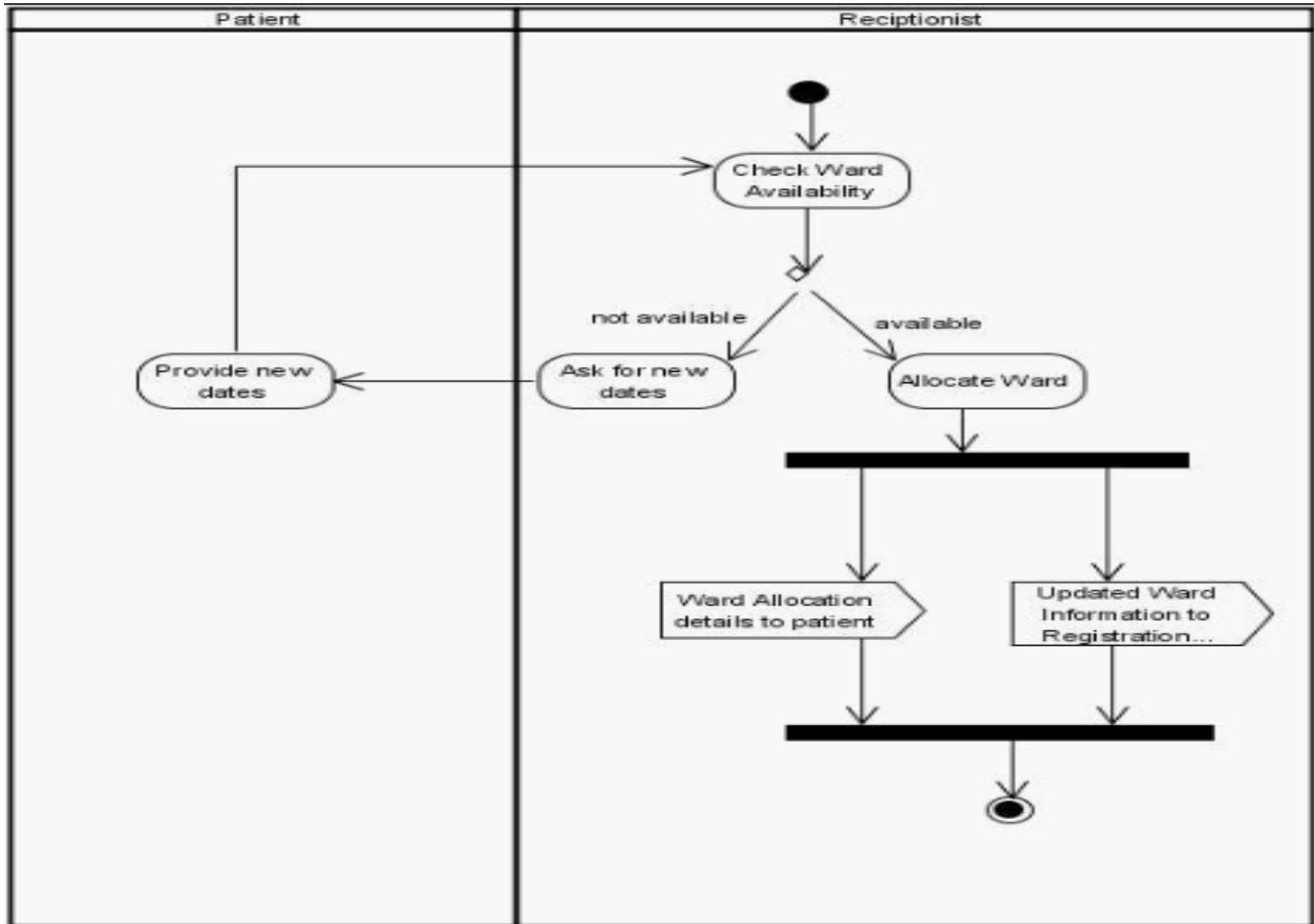


Swim lane Diagram: Hospital Management System

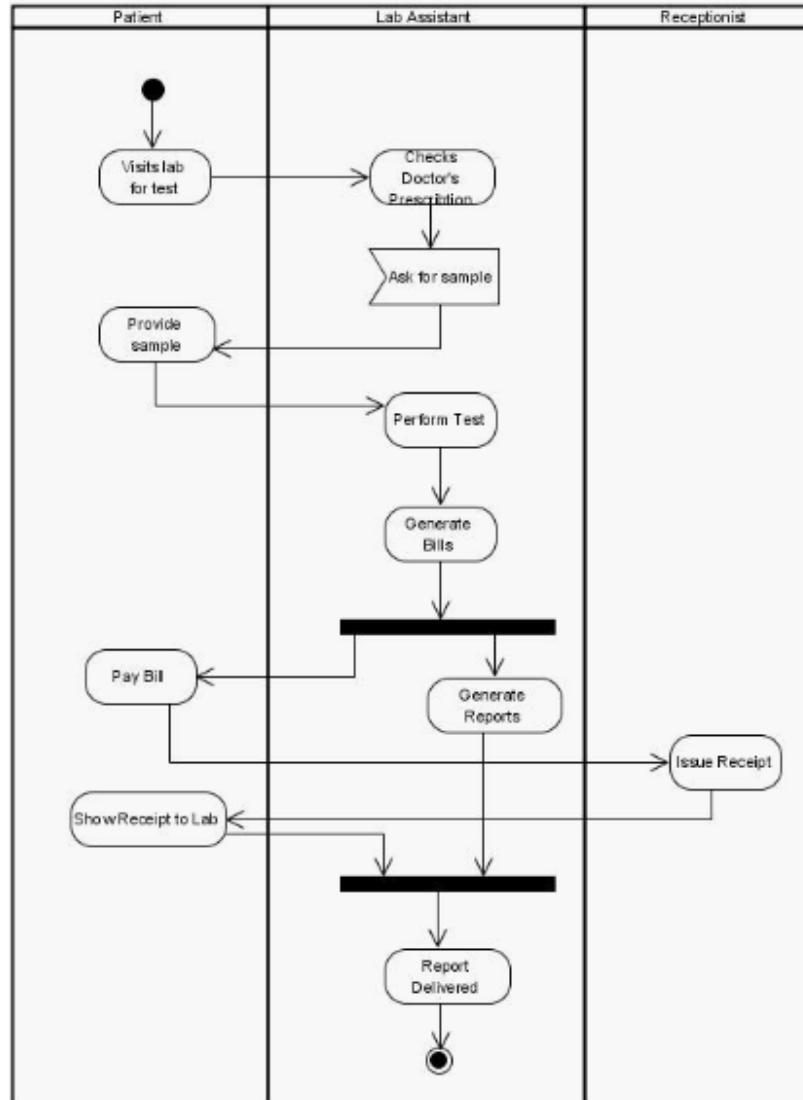
Swim lane Diagram:Registration



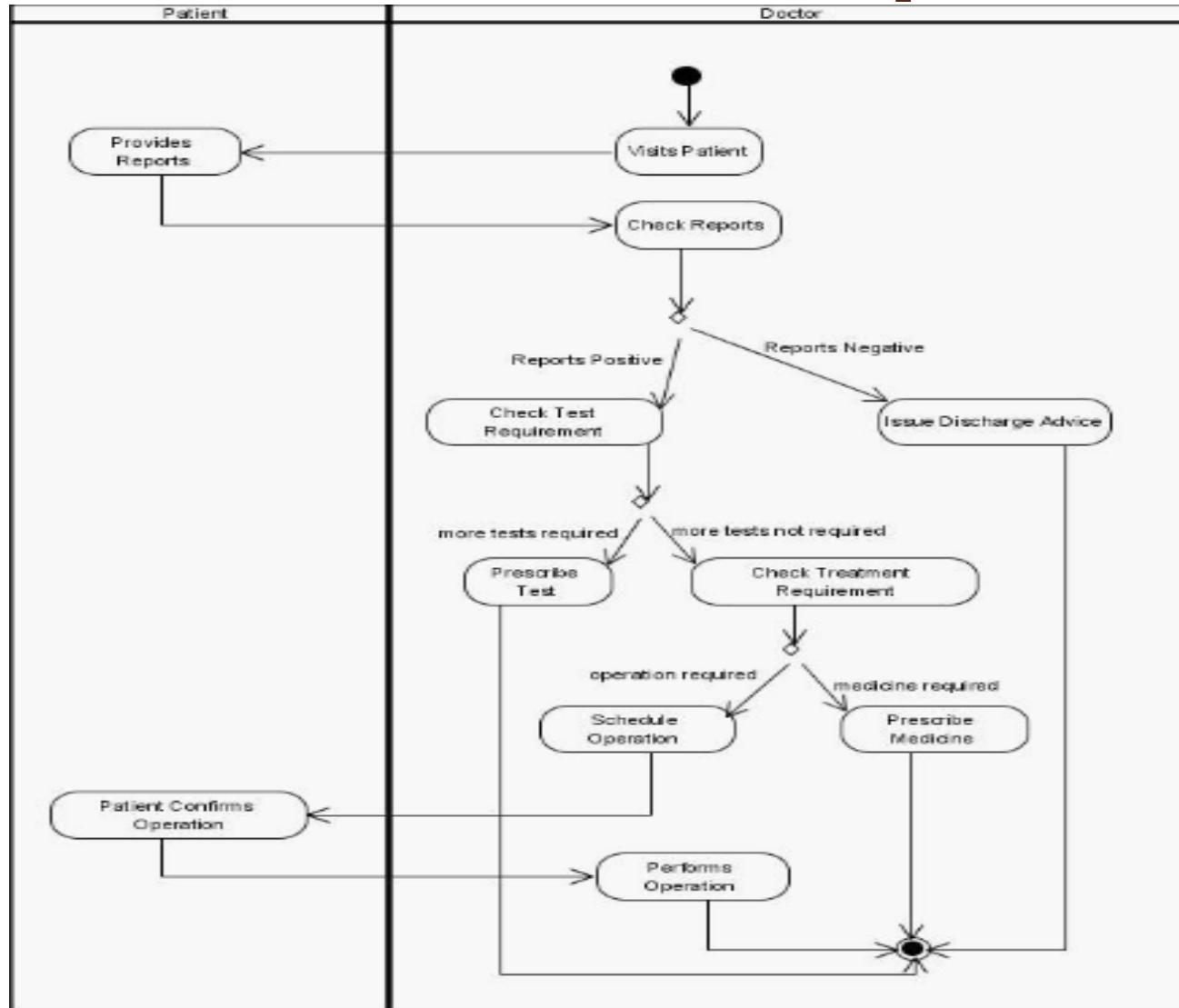
Swim lane Diagram for Ward Allocation:-



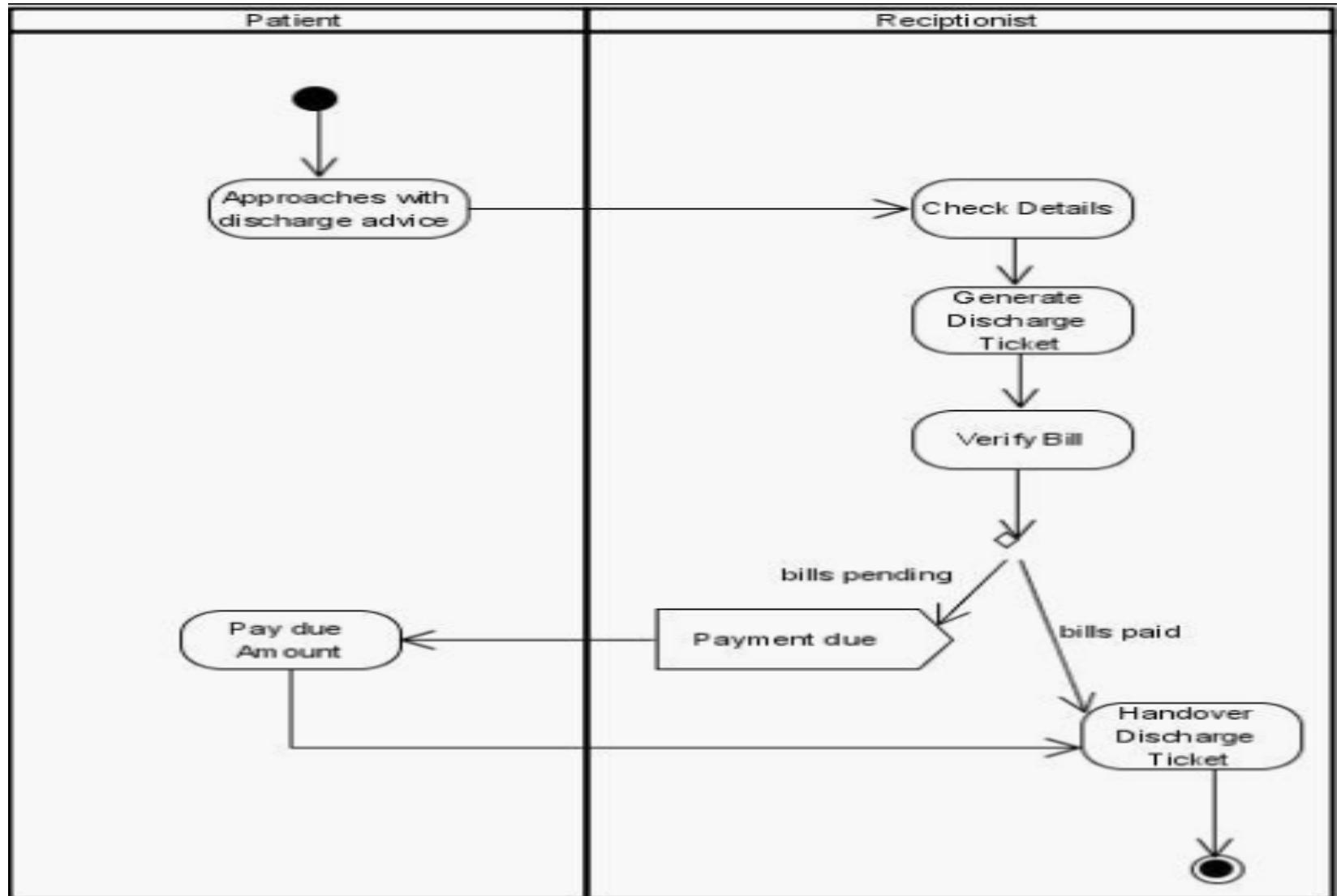
Swim lane Diagram for Tests to Perform:-



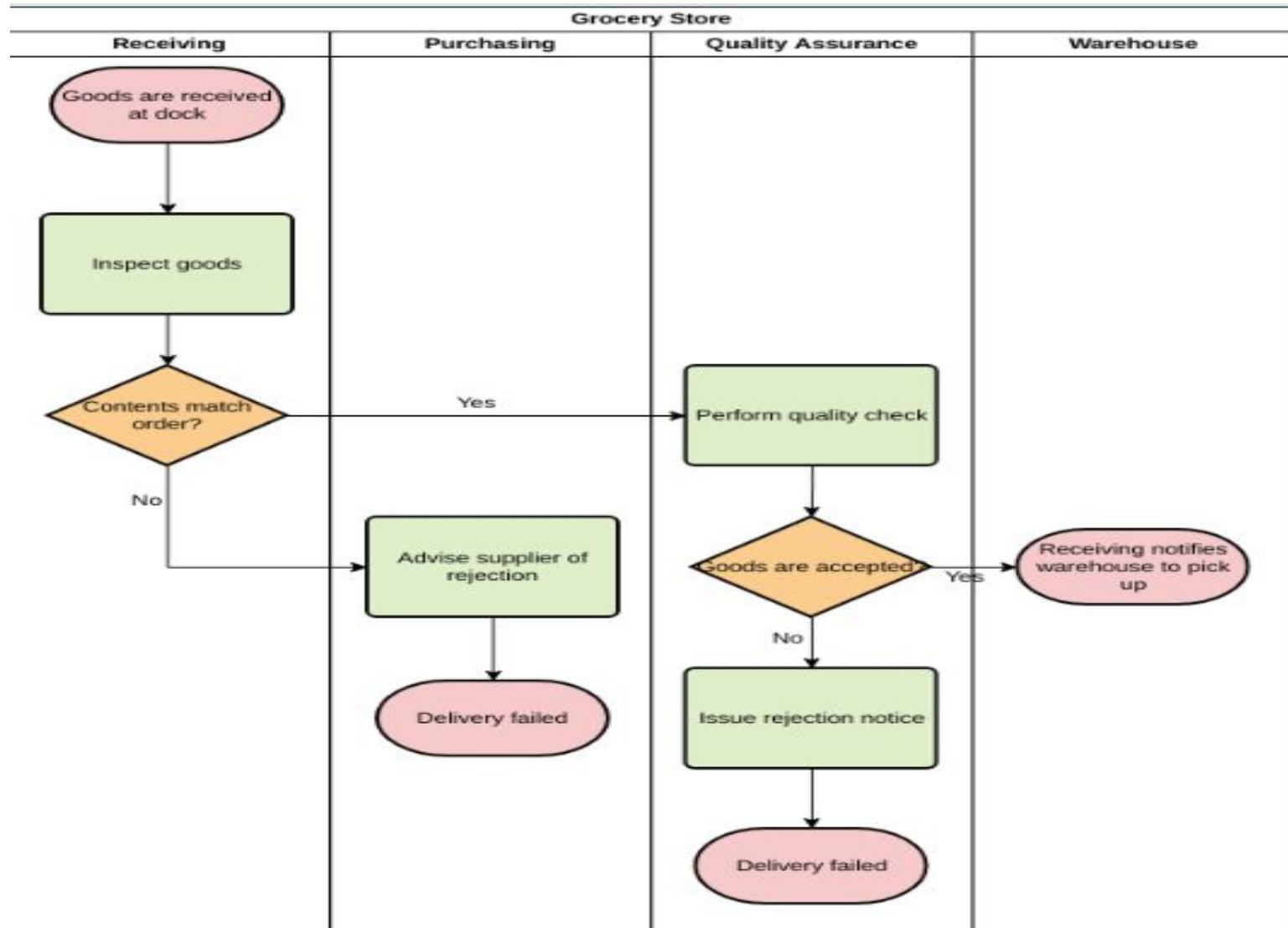
Swim lane Diagram for **Treatment and Operations:-**



Swim lane Diagram Discharge:-



Swim Lane Diagram for Receiving Goods



Swim Lane Diagram for Login System

Swimlane Diagram:

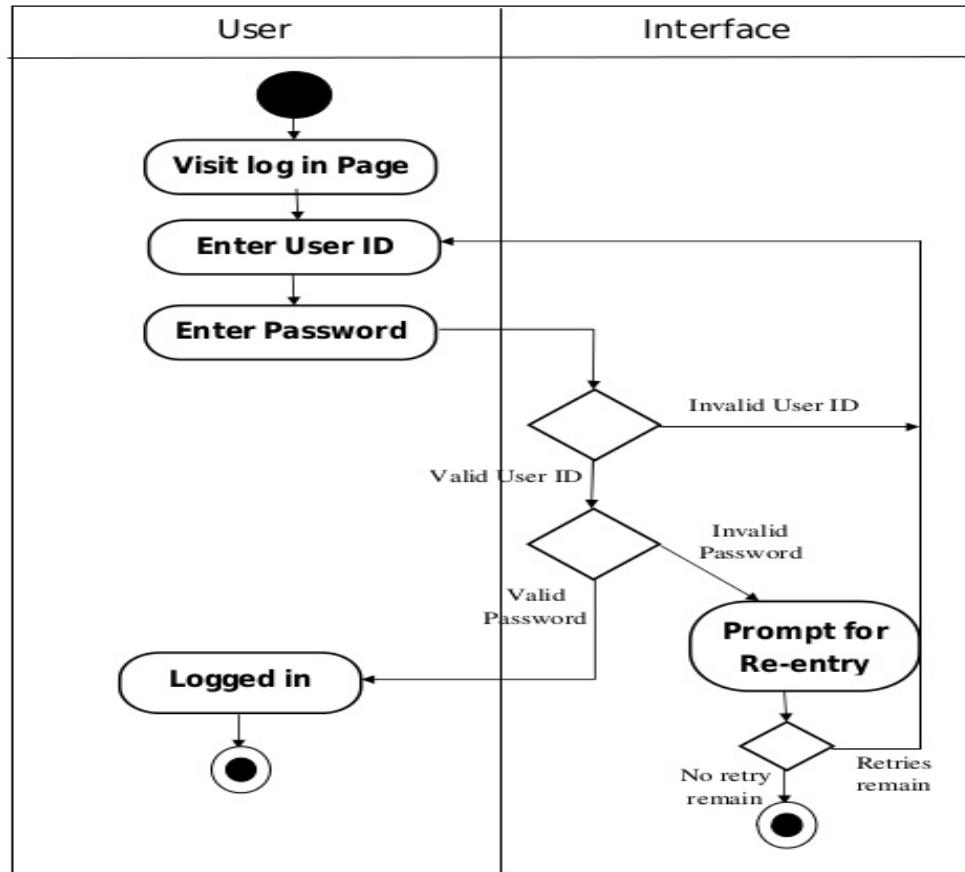
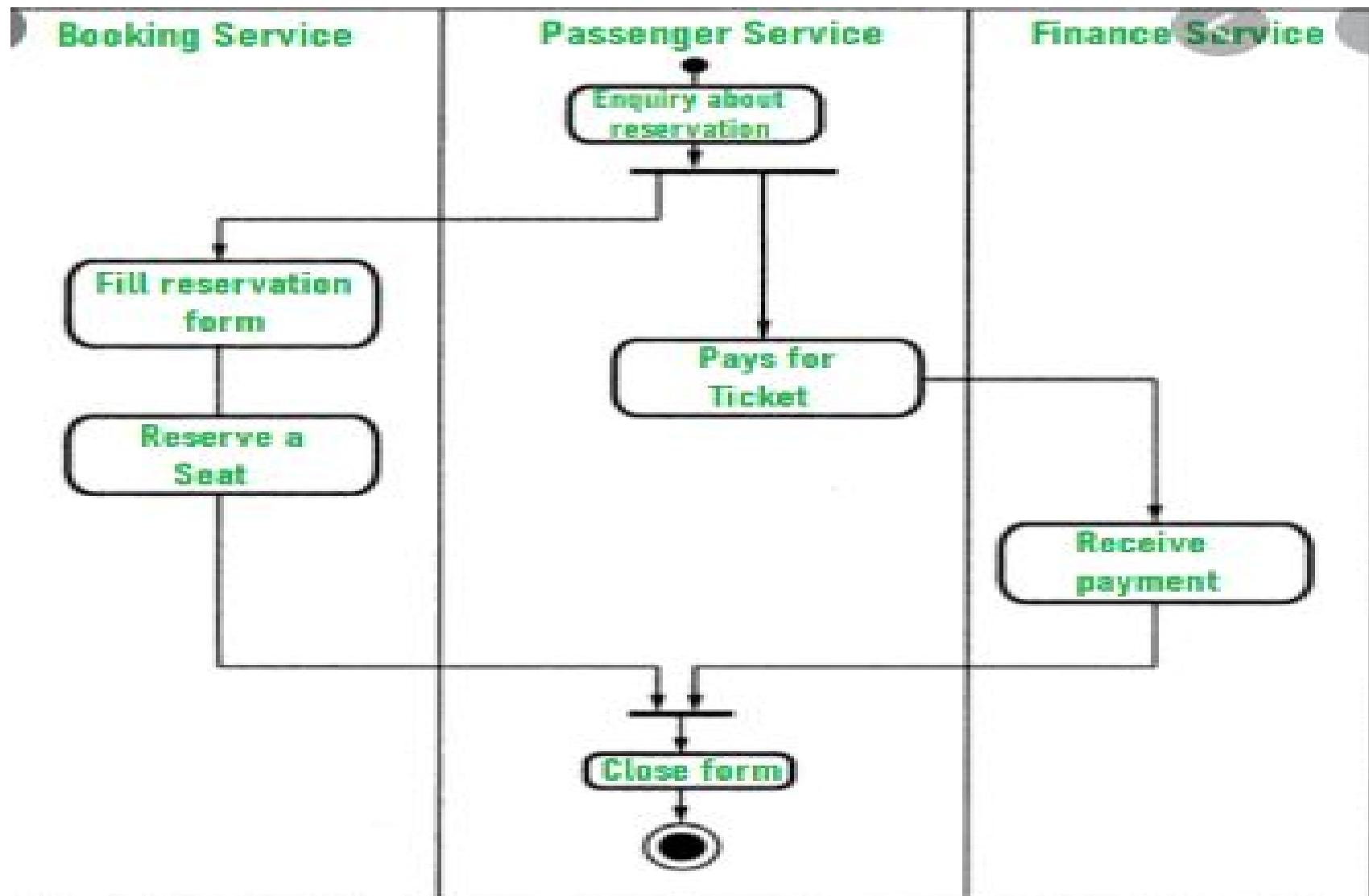


Fig 1.1: Swimlane Diagram of Sign In

Swimlane Diagram for Reserving a Ticket





Behavioural Model

- Sequence Diagram
- State Diagram

Introduction to Sequence diagram

- Sequence diagram is a type of behavioral representations in UML
- It indicates how events cause transitions from object to object
- Once events are identified by examining a use-case, the modeler creates a sequence diagram-a representation of how events cause flow from one object to another as a function of time
- Sequence diagram is an interaction diagram that shows the **set of objects and messages sent and received by those object.**
- It mainly emphasizes on **time ordering and messages.**
- Sequence diagram is a shorthand version of the use case
- It represents key classes and the events that cause behavior to flow from class to class

Activity Diagram And Sequence Diagram

- The **main difference** between activity diagram and sequence diagram is that the **activity diagram represents the flow of activities one after the other in a system** while **the sequence diagram represents the sequence of messages flowing from one object to another.**
- **Activity diagram is focused on Actions within the behavior.** **Sequence diagram is focused on Interactions** (communication between objects) within the behavior.
- The main focus in an **activity diagram is the flow of activities** whereas the main focus in a **sequence diagram is the interaction between objects over a specific period of time.**

Terms and Concepts

Objects or Participants :-

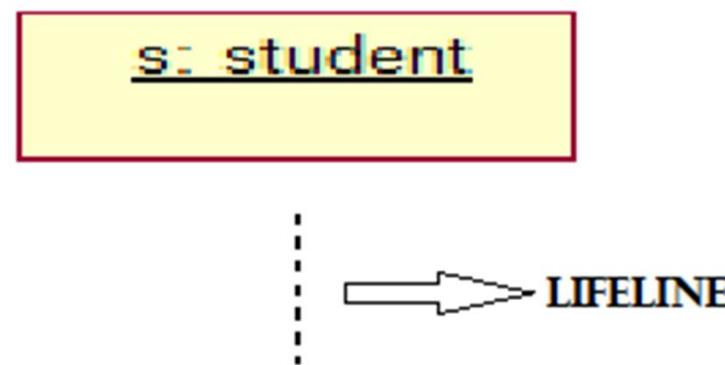
- The sequence diagram is made up of collection of **participants or objects**.
- Participants are system parts that interacts with each other during sequence diagram.
- The participants interact with each other by sending and receiving message
- The object is represented as shown below

Object:Class Name

Terms and Concepts

□ Lifeline:-

- Lifeline represents the existence of an object over a period of time.
- It is represented by vertical dashed line.

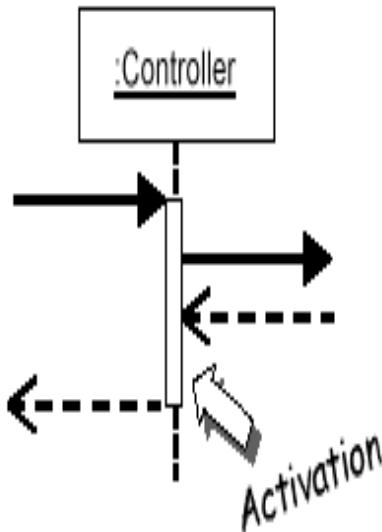


Terms and Concepts

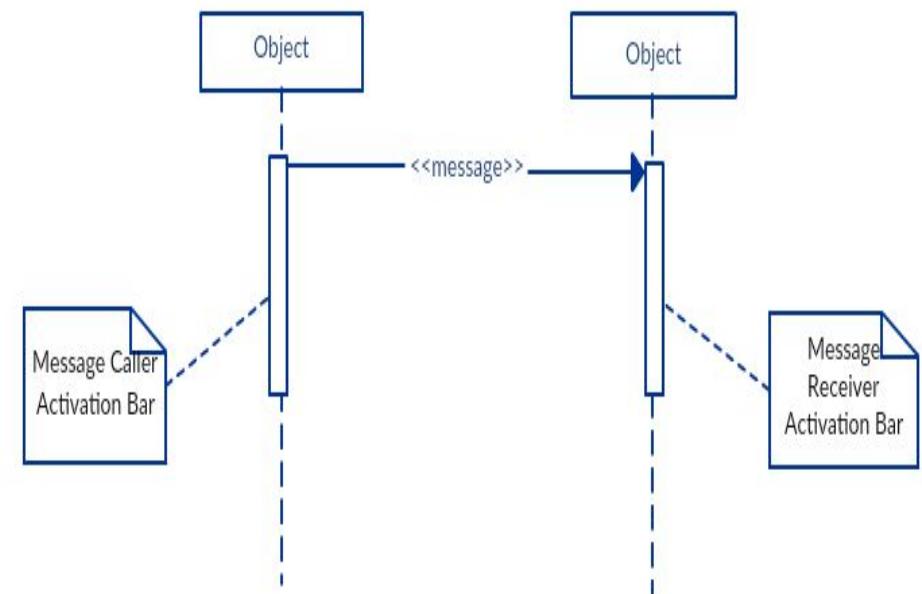
□ Activation bar:-

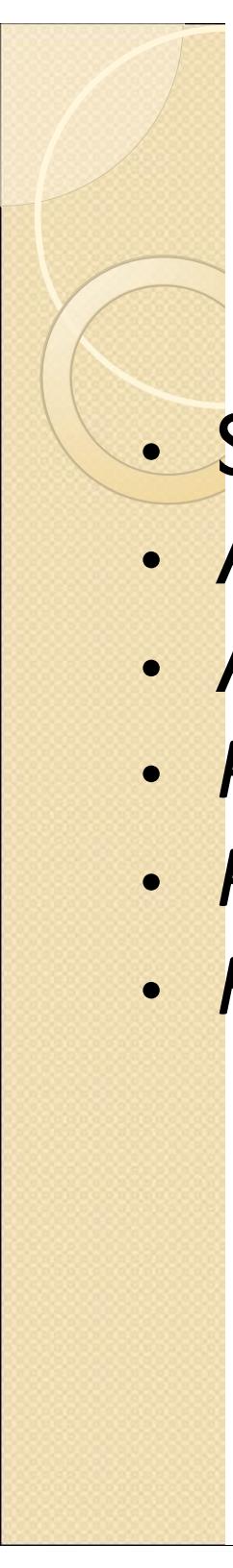
- It is also called as focus of control. It shows the period of time during which an object is performing an action.
- It is represented by tall thin rectangle:

Activation (Focus of Control)



The period of time when
the object is executing and/or
waiting for a return message





Types of Messages

- Synchronous message
- Asynchronous message
- A return message
- *Participant creation message*
- *Participant destruction message*
- *Reflexive message*



Synchronous message

- A synchronous message waits for a reply before the interaction can move forward.
- The sender waits until the receiver has completed the processing of the message.
- The caller continues only if it receives a reply message(when it knows that the receiver has processed the previous message)

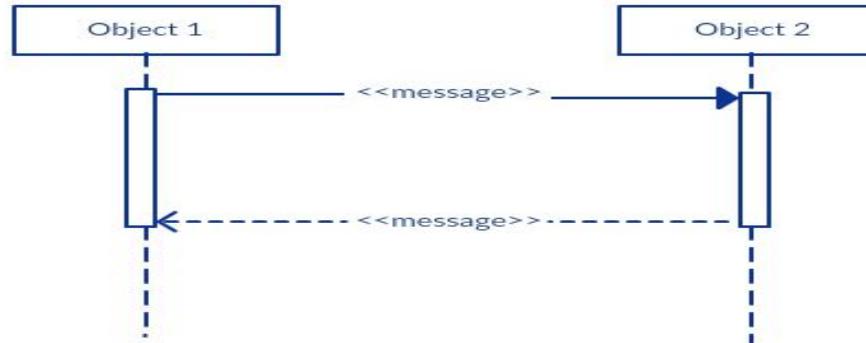


Asynchronous message

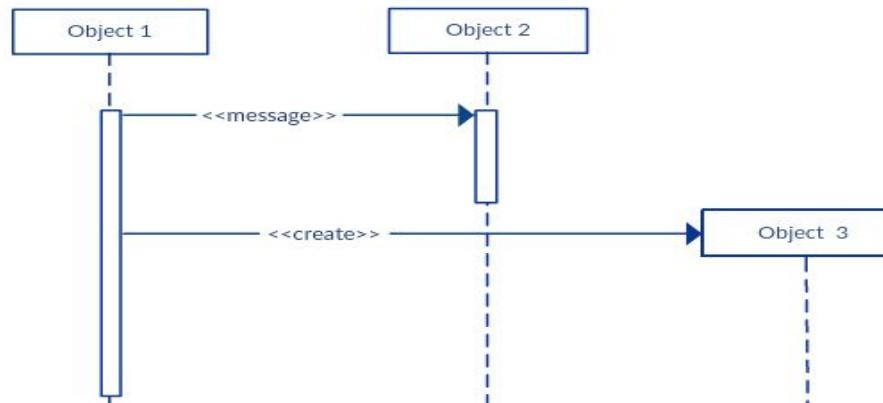
- An asynchronous message does not wait for a reply from the receiver.
- The interaction moves forward irrespective of the receiver processing the previous message or not.

Types of Messages

- A **return message** is used to indicate that the message receiver is done processing the message and is returning control over to the message caller.



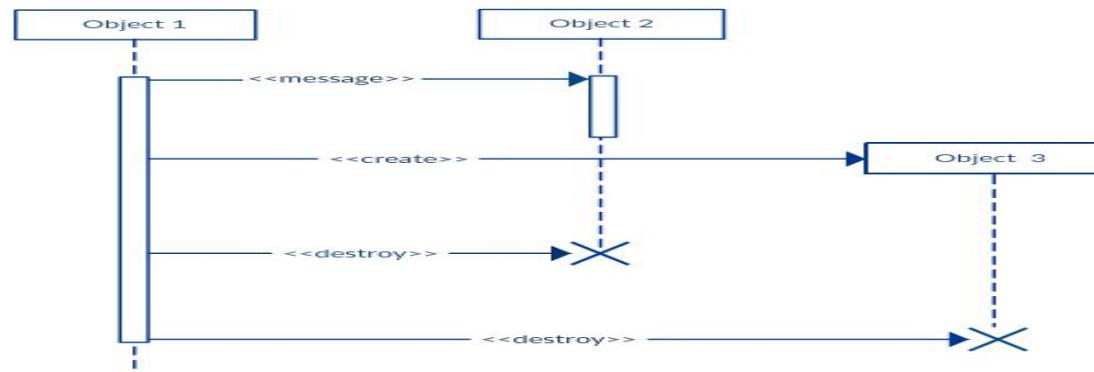
- **Participant creation message**
- Objects do not necessarily live for the entire duration of the sequence of events. Objects or participants can be created according to the message that is being sent.



Types Of Messages

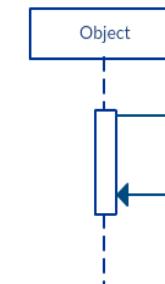
Participant destruction message

- participants when no longer needed can also be deleted from a sequence diagram. This is done by adding an ‘X’ at the end of the lifeline of the said participant.



Reflexive message

- When an object sends a message to itself, it is called a reflexive message. It is indicated with a message arrow that starts and ends at the same lifeline as shown in the example below.

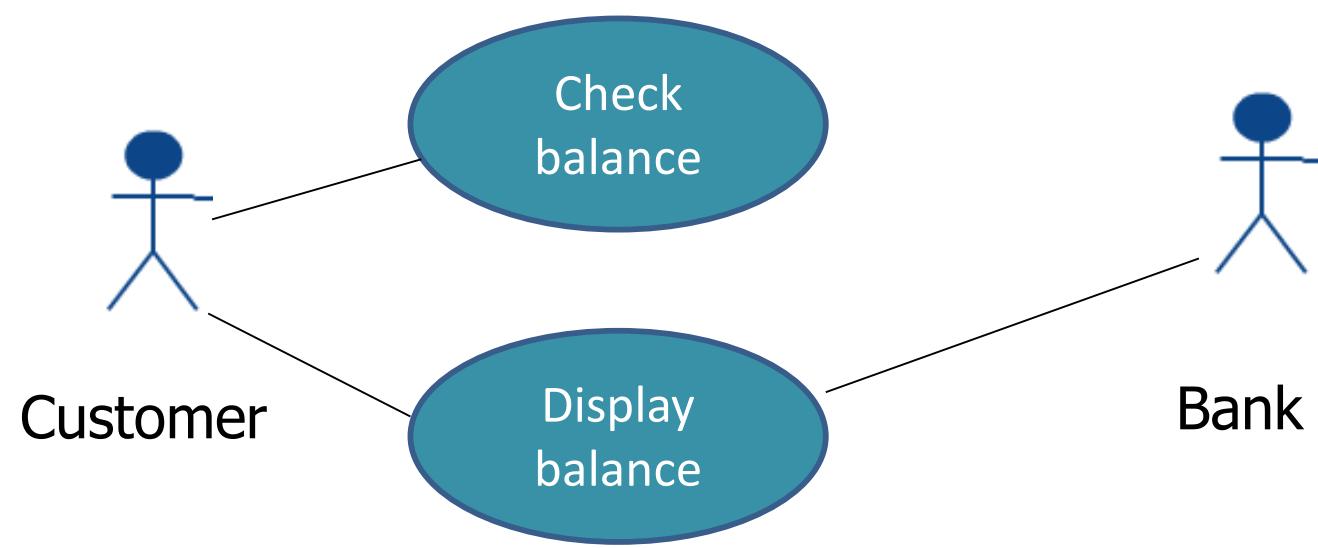


Exercise:

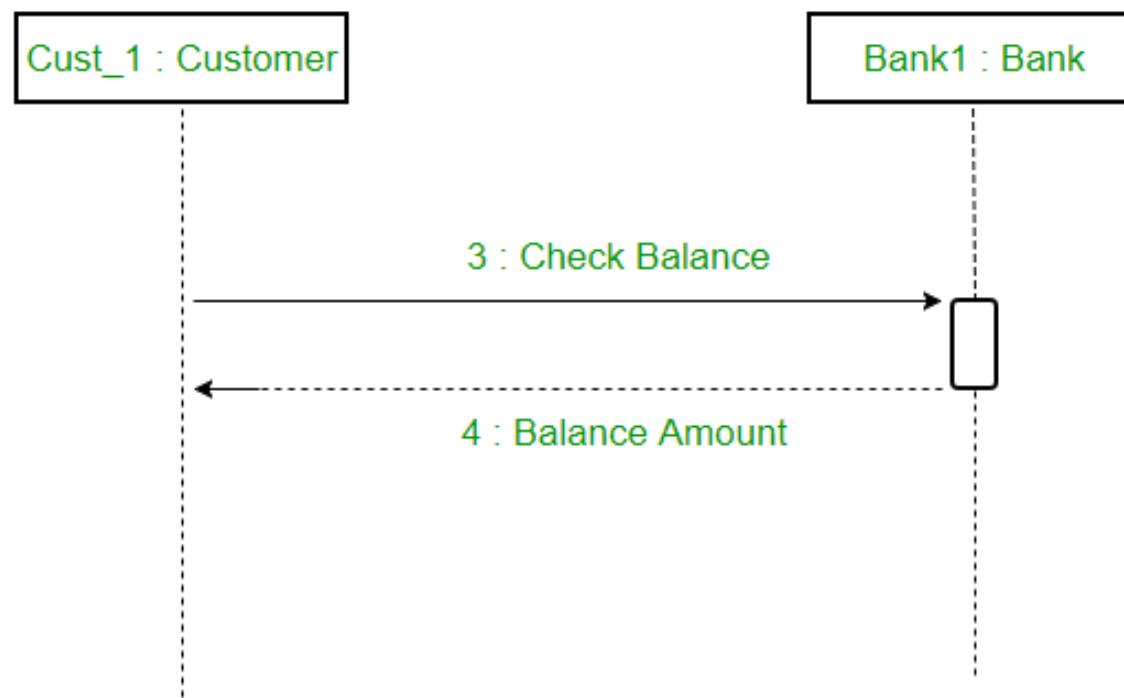
To generate sequence diagram using use case

Steps:-

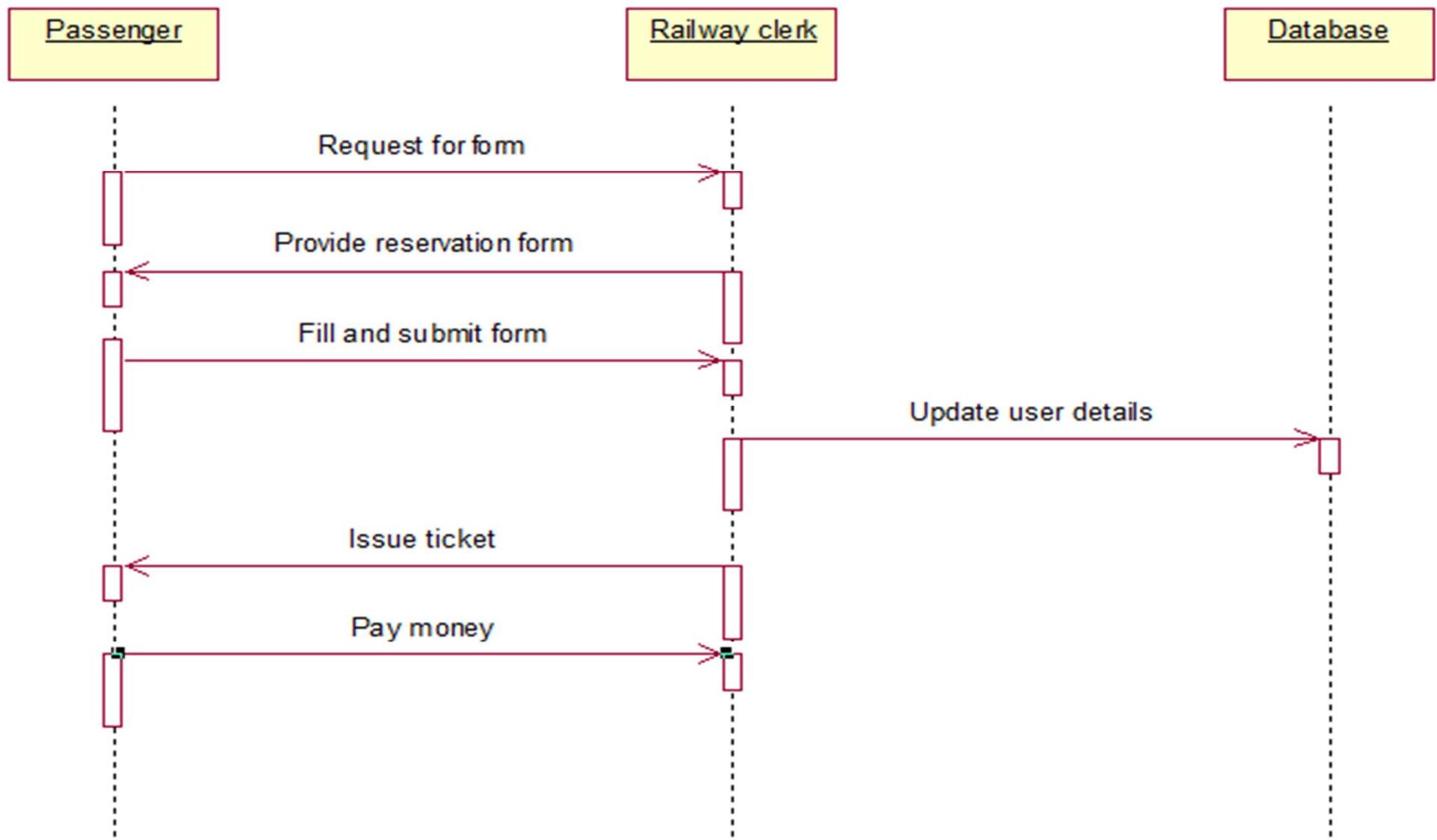
1. Designate actors and business system—Who is taking part?
2. Designate initiators—Who starts interactions?
3. Describe the message exchange between actors and business system—Which messages are being exchanged?
4. Identify the course of interactions—What is the order?
5. Insert additional information—What else is important?
6. Verify the view—Is everything correct?



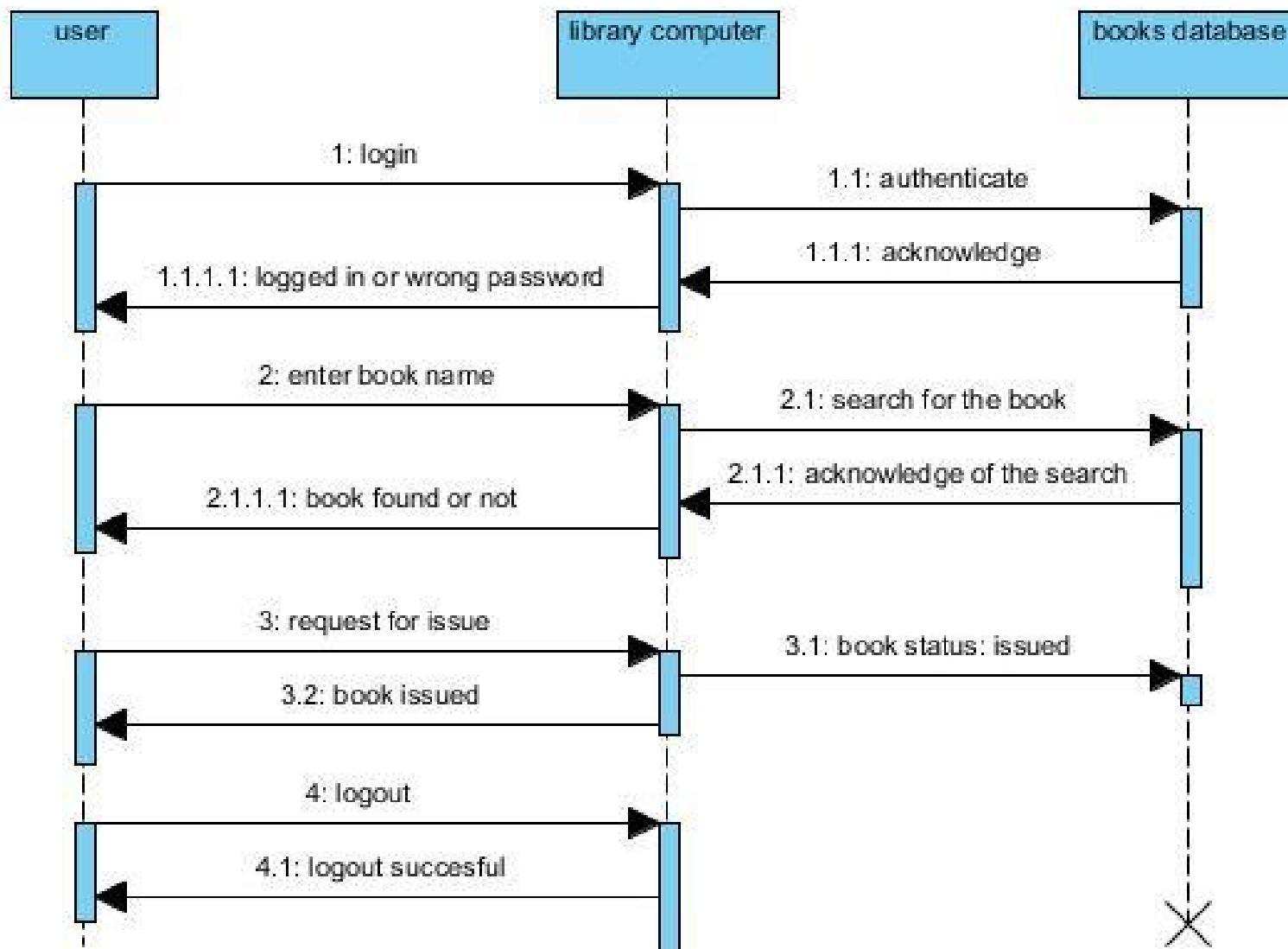
example



Sequence diagram of Railway reservation system

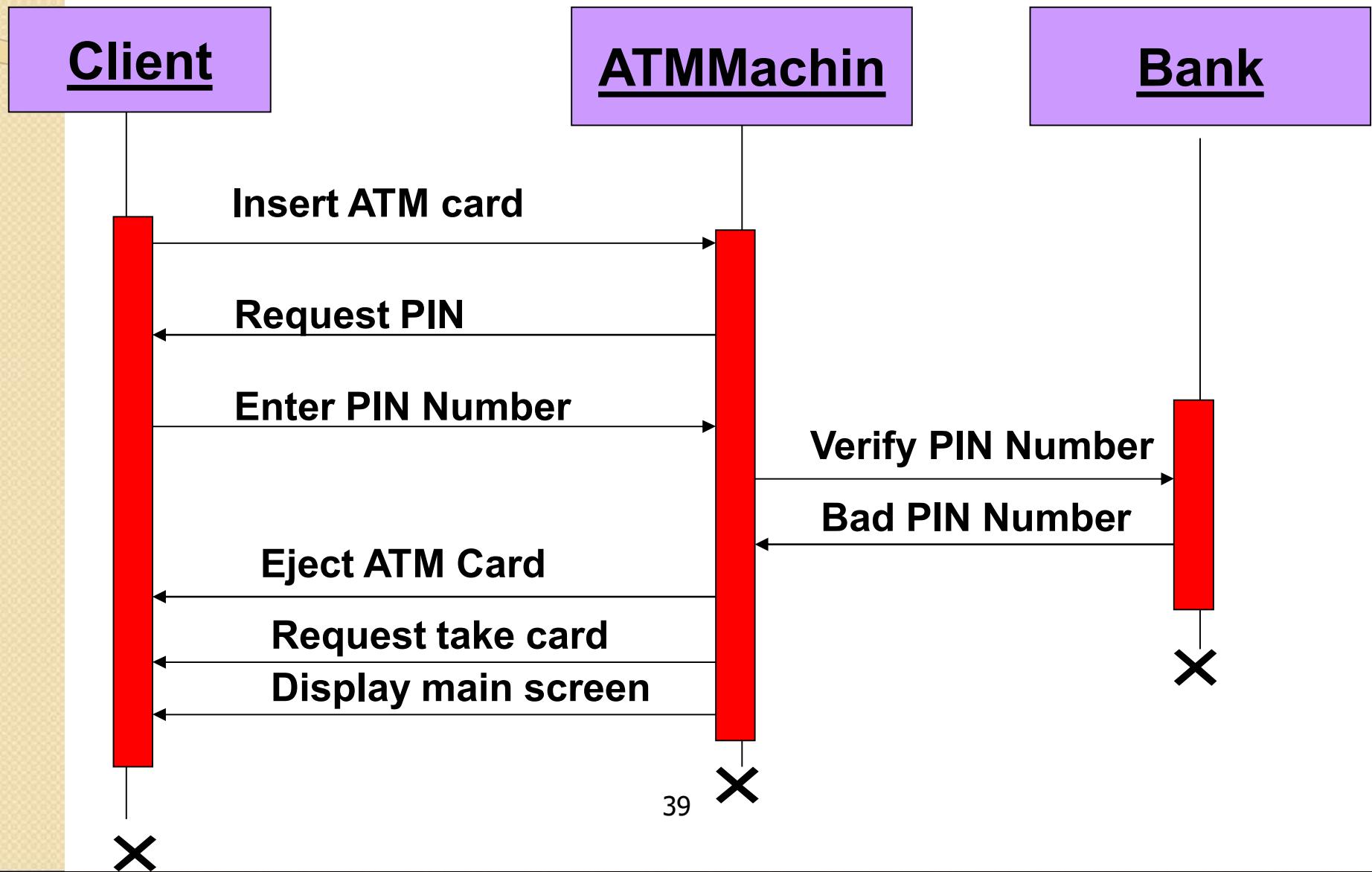


Library

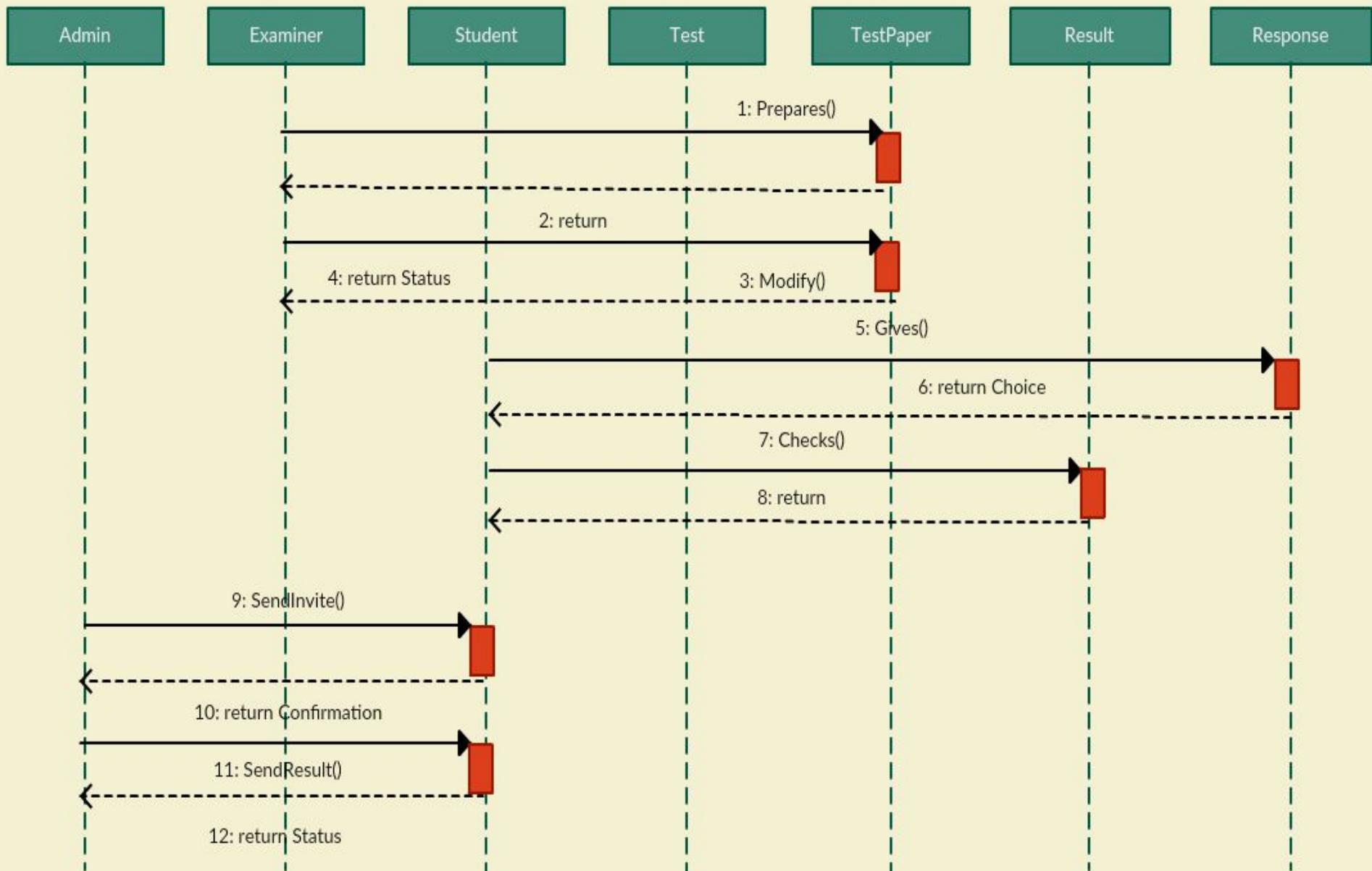


Sequence Diagram

Scenario: Invalid Pin Number



Sequence Diagram of an Online Exam System



Sequence Diagram

The Sequence diagram represents the UML, which is used to visualize the sequence of calls in a system that is used to perform a specific functionality.

The Sequence diagram shows the message flow from one object to another object.

Sequence diagram is used for the purpose of dynamic modelling.

Sequence diagram is used to describe the behavior of several objects in a single use case

Sequence diagram is mainly used to represent the time order of a process.

Activity Diagram

The Activity diagram represents the UML, which is used to model the workflow of a system.

The Activity diagram shows the message flow from one activity to another.

Activity diagram is used for the purpose of functional modelling.

Activity diagrams is used to describe the general sequence of actions for several objects and use cases.

Activity diagram is used to represent the execution of the process.



Behavioural Model : State Chart Diagram

State Chart Diagram

- State chart diagram describes different states of a component in a system. The states are specific to a component/object of a system.
- A Statechart diagram describes a state machine. State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.

Purpose of Statechart Diagrams

- Statechart diagram is one of the UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events.
- Statechart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

Purpose of Statechart Diagrams

- Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered.
- The most important purpose of Statechart diagram is to model lifetime of an object from creation to termination.

Following are the main purposes of using Statechart diagrams

- To model the dynamic aspect of a system.
- To model the life time of a reactive system.
- To describe different states of an object during its life time.
- Define a state machine to model the states of an object.

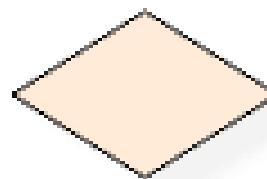
Notation and Symbol for State Machine



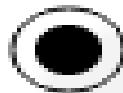
initial state



state-box



decision-box



final-state

How to Draw a Statechart Diagram?

- Statechart diagram is used to describe the states of different objects in its life cycle. Emphasis is placed on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately.
- Statechart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.



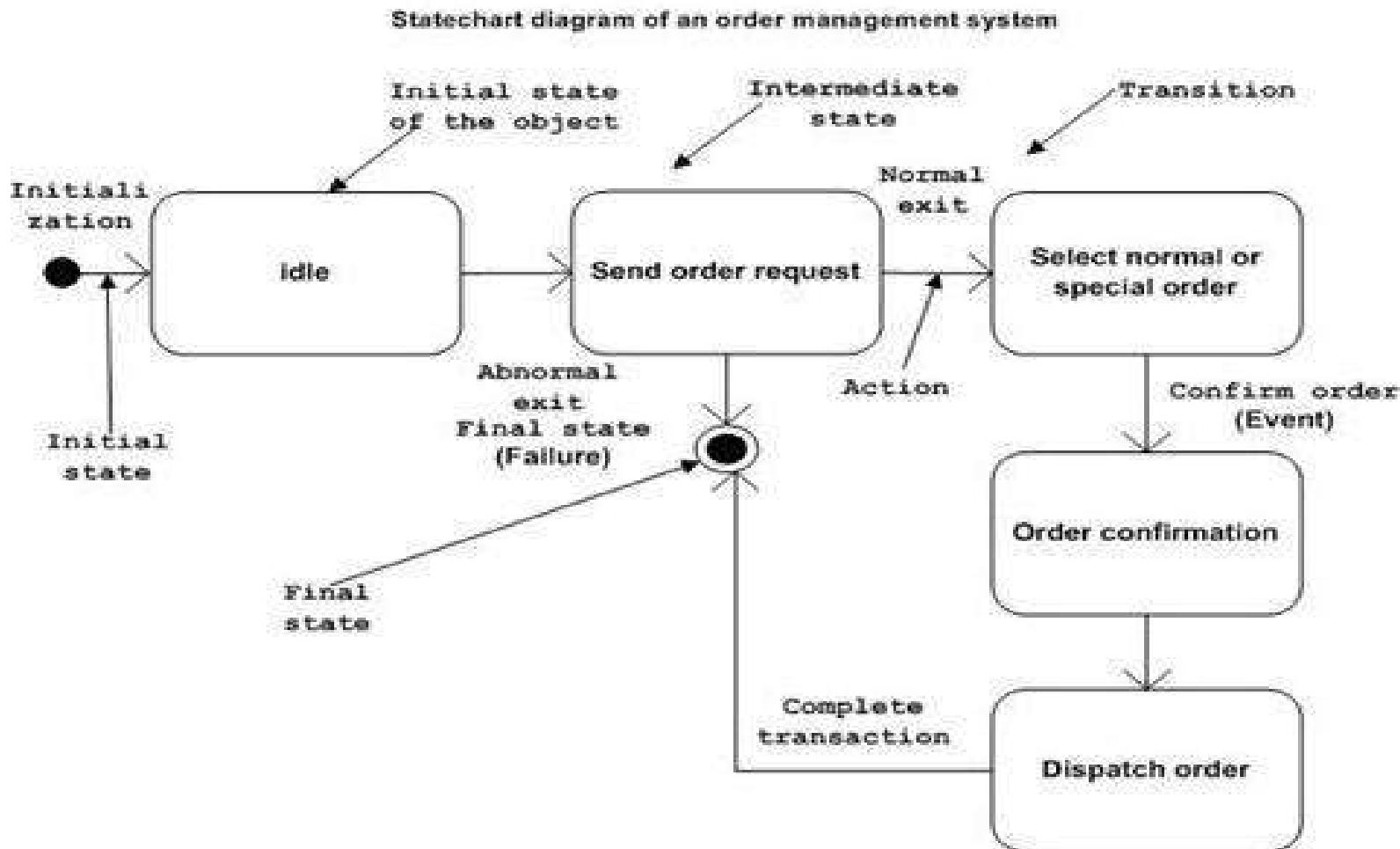
How to Draw a Statechart Diagram?

- Before drawing a Statechart diagram we should clarify the following points –
- Identify the important objects to be analyzed.
- Identify the states.
- Identify the events.

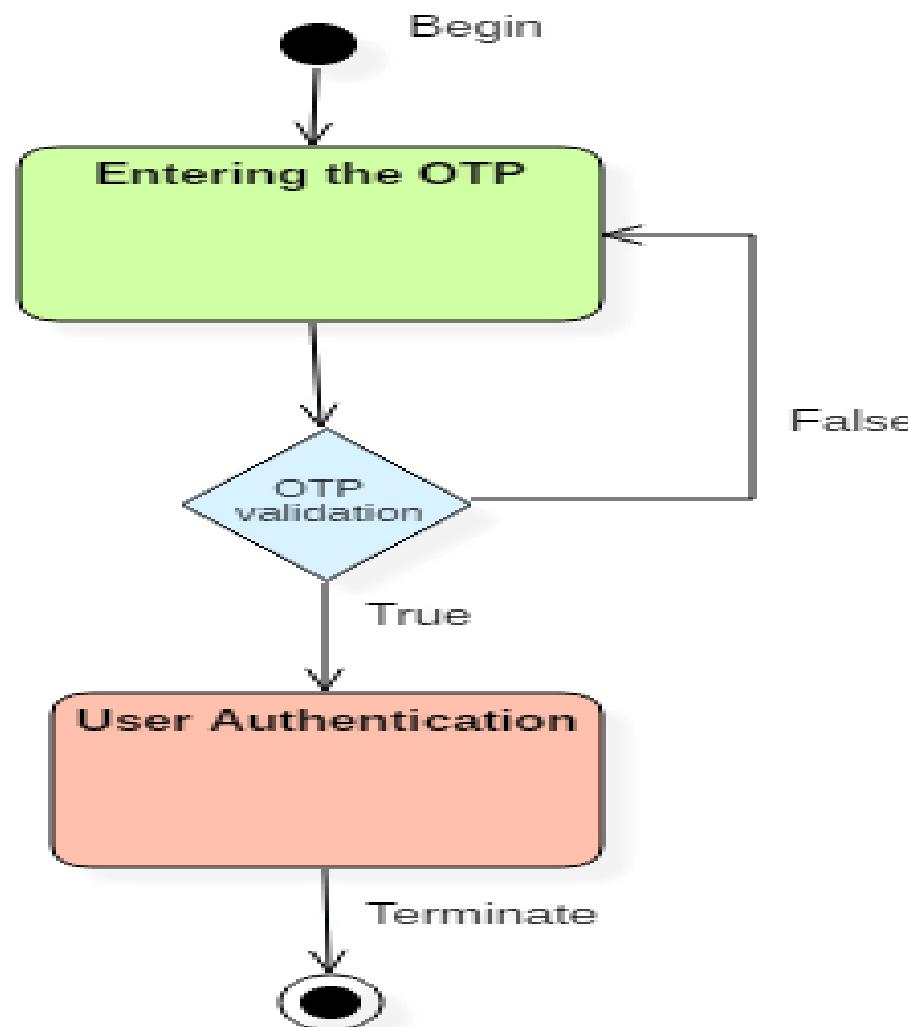
Following is an example of a Statechart diagram where the state of Order object is analyzed

- The first state is an idle state from where the process starts. The next states are arrived for events like send request, confirm request, and dispatch order. These events are responsible for the state changes of order object.
- During the life cycle of an object (here order object) it goes through the following states and there may be some abnormal exits. This abnormal exit may occur due to some problem in the system. When the entire life cycle is complete, it is considered as a complete transaction as shown in the following figure. The initial and final state of an object is also shown in the following figure.

State chart diagram of an order management system



State chart diagram represents the user authentication process.





Class-based Model

Class Diagram

5/25/2021

Class Diagram

- Class diagram is a static diagram.
- It represents the static view of an application.
- Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.
- Class diagram describes the attributes and operations of a class and also the constraints imposed on the system.

5/25/2021

A UML class diagram is made up of:

- A set of classes and
- A set of relationships between classes

What is a Class

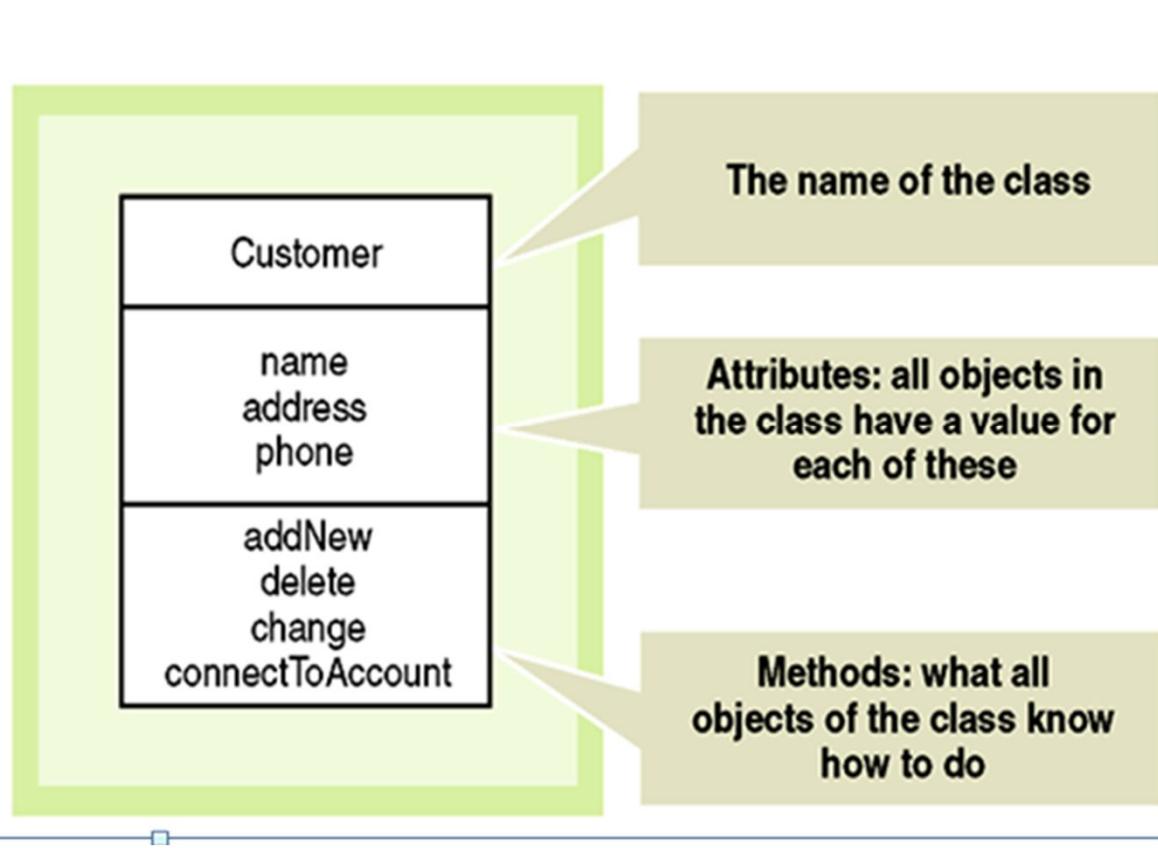
- A description of a group of objects all with similar roles in the system, which consists of:
- **Structural features** (attributes) define what objects of the class "know"
 - Represent the state of an object of the class
 - Are descriptions of the structural or static features of a class
- **Behavioral features** (operations) define what objects of the class "can do"
 - Define the way in which objects may interact
 - Operations are descriptions of behavioral or dynamic features of a class

Class Diagram

- Classes: Entities with common features, i.e. attributes and operations.
- Represented as solid outline rectangle with compartments.
- Compartments for **name, attributes, and operations**.
- Attribute and operation compartments are optional depending on the purpose of a diagram.

UML Class Representation

- A class represents a set of objects having similar attributes, operations, relationships and behavior.



Different representations of the LibraryMember class

LibraryMember

Member Name
Membership Number
Address
Phone Number
E-Mail Address
Membership Admission Date
Membership Expiry Date
Books Issued

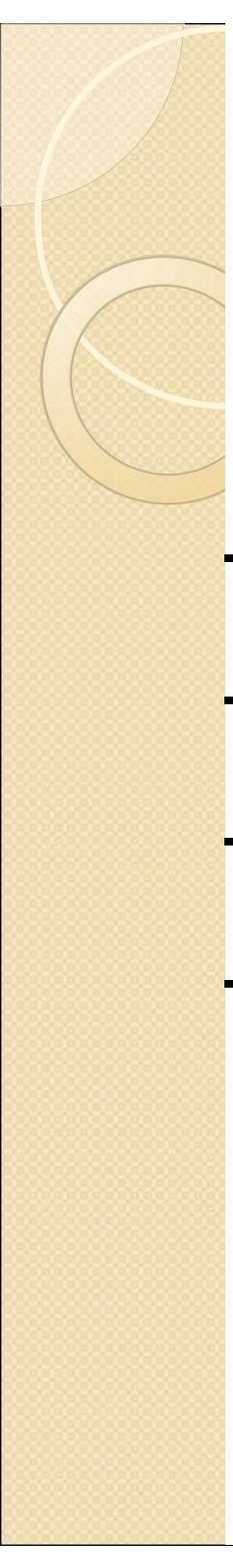
issueBook();
findPendingBooks();
findOverdueBooks();
returnBook();
findMembershipDetails();

LibraryMember

issueBook();
findPendingBooks();
findOverdueBooks();
returnBook();
findMembershipDetails();

LibraryMember

Example UML
Classes

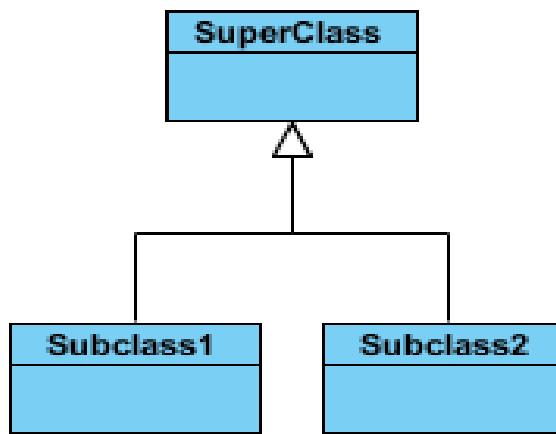


What are the Different Types of Relationships Among Classes?

- Inheritance(Generalization ,Specialization)**
- Simple Association**
- Aggregation/Composition**
- Dependency**

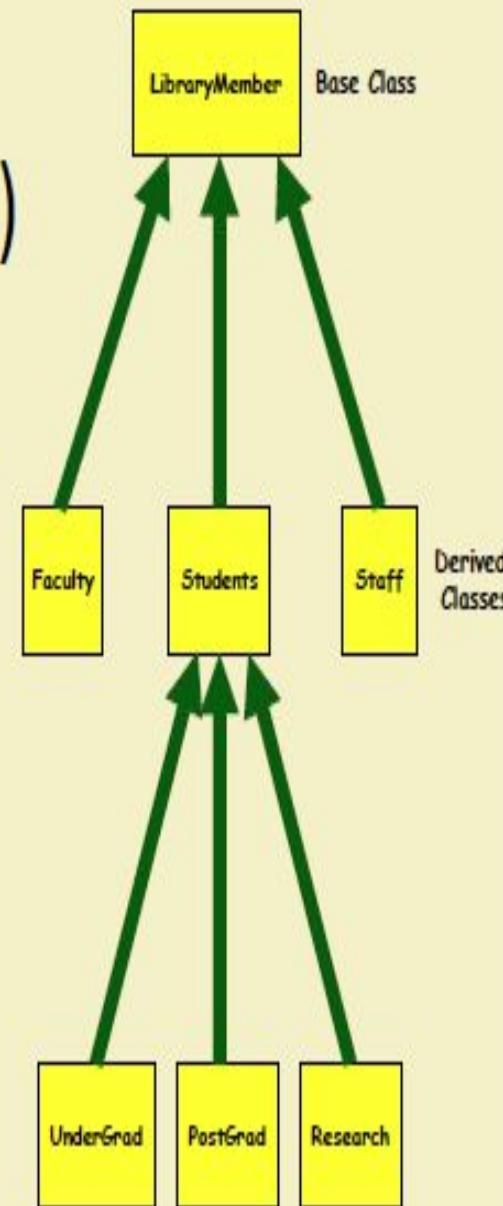
Inheritance (or Generalization):

- Represents an "is-a" relationship.
- SubClass1 and SubClass2 are specializations of Super Class.
- A solid line with a hollow arrowhead that point from the child to the parent class

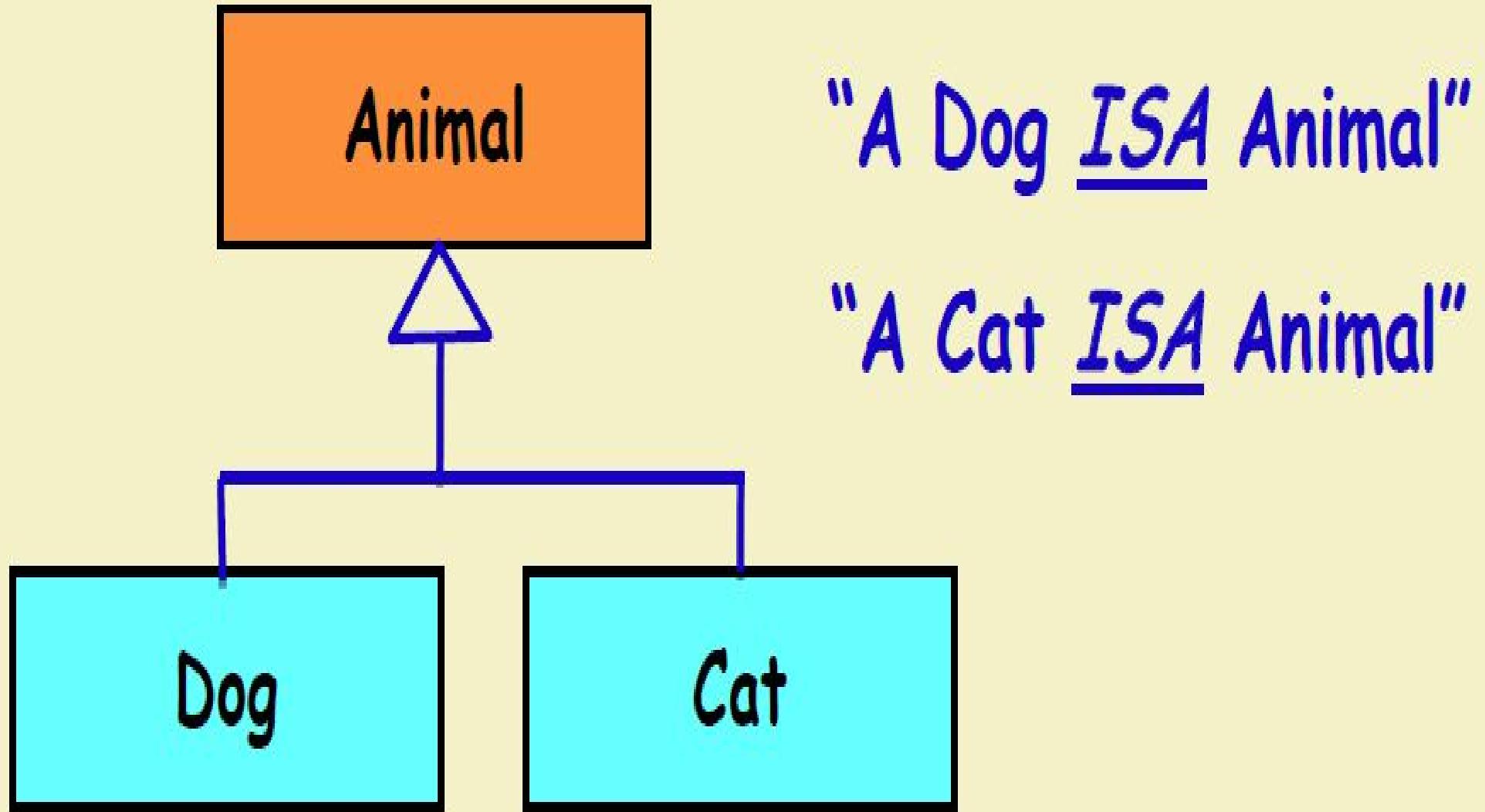


Inheritance

- Allows to define a new class (derived class) by extending an existing class (base class).
 - Represents generalization-specialization
 - Allows redefinition of the existing methods (method overriding).

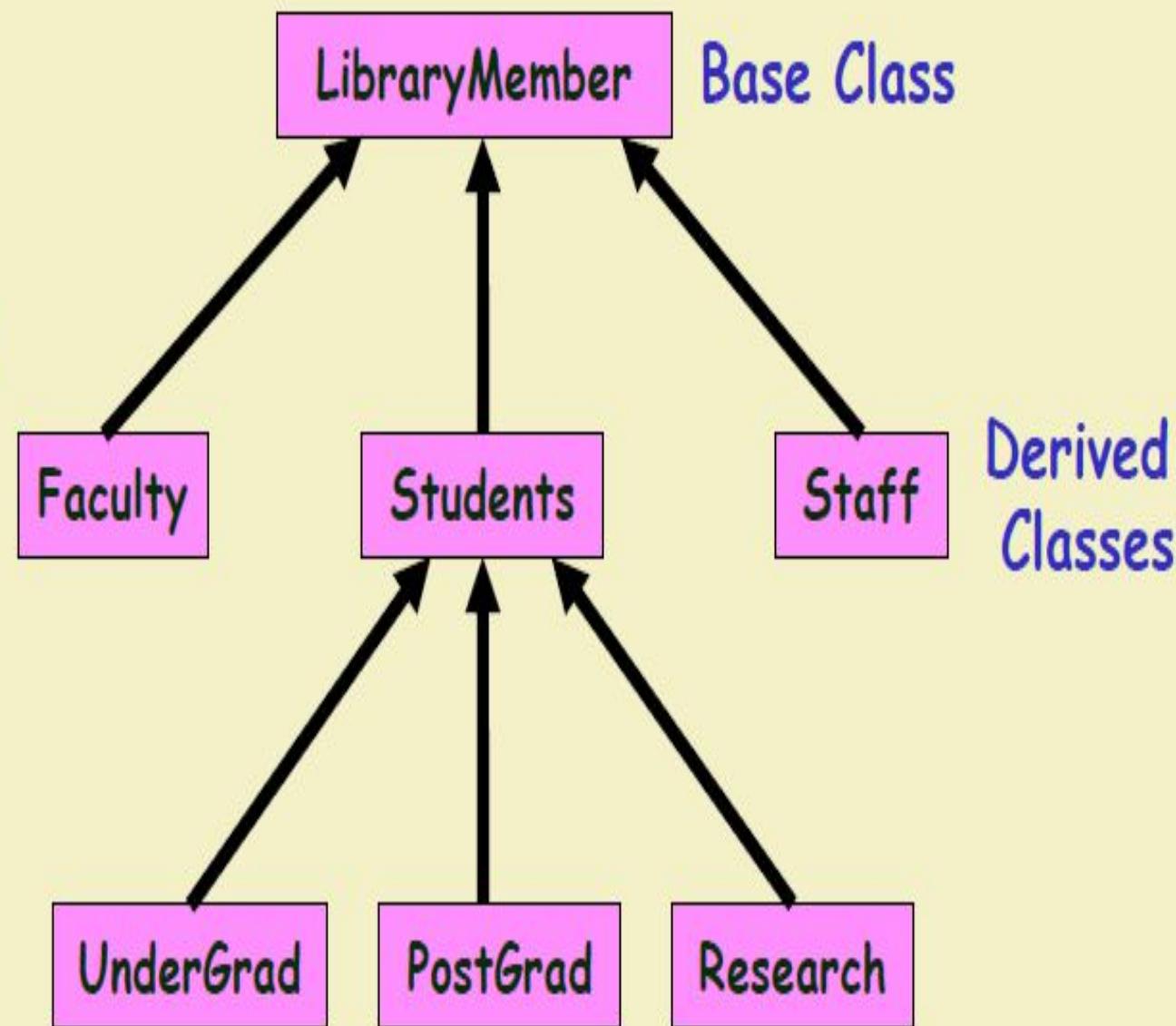


Inheritance Example



Inheritance

- Lets a subclass inherit attributes and methods from a base class.



A Generalization/Specialization Hierarchy for Motor Vehicles

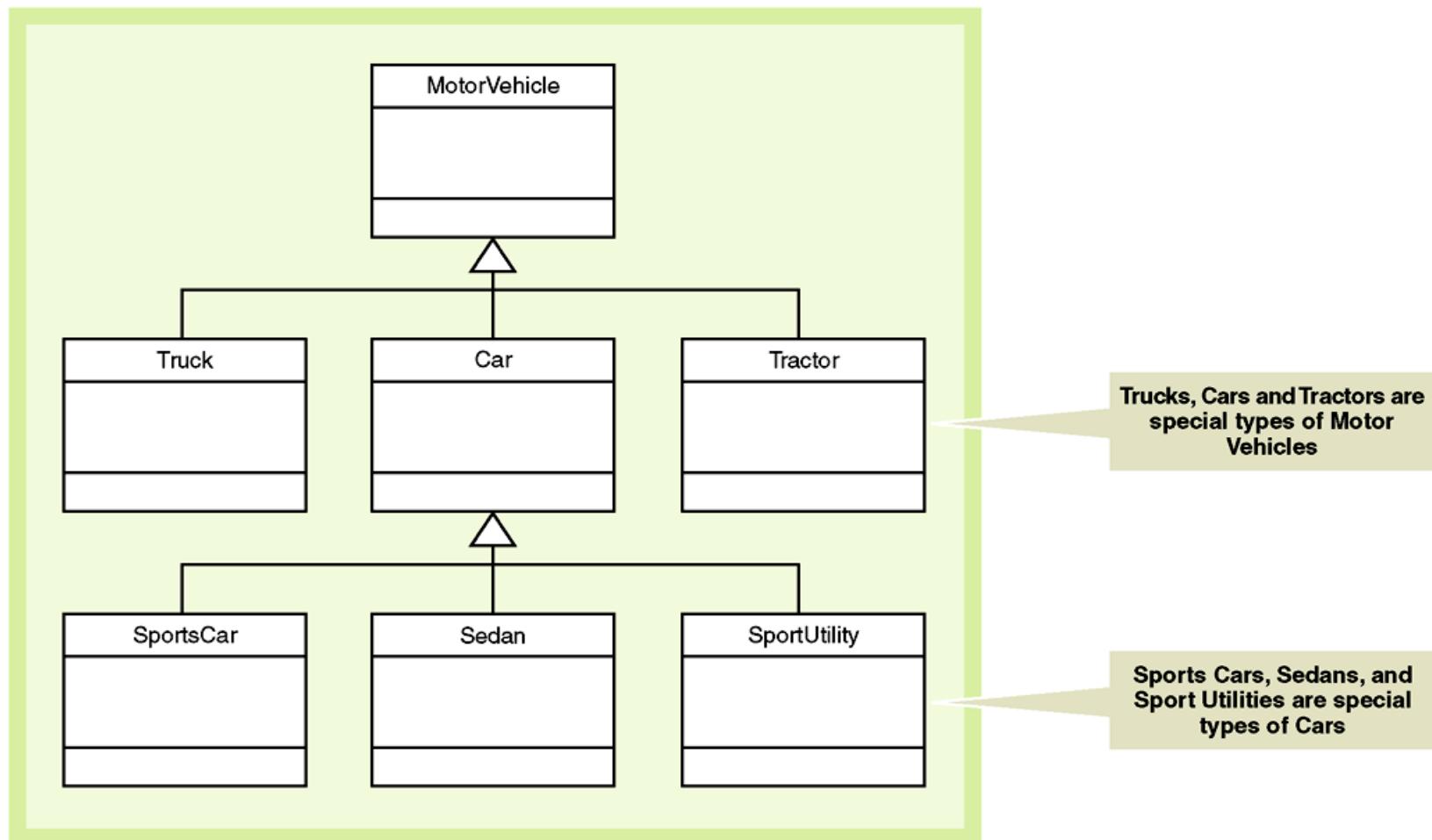


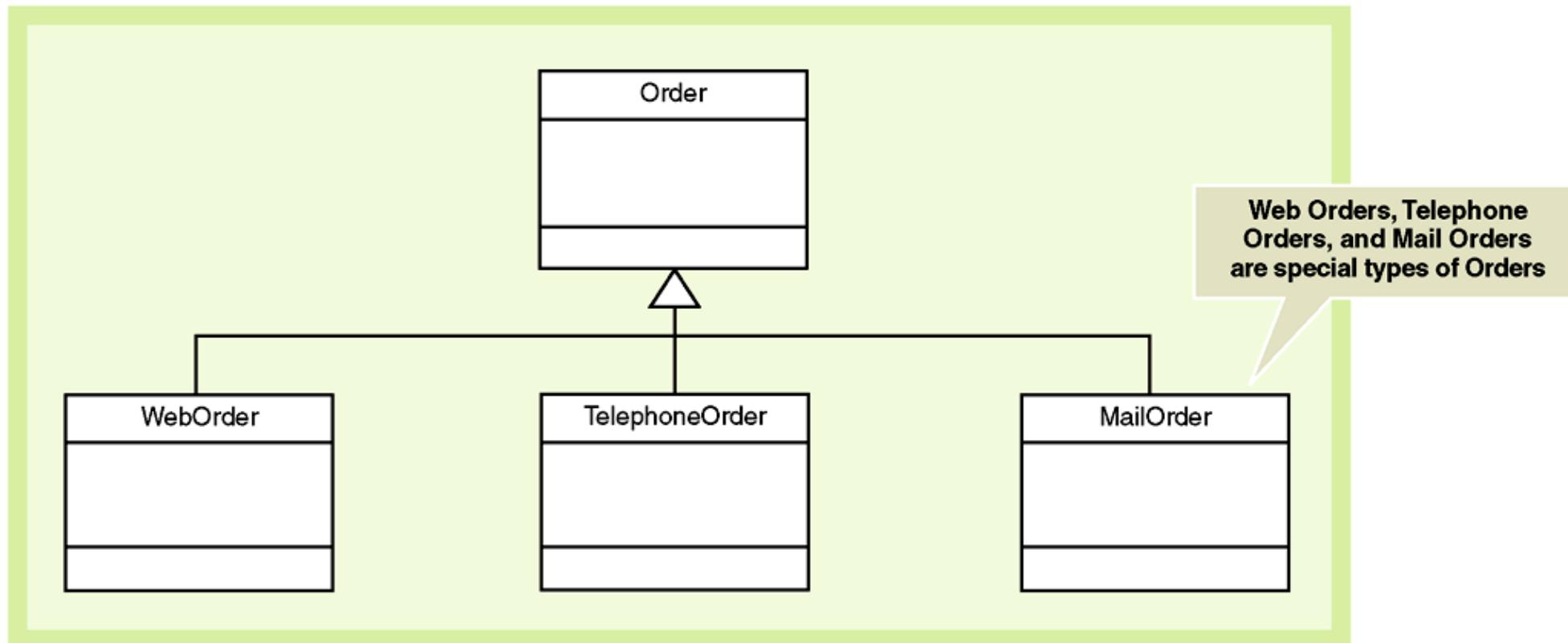
FIGURE 5-29

A generalization/specialization hierarchy for motor vehicles.

A Generalization/Specialization Hierarchy for Orders

FIGURE 5-30

A generalization/specialization hierarchy for orders.

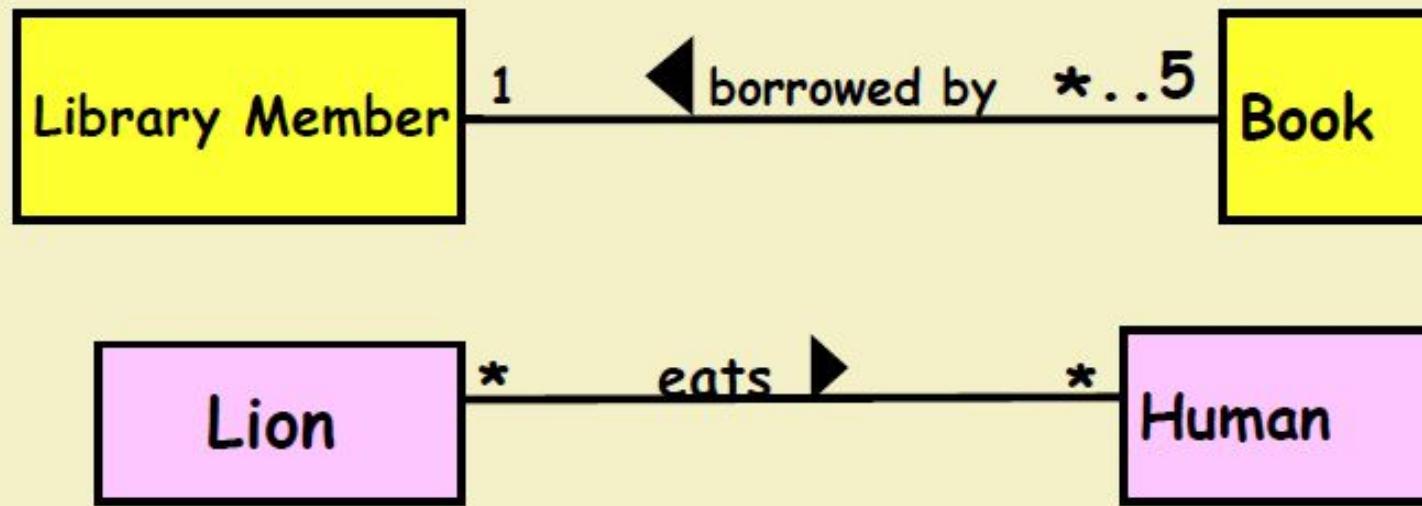


Simple Association:

- A structural link between two peer classes.
- There is an association between Class1 and Class2
- A solid line connecting two classes



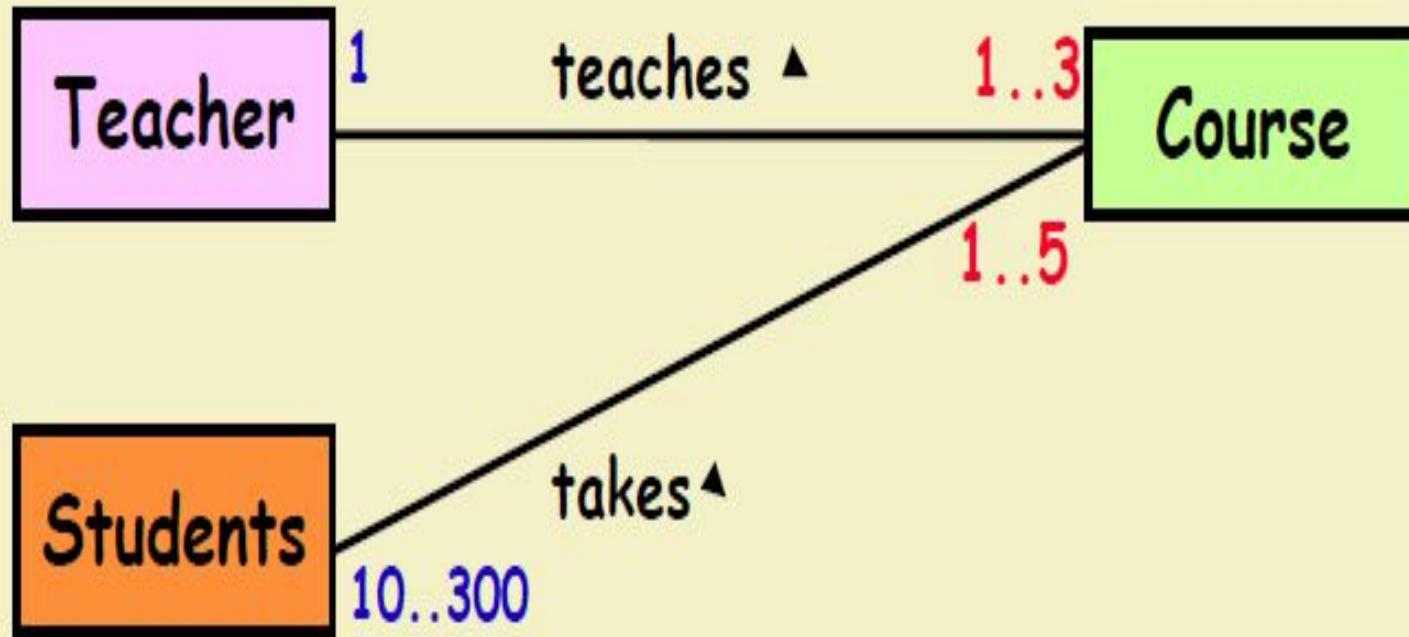
Association - More Examples



Multiplicity: The number of objects from one class that relate with a single object in an associated class.

Association - Multiplicity

- A teacher teaches 1 to 3 courses (subjects)
- Each course is taught by only one teacher.
- A student can take between 1 to 5 courses.
- A course can have 10 to 300 students.

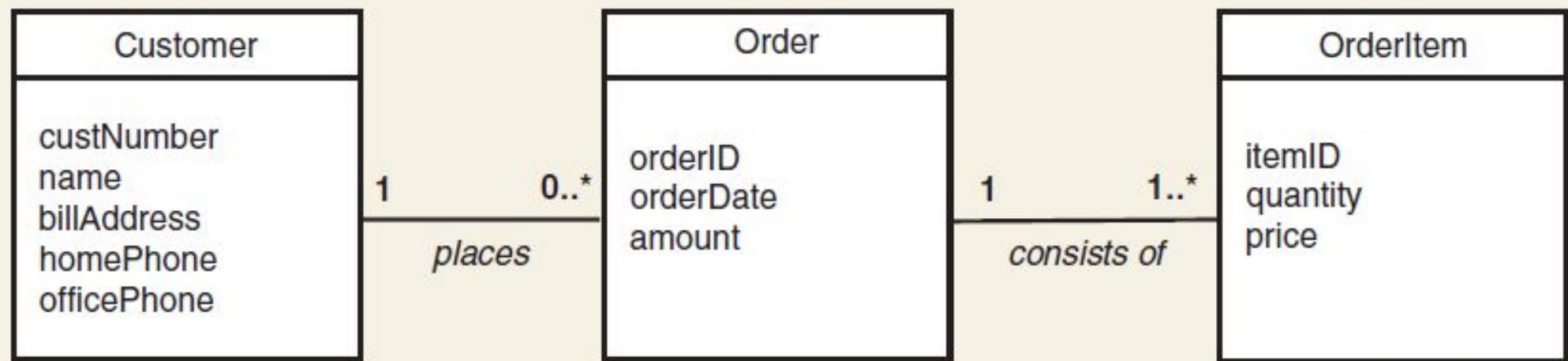
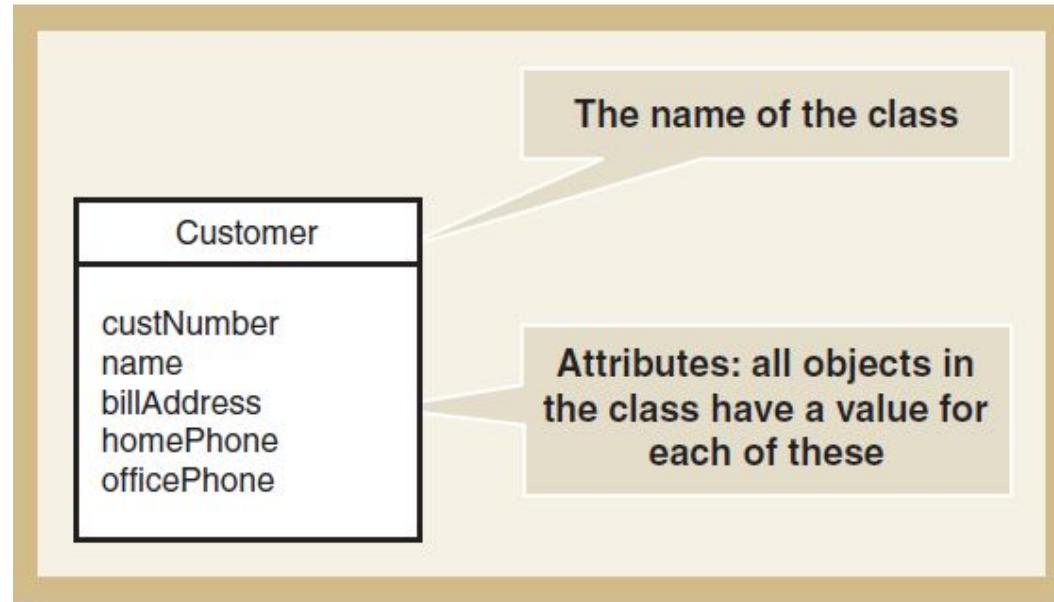


Quiz: Draw Class Diagram

- A Student can take up to five Courses.
- A student needs to enroll in at least one course.
- Up to 300 students can enroll in a course.
- An offered subject in a semester should have at least 10 registered students.



Domain model class diagram

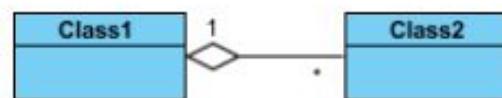


Multiplicity of association

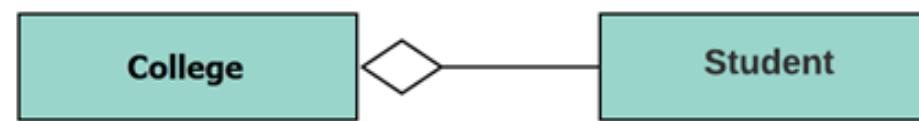
Indicator	Meaning
0..1	Zero or one
1	One only
0..*	0 or more
1..* *	1 or more
n	Only n (where n > 1)
0..n	Zero to n (where n > 1)
1..n	One to n (where n > 1)

Aggregation:

- A special type of association. It represents a "part of" relationship.
- Class2 is part of Class1.
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- Objects of Class1 and Class2 have separate lifetimes.
- A solid line with an unfilled diamond at the association end connected to the class of composite

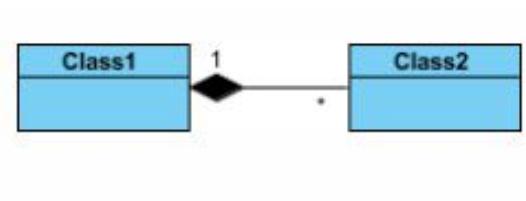


Aggregation



Composition:

- A special type of aggregation where parts are destroyed when the whole is destroyed.
- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.
- A solid line with a filled diamond at the association connected to the class of composite



Aggregation & Composition

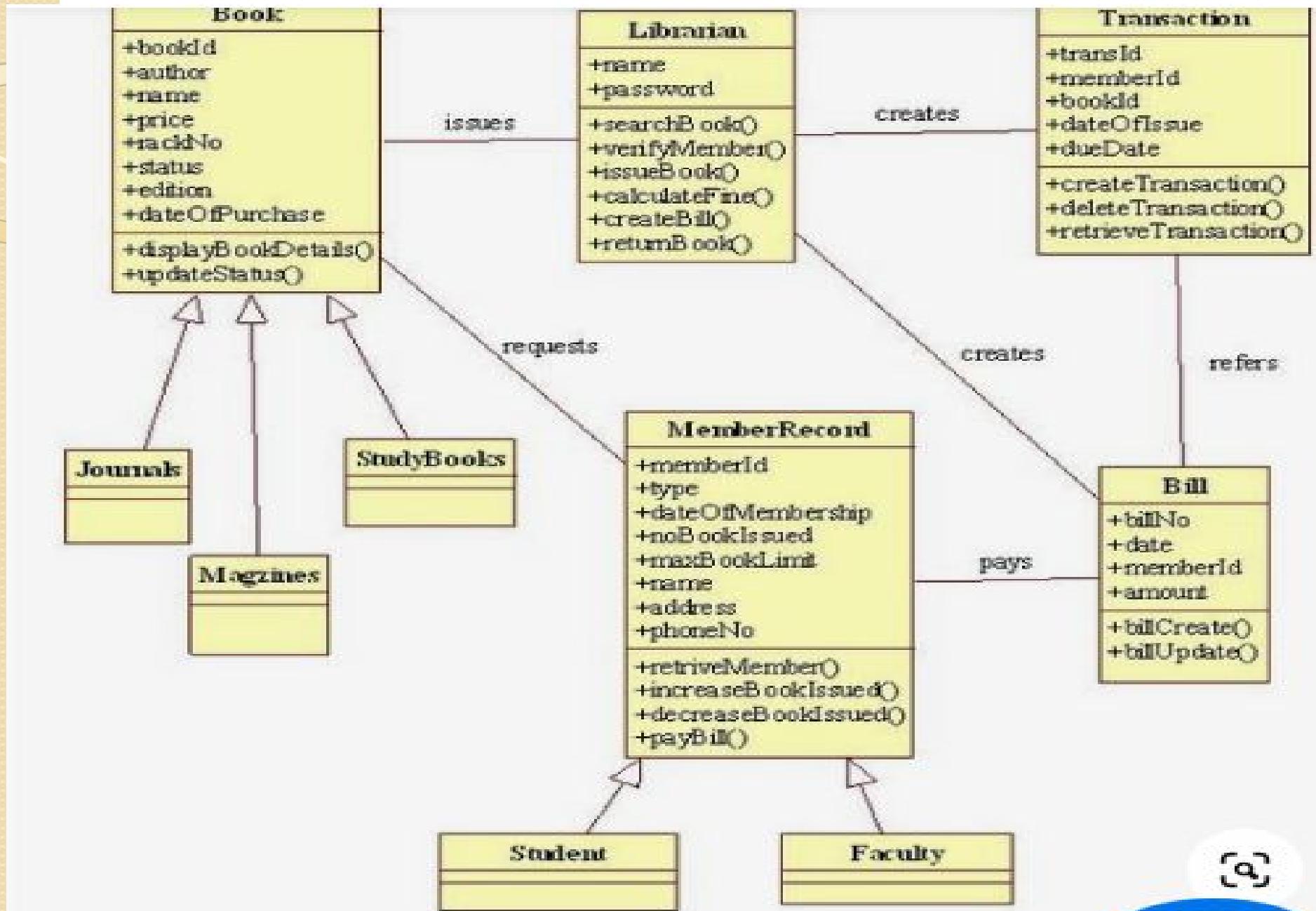
- **Aggregation** implies a relationship where the child can exist independently of the parent.
Example: **Class** (parent) and **Student** (child).
Delete the Class and the Students still exist.
- **Composition** implies a relationship where the child cannot exist independent of the parent.
Example: **House** (parent) and **Room** (child).
Rooms don't exist separate to a House.

Dependency:

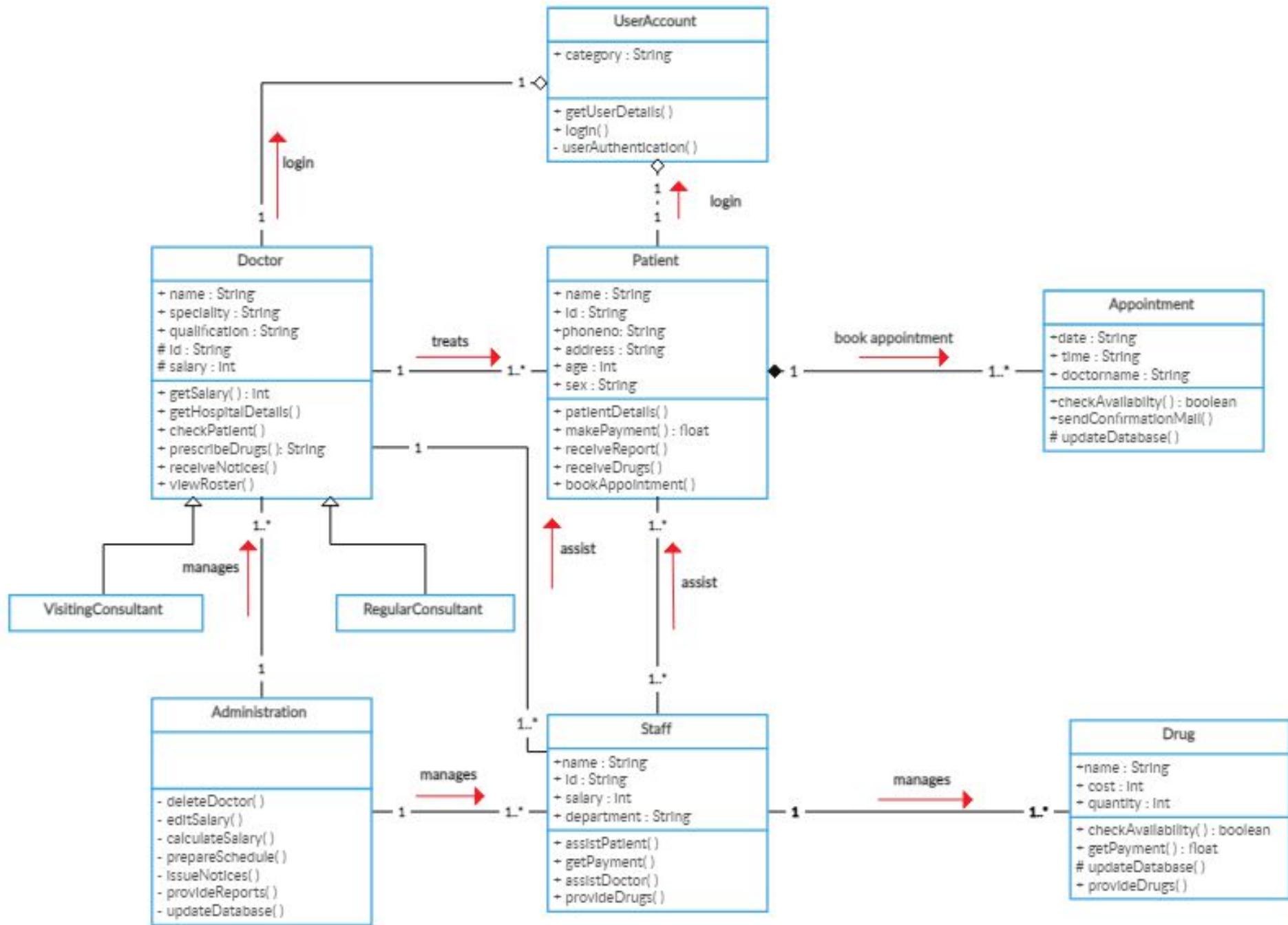
- Exists between two classes if the changes to the definition of one may cause changes to the other (but not the other way around).
- Class1 depends on Class2
- A dashed line with an open arrow



Class Diagram for Library Management System



Hospital Management System



Collaboration Diagram

Collaboration Diagram

- Collaboration diagrams are like sequence diagrams because of interaction and behavior factors.
- They are more concerned about object organization rather than sequence diagram that are more focused on a time sequence.
- They are also known as “Communication Diagram.”
- These are used to represent the flow of messages between the objects.

Collaboration Diagram

- To create a **collaboration diagram** from a **sequence diagram**: Right-click in the background of the **sequence diagram** and select Create Default **Collaboration Diagram** in the contextual menu.
- Select Tools→Create Default **Collaboration Diagram**.

Collaboration Diagram

- The collaboration diagram and sequence diagram shows similar information but in a distinct form.
- It can portray the architecture of an object inside the system.
- It can be used to depict the relationship among various objects within the system.
- The collaboration diagram shows the nature of a specific use case.
- They are used to determine the interfaces and class responsibilities.
- Objects collaborate through passing messages to each other.



Notations

- **Objects**
- **Links**
- **Messages**



Objects

- It is represented through an object symbol depicting the object's name and their class underlined, isolated by a colon.
- You can apply the objects within the collaboration diagram as follows:
- Every object in the collaboration has a name and has a specified class.

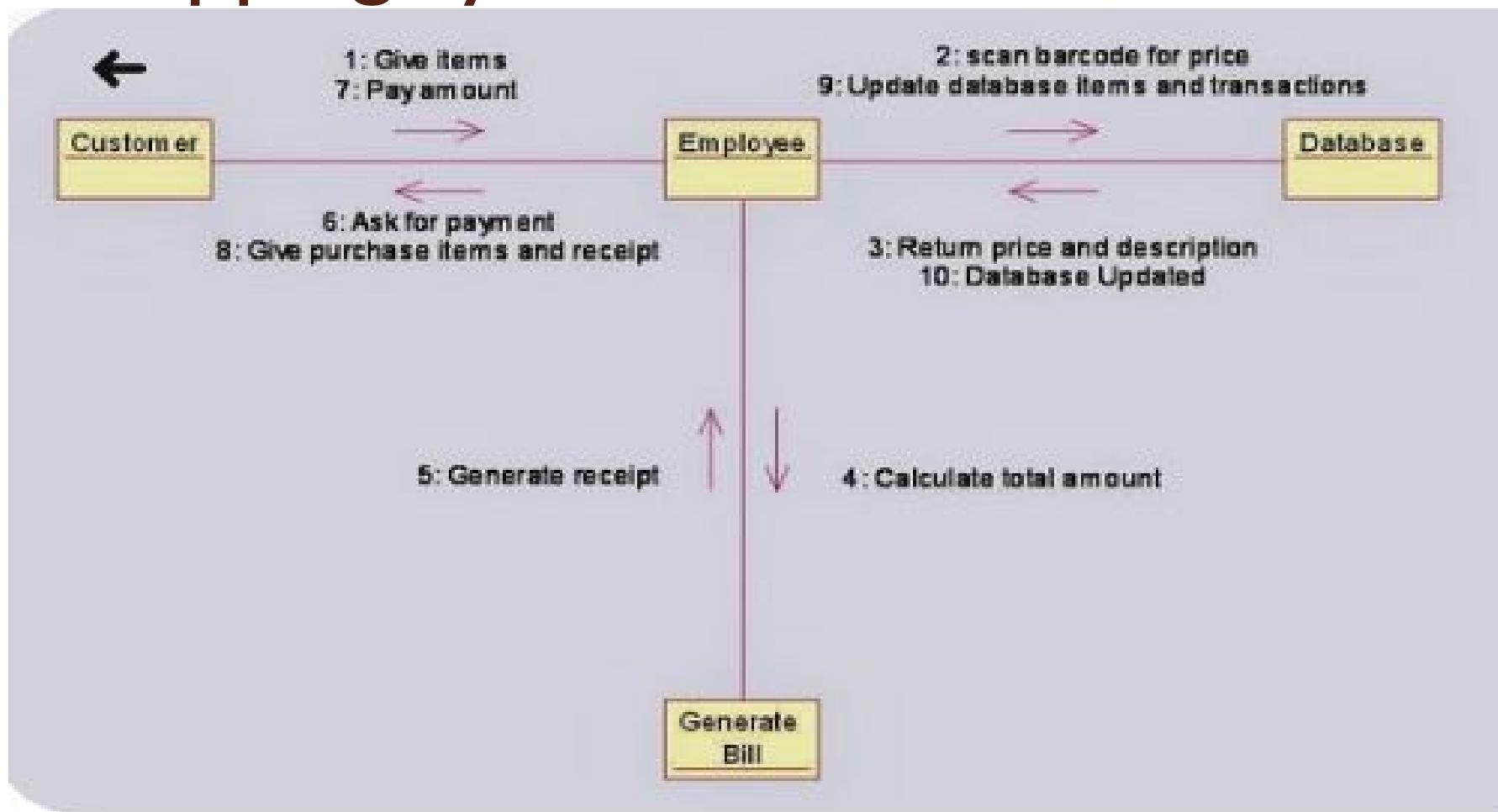
Links

- Links connect actors and objects. Every link correlates to the association in a class diagram.
- A link can be a relation between objects for which messages are transferred. In these diagrams, a link can be displayed just like a sturdy line among two objects.

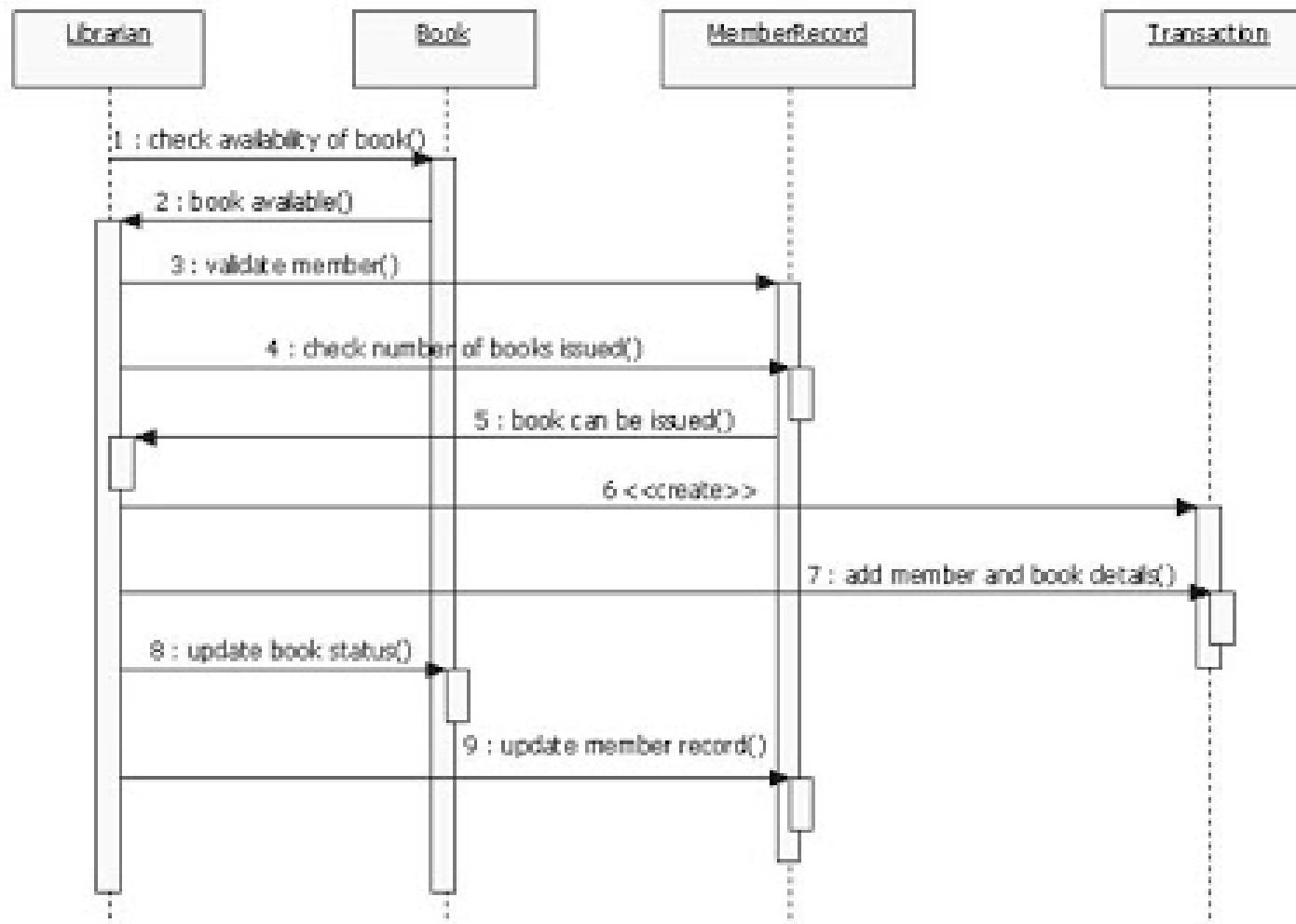
Messages

- It is communication among objects that transmits information with an expectation that action will arise.
- A message can be represented as a specified arrow positioned near the link in the collaboration diagram.

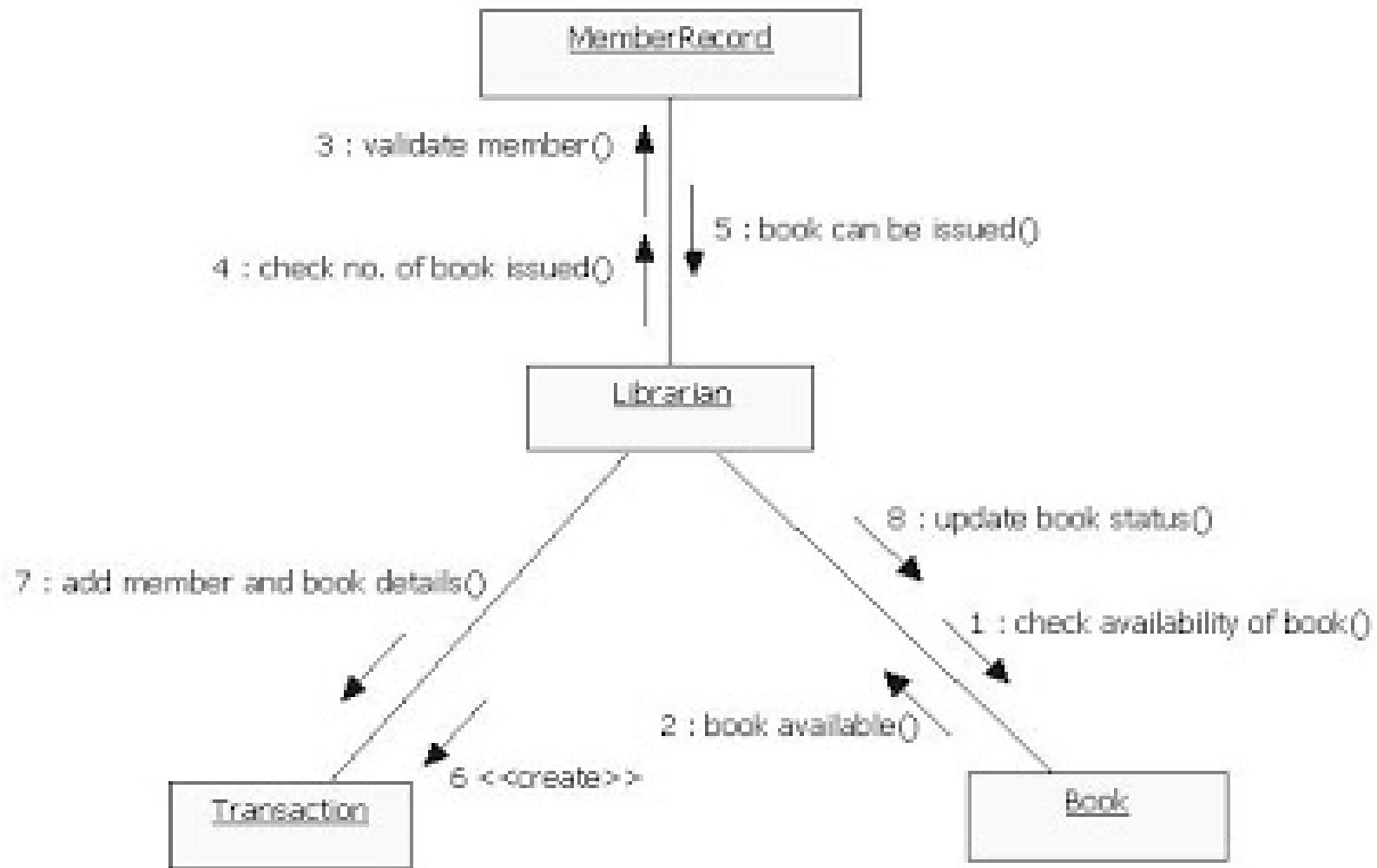
Collaboration diagram for Online Shopping System



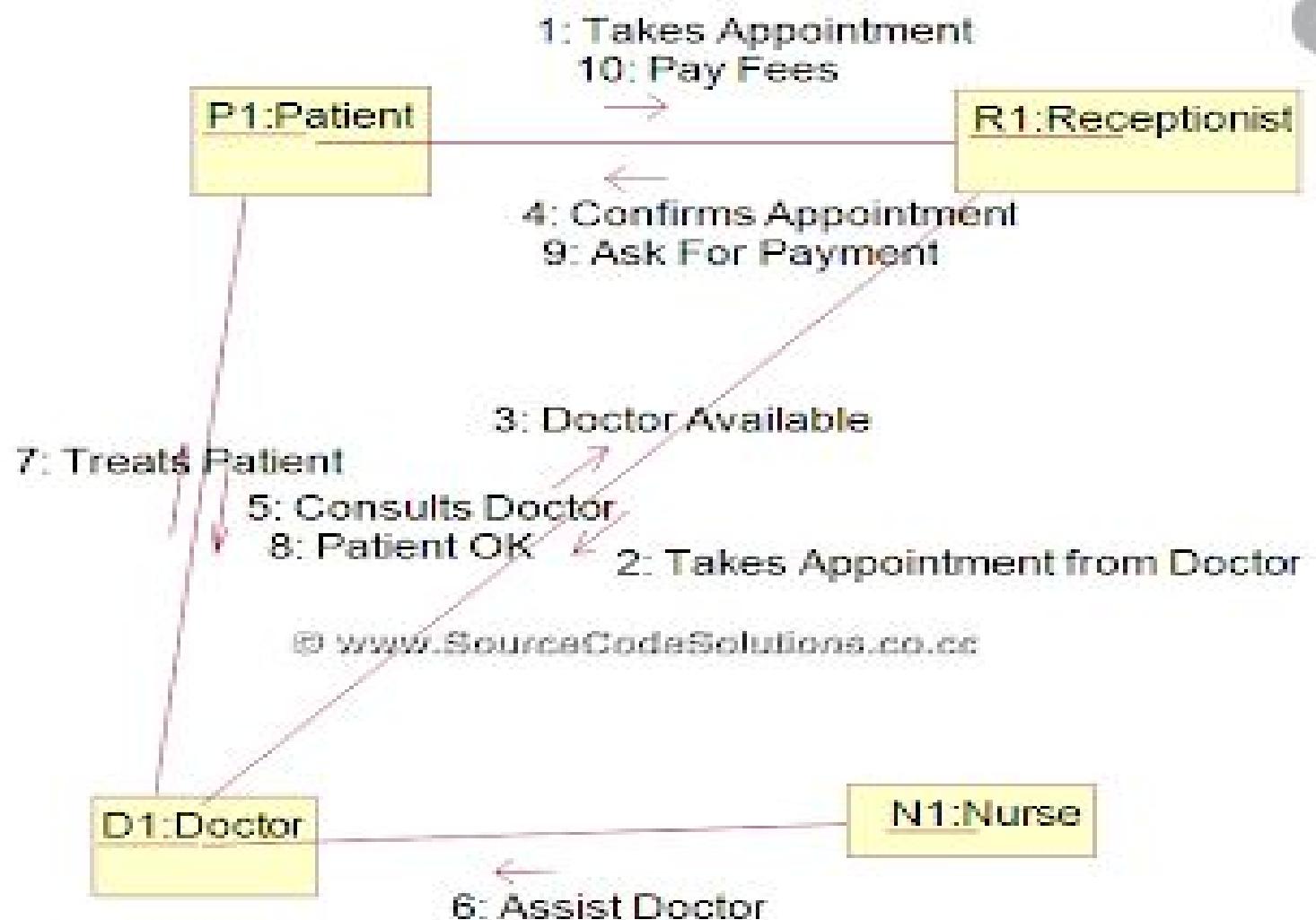
Sequence diagram for Issue Book



Collaboration diagram for Issue book



Collaboration diagram for Online Hospital Management System



Software Engineering(SE)

CSC 601



Subject Incharge

Varsha Nagpurkar
Assistant Professor
Room No. 407

email: varshanagpurkar@sfit.ac.in



Module-3

Project Scheduling and Tracking

5/25/2021

Module-3 Syllabus

3.0	Project Scheduling and Tracking	08
3.1	Management Spectrum, 3Ps (people, product and process)	
3.2	Process and Project metrics	
3.3	Software Project Estimation: LOC, FP, Empirical Estimation Models - COCOMO II Model, Specialized Estimation Techniques	
3.4	Project scheduling: Defining a Task Set for the Software Project, Timeline charts, Tracking the Schedule, Earned Value Analysis	

THE MANAGEMENT SPECTRUM

- Effective software project management focuses on the four P's: people, product, process, and project.
- The People
- Every organization needs to continually improve its ability to attract, develop, motivate, organize, and retain the workforce needed to accomplish its strategic business objectives”.
- The people management capability maturity(PM-CMM) model developed by Software Engineering Institute(SEI) defines the following key practice areas for software people:

THE MANAGEMENT SPECTRUM-The People

- Recruiting, selection, performance management, training, compensation, career development, organization and work design, and team/culture development
- Organizations that achieve high levels of maturity in the people management area have a higher likelihood of implementing effective software project management practices.
- The People-CMM is a companion to the Software Capability Maturity Model–Integration that guides organizations in the creation of a mature software process

THE MANAGEMENT SPECTRUM-The People-Stakeholders

- The software process(and every software project)is populated by stakeholders who can be categorized into one of five categories:
 - 1.Senior managers- who define the business issues that often have a significant influence on the project.
 2. Project (technical) managers- who must plan, motivate, organize, and control the practitioners who do software work

THE MANAGEMENT SPECTRUM-The People-Stakeholders

3. Practitioners- who deliver the technical skills that are necessary to engineer a product or application.
4. Customers-who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.
5. End users- who interact with the software once it is released for production use.

THE MANAGEMENT SPECTRUM-The People-Team Leaders

- In an excellent book of technical leadership, Jerry Weinberg suggests a MOI Model of leadership
- 1. Motivation- The ability to encourage (by “push or pull”) technical people to produce to their best ability.
- 2. Organization.- The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product

THE MANAGEMENT SPECTRUM-The People-Team Leaders

- 3. Ideas or innovation.The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

Characteristics that define effective project management

- Problem solving-Project manager can diagnose the technical and organizational issues that are most relevant, systematically structure a solution or properly motivate other practitioners to develop the solution
- Managerial identity- A good project manager must take charge of the project. She must have the confidence to assume control when necessary and the assurance to allow good technical people to follow their instincts.

Characteristics that define effective project management

- Achievement-To optimize the productivity of a project team,a manager must reward initiative and accomplishment
- Influence and team building-An effective project manager must be able to “read” people;she must be able to understand verbal and nonverbal signals and react to the needs of the people sending these signals

The Software Team

- The “best” team structure depends on the management style of the organization, the number of people who will populate the team and their skill levels, and the overall problem difficulty
- Marilyn Mantei from The University of Michigan suggested following 7 factors for considerations before structuring a team

The Software Team

- The difficulty of the problem to be solved
- The “size” of the resultant program(s) in lines of code or function points
- The time that the team will stay together(team lifetime)
- The degree to which the problem can be modularized
- The required quality and reliability of the system to be built

The Software Team

- The rigidity of the delivery date
- The degree of communication required for the project

The Software Team-

To achieve a high-performance team:

- Team members must have trust in one another
- The distribution of skills must be appropriate to the problem
- Independent-minded person may have to be excluded from the team, if team unity is to be maintained
 - Regardless of team organization, the objective for every project manager is to help create a team that exhibits unity among the team members

Agile teams

- In recent years, agile software development has been proposed
- The agile philosophy provides the following
 - It encourages customer satisfaction by early incremental delivery of software
 - It provides small highly motivated project teams
 - It consists of informal methods
 - It provides minimal software engineering work products
 - It results in overall development simplicity

Agile teams

- The small sized and highly motivated project team, also called an agile team, adopts many of the characteristics of successful software project teams.

Coordination and Communication Issues

- There are many reasons that software projects get into trouble
- The scale of many development efforts is large, leading to complexity, confusion, and significant difficulties in coordinating team members
- Uncertainty is common, resulting in a continuing stream of changes
- Interoperability-New software must communicate with existing software and conform to predefined constraints imposed by the system

Coordination and Communication Issues

- Characteristics of modern software-Scale,uncertainty, and interoperability-are facts of life
- To deal with them effectively,a Software Engineering team must establish effective methods for coordinating the people who do the work
- To accomplish this mechanisms for formal and informal communication among team members and between multiple teams must be established

Coordination and Communication Issues

- Formal communication is accomplished through “writing”, structured meetings, and other relatively noninteractive and impersonal communication channels
- Informal communication is more personal-Members of a team share ideas on an adhoc basis, ask for help as problem arise, and interact with one another on a daily basis

The Product

- As requirements may change regularly as the project proceeds,a plan is needed
- So the first activity in software project management is the determination of software scope
- Scope is defined by answering the following questions:
 - Context-How does the software to be built fit into a larger system,product,or business context, and what constraints are imposed as a result of the context?



The Product

- Information objectives-What customer-visible data objects are produced as output from the software? What data objects are required for input?
- Function and performance-What functions does the software perform to transform input data into output?

The Product

- Software project scope must be unambiguous and understandable at the management and technical levels
- Software scope must be bounded-
 - Quantitative data(e.g.,number of simultaneous users,size of mailing list,maximum allowable response time)are stated explicitly;constraints and/or limitations(e.g.,product cost restricts memory size)are noted



The Process

- The framework activities that characterize the software process are applicable to all software projects
- The project manager must decide which process model is most appropriate for
 - The customers who have requested the product and the people who will do the work
 - The characteristics of the product itself, and
 - The project environment in which the software team works

The Process

- When a process model has been selected, the team then defines a preliminary project plan based on the set of process framework activities.
- Once the preliminary plan is established, process decomposition begins
- That is, a complete plan, reflecting the work tasks required to populate the framework activities, must be created

The Project

- To manage a successful software project, we must understand what can go wrong so that problems can be avoided
- In an excellent paper on software projects, John Reel defines 10 signs that indicate that an information systems project is in risk
 - Software people don't understand their customer's needs.
 - The product scope is poorly defined
 - Changes are managed poorly
 - The chosen technology changes
 - Business needs change
 - Deadlines are unrealistic
 - Users are resistant
 - Sponsorship is lost
 - The project team lacks people with appropriate skills
 - Managers and practitioners avoid best practices and lessons learned

The Project

- How does a manager act to avoid the problems just noted?
- Reel suggests a five-part common-sense approach to software projects:
 - Start on the right foot
 - Maintain momentum
 - Track progress
 - Make smart decisions
 - Conduct a postmortem analysis

The Project-A five part common-sense approach to software projects

- Start on the right foot-Accomplished by working hard to understand the problem that is to be solved and then setting realistic objectives and expectations for everyone who will be involved in the project. It is accomplished by building the right team
- Maintain Momentum- Many projects get off to a good start and then slowly disintegrate. To maintain momentum , the project manager must provide incentives to keep turnover of personnel to an absolute minimum
- Track Progress-For a software project,progress is tracked as work products(e.g.,models,source code,sets of test cases)are produced and approved

The Project-A five part common-sense approach to software projects

- Make smart decisions-Whenever possible,decide to use commercial off-the shelf software or existing software components,decide to avoid custom interfaces when standard approaches are available,decide to identify and then avoid risks,and decide to allocate more time than you think is needed to complex and risky tasks
- Conduct a postmortem analysis-Establish a consistent mechanism for extracting lessons learned for each project.Evaluate the planned and actual schedules,collect and analyze software project metrics,get feedback from team members and customers



Metrics in the Process and Project Domains



Measurements and Metrics

- **Measurements and Metrics**
- A **measurement** is an indication of the size, quantity, amount or **dimension** of a particular attribute of a product or process.
- A **Metric** is a **measurement** of the degree that any attribute belongs to a system, product or process. For example the number of errors per person per hour would be a **metric**

What are Metrics

- Software process and project metrics are quantitative measures
- They are a management tool
- They offer insight into the effectiveness of the software process and the projects that are conducted using the process as a framework
- Basic quality and productivity data are collected
- These data are analyzed, compared against past averages, and assessed

What are Metrics

- The goal is to determine whether quality and productivity improvements have occurred
- The data can also be used to pinpoint problem areas
- Remedies can then be developed and the software process can be improved

Uses of Measurement

- Can be applied to the software process with the intent of improving it on a continuous basis
- Can be used throughout a software project to assist in estimation, quality control, productivity assessment, and project control
- Can be used to help assess the quality of software work products and to assist in tactical decision making as a project proceeds

Metrics in the Process Domain

- Process metrics are collected across all projects and over long periods of time
- They are used for making strategic decisions
- The intent is to provide a set of process indicators that lead to long-term software process improvement
- The only way to know how/where to improve any process is to
 - Measure specific attributes of the process
 - Develop a set of meaningful metrics based on these attributes
 - Use the metrics to provide indicators that will lead to a strategy for improvement

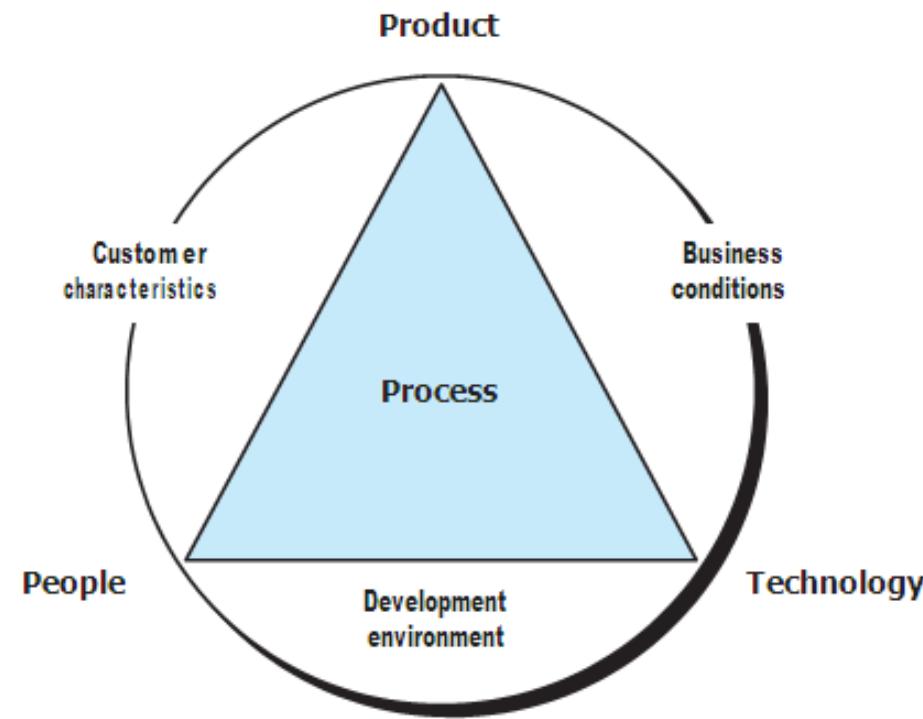
Metrics in the Process Domain

- We measure the effectiveness of a process by deriving a set of metrics based on outcomes of the process such as
 - Errors uncovered before release of the software
 - Defects delivered to and reported by the end users
 - Work products delivered
 - Human effort expended
 - Calendar time expended
 - Conformance to the schedule
 - Time and effort to complete each generic activity

Etiquette of Process Metrics

- Use common sense and organizational sensitivity when interpreting metrics data
- Provide regular feedback to the individuals and teams who collect measures and metrics
- Don't use metrics to evaluate individuals
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them
- Never use metrics to threaten individuals or teams
- Metrics data that indicate a problem should not be considered “negative”
 - Such data are merely an indicator for process improvement
- Don't obsess on a single metric to the exclusion of other important metrics

Determinants for software quality and organizational effectiveness





Determinants for software quality and organizational effectiveness

- Process sits at the center of a triangle connecting 3 factors that have a profound influence on software quality and organizational performance.
- The skill and motivation of people have been shown [Boe81] to be the most influential factors in quality and performance.
- The complexity of the product can have a substantial impact on quality and team performance.
- The technology (i.e., the software engineering methods and tools) that populates the process also has an impact.



Determinants for software quality and organizational effectiveness

- In addition, the process triangle exists within a circle of environmental conditions that include the development environment (e.g., integrated software tools), business conditions (e.g., deadlines, business rules), and customer characteristics (e.g., ease of communication and collaboration).



Metrics in the Project Domain

Metrics in the Project Domain

- Project metrics enable a software project manager to
 - Assess the status of an ongoing project
 - Track potential risks
 - Uncover problem areas before their status becomes critical
 - Adjust work flow or tasks
 - Evaluate the project team's ability to control quality of software work products
- Many of the same metrics are used in both the process and project domain
- Project metrics are used for making tactical decisions
 - They are used to adapt project workflow and technical activities

Use of Project Metrics

- The first application of project metrics occurs during estimation
 - Metrics from past projects are used as a basis for estimating time and effort
- As a project proceeds, the amount of time and effort expended are compared to original estimates
- As technical work commences, other project metrics become important
 - Production rates are measured (represented in terms of models created, review hours, function points, and delivered source lines of code)
 - Error uncovered during each generic framework activity (i.e, communication, planning, modeling, construction, deployment) are measured

Use of Project Metrics

- Project metrics are used to
 - Minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks
 - Assess product quality on an ongoing basis and, when necessary, to modify the technical approach to improve quality
- In summary
 - As quality improves, defects are minimized
 - As defects go down, the amount of rework required during the project is also reduced
 - As rework goes down, the overall project cost is reduced

University Questions

- Explain four P's of Software Engineering
- What is process and project metrics

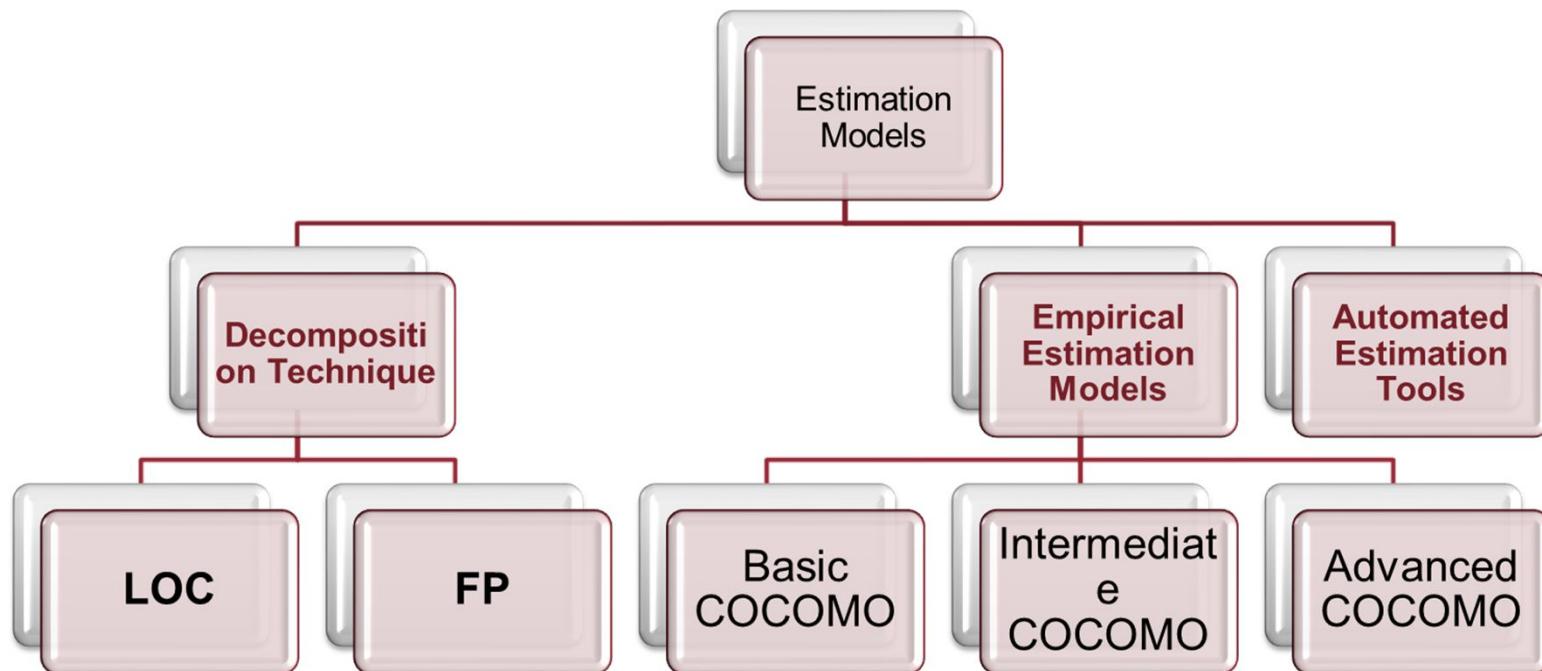
3.3 Software Project Estimation

- LOC
- FP

Software Measurement

- It can be categorized in two ways
 - Direct measures of the software process(e.g.,cost and effort applied) and product(e.g.,lines of code(LOC) produced,execution speed, and defects reported over some set period of time)
 - Indirect measures of the product that include functionality,quality,complexity,efficiency,reliability, maintainability etc.

Software Estimation Methods



Software Estimation:-

- **Decomposition Technique**
 - Lines of Code
 - Function Point
- **Empirical Estimation Models**
 - **Constructive Cost Model (COCOMO)**
 - Model 1 : Basic COCOMO
 - Model 2 : Intermediate COCOMO
 - Model 3 : Advanced COCOMO
- **Automated Estimation Tools**

- The five essential, fundamental metrics:
 - Size (LOC, etc.)
 - Cost (in dollars)
 - Duration (in months)
 - Effort (in person-month)
 - Quality (number of faults detected)

Steps in Estimating

- Estimate cost and effort
 - Decompose the problem
 - Develop two or more estimates using size, function points, process tasks or use-cases
 - Reconcile the estimates

Estimation

- Software cost estimation is the process of predicting the effort required to develop a software system.
- Estimation of resources, cost, and schedule for a software engineering effort requires
 - experience
 - access to good historical information
- Estimation carries inherent risk and this risk leads to uncertainty

Project Estimation



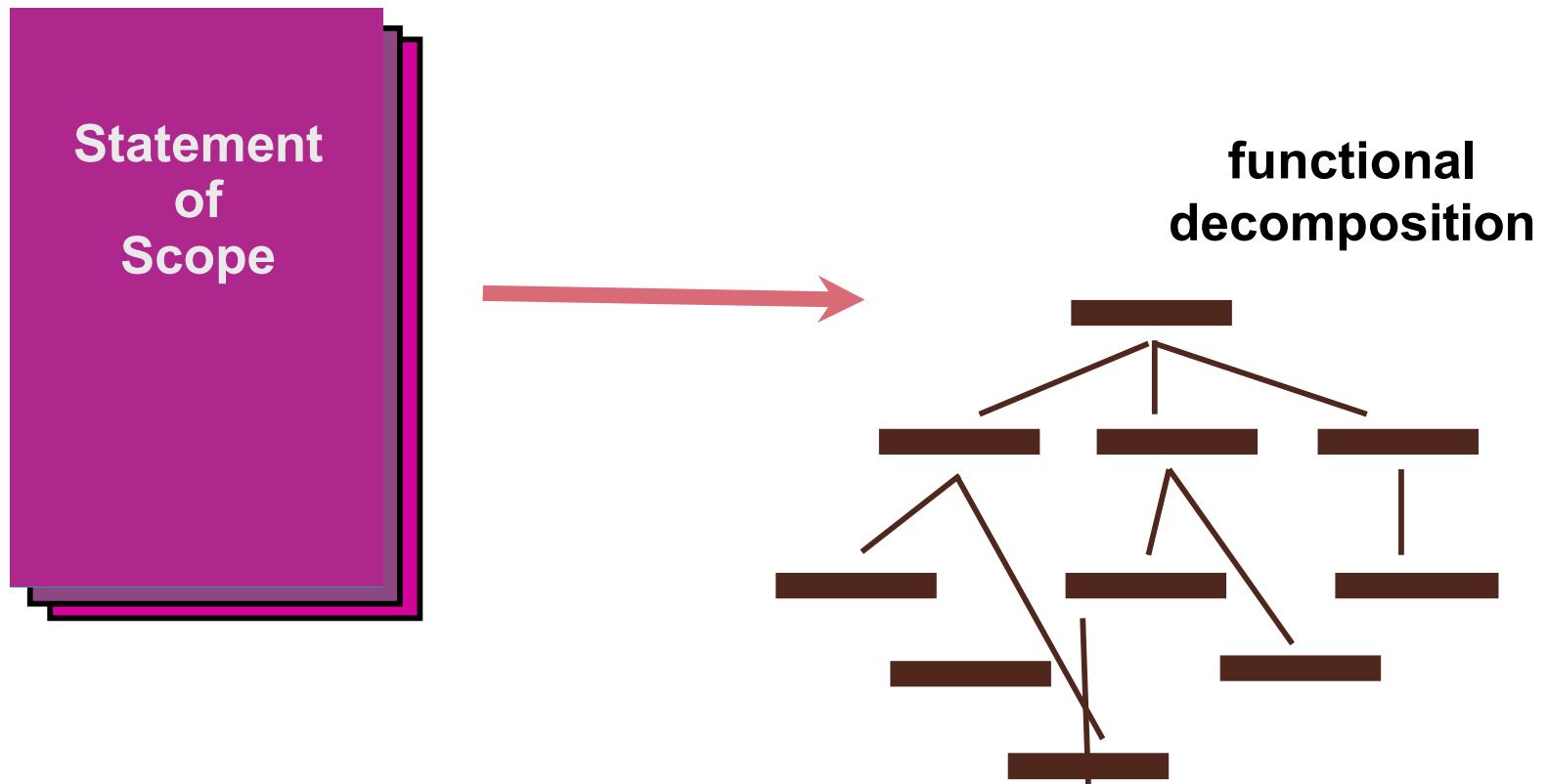
- Project scope must be understood
- Elaboration (decomposition) is necessary
- Historical metrics are very helpful
- At least two different techniques should be used
- Uncertainty is inherent in the process

Estimation Techniques

- Past (similar) project experience
- Conventional estimation techniques
 - task breakdown and effort estimates
 - size (e.g., FP) estimates
- Empirical models
- Automated tools



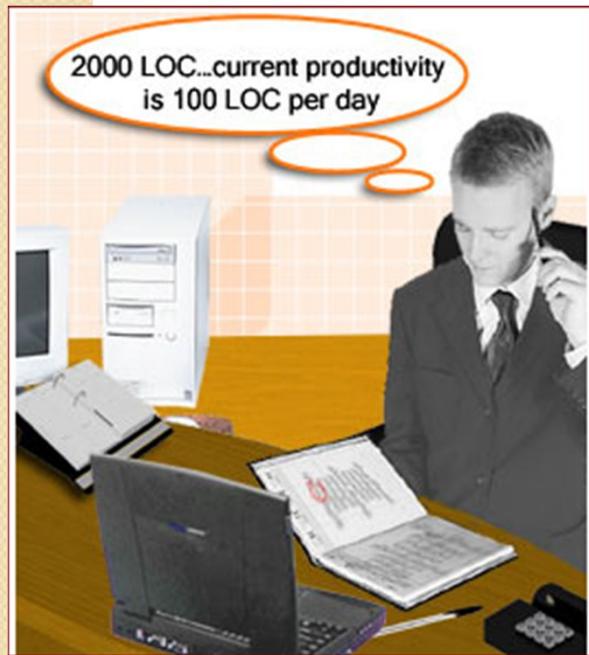
Functional Decomposition



Conventional Methods: LOC/FP Approach

- compute LOC (lines of code)/FP (function points) using estimates of information domain values
- use historical data to build estimates for the project

Example: LOC



Ap

Function	Estimated LOC
user interface and control facilities (UICF)	2,300
two-dimensional geometric analysis (2D GA)	8,300
three-dimensional geometric analysis (3D GA)	6,800
database management (DBM)	3,300
computer graphics display facilities (CGDF)	4,900
peripheral control (PC)	2,100
design analysis modules (DAM)	8,400
<i>estimated lines of code</i>	33,200

Average productivity for systems of this type = 620 LOC/pm.

Burdened labor rate = \$8000 per month, the cost per line of code is approximately \$13.

Based on the LOC estimate and the historical productivity data, the total estimated project cost is \$431,600

LOC Based Estimation

- You are required to establish estimates for Virtual Classroom System. For first increment you have to provide automation of course registration, Attendance, lectures.
- LOC estimates for each Functionality are provided:
 - User screens 3000
 - Course registration 1550
 - Attendance 1800
 - Lectures 3000
 - Student query 800

Suppose productivity of your Organization is 500 LOC/pm
Labor rate is \$8000/pm

Calculate Effort and Total cost of this increment

Total LOC

- User screens 3000
- Course registration 1550
- Attendance 1800
- Lectures 3000
- Student query 800
= Total LOC=10150 LOC

- LOC/ Productivity
- = $10150/500$
- = $20.3 \text{ pm or } 21 \text{ pm}$



$$\begin{aligned}\text{Total cost} &= \text{Rate} * \text{pm} \\ &= 8000 * 21 \\ &= 168000\text{Rs.}\end{aligned}$$

$$\begin{aligned}\text{Cost per LOC} &= 168000/10150 \\ &= 16.55 \text{ Rs.}\end{aligned}$$

Total Cost

- Total cost=Rate * pm
= 8000 * 21
= 168000Rs.
- Cost per LOC
 $=168000/10150$
=16.55 Rs.



Function Points

- The function point metric(FP),first proposed by Albrecht,can be used effectively as a means for measuring the functionality delivered by the system
- Using historical data,the FP can then be used to
 - i)estimate the cost or effort required to design,code, and test the software;
 - ii)predict the number of errors that will be encountered during testing,
 - and iii)forecast the number of components and/or the number of projected source lines in the implemented system

Function Points

- Function points are derived using an empirical relationship based on countable(direct)measures of software's information domain and assessment of software complexity.
- Information domain values are defined in the following manner

Function Points

- Number of external inputs(EIs)-Each external input originates from a user or is transmitted from another application and provides distinct application-oriented data or control information. Inputs are often used to update internal logical files
- Number of external outputs(EOs)-Each external output is derived within the application and provides information to the user.External output refers to reports,screens,error messages,and so on

Function Points

- Number of external inquiries(EQs)-An external inquiry is defined as an online input that results in generation of some immediate software response in the form of an on-line output
- Number of internal logical files(ILFs)-Each internal logical file is a logical grouping of data that resides within the application's boundary and is maintained via external inputs

Function Points

- Number of external interface files(EIFs)-
Each external interface file is a logical grouping of data that resides external to the application but provides data that may be of use to the application

Function Points

- A function point is a unit of measurement to express the amount of business functionality an information system (as a product) provides to a user. The cost (in dollars or hours) of a single unit is calculated from past projects

Important Factors to be consider for FP are...

Number of external inputs – from user or anc

Number of external inquiries – request from
in on-line output (E.G. IRCTC)

Number of external outputs

Number of internal logical files (maintained by system)

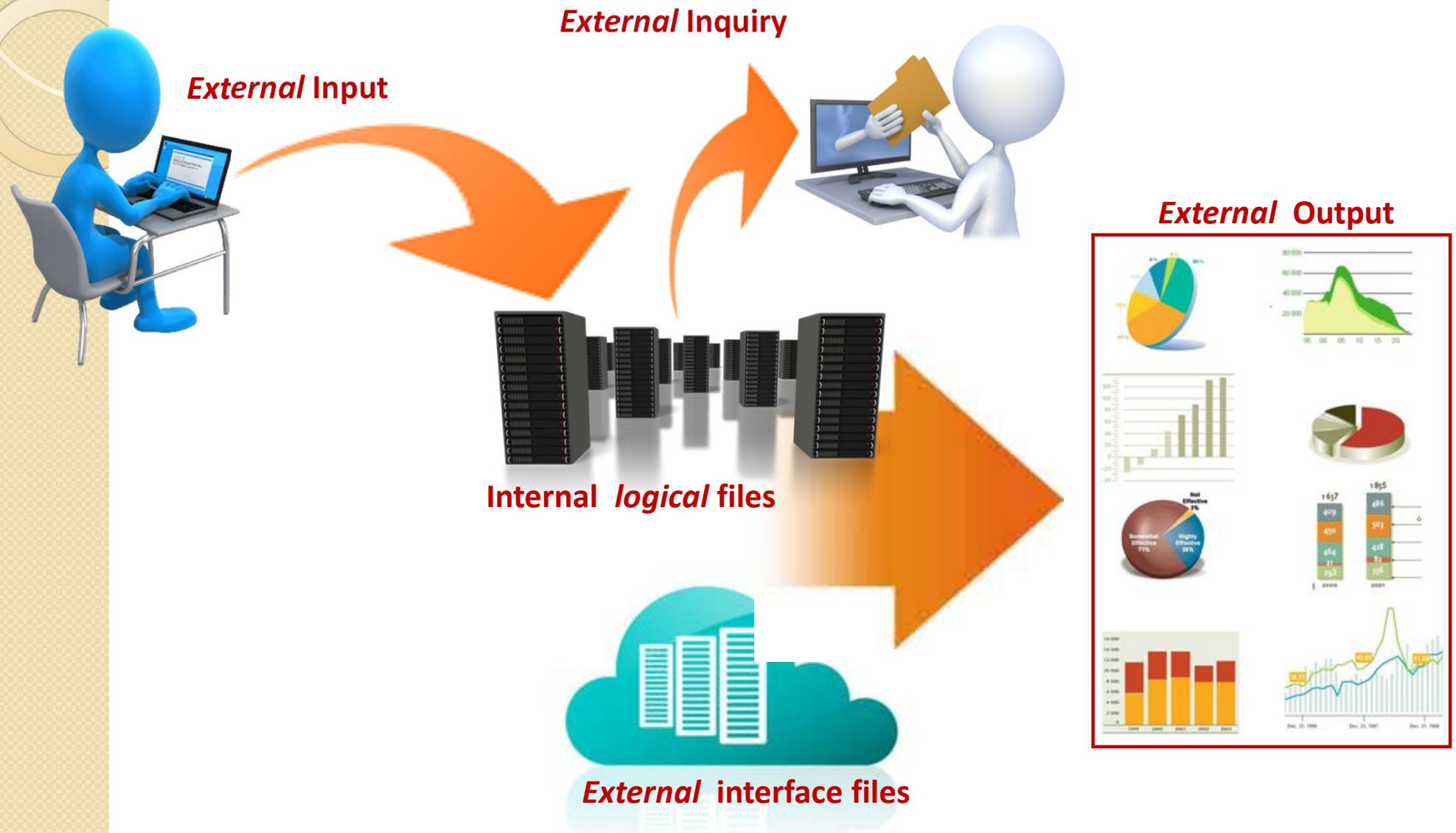
Number of external interface files (provides data but not maintained by system, The **external interface file** is an internal logical file for another application. An application may count a **file** as either a EIF or ILF not both.) (Library system may use students DB for calculating fine)



Size estimation

Funct

67



Count-Total

- Count-Total = sum(number X weight)
- Where weights are:

	Simple	Average	Complex
External Inp	3	4	6
External Out	4	5	7
External Inq	3	4	6
Internal Files	7	10	15
External Files	5	7	10

Value adjustment factors(F_i)

- **Total of 1-5 rating of following 14 questions:**
 - Does the system require reliable back-up/recovery?
 - Are specialized data communications required?
 - Are there distributed processing functions?
 - Is performance critical?
 - Will run in heavily utilized operating environment?
 - On-line data entry required?
 - For on-line data entry, will it require multiple screens?
 - Are ILF's updated on-line?
 - Are input, output, files, or inquiries complex?
 - Is the internal processing complex?
 - Is the code designed to be reusable?
 - Are conversion and installation included?
 - Is the system designed for installation in different organizations?
 - Is the application designed to facilitate change and ease of use?

General System Characteristic	Brief Description
1. Data communications	How many communication facilities are there to aid in the transfer or exchange of information with the application or system?
2. Distributed data processing	How are distributed data and processing functions handled?
3. Performance	Was response time or throughput required by the user?
4. Heavily used configuration	How heavily used is the current hardware platform where the application will be executed?
5. Transaction rate	How frequently are transactions executed daily, weekly, monthly, etc.?
6. On-Line data entry	What percentage of the information is entered On-Line?
7. End-user efficiency	Was the application designed for end-user efficiency?
8. On-Line update	How many ILF's are updated by On-Line transaction?
9. Complex processing	Does the application have extensive logical or mathematical processing?
10. Reusability	Was the application developed to meet one or many user's needs?
11. Installation ease	How difficult is conversion and installation?
12. Operational ease	How effective and/or automated are start-up, back-up, and recovery procedures?
13. Multiple sites	Was the application specifically designed, developed, and supported to be installed at multiple sites for multiple organizations?
14. Facilitate change	Was the application specifically designed, developed, and supported to facilitate change?

Examlpe



FP based Estimation

- For above mentioned system
 - Student can be registered
 - Courses can be added/updated.
 - Student can view courses
 - Student can inquire his/her registration status

	Count	Simple	Average	Complex
External Inp	3*	3	4	6 = 9
External Out	4*	4	5	7 = 16
External Inq	1*	3	4	6 = 3
Internal Files	4*	7	10	15 = 28
External Files	0*	5	7	10 = 0

Count Total =56

Calculation of sum(F_i)

- Total of 0-5 rating of following 14 questions:
 - Does the system require reliable back-up/recovery? 3
 - Are specialized data communications required? 2
 - Are there distributed processing functions? 0
 - Is performance critical? 2
 - Will run in heavily utilized operating environment? 1
 - On-line data entry required? 4
 - For on-line data entry, will it require multiple screens? 5
 - Are ILF's updated on-line? 5
 - Are input, output, files, or inquiries complex? 2
 - Is the internal processing complex? 3
 - Is the code designed to be reusable? 0
 - Are conversion and installation included? 1
 - Is the system designed for installation in different organizations? 0
 - Is the application designed to facilitate change and ease of use? 1

- The estimated number of FP is derived:
$$FP_{estimated} = count-total \times [0.65 + 0.01 \times \sum (F_i)]$$
$$FP_{estimated} = 56 * (0.65 + 0.01 * 29)$$
$$= 52.64 \text{ or } 53$$
- If organizational average productivity = 5 FP/pm.
- labor rate = \$8000 per month,
- Cost = labor rate * Person Month
- Based on the FP estimate and the historical productivity data,
- **the total estimated project cost is 84800Rs. and the estimated effort is 10.6 person-months.**

Function Point (FP) is an element of software development which helps to approximate the cost of development early in the process. It may measures functionality from user's point of view.

Counting Function Point (FP):

- Step-1:

$$F = 14 * \text{scale}$$

Scale varies from 0 to 5 according to character of Complexity Adjustment Factor (CAF). Below table shows scale:

- 0 - No Influence
- 1 - Incidental
- 2 - Moderate
- 3 - Average
- 4 - Significant
- 5 - Essential

- **Step-2: Calculate Complexity Adjustment Factor (CAF).**

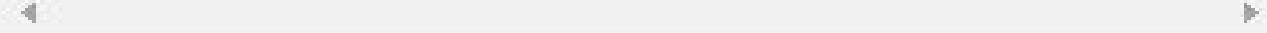
$$CAF = 0.65 + (0.01 * F)$$

- - -

- - -

Step-3: Calculate Unadjusted Function Point (UFP).
TABLE (Required)

Function Units	Low	Avg	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10



Multiply each individual function point to corresponding values in TABLE.



- **Step-4: Calculate Function Point.**

$$FP = UFP * CAF$$

Example:

Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average.

User Input = 50

User Output = 40

User Inquiries = 35

User Files = 6

External Interface = 4

- **Step-1:** As complexity adjustment factor is average (given in question), hence,

scale = 3.

$$F = 14 * 3 = 42$$

- **Step-2:**

$$CAF = 0.65 + (0.01 * 42) = 1.07$$

Step-3: As weighting factors are also average (given in question) hence we will multiply each individual function point to corresponding values in TABLE.

$$P = (50*4) + (40*5) + (35*4) + (6*10) + (4*7) = 628$$



► Step-4:

$$\text{Function Point} = 628 * 1.07 = 671.96$$

Cost Estimation

- Assume organizational average productivity = 25 FP/pm.
- labor rate = \$8000 per month,
- No of months required=FP/average productivity
 - $=672/25=26.88=27\text{months}$
- Estimated cost for project=\$8000*27=\$2,16,000
- Based on the FP estimate and the historical productivity data,
- **the total estimated project cost is \$2,16,000 and the estimated effort is 27 months.**

University Questions

- What is cost estimation? Explain LOC method.(5 marks)
- Differentiate between FP based and LOC based cost estimation techniques.(10 marks)

Project Estimation Techniques

- Estimation of various project parameters is an important project planning activity. The different parameters of a project that need to be estimated include-
 - Project Size
 - Effort required to complete the project,
 - Project duration and
 - Cost
- Accurate estimation of these parameters is important for resource planning and scheduling
- Estimation Techniques can be classified as
 - Empirical Estimation Techniques
 - Heuristic Estimation Techniques
 - Analytical Estimation Techniques



Empirical Estimation Techniques

- Empirical Estimation Techniques are based on making an educated guess of the project parameters and common sense
- This technique is based on prior experience of development of similar products and projects
- An educated guess based on past experience
- Two popular empirical estimation techniques are
 - Expert Judgement Technique
 - Delphi Cost Estimation

Expert Judgement Technique

- In this an expert makes an educated guess of the problem size after analyzing the problem thoroughly
- The expert estimates the cost of the different components of the system:e.g GUI,database module,communication module,billing module etc
- Combines them to arrive at the overall estimate

Delphi Cost Estimation

- Is carried out by a team comprising of a group of experts and a coordinator
- The coordinator provides each estimator with a copy of the SRS document and a form for recording his cost estimate
- Estimators complete their individual estimates anonymously and submit to the coordinator



Heuristic Techniques

- In Heuristic techniques the relationship that exists among the different project parameters can modeled using suitable mathematical expressions
- Once the independent parameters are known, the dependent parameters can be easily determined by substituting the values of the independent parameters in the corresponding mathematical expressions

Heuristic Techniques

- Estimated parameter is the dependent parameter to be estimated. The dependent parameters to be estimated could be effort, duration, staff size etc.
- Assume that the characteristic to be estimated can be expressed in terms of some mathematical expression
 - Can be classified as Single variable and multivariable models

COCOMO Model-Constructive Cost Model

- Was first proposed by Dr. Barry Boehm in 1981
- Is a heuristic estimation technique-this technique assumes that relationship among different parameters can be modeled using some mathematical expression
- This approach implies that size is primary factor for cost, other factors have lesser effects
- Constructive implies that the complexity
- COCOMO prescribes a three stage process for project estimation

COCOMO Model-Constructive Cost Model

- An initial estimate is obtained, and over next two stages the initial estimate is refined to arrive at a more accurate estimate
- Projects used in this model have following attributes
 - Ranging in size from 2000 to 10000 lines of code
 - Programming languages ranging from assembly to PL/I
 - These projects were based on the waterfall model of software development

COCOMO Model-Constructive Cost Model

- The Constructive Cost Model (COCOMO) is a procedural software cost estimation model developed by Barry W. Boehm.
- The model parameters are derived from fitting a regression formula using data from historical projects

COCOMO Model-Constructive Cost Model

- Boehm stated that any software development project can be classified into three categories
 - Organic
 - Semi-detached
 - Embedded

COCOMO Model-Constructive Cost Model

- **Organic-**
 - If the project deals with developing a well understood application program
 - The size of development is reasonably small and experienced
 - The team members are experienced in developing similar kind of projects
 - Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.

COCOMO Model-Constructive Cost Model

- Semidetached-
 - If the development team consists of a combination of both experienced and inexperienced staff
 - Team members have limited experience about some aspects but are totally unfamiliar with some aspects of the system being developed
 - Mixed experience
 - Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.

COCOMO Model-Constructive Cost Model

- **Embedded-**
 - If the software being developed is strongly coupled to complex hardware
 - Software projects that must be developed within a set of tight software,hardware and operational constraints
 - For Example: ATM,Air Traffic control.

Mode	Project size	Nature of Project	Innovation	Deadline of the project	Development Environment
Organic 	Typically 2-50 KLOC 	Small size project, experienced developers in the familiar environment. For example, pay roll, inventory projects etc.	Little	Not tight	Familiar & In house
Semi detached 	Typically 50-300 KLOC	Medium size project, Medium size team, Average previous experience on similar project. For example: Utility systems like compilers, database systems, editors etc.	Medium	Medium	Medium
Embedded 	Typically over 300 KLOC	Large project, Real time Systems, Complex interfaces. Very little previous experience For example: ATMs, Air Traffic Control etc.	Significant	Tight	Complex Hardware/ customer Interfaces required

PERSON MONTH(PM)

- The effort estimation is expressed in units of person-months (PM).
- An effort of 100 PM does not imply that 100 persons should work for 1 month nor does it imply that 1 person should be employed for 100 months, but it denotes the area under the person-month curve .
- It is the area under the person-month plot.

Person month is a measurement unit for effort in software engineering. 1 person month means effort put by a person in one month. But 100 person does not mean, work effort put by 100 person in one month or 1 person in 100 months. As requirement of staff varies time to time in the development so there is not constant no of people is there to work. It is calculated from the graph(calculate area under time axis) between no of people working and time(in month).

COCOMO Model-Cost Constructive Model

- According to Boehm, software cost estimation should be done through three stages:
- Basic COCOMO Model
- Intermediate COCOMO Model
- Detailed COCOMO Model

1. BASIC COCOMO MODEL

- Basic COCOMO model computes software development effort, time and cost as a function of program size. Program size is expressed in estimated thousands of source lines of code (SLOC, KLOC).

$$\begin{aligned} \text{EFFORT} &= a_1 \times (\text{KLOC})^{a_2} \text{ PM} \\ T_{dev} &= b_1 \times (\text{Effort})^{b_2} \text{ Months} \end{aligned}$$

Where

- KLOC is the estimated number of delivered lines (expressed in thousands) of code for project, estimated size of software product.
- The coefficients a_1 , a_2 , b_1 , and b_2 are constants for each category of software products.
- T_{dev} is the estimated time to develop the software, in months.
- Effort is the total efforts required to develop the software product, expressed in Person Months (PM)

ESTIMATION OF DEVELOPMENT EFFORT

Organic	: Effort = $2.4(KLOC)^{1.05}$	PM
Semi-detached	: Effort = $3.0(KLOC)^{1.12}$	PM
Embedded	: Effort = $3.6(KLOC)^{1.20}$	PM

ESTIMATION OF DEVELOPMENT TIME

Organic	: $T_{dev} = 2.5(Effort)^{0.28}$	Months
Semi-detached	: $T_{dev} = 2.5(Effort)^{0.33}$	Months
Embedded	: $T_{dev} = 2.5(Effort)^{0.32}$	Months

CALCULATING EFFORT AND PRODUCTIVITY

When effort and development time are known, the average staff size to complete the project may be calculated as:

$$\text{Average staff size (SS)} = \frac{E}{D} \text{ Persons}$$

When project size is known, the productivity level may be calculated as:

$$\text{Productivity (P)} = \frac{KLOC}{E} \text{ KLOC / PM}$$

Software Project	a_1	a_2	b_1	b_2
Organic	2.4	1.05	2.5	0.38
Semi - Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

- Example: consider a software project using semi-detached mode with 30,000 lines of code . We will obtain estimation for this project as follows:

(1) Effort estimation $E = a_1(\text{KLOC}) \exp(a_2)$ person-months

$E = 3.0(30)\exp(1.12)$ where lines of code = 30000 = 30 KLOC

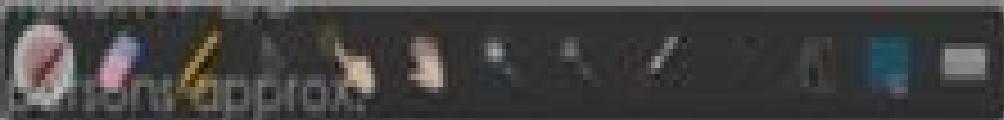
$E = 135$ person-month

(2) Duration estimation $D = b_1(E) \exp(b_2)$ months = $2.5(135)\exp(0.35)$

$D = 14$ months

(3) Person estimation $N = E/D$

$= 135/14 N = 10$



CALCULATING EFFORT AND PRODUCTIVITY

When effort and development time are known, the average staff size to complete the project may be calculated as:

$$\text{Average staff size (SS)} = \frac{E}{D} \text{ Persons}$$

When project size is known, the productivity level may be calculated as:

$$\text{Productivity (P)} = \frac{KLOC}{E} \text{ KLOC / PM}$$

$$\frac{30000}{135}$$

Merits-

- Basic COCOMO is good for quick , rough and early estimate of software costs.

Demerits-

- It does not account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and so on.
- The accuracy of this model is limited because it does not consider certain factors for cost estimation of software.

Example on Basic COCOMO Mode

- **Example1:** Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.
- $\text{Effort} = a_1 * (\text{KLOC})^{\exp a_2}$ PM
- $\text{Time} = b_1 * (\text{effort})^{\exp b_2}$ months

Software Project	a_1	a_2	b_1	b_2
Organic	2.4	1.05	2.5	0.38
Semi-Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

- **Solution:** The basic COCOMO equation takes the form:
 $\text{Effort} = a_1 * (\text{KLOC})^{a_2} \text{ PM}$
 $T_{\text{dev}} = b_1 * (\text{efforts})^{b_2} \text{ Months}$
Estimated Size of project= 400 KLOC
- **(i)Organic Mode**
- $E = 2.4 * (400)^{1.05} = 1295.31 \text{ PM}$
 $D = 2.5 * (1295.31)^{0.38} = 38.07 \text{ Months}$

- **(ii) Semidetached Mode**
- $E = 3.0 * (400) 1.12 = 2462.79 \text{ PM}$
 $D = 2.5 * (2462.79) 0.35 = 38.45 \text{ M}$
- **(iii) Embedded Mode**
- $E = 3.6 * (400) 1.20 = 4772.81 \text{ PM}$
 $D = 2.5 * (4772.8) 0.32 = 38 \text{ M}$

Software Projects	a ₁	a ₂	b ₁	b ₂
Organic	2.4	1.05	2.5	0.38
Semi-Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

- **Example2:** A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.
- Semidetached
- $\text{Effort} = a_1 * (\text{KLOC})^{\exp a_2}$
- $T = b_1 * (\text{effort})^{\exp b_2}$ months
- Avg staff size = E/T
- Productivity = KLOC/Effort

- **Solution:** The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.
 - Hence $E = 3.0(200) \cdot 1.12 = 1133.12 \text{ PM}$
 $D = 2.5(1133.12)0.35 = 29.3 \text{ PM}$


$$\text{Average Staff Size (SS)} = \frac{E}{D} \text{ Persons}$$

$$= \frac{1133.12}{29.3} = 38.67 \text{ Persons}$$

$$\text{Productivity} = \frac{\text{KLOC}}{E} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

$$P = 176 \text{ LOC/PM}$$

Intermediate COCOMO

- Basic COCOMO model assumes
 - effort and development time depend on product size alone.
- However, several parameters affect effort and development time:
 - Reliability requirements
 - Availability of CASE tools and modern facilities to the developers
 - Size of data to be handled

Intermediate COCOMO

- For accurate estimation,
 - the effect of all relevant parameters must be considered:
 - Intermediate COCOMO model recognizes this fact:
 - refines the initial estimate obtained by the basic COCOMO by using a set of 15 cost drivers (multipliers).

Intermediate COCOMO (CONT.)

- Rate different parameters on a scale of one to three:
 - multiply cost driver values with the estimate obtained using the basic COCOMO.

Intermediate COCOMO (CONT.)

- Cost driver classes:
 - **Product:** Inherent complexity of the product, reliability requirements of the product, etc.
 - **Computer:** Execution time, storage requirements, etc.
 - **Personnel:** Experience of personnel, etc.
 - **Development Environment:** Sophistication of the tools used for software development.

INTERMEDIATE COCOMO MODEL

- Intermediate COCOMO computes software development effort as function of program size and a set of "cost drivers" that include subjective assessment of product, hardware, personnel and project attributes.
- The Intermediate COCOMO model refines the initial estimate obtained from the basic COCOMO by scaling the estimate up or down based on attributes of software development.
- This model uses a set of 15 cost drivers, these cost drivers are multiplied with the initial cost and effort estimates to scale the estimates up and down.
- This extension considers a set of "cost drivers", each with a number of subsidiary attributes:
 - Product attributes
 - ❖ Required software reliability
 - ❖ Size of application database
 - ❖ Complexity of the product

- Hardware attributes
 - ◊ Run-time performance constraints
 - ◊ Memory constraints
 - ◊ Volatility of the virtual machine environment
 - ◊ Required turnaround time
- Personnel attributes
 - ◊ Analyst capability
 - ◊ Software engineering capability
 - ◊ Applications experience
 - ◊ Virtual machine experience
 - ◊ Programming language experience
- Project attributes
 - ◊ Use of software tools
 - ◊ Application of software engineering methods
 - ◊ Requirements



Cost Drivers	Very Low	Low	Nominal	High	Very High
Product Attributes					
Required Software Reliability	0.75	0.88	1.00	1.15	1.40
Size of Application Database		0.94	1.00	1.08	1.16
Complexity of The Product	0.70	0.85	1.00	1.15	1.30
Hardware Attributes					
Runtime Performance Constraints			1.00	1.11	1.30
Memory Constraints			1.00	1.06	1.21
Volatility of the virtual machine environment	0.87	1.00	1.15	1.30	
Required turnabout time	0.94	1.00	1.07	1.15	

Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project Attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

Cost Drivers	Very Low	Low	Nominal	High	Very High	Extra High
Required Reliability	.75	.88	1.00	1.15	1.40	1.40
Database Size	.94	.94	1.00	1.08	1.16	1.16
Product Complexity	.70	.85	1.00	1.15	1.30	1.65
Execution Time Constraint	1.00	1.00	1.00	1.11	1.30	1.66
Main Storage Constraint	1.00	1.00	1.00	1.06	1.21	1.56
Virtual Machine Volatility	.87	.87	1.00	1.15	1.30	1.30
Comp Turn Around Time	.87	.87	1.00	1.07	1.15	1.15
Analyst Capability	1.46	1.19	1.00	.86	.71	.71
Application Experience	1.29	1.13	1.00	.91	.82	.82
Programmers Capability	1.42	1.17	1.00	.86	.70	.70
Virtual machine Experience	1.21	1.10	1.00	.90	.90	.90
Language Experience	1.14	1.07	1.00	.95	.95	.95
Modern Prog Practices	1.24	1.10	1.00	.91	.82	.82
SW Tools	1.24	1.10	1.00	.91	.83	.83
Required Dev Schedule	1.23	1.08	1.00	1.04	1.10	1,10

Intermediate COCOMO equation:

$$E = a_i (KLOC) b_i * EAF$$

$$D = c_i (E) d_i$$

Intermediate COCOMO Model

Software Projects	a	b
Organic	3.2	1.05
Semi Detached	3.0	1.12
Embedded	2.8	1.20

Example:

Consider a project having 30,000 lines of code which in an embedded software with critical area hence reliability is high. The estimation can be

$$E = a_1 (\text{KLOC})^{a_2} \times (\text{EAF})$$

As reliability is high EAF=1.15(product attribute)

$$2.8(30) \exp 1.20 * 1.15$$

$$\text{productivity} = \text{KLOC/Effort} = 30/191$$

Software Project	a_1	a_2	b_1	b_2
Organic	2.4	1.05	2.5	0.38
Semi-Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Example:

Consider a project having 30,000 lines of code which in an embedded software with critical area hence reliability is high. The estimation can be

$$E = a_1 (KLOC)^{b_1} \times (EAF)$$

As reliability is high EAF=1.15(product attribute)

$$a_1 = 2.8$$

$$b_1 = 1.20 \text{ for embedded software}$$

$$E = 2.8(30)^{1.20} \times 1.15$$

$$= 191 \text{ person month}$$

$$D = b_2 (E)^{b_2}$$

$$= 2.5(191)^{0.32}$$

$$= 13 \text{ months approximately}$$

$$N = E/D = 191/13$$

$$N = 15 \text{ persons approx.}$$

Merits:

- This model can be applied to almost to entire software product for easy and rough cost estimation during early stage.
- It can be applied at the software product component level for obtaining more accurate cost estimation.

Demerits:

- The effort multipliers are not dependent on phases.
- A product with many components is difficult to estimate.

SHORTCOMING OF BASIC AND INTERMEDIATE COCOMO MODELS

- Both models:
 - ❖ consider a software product as a single homogeneous entity:
 - ❖ However, most large systems are made up of several smaller sub-systems.
 - ❖ Some sub-systems may be considered as organic type, some may be considered embedded, etc.
 - ❖ For some the reliability requirements may be high, and so on.
 - ❖ So, complete COCOMO was proposed to overcome these limitations of basic and intermediate COCOMO.

Complete COCOMO

- Cost of each sub-system is estimated separately.
- Costs of the sub-systems are added to obtain total cost.
- Reduces the margin of error in the final estimate.

Complete COCOMO Example

- A Management Information System (MIS) for an organization having offices at several places across the country:
 - Database part (**semi-detached**)
 - Graphical User Interface (GUI) part (**organic**)
 - Communication part (**embedded**)
- Costs of the components are estimated separately:
 - summed up to give the overall cost of the system.

Relation between LOC and FP

- Relationship:

- $LOC = Language \ Factor * FP$
- where
 - **LOC** (Lines of Code)
 - **FP** (Function Points)

Relation between LOC and FPs

Language	LOC/FP
assembly	320
C	128
Cobol	105
Fortan	105
Pascal	90
Ada	70
OO languages	30
4GL languages	20

Language	QSM SLOC/FP Data			
	Avg	Median	Low	High
ABAP (SAP) *	28	18	16	60
ASP*	51	54	15	69
Assembler *	119	98	25	320
Brio +	14	14	13	16
C *	97	99	39	333
C++ *	50	53	25	80
C# *	54	59	29	70
COBOL *	61	55	23	297
Cognos Impromptu Scripts +	47	42	30	100
Cross System Products (CSP) +	20	18	10	38
Cool:Gen/IEF *	32	24	10	82
Datastage	71	65	31	157
Excel *	209	191	131	315
Focus *	43	45	45	45
FoxPro	36	35	34	38
HTML *	34	40	14	48

Java *	53	53	14	134
JavaScript *	47	53	31	63
JCL *	62	48	25	221
LINC II	29	30	22	38
Lotus Notes *	23	21	19	40
Natural *	40	34	34	53
.NET *	57	60	53	60
Oracle *	37	40	17	60
PACBASE *	35	32	22	60
Perl *	24	15	15	60
PL/1 *	64	80	16	80
PL/SQL *	37	35	13	60
Powerbuilder *	26	28	7	40
REXX *	77	80	50	80
Sabretalk *	70	66	45	109
SAS *	38	37	22	55
Siebel *	59	60	51	60
SLOGAN *	75	75	74	75
SQL *	21	21	13	37
VB.NET *	52	60	26	60
Visual Basic *	42	44	20	60



Elaborate COCOMO method of cost estimation

Software Engineering(SE)

CSC 601



Subject Incharge

Varsha Nagpurkar
Assistant Professor
Room No. 407

email: varshanagpurkar@sfit.ac.in



Module-3

Project Scheduling and Tracking

5/25/2021

Module-3 Syllabus

3.0	Project Scheduling and Tracking	08
3.1	Management Spectrum, 3Ps (people, product and process)	
3.2	Process and Project metrics	
3.3	Software Project Estimation: LOC, FP, Empirical Estimation Models - COCOMO II Model, Specialized Estimation Techniques	
3.4	Project scheduling: Defining a Task Set for the Software Project, Timeline charts, Tracking the Schedule, Earned Value Analysis	

THE MANAGEMENT SPECTRUM

- Effective software project management focuses on the four P's: people, product, process, and project.
- The People
- Every organization needs to continually improve its ability to attract, develop, motivate, organize, and retain the workforce needed to accomplish its strategic business objectives”.
- The people management capability maturity(PM-CMM) model developed by Software Engineering Institute(SEI) defines the following key practice areas for software people:

THE MANAGEMENT SPECTRUM-The People

- Recruiting, selection, performance management, training, compensation, career development, organization and work design, and team/culture development
- Organizations that achieve high levels of maturity in the people management area have a higher likelihood of implementing effective software project management practices.
- The People-CMM is a companion to the Software Capability Maturity Model—Integration that guides organizations in the creation of a mature software process

THE MANAGEMENT SPECTRUM-The People-Stakeholders

- The software process(and every software project)is populated by stakeholders who can be categorized into one of five categories:
 - 1.Senior managers- who define the business issues that often have a significant influence on the project.
 2. Project (technical) managers- who must plan, motivate, organize, and control the practitioners who do software work

THE MANAGEMENT SPECTRUM-The People-Stakeholders

3. Practitioners- who deliver the technical skills that are necessary to engineer a product or application.
4. Customers-who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.
5. End users- who interact with the software once it is released for production use.

THE MANAGEMENT SPECTRUM-The People-Team Leaders

- In an excellent book of technical leadership, Jerry Weinberg suggests a MOI Model of leadership
- 1. Motivation- The ability to encourage (by “push or pull”) technical people to produce to their best ability.
- 2. Organization.- The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product

THE MANAGEMENT SPECTRUM-The People-Team Leaders

- 3. Ideas or innovation.The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

Characteristics that define effective project management

- Problem solving-Project manager can diagnose the technical and organizational issues that are most relevant, systematically structure a solution or properly motivate other practitioners to develop the solution
- Managerial identity- A good project manager must take charge of the project. She must have the confidence to assume control when necessary and the assurance to allow good technical people to follow their instincts.

Characteristics that define effective project management

- Achievement-To optimize the productivity of a project team,a manager must reward initiative and accomplishment
- Influence and team building-An effective project manager must be able to “read” people;she must be able to understand verbal and nonverbal signals and react to the needs of the people sending these signals

The Software Team

- The “best” team structure depends on the management style of the organization, the number of people who will populate the team and their skill levels, and the overall problem difficulty
- Mantei suggested following 7 factors for considerations before structuring a team

The Software Team

- The difficulty of the problem to be solved
- The “size” of the resultant program(s) in lines of code or function points
- The time that the team will stay together(team lifetime)
- The degree to which the problem can be modularized
- The required quality and reliability of the system to be built

The Software Team

- The rigidity of the delivery date
- The degree of communication required for the project

The Software Team-

To achieve a high-performance team:

- Team members must have trust in one another
- The distribution of skills must be appropriate to the problem
- Independent-minded person may have to be excluded from the team, if team unity is to be maintained
 - Regardless of team organization, the objective for every project manager is to help create a team that exhibits unity among the team members

5/25/2021

Agile teams

- In recent years, agile software development has been proposed
- The agile philosophy provides the following
 - It encourages customer satisfaction by early incremental delivery of software
 - It provides small highly motivated project teams
 - It consists of informal methods
 - It provides minimal software engineering work products
 - It results in overall development simplicity

Agile teams

- The small sized and highly motivated project team, also called an agile team, adopts many of the characteristics of successful software project teams.

Coordination and Communication Issues

- There are many reasons that software projects get into trouble
- The scale of many development efforts is large, leading to complexity, confusion, and significant difficulties in coordinating team members
- Uncertainty is common, resulting in a continuing stream of changes
- Interoperability-New software must communicate with existing software and conform to predefined constraints imposed by the system

Coordination and Communication Issues

- Characteristics of modern software-Scale,uncertainty, and interoperability-are facts of life
- To deal with them effectively,a Software Engineering team must establish effective methods for coordinating the people who do the work
- To accomplish this mechanisms for formal and informal communication among team members and between multiple teams must be established

Coordination and Communication Issues

- Formal communication is accomplished through “writing”, structured meetings, and other relatively noninteractive and impersonal communication channels
- Informal communication is more personal-Members of a team share ideas on an adhoc basis, ask for help as problem arise, and interact with one another on a daily basis

The Product

- As requirements may change regularly as the project proceeds,a plan is needed
- So the first activity in software project management is the determination of software scope
- Scope is defined by answering the following questions:
 - Context-How does the software to be built fit into a larger system,product,or business context, and what constraints are imposed as a result of the context?

- Information objectives-What customer-visible data objects are produced as output from the software? What data objects are required for input?
- Function and performance-What functions does the software perform to transform input data into output?

- Software project scope must be unambiguous and understandable at the management and technical levels
- Software scope must be bounded-
 - Quantitative data(e.g.,number of simultaneous users,size of mailing list,maximum allowable response time)are stated explicitly;constraints and/or limitations(e.g.,product cost restricts memory size)are noted

The Process

- The framework activities that characterize the software process are applicable to all software projects
- The project manager must decide which process model is most appropriate for
 - The customers who have requested the product and the people who will do the work
 - The characteristics of the product itself, and
 - The project environment in which the software team works

- When a process model has been selected, the team then defines a preliminary project plan based on the set of process framework activities.
- Once the preliminary plan is established, process decomposition begins
- That is, a complete plan, reflecting the work tasks required to populate the framework activities, must be created



The Project

CHAPTER 3

Project Planning and Scheduling

“Failing to plan is planning to fail”

- **Planning:**

- “what” is going to be done, “how”, “where”, by “whom”, and “when”
- for effective monitoring and control of complex projects

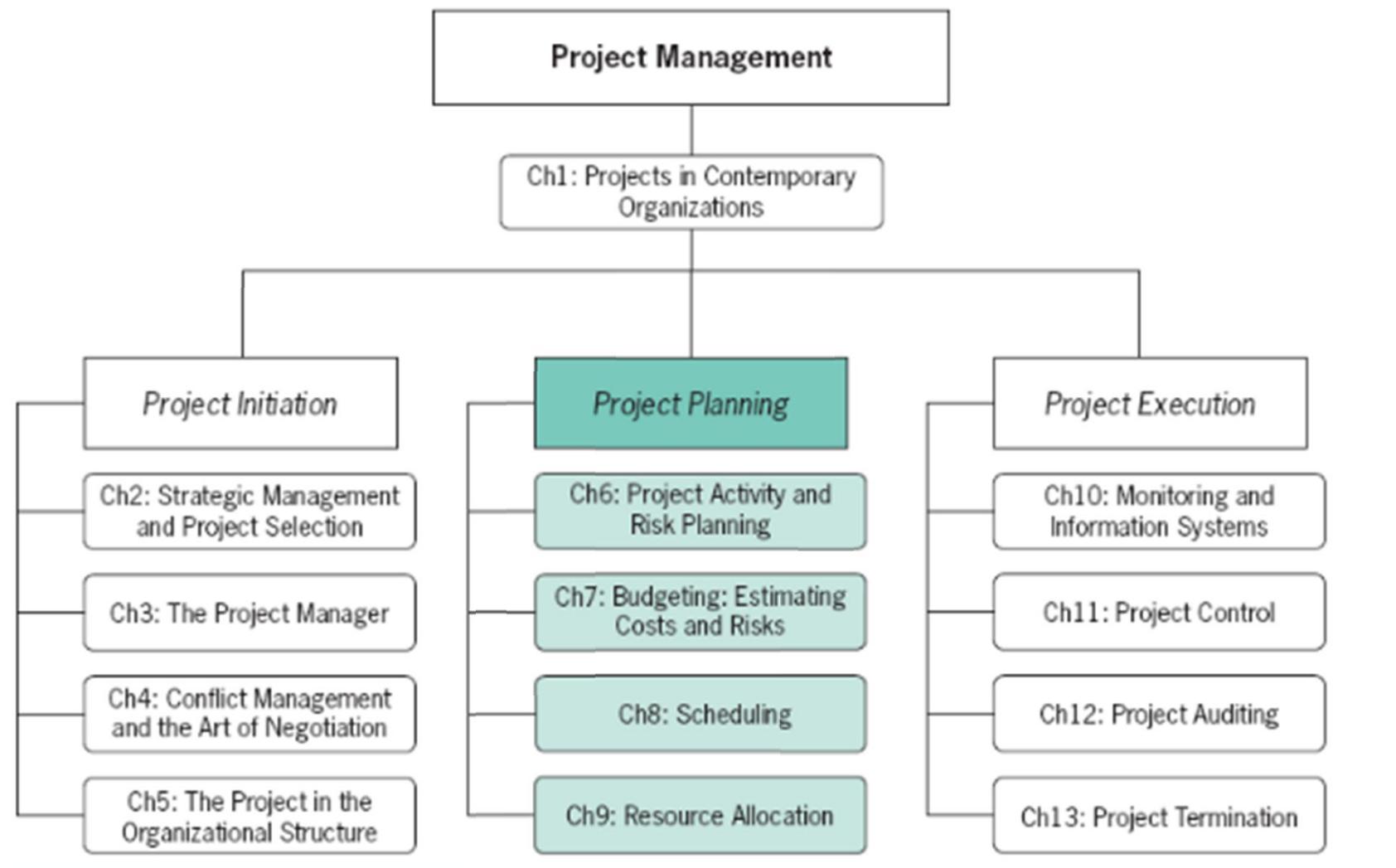
“It’s about time”

- **Scheduling:**
 - “what” will be done, and “who” will be working
 - relative timing of tasks & time frames
 - a concise description of the plan

“Once you plan your work, you must work your plan”

- Planning and Scheduling occurs:
 - **AFTER** you have decided how to do the work
 - “The first idea is not always the best idea.”
- Requires discipline to “work the plan”
 - The act of development useful,
 - But need to monitor and track
 - only then, is a schedule an effective management tool
 - **as-built schedules**

Project Management



Contents

- Project Planning and Scheduling: Work Breakdown structure (WBS) and linear responsibility chart,
- Project cost estimation and budgeting,
- Top down and bottoms up budgeting,
- Networking and Scheduling techniques.
- PERT, CPM, GANTT chart,
- Introduction to Project Management Information System (PMIS).



- As the Beagle 2 Mars probe designed jointly by the European Space Agency and British National Space Center headed to Mars in December of 2003, contact was lost and it was never heard from again.
- In retrospect, it appears that inadequate project planning (and preplanning) was to blame.
- Excessive pressure on time, cost, and weight compromised the mission right from the start.
- With insufficient public funding, the design team had to spend much of their time raising private funds instead of addressing difficult technical issues.
- In addition, late changes forced the team to reduce the Beagle's weight from 238 pounds to 132 pounds! And when the three airbags failed to work properly in testing, a parachute design was substituted but inadequately tested due to lack of time.

- Requisite financing be available at the outset of a project
- Formal project reviews be conducted on a regular basis
 - Milestones should be established where all stakeholders reconsider the project
 - Expectations of potential failure should be included in the funding consideration
 - Robust safety margins should be included and funded for uncertainties

PLANNING

- The purpose of planning is to facilitate the later accomplishment.
- Smooth the path from idea to outcome.
- Planning can be interpreted as “setting objective”, " defining the scope" , “Identifying requirements” .
- Budget and schedule are main important parts of proj plan.

Initial Project Coordination and the Project Charter

- Project launch meetings are used to decide on participating in the project with clear goal by senior mgmt people.
- Discussion should be according to complexity of project.
- Change management (“change without communication”)
- Outcomes include:
 - Technical scope
 - Areas of responsibility
 - Delivery dates or budgets
 - Risk management group
 - Clear deliverables

Project Management in Practice

Child Support Software a Victim of Scope Creep



In March 2003, the United Kingdom's Child Support Agency (CSA) started using their new £456 million (\$860 million) software system for receiving and disbursing child support payments. However, by the end of 2004 only about 12 percent of all applications had received payments, and even those took about three times longer than normal to process. CSA thus threatened to scrap the entire system and withhold £1 million (\$2 million) per month in service payments to the software vendor. The problem was thought to be due to both scope creep and the lack of a risk management strategy. The vendor claimed that the proj-

ect was disrupted constantly by CSA's 2500 change requests, while CSA maintained there were only 50, but the contract did not include a scope management plan to help define what constituted a scope change request. And the lack of a risk management strategy resulted in no contingency or fallback plans in case of trouble, so when project delays surfaced and inadequate training became apparent, there was no way to recover.

Source: Project Management Institute. "Lack of Support," *PM Network*, January, 2005, p. 1.

Outside Clients

- When it is for outside clients, specifications cannot be changed without the client's permission
- Client may place budget constraints on the project
- May be competing against other firms
- Sales vs.. Design/Functional

Project Charter Elements

- Purpose
- Objectives
- Overview
- Schedules
- Resources
- Personnel
- Risk management plans
- Evaluation methods

Systems Integration

- Performance
- Effectiveness
- Cost

Starting the Project Plan: The WBS

- What is to be done
- When it is to be started and finished
- Who is going to do it

Starting the Project Plan: The WBS

Continued

- Some activities must be done sequentially
- Some activities may be done simultaneously
- Many things must happen when and how they are supposed to happen
- Each detail is uncertain and subjected to risk

Hierarchical Planning

- Major tasks are listed
- Each major task is broken down into detail
- This continues until all the activities to be completed are listed
- Need to know which activities “depend on” other activities

A Form to Assist Hierarchical Planning

ACTION PLAN				
Deliverables _____ _____				
Measure(s) of accomplishment _____ _____				
Key constraints and assumptions _____ _____				
TASKS	ESTIMATED RESOURCES	IMMEDIATE PREDECESSOR TASKS	ESTIMATED TIME DURATION(S)	ASSIGNED TO

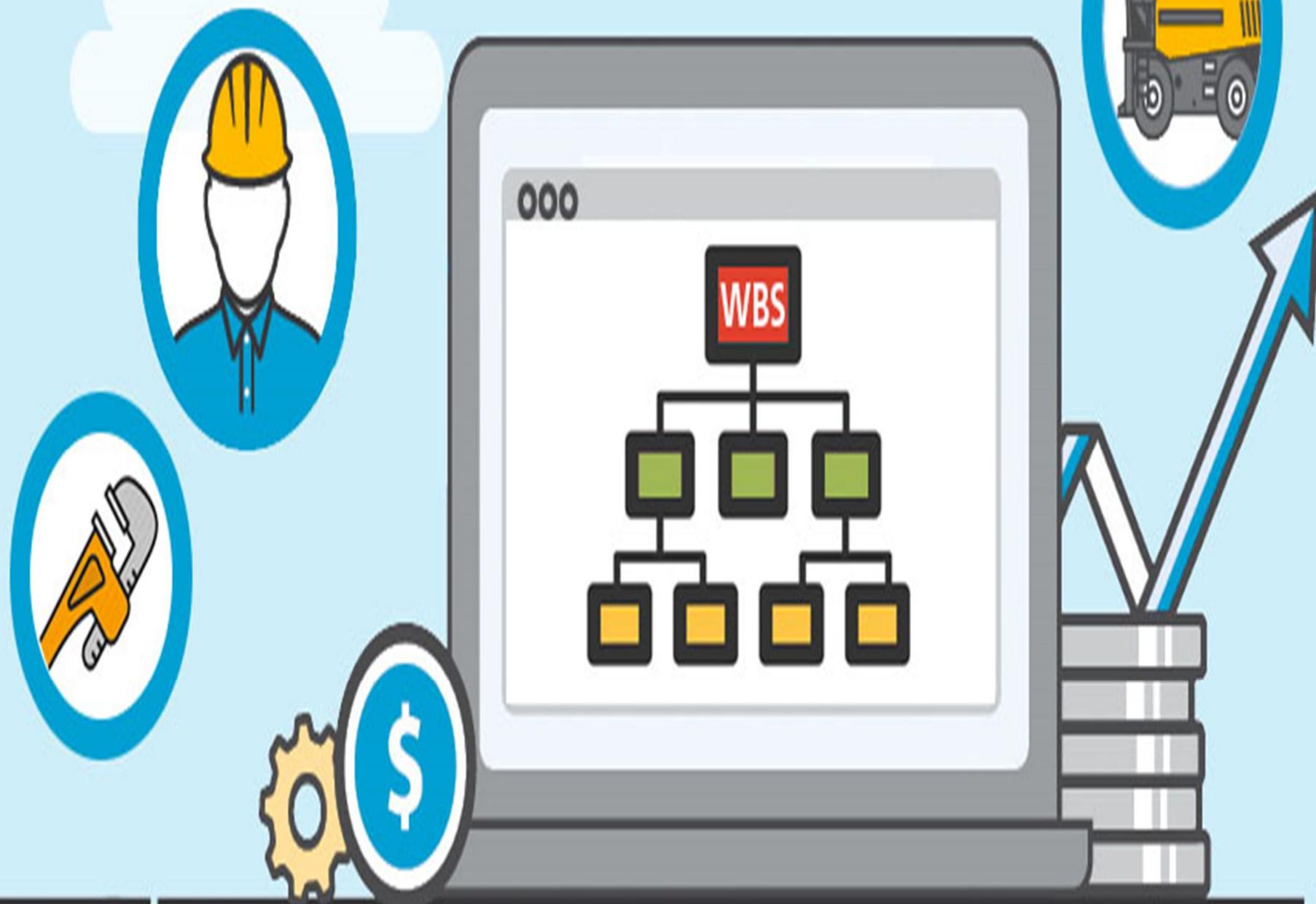
Figure 6-2

Career Day

ACTION PLAN Objective Career Day				
Steps	Responsibility	Time (weeks)	Prec.	Resources
1. Contact Organizations				
a. Print forms	Secretary	6	-	Print shop
b. Contact organizations	Program manager	15	1.a	Word processing
c. Collect display information	Office manager	4	1.b	
d. Gather college particulars	Secretary	4	1.b	
e. Print programs	Secretary	6	1.d	Print shop
f. Print participants' certificates	Graduate assistant	8	-	Print Shop
2. Banquet and Refreshments				
a. Select guest speaker	Program manager	14	-	
b. Organize food	Program manager	3	1.b	Caterer
c. Organize liquor	Director	10	1.b	Dept. of Liquor Control
d. Organize refreshments	Graduate assistant	7	1.b	Purchasing
3. Publicity and Promotion				
a. Send invitations	Graduate assistant	2	-	Word processing
b. Organize gift certificates	Graduate assistant	5.5	-	
c. Arrange banner	Graduate assistant	5	1.d	Print shop
d. Contact faculty	Program manager	1.5	1.d	Word processing
e. Advertise in college paper	Secretary	5	1.d	Newspaper
f. Class announcements	Graduate assistant	1	3.d	Registrar's Office
g. Organize posters	Secretary	4.5	1.d	Print shop
4. Facilities				
a. Arrange facility for event	Program manager	2.5	1.c	
b. Transport materials	Office manager	.5	4.a	Movers

Figure 6-4

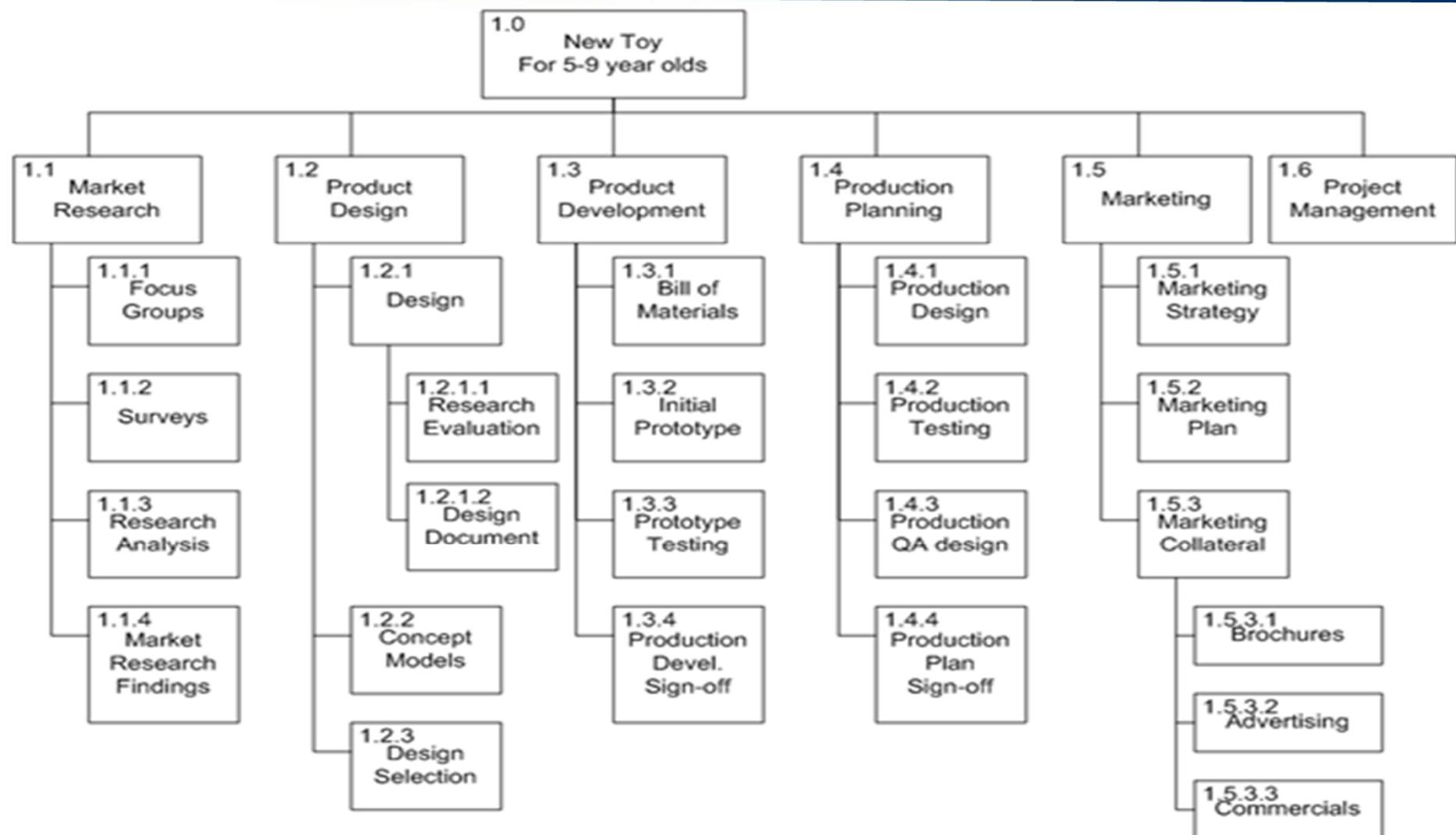
Work Breakdown Structure



The Work Breakdown Structure (WBS)

- A hierarchical planning process
- Breaks tasks down into successively finer levels of detail
- Continues until all meaningful tasks or work packages have been identified
- These make tracking the work easier
- Need separate budget/schedule for each task or work package

WBS EXAMPLE



A Visual WBS

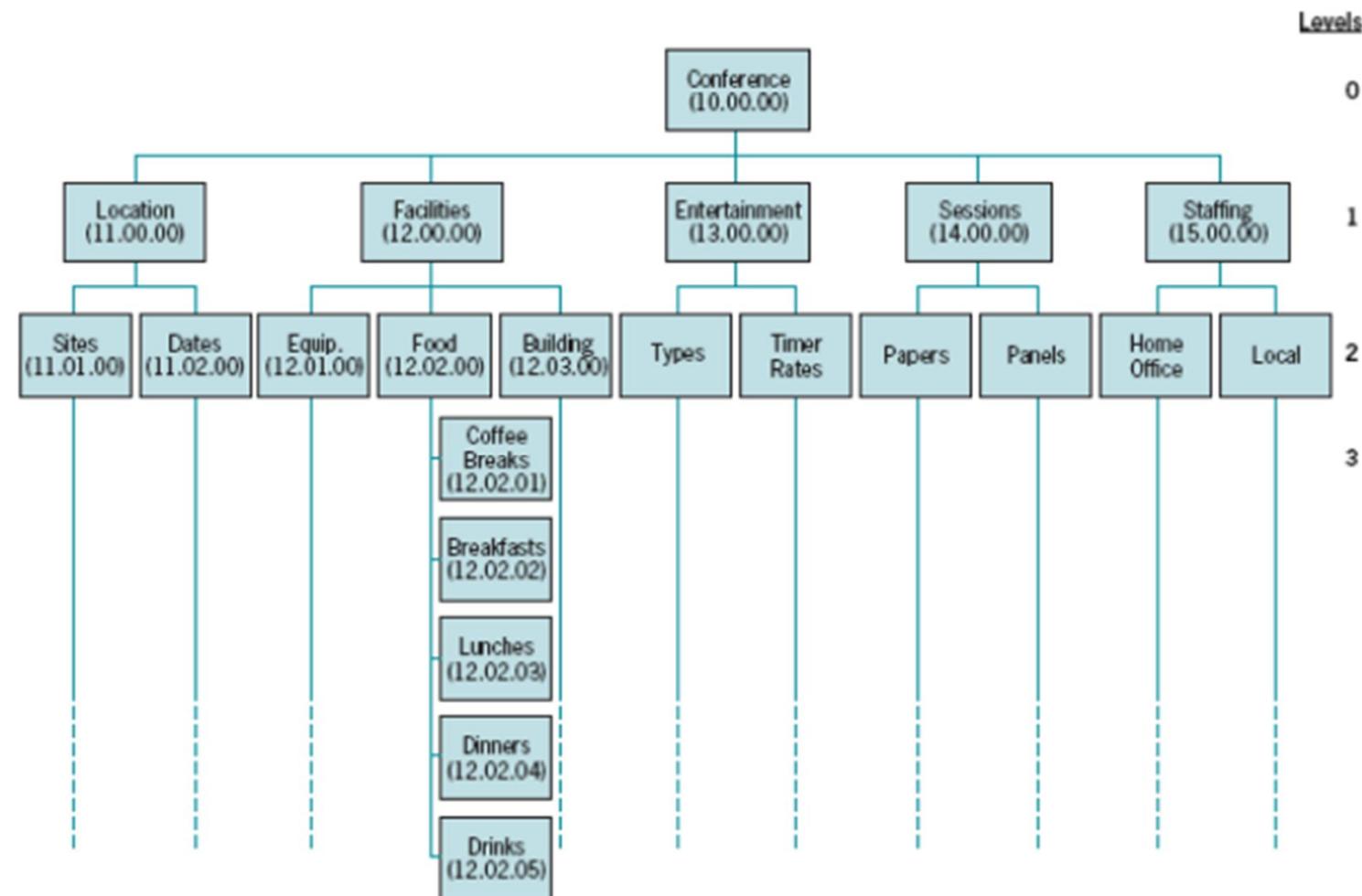
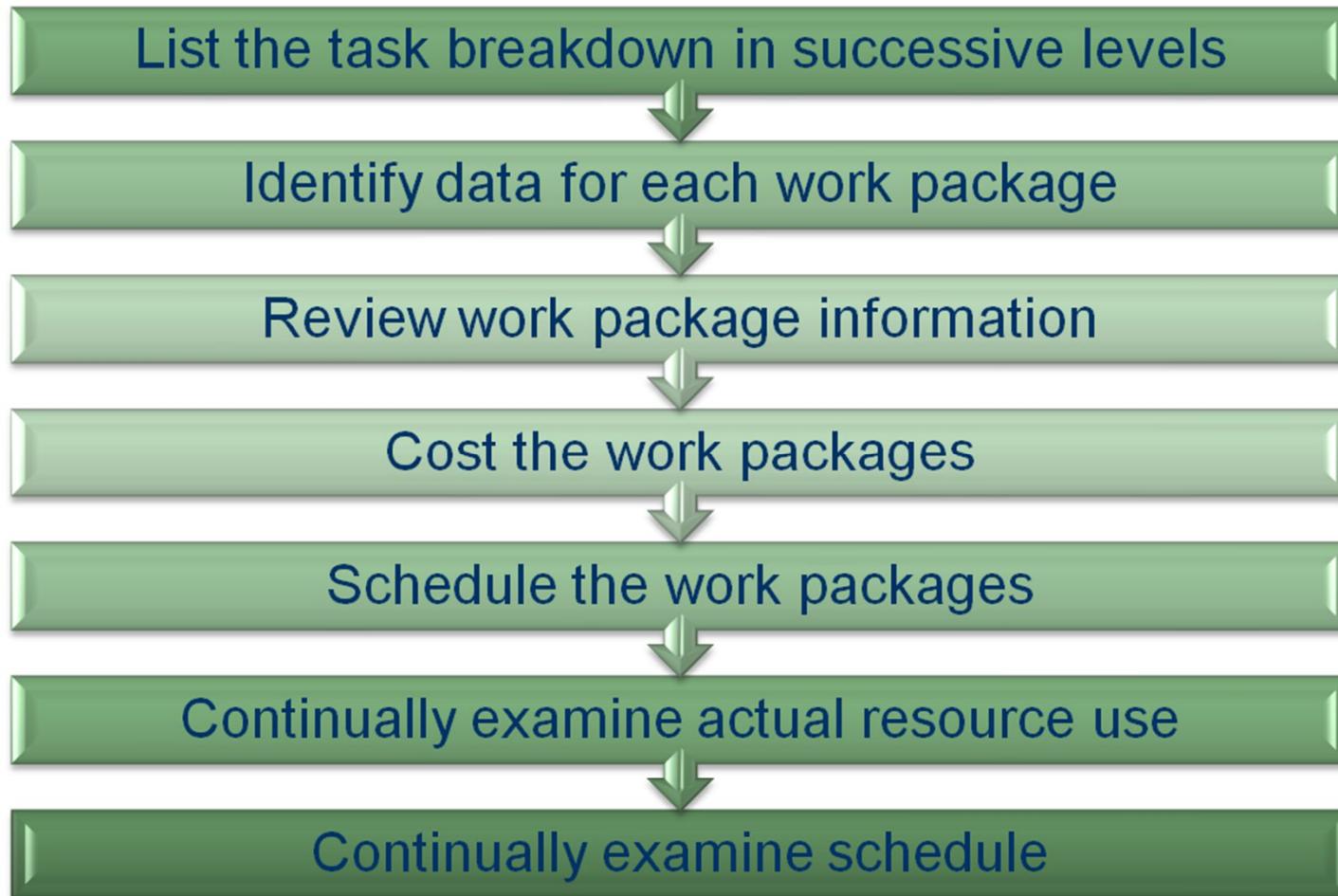


Figure 6-3

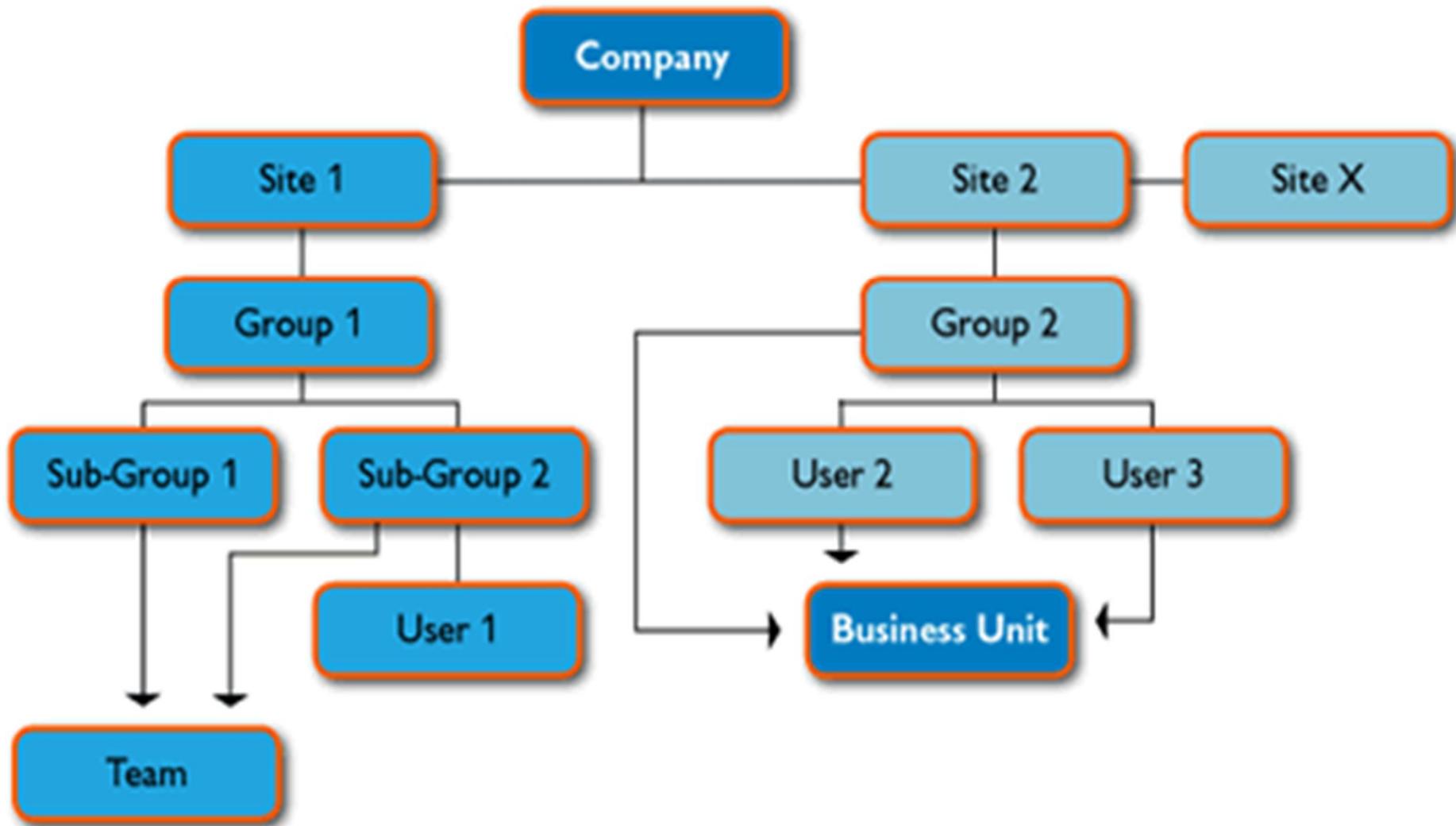
Steps to Create a WBS

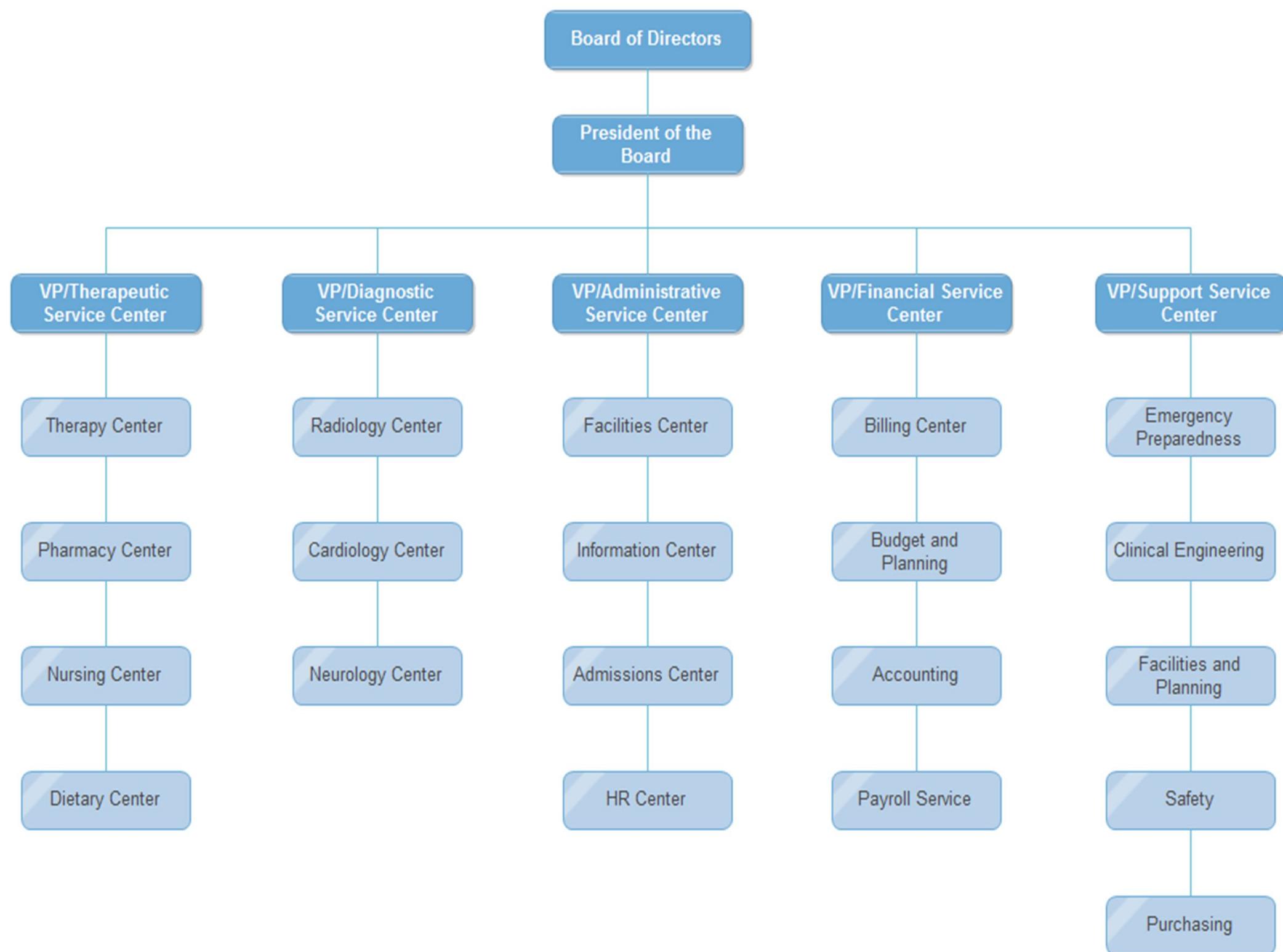


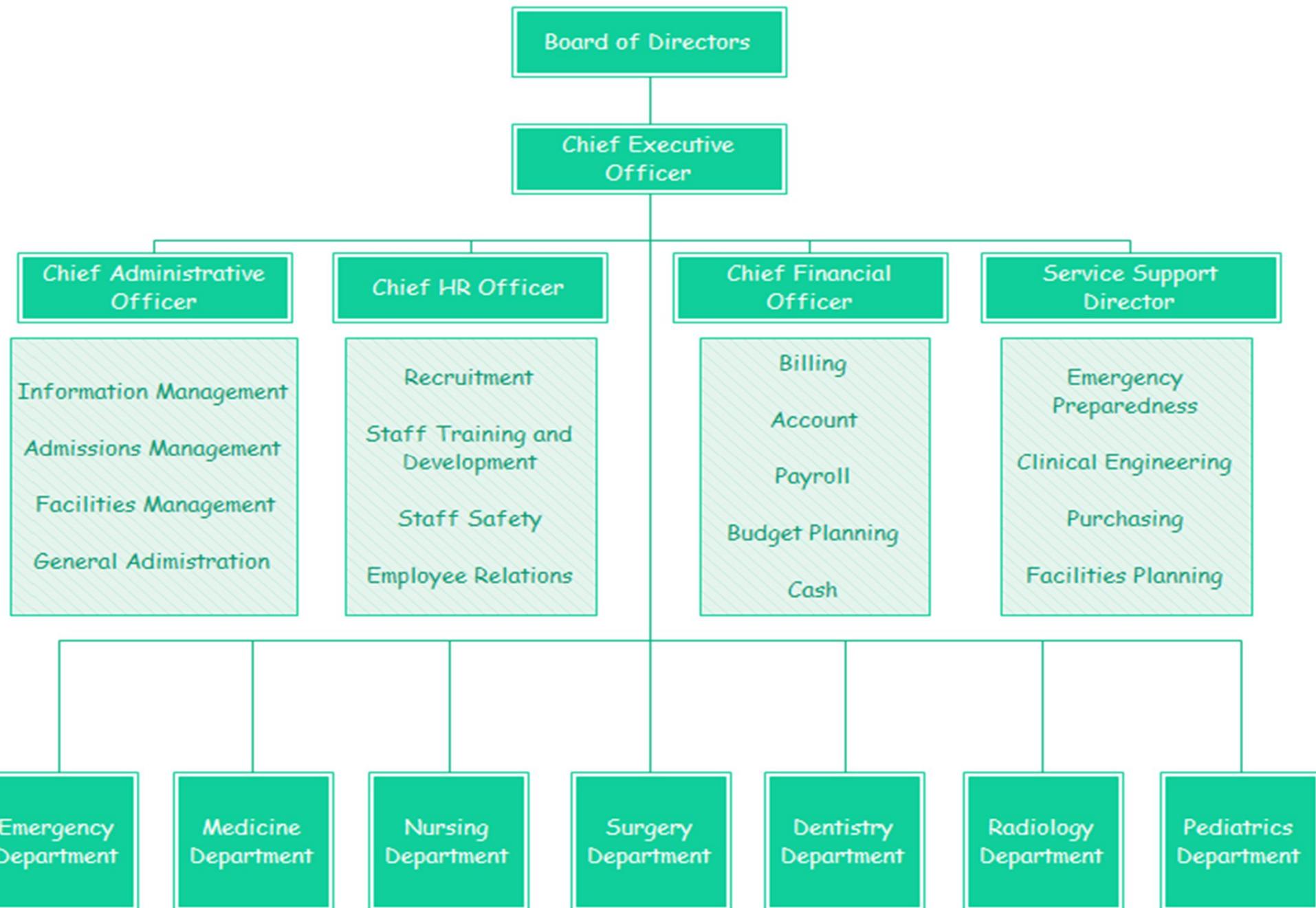
Human Resources

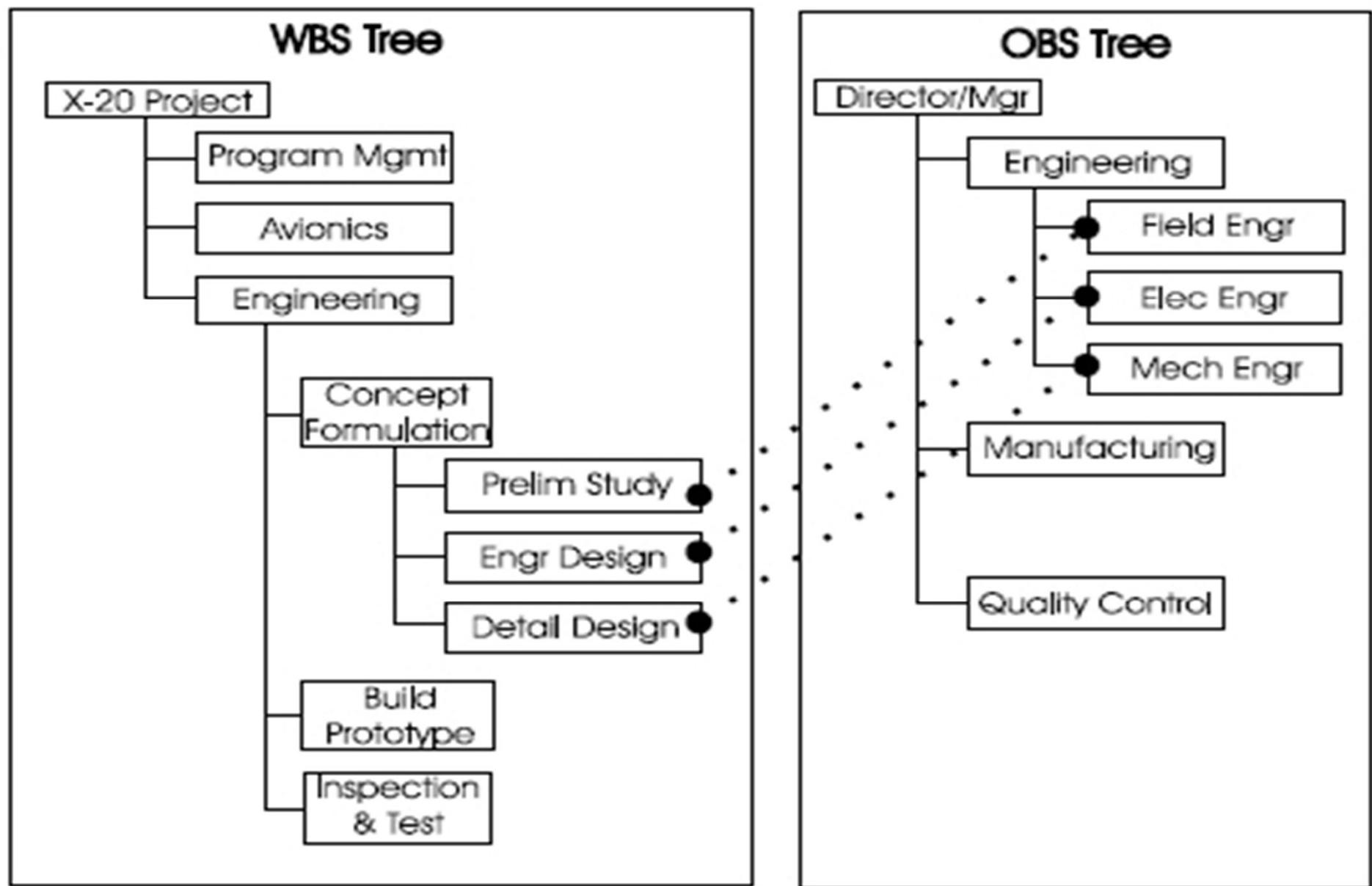
- Useful to create a table that shows staff needed to execute **WBS** tasks
- One approach is a **organizational breakdown structure(OBS)**
 - Organizational units responsible for each WBS element
 - Who must approve changes of scope
 - Who must be notified of progress
- WBS and OBS may not be identical

OBS









The Responsibility (RACI) Matrix

- Another approach is the **Responsible, Accountable, Consult, Inform (RACI)** matrix
 - Also known as a responsibility matrix, a **linear responsibility chart**, an assignment matrix, a responsibility assignment matrix
- Shows critical interfaces
- Keeps track of who must approve what and who must be notified



DoveCarpenter

"MISS WILCOX, SEND IN SOMEONE TO BLAME."



Responsible



Accountable



Consulted



Informed

Step	Project Initiation	Project Executive	Project Manager	Business Analyst	Technical Architect	Application Developers
1	Task 1	C	A/R	C	I	I
2	Task 2	A	I	R	C	I
3	Task 3	A	I	R	C	I
4	Task 4	C	A	I	R	I

Sample RACI Matrix

		Responsibility					
WBS		Project Office				Field Oper.	
Subproject	Task	Project Manager	Contract Admin.	Project Eng.	Industrial Eng.	Field Manager	
Determine need	A1	○		●	▲		
	A2	■	○	▲	●		
Solicit quotations	B1	○	■	▲		●	
Write approp. request.	C1	■	▲	○	●		
	C2		●	○	▲		
	C3	●	■	▲		■	
"	"						
"	"						
"	"						

Legend:

- ▲ Responsible
- Support
- Notification
- Approval

	Vice-president	General manager	Project manager	Manager engineering	Manager software	Manager manufacturing	Manager marketing	Subprogram manager manufacturing	Subprogram manager software	Subprogram manager hardware	Subprogram manager services
Establish project plan	6	2	1	3	3	3	3	4	4	4	4
Define WBS		5	1	3	3	3	3	3	3	3	3
Establish hardware specs		2	3	1	4	4	4				
Establish software specs		2	3	4	1		4				
Establish interface specs		2	3	1	4	4	4				
Establish manufacturing specs		2	3	4	4	1	4				
Define documentation		2	1	4	4	4	4				
Establish market plan	5	3	5	4	4	4	1				
Prepare labor estimate			3	1	1	1		4	4	4	4
Prepare equipment cost estimate		3	1	1	1			4	4	4	4
Prepare material costs			3	1	1	1		4	4	4	4
Make program assignments			3	1	1	1		4	4	4	4
Establish time schedules		5	3	1	1	1	3	4	4	4	4

1 Actual responsibility 4 May be consulted
 2 General supervision 5 Must be notified
 3 Must be consulted 6 Final approval

Figure 6-8 Simplified linear responsibility chart.

Legend:

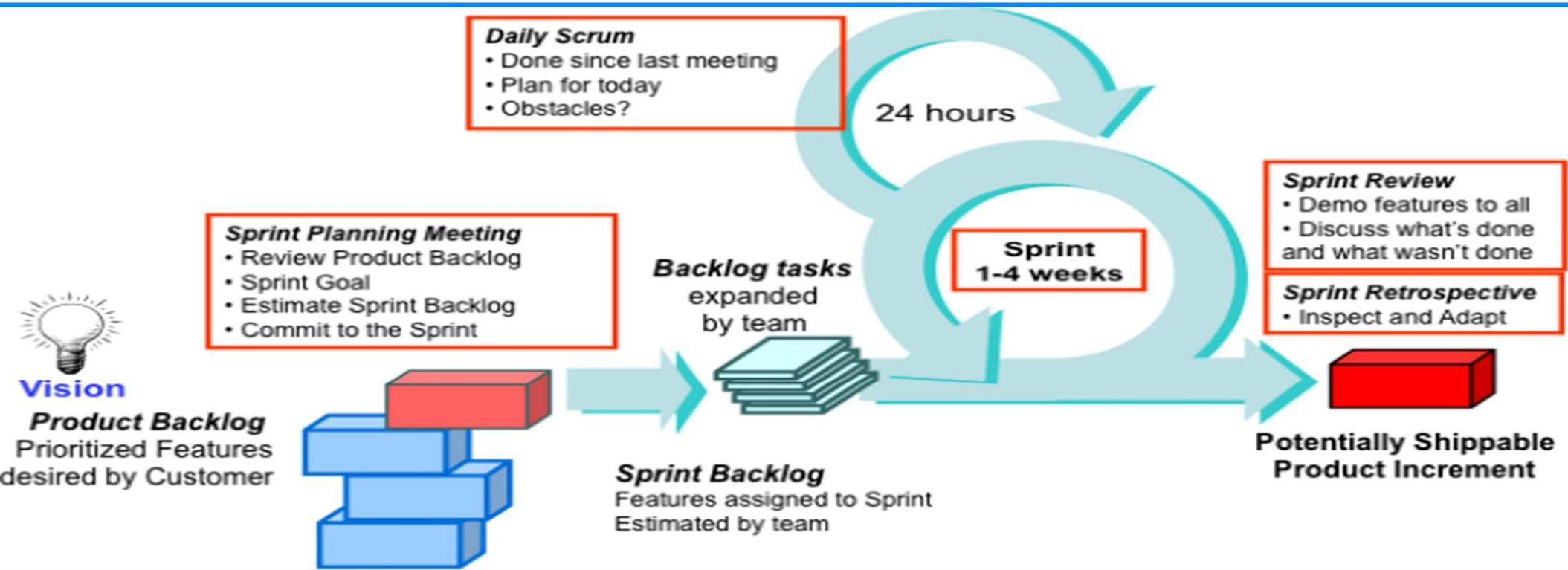
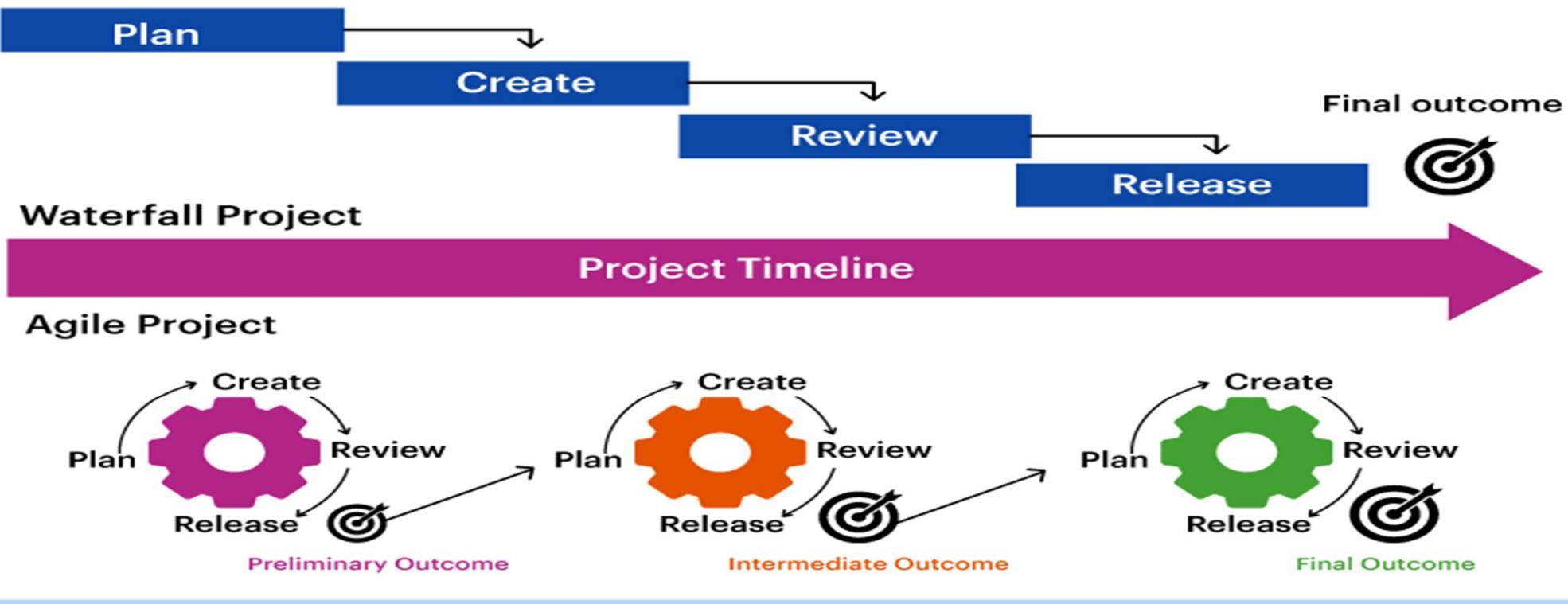
- Project completion
 - Contractual commitment
 - △ Planned completion
 - ▲ Actual completion
 - ^ Status date
 - Milestone planned
 - Milestone achieved
 - Planned progress
 - Actual progress

Note: As of Jan. 31, 2003, the project is one month behind schedule. This is due mainly to the delay in task C1, which was caused by the late completion of A2.

Figure 6-10 Projected baseline schedule.

Agile Project Planning and Management

- When scope cannot be determined in advance, traditional planning does not work
- Agile project management was developed to deal with this problem in IT
- Small teams are located at a single site
- Entire team collaborates
- Team deals with one requirement at-a-time with the scope frozen
- It is a charting system that illustrates the task's goal and the required action for each person



Agile planning – iteration

...

#1 iteration: weeks 1-2

	Owner	Status	Priority	Due date	
Review dev environment		Done	High	Jan 6	
Plan release		Done	Low	Jan 9	
Analyze deployment's data		Done	Mid	Jan 13	

#1 iteration: weeks 3-4

	Owner	Status	Priority	Due date	
Release plan review		For review	High	Jan 19	
Build release		Working on it	Low	Jan 23	
Test plan approval		For review	Mid	Jan 16	
Test user acceptance		Ongoing	High	Jan 26	

Iteration #1: Week 1

	Status	Priority	Type	Deadline	Time Est.	Time Spent
Decide Business ...	Done	Must Have	MILESTONE!	Feb 10	5 Hours	5 Hours
Feasibility Study	Working on it	High	Improvements	Feb 12	10 Hours	2 Hours
plan	Not Started	MILESTONE	MILESTONE!	Feb 14	7 Hours	6.75 Hours
Design	Waiting for Review	High	Increment			
+ Add						
					22 Hours sum	13.75 Hours sum

Iteration #2: Week 3

	Status	Priority	Type	Deadline	Time Est.	Time Spent
Release Plan Revie...	Working on it	High	Increment	Jan 20	2 Hours	
Build Release	Needs Design	Low	Increment	Jan 22	6 Hours	
Test Plan Approval	Needs Product	Medium	Increment		12 Hours	
Test User Accept...	Not Started	Must Have	Increment		14 Hours	
+ Add						
					34 Hours sum	0 Hours sum

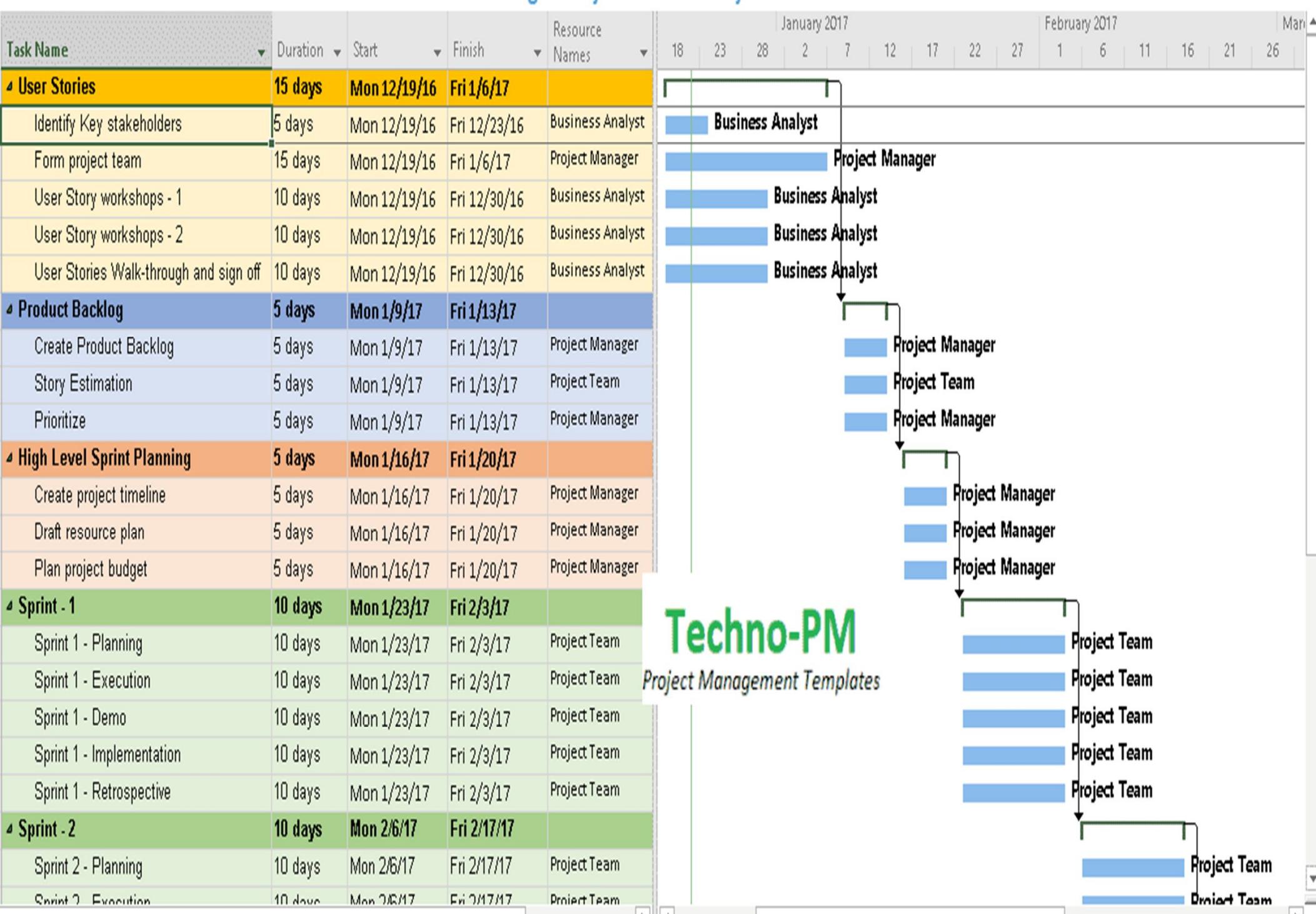
Project Management-How to use projectlibre.com for time line chart

- <https://www.youtube.com/watch?v=oiVnWX-J5Mo>

Project Scheduling-Timeline Charts

- When creating a software project schedule, the planner begins with a set of tasks (the work breakdown structure)
- If automated tools are used, the work breakdown is input as a task network or task outline
- Effort, duration, and start date are then input for each task
- In addition, tasks may be assigned to specific individuals
- As a consequence of this input, a timeline chart, also called a Gantt chart, is generated

Agile Project Plan MS Project



-
- 1. Analysis 2 days 7th May(Friday) 8 am to 5pm 10th May(Monday)(18 hrs)
 - 2. Design 3 days 11th May to 14th May

Interface Coordination Through Integration Management

- Managing a project requires a great deal of coordination
- Projects typically draw from many parts of the organization as well as outsiders
- All of these must be coordinated
- The RACI matrix helps the project manager accomplish this

Integration Management

- If the project is large multidisciplinary team(MT) will work together
- Integration will be Coordinating the work and timing of different groups
- RACI Matrix will be useful here as conflict and uncertainty can be arise in MT(e,g, Bank merging)
- Divide the project in to phases and then accomplish the task by teams

Managing Projects by Phases and Phase-Gates

- Break objectives into shorter term sub-objectives
- Project life cycle is used for breaking a project up into component **phases**
- Focus on specific, short-term output
- Lots of feedback between disciplines
- **Phase-Gate** : create different phases and keep “Review” as a gate to proceed in next phase

Risk Management

- Projects are risky, uncertainty is high
- Project manager must manage this risk
- This is called “risk management”
- Risk varies widely between projects
- Risk also varies widely between organizations
- Risk management should be built on the results of prior projects

Parts to Risk Management

- Risk management planning
- Risk identification
- Qualitative risk analysis
- Quantitative risk analysis
- Risk response planning
- Risk monitoring and control
- The risk management register

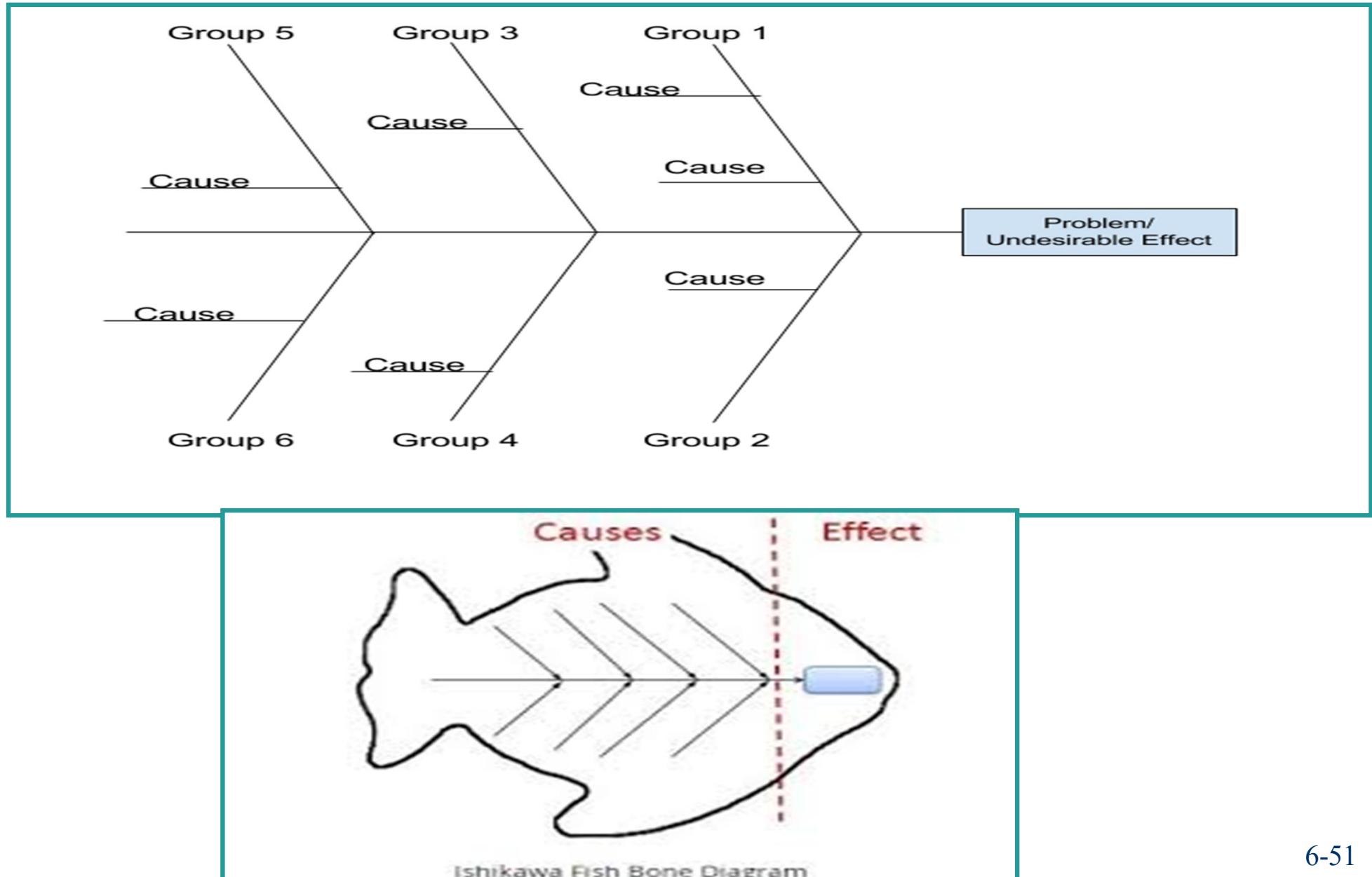
Risk Management Planning

- Need to know the risk involved before selecting a project
- Risk management plan must be carried out before the project can be formally selected
- At first, focus is on externalities
 - Track and estimate project survival
- Project risks take shape during planning
- Often handled by project office
- May include analytical techniques.

Risk Identification

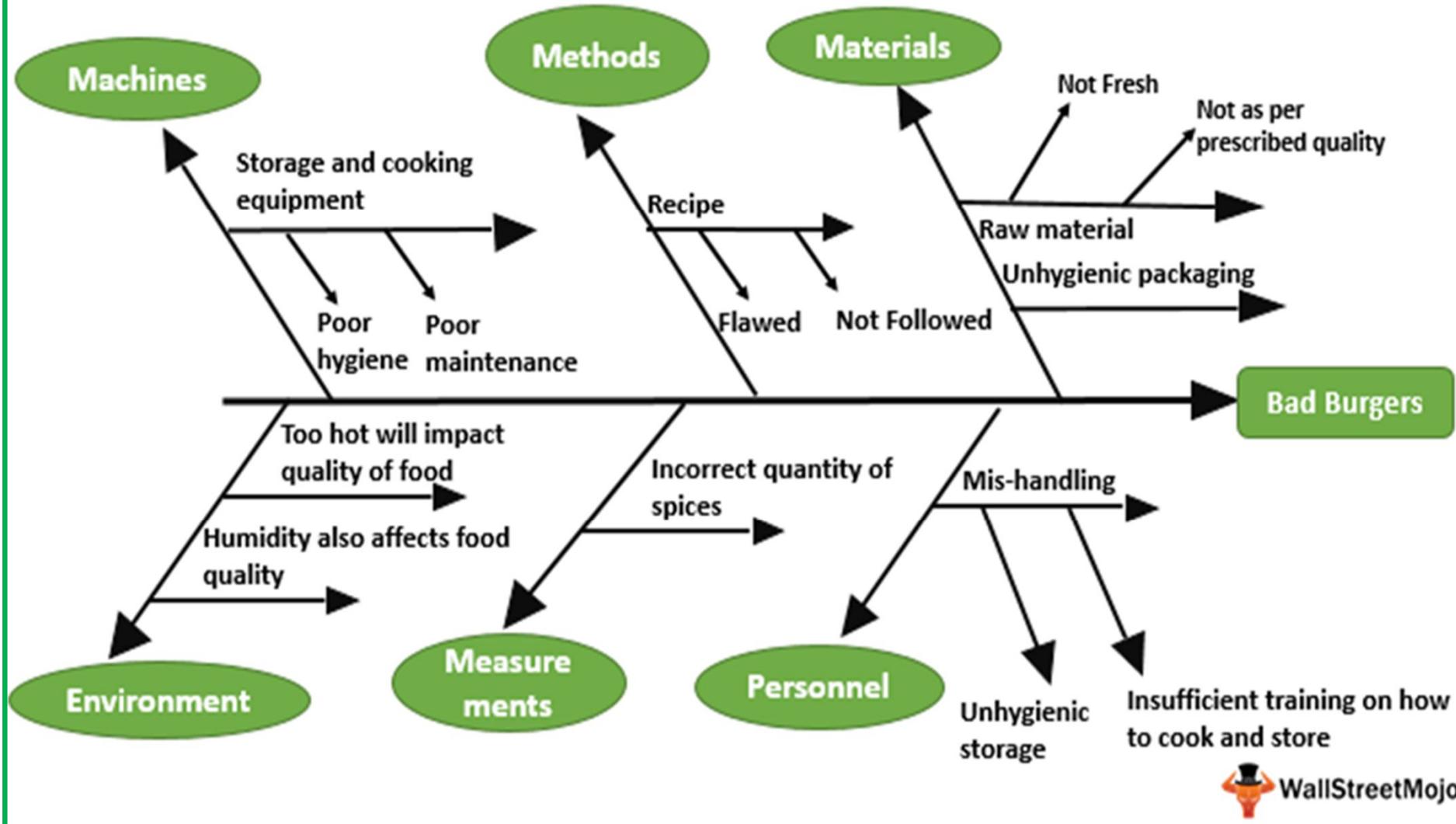
- Risk is dependent on technology and environmental factors e.g. economic, cultural factors
- **Delphi method** is useful for identifying project risks
- Other methods include brainstorming, nominal group techniques, checklists, and attribute listing
- May also use **cause-effect** diagrams(fish bone diagram), flow charts, influence charts, **SWOT** analysis

Risk Identification(Cause-Effect Analysis-Fish Bone Diagram)

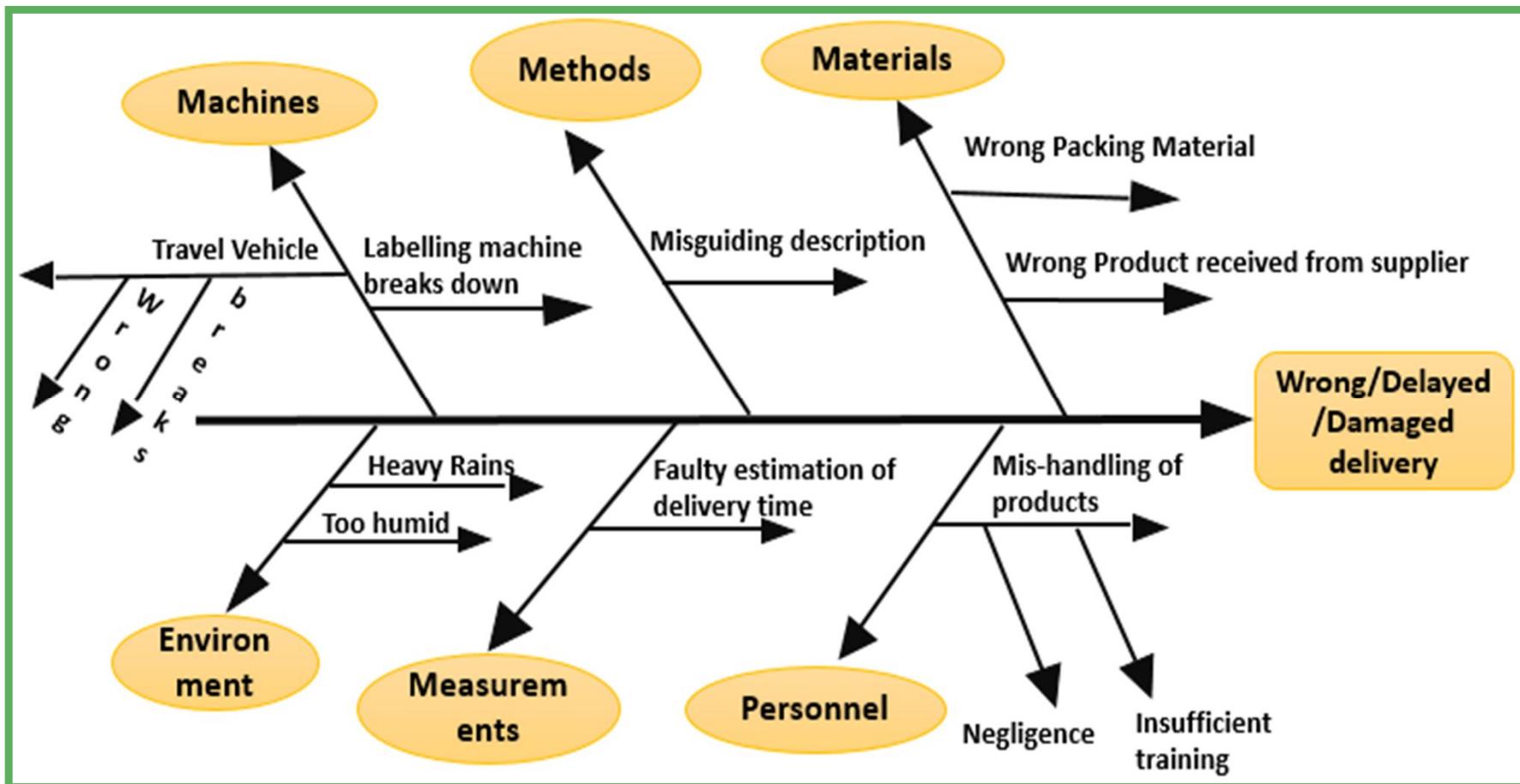


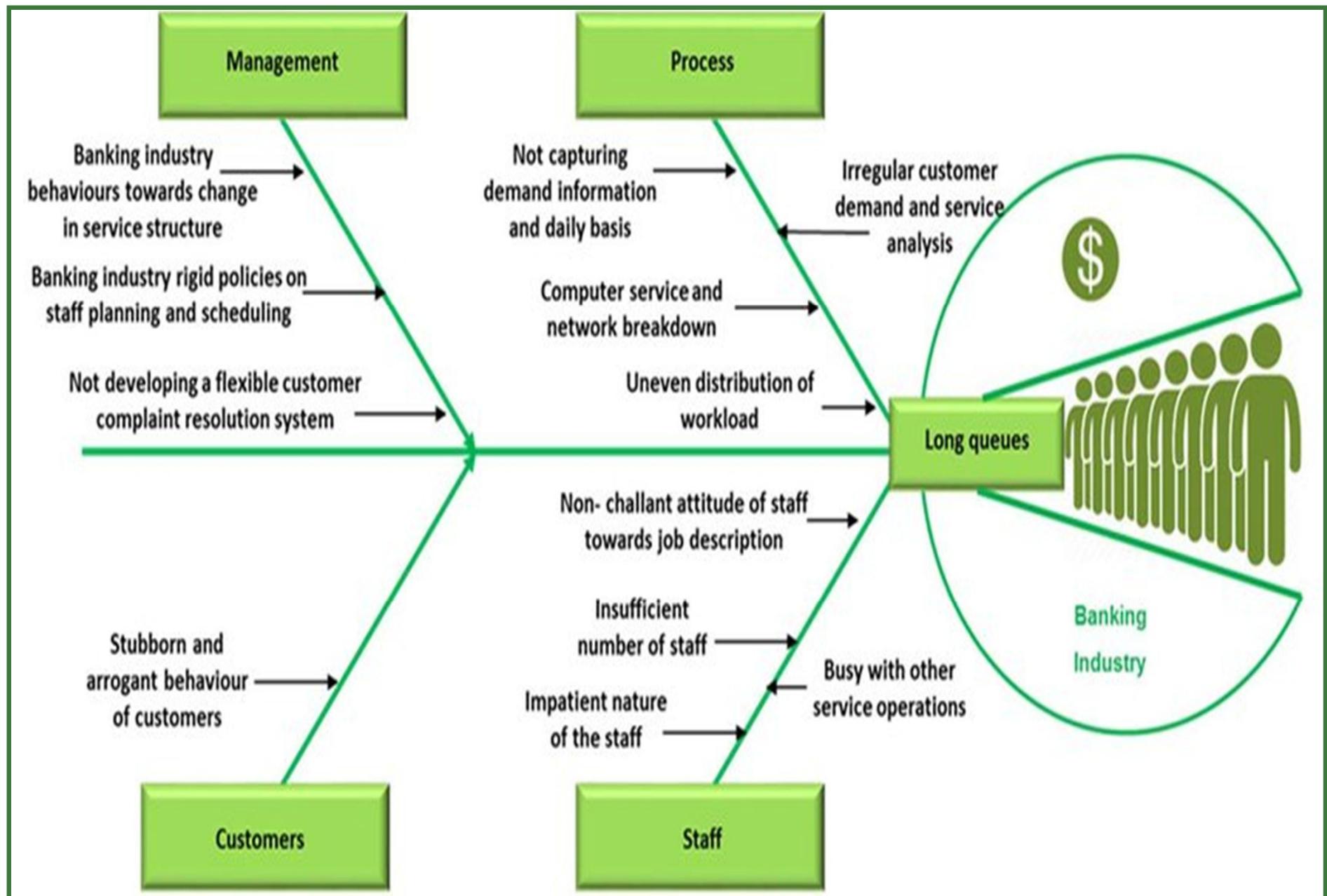
example

Fishbone Diagram



Fish bone diagram for “Delayed/Damage Delivery”

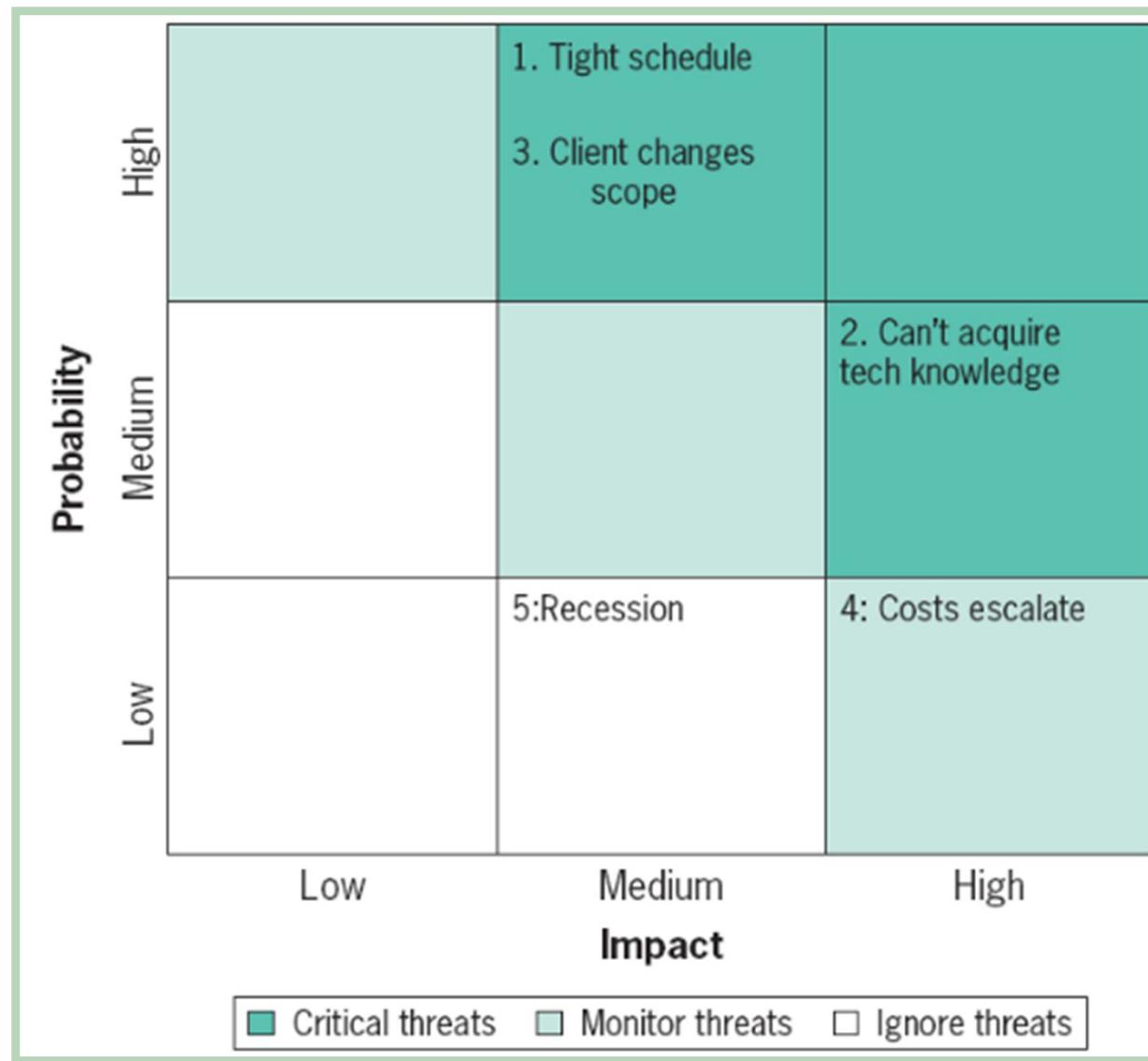




Qualitative Risk Analysis

- Purpose is to prioritize risks
- Identify the **probability of occurrence** of risk and a sense of the **impact** is also needed(e.g. low, medium, high)
- Each objective should be scaled and weighted
- Construct a risk matrix
- Same approach can be used for opportunities

Risk Matrix



		CONSEQUENCES				
		Negligible	Minor	Moderate	Significant	Severe
PROBABILITY	Almost Certain (81%-100%)	Low Risk	Moderate Risk	High Risk	Extreme Risk	Extreme Risk
	Likely (61%-80%)	Minimum Risk	Low Risk	Moderate Risk	High Risk	Extreme Risk
	Moderate (41%-60%)	Minimum Risk	Low Risk	Moderate Risk	High Risk	High Risk
	Unlikely (21%-40%)	Minimum Risk	Low Risk	Low Risk	Moderate Risk	High Risk
	Rare (1%-20%)	Minimum Risk	Minimum Risk	Low Risk	Moderate Risk	High Risk

	Consequence					
Likelihood	Insignificant	Minor	Moderate	Major	Critical	
Rare	LOW Accept the risk Routine management	LOW Accept the risk Routine management	LOW Accept the risk Routine management	MEDIUM Specific responsibility and treatment	HIGH Quarterly senior management review	
Unlikely	LOW Accept the risk Routine management	LOW Accept the risk Routine management	MEDIUM Specific responsibility and treatment	MEDIUM Specific responsibility and treatment	HIGH Quarterly senior management review	
Possible	LOW Accept the risk Routine management	MEDIUM Specific responsibility and treatment	MEDIUM Specific responsibility and treatment	HIGH Quarterly senior management review	HIGH Quarterly senior management review	
Likely	MEDIUM Specific responsibility and treatment	MEDIUM Specific responsibility and treatment	HIGH Quarterly senior management review	HIGH Quarterly senior management review	EXTREME Monthly senior management review	
Almost certain	MEDIUM Specific responsibility and treatment	MEDIUM Specific responsibility and treatment	HIGH Quarterly senior management review	EXTREME Monthly senior management review	EXTREME Monthly senior management review	

Likelihood scale	Criteria	Description
Rare	0 - 5%	Extremely unlikely or virtually impossible
Unlikely	6 - 25%	Unlikely to occur
Possible	26 - 50%	Fairly likely to occur
Likely	51 - 75%	More likely to occur
Almost certain	>75%	Almost certain will occur

Quantitative Risk Analysis

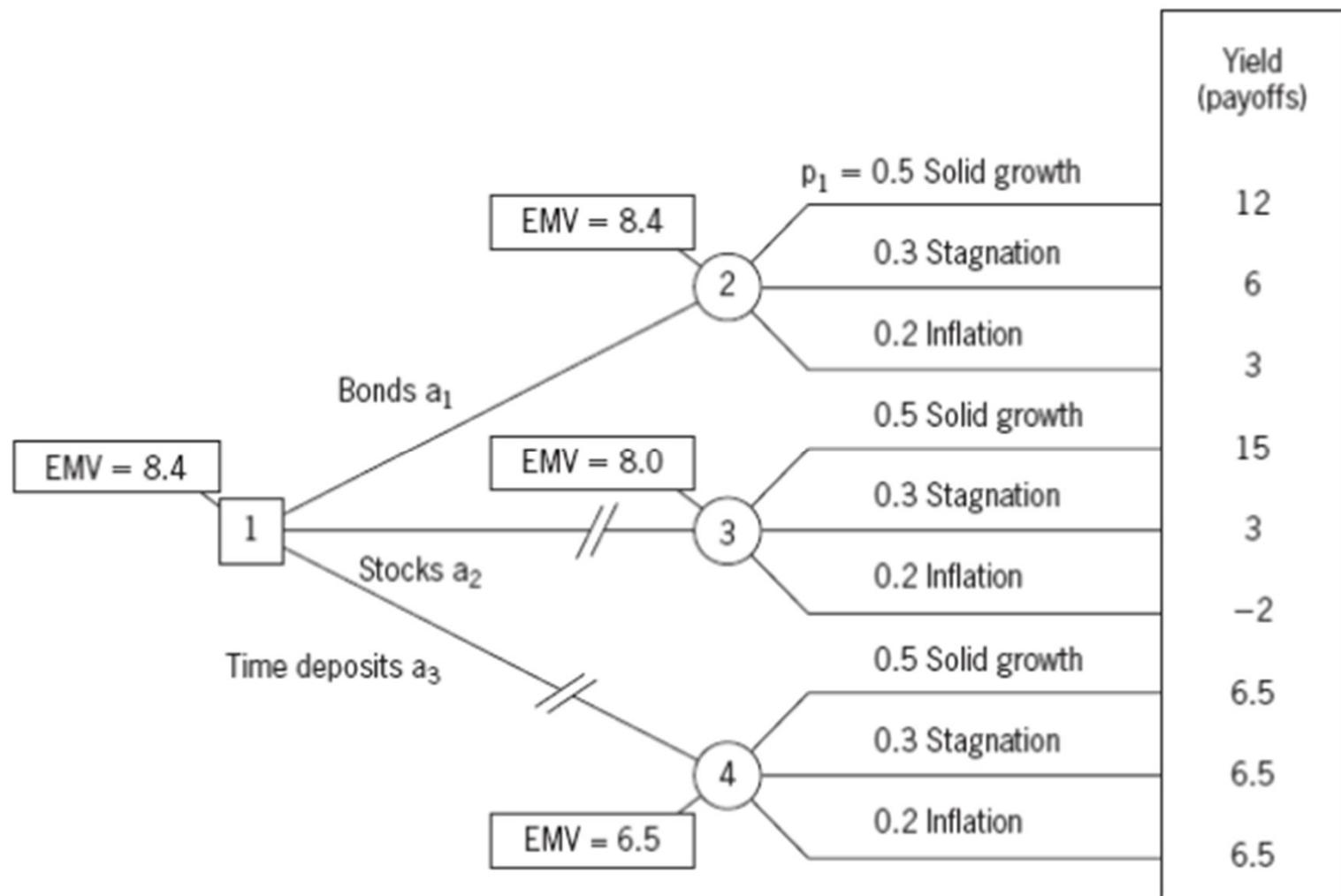
1. List ways a project can fail
2. Evaluate severity (**S**) by “*Failure mode and effect analysis*”(FMEA) **FORM 1-10 POINTS**
3. Estimate likelihood(**L**) from 1 -10 points
4. Estimate the inability to detect (**D**)
5. Find the **risk priority number** (RPN)
$$(\text{RPN} = S \times L \times D)$$
6. Consider ways to reduce the S, L, and D for each cause of failure

A FMEA Example

<i>Threat</i>	<i>Severity, S</i>	<i>Likelihood, L</i>	<i>Ability to Detect, D</i>	<i>RPN</i>
1. Tight schedule	6	7.5	2	90
2. Can't acquire tech knowledge	8.5	5	4	170
3. Client changes scope	4	8	5	160
4. Costs escalate	3	2	6	36
5. Recession	4	2.5	7	70

Highest is 170 i.e. “cant acquire tech knowledge”,
Now try to reduce this risk by taking appropriate action

Decision Tree Analysis



Risk Response Planning

Threats

- Avoid
- Transfer
- Mitigate
- Accept

Opportunities

- Exploit
- Share
- Enhance
- Accept

The Risk Management Register

- Environments that may impact projects
- Assumptions made
- Risks identified
- List of *categories* and *key words*
- Estimates on risk, states of project's environment, or on project assumptions
- Minutes
- Actual outcomes

Budgeting: Estimating Costs and Risks

Budgeting

- A plan for the costs of project resources
- A budget implies constraints
- Thus, it implies that managers will not get everything they want or need

Budgeting Continued

- The budget for an activity also implies management support for that activity
- Higher the budget, relative to cost, higher the managerial support
- The budget is also a control mechanism
 - Many organizations have controls in place that prohibit exceeding the budget
 - Comparisons are against the budget

Estimating Project Budgets

- On most projects
 - Material + Labor + Equipment + Capital + Overhead + Profits = Bid
- In other words
 - Resources + Profits = Bid
- So we are left with the task of forecasting resources

Estimating Project Budgets Continued

- Like any forecast, this includes some uncertainty
- There is uncertainty regarding usage and price
 - Especially true for material and labor
- The more standardized the project and components, the lower the uncertainty
- The more experienced the cost estimator, the lower the uncertainty

Rules of Thumb

- Some estimates are prepared by rules of thumb
 - Construction cost by square feet
 - Printing cost by number of pages
 - Lawn care cost by square feet of lawn
- These rules of thumb may be adjusted for special conditions
- However, this is still easier than starting the estimate from scratch

Estimating Budgets is Difficult

- There may not be as much historical data or none at all
- Even with similar projects, there may be significant differences
- Many people have input to the budget

Estimating Budgets is Difficult Continued

- Multiple people have some control over the budget
- There is more “flexibility” regarding the estimates of inputs (material and labor)
- The accounting system may not be set up to track project data
- Usage of labor and material is very lumpy over time

Types of Budgeting

- Top-down
- Bottom-up
- Negotiated

Top-Down Budgeting

- Top managers estimate/decide on the overall budget for the project
- These trickle down through the organization where the estimates are broken down into greater detail at each lower level
- The process continues to the bottom level

Advantages

- Overall project budgets can be set/controlled very accurately
 - A few elements may have significant error
- Management has more control over budgets
- Small tasks need not be identified individually

Disadvantages

- More difficult to get buy in
- Leads to low level competition for larger shares of budget

Bottom-Up Budgeting

- Project is broken down into work packages
- Low level managers price out each work package
- Overhead and profits are added to develop the budget

Advantages

- Greater buy in by low level managers
- More likely to catch unusual expenses

Disadvantages

- People tend to overstate their budget requirements
- Management tends to cut the budget

Work Element Costing

- Labor rates include overhead and personal time
- Direct costs usually do not include overhead
- General and administrative (G&A) charge

An Iterative Budgeting Process— Negotiation-in-Action

- Most projects use some combination of top-down and bottom-up budgeting
- Both are prepared and compared
- Any differences are negotiated

Category Budgeting Versus Program/Activity Budgeting

- Organizations are used to budgeting (and collecting data) by activity
- These activities correspond to “line items” in the budget
 - Examples include phone, utilities, direct labor,...
- Projects need to accumulate data and control expenses differently
- This resulted in program budgeting

Typical Monthly Budget

	Current			
	Actual	Budget	Variance	Pct.
<i>Corporate—Income Statement</i>				
Revenue				
8430 Management fees				
8491 Prtnsp reimb—property mgmt	7,410.00	6,222.00	1,188.00	119.0
8492 Prtnsp reimb—owner acquisition	.00	3,750.00	3,750.00-	.0
8493 Prtnsp reimb—rehab	.00	.00	.00	.0
8494 Other income	.00	.00	.00	.0
8495 Reimbursements—others	.00	.00	.00	.0
Total revenue	7,410.00	9,972.00	2,562.00-	74.3
<i>Operating expenses</i>				
Payroll & P/R benefits				
8511 Salaries	29,425.75	34,583.00	5,157.25	85.0
8512 Payroll taxes	1,789.88	3,458.00	1,668.12	51.7
8513 Group ins & med reimb	1,407.45	1,040.00	387.45-	135.3
8515 Workmen's compensation	43.04	43.00	.04-	100.0
8516 Staff apartments	.00	.00	.00	.0
8517 Bonus	.00	.00	.00	.0
Total payroll & P/R benefits	32,668.12	39,124.00	6,457.88	83.5
Travel & entertainment expenses				
8512 Travel	456.65	300.00	156.65-	152.2
8522 Promotion, entertainment & gift	69.52	500.00	430.48	13.9
8523 Auto	1,295.90	1,729.00	433.10	75.0
Total travel & entertainment exp	1,822.07	2,529.00	706.93	72.1
Professional fees				
8531 Legal fees	419.00	50.00	369.00-	838.0
8532 Accounting fees	289.00	.00	289.00-	.0
8534 Temporary help	234.58	200.00	34.58-	117.2

Project Budget by Task & Month

Task	Estimate	Monthly Budget (£)								
		1	2	3	4	5	6	7	8	
A	7000	5600	1400							
B	9000		3857	5143						
C	10000		3750	5000	1250					
D	6000		3600	2400						
E	12000				4800	4800	2400			
F	3000				3000					
G	9000			2571	5143	1286				
H	5000					3750	1250			
I	8000						2667	5333		
J	6000								6000	
		75000	5600	12607	15114	14193	9836	6317	5333	6000
										6000

Source: Harrison, 1983.

Improving The Process of Cost Estimation

- Inputs from a lot of areas are required to estimate a project
- May have a professional cost estimator to do the job
- Project manager will work closely with cost estimator when planning a project
- We are primarily interested in estimating direct costs
- Indirect costs are not a major concern

Problems

- Even with careful planning, estimates are wrong
- Most firms add 5-10 percent for contingencies

Learning Curves

- Human performance usually improves when a task is repeated
- This happens by a fixed percent each time the production doubles
- Percentage is called the learning rate

Learning Curve Calculations

$$T_n = T_1 n^r$$

$$r = \frac{\log(\text{rate})}{\log 2}$$

$$\text{Total time} = T_1 \sum_{n=1}^N n^r$$

T_n = Time for n th unit
 T_1 = Time for first unit
 n = Number of units
 r = log decimal
rate/log 2

Other Factors

- Escalation
- Waste
- Bad Luck

Making Better Estimates

- Projects are known for being over budget
 - It is unlikely that this is due to deliberate underestimating
- There are two types of errors
 - Random
 - Bias
- There is nothing we can do about random errors
 - Tend to cancel each other
- Eliminate systematic errors

Risk Estimation

- Duration of project activities varies
- Amounts of various resources needed varies
- Value of accomplishing a project varies
- Can reduce but not eliminate ambiguity
- Want to describe uncertainties in a way that provides useful insight to their nature

Applying Risk Analysis

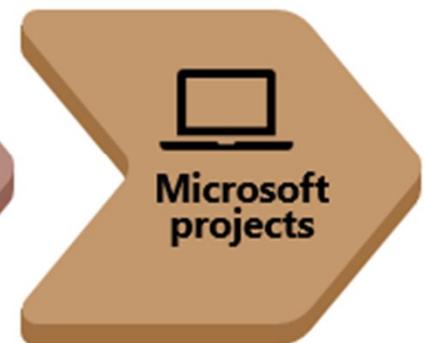
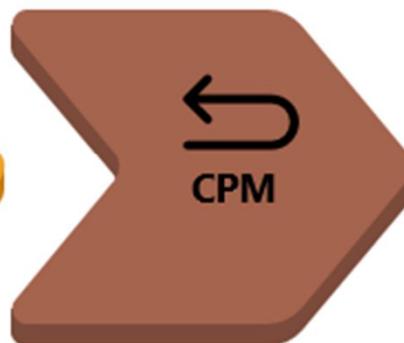
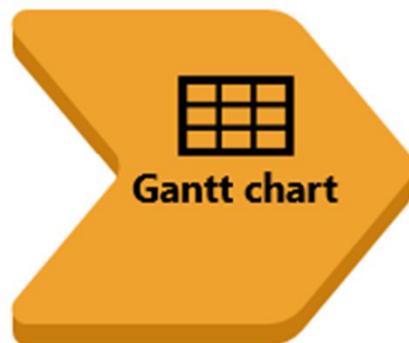
- Must make assumptions about probability distributions
 - Key parameters
 - Variables
- Estimate the risk profiles of the outcomes of the decision
 - Also know as *probability distributions*
- Simulation is often used

General Simulation Analysis

- Simulation combined with sensitivity analysis is useful for evaluating projects
- Would support project if NPV is positive and is the best use of funds
- Should avoid full-cost philosophy
 - Some overheads are not affected by changes
- Analysis gives a picture in terms of costs and times that will be affected
- Project is then reviewed using simulation

Scheduling

Project Scheduling Techniques



Useful Abbreviations

- **CPM** - Critical Path Method
- **PERT** - Program Evaluation and Review Technique

Background

- Schedule is the conversion of a project action plan into an operating timetable
- Basis for monitoring a project
- One of the major project management tools
- Work changes daily, so a detailed plan is essential
- Not all project activities need to be scheduled at the same level of detail

Background Continued

- Most of the scheduling is at the WBS level, not the work package level
- Only the most critical work packages may be shown on the schedule
- Most of the scheduling is based on network drawings

Network Scheduling Advantage

- Consistent framework
- Shows interdependences
- Shows when resources are needed
- Ensures proper communication
- Determines expected completion date
- Identifies critical activities

Network Scheduling Advantage

Continued

- Shows which of the activities can be delayed
- Determines start dates
- Shows which task must be coordinated
- Shows which task can be run parallel
- Relieves some conflict
- Allows probabilistic estimates

Network Scheduling Techniques: PERT (ADM) and CPM (PDM)

- PERT was developed for the Polaris missile/submarine project in 1958
- CPM developed by DuPont during the same time
- Initially, CPM and PERT were two different approaches
 - CPM used deterministic time estimates and allowed project crunching
 - PERT used probabilistic time estimates
- Microsoft Project (and others) have blended CPM and PERT into one approach

Terminology

- **Activity** - A specific task or set of tasks that are required by the project, use up resources, and take time to complete
- **Event** - The result of completing one or more activities
- **Network** - The combination of all activities and events that define a project
 - Drawn left-to-right
 - Connections represent predecessors

Terminology Continued

- **Path** - A series of connected activities
- **Critical** - An activity, event, or path which, if delayed, will delay the completion of the project
- **Critical Path** - The path through the project where, if any activity is delayed, the project is delayed
 - There is always a critical path
 - There can be more than one critical path

Terminology Continued

- **Sequential Activities** - One activity must be completed before the next one can begin
- **Parallel Activities** - The activities can take place at the same time
- **Immediate Predecessor** - That activity that must be completed just before a particular activity can begin

Terminology Continued

- **Activity on Arrow** - Arrows represent activities while nodes stand for events
- **Activity on Node** - Nodes stand for events and arrows show precedence

AON and AOA Format

Figure 8-2

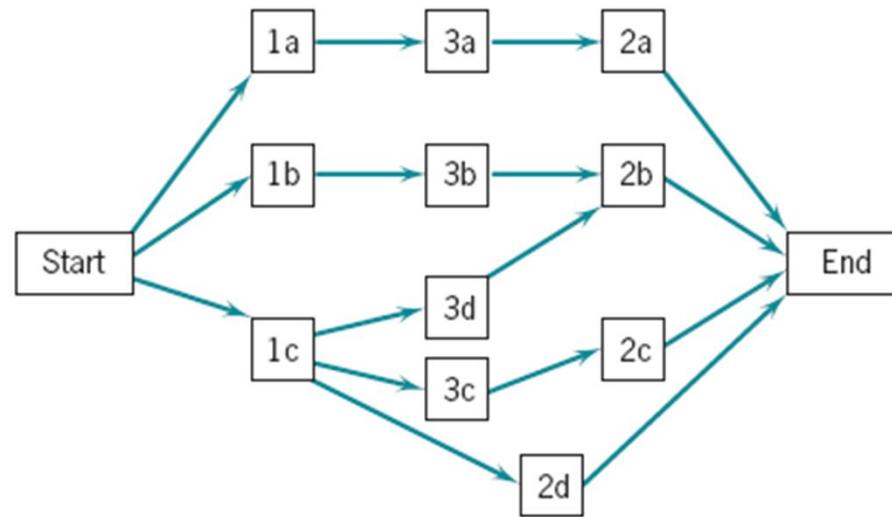
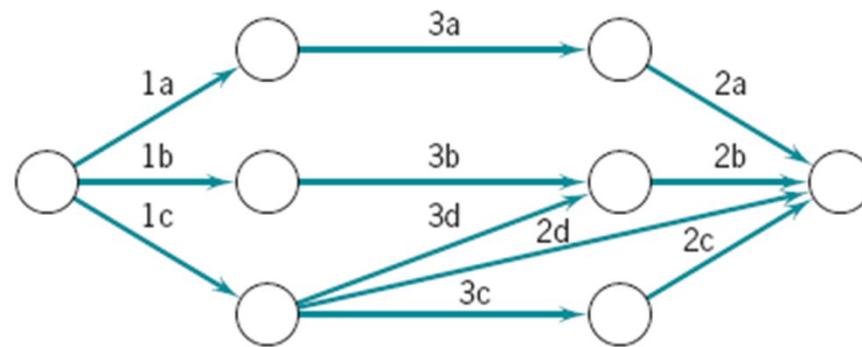


Figure 8-3



Constructing the Network

- Begin with START activity
- Add activities without precedences as nodes
 - There will always be one
 - May be more
- Add activities that have those activities as precedences
- Continue

Gantt (Bar) Charts

- Developed by Henry L. Gantt
- Shows planned and actual progress
- Easy-to-read method to know the current status

Advantages and Disadvantage

- Advantages
 - Easily understood
 - Provide a picture of the current state of a project
- Disadvantage
 - Difficult to follow complex projects

Microsoft Project Gantt Chart

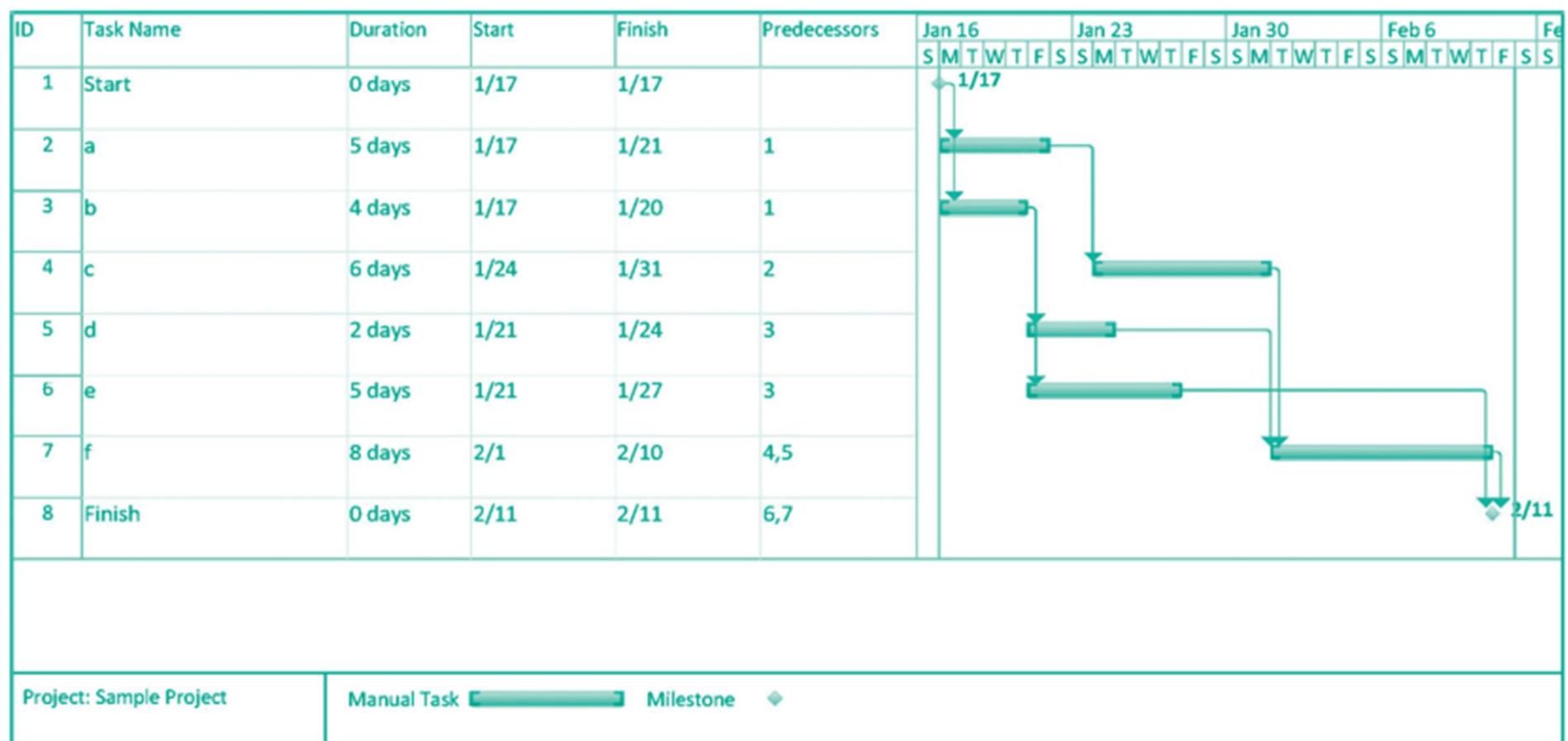


Figure 8-11

Microsoft Project AON Network

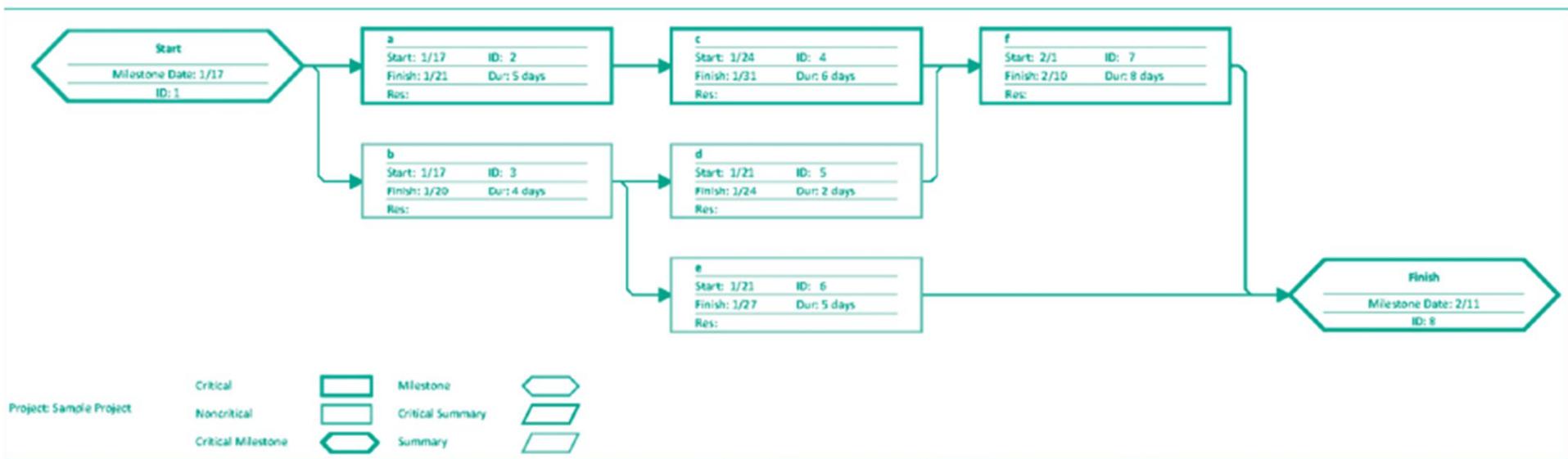


Figure 8-12

Solving the Network

<i>Activity</i>	<i>Optimistic Time</i>	<i>Most Likely Time</i>	<i>Pessimistic Time</i>	<i>Immediate Predecessor Activities</i>
a	10	22	22	—
b	20	20	20	—
c	4	10	16	—
d	2	14	32	a
e	8	8	20	b, c
f	8	14	20	b, c
g	4	4	4	b, c
h	2	12	16	c
i	6	16	38	g, h
j	2	8	14	d, e

The AON Network from the previous table

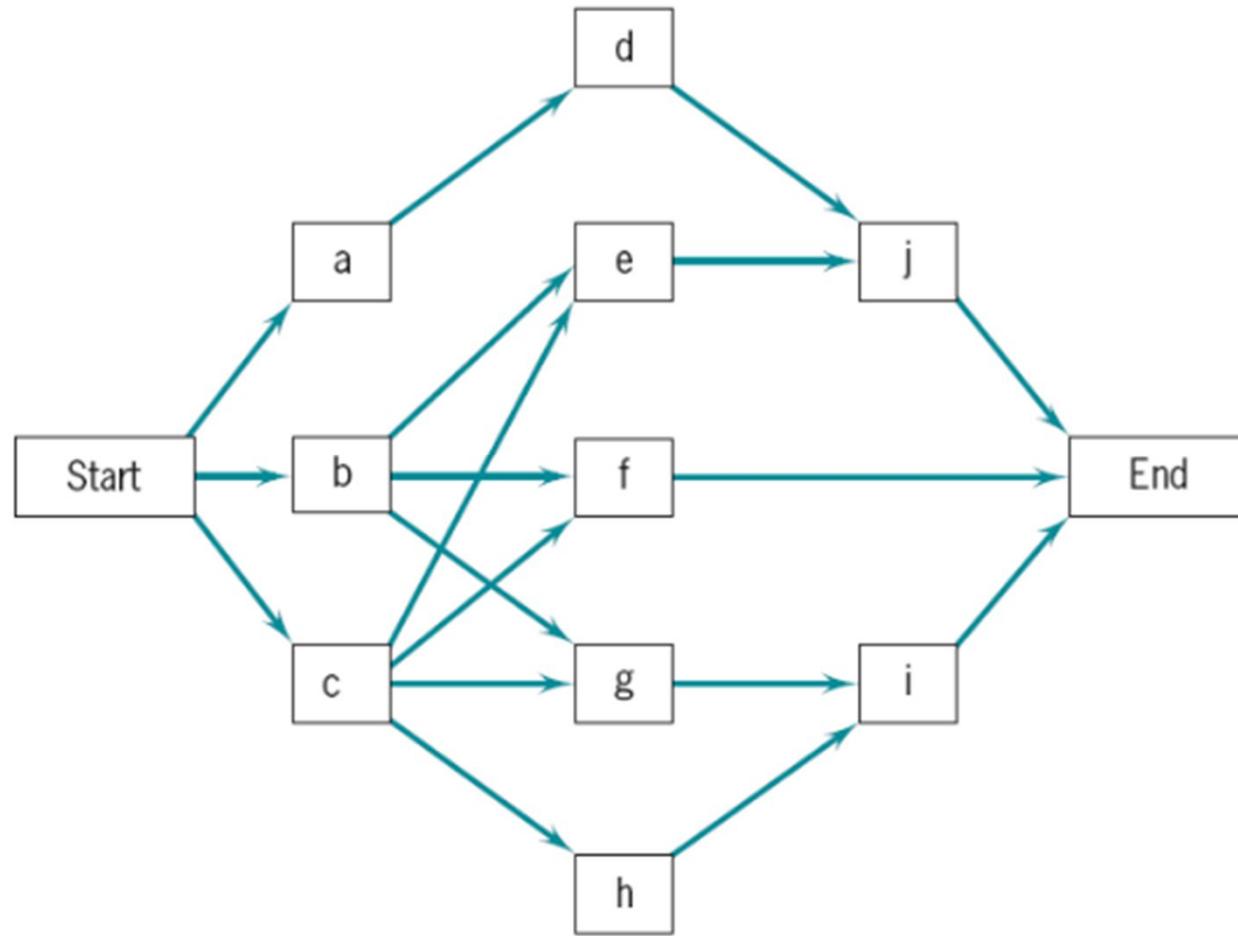


Figure 8-13

Calculating Activity Times

$$TE = \frac{(a + 4m + b)}{6}$$

$$\sigma^2 = \left(\frac{(b - a)}{6} \right)^2$$

$$\sigma = \sqrt{\sigma^2}$$

The Results

<i>Activity</i>	<i>Expected Time, TE</i>	<i>Variance, σ^2</i>	<i>Standard Deviation, σ</i>
a	20	4	2
b	20	0	0
c	10	4	2
d	15	25	5
e	10	4	2
f	14	4	2
g	4	0	0
h	11	5.4	2.32
i	18	28.4	5.33
j	8	4	2

Table 8-2

Critical Path and Time

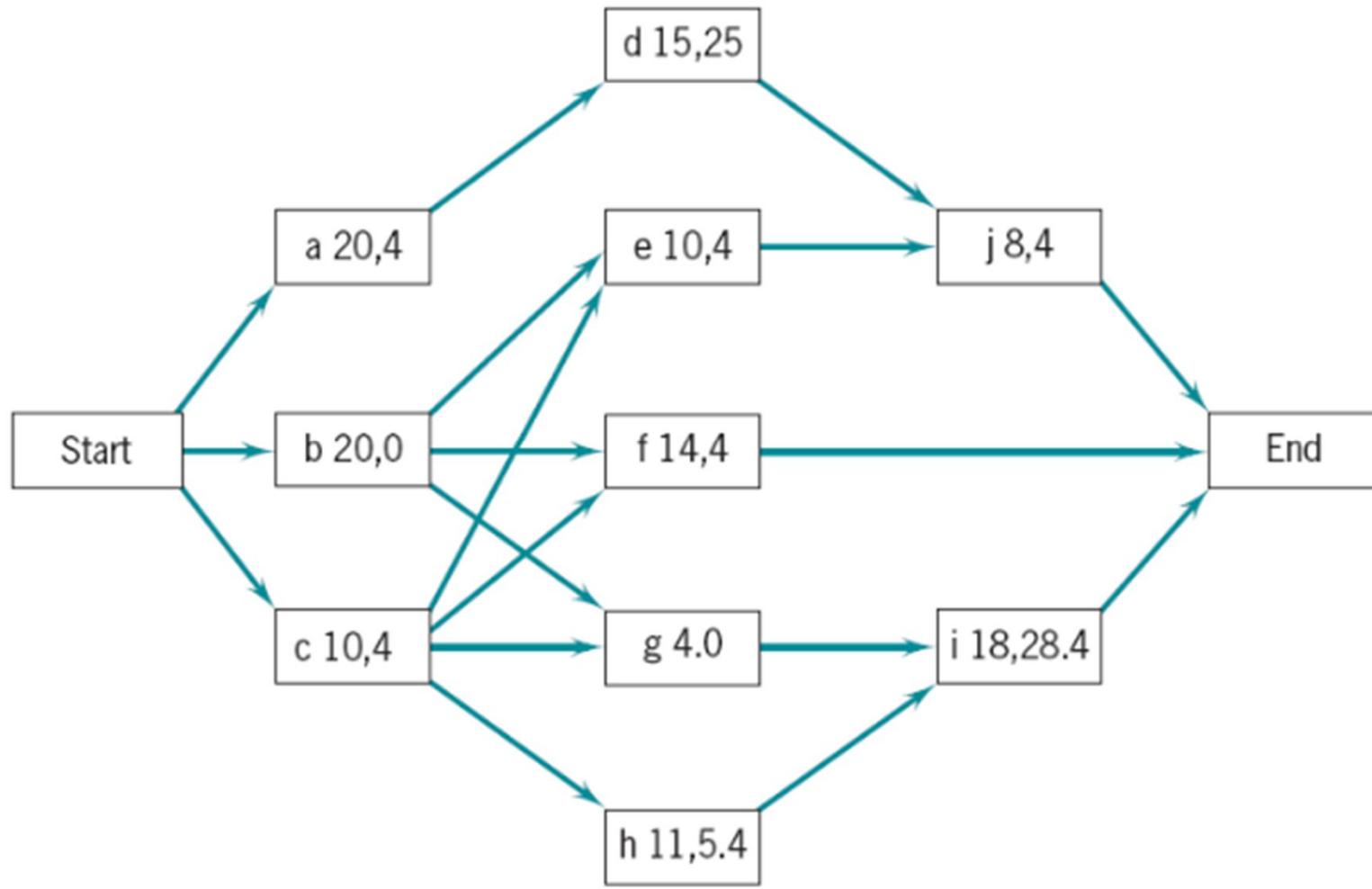


Figure 8-15

Critical Path and Time Continued

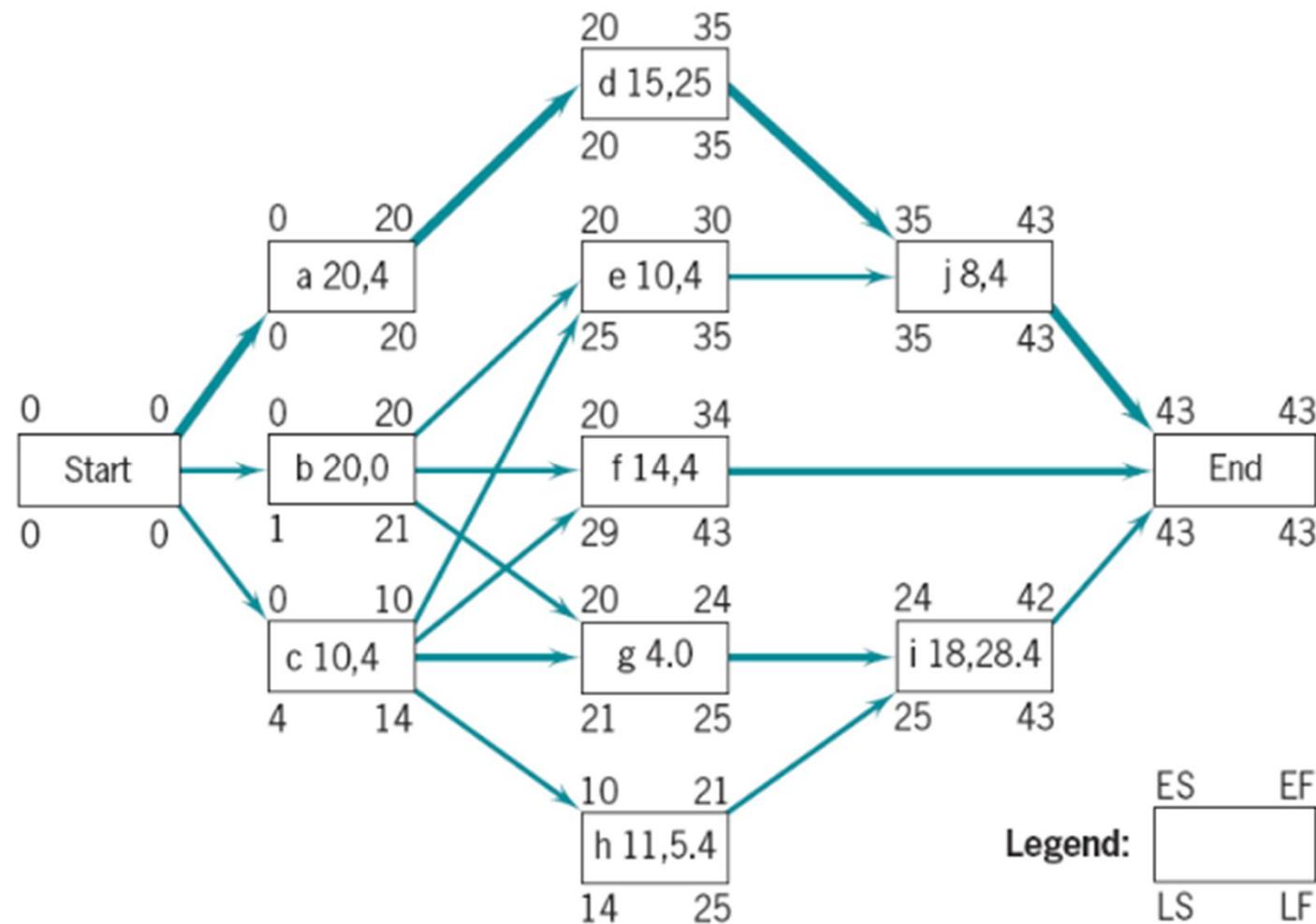


Figure 8-16

Slack

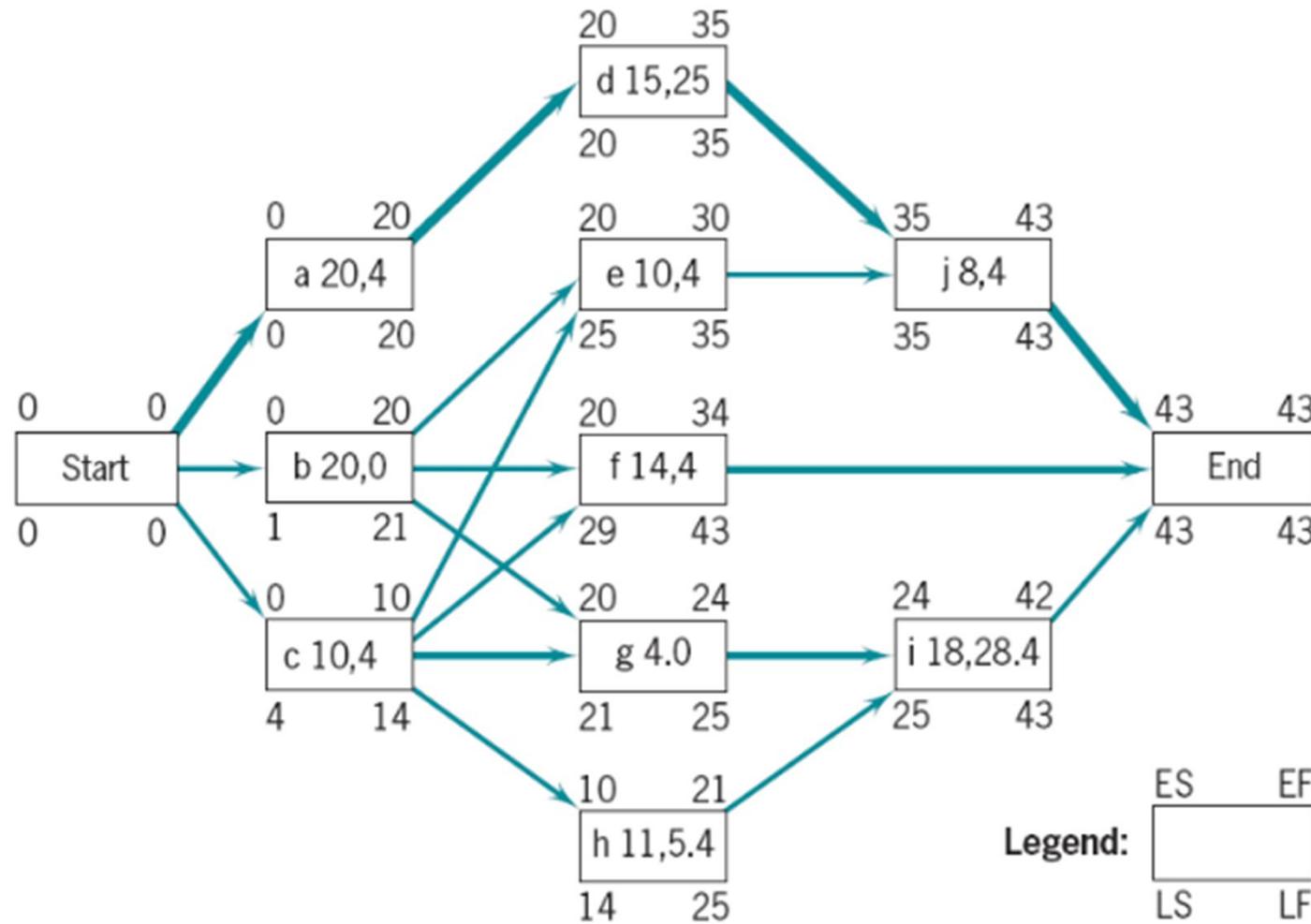


Figure 8-16

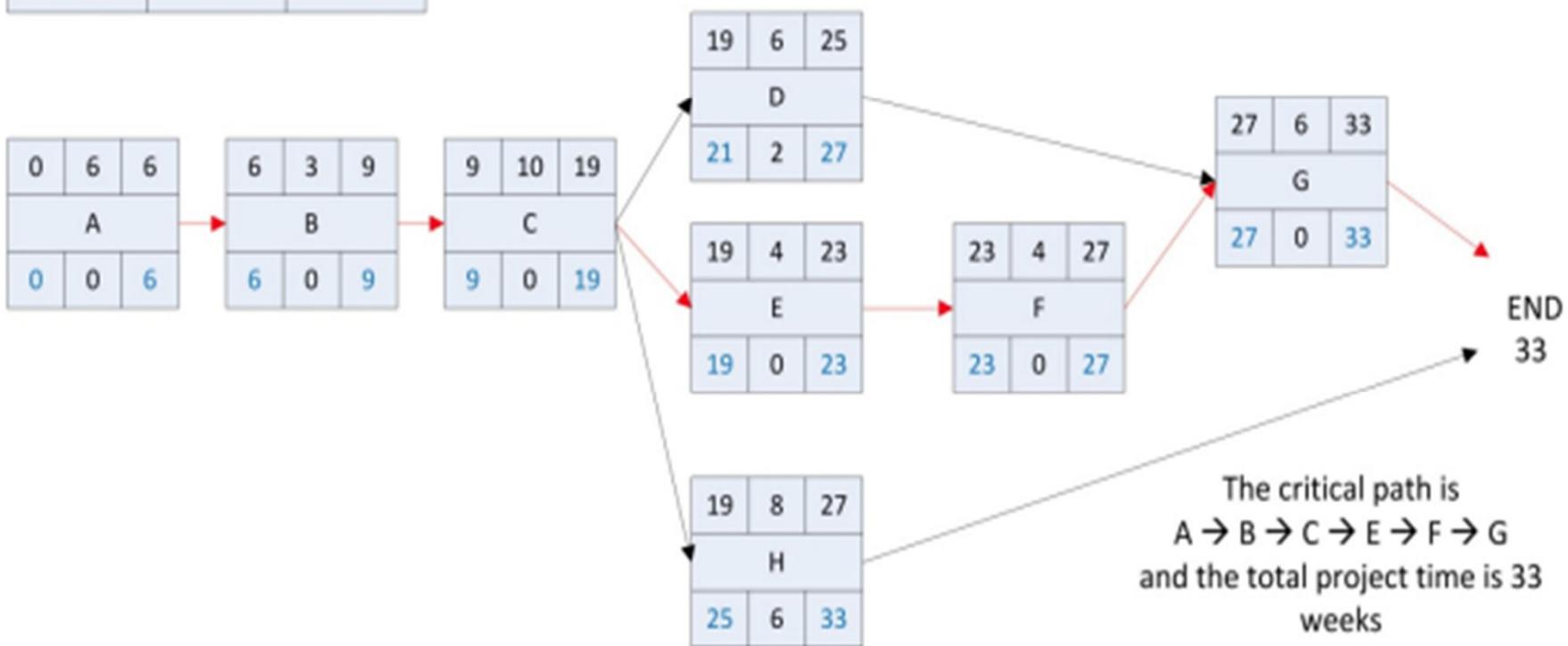
Slack Values

<i>Activity</i>	<i>LS</i>	<i>ES</i>	<i>Slack</i>
a	0	0	0
b	1	0	1
c	4	0	4
d	20	20	0
e	25	20	5
f	29	20	9
g	21	20	1
h	14	10	4
i	25	24	1
j	35	35	0

Precedence Diagramming

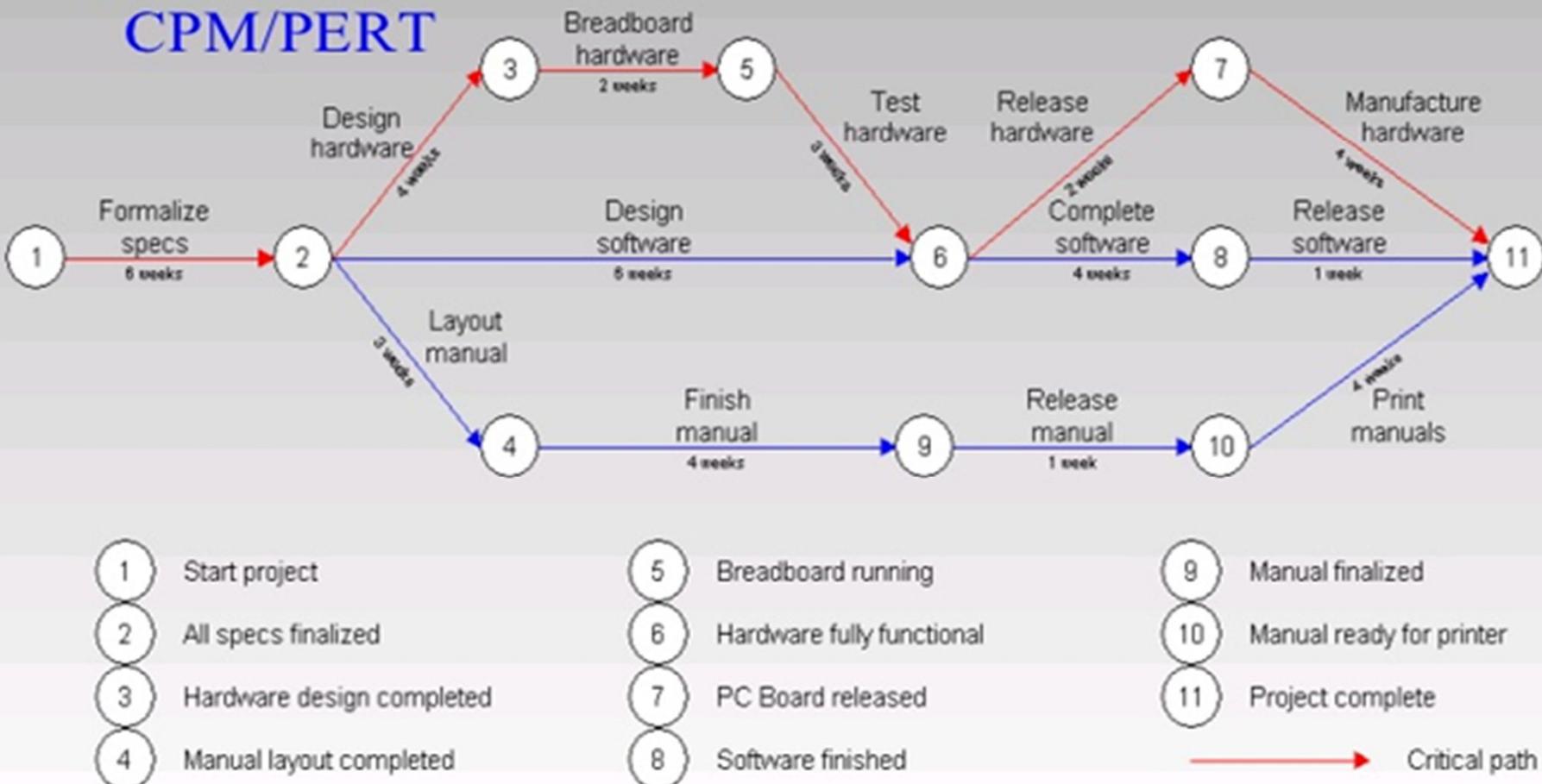
- Finish to start
- Start to start
- Finish to finish
- Start to finish

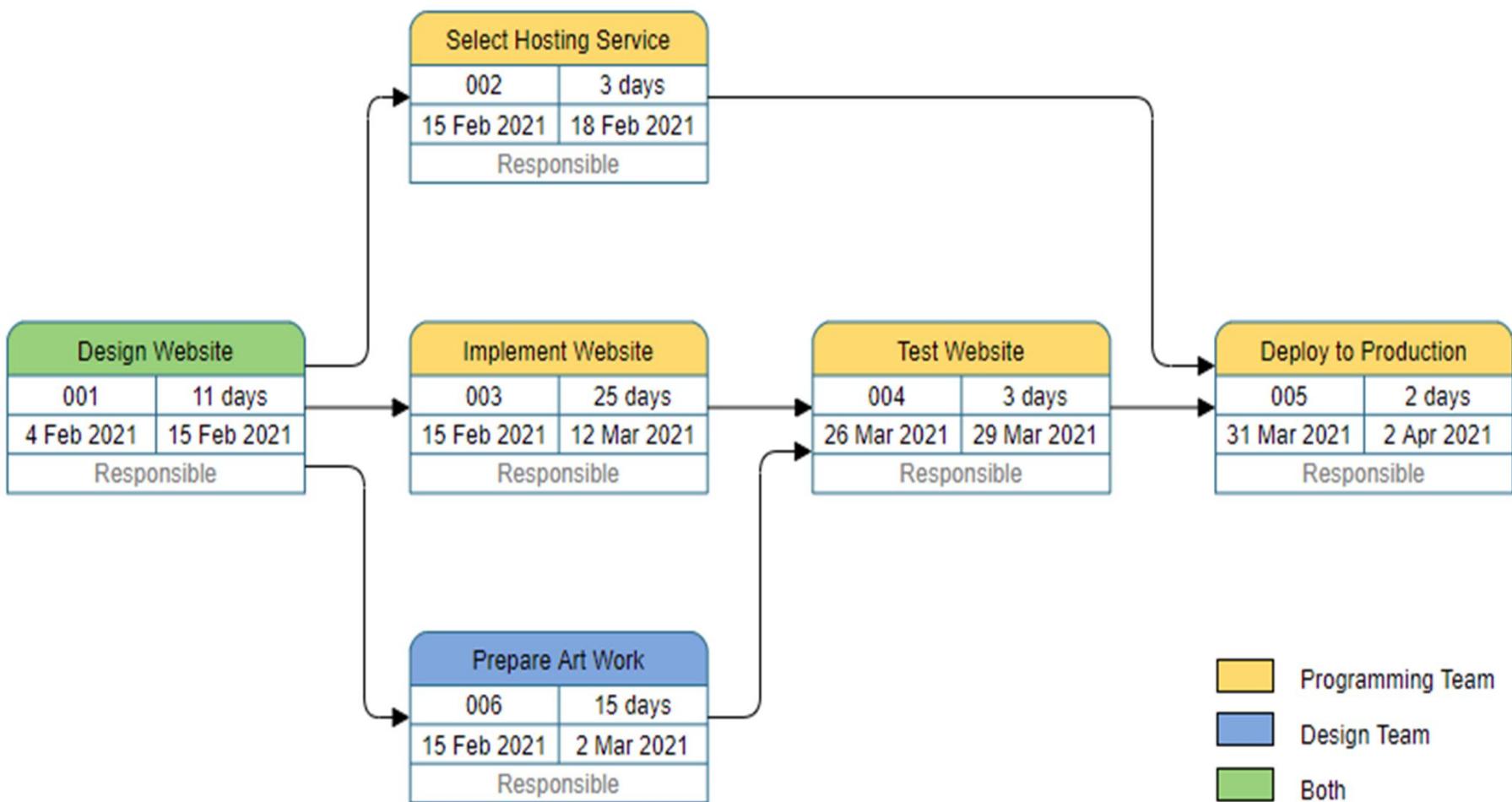
Early Start	Duration	Early Finish
Task Name		
Late Start	Slack	Late Finish



PERT/CPM Chart - PC Card

Project Management CPM/PERT





Precedence Diagramming Conventions

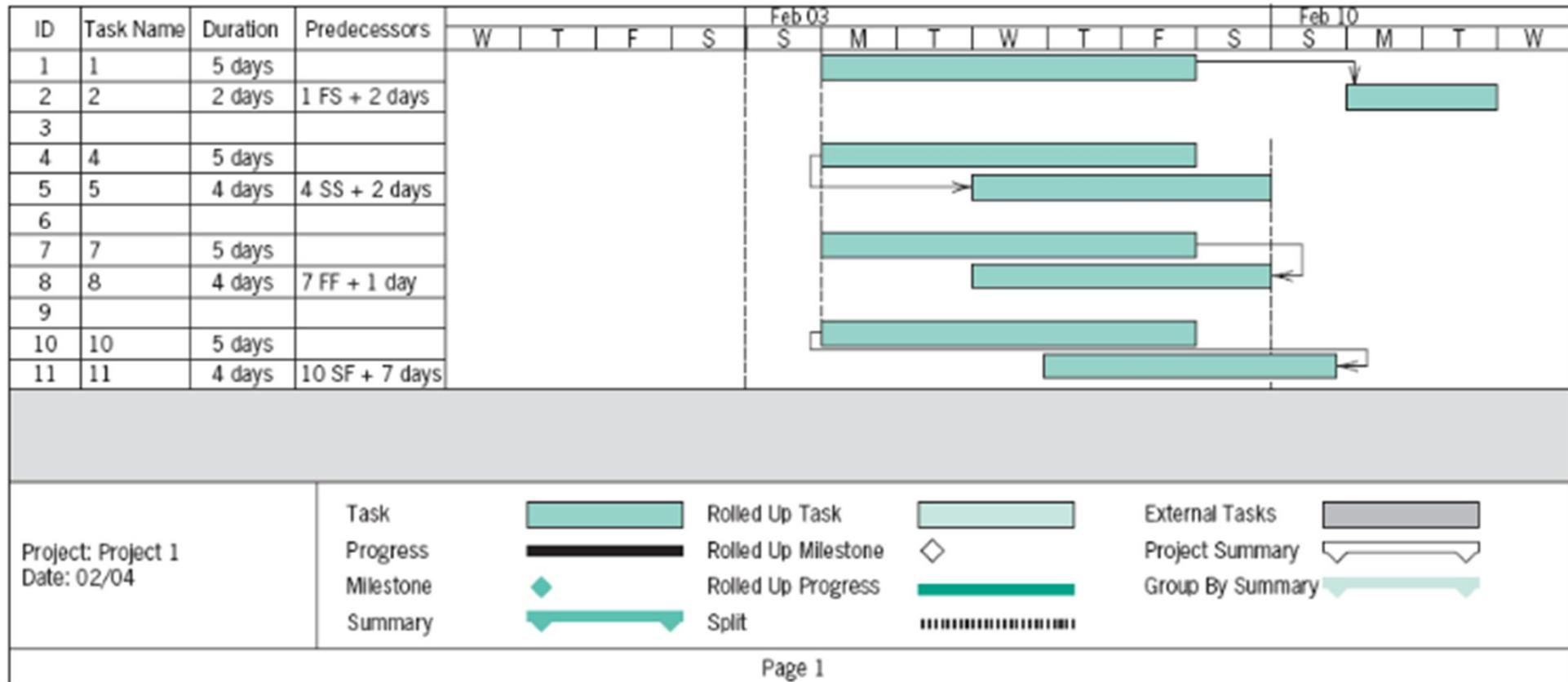


Figure 8-17

Microsoft Projects

<i>ID</i>	<i>Task Name</i>	<i>Predecessors</i>	<i>Duration</i>	<i>Optimistic Duration</i>	<i>Expected Duration</i>	<i>Pessimistic Duration</i>
1	Start		0 days	0 days	0 days	0 days
2	a	1	20 days	10 days	22 days	22 days
3	b	1	20 days	20 days	20 days	20 days
4	c	1	10 days	4 days	10 days	16 days
5	d	2	15 days	2 days	14 days	32 days
6	e	3, 4	10 days	8 days	8 days	20 days
7	f	4, 3	14 days	8 days	14 days	20 days
8	g	3, 4	4 days	4 days	4 days	4 days
9	h	4	11 days	2 days	12 days	16 days
10	i	9, 8	18 days	6 days	16 days	38 days
11	j	5, 6	8 days	2 days	8 days	14 days
12	Finish	10, 11, 7	0 days	0 days	0 days	0 days

Table 8-4

Gantt Chart

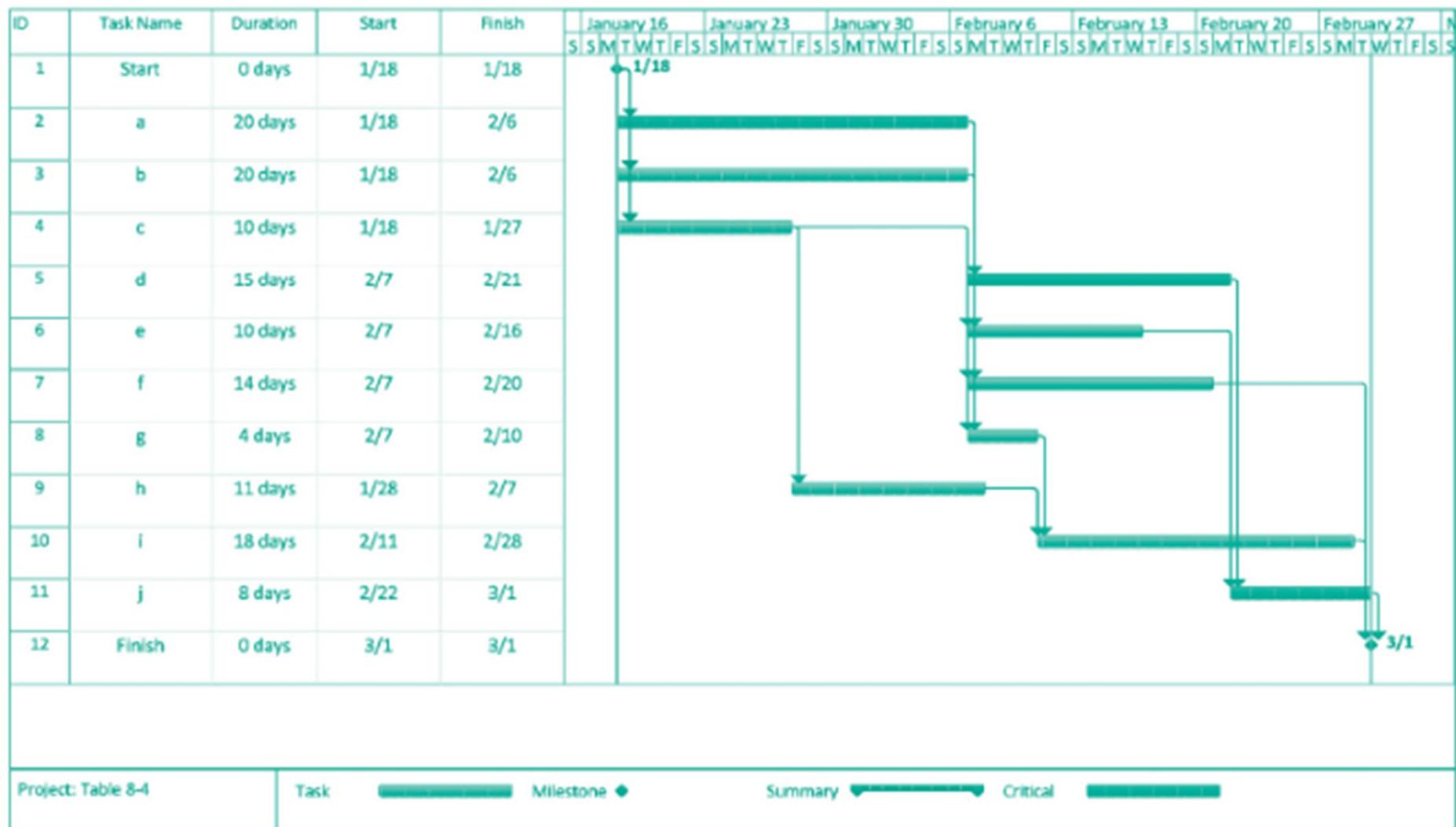


Figure 8-18

AON Network

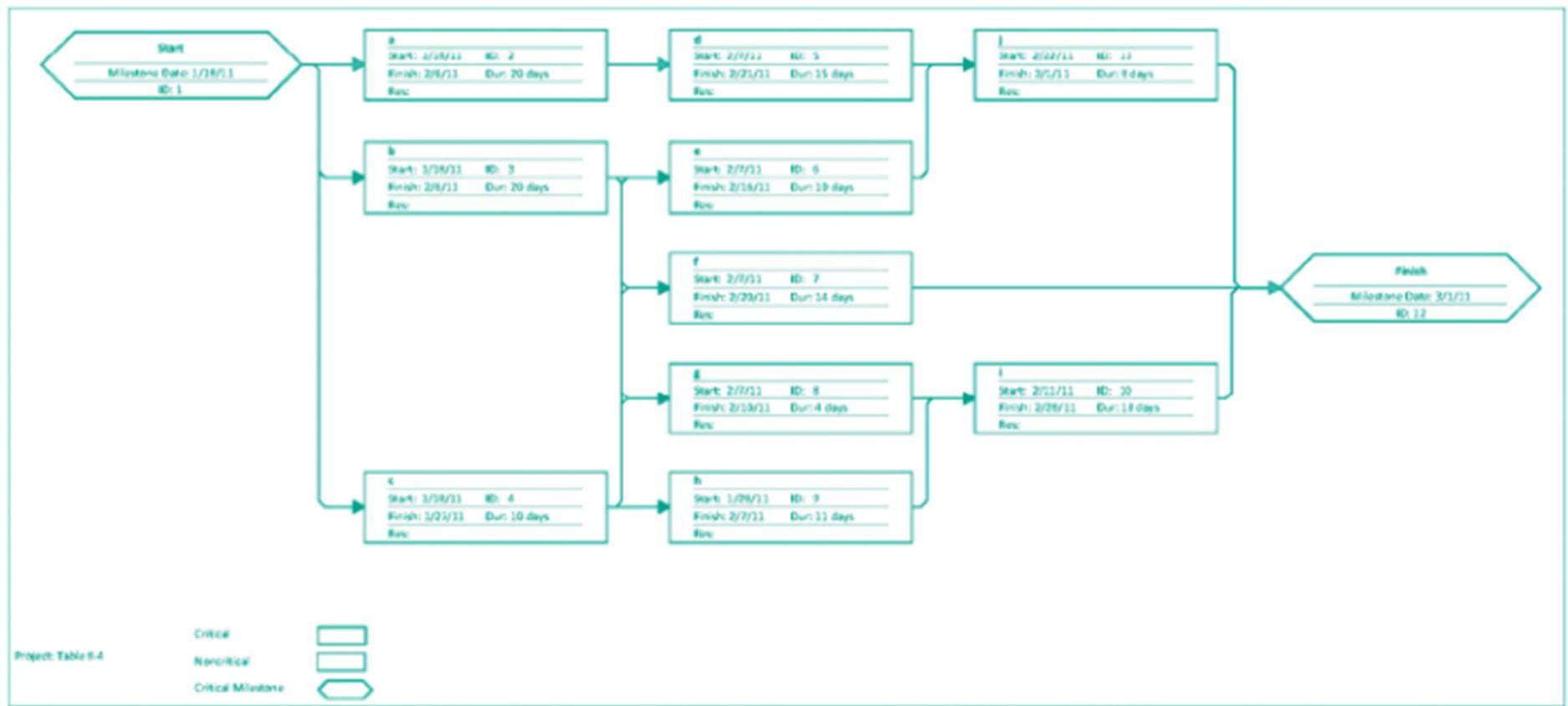


Figure 8-19

Microsoft Project Calendar

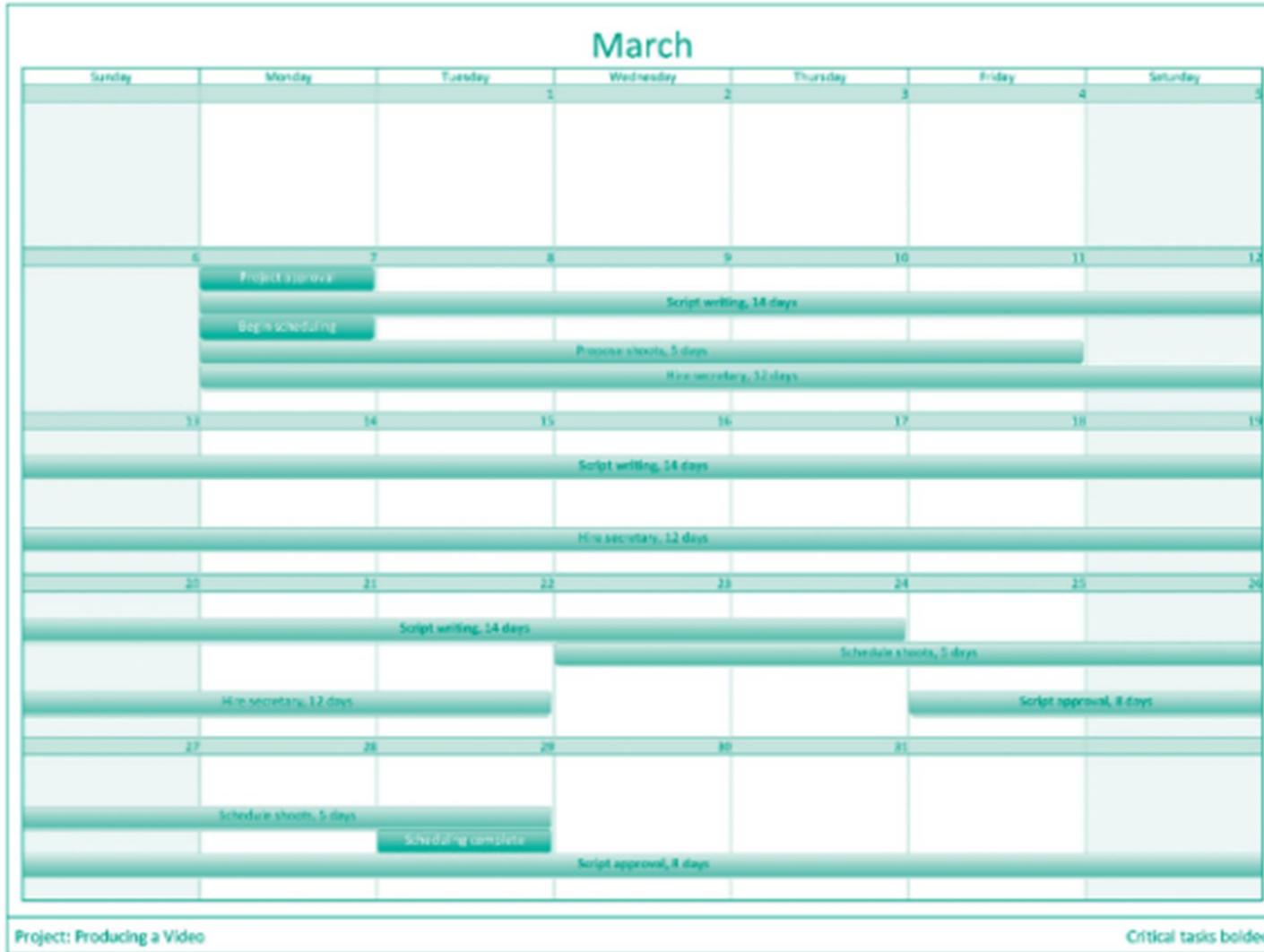


Figure 8-23

Uncertainty of Project Completion Time

- Assume activities are statistically independent
- Variance of a set of activities is the sum of the individual variances
- Interested in variances along the critical path

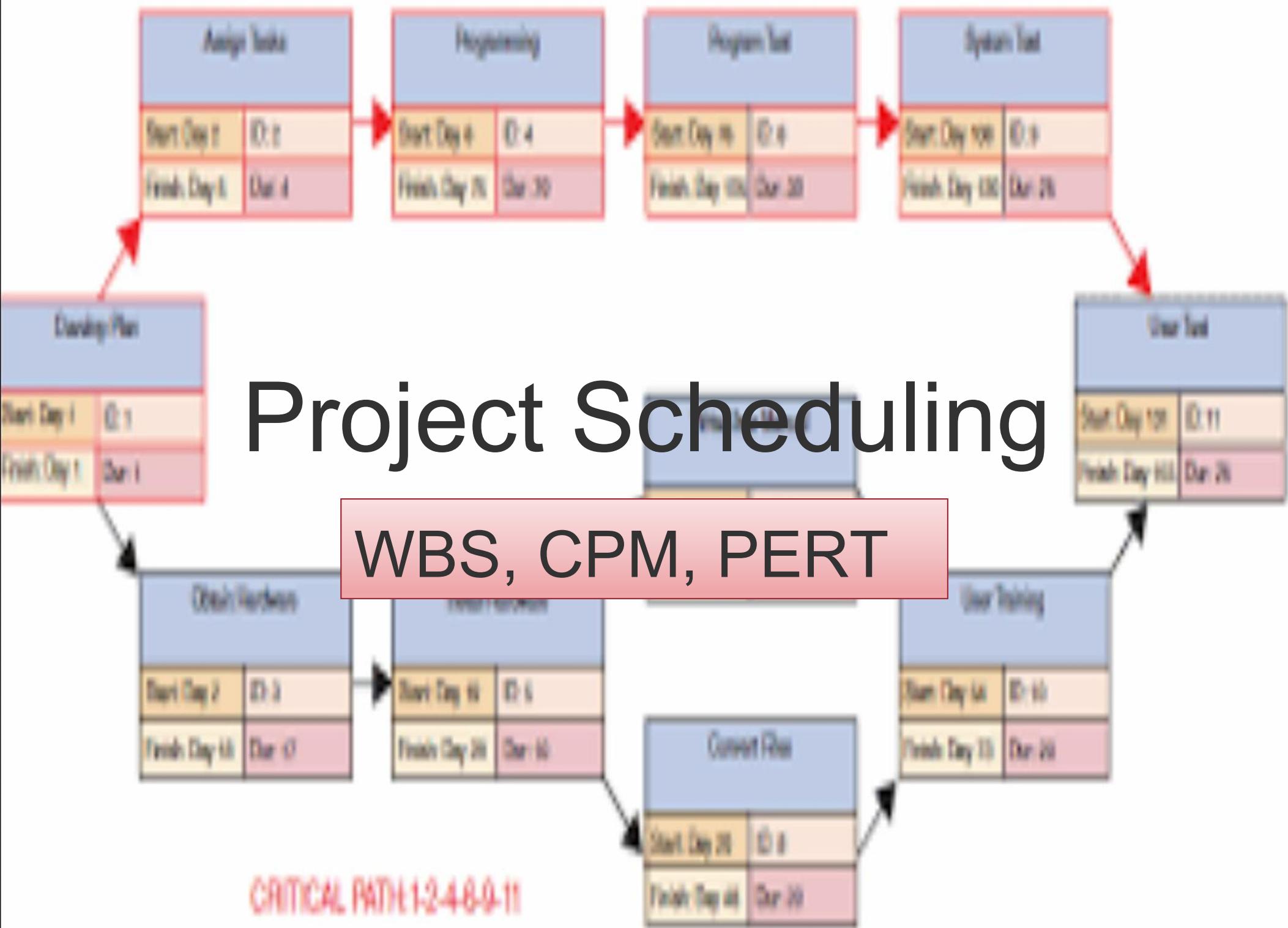
Example

$$Z = \frac{(D - \mu)}{\sqrt{\sigma^2}} = \frac{(50 - 43)}{\sqrt{33}} = \frac{7}{5.745} = 1.22$$

$$D = \mu + \sigma \cdot Z = 43 + 5.745(1.645) = 52.45$$

Toward Realistic Time Estimates

- Calculations are based on 1% chance of beating estimates
- Calculations can also be based on 5% or 10%
- Changing the percentage requires changing the formulae for variance
- When using 5%, the divisor changes to 3.29
- When using 10%, the divisor changes to 2.56



INTRODUCTION

- Schedule converts action plan into operating time table
- Basis for monitoring and controlling project
- Sometimes customer specified/approved requirement
- Based on Work Breakdown Structure (WBS)

Sequence the Work Activities

▲ Milestone Chart

▲ Gantt chart

▲ Network Techniques

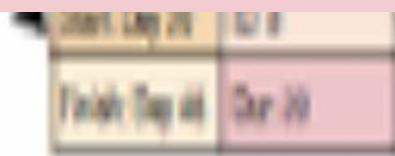
- CPM (Critical Path Method)
- PERT (Program Evaluation and Review Technique)

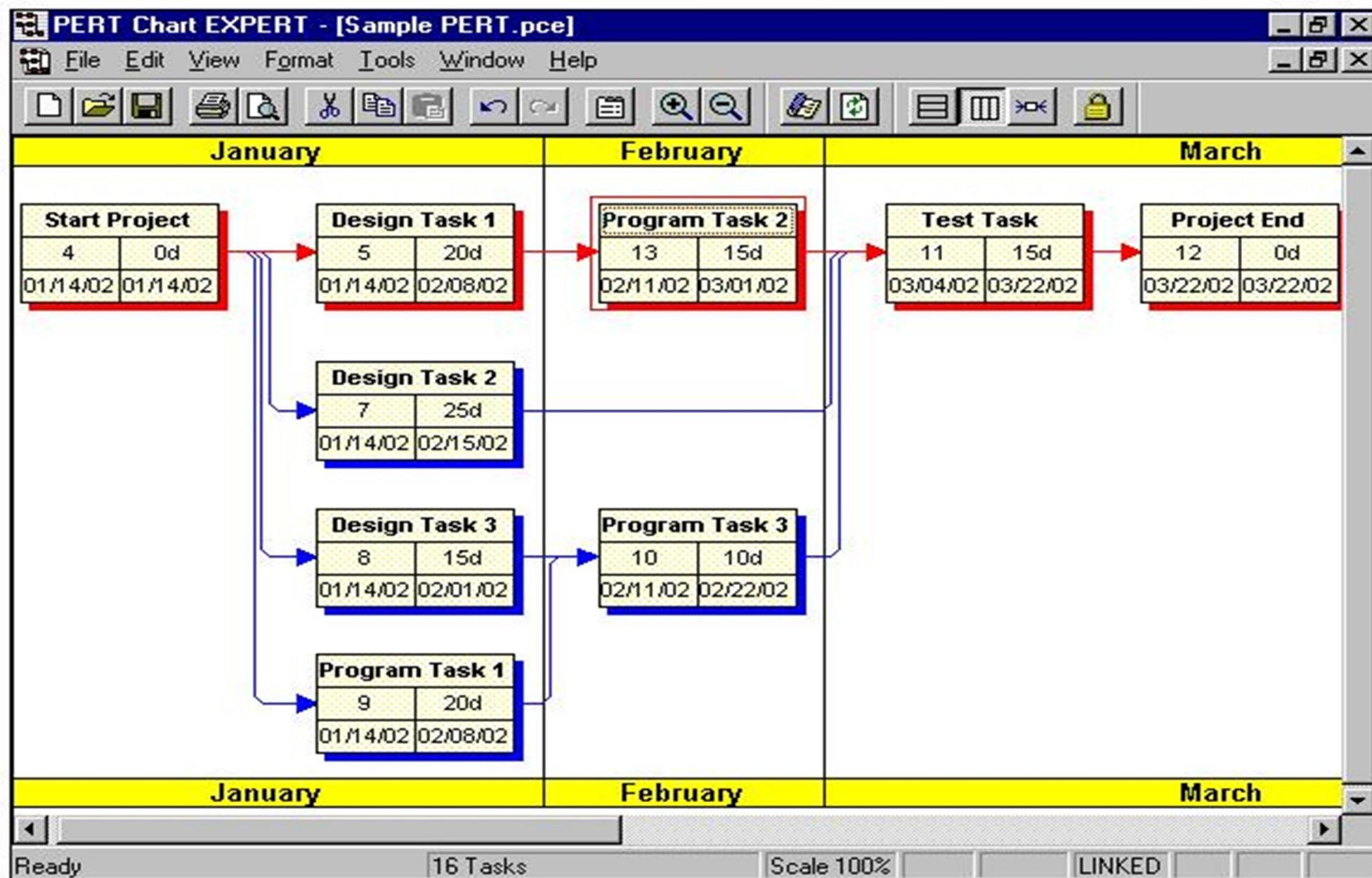
CRITICAL PATH 1-2-4-6-9-11

Network Techniques

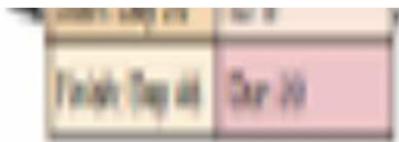
- A *precedence network* diagram is a graphic model portraying the sequential relationship between key events in a project.
- Initial development of the network requires that the project be defined and thought out.
- The network diagram clearly and precisely communicates the plan of action to the project team and the client.

CRITICAL PATH 1-2-4-6-9-11





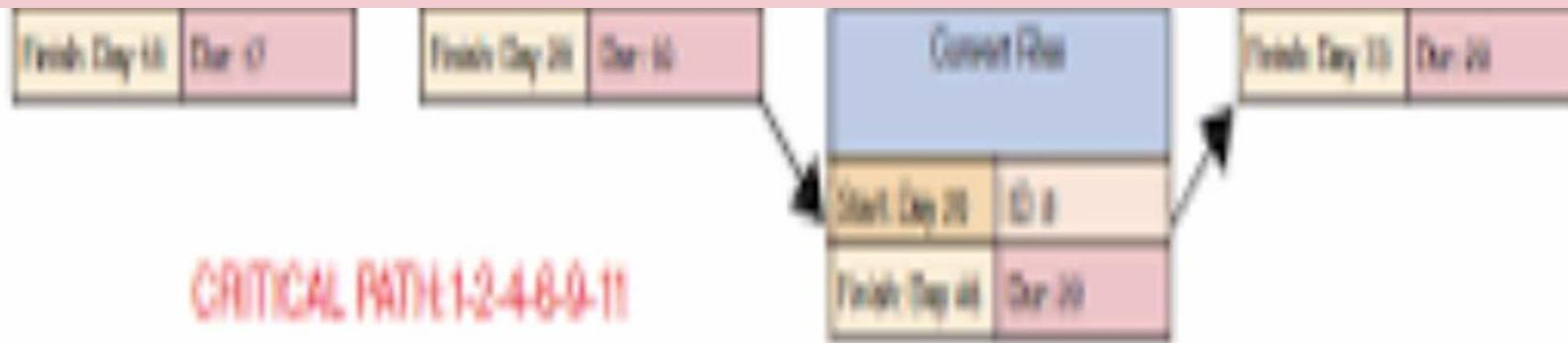
CRITICAL PATH 1-2-4-8-9-11



CPM

Critical Path Method (CPM) tries to answer the following questions:

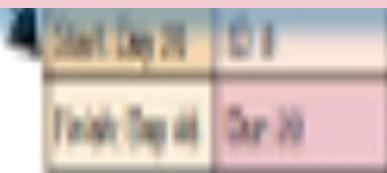
1. What is the duration of the project?
2. By how much (if at all) will the project be delayed if any one of the activities takes N days longer?
3. How long can certain activities be postponed without increasing the total project duration?



Critical Path

- Sequence of activities that have to be executed one after another
- Duration times of these activities will determine the overall project time, because there is no slack/float time for these activities
- If any of the activities on the critical path takes longer than projected, the entire project will be delayed by that same amount
- Critical path = Longest path in the precedence network (generally, the longest in time)

CRITICAL PATH 1-2-4-6-9-11



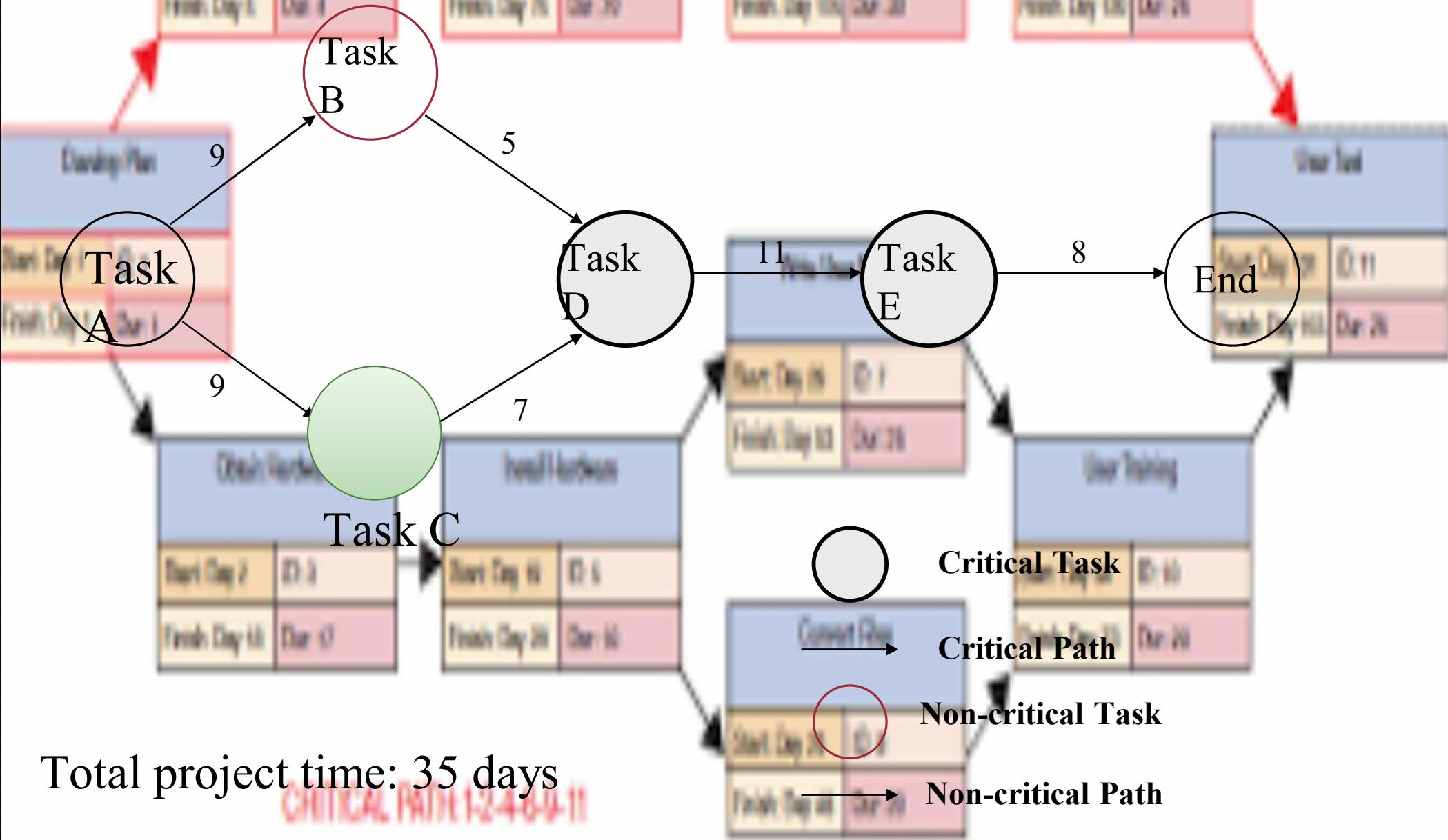
Task sequence/effort table

Task	Immediate prerequisite tasks	Effort (person-days)
A	None	9
B	A	5
C	A	7
D	B,C	11
E	D	8

Total effort required: 40 person-days

CRITICAL PATH 1-2-4-8-9-11

Graphical representation of tasks from previous table



Critical versus Non-Critical Paths

Critical path

- The path that takes the most time to complete.

Critical task (critical activity)

- A task that resides on the critical path.

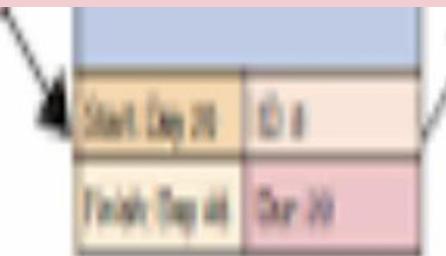
Non-critical path

- Any path that is not a critical path and thus takes less effort to complete than the critical path.

Non-critical task (non-critical activity)

- Any activity that resides on a non-critical path **and** not a critical path. These tasks may accept some delay in completion.

Critical Path 1-2-4-6-9-11



Critical Path Method (CPM)

“The critical path method analyses the precedence of activities to predict the total project duration. The method calculates which sequence activities (path) has the least amount of flexibility”

CRITICAL PATH 1-2-4-6-9-11

Task Scheduling

- **Forward-pass scheduling**

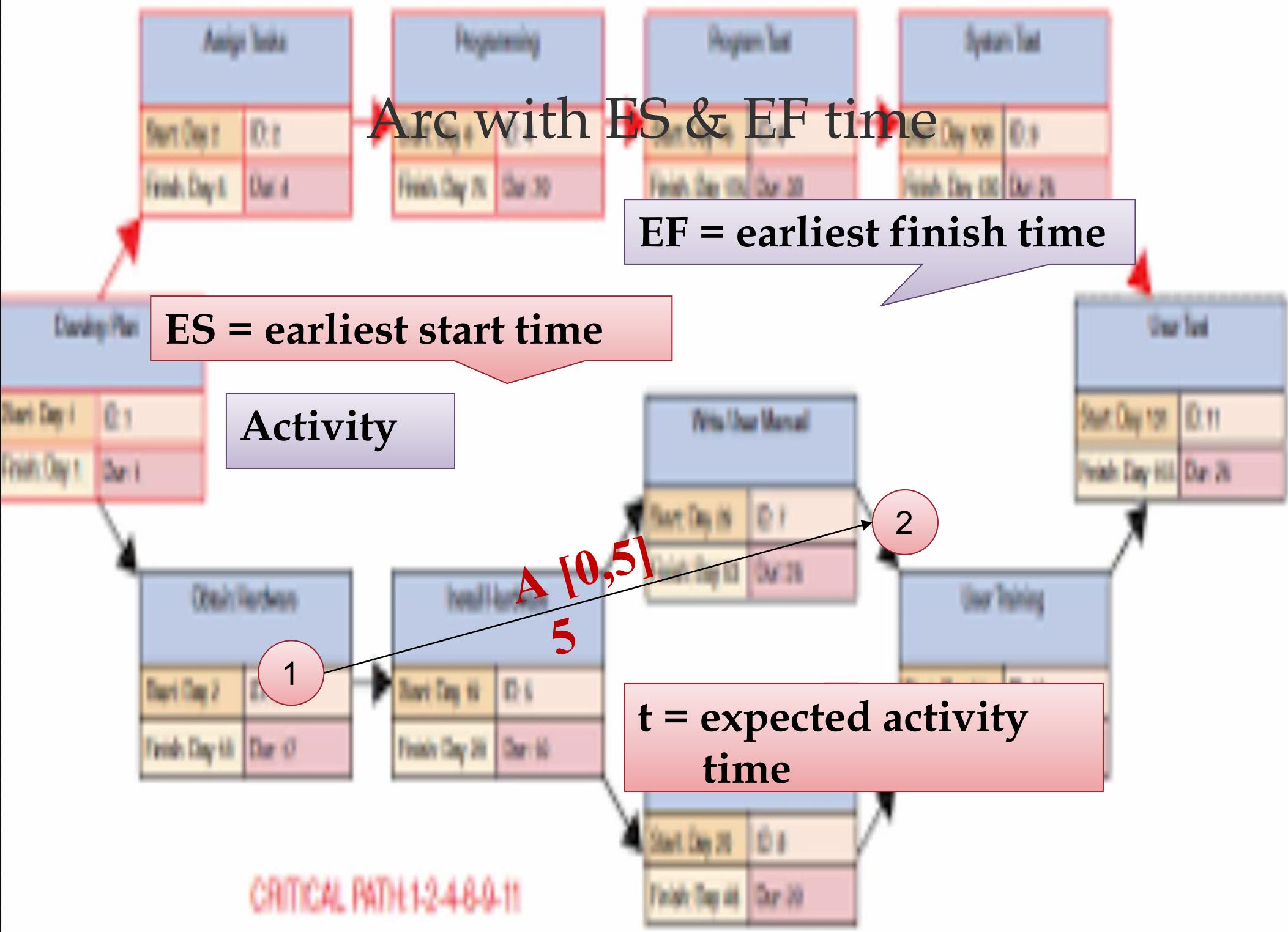
- Early start time (ES) and early finish (EF)

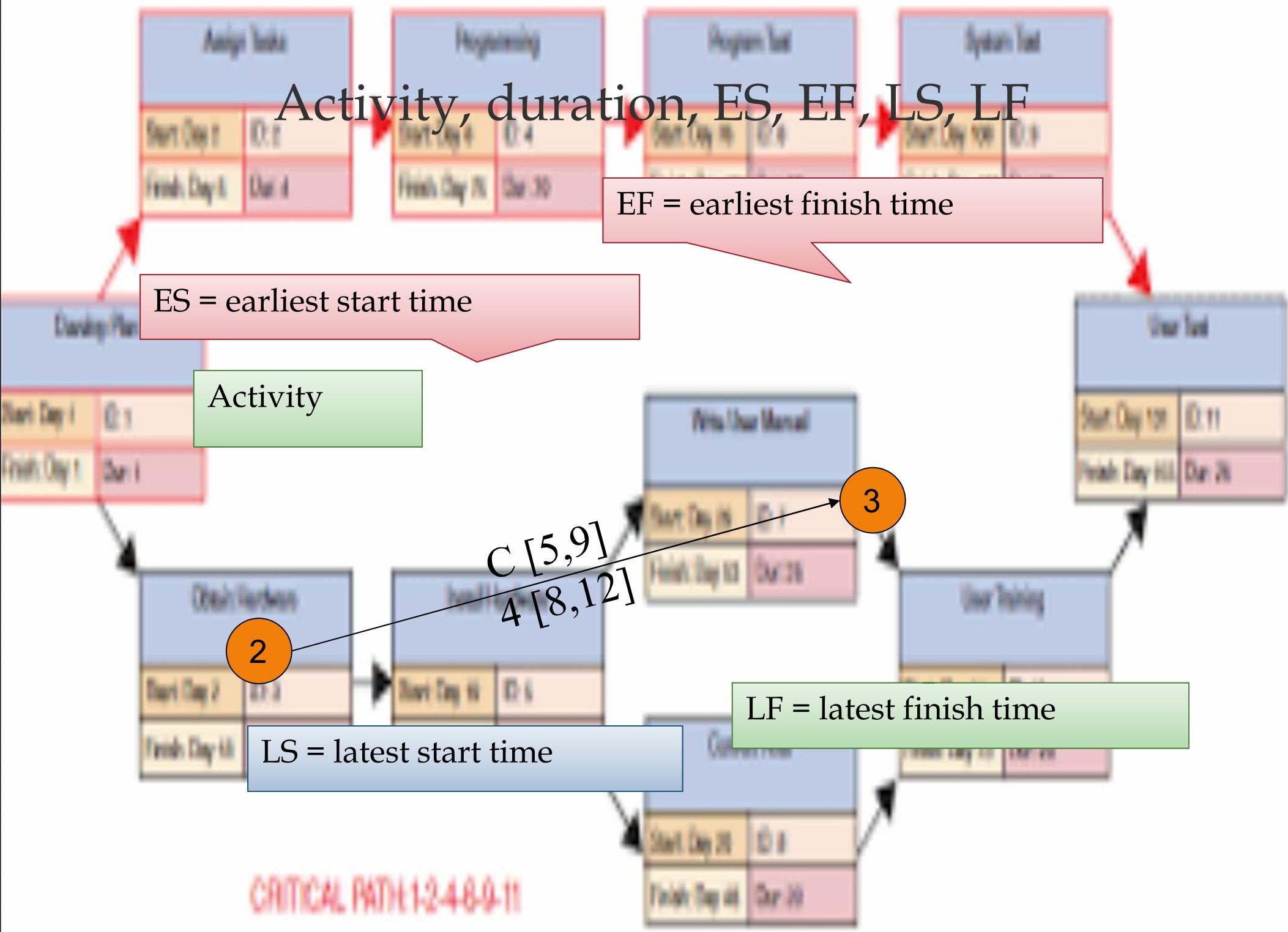
- **Backward-pass scheduling**

- Late start (LS) and Late finish (LF)

- **Slack time**

CRITICAL PATH 1-2-4-6-9-11





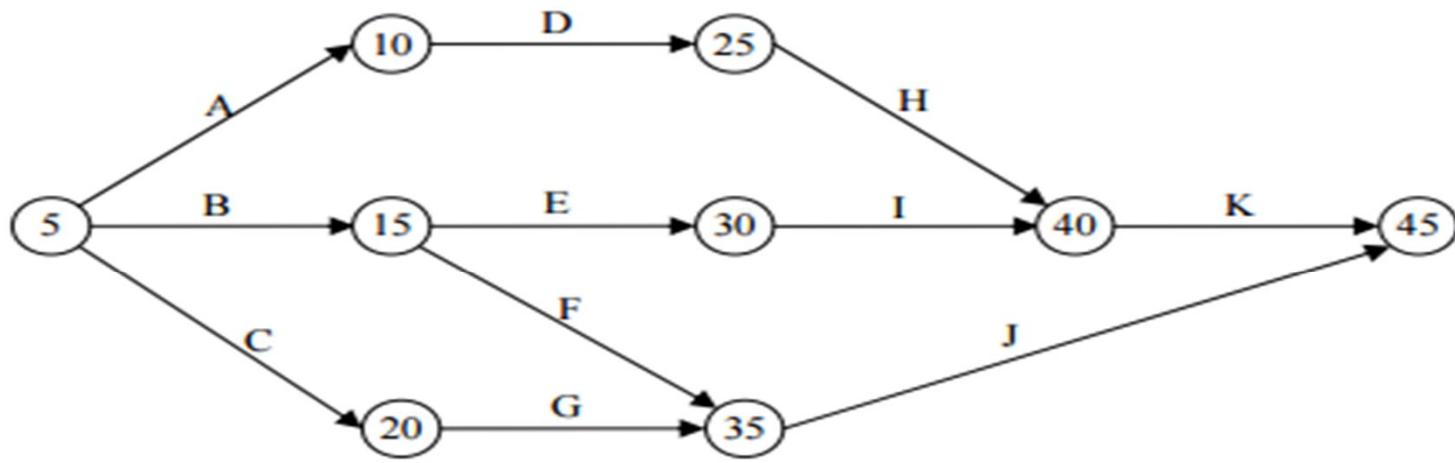


Figure 3.18: AOA network

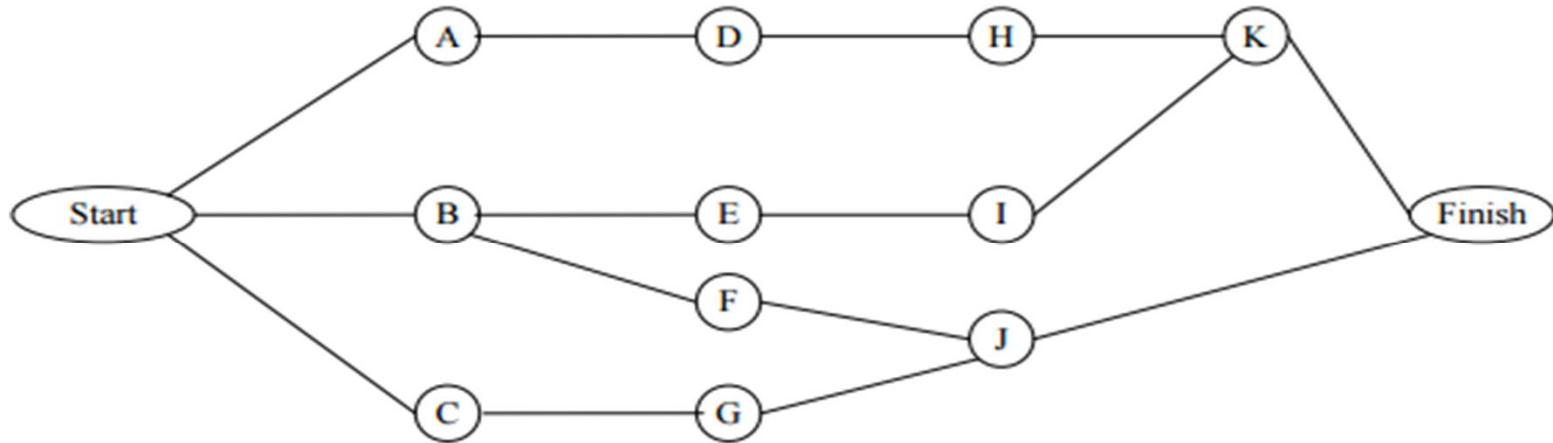
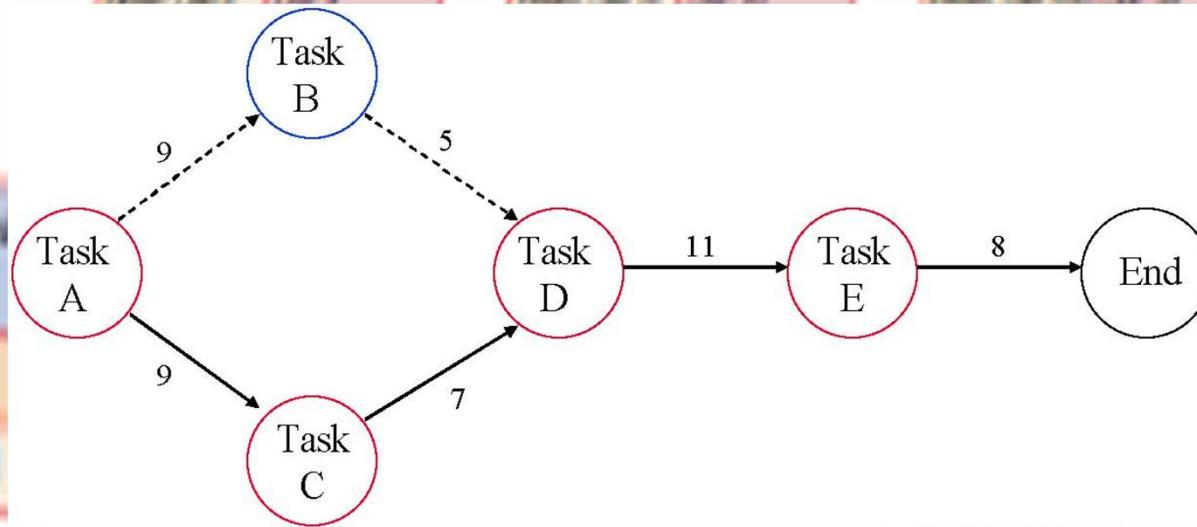


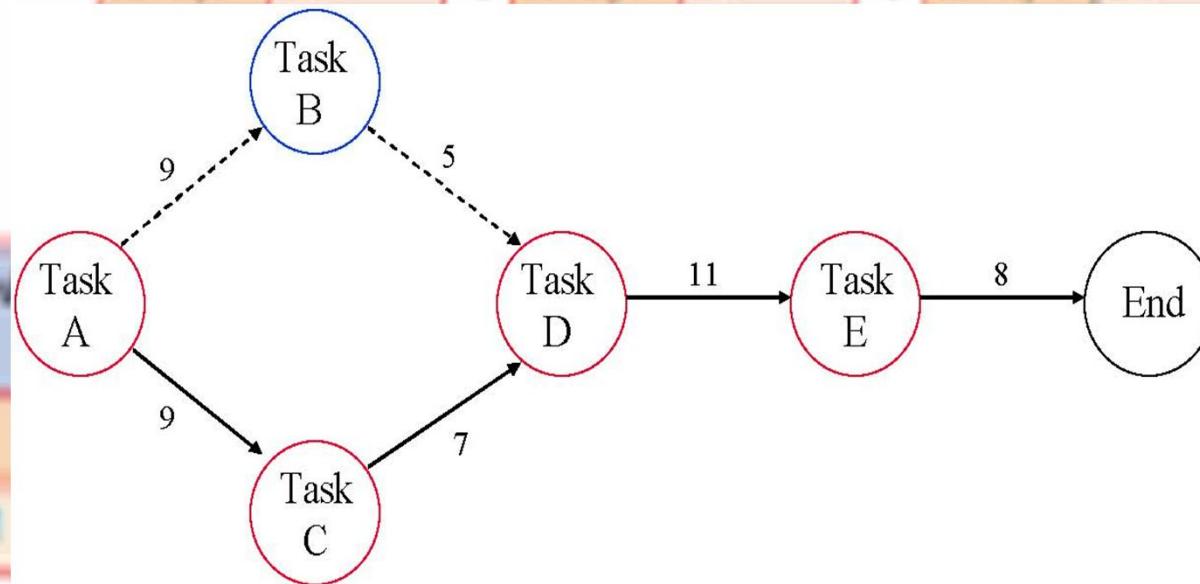
Figure 3.19: AON network

Forward-pass scheduling



Task	Tasks precedence	Task length	Earliest possible start time (ES)	Earliest possible finish time (EF)
A	None	9	0	9
B	A	5	9	14
C	A	7	9	16
D	B,C	11	16	27
E	D	8	27	35

Backward-pass scheduling



Task	Tasks precedence	Task length	Late start time (LS)	Late finish time (LF)
A	None	9	0	9
B	A	5	11	16
C	Finish A	7	9	16
D	B,C	11	16	27
E	D	8	27	35

CRITICAL PATH: A-B-C-D-E

Slack time

- **Total slack time of an activity**
 - The difference in start time between a **non-critical task's late start time** and its **early start time** or its **late finish time** and **early finish time**.

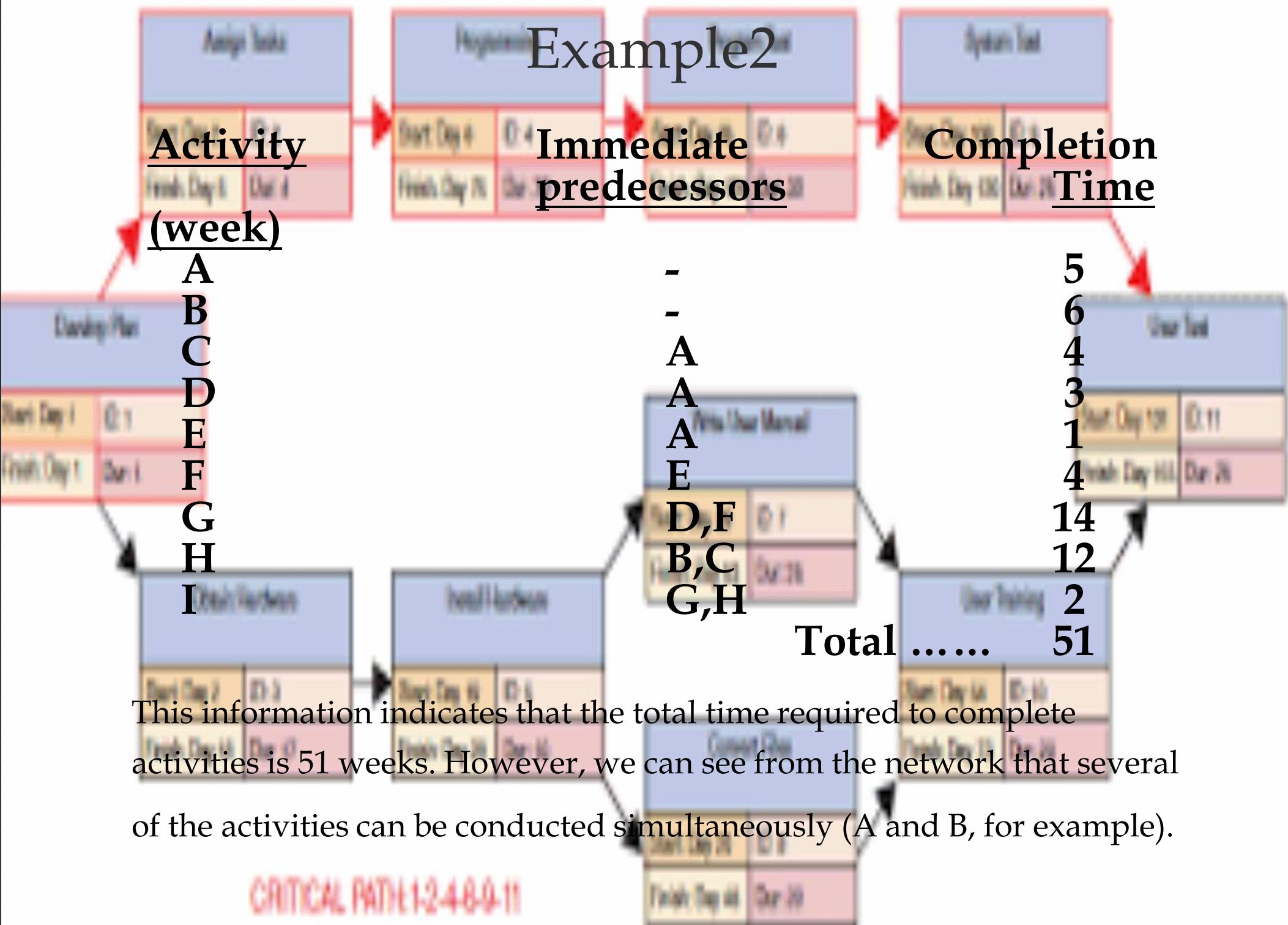
Total slack time of a task = LS - ES

Or

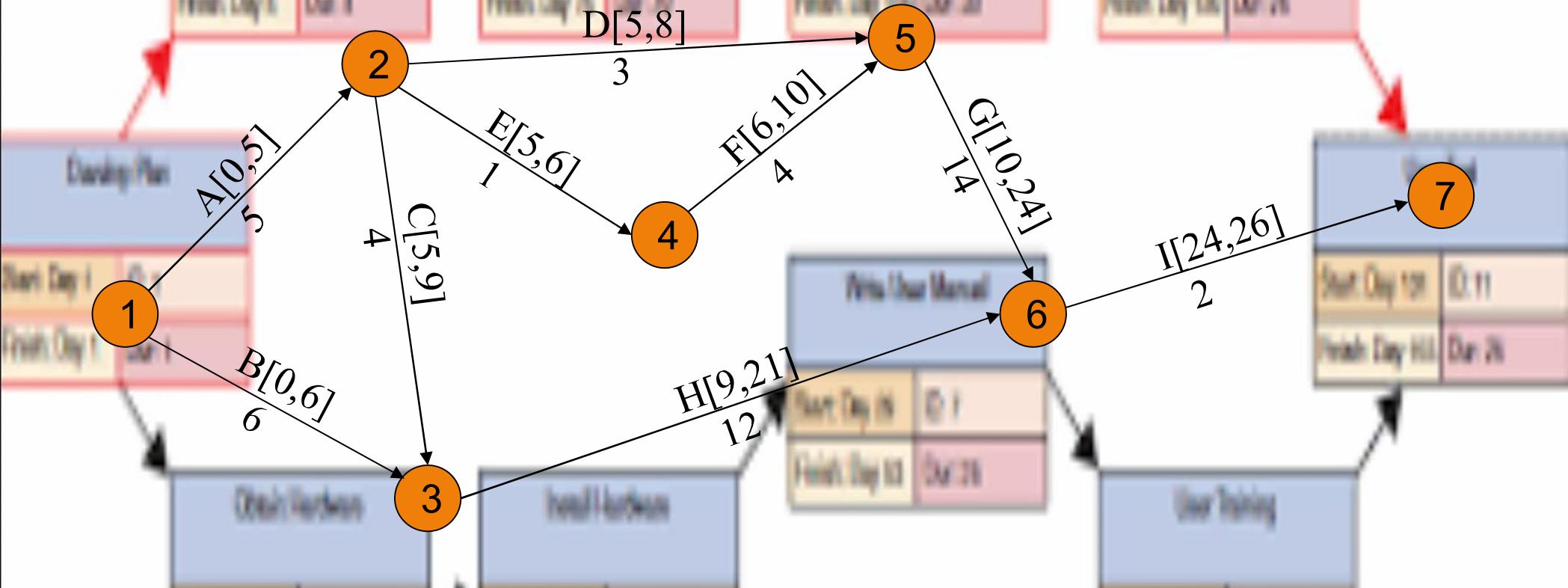
Total slack time of a task = LF – EF

e.g., Activity **B**: $LS - ES (11 - 9)$ or $LF - EF (16 - 14) \Rightarrow 2$

- **Total slack time**
 - The maximum allowable delay for all non-critical activities.



Network with ES & EF time



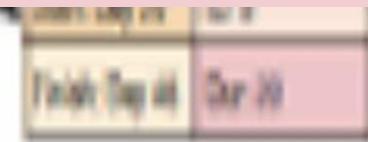
Earliest start time rule:

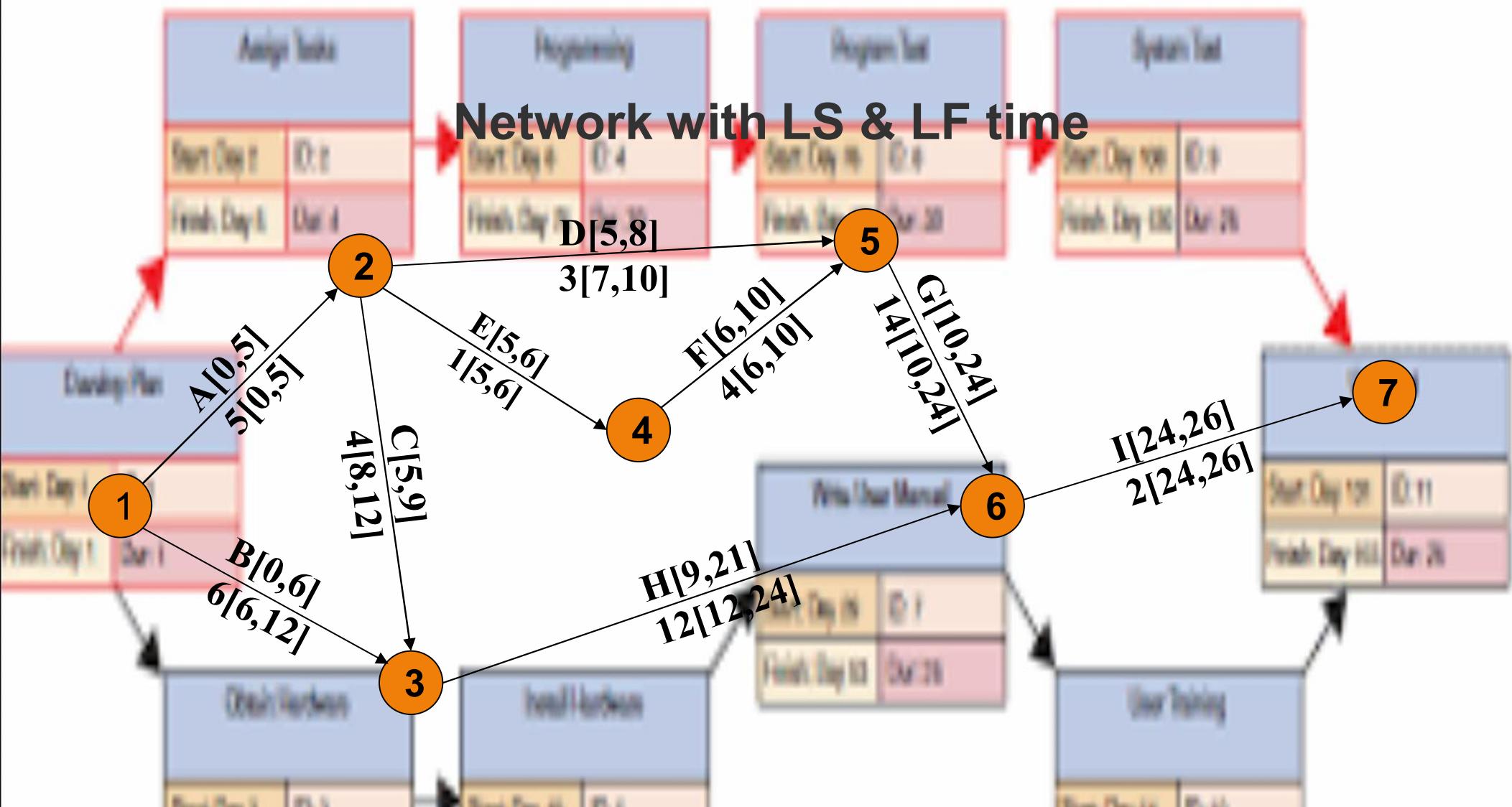
The earliest start time for an activity leaving a particular node is equal to the *largest of the earliest finish times* for all activities entering the node.

Latest start & latest finish time

- To find the critical path we need a backward pass calculation.
- Starting at the completion point (node 7) and using a latest finish time (LF) of 26 for activity I, we trace back through the network computing a latest start (LS) and latest finish time for each activity
- The expression $LS = LF - t$ can be used to calculate latest start time for each activity.
- For example, for activity I, $LF = 26$ and
- $t= 2$, thus the latest start time for activity I is
$$LS = 26 - 2 = 24$$

CRITICAL PATH 1-2-4-8-9-11





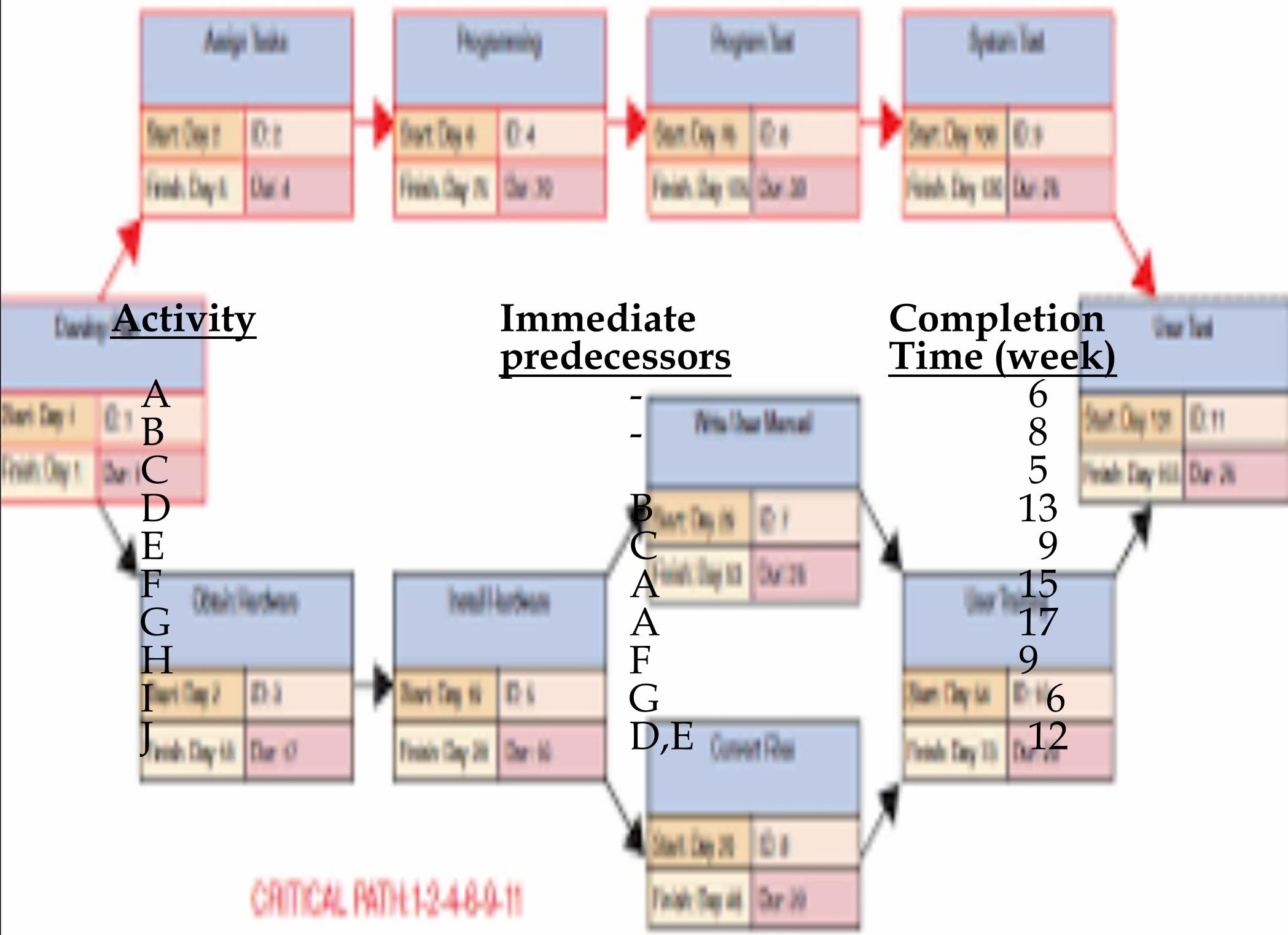
Latest finish time rule:

The latest finish time for an activity entering a particular node is equal to the smallest of the latest start times for all activities leaving the node.

Activity schedule for our example

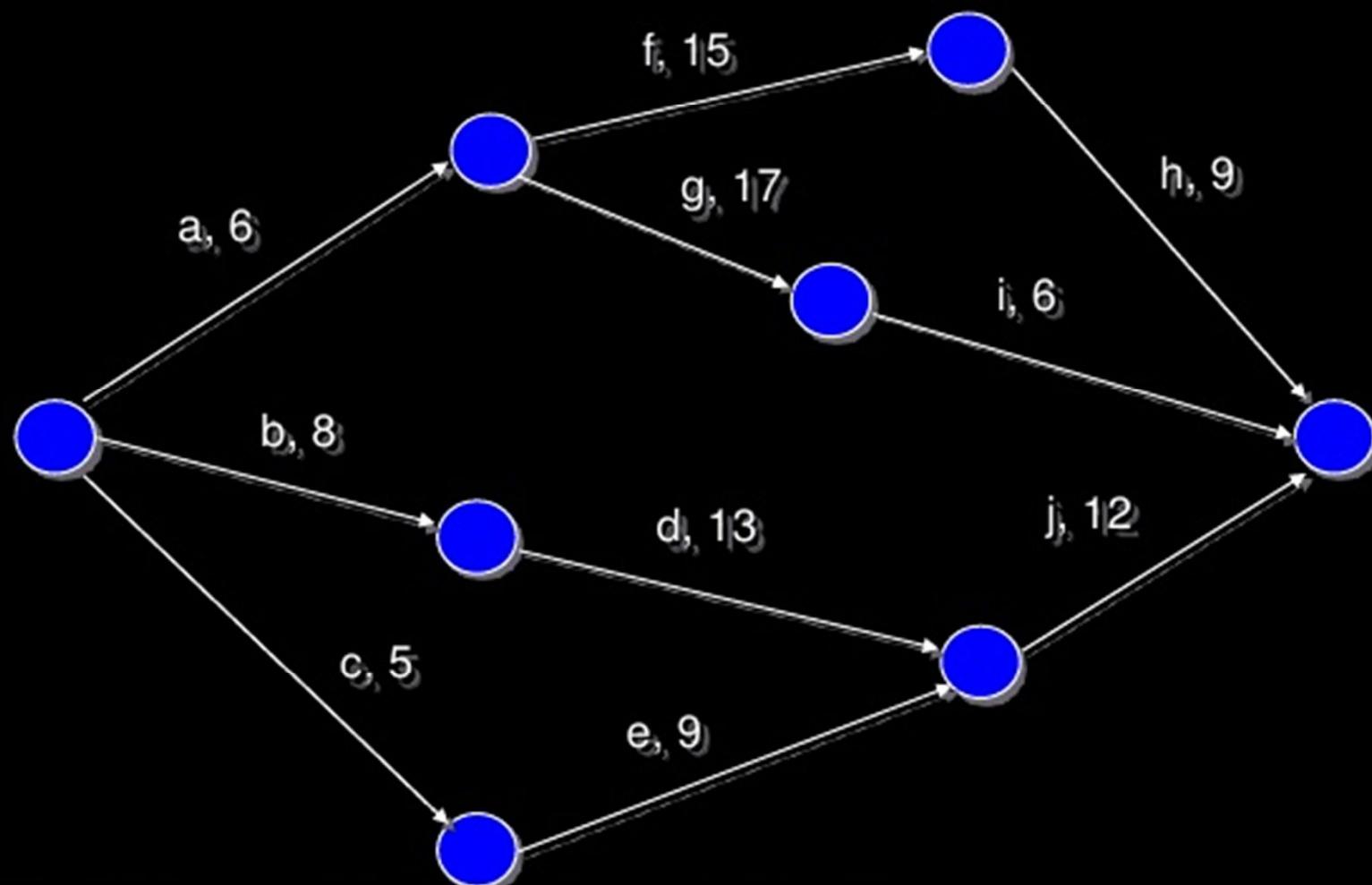
Activity	Earliest start (ES)	Latest start (LS)	Earliest finish (EF)	Latest finish (LF)	Slack (LS-ES)	Critical path
A	0	0	5	5	0	Yes
B	0	6	6	12	6	
C	5	8	9	12	3	
D	5	7	8	10	2	
E	5	5	6	6	0	Yes
F	6	6	10	10	0	Yes
G	10	10	24	24	0	Yes
H	9	12	21	24	3	
I	24	24	26	26	0	Yes

CRITICAL PATH A-B-C-D-E-F-G-I



CPM Example:

- CPM Network:-



IMPORTANT QUESTIONS

- **What is the total time to complete the project?**
 - 26 weeks if the individual activities are completed on schedule.
- **What are the scheduled start and completion times for each activity?**
 - ES, EF, LS, LF are given for each activity.
- **What activities are *critical* and must be completed as scheduled in order to keep the project on time?**
 - Critical path activities: A, E, F, G, and I.
- **How long can *non-critical* activities be delayed before they cause a delay in the project's completion time**
 - Slack time available for all activities are given.

Importance of Float (Slack) and Critical Path

1. Slack or Float shows how much allowance each activity has, i.e how long it can be delayed without affecting completion date of project.
2. Critical path is a sequence of activities from start to finish with zero slack. Critical activities are activities on the critical path.
3. Critical path identifies the minimum time to complete project
4. If any activity on the critical path is shortened or extended, project time will be shortened or extended accordingly

CRITICAL PATH 1-2-4-6-9-11

Importance of Float (Slack) and Critical Path (cont)

5. So, a lot of effort should be put in trying to control activities along this path, so that project can meet due date. If any activity is lengthened, be aware that project will not meet deadline and some action needs to be taken.
6. If can spend resources to speed up some activity, do so only for critical activities.
7. Don't waste resources on non-critical activity, it will not shorten the project time.
8. If resources can be saved by lengthening some activities, do so for non-critical activities, up to limit of float.
9. Total Float belongs to the path



PERT For Dealing With Uncertainty

- So far, times can be estimated with relative certainty, confidence
- For many situations this is not possible, e.g Research, development, new products and projects etc.

- Use 3 time estimates

m = most likely time estimate, mode.

a = optimistic time estimate,

b = pessimistic time estimate, and

$$\text{Expected Value (TE)} = (a + 4m + b) / 6$$

$$\text{Variance (V)} = ((b - a) / 6)^2$$

$$\text{Std Deviation } (\delta) = \text{SQRT (V)}$$

Precedences And Project Activity Times

	Immediate Predecessor	Optimistic Time	Most Likely Time	Pessimistic Time	EXP	Var	S.Dev
Activity					TE	V	σ
a	-	10	22	22	20	4	2
b	-	20	20	20	20	0	0
c	-	4	10	16	10	4	2
d	a	2	14	32	15	25	5
e	b,c	8	8	20	10	4	2
f	b,c	8	14	20	14	4	2
g	b,c	4	4	4	4	0	0
h	c	2	12	16	11	5.4	2.32
i	g,h	6	16	38	18	28.4	5.33
j	d,e	2	8	14	8	4	2

CRITICAL PATH 1-2-4-6-9-11

Finish Day 40 Sat 11

- Now, following is the process we follow with the two values:
- For every activity in the critical path, E and V are calculated.
- Then, the total of all Es are taken. This is the overall expected completion time for the project.
- Now, the corresponding V is added to each activity of the critical path. This is the variance for the entire project.
- This is done only for the activities in the critical path as only the critical path activities can accelerate or delay the project duration.
- Then, standard deviation of the project is calculated. This equals to the square root of the variance (V).

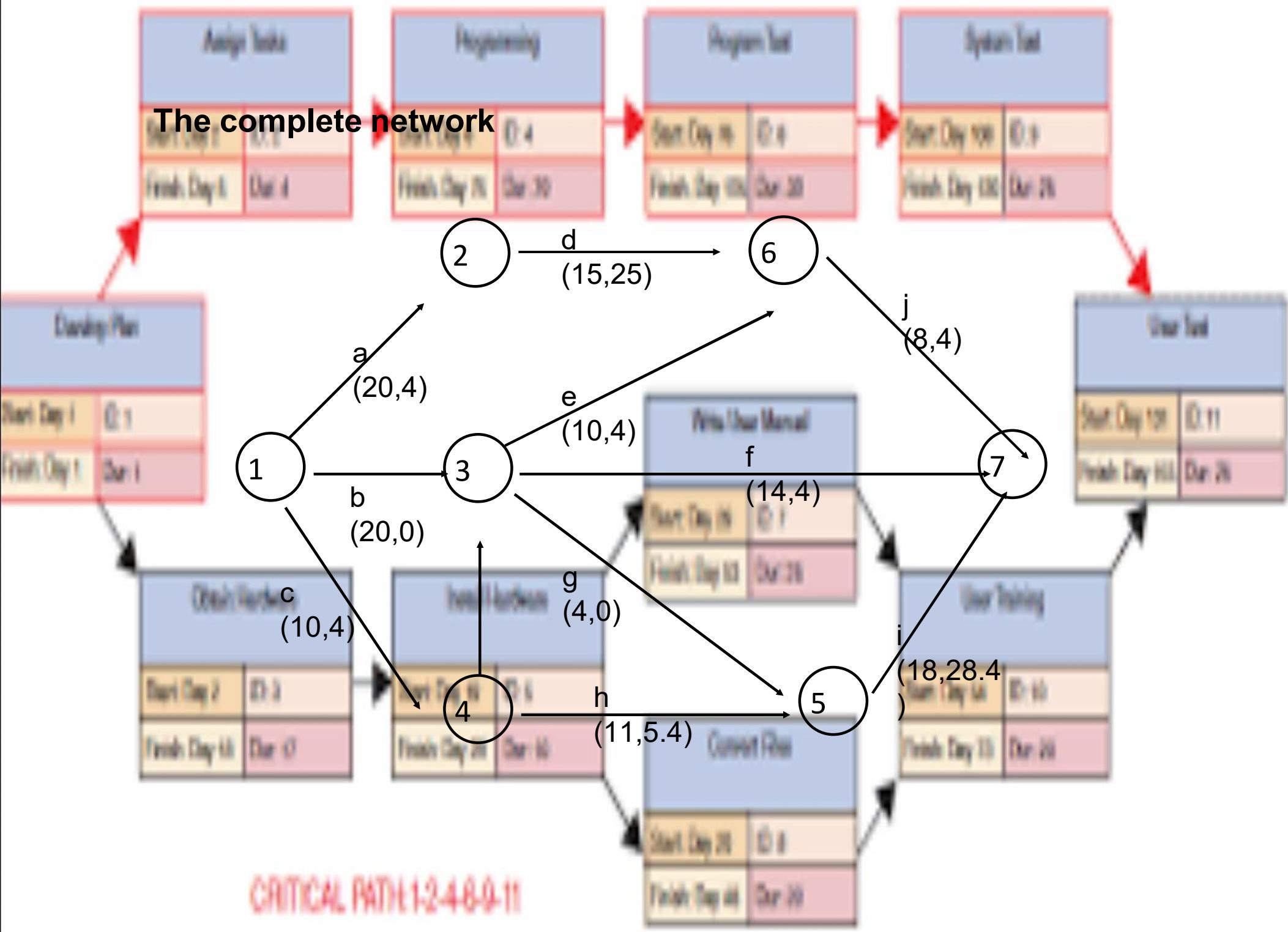
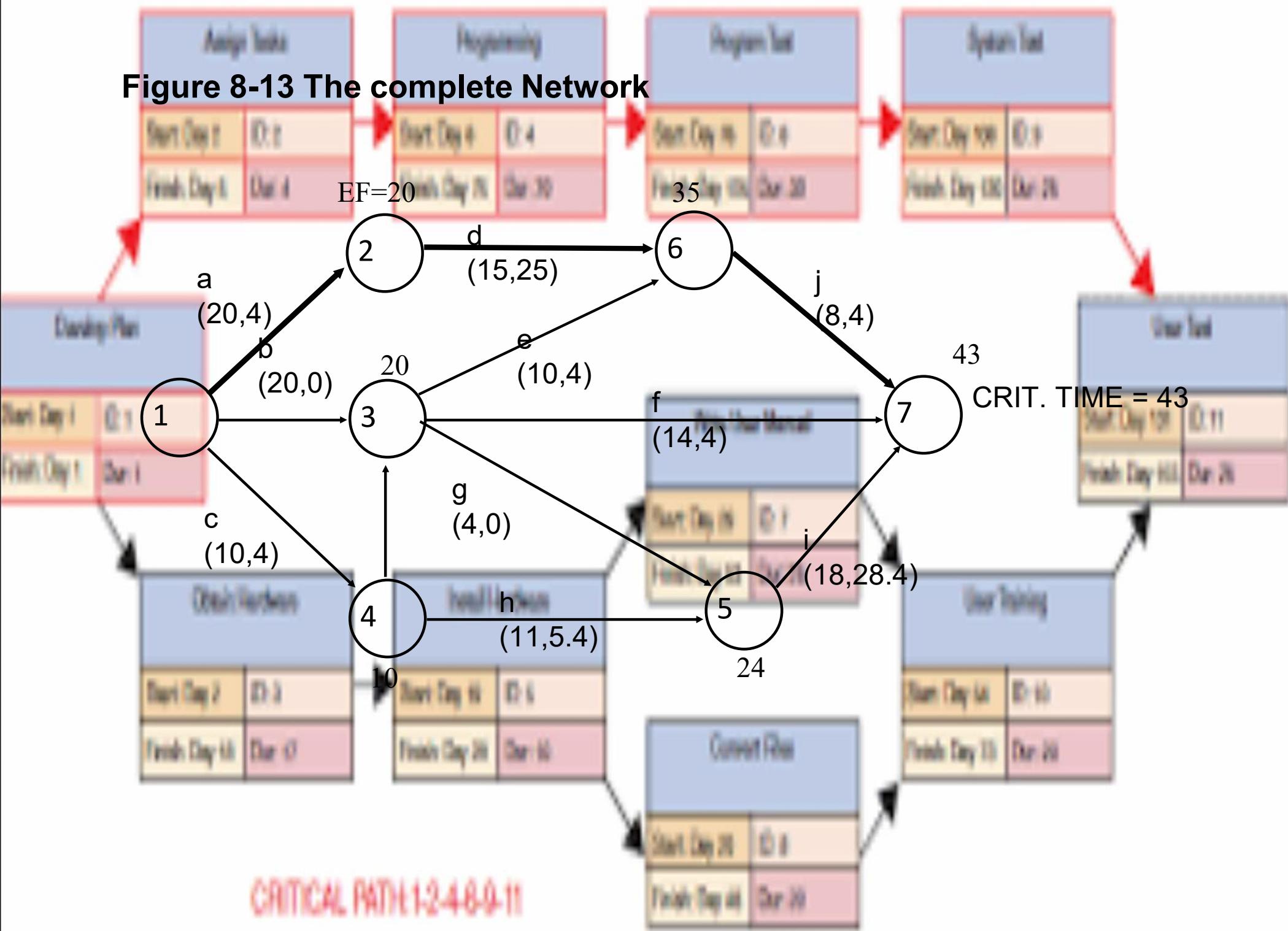


Figure 8-13 The complete Network



Critical Path Analysis (PERT)

Activity	LS	ES	Slacks	Critical ?
a	0	0	0	Yes
b	1	0	1	
c	4	0	4	
d	20	20	0	Yes
e	25	20	5	
f	29	20	9	
g	21	20	1	
h	14	10	4	
i	25	24	1	
j	35	35	0	Yes

CRITICAL PATH 1-2-4-6-9-11

Comparison Between CPM and PERT

	CPM	PERT
1	Uses network, calculate float or slack, identify critical path and activities, guides to monitor and controlling project	Same as CPM
2	Uses one value of activity time	Requires 3 estimates of activity time Calculates mean and variance of time
3	Used where times can be estimated with confidence, familiar activities	Used where times cannot be estimated with confidence. Unfamiliar or new activities
4	Minimizing cost is more important	Meeting time target or estimating percent completion is more important
5	Example: construction projects, building one off machines, ships, etc	Example: Involving new activities or products, research and development etc

CRITICAL PATH 1-2-4-6-9-11



Consistent framework for planning, scheduling, monitoring, and controlling project.

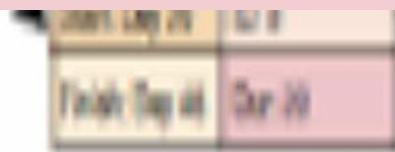
- Shows interdependence of all tasks, work packages, and work units.
- Helps proper communications between departments and functions.
- Determines expected project completion date.
- Identifies so-called critical activities, which can delay the project completion time.



BENEFITS OF CPM / PERT NETWORK (cont.)

- Identified activities with slacks that can be delayed for specified periods without penalty, or from which resources may be temporarily borrowed
- Determines the dates on which tasks may be started or must be started if the project is to stay in schedule.
- Shows which tasks must be coordinated to avoid resource or timing conflicts.
- Shows which tasks may run in parallel to meet project completion date

CRITICAL PATH 1-2-4-6-9-11



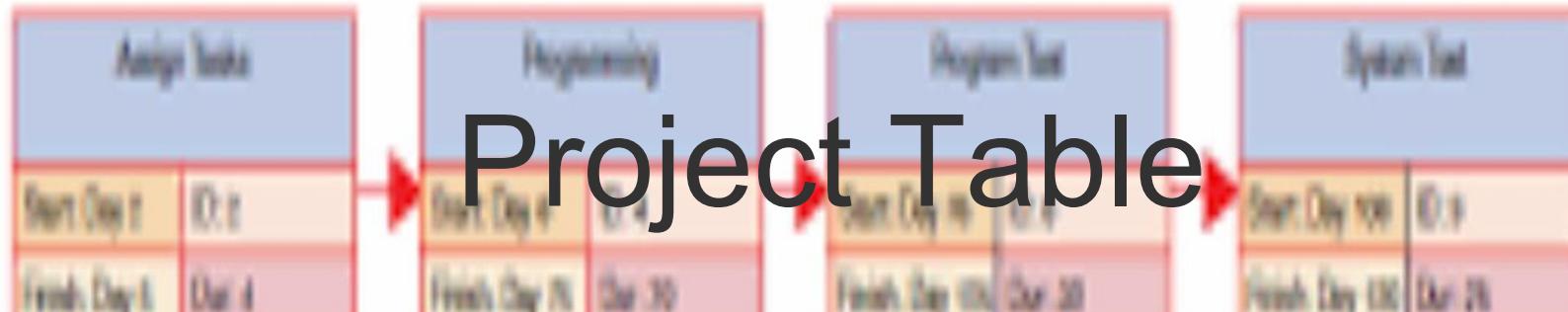
Timeline Charts

- The planner always begins with a set of tasks (the work breakdown structure).
- If automated tools are used, the work breakdown is input as a task network or task outline.
- Effort, duration, and start date are then input for each task. In addition, tasks may be assigned to specific individuals.
- A timeline chart enables you to determine what tasks will be conducted at a given point in time.
- A timeline chart can be developed for the entire project.
- Alternatively, separate charts can be developed for each project function or for each individual working on the project.

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

Contd.

- All project tasks are listed in the left-hand column.
- The horizontal bars indicate the duration of each task.
- When multiple bars occur at the same time on the calendar, task concurrency is implied.
- The diamonds indicate milestones.
- Once the information necessary for the generation of a timeline chart has been input, the majority of software project scheduling tools produce *project tables*
- It is a tabular listing of all project tasks, their planned and actual start- and end-dates, and a variety of related information.
- Project tables enable the project manager to track progress.



Project Table

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

CRITICAL PATH 1-2-4-8-9-11

Finish Day 11 | Dur. 10

Tracking the Project Schedule

- It is a road map for the Software Project
- It defines the tasks and milestones
- Tracking can be done by:
 - Conducting periodic project status meetings
 - Evaluating the results of all reviews
 - Determining whether milestones were reached by the scheduled date
 - Compare actual start date to planned start date
 - Meeting informally with professionals to get their subjective opinion

Tracking Earned Value

- The earned value system provides a common value scale for every task, regardless of the type of work being performed.
- The total hours to do the whole project are estimated, and every task is given an earned value based on its estimated percentage of the total.
- Basically, earned value is useful as it provides a quantitative technique of assessing progress on the project as a whole

EARNED VALUE ANALYSIS

- The earned value system provides a common value scale for every task, regardless of the type of work being performed.
- Simply stated, earned value is a measure of progress.
- It enables us to assess the “percent of completeness” of a project using quantitative analysis
- The total hours to do the whole project are estimated, and every task is given an earned value based on its estimated percentage of the total.

Tracking Earned Value

Example , Consider the following project

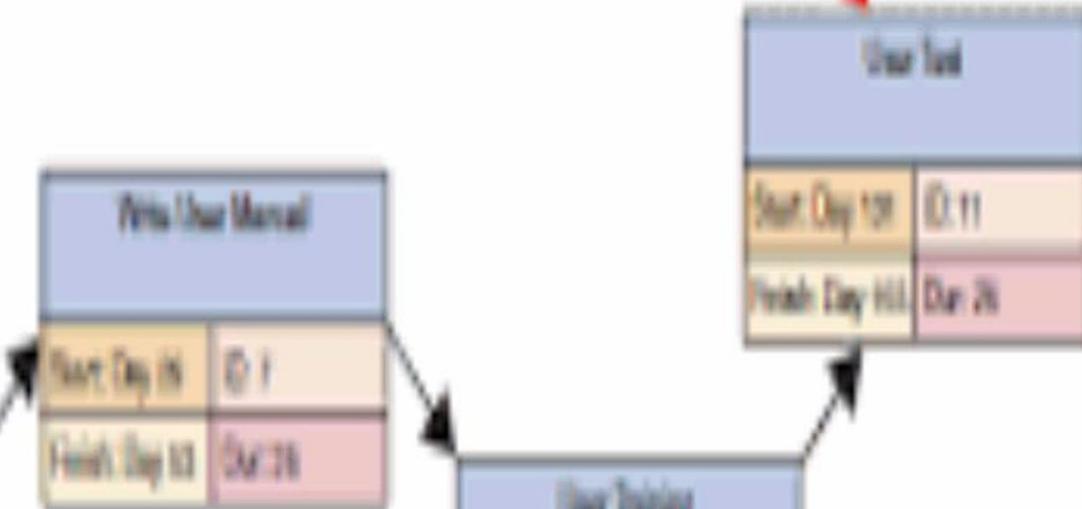
Task	Plan Minutes	Planned Value		Completion		Earned Value
		Unit	Cummulative	Plan	Actual	
Planning	180					
Requirements & Design						
Scope	60					
Product Features	60					
User Profile	45					
Operating Env.	30					
Design/Imp						
Constraint	30					
Assumptions	60					
Functional Requirements	300					
UI Requirements	120					
Use Cases	240					
Architectural Design	120					
DLD Module X	120					
DLD Module Y	100					
DLD Module Z	60					
Implementation						
Module X	120					
Module Y	120					
Module Z	60					
Testing						
Unit Testing X	180					
Unit Testing Y	120					
Unit Testing Z	60					
Int/Sys Test	120					
Total	2305					

Tracking a Project with Earned Value



In the example below, compiled at the end of the second week, we see that the group has completed the following tasks with their associated earned value:

- Daily Plan
- Planning - 7.81%
 - Scope - 2.64%
 - Product Features - 2.64%
 - User Profile - 1.98%
 - Assumptions - 2.64%
 - UI requirements - 5.21%



By summing the earned values for the completed tasks we determine that at the end of week 2, the cumulative earned value (the percent complete) is $7.81+2.64+2.64+1.98+2.64+5.21=22.91\%$.

Now, knowing the earned value the project group can determine if they are ahead, behind or on schedule to complete on time

CRITICAL PATH: 1-2-4-8-9-11

Tracking a Project with Earned Value

Looking at the cumulative planned value in the table, we see that at the end of week 2 to be on schedule, the group should have been 33.19% complete.

By the above calculation that they are actually only 22.91% complete, it appears that the project is falling behind schedule.

Task	Plan Minutes	Planned Value		Completion		Earned Value
		Unit	Cummulative	Plan	Actual	
Planning	180	7.81%	7.81%	Wk1	Wk1	7.81%
Requirements & Design						
Scope	60	2.60%	10.41%	Wk1	Wk2	2.64%
Product Features	60	2.60%	13.02%	Wk2	Wk2	2.64%
User Profile	45	1.95%	14.97%	Wk2	Wk1	1.98%
Operating Env.	30	1.30%	16.27%	Wk2		
Design/Imp Constraint	30	1.30%	17.57%	Wk2		
Assumptions	60	2.60%	20.17%	Wk2	Wk2	2.64%
Functional Requirements	300	13.02%	33.19%	Wk2		
UI Requirements	120	5.21%	38.39%	Wk3	Wk2	5.21%
Use Cases	240	10.41%	48.81%	Wk3		
Architectural Design	120	5.21%	54.01%	Wk4		
DLD Module X	120	5.21%	59.22%	Wk4		
DLD Module Y	100	4.34%	63.56%	Wk4		
DLD Module Z	60	2.60%	66.16%	Wk4		
Implementation						
Module X	120	5.21%	71.37%	Wk5		
Module Y	120	5.21%	76.57%	Wk5		
Module Z	60	2.60%	79.18%	Wk6		
Testing						
Unit Testing X	180	7.81%	86.98%	Wk5		
Unit Testing Y	120	5.21%	92.19%	Wk6		
Unit Testing Z	60	2.60%	94.79%	Wk6		
Int/Sys Test	120	5.21%	100.00%	Wk6		
Total	2305	100.00%				

Tracking a Project with Earned Value

Knowing this allows the project team to take action.

Such action may involve

- adjusting work practices,
- negotiating with the client for an extension to the deadline,
- etc...

It is argued that earned value provides accurate and reliable readings of performance as early as 15% into a project. This kind of quantitative project management is essential if projects are to be consistently delivered on time.

CRITICAL PATH 1-2-4-8-9-11

To determine the earned value

- **The budgeted cost of work scheduled (BCWS) is determined for each work task represented in the schedule.**
 - **BCWS_i** is the effort planned for work task i.
 - To determine progress at a given point along the project schedule, the value of BCWS is the sum of the BCWS_i values for all work tasks for that point in time.
- The BCWS values for all work tasks are summed to derive the **budget at completion, BAC**.
- Hence, $BAC = \sum (BCWS_k)$ for all tasks k

- Next, the value for **budgeted cost of work performed (BCWP)** is computed.
- The value for BCWP is the sum of the BCWS values for all work tasks that have actually been completed by a point in time on the project schedule.
 - BCWS - budget of the activities that were planned to be completed
 - BCWP - budget of the activities that actually were completed."
- Given values for BCWS, BAC, and BCWP, important progress indicators can be computed:
 - Schedule performance index, $SPI = BCWP/BCWS$
 - Schedule variance, $SV = BCWP - BCWS$
- SPI is an indication of the efficiency with which the project is utilizing scheduled resources.
- SPI value close to 1.0 indicates efficient execution of the project schedule.

CRITICAL PATH 1-2-4-6-9-11

- Percent scheduled for completion = BCWS/BAC
 - . Indication of the percentage of work that should have been completed by time t .

- Percent complete = BCWP/BAC
 - . provides a quantitative indication of the percent of completeness of the project at a given point in time, t .

- Actual cost of work performed, ACWP, is the sum of the effort actually expended on work tasks that have been completed by a point in time on the project schedule. It is then possible to compute

- . Cost performance index, CPI = BCWP/ACWP
 - . Cost variance, CV = BCWP – ACWP
- CPI value close to 1.0 provides a strong indication that the project is within its defined budget

CRITICAL PATH 1-2-4-6-9-11

Software Engineering(SE)

CSC 601



Subject Incharge

Varsha Nagpurkar
Assistant Professor
Room No. 407

email: varshanagpurkar@sfit.ac.in

Module-4 Syllabus

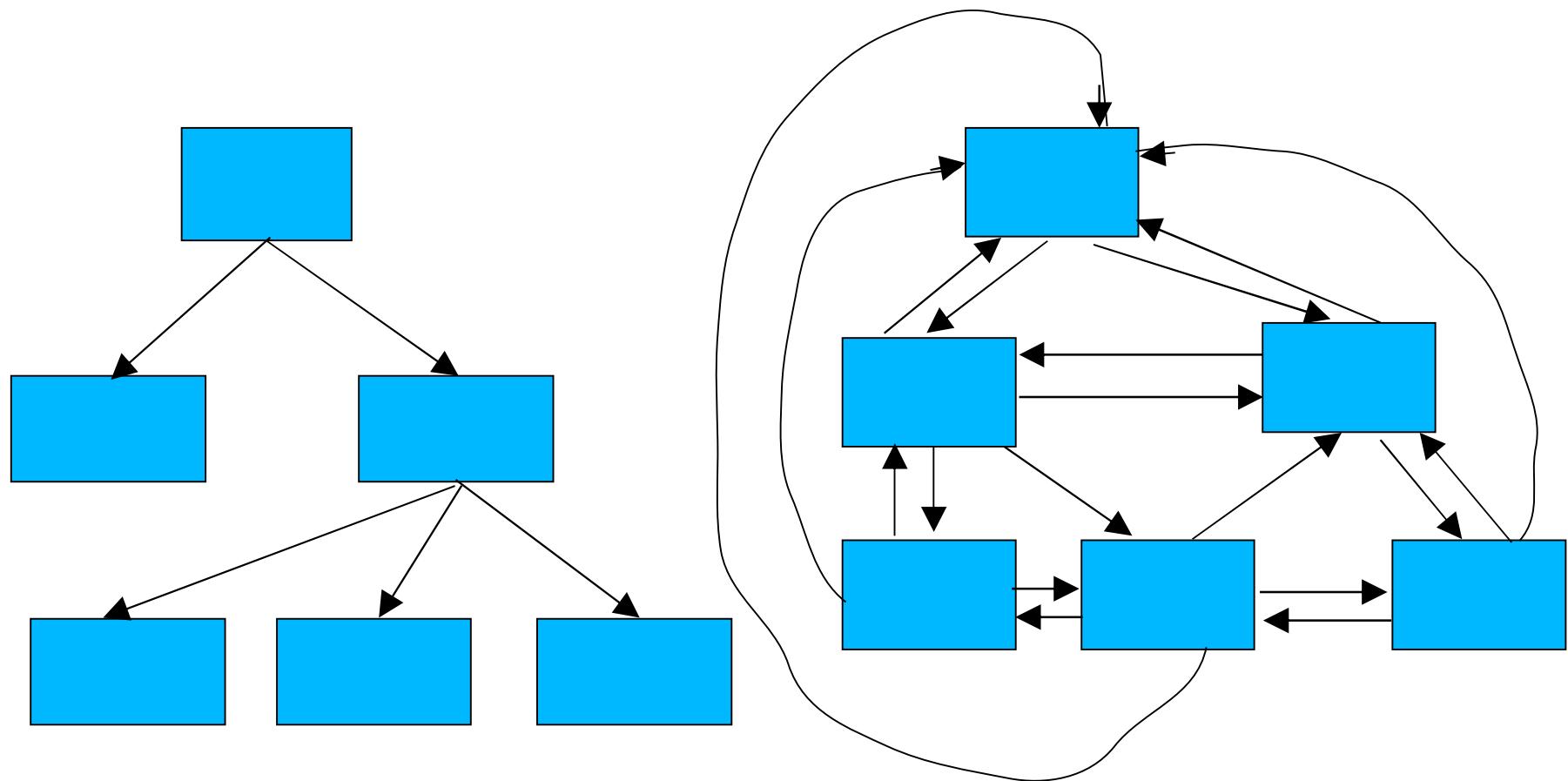
4.0	Software Design	10
4.1	Design Principles, Design Concepts, Effective Modular Design – Cohesion and Coupling	
4.2	Architectural Design	
4.3	Component-level design	
4.4	User Interface Design	



Modularity

- ◆ Modularity derives from the architecture.
- ◆ Modularity is a logical partitioning of the software design that allows complex software to be manageable for purposes of implementation and maintenance.
- ◆ The logic of partitioning is based on related functions, implementation considerations, data links, or other criteria.
- ◆ Modularity does imply interface overhead related to information exchange between modules and execution of modules.

Modularity





Modularity

- ◆ In technical terms, modules should display:
 - high cohesion
 - low coupling.

Cohesion

- It implies that a component or class encapsulates only attributes and operations that are closely related to one another and to the class or component itself
- Cohesion is the “single-mindedness” of a component
- Cohesion and coupling are necessary in making any software reliable and extendable



Cohesion and Coupling

- ◆ Cohesion is a measure of:
 - functional strength of a module.
 - A cohesive module performs a single task or function.
- ◆ Coupling between two modules:
 - a measure of the degree of interdependence or interaction between the two modules.

Cohesion and Coupling

- ◆ A module having high cohesion and low coupling:
 - functionally independent of other modules.
 - Complexity of design is reduced
 - an error existing in one module does not directly affect other modules.
 - Reuse of modules is possible

```
// Less cohesive class design  
  
class BudgetReport {  
  
    void connectToRDBMS() {  
  
    }  
    void generateBudgetReport() {  
    }  
    void saveToFile() {  
    }  
    void print() {  
    }  
}
```

More cohesive class design

```
// More cohesive class design  
  
class BudgetReport {  
    Options getReportingOptions() {  
    }  
    void generateBudgetReport(Options o) {  
    }  
}  
class ConnectToRDBMS {  
    DBconnection getRDBMS() {  
    }  
}  
class PrintStuff {  
    PrintOptions getPrintOptions() {  
    }  
}  
class FileSaver {  
    SaveOptions getFileSaveOptions() {  
    }  
}
```

Cohesion



Degree of cohesion

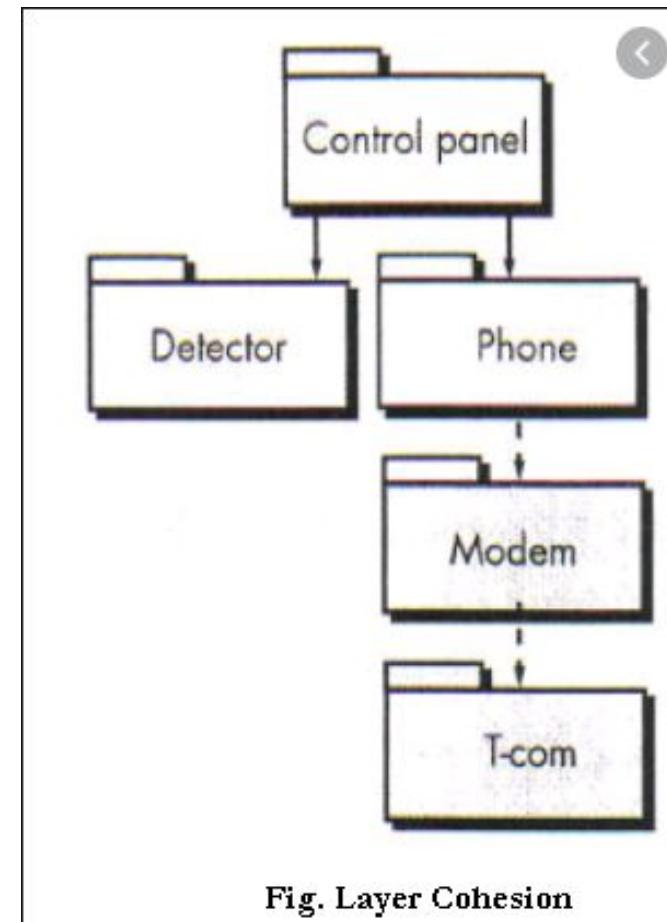
Functional cohesion

- Exhibited primarily by operations, this level of cohesion occurs when a module performs one and only one computation and then returns a result
- Different elements of a module cooperate:
 - to achieve a **single function**,
 - e.g. managing an employee's pay-roll.

Layer cohesion

- Exhibited by packages, components, and classes, it occurs when a higher layer accesses the services of a lower layer, but lower layers do not access higher layers
- For example, the Safehome security function requirement to make an outgoing phone call if an alarm is sensed

Layer Cohesion



Communicational cohesion

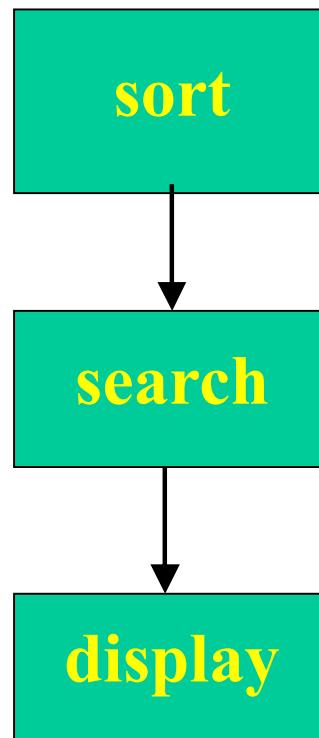
- All operations that access the same data are defined within one class
- All functions of the module:
 - reference or update the same data structure
 - Example:
 - Update record in data base and send it to the printer
 - Update a record on a database
 - Print the record

Cohesion

- Classes and components that exhibit functional, layer, and communicational cohesion are relatively easy to implement, test and maintain
- The designer should strive to achieve these levels of cohesion

Sequential cohesion

- ◆ Def: The output of one part is the input to another



Sequential cohesion

- Components or operations are grouped in a manner that allows the first to provide input to the next and so on
- The intent is to implement a sequence of operations

Procedural cohesion

- ◆ Def: Elements of a component are related only to ensure a particular order of execution
- ◆ The set of functions of the module:
 - certain sequence of steps have to be carried out in a certain order for achieving an objective,
 - E.g. Bank Transactions

Procedural cohesion

- Components or operations are grouped in a manner that allows one to be invoked immediately after the preceding one was invoked, even when there is no data passed between them

Temporal

- Operations that are performed to reflect a specific behaviour or state,
- e.g.,an operation performed at start-up or all operations performed when an error is detected

Temporal cohesion

- ◆ Def: Elements are related by timing involved
- ◆ Example:
 - The set of functions responsible for
 - initialization,
 - start-up, shut-down of some process, etc.
- ◆ Example: An exception handler that
 - Closes all open files
 - Creates an error log
 - Notifies user



Example

- ◆ A system initialization routine: this routine contains all of the code for initializing all of the parts of the system.
- ◆ Lots of different activities occur, all at init time.



Logical cohesion

- ◆ All elements of the module perform similar operations:
 - e.g. **error handling, data input, data output, etc.**
- ◆ An example of logical cohesion:
 - a set of print functions or plot graphs to generate an output report arranged into a single module.



Example

- ◆ A component reads inputs from tape, disk, and network.
- ◆ All the code for these functions are in the same component.
- ◆ Operations are related, but the functions are significantly different.

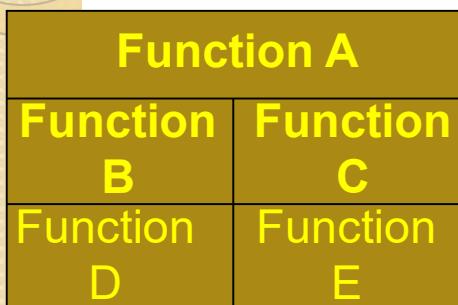
Utility cohesion

- Components, classes, or operations that exist within the same category but are otherwise unrelated are grouped together
- For example, a class called Statistics exhibits utility cohesion if it contains all attributes and operations required to compute six simple statistical measures

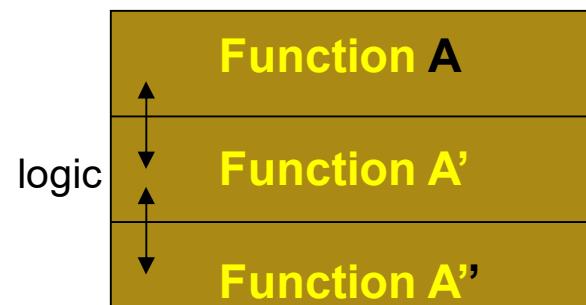
Cohesion

- These levels of cohesion are less desirable and should be avoided when design alternatives exist

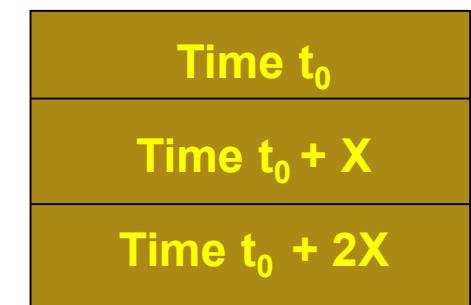
Examples of Cohesion



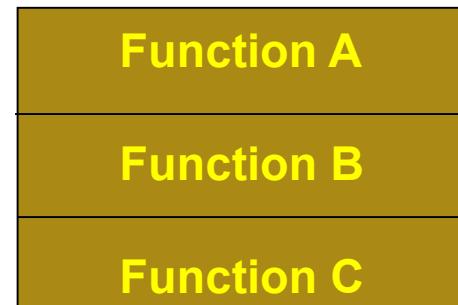
Coincidental
Parts unrelated



Logical
Similar functions

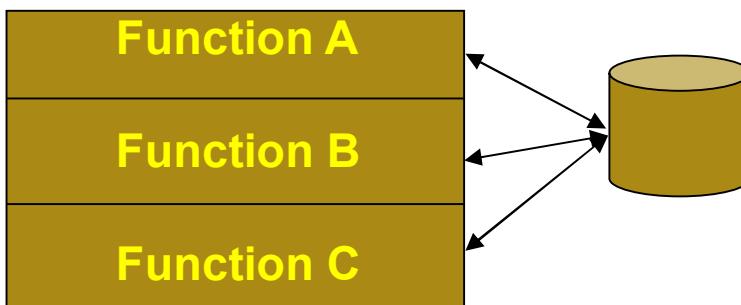


Temporal
Related by time

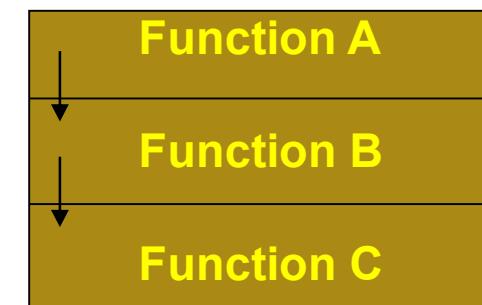


Procedural
Related by order of functions

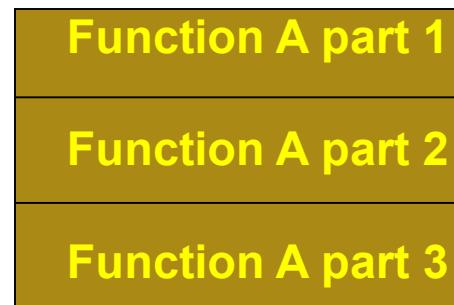
Examples of Cohesion (Cont.)



Communicational
Access same data



Sequential
Output of one is input to another



Functional
Sequential with complete, related functions

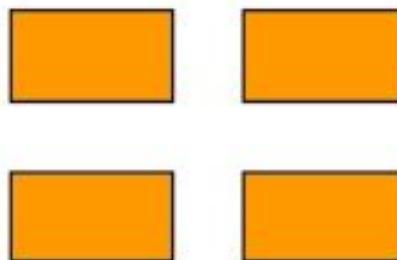


Coupling

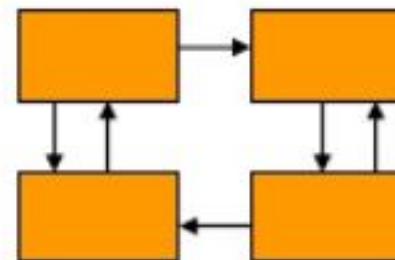
- ◆ Coupling indicates:
 - how closely two modules interact or how interdependent they are.
 - The degree of coupling between two modules depends on their interface complexity.

Coupling

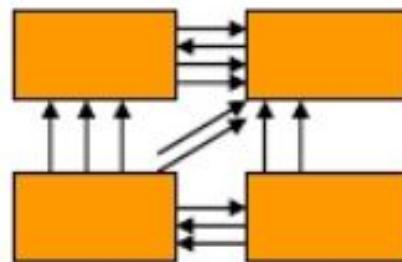
- Degree of dependence among components.



No dependencies



Loosely coupled-some dependencies



Highly coupled-many dependencies

- High coupling makes modifying parts of the system difficult, e.g., modifying a component affects all the components to which the component is connected.

Coupling

- Communication and collaboration are essential elements of any object-oriented system
- As the amount of communication and collaboration increases(i.e.,as the degree of “connectedness” between classes grows),the complexity of the system also increases
- And as complexity rises,the difficulty of implementing,testing, and maintaining

Coupling

- Coupling is a qualitative measure of the degree to which classes are connected to one another
- As classes(and components) become more dependent,coupling increases
- An important objective in component-level design is to keep coupling as low as is possible

Types of coupling

Try to achieve

Data Coupling

Stamp Coupling

Control Coupling

Common coupling

Content coupling

Avoid

Degree of coupling

Content Coupling : (worst) When a module uses/alters data in another module

Common Coupling : 2 modules communicating via **global** data

External Coupling : Modules are tied to an environment external to the software

Control Coupling : 2 modules communicating with a control flag

Stamp Coupling : Communicating via a data structure **passed** as a parameter. The data structure holds **more** information than the recipient needs.

Data Coupling : (best) Communicating via parameter passing. The parameters passed are only those that the recipient needs.

No data coupling : independent modules.

Content coupling

- It occurs when one component “surreptitiously(in a secret or unauthorized way)modifies data that is internal to another component
- When a module alters/uses data in another module
- This violates information hiding-a basic design concept



Content coupling

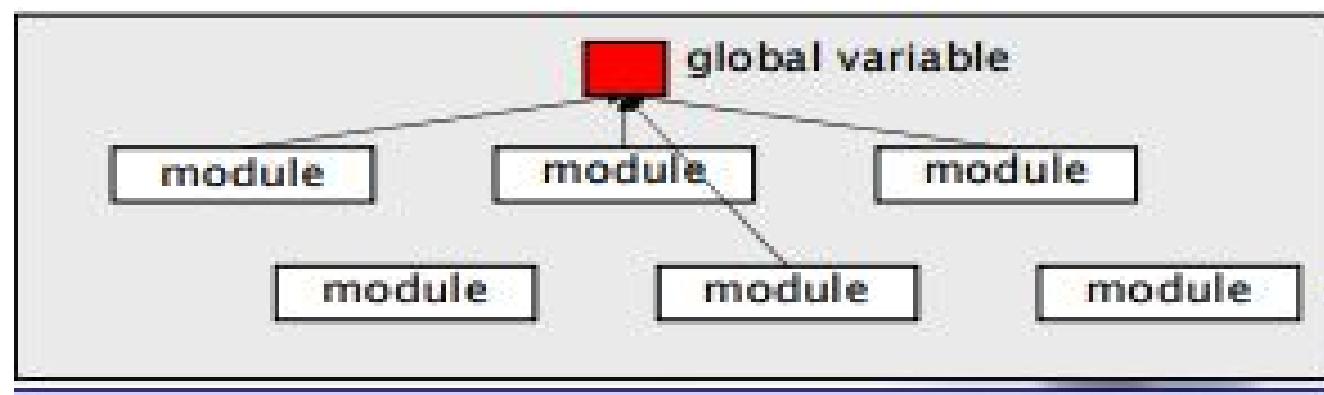
- ◆ Def: One component modifies another
- ◆ Example:
 - Component directly modifies another's data
 - Component modifies another's code, e.g., jumps (goto) into the middle of a routine
- ◆ **The degree of coupling increases**
 - **from data coupling to content coupling**

Common Coupling

- Occurs when a number of components all make use of a global variable
- Two modules communicating via global data
- Although this is sometimes necessary(e.g.for establishing default values that are applicable throughout an application),common coupling can lead to uncontrolled error propagation and unforeseen side effects when changes are made

Common Coupling

- ◆ Def: More than one component share data such as **global data structures**.
- ◆ Component passes control parameters to coupled components.



Common Coupling

- Two modules are common-coupled if they both have access to the same global variables

Drawbacks :

- It contradicts the spirit of structured programming
- Modules can have side effects
- Common coupled modules are difficult to reuse
- As a consequence of common coupling, a module may be exposed to more data than it needs

Control Coupling

- Occurs when operation A() invokes operation B() and passes a control flag to B
- Two modules communicating with the control flag
- The control flag then “directs” logical flow within B
- The problem with this form of coupling is that an unrelated change in B can result in the necessity to change the meaning of the control flag that A passes
- If this is overlooked, an error will result

Control coupling

- ◆ Data from one module is used to direct the order of instruction execution in another.
- ◆ Component passes control parameters to coupled components.
- ◆ Example 1: sort that takes a comparison function as an argument.
- ◆ Example 2: Main function controls all function
- ◆ Example 3: a flag set in one module and tested in another module

Stamp coupling

- Occurs when ClassB is declared as a type for an argument of an operation of Class A.
- Because ClassB is now a part of the definition of ClassA, modifying the system becomes more complex
- Communicating via a data structure passed as a parameter .The data structure holds more information than the recipient needs

Stamp coupling

- ◆ Def: Component passes a data structure to another component that does not have access to the entire structure.

Customer Billing System

- ◆ The print routine of the customer billing accepts customer data structure as an argument, parses it, and prints the name, address, and billing information
- ◆ Accessing date of birth to calculate age

Data coupling(Best)

- Occurs when operations pass long strings of data arguments
- The “bandwidth” of communication between classes and components grows and the complexity of the interface increases
- Testing and maintenance are more difficult
- Communicating via parameter passing
- The parameters passed are only those that the recipient needs



Data coupling

- ◆ Two modules are data coupled,
- ◆ if they communicate via a parameter:
 - an elementary data item,
 - Sharing of same data.
 - Withdraw , update and check balance
Shares common data

Routine call coupling

- Occurs when one operation invokes another
- This level of coupling is common and is often quite necessary
- However, it does increase the connectedness of a system

Type use coupling

- Occurs when component A uses a data type defined in component B(e.g.,this occurs whenever “a class declares an instance variable or a local variable as having another class for its type”)
- If the type definition changes,every component that uses the definition must also change



Inclusion or import

- Occurs when component A imports or includes a package or the content of component B

External

- Occurs when a component communicates or collaborates with infrastructure components(e.g.operating system functions,database capability,telecommunication functions)
- Although this type of coupling is necessary,it should be limited to a small number of components or classes within a system

COHESION

- The measure of strength of the association of elements within a module
- It is the degree to which the responsibility of a single component form a meaningful unit
- It is a property or characteristic of an individual module

COUPLING

- The measure of interdependence of one module to another
- It describes the relationship between software components
- It is a property of a collection of modules

Benefits of high cohesion and low coupling

High Cohesion

Cohesion refers to the measure of how strongly-related the functions of a module are. Low cohesion refers to modules that have different unrelated responsibilities. High cohesion refers to modules that have functions that are similar in many aspects.

The benefits of high cohesion are

- ★ Readability – (closely) related functions are contained in a single module
- ★ Maintainability – debugging tends to be contained in a single module
- ★ Reusability – classes that have concentrated functionalities are not polluted with useless functions

Low Coupling

Coupling refers to the relationship of a module with another module. A module is said to be highly coupled with another module if changes to it will result to changes to the other module. And a module is said to be loosely coupled if a module is independent of any other modules. This can be achieved by having a stable interface that effectively hides the implementation of another module.

~~Benefits of low coupling are~~

- ★ maintainability – changes are confined in a single module
- ★ testability – modules involved in unit testing can be limited to a minimum
- ★ readability – classes that need to be analyzed are kept at a minimum

University Questions

- Explain coupling and cohesion? Explain the types of cohesion with examples.(10 marks-Dec-19)
- Explain coupling and cohesion? Explain the types of coupling with examples.(10 marks-May-19)



User Interface Design

5/25/2021

St.Francis Institute of Technology

Software Engineering

Ms. Varsha Nagpurkar

User Interface Design

- User interface is considered as a front-end application view which is interacted by the user so as to use the software
- User is able to manipulate as well as control the software and hardware with the help of user interface
- Now-a-day,user interface appears in each and every place where there is existence of digital technology,right from desktops,smart phones,cars,music players,airplanes,ships etc.

User Interface Design

- User interface is an integral component of software and is designed in such a manner that it should be able to provide the user insight of the software
- UI offers a fundamental platform for the communication of user and computer
- The form of UI can be graphical, text-based, audio-video based. It is usually depends upon the underlying platform(hardware and software combination)

User Interface Design

The popularity of software depends upon following aspects

- Attractive
- Simple to use
- Responsive in short time
- Clear to understand
- Consistent on all interfacing screens

Categories of UI

- Command Line Interface
- Graphical User Interface

Command Line Interface

- In this approach, user communicates with computer system with the help of commands
- In Unix/Linux system the command interpreter is known as shell.
- Shell is nothing but the interface between user and operating system

Graphical User Interface(GUI)

- Another approach to interface with the computer system is through a user friendly graphical user interface or GUI
- Instead of directly entering commands through a command-line interface,a GUI allows a mouse-based,window-and-menu based system as an interface

Characteristics of Good User Interface

- Clarity
- Conclusion
- Familiarity
- Responsiveness
- Consistency
- Aesthetics
- Efficiency
- Attractive
- Forgiveness

Benefits of Good Interface Design

- Higher revenue
- Increased user efficiency and satisfaction
- Reduced development costs
- Reduced support costs

The Golden Rules

- The following 3 rules form the basis for a set of user interface design principles and guidelines
 - Place the user in control
 - Reduce the user's memory load
 - Make the interface consistent



Mandel defines a number of design principles that allow the user to maintain control

- **Define interaction modes in a way that does not force a user into unnecessary or undesired actions-**
- An interaction mode is the current state of the interface
- For example, if spell check is selected in a word-processor menu, the software moves to a spell-checking mode
- There is no reason to force the user to remain in spell-checking mode if the user desires to make a small text edit along the way
- The user should be able to enter and exit the mode with little or no effort

Mandel defines a number of design principles that allow the user to maintain control

- **Provide for flexible interaction.-**
- Because different users have different interaction preferences, choices should be provided. For example, software might allow a user to interact via keyboard commands, mouse movement, a digitizer pen, a multitouch screen, or voice recognition commands. But every action is not amenable to every interaction mechanism. Consider, for example, the difficulty of using keyboard command (or voice input) to draw a complex shape.

Mandel defines a number of design principles that allow the user to maintain control

- **Allow user interaction to be interruptible and undoable.-**
- Even when involved in a sequence of actions, the user should be able to interrupt the sequence to do something else (without losing the work that had been done). The user should also be able to “undo” any action.

Mandel defines a number of design principles that allow the user to maintain control

- **Streamline interaction as skill levels advance and allow the interaction to be customized.-**
- Users often find that they perform the same sequence of interactions repeatedly. It is worthwhile to design a “macro” mechanism that enables an advanced user to customize the interface to facilitate interaction.



Mandel defines a number of design principles that allow the user to maintain control

- **Hide technical internals from the casual user-**
- The user interface should move the user into the virtual world of the application.
- The user should not be aware of the operating system, file management functions, or other arcane computing technology.

Mandel defines a number of design principles that allow the user to maintain control

- **Design for direct interaction with objects that appear on the screen-**
- The user feels a sense of control when able to manipulate the objects that are necessary to perform a task in a manner similar to what would occur if the object were a physical thing.
- For example, an application interface that allows a user to drag a document into the “trash” is an implementation of direct manipulation.

Reduce the User's Memory Load

- **Reduce demand on short-term memory-**
- When users are involved in complex tasks, the demand on short-term memory can be significant. The interface should be designed to reduce the requirement to remember past actions, inputs, and results. This can be accomplished by providing visual cues that enable a user to recognize past actions, rather than having to recall them

Reduce the User's Memory Load

- **Establish meaningful defaults.** The initial set of defaults should make sense for the average user, but a user should be able to specify individual preferences. However, a “reset” option should be available, enabling the redefinition of original default values.

Reduce the User's Memory Load

- **Define shortcuts that are intuitive-**
When mnemonics are used to accomplish a system function (e.g., alt-P to invoke the *print* function), the mnemonic should be tied to the action in a way that is easy to remember (e.g., first letter of the task to be invoked).

Reduce the User's Memory Load

- **The visual layout of the interface should be based on a real-world metaphor.** For example, a bill payment system should use a checkbook and check register metaphor to guide the user through the bill paying process. This enables the user to rely on well-understood visual cues, rather than memorizing an arcane interaction sequence.

Reduce the User's Memory Load

- **Disclose information in a progressive fashion-**
- The interface should be organized hierarchically. That is, information about a task, an object, or some behavior should be presented first at a high level of abstraction. More detail should be presented after the user indicates interest.

Make the Interface Consistent

- **Allow the user to put the current task into a meaningful context-**
- Many interfaces implement complex layers of interactions with dozens of screen images. It is important to provide indicators (e.g., window titles, graphical icons, consistent color coding) that enable the user to know the context of the work at hand. In addition, the user should be able to determine where he has come from and what alternatives exist for a transition to a new task.

Make the Interface Consistent

- **Maintain consistency across a complete product line-**
- A family of applications (i.e., a product line) should implement the same design rules so that consistency is maintained for all interaction.

Make the Interface Consistent

- If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so-
- Once a particular interactive sequence has become a de facto standard (e.g., the use of alt-S to save a file), the user expects this in every application encountered. A change (e.g., using alt-S to invoke scaling) will cause confusion.

Software Engineering(SE)

CSC 601



Subject Incharge

Varsha Nagpurkar
Assistant Professor
Room No. 407

email: varshanagpurkar@sfit.ac.in

Module-5 Syllabus

5.0	Software Risk, Configuration Management & Quality Assurance	08
5.1	Risk Identification, Risk Assessment, Risk Projection, RMMM	
5.2	Software Configuration management, SCM repositories, SCM process	
5.3	Software Quality Assurance Task and Plan, Metrics, Software Reliability, Formal Technical Review (FTR), Walkthrough	

Software Risks

- Whenever we start any business or any development process, we take into consideration the risks involved in accomplishing that task
- Similarly when software development process is started, it is main job of a software manager to look into all types of possible risks involved in the entire process
- It involves focusing on the possible risks that could affect the project development process
 - Schedule of the development process
 - Quality of the application under construction

Software Risks

- If the risk management is effective, then it becomes easier to handle all the problems and
- It is ensured that the project schedules and budget are within acceptable limits and there is no schedule slippage and ultimately no budget slippage

Software risks characteristics

- Uncertainty-the risk may or may not happen;that is,there are no 100% risks
- Loss-if the risk becomes a reality,unwanted consequences or losses will occur



Categories of risks

- Project risks
- Technical risks
- Business risks

Project risks-

- It threaten the project plan
- If project risks become real,it is likely that project schedule will slip and costs will increase
- Project risks identify potential budgetary,schedule,personnel(staffing and organization),resource,stakeholder, and requirements problems

Technical risks

- It threaten the quality and timeliness of the software to be produced
- If a technical risk becomes a reality, implementation may become difficult or impossible
- It identify potential design, implementation, interface, verification, and maintenance problems

Business risks

- Business risks threaten the viability of the software to be built
- Business risks often jeopardize(causes harm/loss) the project or the product
- Top 5 business risks are
 - Building an excellent product or system that no one really wants(market risks)
 - Building a product that no longer fits into the overall business strategy for the company(strategic risks)
 - Building a product that the sales force doesn't understand how to sell(sales risk)
 - Losing the support of senior management due to a change in focus or a change in people(management risks)
 - Losing budgetary or personnel commitment(budget risks)



General categorization of risks proposed by Charette

- Known risks- Are those that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g. unrealistic delivery date, lack of documented requirements or software scope, poor development environment)

General categorization of risks proposed by Charette

- Predictable risks-Are extrapolated from past project experience(e.g.,staff turnover,poor communication with the customer,dilution of staff effort as ongoing maintenance requests are serviced)
- Unpredictable risks-Can and do occur,but are extremely difficult to identify in advance



Risk Strategies

- **Reactive**
- **Proactive**

Reactive Risk Strategy

- It monitors the project for likely risks
- Resources are monitored to guess the possible risks and deal with them so that they do not become the problems for budget slippage and cost slippage
- Software team does nothing about risks until something goes wrong
- Then the team flies into action in an attempt to correct the problem rapidly
- This is often called a fire-fighting mode
- After failure of all these actions, the concept of “crisis management” comes into picture and the project is in deep trouble and uncertainty happens

Proactive Risk Strategy

- It is initiated when the actual technical work begins
- Potential risks are identified, and the probability and impact are assessed, so that the development team members establish an appropriate plan to manage all these risks
- The main goal to avoid the risks initially
- But all the risks cannot be avoided, therefore the development team creates a contingency plan that will control the risks in an effective manner



Risk identification

- It is a systematic attempt to specify threats to the project plan(estimates,schedule,resource loading,etc)
- By identifying known and predictable risks,the project manager takes a first step toward avoiding them when possible and controlling them when necessary

Distinct types of risks for each of the categories

- Generic risks- Are a potential threat to every software project
- Product-specific risks-Can be identified only by those with a clear understanding of the technology,the people, and the environment that is specific to the software that is to be built
- One method for identifying risks is to create a risk item checklist
- Checklist is used for risk identification and focuses on some subset of known and predictable risks

Checklist for known and predictable risks

generic subcategories

- Product size-risks associated with the overall size of the software to be built or modified
- Business impact-risks associated with constraints imposed by management or the marketplace
- Customer characteristics-risks associated with the sophistication of the customer and the developers ability to communicate with the customer in a timely manner
- Staff size and experience-risks associated with the overall technical and project experience of the software engineers who will do the work

Checklist for known and predictable risks generic subcategories

- Process definition-risks associated with the degree to which the software process has been defined and is followed by the development organization
- Development environment-risks associated with the availability and quality of the tools to be used to build the product
- Technology to be built-risks associated with the complexity of the system to be built and the “newness” of the technology that is packaged by the system

1. Risk Identification

Project
Mobile Development

- ✓ Project cost extends? **Project**
- ✓ What if mobile phones become bulky in size? **Business**
- ✓ What if later found radiations coming from mobile is harmful? **Business**
- ✓ What if call connection is difficult? **Technical**

Risk Assessment-Risk Components and Drivers

- The factors that affect the risk components are also called as risk drivers
- Performance risk-The degree of uncertainty that the product will meet its requirements and be fit for its intended use
- Cost risk-The degree of uncertainty that the project budget will be maintained
- Support risk-The degree of uncertainty that the resultant software will be easy to correct, adapt and enhance
- Schedule risk-The degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time

Impact Categories

- The impact of each risk driver on the risk component is divided into
 - Negligible
 - Marginal
 - Critical
 - Catastrophic

Risk component Class of impact	Performance	Cost	Support	Schedule
Negligible impact	If performance failure occurs, then it will cause inconvenience.	Errors can cause no serious impact on the project budget	The technical performance is not compromised. It can be supported with ease.	The delivery is before the deadline.
Marginal impact	There is some degradation in technical performance.	Some schedule slips can cause the cost within limits.	Some small degradation in performance.	The delivery is within the limits.
Critical impact	There is question mark in completion and success of the project.	The failure can cause project delays and thus financial overruns.	Some delay in project delivery due to modifications in the LOC.	There is schedule slippage due to LOC delays.
Catastrophic impact (Disastrous impact)	If performance failure occurs, then simply it is the project failure.	Failure can cause drastic cost overrun and project delays.	The final product is non-responsive and cannot be supported.	Unachievable deadline due to increased cost and financial crunches.



REDMI NOTE 5 PRO
MI DUAL CAMERA



Step 2) Analyze the impact of the risk occurring

- Once the risk is identified ,Each risk should be classified on the basis of following two parameters,
 - The **probability** of occurrence
 - The **impact** on the project
 - Prepare the risk table

Risk Projection & Building a Risk Table

- ***Risk projection***, also called ***risk estimation***, attempts to rate each risk in two ways
 - the likelihood or probability that the risk is real and
 - the consequences of the problems associated with the risk, should it occur.
- The project planner, along with other managers and technical staff, performs four risk projection activities:
 1. establish a scale that reflects the perceived likelihood of a risk,
 2. define the consequences of the risk,
 3. estimate the impact of the risk on the project and the product, and
 4. Note the overall accuracy of the risk projection so that there will be no misunderstandings

Building a Risk Table

Risk exposure is a measure of possible future loss (or losses) which may result from an activity or occurrence.

Risk	Probability	Impact	Exposure	RMMM
Text description of the risk				Risk Mitigation Monitoring & Management
Probability of occurrence				
Impact if occurs (Negligible=1...Catastrophic=4)				

Building Risk Table – table

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
•				
•				
•				

Impact values:

- 1—catastrophic
- 2—critical
- 3—marginal
- 4—negligible

**RMMM = Risk Mitigation,
Monitoring and Management Plan**



Building the Risk Table

- Estimate the **probability** of occurrence
- Estimate the **impact** on the project on a scale of 1 to 4, where
 - 4 = low impact on project success
 - 1 = catastrophic impact on project success
- Determine the exposure:
 - **Risk Exposure = Probability x Impact**



Risk Exposure Example

- **Risk identification.** Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.
- **Risk probability.** 80% (likely).
- **Risk impact.** 60 reusable software components were planned. If only 70 percent can be used, 18 components would have to be developed from scratch (in addition to other custom software that has been scheduled for development).
- Since the average component is 100 LOC and local data indicate that the software engineering cost for each LOC is \$14.00, the overall cost (impact) to develop the components would be $18 \times 100 \times 14 = \$25,200$.
- **Risk exposure.** $RE = 0.80 \times 25,200 \sim \$20,200$.

Step 3) Take COUNTER MEASURES to mitigate the risk(RMMM PLAN)

Risk response

Register Risk

Monitor and Control
Risk

Risk Mitigation, Monitoring, and Management (RMMM)

- mitigation—how can we avoid the risk? It is a problem avoidance activity
- monitoring—what factors can we track that will enable us to determine if the risk is becoming more or less likely?
- It is a project tracking activity with 3 primary objectives
 - To assess whether predicted risks do, in fact, occur
 - To ensure that aversion steps defined for the risk are being properly applied; and
 - To collect information that can be used for future risk analysis properly
- management—what contingency plans do we have if the risk becomes a reality?

Risk information sheet

Risk ID: P02-4-32

Date: 5/9/02

Prob: 80%

Impact: high

Description:

Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.

Refinement/context:

Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards.

Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.

Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.

Mitigation/monitoring:

1. Contact third party to determine conformance with design standards.
2. Press for interface standards completion; consider component structure when deciding on interface protocol.
3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.

Management/contingency plan/trigger:

RE computed to be \$20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly.

Trigger: Mitigation steps unproductive as of 7/1/02

Current status:

5/12/02: Mitigation steps initiated.

Originator: D. Gagne

Assigned: B. Laster

RISK MANAGEMENT

Risk for our project is: planned budget Increase
For risk management using RMMM plan.

Risk Information Sheet					
Project Name: Library Management System					
Risk ID: 007	Date: 13/03/2015	Probability: 67%	Impact: Medium		
Origin: Viraj Kelkar		Assigned to: Sanket Kudalkar			
Description: More bugs are arising in the system. System is unstable. Need to be updated as the technique gets modern.					
Refinement/ Context: System is unstable due to the unexpected developer. Because of such problem the solution to the critical problem is harder to find.					
Mitigation / Monitoring Bugs must be found and simultaneously solutions to those bugs should also be found. Testing of each module should be done.					
Contingency plan and trigger Experts should be hire in case of emergency or in case of issue. As computed the risk exposure to be Rs.45,000. Allocate this amount within the project management cost. Develop revised schedule and allocate more skilled staff accordingly.					
Status / date Mitigation step initiated.					
Approval Tejas Kondhalkar	Closing date 13/3/2015				

Video Link for RMMM

- <https://www.youtube.com/watch?v=Azsze5LlrOg>

Software Engineering(SE)

CSC 601



Subject Incharge

Varsha Nagpurkar
Assistant Professor
Room No. 407

email: varshanagpurkar@sfit.ac.in

Software Engineering-Syllabus

- Software Re-engineering
- Reverse Engineering

Software Re-engineering

 Clip slide

What is it?

- Consider any technology, you use it regularly.
- It is getting old, breaks too often, takes longer to repair and no longer represents the newest technology.
- If the product is hardware ,throw it away and buy newer model.
- But, if it is custom-built software,need to rebuild it.

*“Create a product with added functionality,
better performance and reliability and
improved maintainability”*

Software Re-engineering

Who does it?

- At business level, reengineering performed by business specialist.
- At software level, reengineering performed by software engineers.

Software Re-engineering-Business Process Reengineering(BPR)

 Clip slide

Business Process Reengineering (BPR)(I)

- A *business process*; is a set of logically related tasks performed to achieve a defined business outcome.
- Within the business process, people, equipment, material resources and business procedures are combined.
- Examples; Designing a new product, Hiring a new employee...Each demands a set of tasks and needs diverse resources.

Software Re-engineering-Business Process Reengineering(BPR)

Business Process Reengineering (BPR)(II)

- Overall business;
 - The business
 - business systems
 - business process
 - business sub-processes
- BPR can be applied at any level of hierarchy, but towards to upward, the risks grow dramatically.
- Most BPR efforts focus on individual processes or sub-processes.

Software Re-engineering-Business Process Reengineering(BPR)

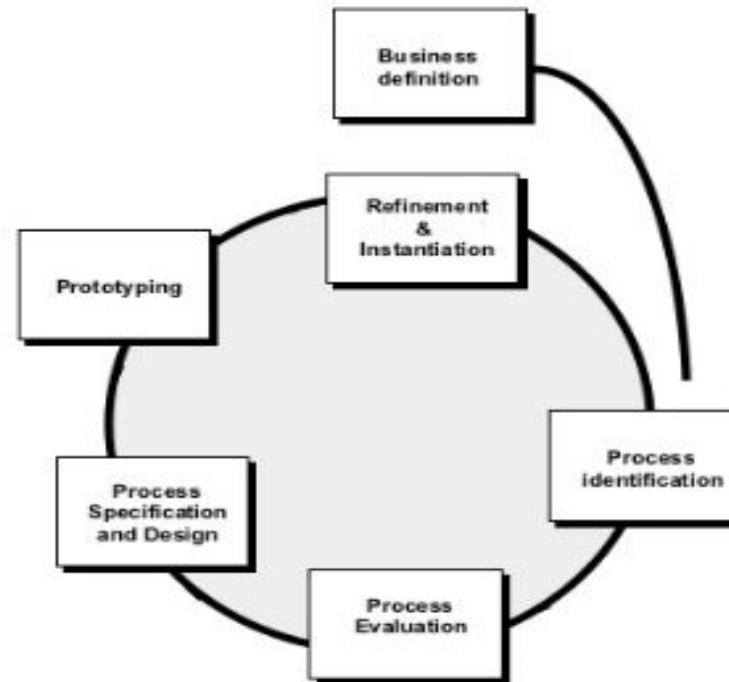
 Clip slide

BPR Model(I)

- Like most engineering activities, business process reengineering is iterative.
- There is no start and end to BPR.
- It is an evolutionary process.
- The model defines the six activities.

Software Re-engineering-Business Process Reengineering(BPR)

BPR Model(II)



Software Re-engineering-Business Process Reengineering(BPR)

BPR Model(III)

- **Business redefinition**
 - business goals identified in the context of key drivers
 - cost reduction
 - time reduction
 - quality improvement
 - empowerment
- **Process identification**
 - processes that are critical to achieving business goals are identified and prioritized
- **Process evaluation**
 - existing processes are analyzed and measured
 - process costs and time are noted
 - quality/performance problems are isolated

Software Re-engineering-Business Process Reengineering(BPR)

BPR Model(IV)

- **Process specification and design**
 - based on information obtained during three BPR activities, use-cases are prepared for each process to be redesigned
 - new tasks are designed for each process
- **Prototyping**
 - used to test redesigned processes before integrating them into the business
- **Refinement and instantiation**
 - based on feedback from the prototype, business processes are refined
 - refined processes then instantiated within a business system

Software Re-engineering-Business Process Reengineering(BPR)

Software Reengineering

- Much of the software we depend on today is average 10 to 15 years old.
- Even when these programs were created using the best design and coding techniques known at the time(and most were not), they were created when program size and storage space were principle concerns.
- They were then migrated to new platforms.
- Adjusted for changes in machine and operating system technology.
- Enhanced to meet new users.

“The result is poorly designed structures, poor coding, poor logic and poor documentation”

Software Re-engineering-Process Model

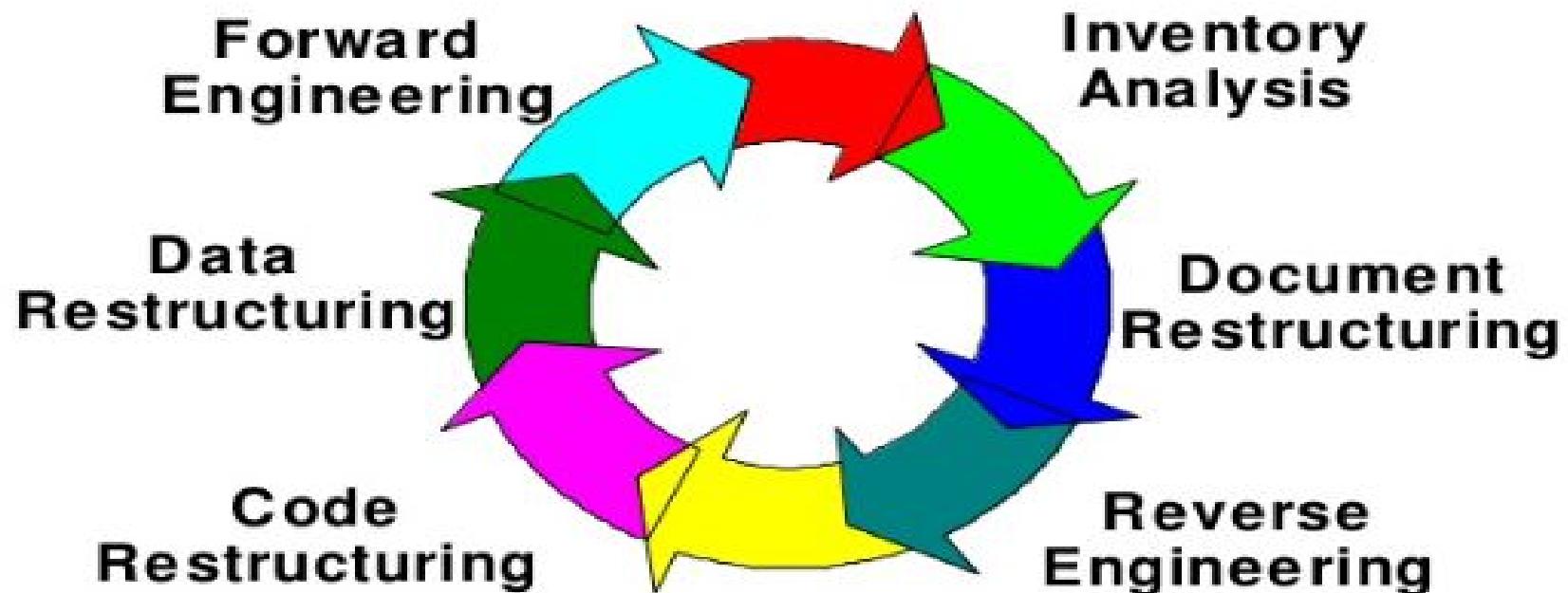
Software Reengineering Process Model(I)

- Reengineering takes time.
- It costs significant amounts of money, it absorbs resources.
- So, it is accomplished in a few months or even a few years.

Software Re-engineering-Process Model

 Clip slide

Software Reengineering Process Model(II)



Software Re-engineering-Process Model

Inventory Analysis

- **Build a table that contains all applications**
- **Establish a list of criteria, e.g.,**
 - name of the application
 - year it was originally created
 - number of substantive changes made to it
 - total effort applied to make these changes
 - date of last substantive change
 - effort applied to make the last change
 - system(s) in which it resides
 - applications to which it interfaces, ...
- **Analyze and prioritize to select candidates for reengineering**

Software Re-engineering-Process Model

Document Restructuring

 Clip slide

- Options range from *doing nothing* to *regeneration of all documentation* for critical system.
 - “Creating is far too time consuming”. In some cases correct approach. It is not possible to recreate documentation for hundreds of computer programs.
 - “Documentation must be updated, but have limited resources”. It may not be necessary to fully redocument an application. Rather then portions of the system that are currently undergoing change are fully documented.
 - “The system is business critical and must be fully redocumented.”

Reverse Engineering(I)

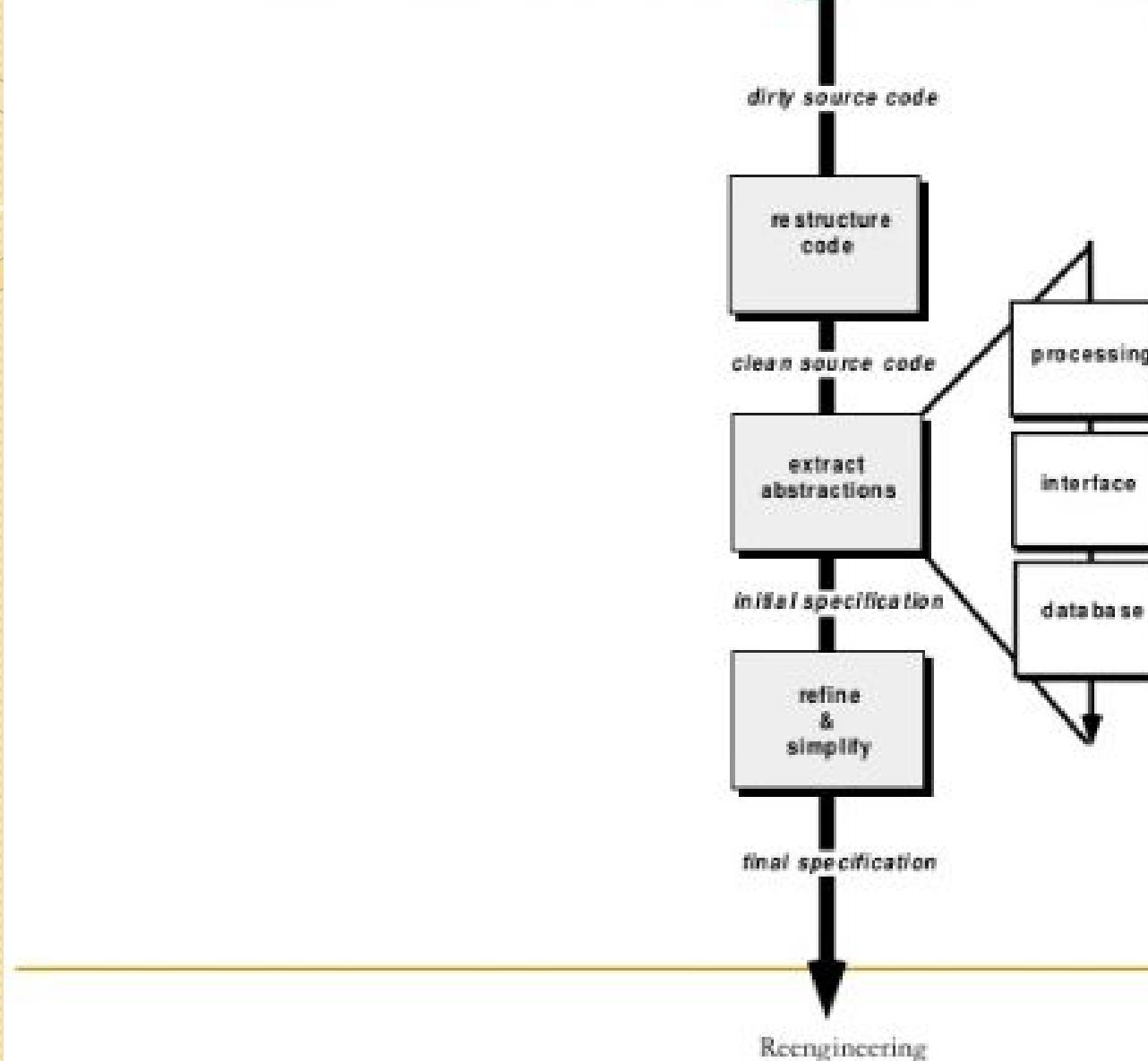
- The term “Reverse Engineering” has its origins in the hardware world.
- A company disassembles a competitive hardware product, to understand its competitor’s design and manufacturing secrets.
- For software quite similar. But not a competitor’s software. It is company’s own program.

“Anayzing a program in an effort to create a representation of the program at a higher level of abstraction than source code”

Reverse Engineering(II)

- Reverse engineering conjures a “*magic slot*”. We feed undocumented source listing into the slot and out the other end comes full documentation for the computer program.
- Reverse engineering can extract design information from source code, but some properties exist;
 - **Abstraction Level** : Sophistication of design information that can be extracted from source code. Should be as high as possible.
 - **Completeness** : Amount of detail gathered at the abstraction level. Decreases as abstraction level increases.
 - **Directionality**

Reverse Engineering(III)



Reverse Engineering(IV)

■ R.E. To Understand Processing

- ❑ The overall functionality of the entire application must be understood.
- ❑ A block diagram, representing the interaction between functional abstractions is created.
- ❑ Things become more complex when the code inside a component is considered.
- ❑ In almost every component has a section of code prepares data for processing, a different section of code does processing and another section which prepares the result.
- ❑ For large system CASE tools are used to parse the semantics of existing code.

Reverse Engineering(V)

■ R.E. To Understand Data

- Internal data structures
 - Definition of classes
 - Identify local data structures that record information about global data structures
 - Identify relationships between these local data structures & the global data structures
 - Group variables that have logical connections
- Database structure
 - Build initial object model
 - Determine candidate keys
 - Refine tentative classes – possibly combine to single class
 - Define generalizations
 - Discover associations

Reverse Engineering(VI)

■ R.E. User Interfaces

- To fully understand an existing user interface, the structure and behavior of the interface must be specified.
 - What are the basic actions?(keystrokes, mouse clicks...)
 - What is the response of the system to these actions.
- It is important to note that a replacement GUI may not mirror old interface. In fact, it may be radically different.

Restructuring(I)

- **Does not modify the overall architecture**
 - Code Restructuring
 - Data Restructuring
- **Advantages**
 - Adhere to modern software engineering standards.
 - Frustration among software engineers reduced.
 - Maintenance efforts reduced.
 - Easier to test & debug.

Restructuring(II)

■ Code Restructuring

- The most common type of restructuring.
- If modules were coded in a way that makes them difficult to understand, test and maintain.
- To accomplish this activity
 - The source code is analyzed using a restructuring tool
 - Violations of structured programming constructs are noted.
 - Code then restructured manual or automatically.
 - The resultant restructured code is reviewed and tested.

Restructuring(III)

Code Restructuring

```
Start: Get (Time-on, Time-off, Time, Setting, Temp, Switch)
    if Switch = off goto off
    if Switch = on goto on
    goto Cntrld

off: if Heating-status = on goto Sw-off
    goto loop

on: if Heating-status = off goto Sw-on
    goto loop

Cntrld: if Time = Time-on goto on
    if Time = Time-off goto off
    if Time < Time-on goto Start
    if Time > Time-off goto Start
    if Temp > Setting then goto off
    if Temp < Setting then goto on

Sw-off: Heating-status := off
    goto Switch

Sw-on: Heating-status := on
Switch: Switch-heating
loop: goto Start
```

Spaghetti Control Logic

```
loop
    - The Get statement finds values for the given variables from the system's
    - environment.

    Get (Time-on, Time-off, Time, Setting, Temp, Switch) ;
    case Switch of
        when On => if Heating-status = off then
            Switch-heating ; Heating-status := on ;
            end if ;
        when Off => if Heating-status = on then
            Switch-heating ; Heating-status := off ;
            end if ;
        when Controlled =>
            if Time >= Time-on and Time <= Time-off then
                if Temp > Setting and Heating-status = on then
                    Switch-heating; Heating-status = off;
                elseif Temp < Setting and Heating-status = off then
                    Switch-heating; Heating-status := on ;
                    end if;
                end if;
            end case ;
    end loop ;
```

Structured Control Logic

Restructuring(IV)

■ Data Restructuring

- Firstly “Data Analysis” must be done
 - Current data architecture is examined in detail and data models are defined.
 - Data objects and attributes identified, and existing data structures are reviewed for quality.
- Then “Data Redesign” commences.
 - Clarifies data definitions to achieve consistency among data item names and physical record formats.
- “Data Name Rationalization”
 - Ensures all data naming conventions conform to local standard.

Forward Engineering (I)

- Also called “renovation” or “reclamation”
- Uses design information from existing software to alter the existing system in an effort to improve its overall quality.

Forward Engineering (II)

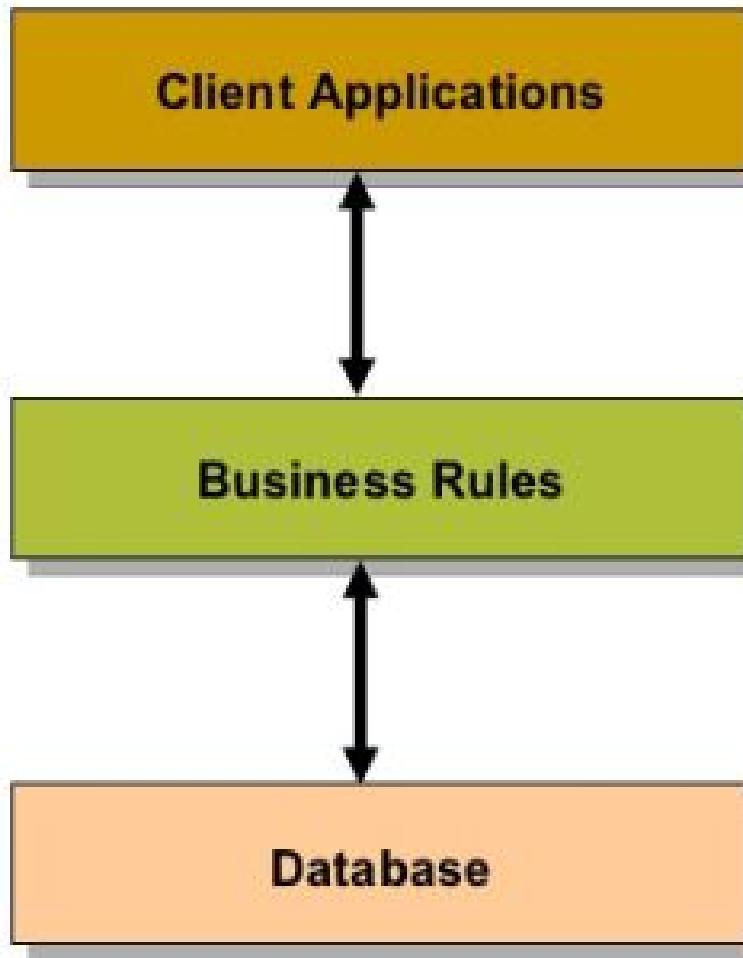
■ Options for poorly designed source code

- Change necessary lines of source code
- Understand bigger picture & use this to guide modifications
- Redesign, recode, & test portions that need modification
- Completely redesign, recode, and test system

Forward Engineering (III)

- **F.E. For Client/server architectures**
 - Distribution of resources.
 - Migrate application functionality to client.
 - New GUI interfaces are implemented at client.
 - Database functions are allocated at server.
 - Specialized functionality may remain at the server site.
 - New communications, security and control.
requirement must be established at both the client & server.
- **Three layers of abstractions can be identified**

Forward Engineering (IV)



-Implement business functions that required by specific group of end-users

-Represents software at both the client and server.
-Software performs control and coordination tasks to ensure the transactions between client and server.

-Manages transactions and queries from server applications

Forward Engineering (V)

■ F.E. For Object-Oriented Architectures

- Existing software is reverse engineered so that appropriate data, functional and behavioral models can be created.
- Data models are used to establish the basis for the definition of classes.
- Then class hierarchies, object relationship models are defined.

Forward Engineering (VI)

■ Forward Engineering User interfaces

- ❑ Understand original interface & data
- ❑ Remodel behavior of old interface into abstractions in context of GUI
- ❑ Add improvements that make the mode of interaction more efficient.
- ❑ Build and integrate new GUI.

Software Engineering(SE)

CSC 601



Subject Incharge

Varsha Nagpurkar
Assistant Professor
Room No. 407

email: varshanagpurkar@sfit.ac.in



How to Write Test Cases

What is a Test Case?

- A **TEST CASE** is a set of actions executed to verify a particular feature or functionality of your software application.
- A Test Case contains test steps, test data, precondition, postcondition developed for specific test scenario to verify any requirement.
- The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements of the customer.

Test Scenario Vs Test Case

- Test scenarios are rather vague and cover a wide range of possibilities
- For a Test Scenario: Check Login Functionality there many possible test cases are:
 - Test Case 1: Check results on entering valid User Id & Password
 - Test Case 2: Check results on entering Invalid User ID & Password
 - Test Case 3: Check response when a User ID is Empty & Login Button is pressed, and many more

How to Write Test Cases in Manual Testing

- Test Case for the scenario: Check Login Functionality

ALREADY REGISTERED?

Email address

Password

Forgot your password?

Sign in

Check Login Functionality

Step 1) A simple test case to explain the scenario would be

Test Case #	Test Case Description
1	Check response when valid email and password is entered

Step 2) In order to execute the test case, you would need Test Data. Adding it below

Test Case #	Test Case Description	Test Data
1	Check response when valid email and password is entered	Email: guru99@email.com Password: INf9^Oti7^2h

Step 3) In order to execute a test case, a tester needs to perform a specific set of actions on the AUT. This is documented as below:

Test Case Case #	Test Case Description	Test Steps	Test Data
1	Check response when valid email and password is entered	1) Enter Email Address 2) Enter Password 3) Click Sign in	Email: guru99@email.com Password: INf9^Oti7^2h

Step 4) The goal of test cases in software testing is to check behavior of the AUT for an expected result. This needs to be documented as below

Test Case #	Test Case Description	Test Data	Expected Result
1	Check response when valid email and password is entered	Email: guru99@email.com Password: INf9^Oti7^2h	Login should be successful

Step 5

Test Case #	Test Case Description	Test Data	Expected Result	Actual Result	Pass/Fail
1	Check response when valid email and password is entered	Email: guru99@email.com Password: lNf9^Oti7^2h	Login should be successful	Login was successful	Pass

Software Engineering(SE)

CSC 601



Subject Incharge

Varsha Nagpurkar

Assistant Professor

Room No. 407

email: varshanagpurkar@sfit.ac.in

CHAPTER 6

Software Testing



Sweetheart, I am sorry. They are just shipping
one bug with this release. You and the kids can
join me in service pack1.

Module-6

6.0	Software Testing and Maintenance	10
6.1	Strategic Approach to Software Testing, Unit testing, Integration testing Verification, Validation Testing, System Testing	
6.2	Software Testing Fundamentals, White-Box Testing, Basis Path Testing, Control Structure Testing, Black-Box Testing,	
6.3	Software maintenance and its types, Software Re-engineering, Reverse Engineering	

*

A Strategic approach to Software Testing

- Testing is a set of activities that can be planned in advance and conducted systematically
- For implementing testing ,there are various testing strategies defined in software engineering literature
- All these strategies provide a testing template to the developers
- The testing templates should possess following characteristics that is applicable to any software development process

*

Characteristics of Testing Template

- For effective testing, formal technical reviews must be conducted by the development team. Frequent communication between developer and customer resolves most of the complexities and confusion in the beginning itself. Thus before start of the testing process, number of errors may be uncovered and then fixed.
- Testing begins at the component level and works “outward” toward the integration of the entire computer-based system.
- Different testing techniques are appropriate at different points in time.
- Testing is conducted by the developer of the software and (for large projects) an independent test group.
- Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

What's our target for Testing?

Expected System



Actual System



Are the expected behaviour (specified in requirement specification and system models) the same as the observed behaviour of the actual system?

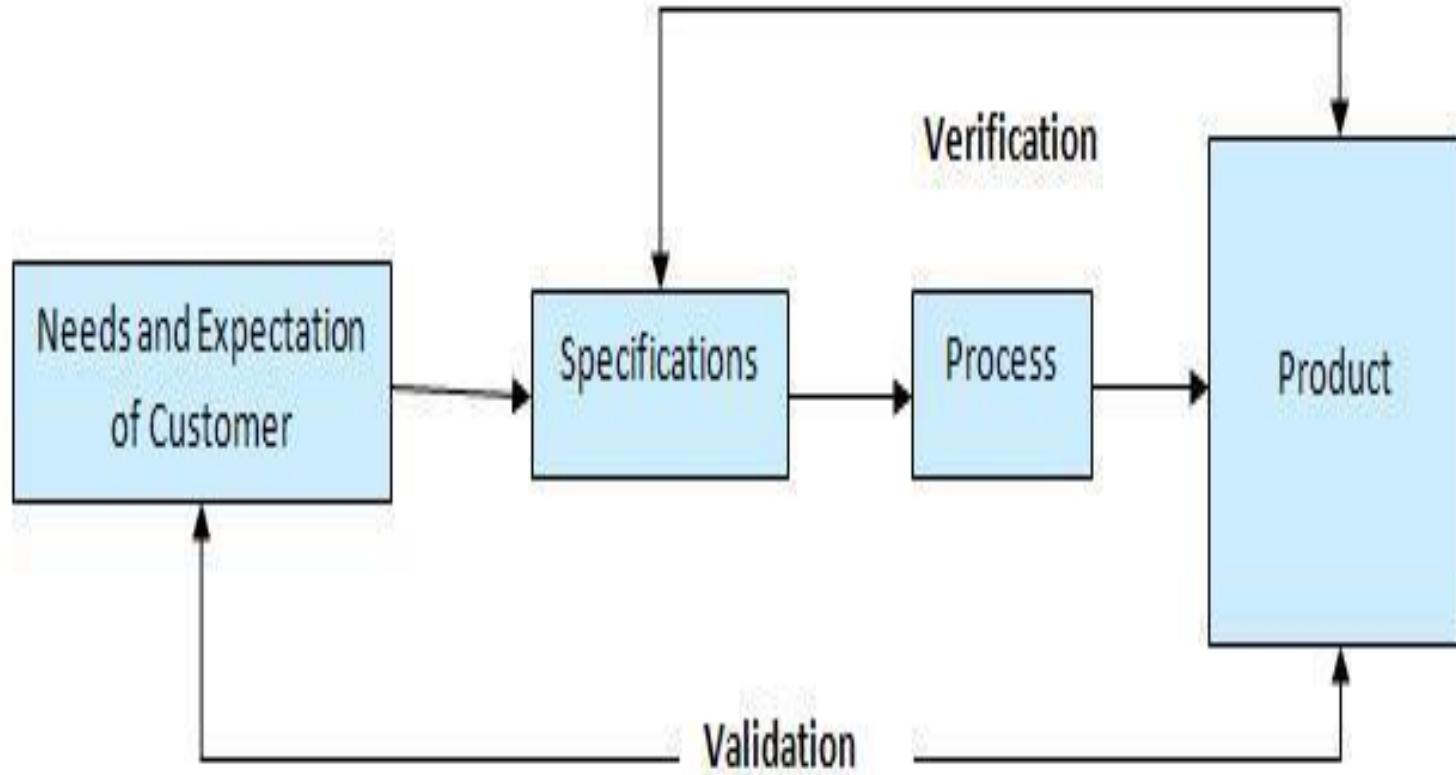
V & V

Verification refers to the set of activities/tasks that ensure that software correctly implements a specific function.

Validation refers to a different set of activities/tasks that ensure that the software that has been built is traceable to customer requirements. Boehm [Boe81] states this another way:

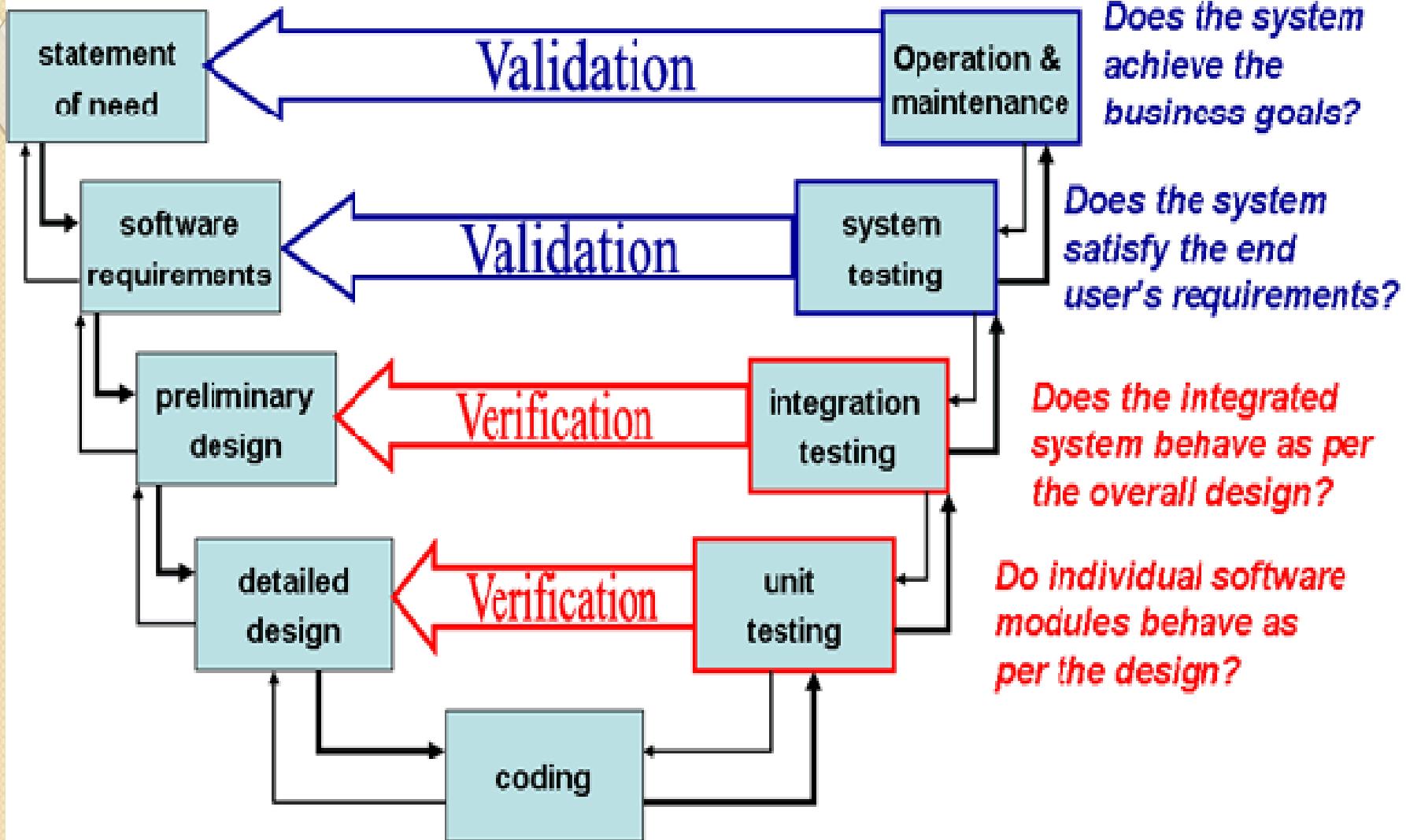
Verification: "Are we building the product right?"

Validation: "Are we building the right product?"



VERIFICATION	VALIDATION
1. Verification is the process of determining whether or not a product of given phase of software development fulfil the support specification.	2. Validation is the process of evaluating software at the end of software development process to determine whether it is in accordance with the software requirement.
2. It does not involve executing the code.	2. It always involves executing the code.
3. Verification evaluates each phase end product in order to ensure concurrency.	3. Validation test software specification in order to ensure its user requirements
4. Verification uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.	4. Validation uses methods like black box (functional) testing, glass box testing, and white box (structural) testing etc.
5. Verification is to check whether the software conforms to specifications.	5. Validation is to check whether software meets the customer expectations and requirements.
6. Verification ensure that each step in the process off software development yield the right product.	6. Validation ensures that the software being developed will satisfy functional and other requirements.
7. Target is requirements specification, application and software architecture, high level, complete design, and database design etc.	7. Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.
8. Verification process is done at early stage.	8. Validation process is done at Late (End) stage.
9. It generally comes first-done before validation.	9. It generally follows after verification.
10. Verification is done by QA team to ensure that the software is as per the specifications in the SRS document.	10. Validation is carried out with the involvement of testing team.
11. Verification is a static practice of verifying documents, design, code and program.	11. Validation is a dynamic mechanism of validating and testing the actual product.
12. It is human based checking of documents and files.	12. It is computer based execution of program.
13. It can catch errors that validation cannot catch. It is low level exercise.	13. It can catch errors that verification cannot catch. It is High Level Exercise.

Dynamic Testing



Validation

Validation concentrates on the big picture of whether the software can do what the user wants.

The focus is on seeing if the software is suitable for a ***“specific intended use or application”***.

The Validation diagram shows that at whatever stage of development you are you need to check back to the System Specification, User Requirements and Business Case to see if it meets the purpose.

The V-model diagram shows this also includes when tests are developed, because the user is interested in what the software does, and these are the most important tests for them.

So what qualities do we measure?

Correctness (Functional Requirements)

Reliability (Non-functional Requirements)

Robustness

Security

Performance

Usability

Unit Testing

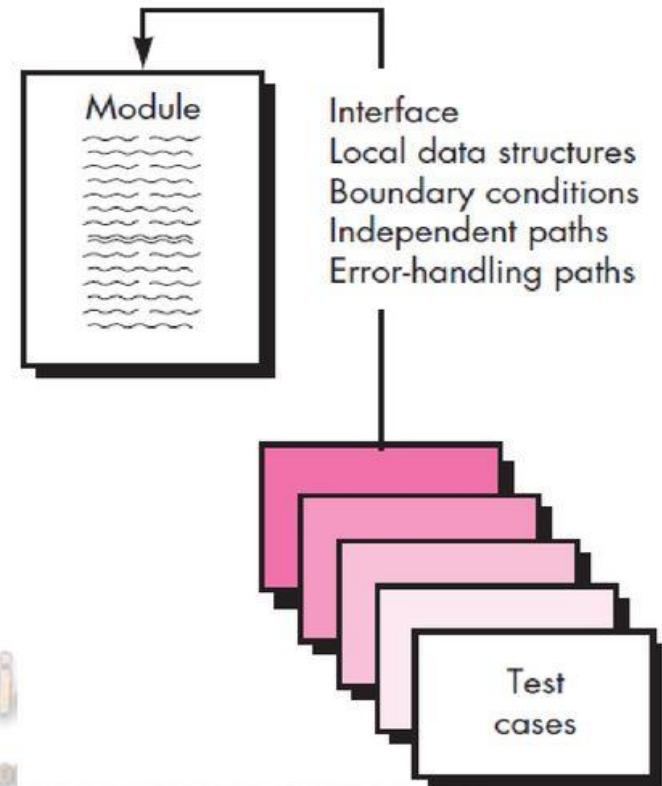
- It focuses verification effort on the smallest unit of software design-the software component or module
- Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module
- It focuses on the internal processing logic and data structures within the boundaries of a component
- This type of testing can be conducted in parallel for multiple components

Unit test considerations

Test Strategies for Conventional Software

Unit Test Considerations

1. Module interface - - tested
2. Local data structures r examined
3. All independent paths thru control structure r exercised
4. Boundary conditions r tested
5. All error-handling paths r tested.



www.jkmateri

www.jkdirectory.blogspot.com

www.jktrafficrules.weebly.com

Unit test considerations

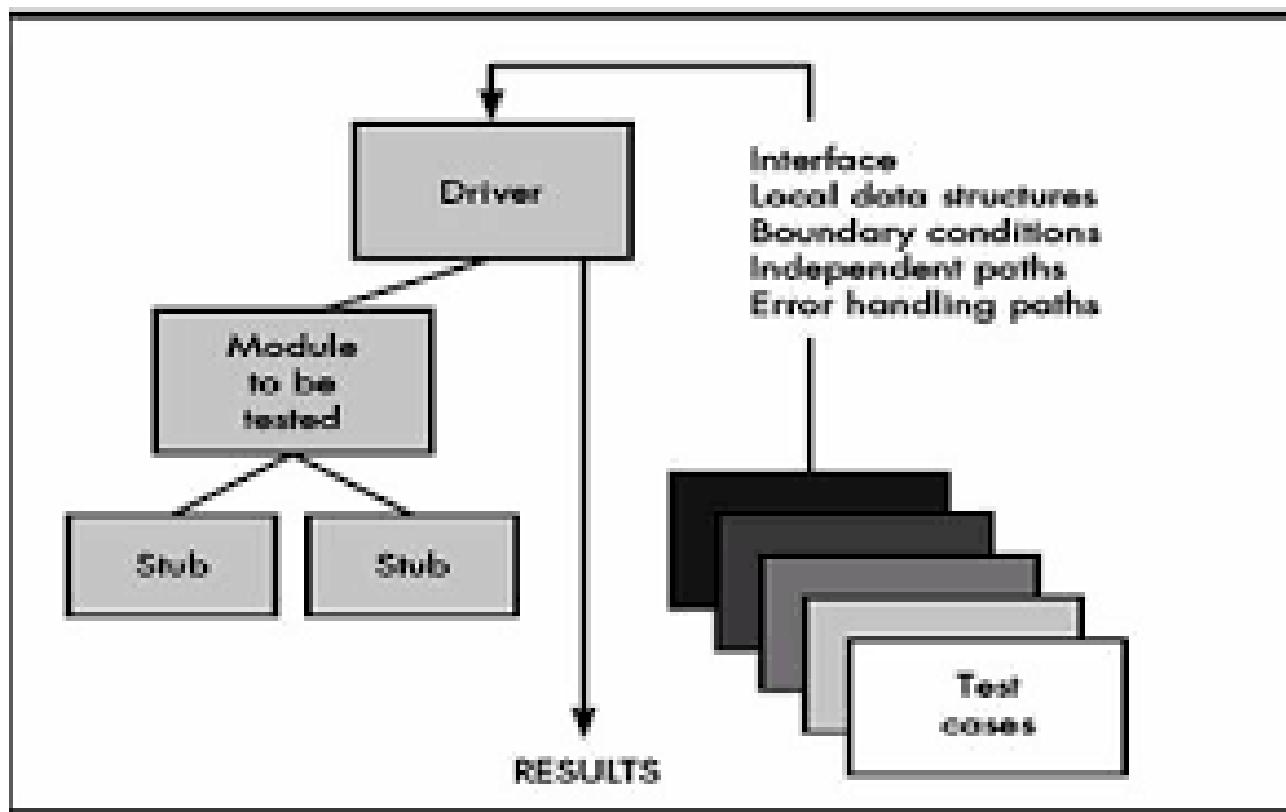
- The Module interface is tested to ensure that information properly flows into and out of the program unit under test
- Local data structures are examined to ensure its integrity in the execution of the algorithm
- All independent paths(basis paths) through the control structure are evaluated to ensure that all statements in a module have been executed at least once
- Boundary conditions are tested to ensure that the module operates properly within boundaries or within limit s
- All error handling paths are tested

Unit test procedures

- Unit testing is normally considered as an adjunct to the coding step
- A review of design information provides guidance for establishing test cases that are likely to uncover errors

*

Unit test environment



Unit test environment

- In most applications a driver is nothing but a “main program” that accepts test case data, passes such data to the component(to be tested),and prints relevant results
- Stubs serve to replace modules that are subordinate to(called by) the component to be tested

INTEGRATION TESTING

- It is a level of software testing where individual units / components are combined and tested as a group.
- The purpose of this level of testing is to expose faults in the interaction between integrated units.
- Test drivers and test stubs are used to assist in Integration Testing.

*

Integration Testing

- It is a systematic technique for constructing software architecture while at the same time conducting tests to uncover errors associated with interfacing
- The objective is to take unit tested components and build a program structure that has been dictated by design
- It is often observed that there is a tendency to attempt always non-incremental approaches. All the components are integrated in the beginning and the program is tested as a whole

*

Integration Testing

- In this approach a set of errors are encountered and correction seems to be very difficult due to isolation of causes and the complications by program since program is expanded over several modules
- After correction of these errors,some new may appear and the process continues,like an infinite loop
- Small increments of the program are tested in an incremental approach
- In incremental integration approach,the error detection and correction is quite simple

Different Incremental Integration Strategies

- Top-down integration
- Bottom-up integration

Top down Integration Testing Approach

- It is an incremental approach to construction of the software architecture
- Modules are integrated by moving downward through the control hierarchy, beginning with the main control module(main program)
- Modules subordinate to the main control module are incorporated into the structure in either a depth-first or breadth-first manner

*

Depth-first Integration

- Depth-first integration integrates all components on a major control path of the program structure
- Selection of a major path is somewhat arbitrary and depends on application-specific characteristics
- For example, selecting the left hand path ,components M1,M2,M5 would be integrated first
- Next,M8 or(if necessary for proper functioning of M2)M6 would be integrated
- Then, the central and right-hand control paths are built

Breadth-first Integration

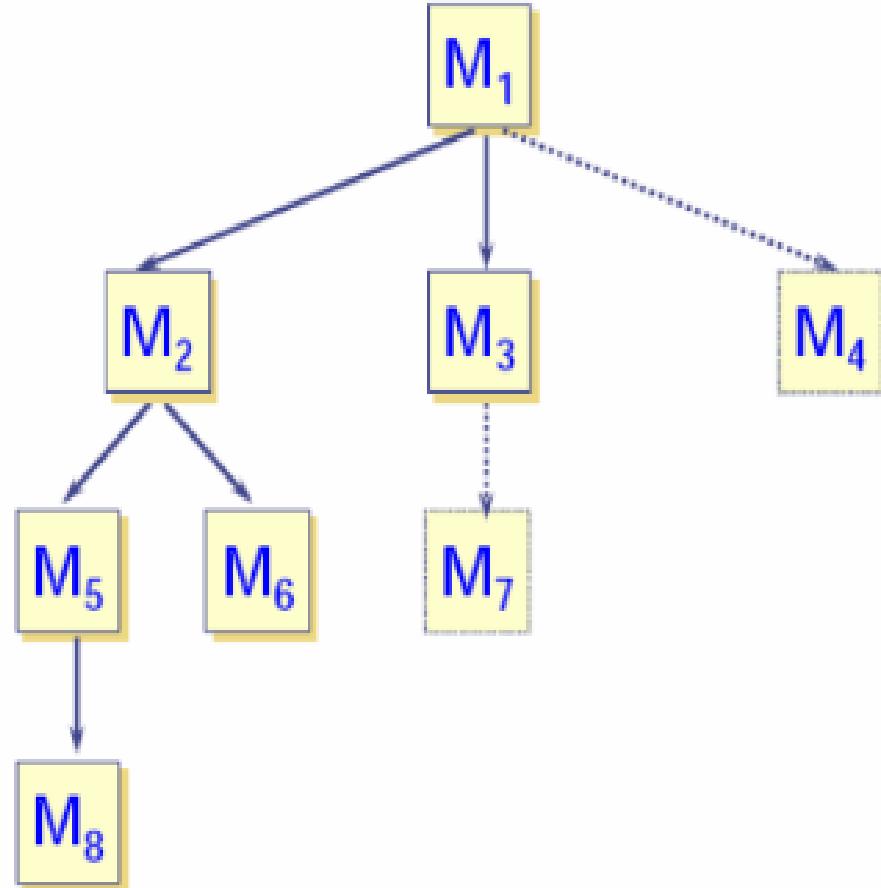
- It incorporates all components directly subordinate at each level,moving across the structure horizontally
- From the figure,components M2,M3, and M4 would be integrated first
- The next control level,M5,M6 and so on,follows

› Top-Down Testing with Depth-First

- M1, M2, M5, M8
- M6
- M3, M7
- M4

› Top-Down Testing with Breath-First

- M1
- M2, M3, M4
- M5, M6, M7
- M8



*

Integration Testing: Top-Down

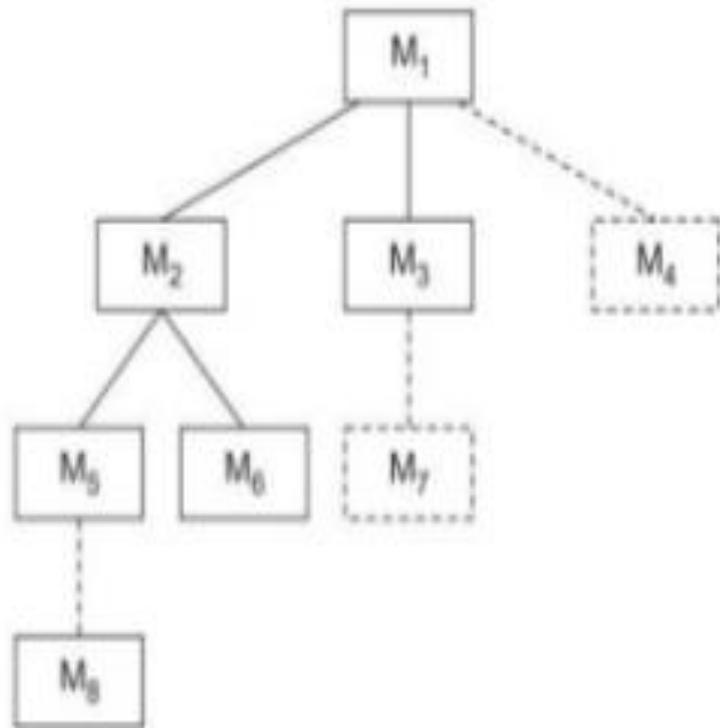
- Modules are integrated by moving downward through the control hierarchy beginning with the main control module.

- **Depth-first integration:**

M1, M2, and M5 would be integrated first. Next, M8 or M6 would be integrated. Then, the central and right-hand control paths are built.

- **Breadth-first integration:**

M2, M3, and M4 would be integrated first. The next control level M5, M6, and so on follows.



Top-Down Integration Testing

Bottom-up integration

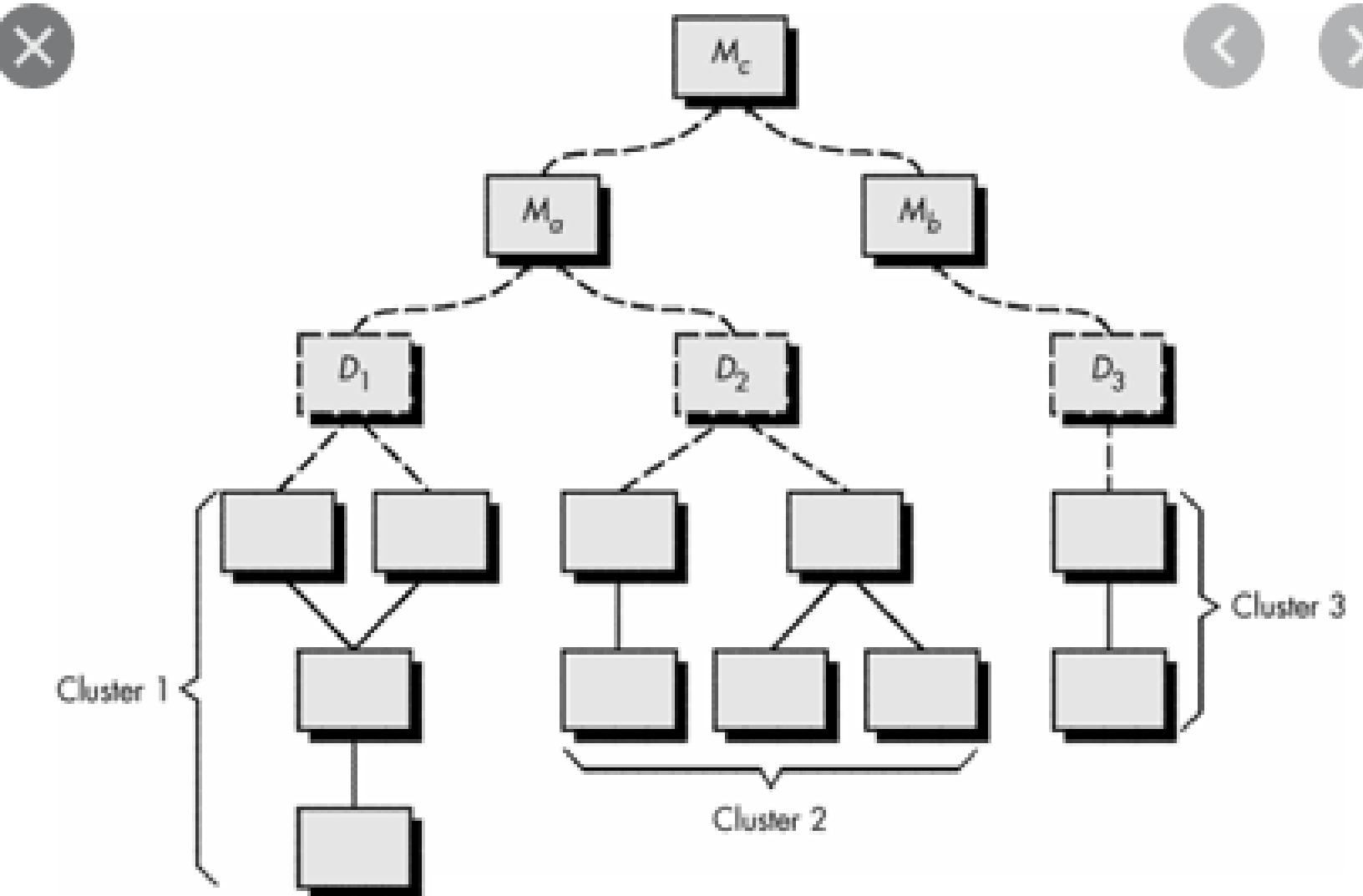
- As the name implies,begins construction and testing with atomic modules(i.e.,components at the lowest levels in the program structure)
- Because components are integrated from the bottom up,processing required for components subordinate to a given level is always available and the need for stubs is eliminated
- A bottom-up integration strategy may be implemented with the following steps

Bottom-up integration

- Low-level components are combined into clusters(sometimes called builds) that perform a specific software subfunction
- A driver(a control program for testing) is written to coordinate test case input and output
- The cluster is tested
- Drivers are removed and clusters are combined moving upward in the program structure

*

Bottom-up integration



Bottom-up integration

- Components are combined to form clusters I,2 and 3.
- Each of the clusters is tested using a driver(shown as a dashed block)
- Components in clusters I and 2 are subordinate to Ma
- Drivers D1 and D2 are removed and the clusters are interfaced directly to Ma
- Similarly,driver D3 for cluster 3 is removed prior to integration with module Mb
- Both Ma and Mb will ultimately be integrated with component Mc

Top-down approach

Undifferentiated and "one-way" approach.

Targets given by executives.

Employers follow instructions (push).

Typically, long execution time.

Typically, short-term impact on company's status quo.

Bottom-up approach

Specific and analytic approach.

Executives set the direction and define the mission.

Employers involvement in identifying and delivering optimizations (share).

Typically, short execution time.

Strong impact on company's *status quo* in the long-term. Tangible and lasting results.

System Testing

- System testing is actually a series of different tests whose purpose is to fully exercise the computer-based system
- Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions

*

Recovery Testing

- Computer-based systems must recover from faults and resume processing within a prescribed time
- In some cases, a system must be fault tolerant; that is processing faults must not cause overall system function to cease
- In other cases, a system failure must be corrected within a specified period of time or severe economic damage will occur



Recovery Testing

- ⌘ Force the software to fail in a variety of ways
- ⌘ Verify that recovery is properly performed
- ⌘ If recovery is automatic, re-initialisation, checkpointing mechanisms, data recovery, and restart are evaluated for correctness
- ⌘ If recovery requires human intervention, mean time to repair is evaluated

Security Testing

- All the computer-based systems that handles sensitive information or causes the actions that can harm or benefit any individuals is tested for illegal attacks to the system
- The examples of illegal activities are:
 - Hackers who try to attack systems for sport
 - Detained employees who try to attack the system for revenge
 - Some mischievous individuals who try to attack system for illicit personal gain
- Security testing verifies whether a protection mechanism is built or not. This mechanism should protect the system from illegal attacks and unauthorized access

*

Security Testing

- In security testing, the tester may work as an intruder that tries to attack the system to check its security
- The tester can do anything to check the system's security.
- For example
 - the tester may try to obtain passwords and may attack the system to break down the defences that were constructed

*

Security Testing

- Overload the system by series of requests to cause denial of service
- Purposely cause system errors
- Attack the system during recovery
- Browse through insecure data
- Find the key to enter into the system



Security Testing

- ⌘ Verify that protection mechanisms built into the system will protect it from improper penetration
- ⌘ Tester plays the role(s) of individual who desires to penetrate the system
- ⌘ System designer to make penetration cost greater than the value of information obtained

Stress Testing

- It executes a system in a manner that demands resources in abnormal quantity,frequency,or volume
- For example,(1)special tests may be designed that generate ten interrupts per second,when one or two is the average rate,(2)input data rates may be increased by an order of magnitude to determine how input functions will respond,(3)test cases that require maximum memory or other resources are executed,(4)test cases that may cause memory management problems are designed,(5)test cases that may cause excessive hunting for disk-resident data are created

*

Performance Testing

- It is designed to test the run-time performance of software within the context of an integrated system
- It occurs throughout all steps in the testing process
- Even at the unit level, the performance of an individual module may be assessed as tests are conducted
- Performance tests are often coupled with stress testing and usually require both hardware and software instrumentation
- That is often necessary to measure resource utilization (e.g., processor cycles)
- External instrumentation can monitor execution intervals, log events (e.g., interrupts) as they occur, and sample machine states on a regular basis

Performance Testing

- It is generally conducted for real-time and embedded systems
- If the software is not functioning as per the requirements, then it is unacceptable

Testing Techniques

- White-Box Testing
- Black-Box Testing

*

White-Box Testing

- Knowing the internal workings of a product, tests can be conducted to ensure that internal operations are performed according to specifications, and all internal components have been adequately exercised
- Sometimes called as glass-box testing, that uses the control structure described as part of component-level design to derive test cases
- In this technique, the software engineer can derive test cases that (1) guarantee that all independent paths within a module have been exercised at least once, (2) exercise all logical decisions on their true and false side,

*

White-Box Testing

- (3) execute all loops at their boundaries and within their operational bounds, and (4) exercise internal data structures to ensure their validity

*

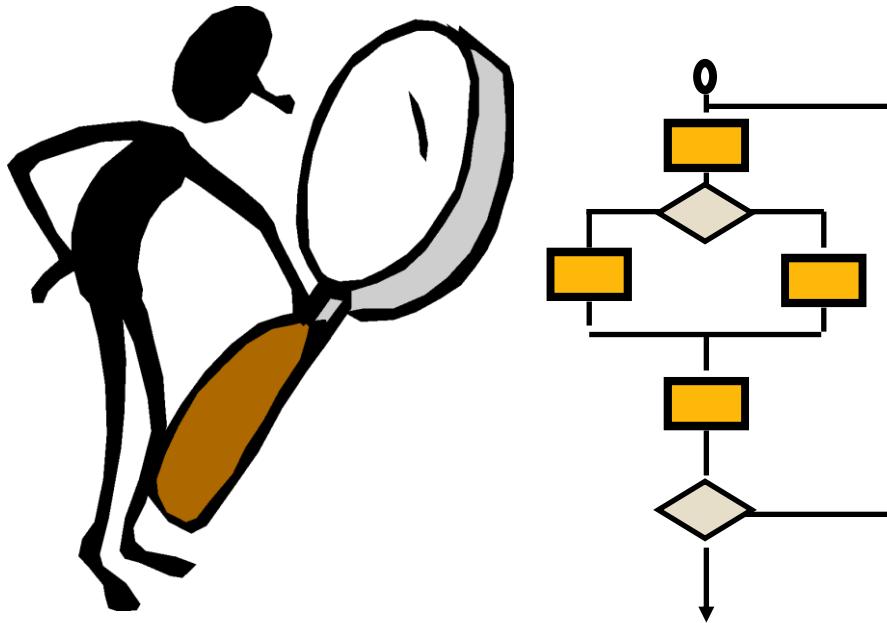
Black-Box Testing

- Knowing the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function



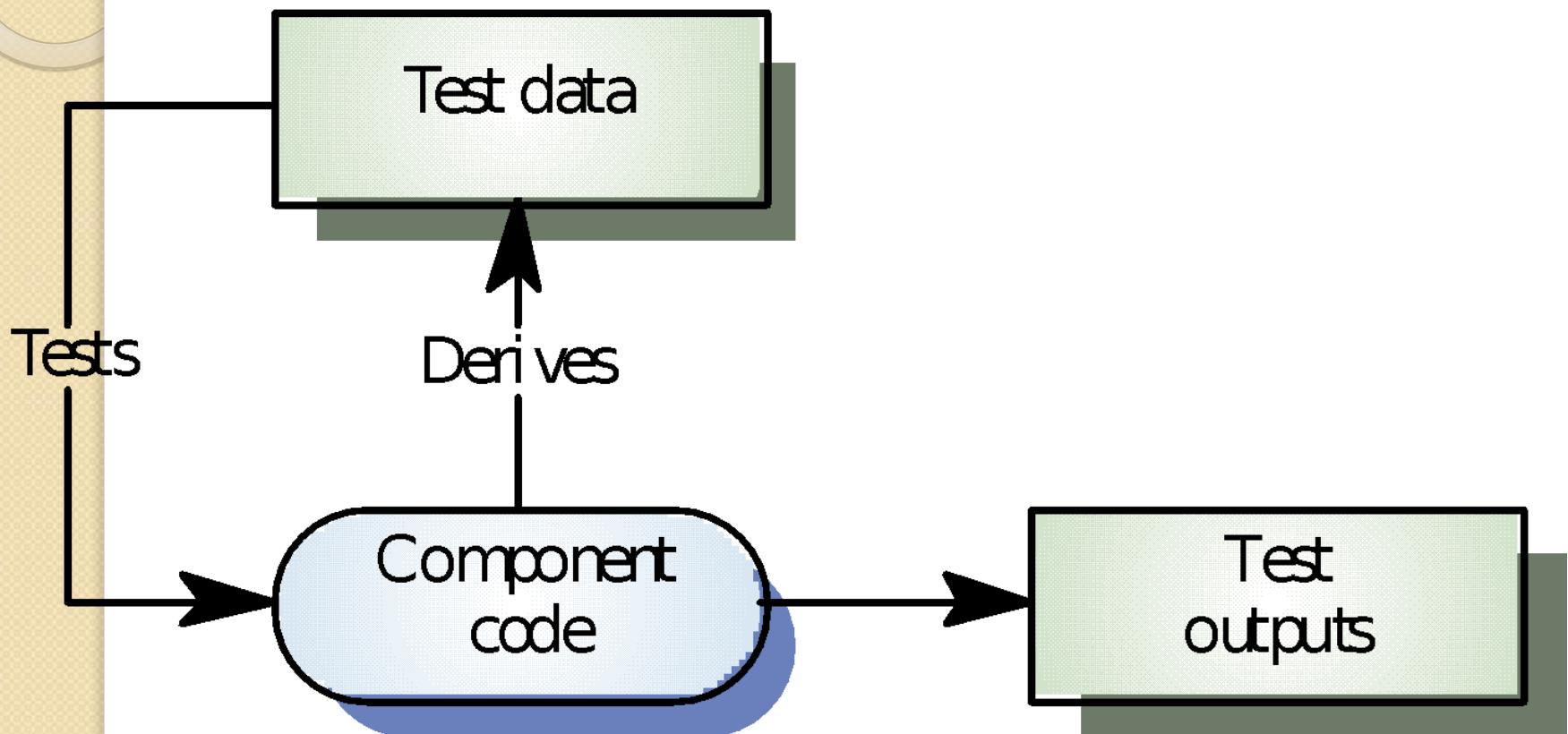
White-box Testing

White-Box Testing



... Our goal is to ensure that all statements and conditions have been executed at least once.

White-box testing



White-box Testing

Uses the control structure part of **component-level** design to derive the test cases

These test cases

- Guarantee that **all independent paths within a module have been exercised at least once**
- Exercise all logical decisions on their true and false sides
- Execute all loops at their boundaries and within their operational bounds, and
- Exercise internal data structures to ensure their validity

Basis Path Testing

White-box testing technique proposed by Tom McCabe

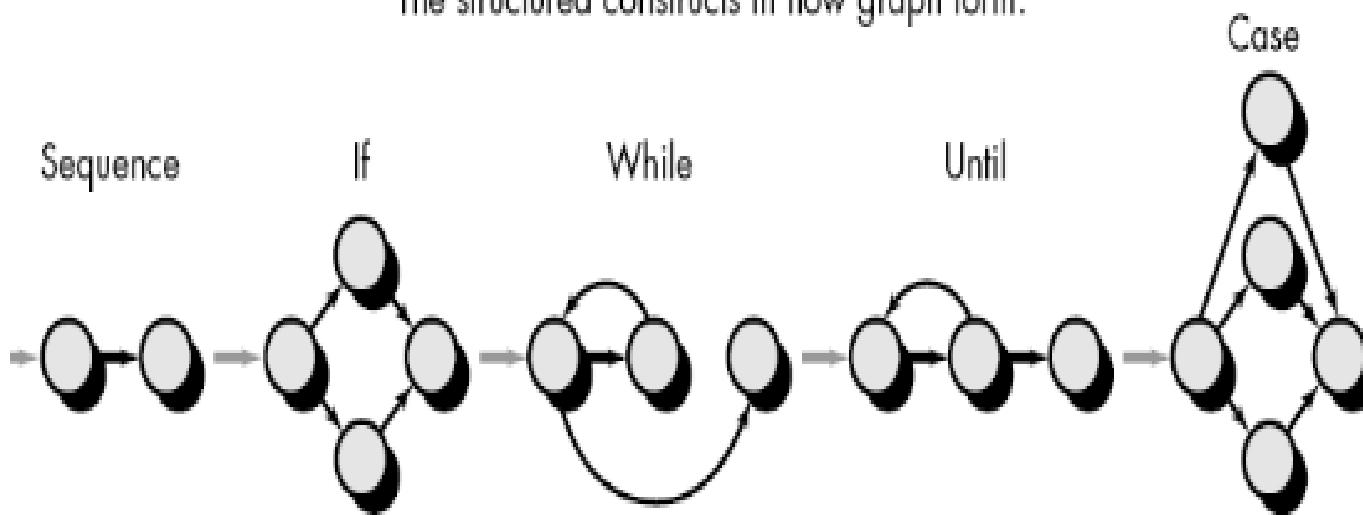
Enables the test case designer to derive a logical complexity measure of a procedural design

Uses this measure as a guide for defining a basis set of execution paths

Test cases derived to exercise the basis set are guaranteed to execute **every statement** in the **program at least one time** during testing

Flow graph notation

The structured constructs in flow graph form:



Where each circle represents one or more
nonbranching PDL or source code statements

Flow Graph Notation

A circle in a graph represents a **node**, which stands for a **sequence** of one or more procedural statements

A node containing a simple conditional expression is referred to as a **predicate node**

A predicate node **has two edges** leading out from it (True and False)

An **edge**, or a link, is an arrow representing flow of control in a specific direction

An edge must start and terminate at a node

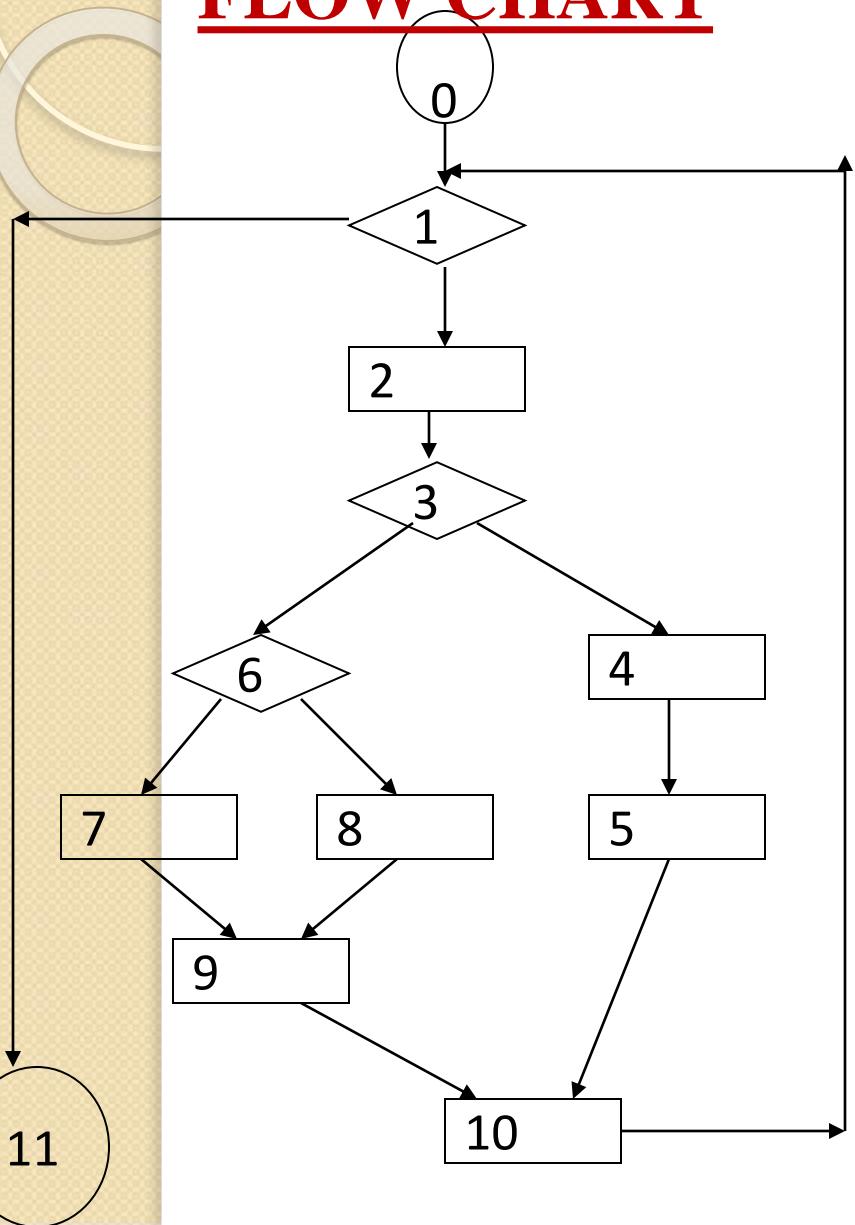
An edge does not intersect or cross over another edge

Areas bounded by a set of edges and nodes are called **regions**

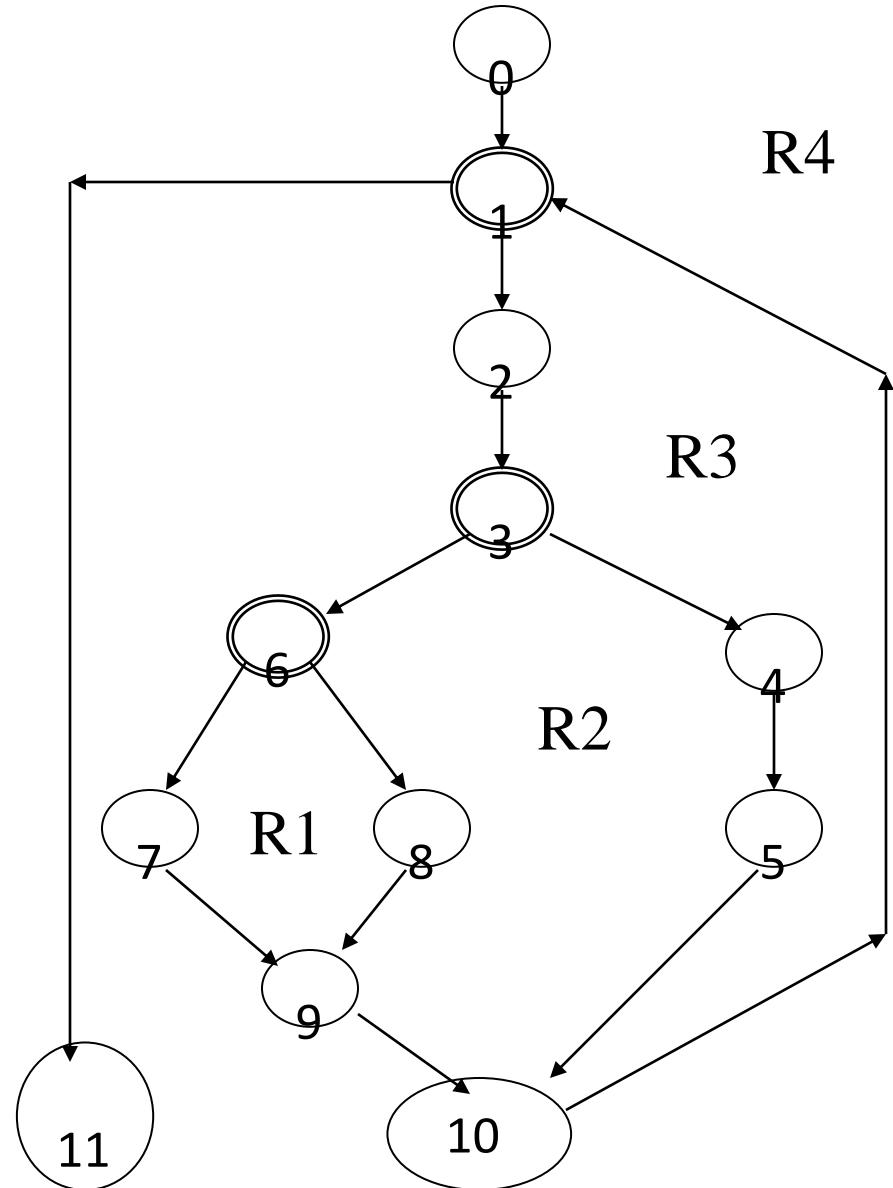
When counting regions, include the **area outside the graph as a region, too**

Flow Graph Example

FLOW CHART



FLOW GRAPH



Independent Program Paths

Must move along **at least one** edge that has not been traversed before by a previous path

Basis set for flow graph on previous slide

- Path 1: 0-1-11
- Path 2: 0-1-2-3-4-5-10-1-11
- Path 3: 0-1-2-3-6-8-9-10-1-11
- Path 4: 0-1-2-3-6-7-9-10-1-11

The **number of paths** in the basis set is determined by the **cyclomatic complexity**

Cyclomatic Complexity

It is a software metric that provides a quantitative measure of the **logical complexity** of a program

Defines the **number of independent paths** in the basis set

Provides an upper bound for the number of tests that must be conducted to ensure **all statements** have been executed **at least once**

Can be computed three ways

$$V(G) = \text{Number of bounded regions} + 1$$

$V(G) = E - N + 2$, where E is the number of edges and N is the number of nodes in graph G

$V(G) = P + 1$, where P is the number of predicate nodes in the flow graph G

Results in the following equations for the example flow graph

Number of regions = 4

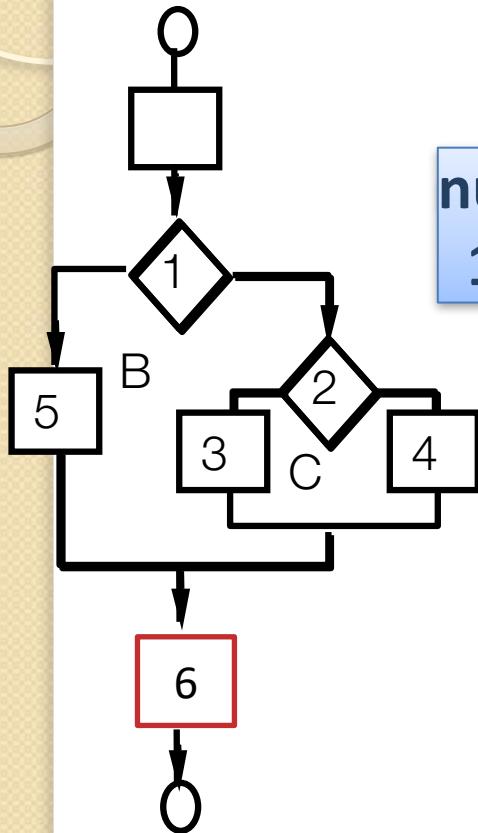
$$V(G) = 14 \text{ edges} - 12 \text{ nodes} + 2 = 4$$

$$V(G) = 3 \text{ predicate nodes} + 1 = 4$$

Basis Path Testing --

First, we compute the cyclomatic Complexity::

number of simple decisions (predicate node) + 1
1,2,= 2 Nodes +1



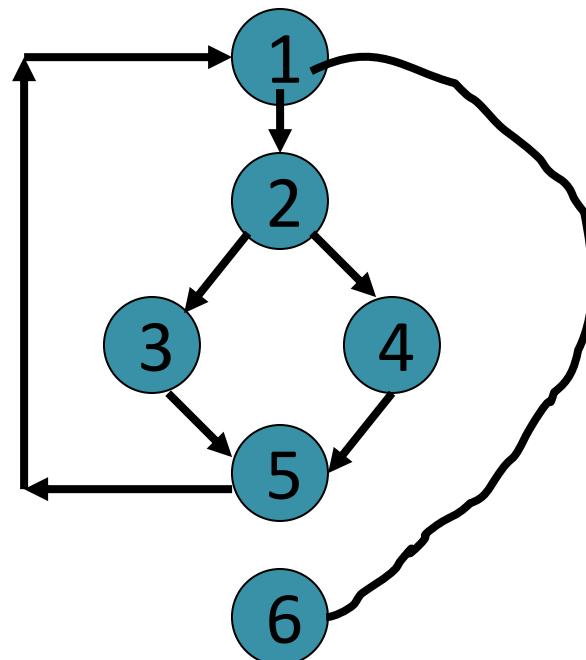
or

number of enclosed areas + 1
B,C= 2areas +1

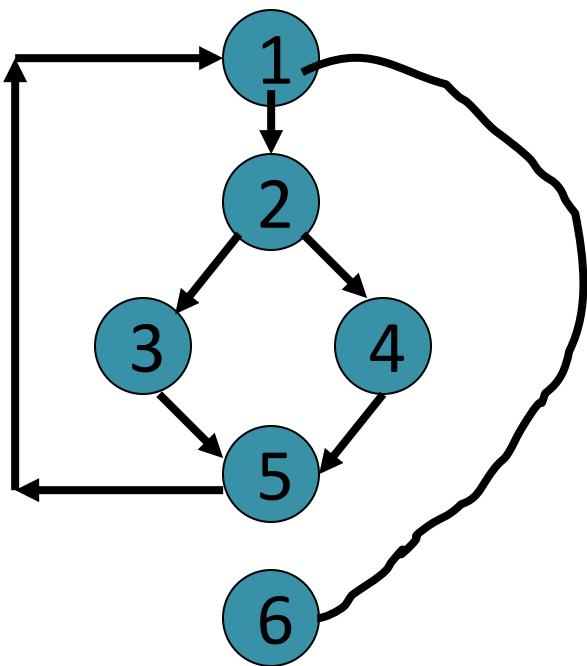
Path to be tested are :
1-5-6 ,1-2-3-6 , 1-2-4-6

Example

```
int f1(int x,int y){  
    1.    while (x != y){  
        2.        if (x>y) then  
            3.            x=x-y;  
        4.        else y=y-x;  
    }  
    5.    return x;    }  
    6.
```



Example Control Flow Graph



Cyclometric **complexity** =
 $V(G) = E - N + 2$

Cyclomatic complexity =
 $7 - 6 + 2 = 3.$

$V(G) = \text{Total number of bounded areas} + 1$

From a visual examination of the CFG:
the number of bounded areas is 2.
cyclomatic complexity = $2+1=3.$

Deriving the Basis Set and Test Cases

Using the design or code as a foundation, draw a corresponding flow graph

Determine the cyclomatic complexity of the resultant flow graph

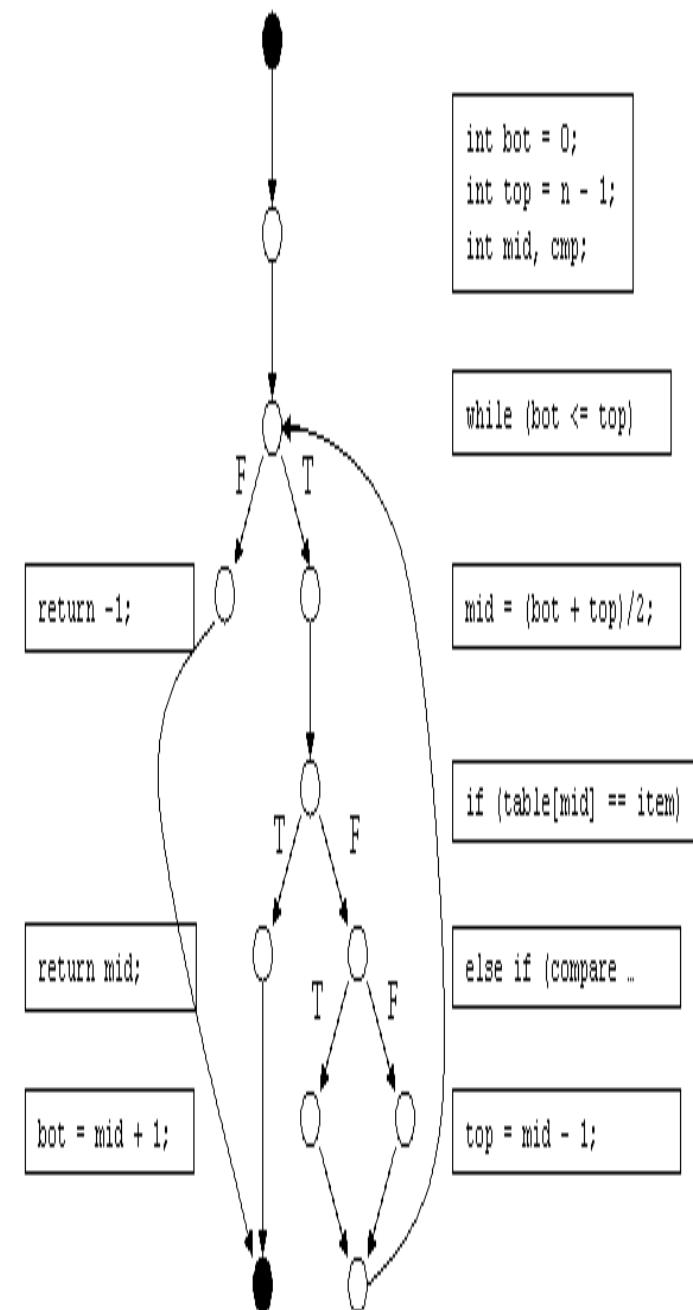
Determine a basis set of linearly independent paths

Prepare test cases that will force execution of each path in the basis set

```

int BinSearch (char *item, char *table[], int n)
{
    int bot = 0;
    int top = n - 1;
    int mid, cmp;
    while (bot <= top) {
        mid = (bot + top) / 2;
        if (table[mid] == item)
            return mid;
        else if (compare(table[mid], item) < 0)
            top = mid - 1;
        else
            bot = mid + 1;
    }
    return -1; // not found
}

```



$CC = \text{number of edges} - \text{number of nodes} + 2$

$$CC = 14 - 12 + 2$$

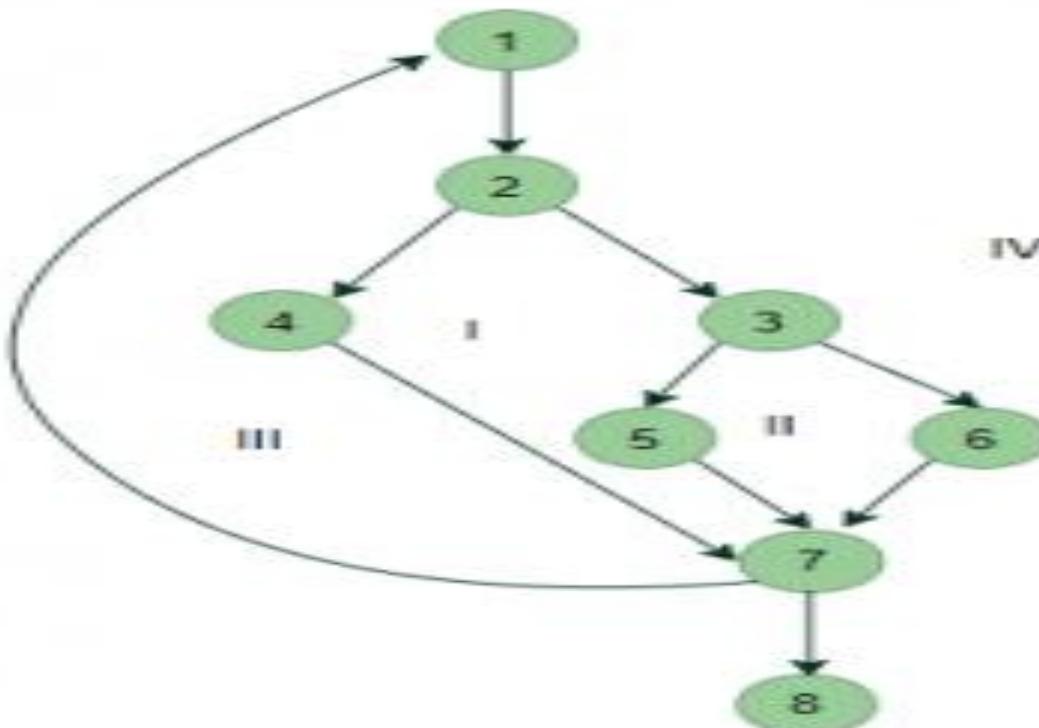
$$= 4$$

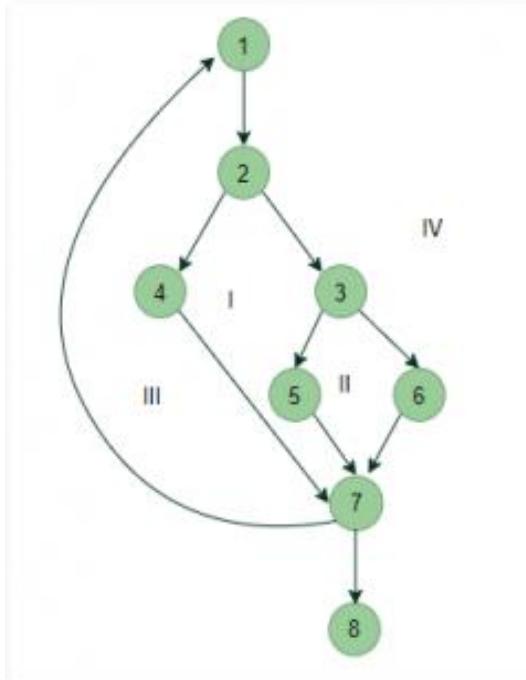
$CC = \text{number of decision point} + 1$

$$= 3 + 1$$

$$= 4$$

- i) 1,2,4,7,8 ii) 1,2,3,5,7,8 iii) 1,2,3,6,7,8 iv) 1,2,4,7,1,2,3,6,7,8





(any one of the above formulae)

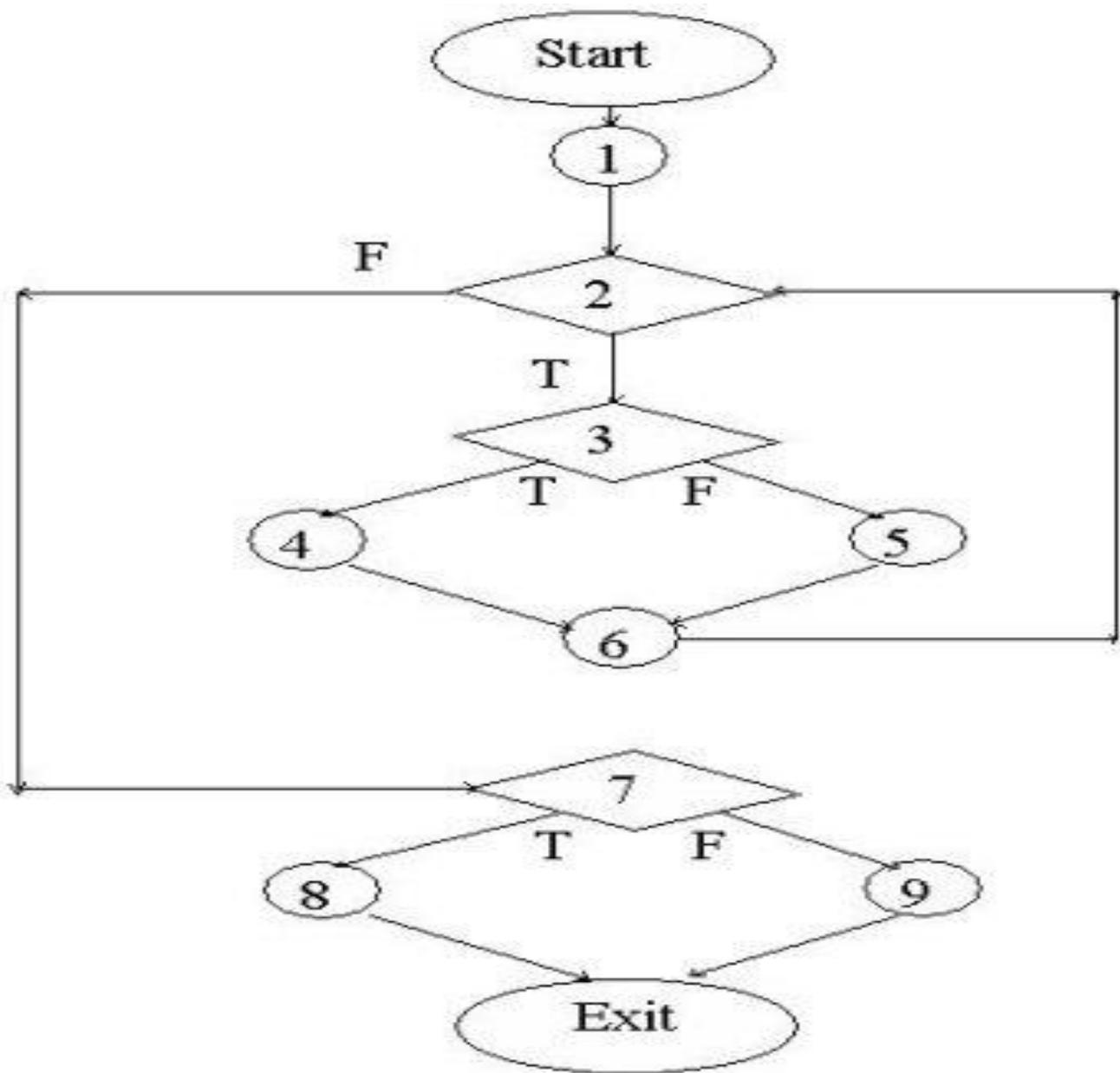
No of independent paths = 4

- #P1: 1 - 2 - 4 - 7 - 8
- #P2: 1 - 2 - 3 - 5 - 7 - 8
- #P3: 1 - 2 - 3 - 6 - 7 - 8
- #P4: 1 - 2 - 4 - 7 - 1 - . . . - 7 - 8

FindMean (FILE ScoreFile)

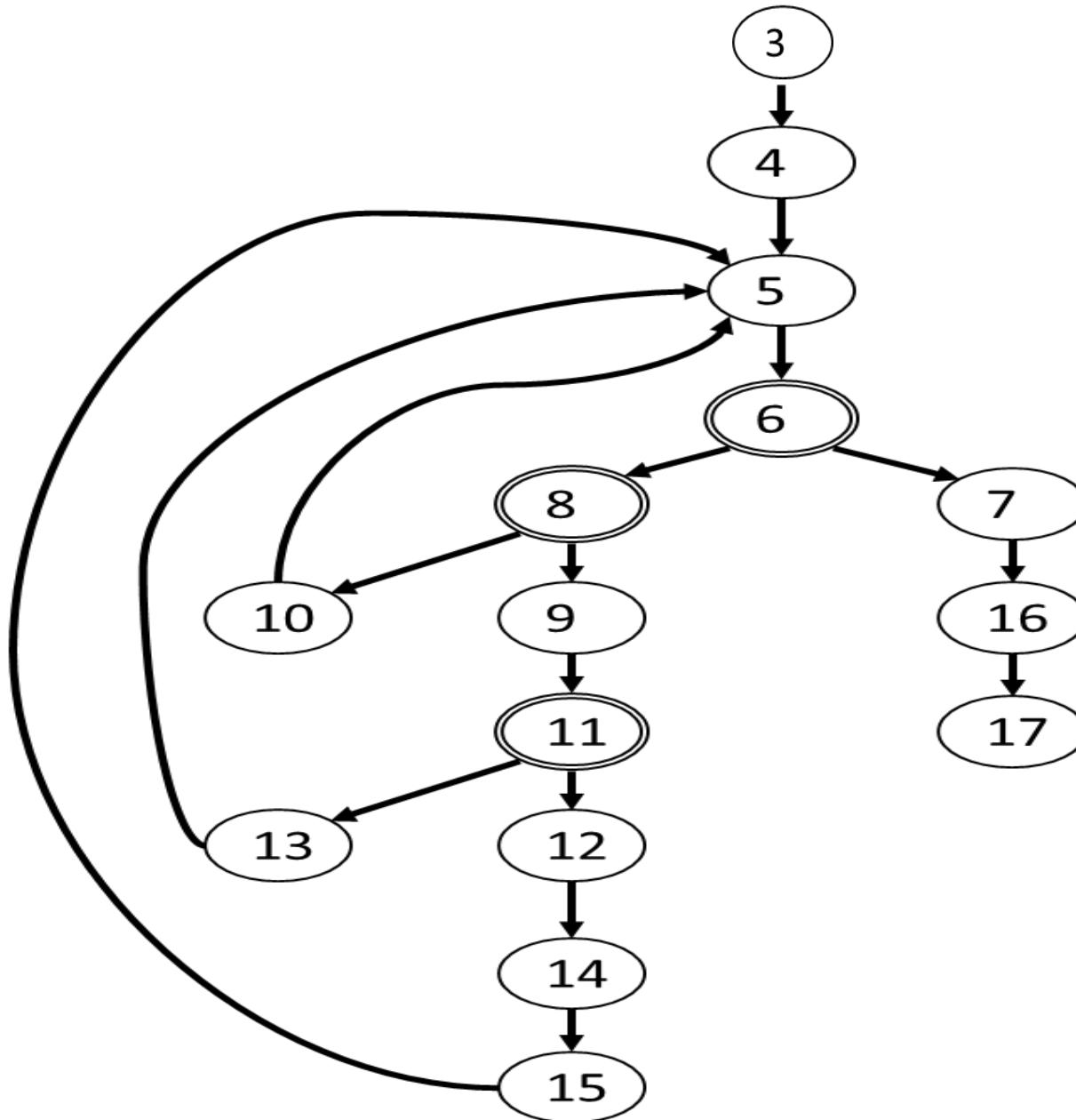
```
{   float SumOfScores = 0.0;
    int NumberOfScores = 0;
    float Mean=0.0; float Score;
    Read(ScoreFile, Score);
    2 while (! EOF(ScoreFile) {
        3 if (Score > 0.0 ) {
            SumOfScores = SumOfScores + Score;
            NumberOfScores++;
            5 }
        6 Read(ScoreFile, Score);
    }
    /* Compute the mean and print the result */
    7 if (NumberOfScores > 0) {
        Mean = SumOfScores / NumberOfScores;
        printf(" The mean score is %f\n", Mean);
    } else
        9 printf ("No scores found in file\n");
```

Constructing the Logic Flow Diagram



A Second Flow Graph Example

```
1 int functionY(void)
2 {
3     int x = 0;
4     int y = 19;
5
6     A: x++;
7     if (x > 999)
8         goto D;
9     if (x % 11 == 0)
10        goto B;
11    else goto A;
12
13    B: if (x % y == 0)
14        goto C;
15    else goto A;
16
17    C: printf("%d\n", x);
18    goto A;
19
20    D: printf("End of list\n");
21    return 0;
22 }
```



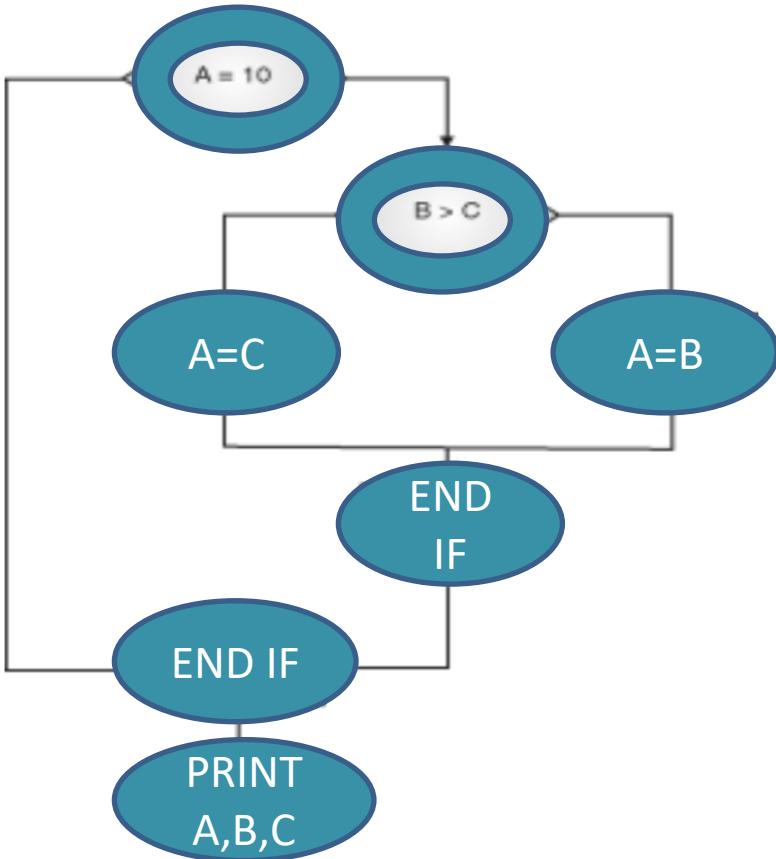
FIND CYCLOMATIC COMPLEXITY

```
IF A = 10 THEN
    IF B > C THEN
        A=B
    ELSE A= C
END IF
END IF

PRINT A
PRINT B
PRINT C
```

SOLUTION

Flow graph



The Cyclomatic complexity is calculated using the formula $C = E - N + 2P$, where E is the number of edges, N is the number of nodes, and P is the number of predicate nodes (decision points). In this flow diagram, there are seven nodes (shapes) and eight edges. There are two predicate nodes (the decision node B > C and the first 'END IF' node). Hence the cyclomatic complexity is $8 - 7 + 2 = 3$.

```
IF A = 10 THEN  
IF B > C THEN  
    A = C  
    A = B  
END IF  
END IF
```

A =
EL

PR
PR
PR

Test Cases

- . Derived during all phases of the development cycle
- . Determine what are your **expected results** before you run a test-case
 - . Then, running a test case provides the **actual results**
- . Test cases are presented as a table.

Test Case Table

Test id. (path/no.)	Description	Input	Expected Results	Actual Results	Pass/ Fail

FIND CYCLOMATIC COMPLEXITY

IF A = 10 THEN

IF B > C THEN

A=B

ELSE A= C

END IF

END IF

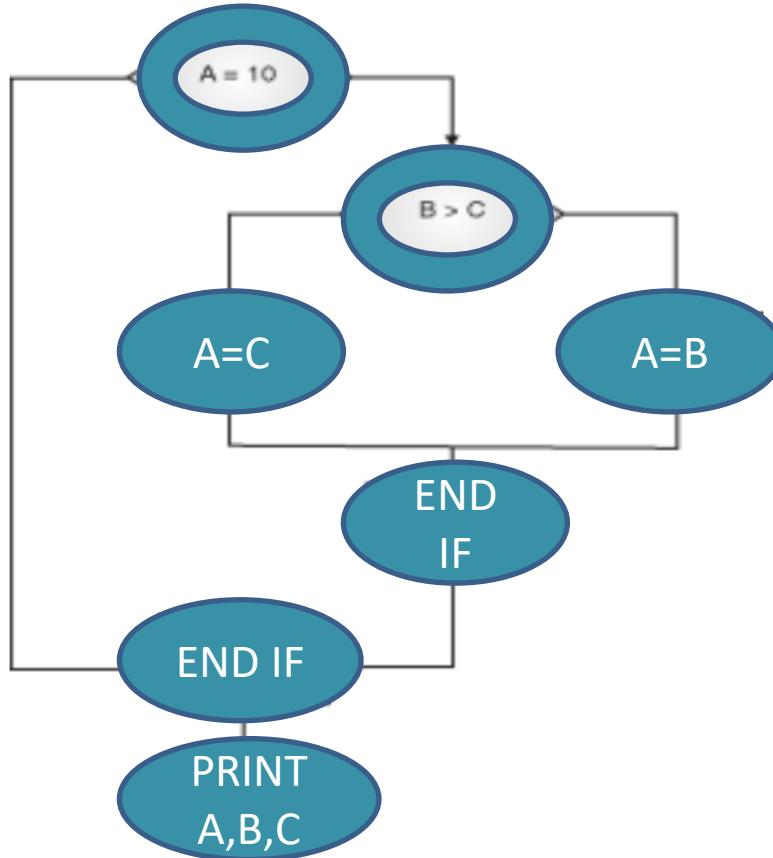
PRINT A

PRINT B

PRINT C

SOLUTION

Flow graph



The Cyclomatic complexity is calculated using the above control flow diagram that shows seven nodes (shapes) and eight edges (lines), hence the cyclomatic complexity is $8 - 7 + 2 = 3$

Test Case Table

Test ID	Description	Input	Expected Result	Actual Result	PASS/FAIL
1	If A=10 ,Compare B & C	10	Comparing B & C	Comparing B & C	PASS
2	If A=10 ,Compare B & C	9	Print A	Print A	Pass
	If A=10 ,Compare B & C	8	Print A	Comparing B & C	Fail
3	If B>C,ASSIGN C to A ,print A,B,C	B=7,C=5	A=5,B=7,C =5	A=5,B=7,C =5	Pass
4	If B>C,ASSIGN C to A ,print A,B,C				

Test ID	Description	Input	Expected Result	Actual Result	Pass/Fail
1	Navigate to the Login Page	User=abc@gmail.com	User should be able to login and navigate to view product menu	User is able to navigate to view product menu	Pass
2	Provide valid user name and password	Username =abc@gmail.com Pwd=asdf	Validate password	User is validated	pass
3	Provide valid user name and password	Username =abc@gmail.com Pwd=qwerty	Validate password	Invalid password	fail

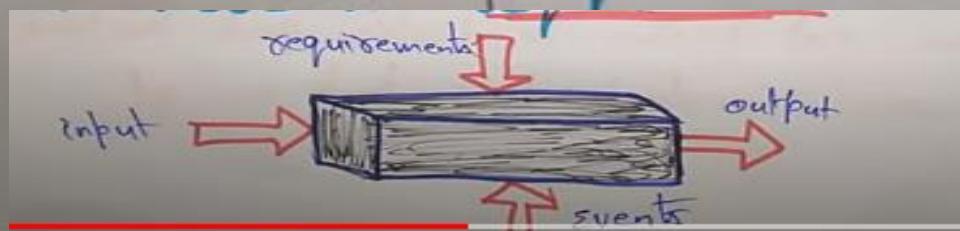


Black-box Testing

Black-box testing, also called behavioral testing focuses on the functional requirements of the software

Black-Box Testing

- ⇒ It is a method of s/w testing that (examines) the functionality of an application without looking into its internal structures or workings.
- ⇒ This method is applied at every level of s/w testing: unit, integration, system and acceptance.
- ⇒ Tests are based on requirements and functionality.



Black-Box Testing

- ⇒ It is carried out to test functionality of the program also called as 'Behavioral testing'.
- ⇒ The tester in this case, has a set of i/p values and desired results.
 - on providing i/p, if the o/p matches with the desired results, the program is tested 'ok'.
- ⇒ In this testing method, the design & structure of code are not known to tester and testing engineer and end users conduct this test on a/

Acceptance Testing

- A testing technique performed to determine whether or not the software system has met the requirement specifications
- Referring to the functional testing of a user story by the software development team during the implementation phase in agile development

*

Black-box Testing Steps

Initially requirements and specifications of the system are examined.

Tester chooses **valid inputs** (positive test scenario) to check. Also some **invalid inputs** (negative test scenario) are chosen.

Tester determines expected outputs for all those inputs.

Software tester constructs test cases with the selected inputs.

The test cases are executed.

Software tester compares the actual outputs with the expected outputs.

Defects if any are fixed and re-tested.

Black box testing strategy:

Equivalence Class Testing: It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage.

Boundary Value Testing: Boundary value testing is focused on the values at boundaries. This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases. It is mostly suitable for the systems where input is within certain ranges.

BVA & EP

i. **BVA (Boundary Value Analysis)** : Boundary Value Analysis focuses on testing boundary conditions e.g.: If we have to derive test conditions to test a text box that accepts age between 18 and 100. Then test conditions would be 17, 18, 19, 50, 99, 100 and 101. As you can see, test conditions focus on testing just below boundary, at boundary and over boundary.

ii. **Equivalence Partitioning (EP)** : Equivalence partitioning focuses deriving test conditions based on classes of data at a minimal of 2 classes i.e. Positive and error classes of data. e.g. If we have to test password text box, Positive class would be valid password and invalid class would be password in different case, incomplete password or different password.

Equivalence Partitioning

A black-box testing method that **divides the input domain of a program into classes** of data from which test cases are derived

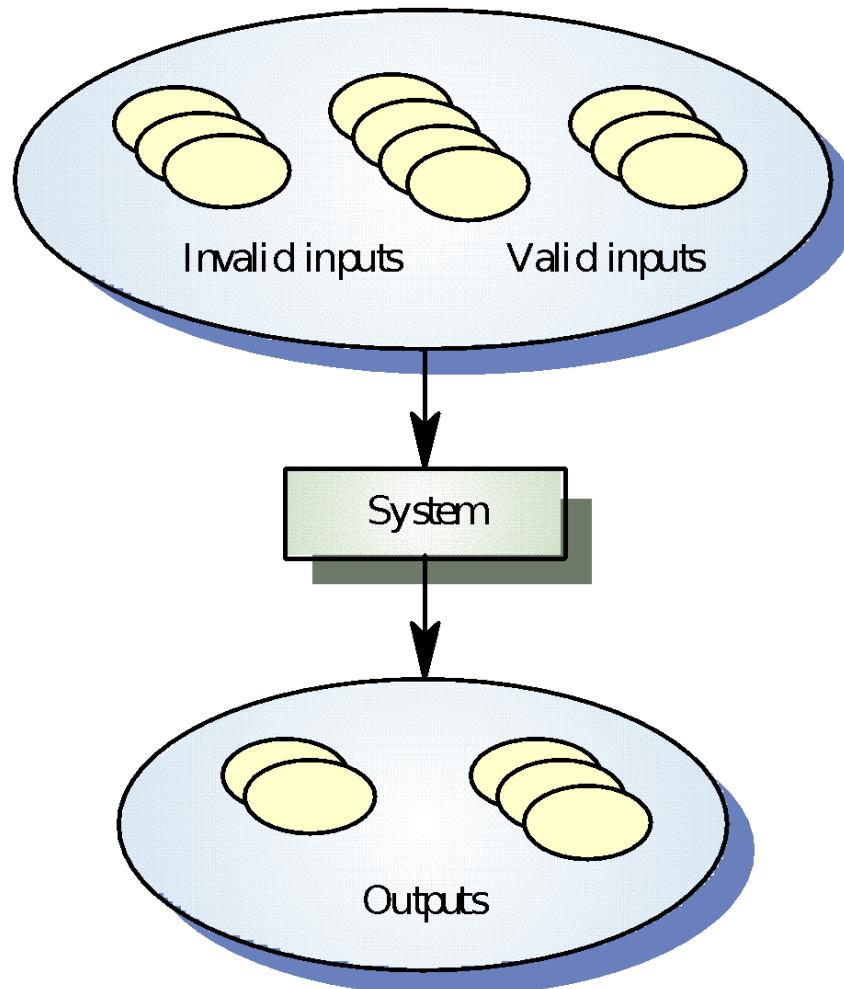
An ideal test case single-handedly uncovers a complete class of errors, thereby reducing the total number of test cases that must be developed

Test case design is based on an evaluation of equivalence classes for an input condition

An equivalence class represents a **set of valid or invalid states for input conditions**

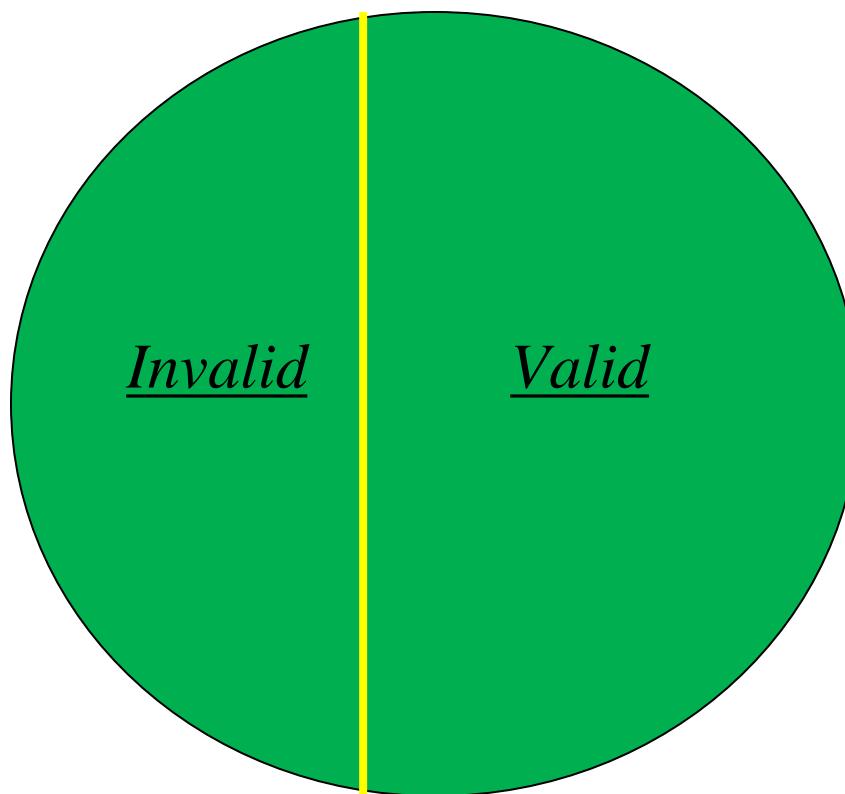
From each equivalence class, test cases are selected so that the largest number of attributes of an equivalence class are exercised at once

Equivalence partitioning



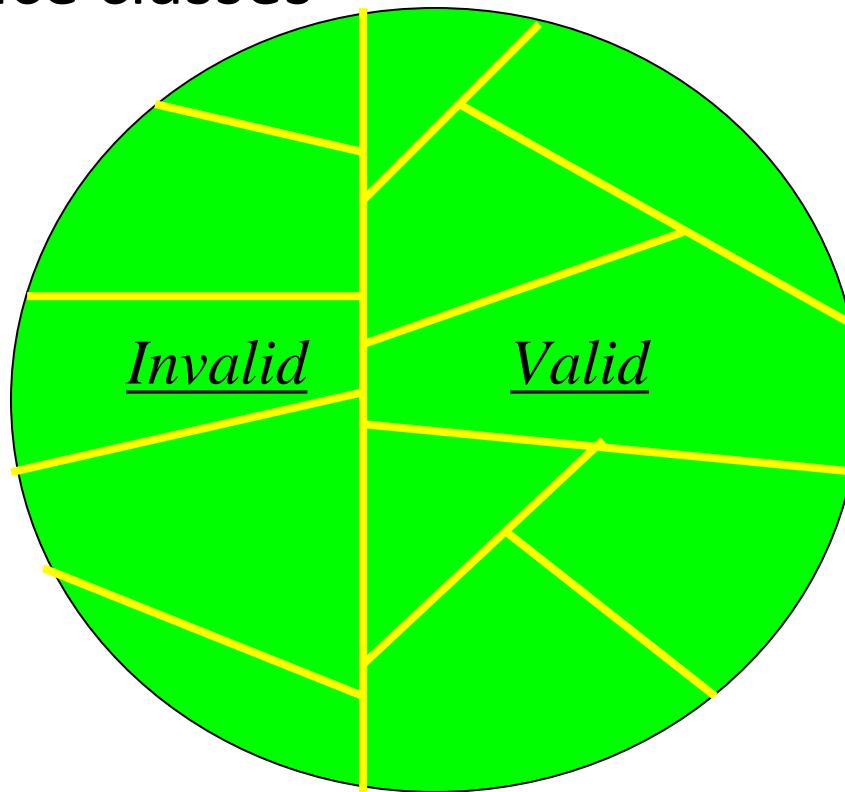
Equivalence Partitioning

First-level partitioning: Valid vs. Invalid values



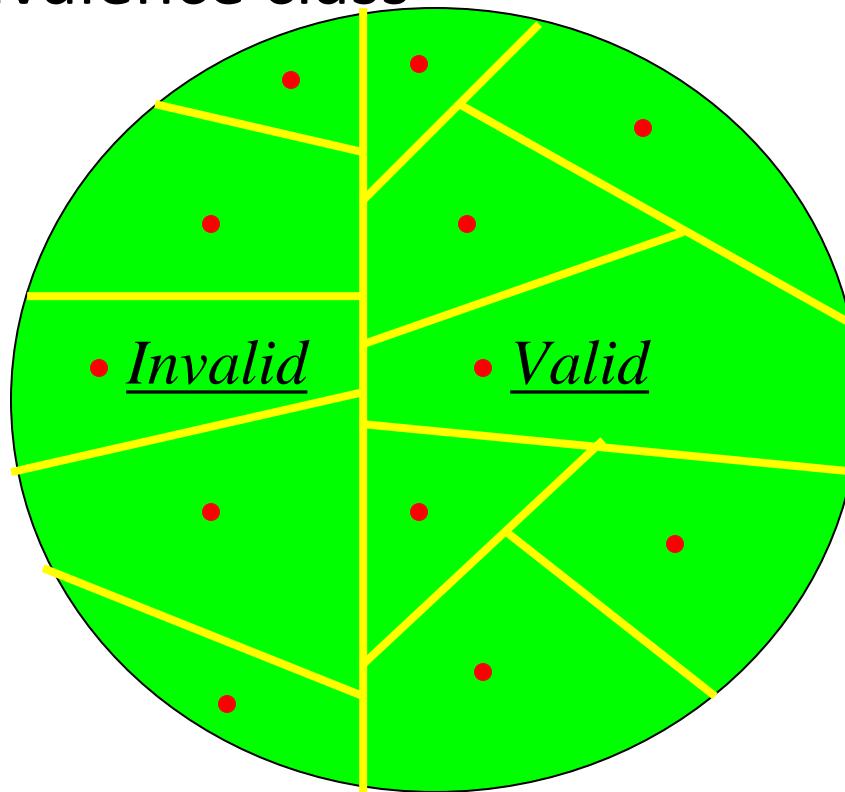
Equivalence Partitioning

Partition valid and invalid values into equivalence classes



Equivalence Partitioning

Create a test case for at least one value from each equivalence class



Guidelines for Defining Equivalence Classes

If an input condition specifies a range, one valid and two invalid equivalence classes are defined

- Input range: 1 – 10 Eq classes: {1..10}, {x < 1}, {x > 10}

If an input condition requires a specific value, one valid and two invalid equivalence classes are defined

- Input value: 250 Eq classes: {250}, { $x < 250$ }, { $x > 250$ }

If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined

- Input set: $\{-2.5, 7.3, 8.4\}$ Eq classes: $\{-2.5, 7.3, 8.4\}$, {any other x}

If an input condition is a Boolean value, one valid and one invalid class are defined

- Input: {true condition} Eq classes: {true condition}, {false condition}

EQUIVALENCE PARTITIONING

- Test design technique
- Divide input test data into partitions
- Test each partition *ONCE* (the assumption is that any input within a partition is equivalent i.e. produces the same output)

EP EXAMPLE - DATE (1 TO 31)

Invalid Partition	Valid Partition	Invalid Partition
<ul style="list-style-type: none">• 0• -1• -2• ...	<ul style="list-style-type: none">• 1• 2• 3• ...• 31	<ul style="list-style-type: none">• 32• 33• ...

EP EXAMPLE - USERNAME (6 TO 10 CHARACTERS)

Invalid Partition

- 0 character
- 1 character
- 2 characters
- 3 characters
- 4 characters
- 5 characters

Valid Partition

- 6 characters
- 7 characters
- 8 characters
- 9 characters
- 10 characters

Invalid Partition

- 11 characters
- 12 characters
- ...

*

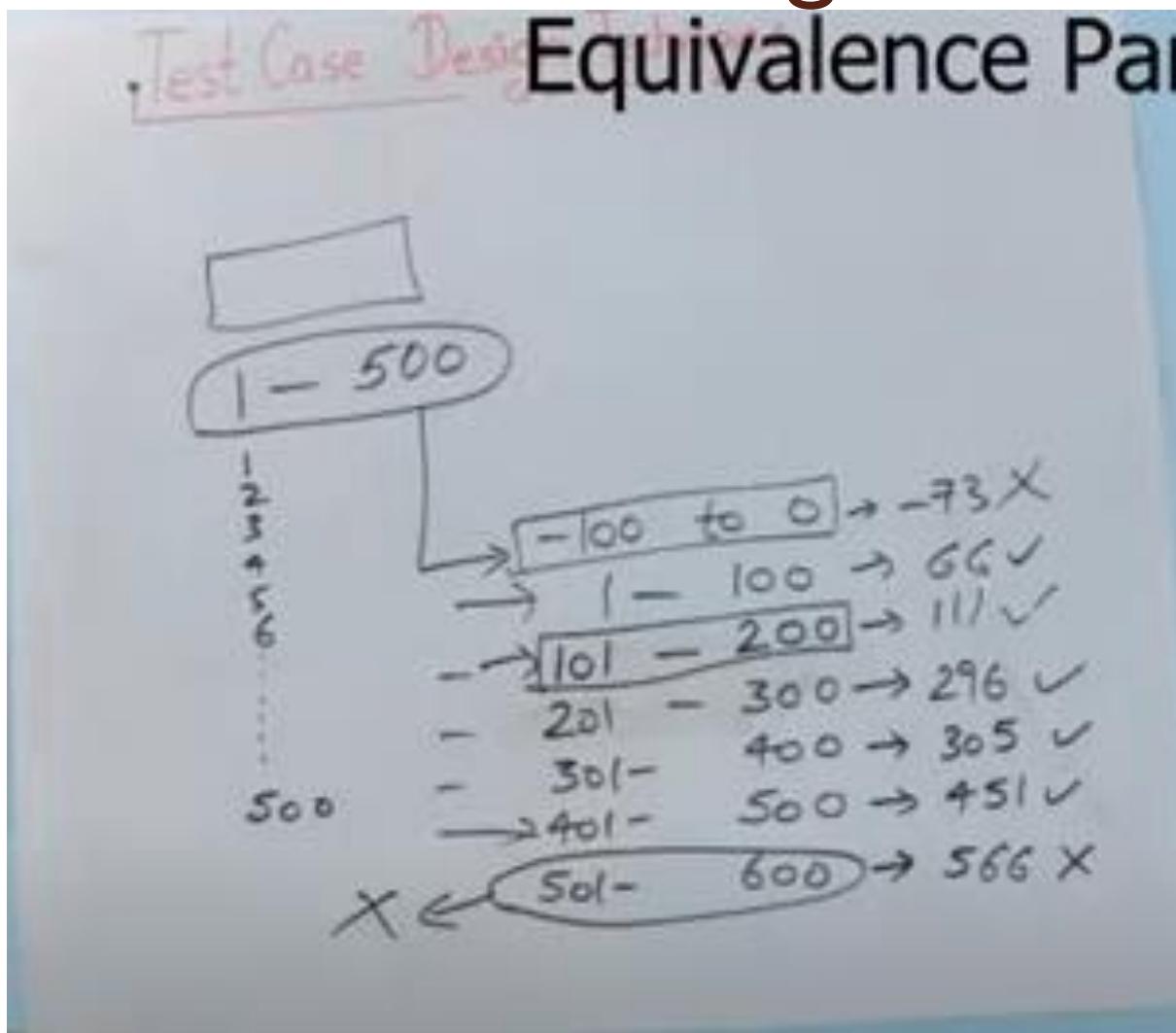
EP EXAMPLE - AGE (18 TO 80 YEARS EXCEPT 60 TO 65 YEARS)



*

Black-Box Testing

Equivalence Partitioning



*

Boundary Value Analysis

A greater number of errors occur at the boundaries of the input domain rather than in the "center"

Boundary value analysis is a test case design method that complements equivalence partitioning

It selects test cases at the edges of a class

Guidelines for Boundary Value Analysis

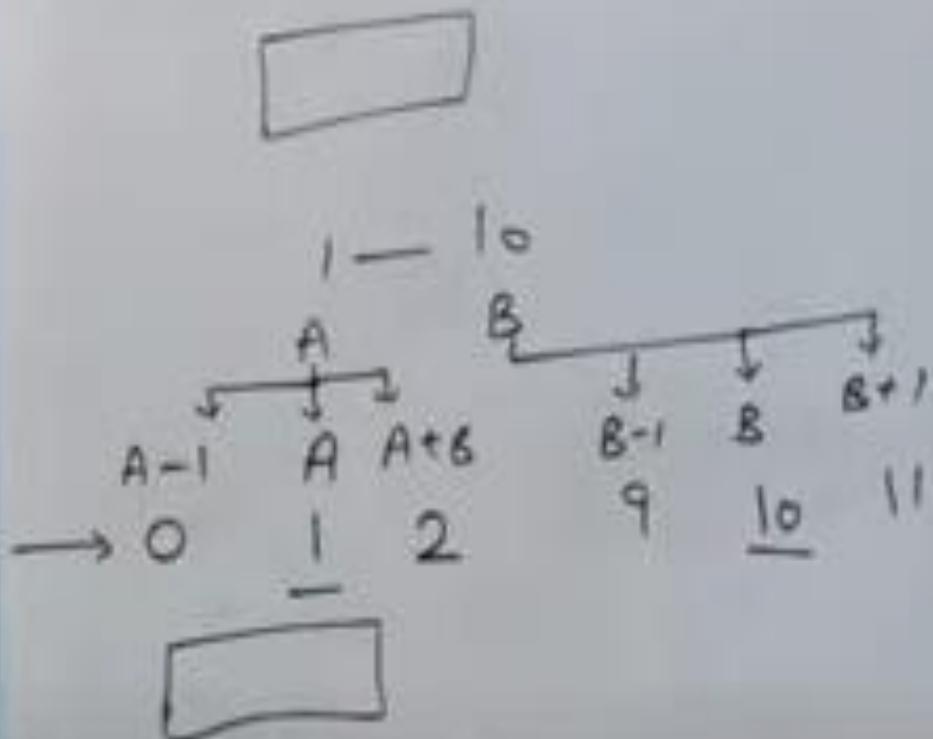
1. If an input condition specifies **a range bounded by values a and b , test cases should be designed with values a and b as well as values **just above and just below a and b****
2. If an input condition specifies a number of values, test case should be developed that **exercise the minimum and maximum numbers. Values just above and just below the minimum and maximum are also tested**

Apply guidelines 1 and 2 to output conditions; produce output that reflects the minimum and the maximum values expected; also test the values just below and just above

BOUNDARY VALUE ANALYSIS

- Test design technique
- Related to Equivalence Partitioning
- Test *BOTH SIDES* of each boundary (the assumption is that the system behaves differently on either side of a boundary)

Boundary Value Analysis



$50 - 55$
49, 50, 51 & 54, 55, 56

*

BVA EXAMPLE - DATE (1 TO 31)

Invalid Partition – Valid Partition Lower Boundary	Invalid Partition – Valid Partition Upper Boundary
Boundary value just below the boundary	Boundary value just above the boundary
0	1

A=1 (LB),B=31(UB)

A-1 A A+1
0 1 2

B-1 B B+1
30 31 32

*

#	Black Box Testing	White Box Testing
1	Black box testing is the <u>Software testing method</u> which is used to test the software without knowing the internal structure of code or program.	White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.
2	This type of testing is carried out by testers.	Generally, this type of testing is carried out by software developers.
3	Implementation Knowledge is not required to carry out Black Box Testing.	Implementation Knowledge is required to carry out White Box Testing.
4	Programming Knowledge is not required to carry out Black Box Testing.	Programming Knowledge is required to carry out White Box Testing.
5	Testing is applicable on higher levels of testing like System Testing, Acceptance testing.	Testing is applicable on lower level of testing like Unit Testing, Integration testing.
6	Black box testing means functional test or external testing.	White box testing means structural test or interior testing.
7	In Black Box testing is primarily concentrate on the functionality of the system under test.	In White Box testing is primarily concentrate on the testing of program code of the system under test like code structure, branches, conditions, loops etc.

8	The main aim of this testing to check on what functionality is performing by the system under test.	The main aim of White Box testing to check on how System is performing.
9	Black Box testing can be started based on Requirement Specifications documents.	White Box testing can be started based on Detail Design documents.
10	The Functional testing, Behavior testing, Close box testing is carried out under Black Box testing, so there is no required of the programming knowledge.	The Structural testing, Logic testing, Path testing, Loop testing, Code coverage testing, Open box testing is carried out under White Box testing, so there is compulsory to know about programming knowledge.

Software Maintenance

- The software maintenance is an activity that actually begins after the software product delivered at the client's end
- In software maintenance, the modifications are carried out or the updates in the software are taken place
- In software maintenance phase no major changes are implemented
- In this, the changes are done in the existing program or the some small new functionality is added

*

Software Maintenance

- Three decades ago, software maintenance was given the name as iceberg, where the potential problems reside inside and it is not visible to the clients. If maintenance is not done properly, it will sink the entire ship, as icebergs do.

Modifiability

- The major cost of the software during its life cycle is the maintenance cost
- When software is designed, the stakeholders want the software should be designed in such a way that future changes can easily be accommodated and implemented with less maintenance cost.
- The modifiability means the ability of the software to be modified
- Modifications include: improvements, corrections, adaptability in changing environments and in changing requirements and the functional specifications

*

Modifiability

- Modifiability is the ease with a software system can be modified to accommodate changes in requirements, environments and functional specifications
- The maintainability involves modifications in requirements as well as correction in the bugs whereas modifiability does not involve in correcting the bugs

Types of maintenance

- Corrective maintenance
- Adaptive maintenance
- Perfective maintenance
- Preventive maintenance

Corrective maintenance

- It is the type of maintenance in which errors are fixed when it is observed during the use of the software
- In this the errors may be caused due to faulty software design, incorrect logic and improper coding
- All these errors are called as residual errors and they prevent the final specification

Adaptive maintenance

- Adaptive maintenance means the implementation of the modification in the system.
- The changes may be of hardware or the operating system environments

*

Perfective maintenance

- It deals with the modified and changed user requirements
- The functional enhancements are taken into consideration
- The function and efficiency of the code is continuously improved

Preventive maintenance

- It is used to prevent the possible errors to occur
- Complexity is minimized and the quality of the program is enhanced

Youtube link for software maintenance

● <https://www.youtube.com/watch?v=8swQr0kckZI>

University Questions

- Differentiate between white box and black box testing(5/10 marks)
- Explain Cyclomatic Complexity.How it is computed?(10 marks)
- What is maintenance?Explain the different types of maintenance.(10 marks)