CMP 519 Software Engineering

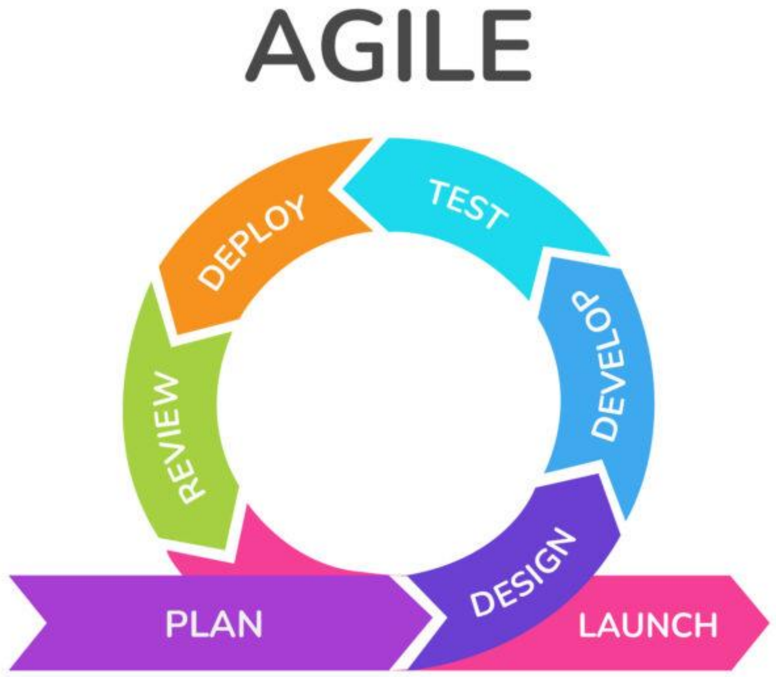
UNIT 02 - SOFTWARE PROCESS MODELS AND PROJECT MANAGEMENT

Instructor: Dr. Suresh Pokharel

DISCLAIMER

These slides are part of teaching materials for Software Engineering. These slides do not cover all aspect of learning Software Engineering, nor are these be taken as primary source of information. As the core textbooks and reference books for learning the subject has already been specified and provided to the students, students are encouraged to learn from the original sources.

Agile Development Principles



TERMINOLOGIES

1. **Epic**: A large, overarching body of work that can be broken down into smaller tasks or user stories. Example: "Develop an e-commerce website."
2. **User Story**: A short description of a feature or requirement from the user’s perspective. Example: "As a user, I want to search for products so that I can find what I need easily."
3. **Story Points**: A unit of measure used to estimate the effort required to complete a user story. Example: Searching for products might be estimated as 3 story points.
4. **Sprint**: A fixed time period (typically 1-4 weeks) during which a specific set of tasks is completed. Example: A sprint could involve creating the search functionality.

TERMINOLOGIES

5. **Stand-Up**: A short daily meeting (often 15 minutes) where team members share progress, challenges, and plans. Example: "Yesterday I integrated the search functionality; today I'll work on testing it. Blocked by incomplete UI design."
6. **Product Backlog**: A prioritized list of all desired features, fixes, and tasks for the product. Example: Includes items like search, login, and checkout.
7. **Sprint Backlog:** A subset of the product backlog selected for completion during a sprint. Example: For one sprint, the sprint backlog might include search and login functionalities.
8. **Scrum Master:** A facilitator responsible for ensuring the team follows Agile practices and removes obstacles.

TERMINOLOGIES

9. **Product Owner:** A person who defines the backlog, prioritizes tasks, and ensures the team delivers value to the stakeholders.
10. **Velocity**: The amount of work completed by a team during a sprint, usually measured in story points.
11. **Definition of Done (DoD):** A checklist of criteria that a task must meet to be considered complete. Example: "Unit tested, code reviewed, and deployed."
12. **Burndown Chart**: A visual representation of work remaining vs. time in a sprint.
13. **Retrospective**: A meeting held at the end of a sprint to discuss what went well, what didn’t, and how to improve.

Use Case: Online Library System

Scenario: Create a simple **Online Library System** with two main features:

- **User Management**: Users can sign up, log in, and view their profiles.
- **Book Search and Reading**: Users can search for books by title, author, or genre and read books online.

AGILE USE CASE: ONLINE LIBRARY SYSTEM

1. **Epic**: "Develop an Online Library System."
2. **Features** (Smaller Goals):
   - User Management
   - Book Search and Reading
3. **User Stories:**
   - *For User Management:*

     "As a user, I want to sign up so I can access my library profile.“

     "As a user, I want to log in so I can access my library profile."
   - *For Book Search and Reading:*

     "As a user, I want to search for books by title or author so I can find my favorite books."
     "As a user, I want to read books online so I don’t need to download them."

AGILE USE CASE: ONLINE LIBRARY SYSTEM

4. **Story Points:**
   - User Management: 3 points (relatively simple login/signup form).
   - Book Search: 5 points (requires database integration and a search algorithm).
   - Online Reading: 8 points (includes designing the reader interface).
5. **Sprint**: The team has 1 sprint (2 weeks) to deliver basic functionality.
6. **Product Backlog:**
   - User Signup
   - Login
   - Book Search Functionality
   - Online Reading Interface

AGILE USE CASE: ONLINE LIBRARY SYSTEM

7. **Sprint Backlog**: For this sprint (half an hour class simulation), the team picks:
   - User Signup
   - Login
   - Book Search
8. **Stand-Up**: During a quick stand-up, students discuss:
   - "Yesterday, we brainstormed ideas for user signup."
   - "Today, we will implement the form UI."
   - "Blocker: Need clarity on whether passwords will be encrypted."

AGILE USE CASE: ONLINE LIBRARY SYSTEM

9. **Definition of Done**:
   - User Signup/Login is done when the form is functional, validated, and connected to a mock backend.
   - Book Search is done when a user can search and see results (mock data).
10. **Retrospective**: At the end of the activity, the class discusses:
    - What went well: "Good collaboration on creating the search UI."
    - What didn't: "Ran out of time for integrating the mock backend."
    - Improvements: "We'll break tasks into smaller parts next time."

AGILITY PRINCIPLES

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

AGILITY PRINCIPLES

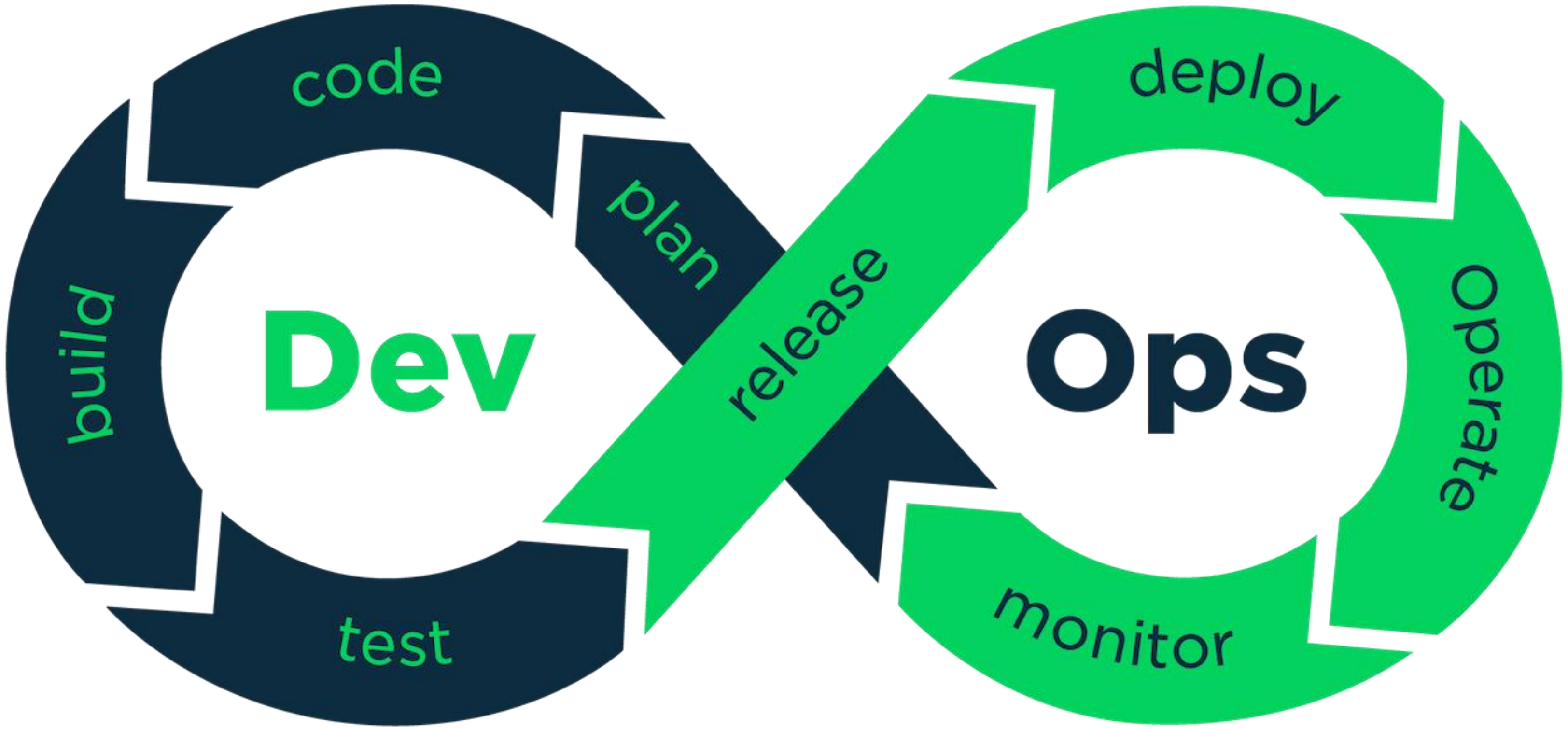
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.

AGILITY PRINCIPLES

11. The best architectures, requirements, and designs emerge from self– organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Scrum and Kanban Methodologies

DevOps Culture and Principles



Project Organization Concepts

Developer’s Responsibilities

- Who is responsible for which part of the system?
- Which part of the system is due by when?
- Who should be contacted when a problem with a specific version of a component is discovered?
- How should a problem be documented?
- What are the quality criteria for evaluating the system?
- In which form should new requirements be communicated to developers?
- Who should be informed of new requirements?
- Who is responsible for talking to the client?

Project

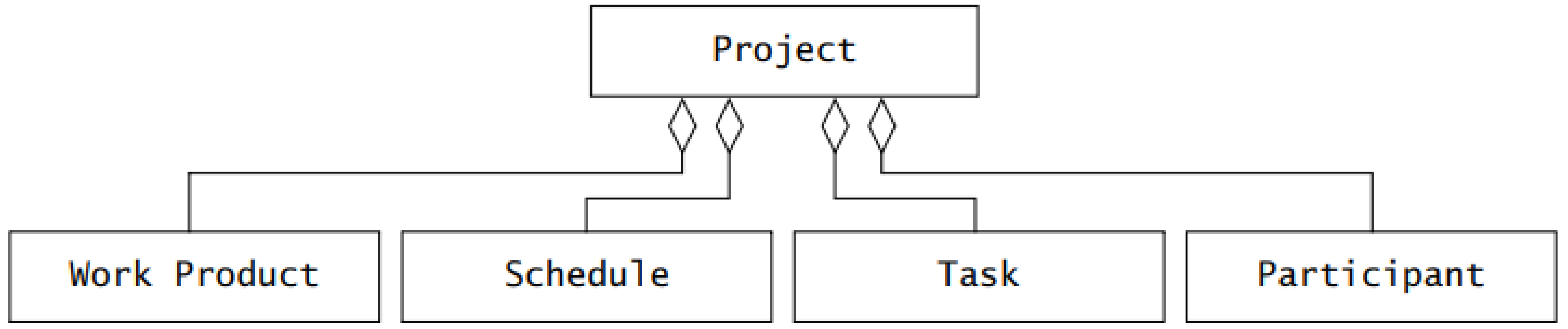


Figure 3-1 Model of a project (UML class diagram).

Project

- **Work product.** This is any item produced by the project, such as a piece of code, a model, or a document. Work products produced for the client are called deliverables.
- **Schedule.** This specifies when work on the project should be accomplished.
- **Participant.** This is any person participating in a project. Sometimes we also call the participant project member.
- **Task.** This is the work to be performed by a project participant to create a work product.

Project

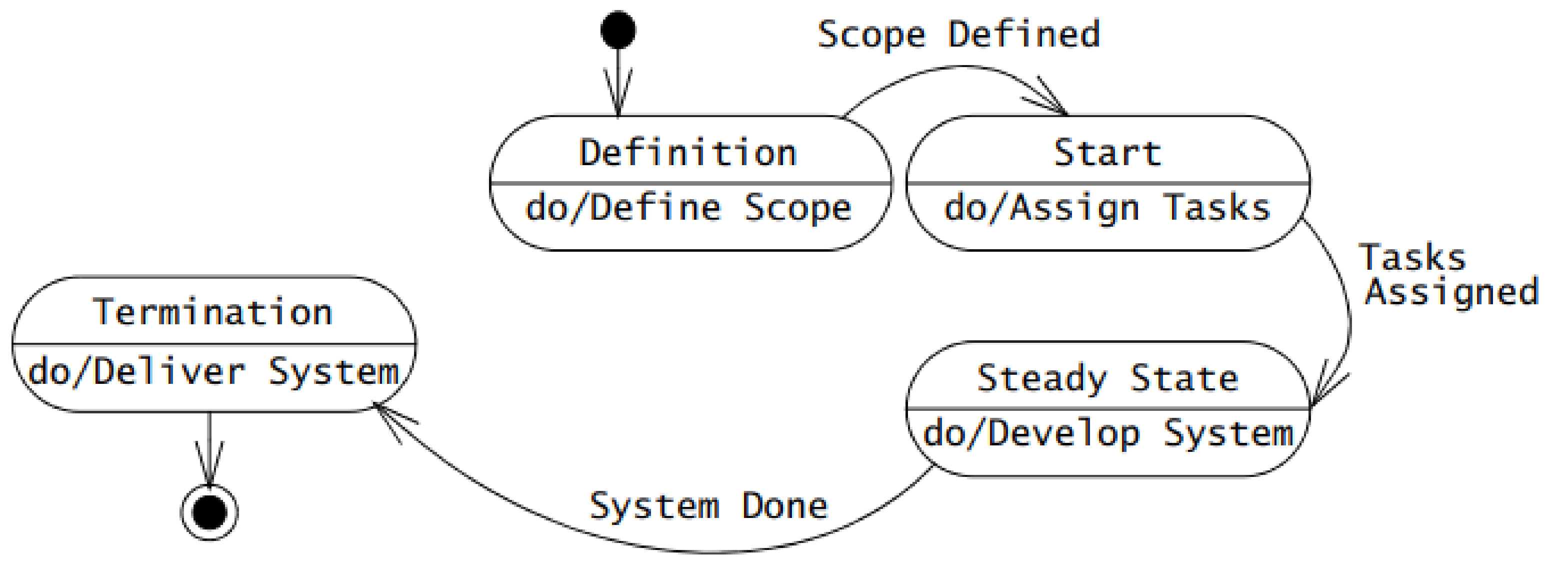


Figure 3-2 States in a software project (UML state machine diagram).

Project Organization

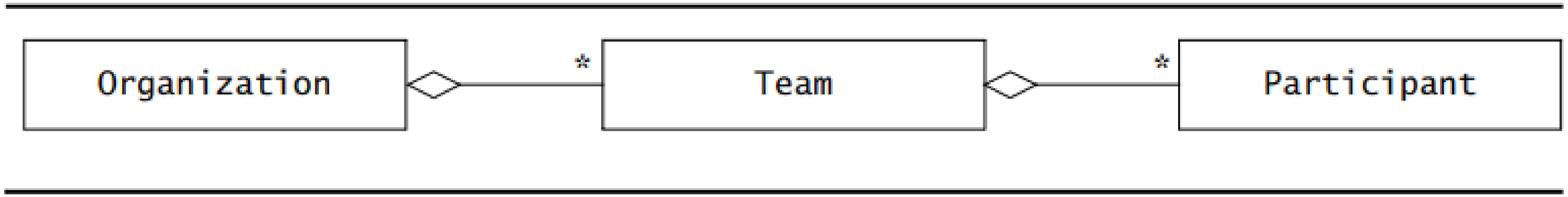


Figure 3-3 A team-based organization consists of organizational units called teams, which consist of participants or other teams (UML class diagram).

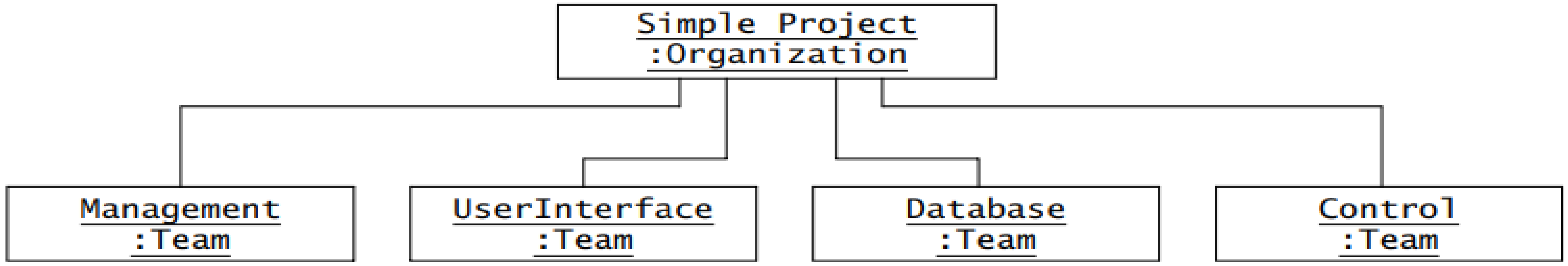


Figure 3-4 Example of a simple project organization (UML instance diagram). Reporting, deciding, and communicating are all made via the aggregation association of the organization.

Project: Type of Interactions

- **Reporting**. This type of interaction is used for reporting status information. For example, a developer reports to another developer that an API (Application Programmer Interface) is ready, or a team leader reports to a project manager that an assigned task has not yet been completed.
- **Decision**. This type of interaction is used for propagating decisions. For example, a team leader decides that a developer has to publish an API, a project manager decides that a planned delivery must be moved up in time. Another type of decision is the resolution of an issue.
- **Communication**. This type of interaction is used for exchanging all the other types of information needed for decision or status. Communication can take many flavors. Examples are the exchange of requirements or design models or the creation of an argument to support a proposal. An invitation to eat lunch is also a communication.

Reporting Structure: Hierarchical

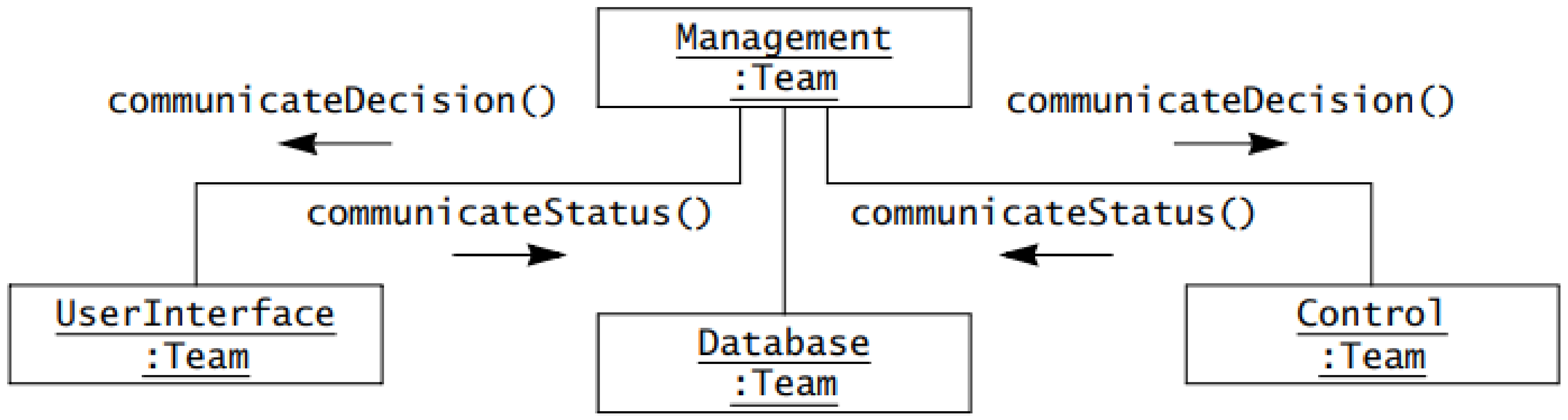


Figure 3-5 Example of reporting structure in a hierarchical organization (UML communication diagram). Status information is reported to the project manager, and corrective decisions are communicated back to the teams by the team leaders. The team leaders and the project manager are called the management team.

Communication Structure: liaison-based

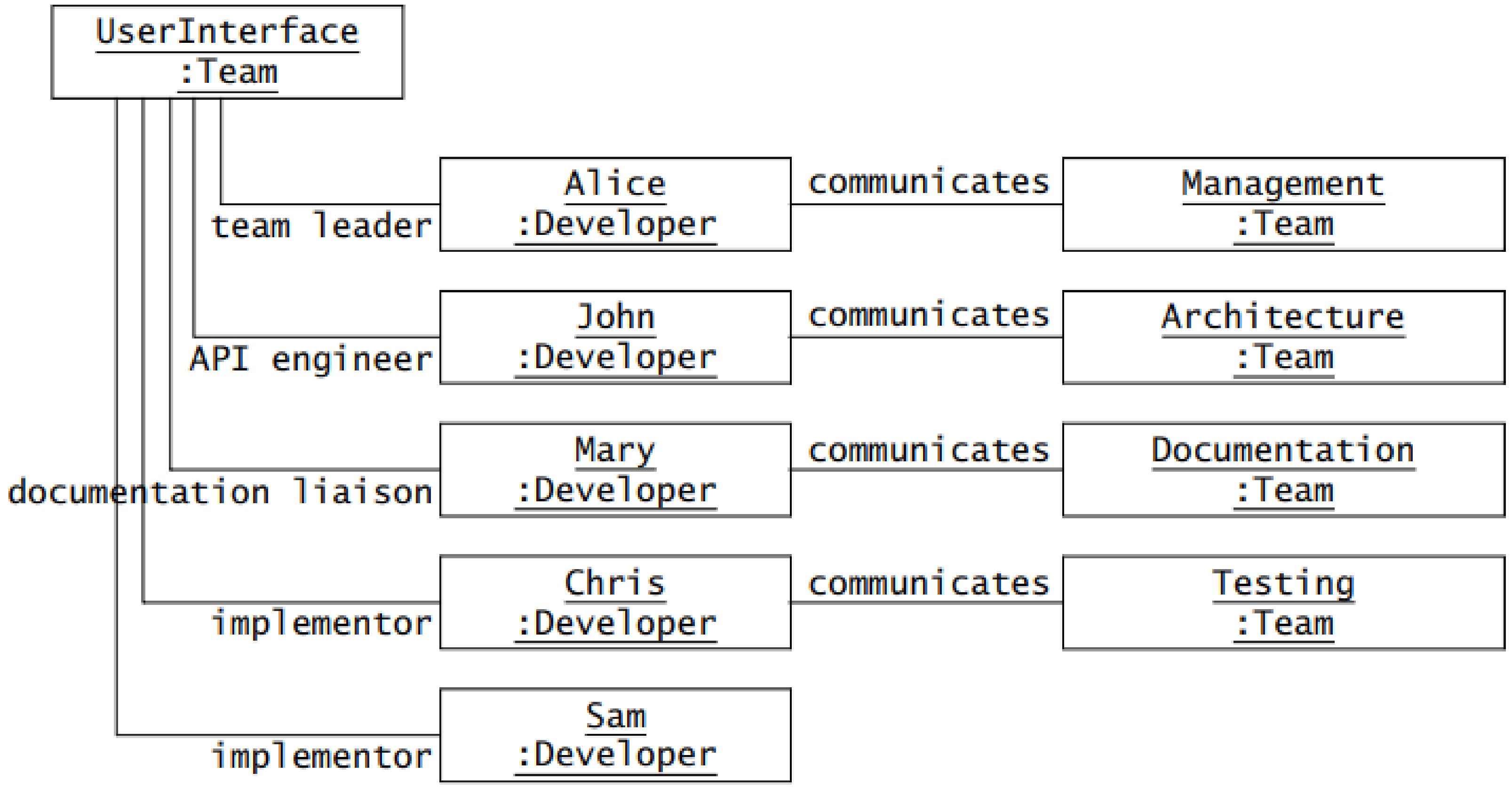
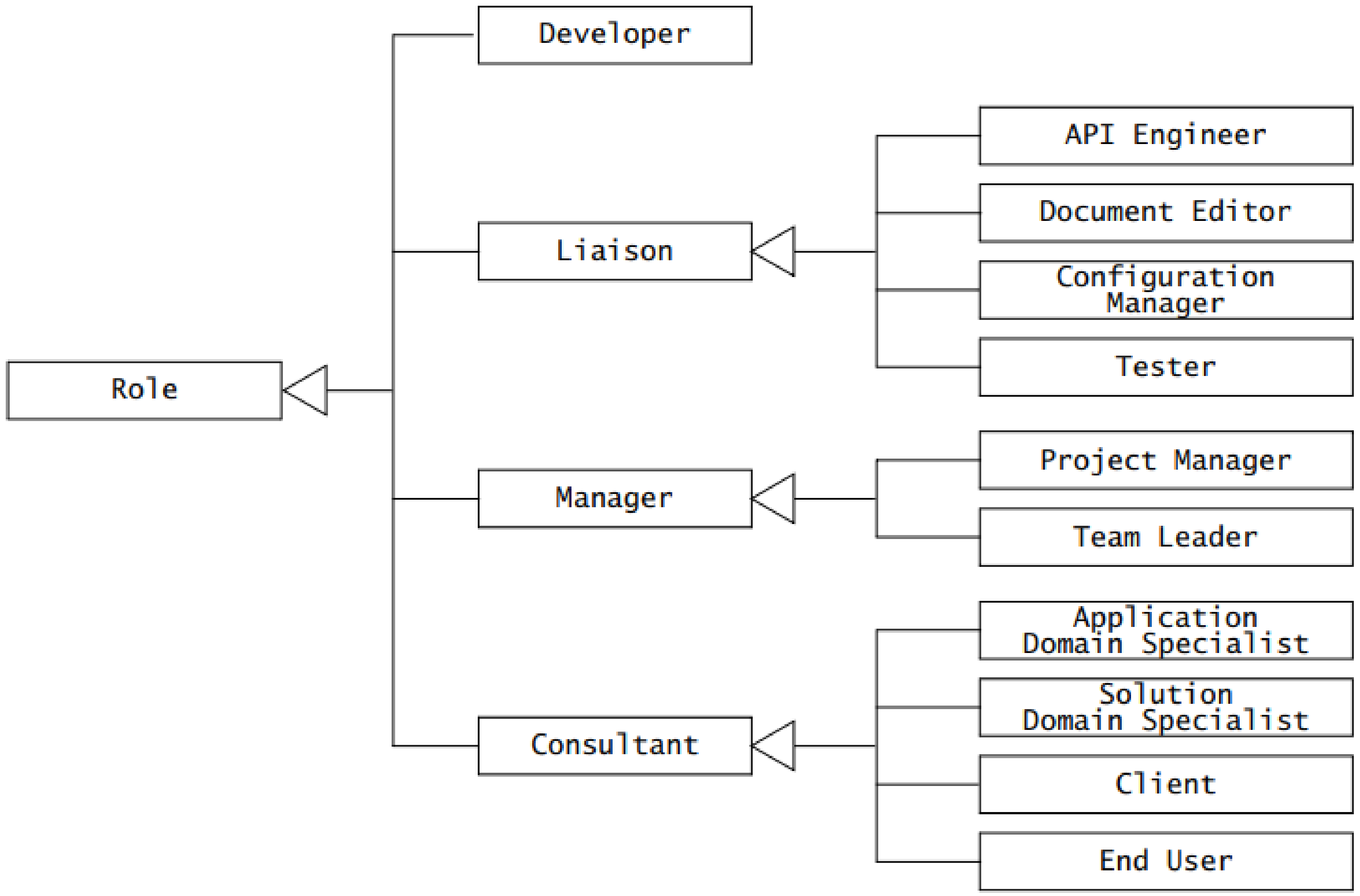


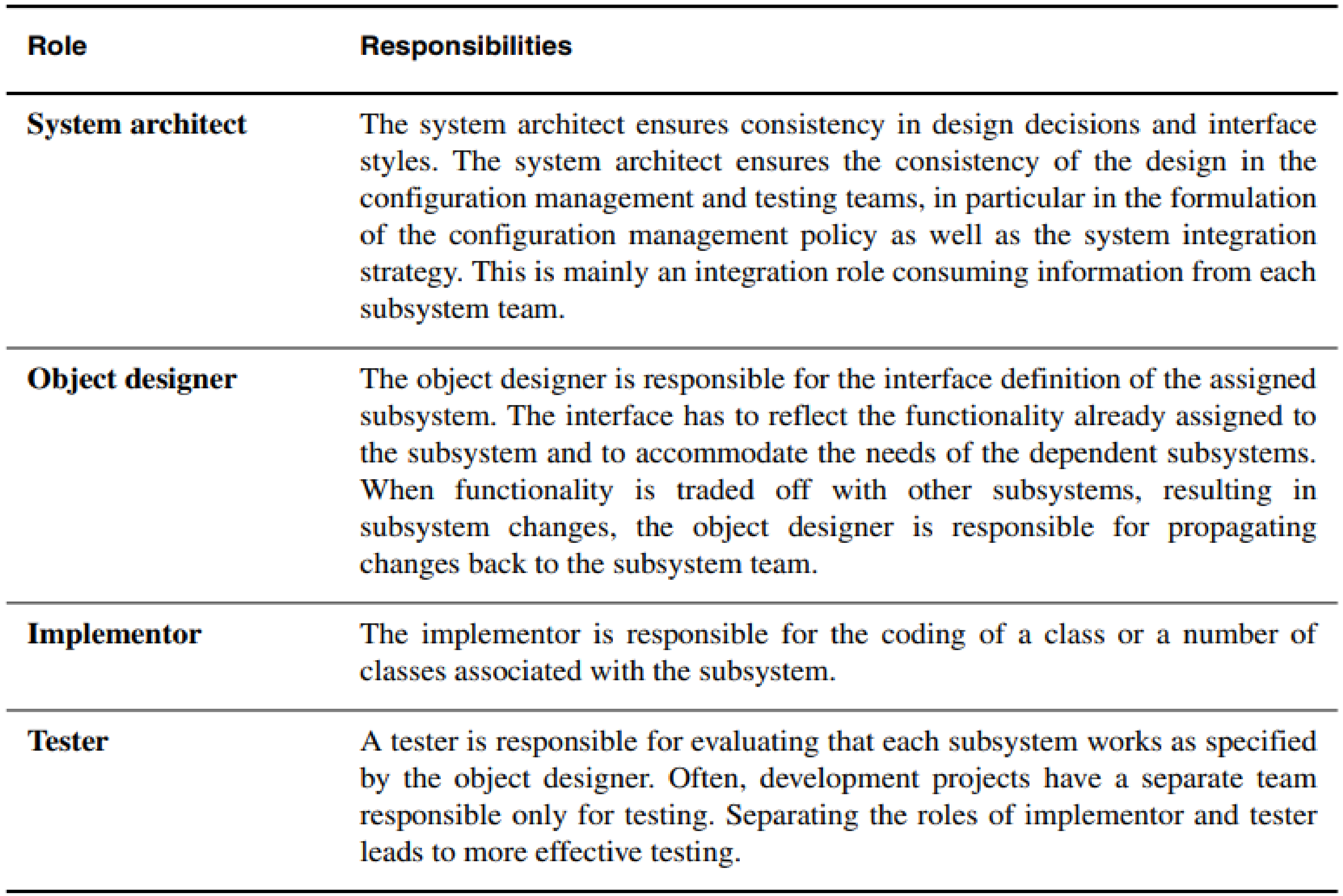
Figure 3-6 Examples of a liaison-based communication structure (UML object diagram). The team is composed of five developers. Alice is the team leader, also called the liaison to the management team. John is the API engineer, also called the liaison to the architecture team. Mary is the liaison to the documentation team. Chris and Sam are implementors and interact with other teams only informally.

Role



Role
Developer
Liaison
Manager
Consultant
API Engineer
Document Editor
Configuration Manager
Tester
Project Manager
Team Leader
Application Domain Specialist
Solution Domain Specialist
Client
End User

Role: Some Examples



Role	Responsibilities
System architect	The system architect ensures consistency in design decisions and interface styles. The system architect ensures the consistency of the design in the configuration management and testing teams, in particular in the formulation of the configuration management policy as well as the system integration strategy. This is mainly an integration role consuming information from each subsystem team.
Object designer	The object designer is responsible for the interface definition of the assigned subsystem. The interface has to reflect the functionality already assigned to the subsystem and to accommodate the needs of the dependent subsystems. When functionality is traded off with other subsystems, resulting in subsystem changes, the object designer is responsible for propagating changes back to the subsystem team.
Implementor	The implementor is responsible for the coding of a class or a number of classes associated with the subsystem.
Tester	A tester is responsible for evaluating that each subsystem works as specified by the object designer. Often, development projects have a separate team responsible only for testing. Separating the roles of implementor and tester leads to more effective testing.

Tasks and Work Product

- **Task**: A well-defined work assignment for a role.
- **Activity**: A group of related tasks.
- **Examples**
  - Role of a Project Manager/Team Leader:
    - Assign tasks to roles.
    - Monitor task progress and completion.
  - Participant’s Role:
    - Carry out the assigned task.
- Work Product:
  - A tangible item resulting from a task.
  - Examples:
    - Object model
    - Class diagram
    - Source code
    - Documents or parts of documents

Work Products, Deliverables, and Work Packages

- Deliverable:
  - A work product delivered to the client.
  - Includes:
    - Software system
    - Documentation
- Internal Work Products:
- Work products not visible to the client.
- Work Package:
  - Describes work needed to complete a task or activity.
  - Components of a Work Package:
    1. Task name and description
    2. Resources needed
    3. Dependencies:
       - Inputs: Work products from other tasks
       - Outputs: Work products from the current task
    4. Links to other tasks

SCHEDULING

- Split project into tasks (= create a WBS)
- Estimate time and resources required to complete each task.
- Organize tasks concurrently to make optimal

  use of workforce.
- Minimize task dependencies to avoid delays

  caused by one task waiting for another to complete.
- Dependent on project managers intuition and experience.

SCHEDULING TECHNIQUES

- Mathematical Analysis
  - Network Diagrams
    - ➢ critical path method (CPM)
    - ➢ Program evaluation and review technique (PERT)

- Bar Charts
  - Milestone Chart
  - Gantt Chart/Time-line chart

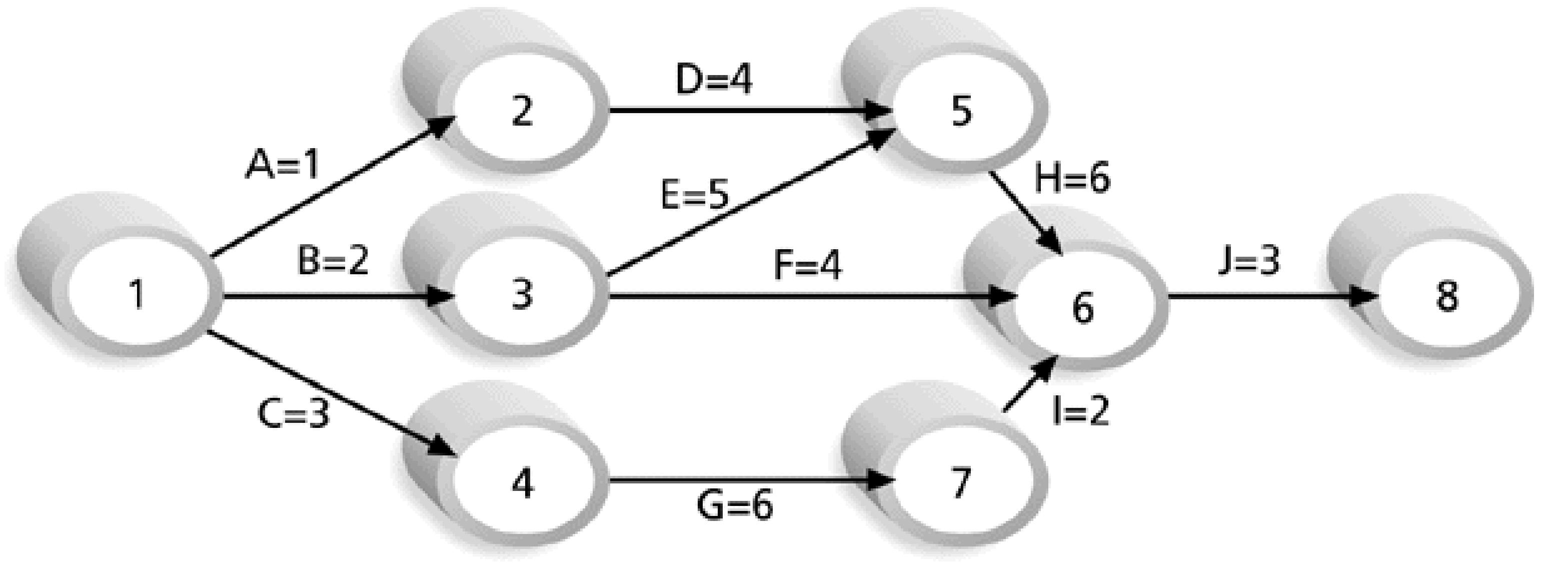
CPM

- CPM is a project network analysis technique used to predict total project duration
- A critical path for a project is the series of activities that determines the *earliest time* by which the project can be completed
- The critical path is the *longest path* through the network diagram and has the least amount of slack or float

CPM

- First develop a good project network diagram
- Add the durations for all activities on each path through the project network diagram
- The longest path is the critical path

CPM



Note: Assume all durations are in days.

Path 1:	A-D-H-J	Length = 1+4+6+3 = 14 days
Path 2:	**B-E-H-J**	**Length = 2+5+6+3 = 16 days**
Path 3:	B-F-J	Length = 2+4+3 = 9 days
Path 4:	C-G-I-J	Length = 3+6+2+3 = 14 days

Since the critical path is the longest path through the network diagram, Path 2, B-E-H-J, is the critical path for Project X.

Determining the Critical Path for Project X

PERT

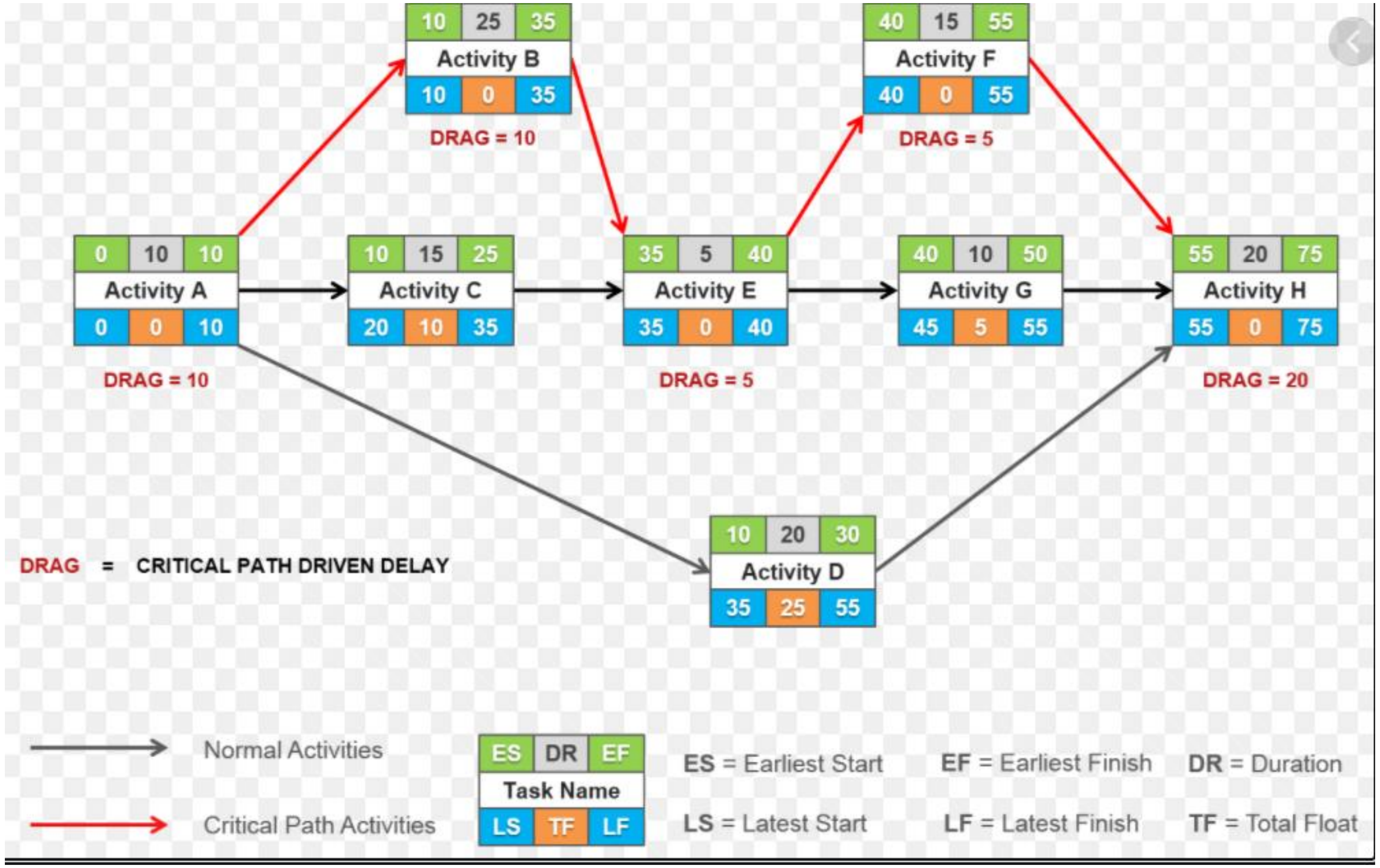
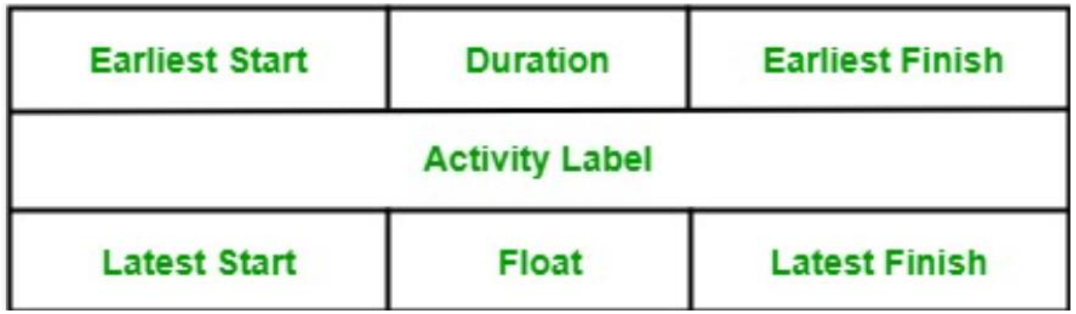


Image- https://www.project-risk-manager.com/blog/pert-vs-cpm/

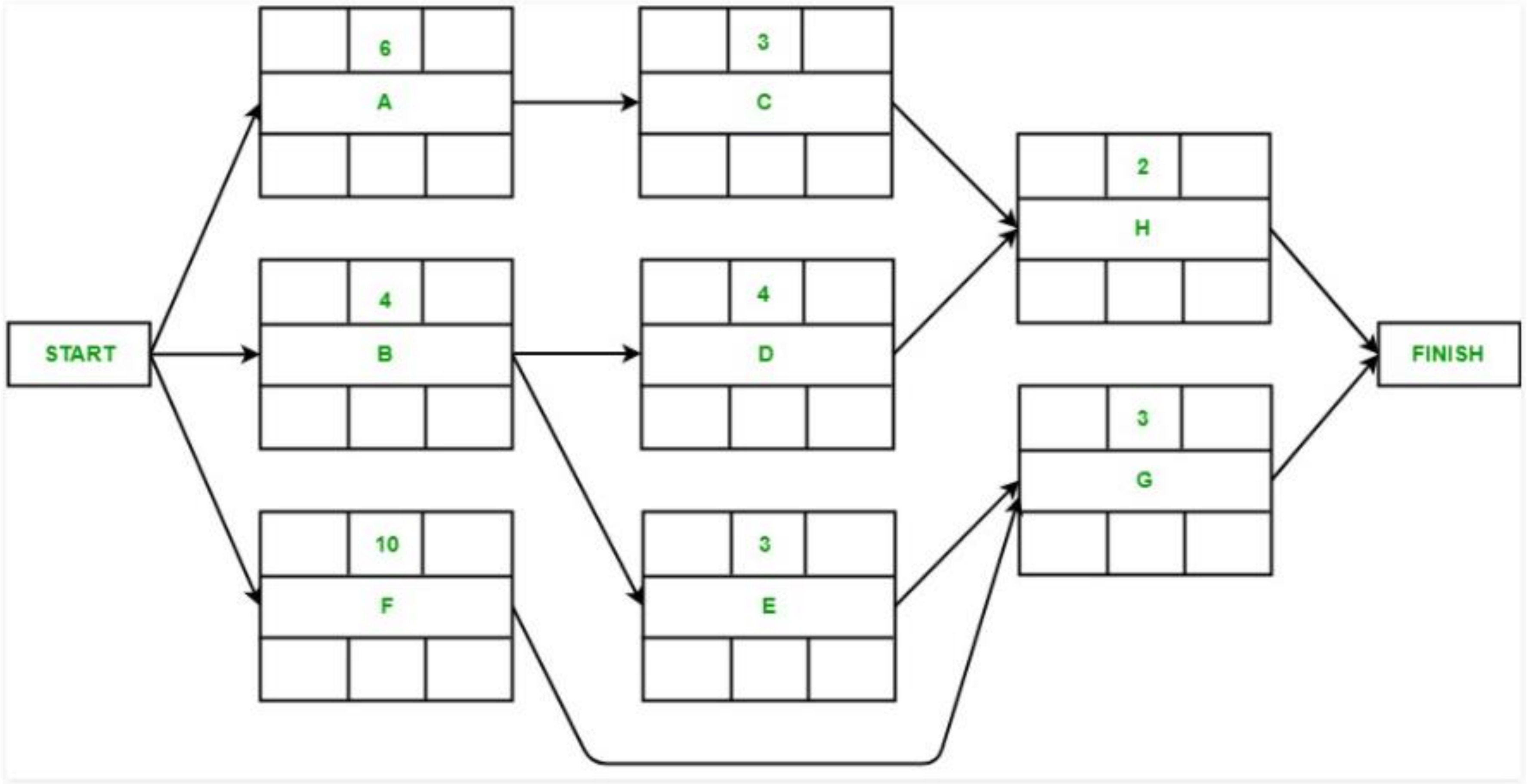
Activity	Duration (in weeks)	Precedents
A	6	–
B	4	–
C	3	A
D	4	B
E	3	B
F	10	–
G	3	E, F
H	2	C, D



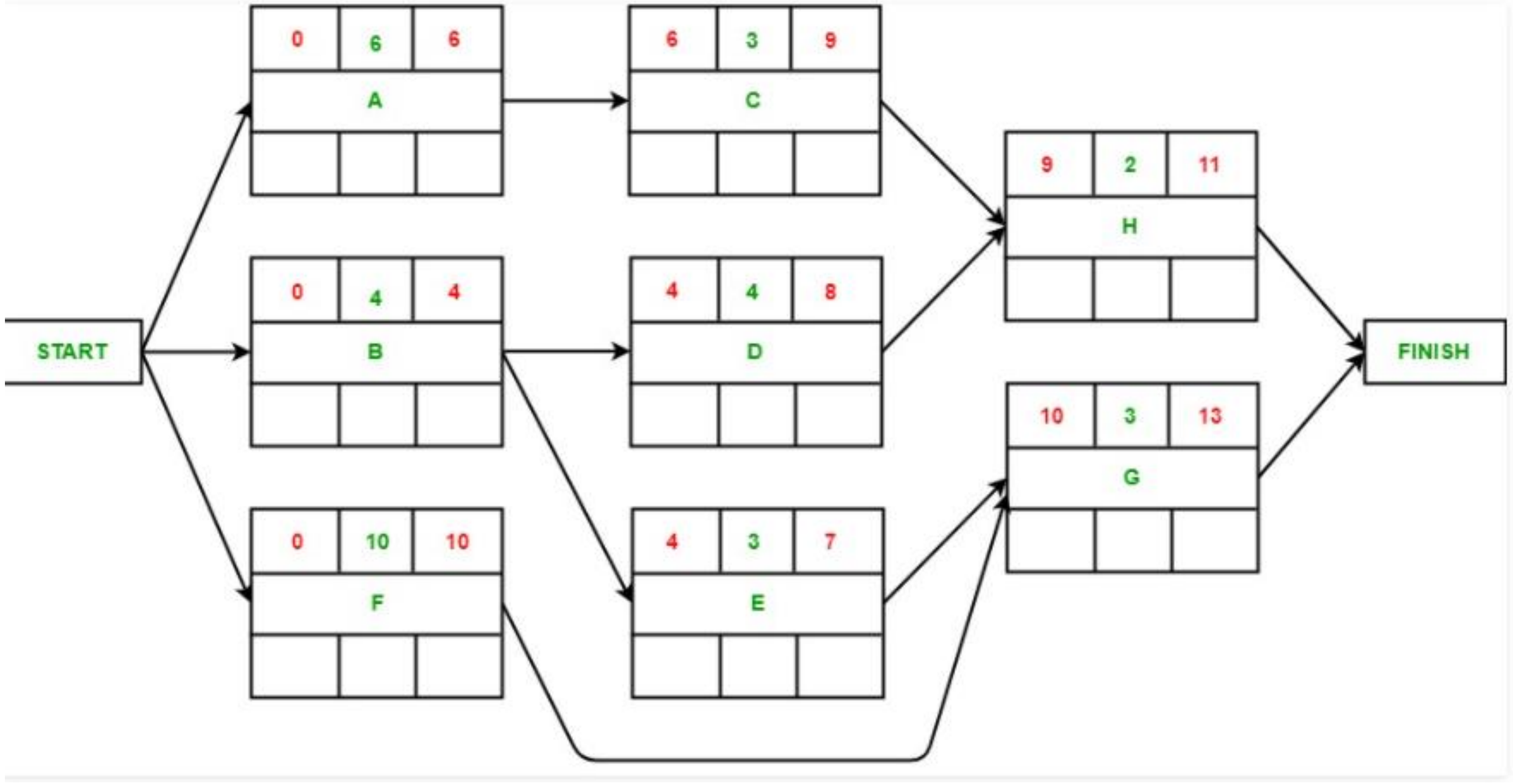
Earliest Start	Duration	Earliest Finish
Activity Label		
Latest Start	Float	Latest Finish

Make PERT diagram and find the critical path on it.

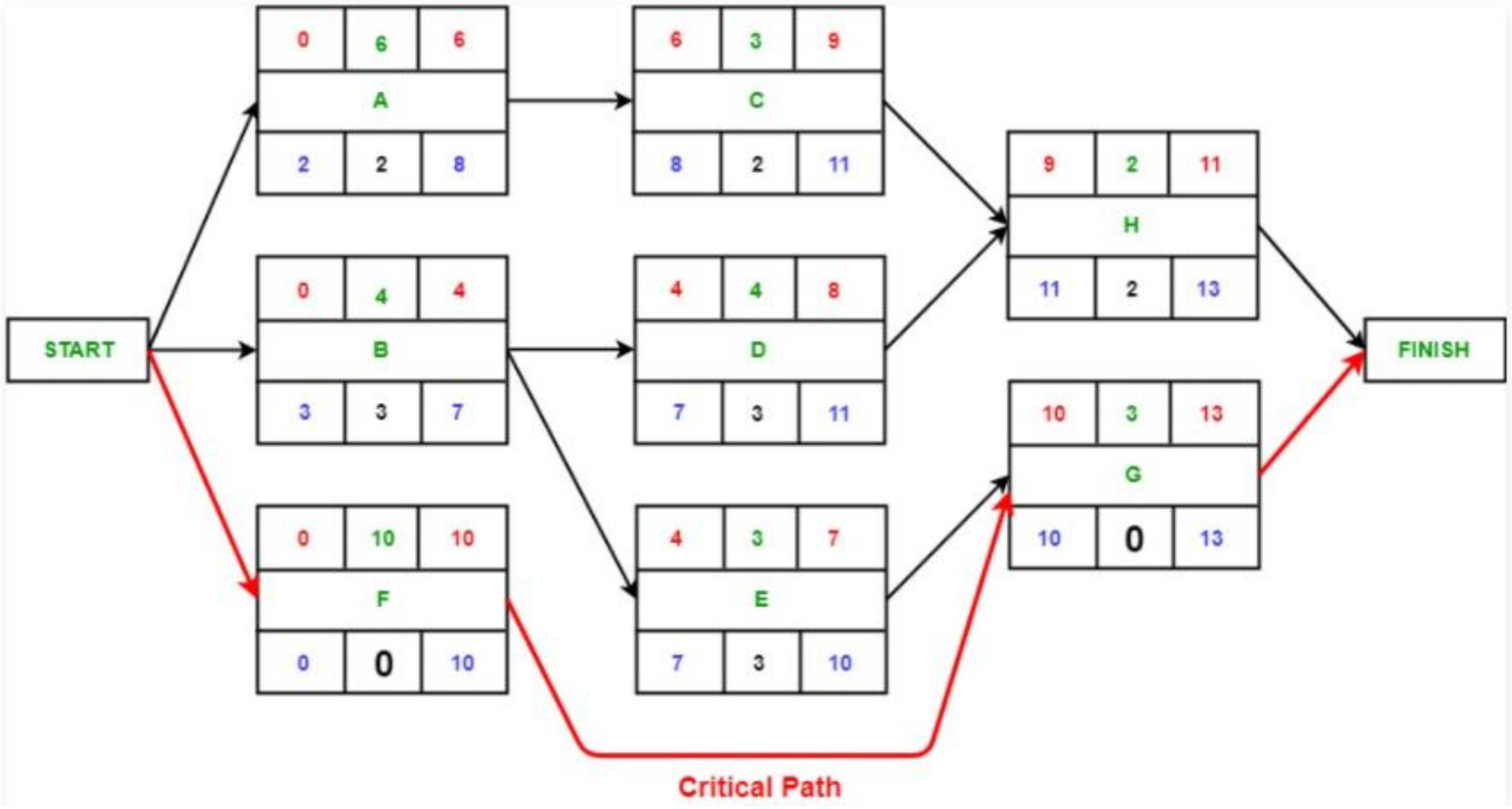
Ref: https://www.geeksforgeeks.org/software-engineering-critical-path-method/



Ref: https://www.geeksforgeeks.org/software-engineering-critical-path-method/



Ref: https://www.geeksforgeeks.org/software-engineering-critical-path-method/

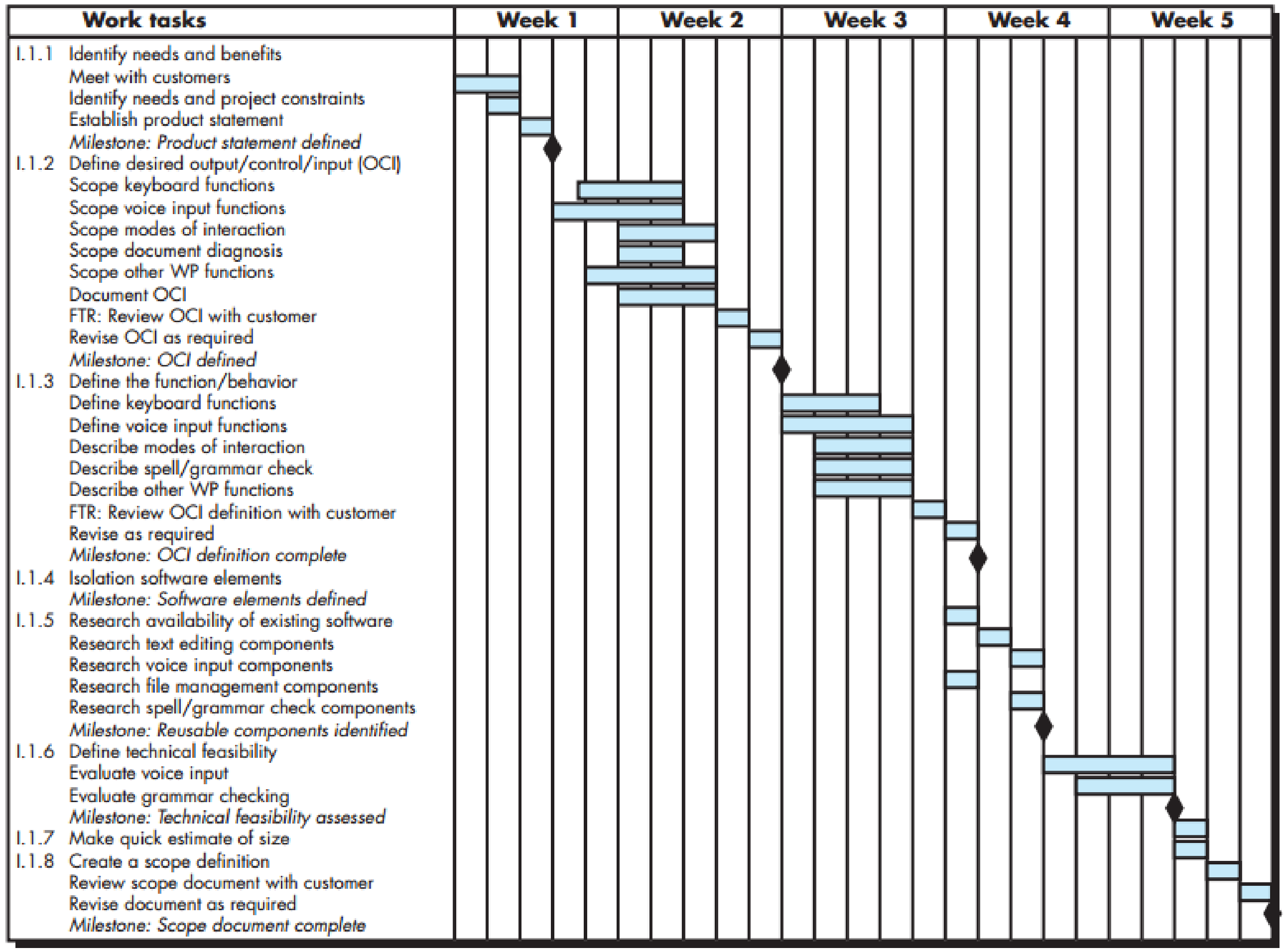

START
A
0 6 6
2 2 8
B
0 4 4
3 3 7
F
0 10 10
0 0 10
C
6 3 9
8 2 11
D
4 4 8
7 3 11
E
4 3 7
7 3 10
H
9 2 11
11 2 13
G
10 3 13
10 0 13
FINISH
Critical Path

Brain storming:

- CPM Vs PERT
- Practice PERT, CPM, GANTT CHART

TIME-LINE CHART/GANTT CHART

FIGURE 34.3 An example time-line chart



Project Communication

Planned Vs Unplanned Communications?

Planned vs Unplanned Communications

- Planned Communications Examples
  - problem inspection, during which developers gather information from the problem statement, the client, and the user about their needs and the application domain
  - status meetings, during which teams review their progress
  - peer reviews, during which team members identify defects and find solutions in preliminary work products
  - client and project reviews, during which the client or project members review the quality of a work product, in particular deliverables
  - releases, during which project participants make available to the client and end users versions of the system and its documentation.

Planned vs Unplanned Communications

- Planned Communications Events
  - Problem presentation
  - Client reviews
  - Project reviews
  - Peer reviews
  - Status reviews
  - Brainstorming
  - Releases
  - Postmortem reviews.

Planned vs Unplanned Communications

- Unplanned Communications Examples
  - requests for clarification, during which participants request specific information from others about the system, the application domain, or the project
  - requests for change, during which participants describe problems encountered in the system or new features that the system should support
  - issue resolution, during which a conflict between different stakeholders is identified, solutions explored and negotiated, and a resolution agreed upon.

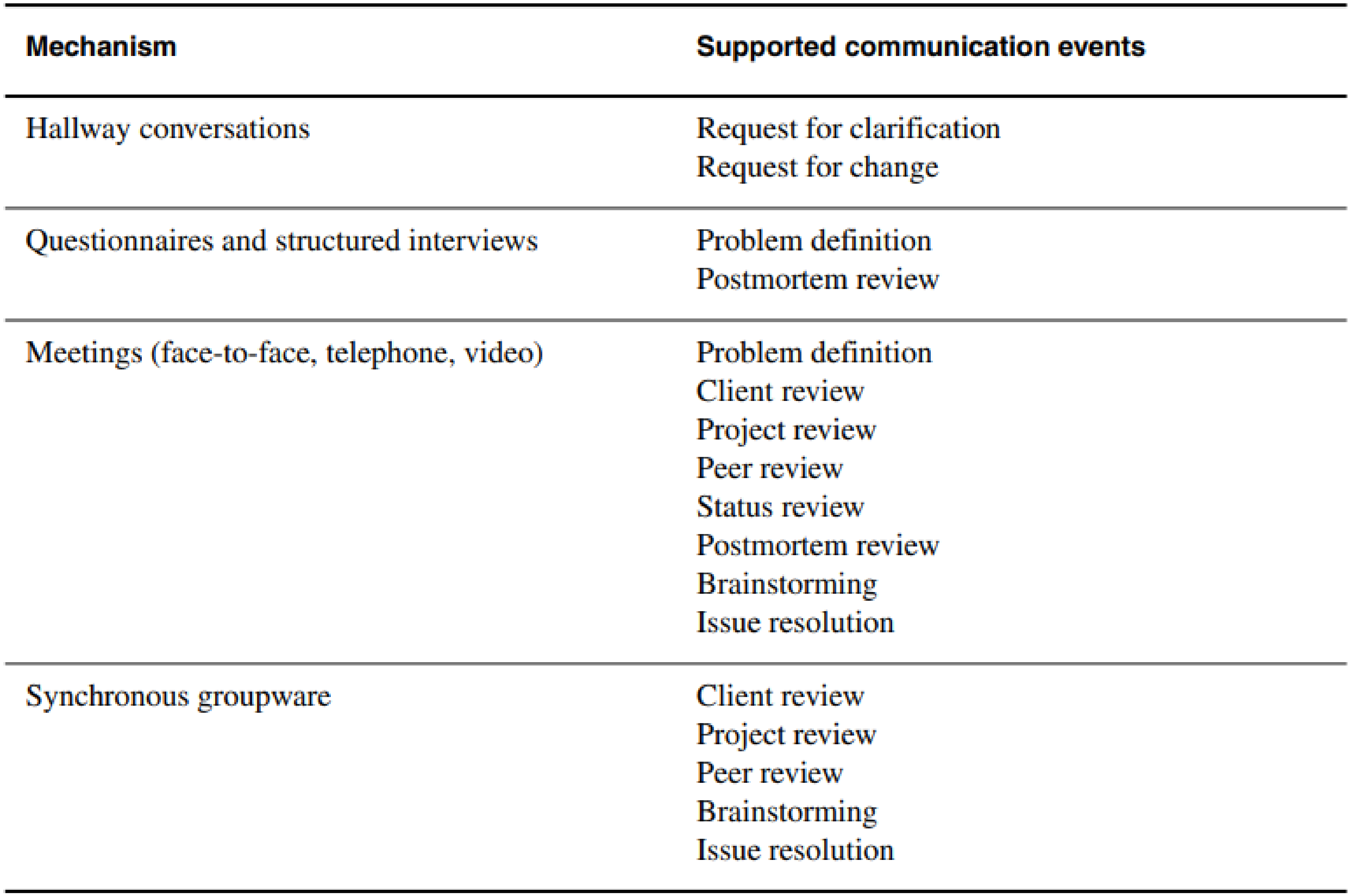
Planned vs Unplanned Communications

- Unplanned Communications Events
    - Requests for clarification
    - Requests for changes
    - Issue resolution.

Communication Mechanisms

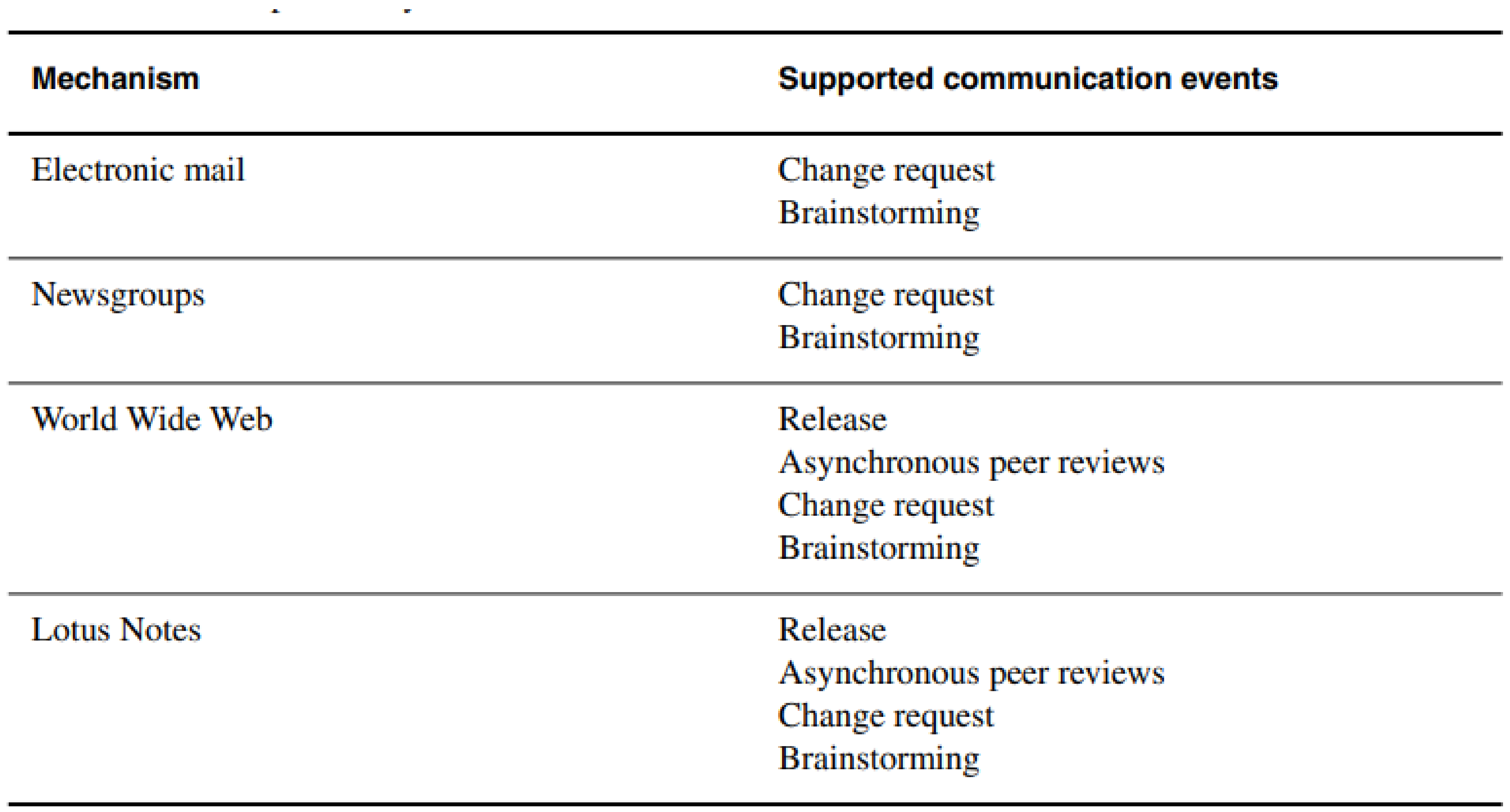
- Synchronous
- Asynchronous

Synchronous Communication Examples



Mechanism	**Supported communication events**
Hallway conversations	Request for clarification
	Request for change
Questionnaires and structured interviews	Problem definition
	Postmortem review
Meetings (face-to-face, telephone, video)	Problem definition
	Client review
	Project review
	Peer review
	Status review
	Postmortem review
	Brainstorming
	Issue resolution
Synchronous groupware	Client review
	Project review
	Peer review
	Brainstorming
	Issue resolution

Asynchronous Communication Examples



Mechanism	Supported communication events
Electronic mail	Change request Brainstorming
Newsgroups	Change request Brainstorming
World Wide Web	Release Asynchronous peer reviews Change request Brainstorming
Lotus Notes	Release Asynchronous peer reviews Change request Brainstorming

Communication Mechanisms and Tools

1. Synchronous Communication Tools
   - Meetings: In-person, virtual (Zoom, Microsoft Teams, Google Meet)
   - Chat Applications: Slack, Microsoft Teams, Discord
   - Phone/Voice Calls: Traditional phone calls or VoIP services
   - Pair Programming Platforms: Visual Studio Live Share, CodeTogether
2. Asynchronous Communication Tools
   - Email: Outlook, Gmail
   - Discussion Forums: GitHub Discussions, Slack Threads, Microsoft Teams Channels
   - Document Sharing Platforms: Google Docs, OneDrive, SharePoint

Communication Mechanisms and Tools

3. Project Management and Collaboration Tools
    - Issue Tracking and Task Boards: Jira, Trello, Asana, ClickUp
    - Collaboration Software: Confluence, Notion, Miro, Mural
    - Version Control: GitHub, GitLab, Bitbucket

4. File and Documentation Management
    - File Sharing: Dropbox, Google Drive, OneDrive
    - Documentation Tools: Confluence, Notion, Google Docs, Markdown-based tools

Communication Mechanisms and Tools

5. Review and Feedback Tools
   - Code Review Platforms: GitHub Pull Requests, GitLab Merge Requests
   - Testing and Bug Tracking Tools: Bugzilla, TestRail, Zephyr
   - Feedback Forms: Google Forms, Typeform
6. Reporting and Visualization Tools
   - Dashboards and Reporting: Tableau, Power BI, Jira Dashboards
   - Time Tracking Tools: Toggl, Harvest, Clockify
7. Hybrid Communication Tools
   - All-in-One Tools: Microsoft Teams, Slack, Zoom (support chat, calls, and file sharing)

SOFTWARE MEETINGS

Importance and Roles in Face-to-Face Meetings

Key Benefits of Meetings:

- Enable participants to share, review, and negotiate solutions
- Effective for resolving issues and building consensus

Challenges of Meetings:

- High resource cost
- Difficult to manage

Key Roles in Meetings:

- **Facilitator:**
  - Organizes and guides the meeting
  - Prepares and shares the agenda before the meeting
- **Minute Taker:**
  - Records the meeting notes and organizes them for distribution
  - Enables sharing outcomes with absent members
- **Timekeeper:**
  - Tracks time during the meeting
  - Ensures the agenda is followed, and discussions stay on track

Meeting Structure and Distributed Meetings

Meeting Agenda Structure:

1. **Header**: Meeting location, time, and participants
2. **Reports**: Items participants will report on
3. **Discussion Items**: Issues to be discussed, including allocated time per item

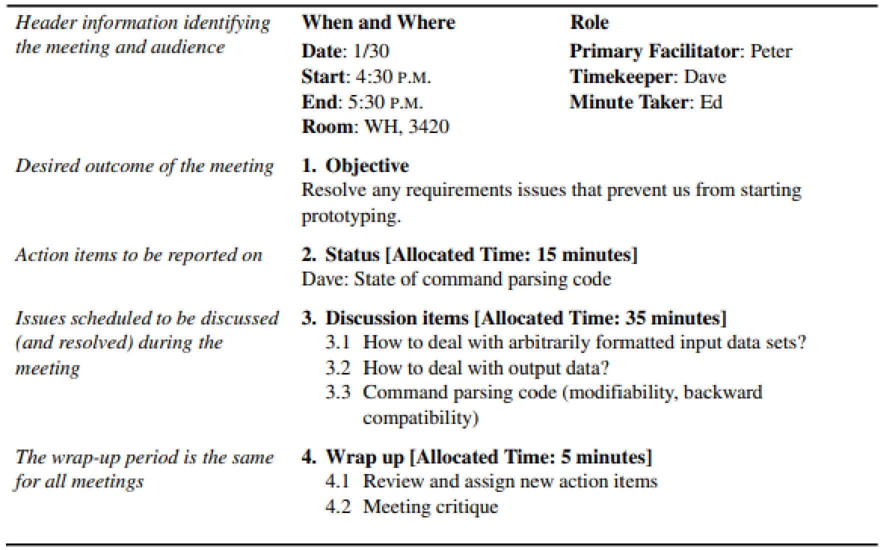
Meeting Minutes Structure:

- **Header**: Actual meeting details
- **Information Sharing**: Key information shared during the meeting
- **Decisions**: Decisions made or unresolved
- **Action Items**: Follow-up actions for participants

Distributed Meetings:

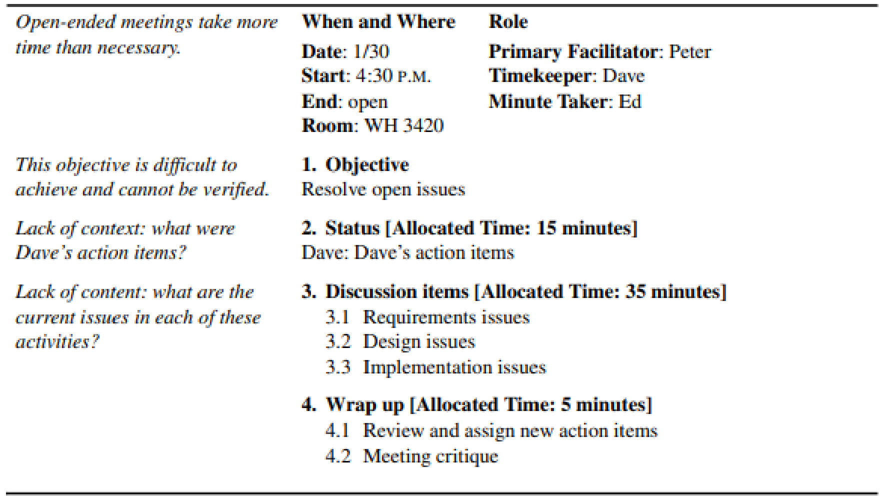
- Use **teleconferencing** or **video conferencing** to reduce costs
- Challenges: Lower bandwidth, reliability, and communication quality
- **Well-structured agenda** and familiarity with participant voices are crucial

Meeting Structure and Distributed Meetings



Header information identifying the meeting and audience	**When and Where** **Date**: 1/30 **Start**: 4:30 P.M. **End**: 5:30 P.M. **Room**: WH, 3420	**Role** **Primary Facilitator**: Peter **Timekeeper**: Dave **Minute Taker**: Ed
Desired outcome of the meeting	**1. Objective** Resolve any requirements issues that prevent us from starting prototyping.	
Action items to be reported on	**2. Status [Allocated Time: 15 minutes]** Dave: State of command parsing code	
Issues scheduled to be discussed (and resolved) during the meeting	**3. Discussion items [Allocated Time: 35 minutes]** 3.1 How to deal with arbitrarily formatted input data sets? 3.2 How to deal with output data? 3.3 Command parsing code (modifiability, backward compatibility)	
The wrap-up period is the same for all meetings	**4. Wrap up [Allocated Time: 5 minutes]** 4.1 Review and assign new action items 4.2 Meeting critique	

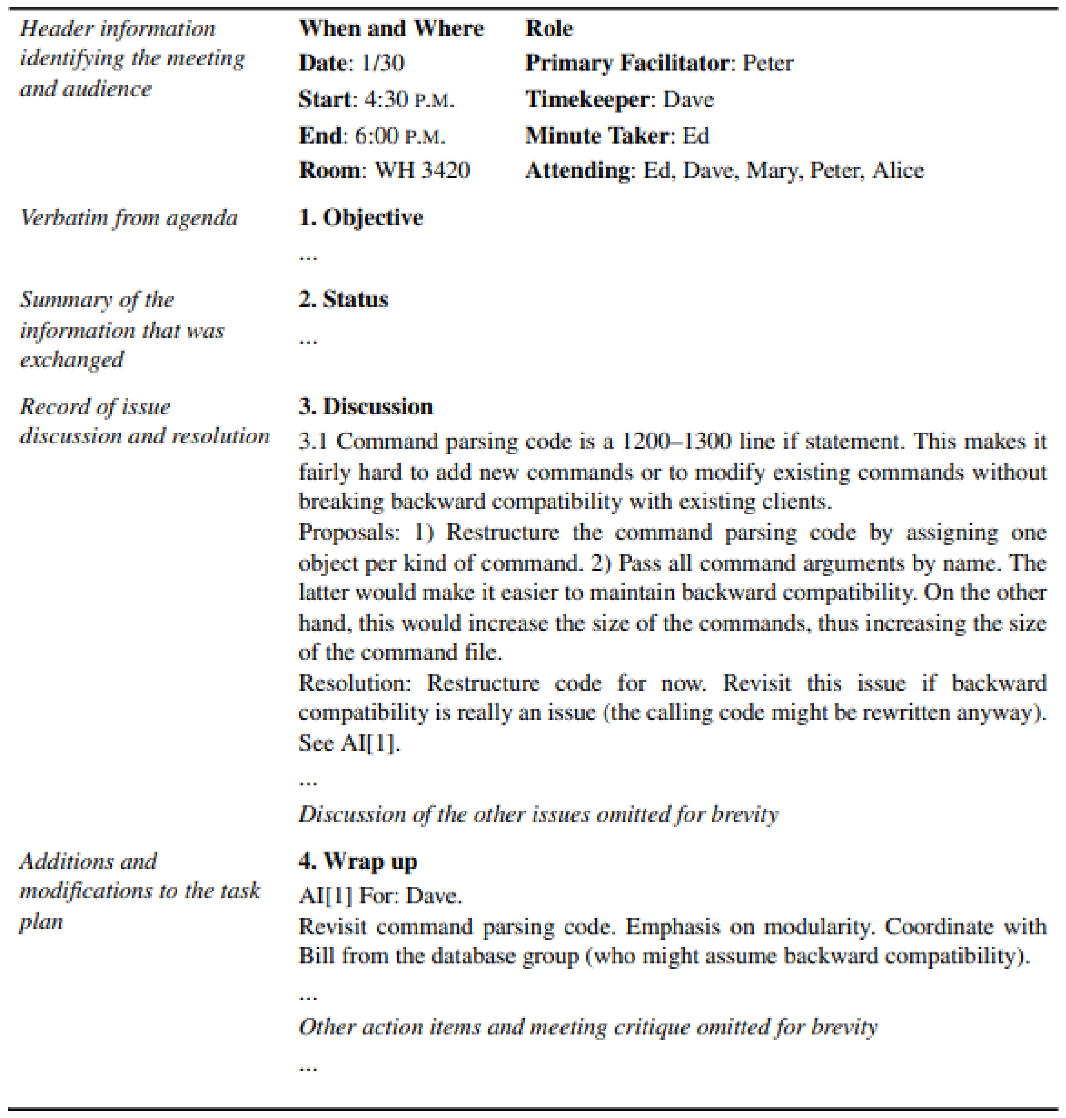
Figure 3-21 An example of a meeting agenda.



Open-ended meetings take more time than necessary.	**When and Where** **Date**: 1/30 **Start**: 4:30 P.M. **End**: open **Room**: WH 3420	**Role** **Primary Facilitator**: Peter **Timekeeper**: Dave **Minute Taker**: Ed
This objective is difficult to achieve and cannot be verified.	**1. Objective** Resolve open issues	
Lack of context: what were Dave's action items?	**2. Status [Allocated Time: 15 minutes]** Dave: Dave's action items	
Lack of content: what are the current issues in each of these activities?	**3. Discussion items [Allocated Time: 35 minutes]** 3.1 Requirements issues 3.2 Design issues 3.3 Implementation issues	
	4. Wrap up [Allocated Time: 5 minutes] 4.1 Review and assign new action items 4.2 Meeting critique	

Figure 3-22 An example of poor meeting agenda.

Meeting Minute



Header information identifying the meeting and audience	**When and Where** **Date**: 1/30 **Start**: 4:30 P.M. **End**: 6:00 P.M. **Room**: WH 3420	**Role** **Primary Facilitator**: Peter **Timekeeper**: Dave **Minute Taker**: Ed **Attending**: Ed, Dave, Mary, Peter, Alice
Verbatim from agenda	**1. Objective** ...	
Summary of the information that was exchanged	**2. Status** ...	
Record of issue discussion and resolution	**3. Discussion** 3.1 Command parsing code is a 1200–1300 line if statement. This makes it fairly hard to add new commands or to modify existing commands without breaking backward compatibility with existing clients. Proposals: 1) Restructure the command parsing code by assigning one object per kind of command. 2) Pass all command arguments by name. The latter would make it easier to maintain backward compatibility. On the other hand, this would increase the size of the commands, thus increasing the size of the command file. Resolution: Restructure code for now. Revisit this issue if backward compatibility is really an issue (the calling code might be rewritten anyway). See AI[1]. ... *Discussion of the other issues omitted for brevity*	
Additions and modifications to the task plan	**4. Wrap up** AI[1] For: Dave. Revisit command parsing code. Emphasis on modularity. Coordinate with Bill from the database group (who might assume backward compatibility). ... *Other action items and meeting critique omitted for brevity* ...	

Figure 3-23 An example of meeting minutes.

SOFTWARE REVIEWS

Reference: Presentations slides from Dr. Jerry Gao

SOFTWARE REVIEWS

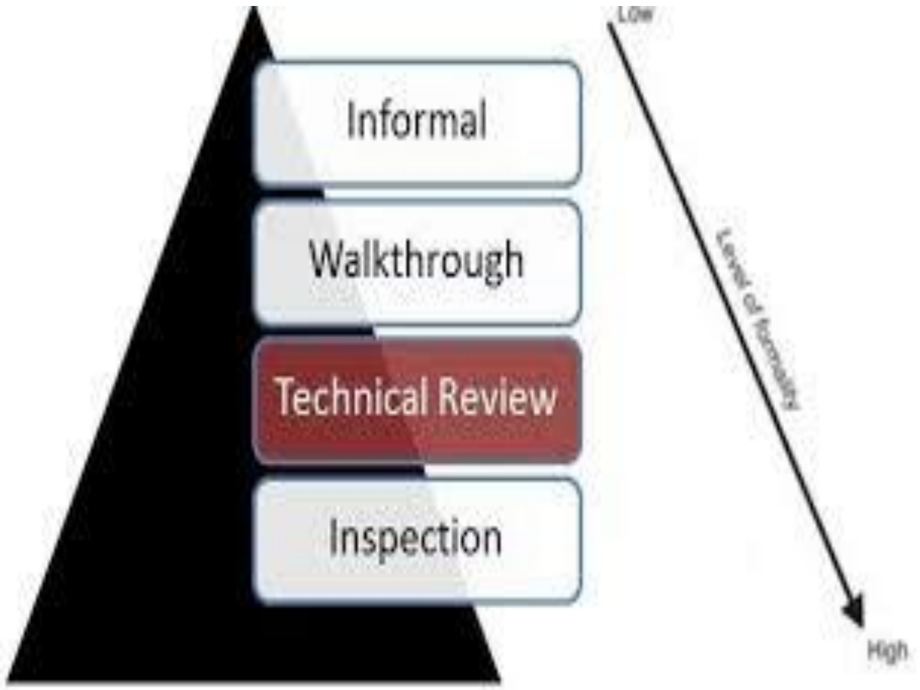
- A systematic inspection of a software by one or more individuals who work together to find and resolve errors and defects in the software during the early stages of Software Development Life Cycle (SDLC).

- Purpose
  - serves to uncover errors in analysis, design, coding, and testing.

- Why
  - To improve the productivity of the development team.
  - To make the testing process time and cost effective.
  - To make the final software with fewer defects.
  - To eliminate the inadequacies.

TYPES OF REVIEWS?

1. Informal reviews
   - informal meeting and informal desk checking
2. Formal reviews
   - Walkthrough, technical reviews, inspection



FORMAL TECHNICAL REVIEWS (FTR)

Objectives of FTR:

- To uncover errors in function, logic, or implementation
- To verify the software under review meets its requirements
- To ensure that the software has been represented according to predefined standards
- To develop software in a uniform manner
- To make projects more manageable

Purposes of FTR:

- Serves as a training ground for junior engineers
- Promote backup and continuity

FORMAL TECHNICAL REVIEWS (FTR)

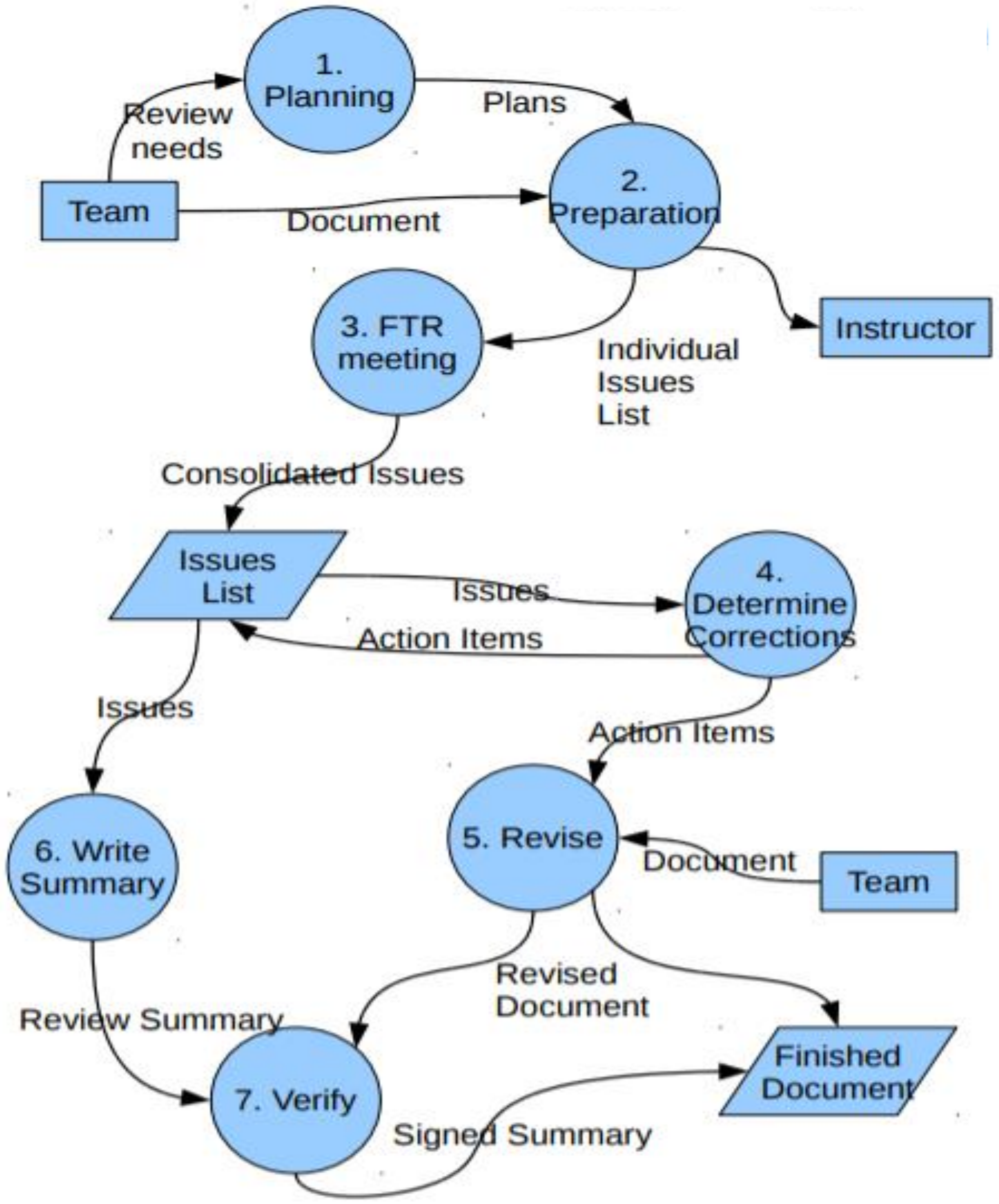
Review meeting's constraints

- 3-5 people involved in a review
- Advanced preparation (no more than 2 hours for each person)
- The duration of the review meeting should be less than 2 hours
- Focus on a specific part of a software product

People involved in a review meeting

- Producer, review leader, 2 or 3 reviewers (one of them is recorder)

FTR PROCESS DIAGRAM



Reference: http://users.csc.calpoly.edu/~jdalbey/308/Resources/ftr_tips.html#resources

BEFORE THE FTR

The preparation of a review meeting

- A meeting agenda and schedule (by review leader)
- Review material and distribution (by the producer)
- Review in advance (by reviewers)

DURING THE FTR

- Moderator, usually QA leader, is responsible for running the meeting and orchestrating how the issues are to be presented.
- Reviewers raise issues based on their advance preparation.
- Get the big problem issues on the table early. Don't get lost in minutely small clerical details.
- Identify problems, don't solve them.
- Focus on material, not the producer
- Elicit questions, don't answer them.
- Don't explain how your product works. An FTR is not a product demo.
- Watch the clock and keep momentum to get through all the issues in the alotted time.
- Recorder actively records all issues raised.

AT THE END OF THE FTR

- Meeting decisions
  - Accept the work product w/o further modification
  - Reject the work product due to errors
  - Accept the work under conditions (such as change and review)

AFTER FTR

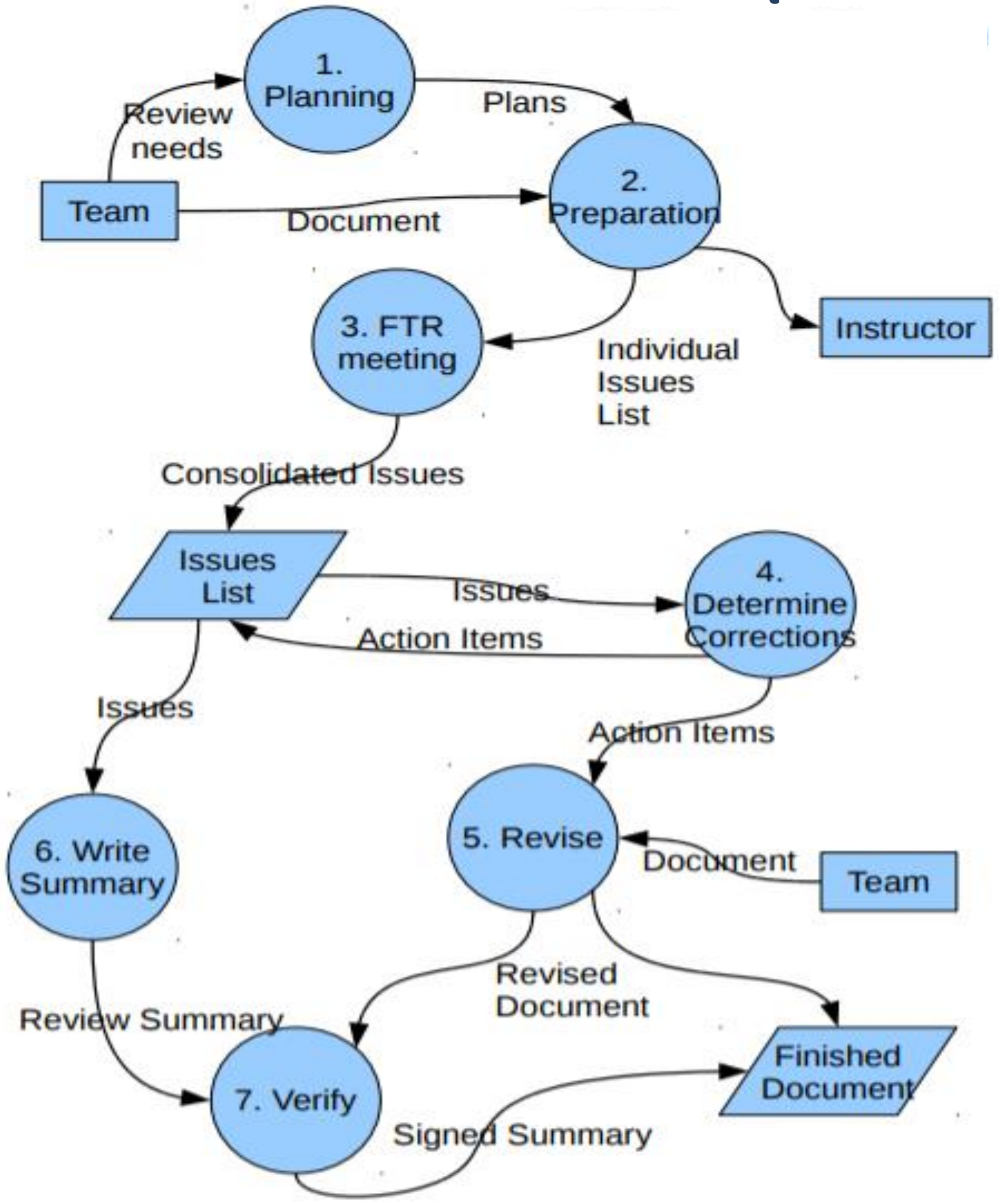
- Write up the issues list.
- Examine the issues raised, determine which to take action on.
- Assign each issue to someone as an action item and revise work product as needed.
- **Review issues list serves two purposes**
  - To identify problem areas in the project
  - To serve as an action item checklist (a follow-up procedure is needed)

- **Review summary report**
  - What was reviewed?
  - Who reviewed it?
  - What were the findings and conclusions?
- Have reviewers sign off on summary report, indicating proper corrective action has been taken for each issue.
- Include signed report in deliverable.

MINIMUM GUIDELINES FOR THE FTR

- Review the product, not the producer
- Set an agenda and maintain it
- Limit debate and rebuttal
- Enunciate problem areas, but don’t attempt to solve every problem noted
- Take written notes
- Limit the number of participants and insist upon advance preparation
- Develop a checklist for each work product that is likely to be reviewed
- Allocate resources and time schedule for FTRs
- Conduct meaningful training for all reviewers
- Review your early reviews

FTR PROCESS DIAGRAM (REVISIT)



Reference: http://users.csc.calpoly.edu/~jdalbey/308/Resources/ftr_tips.html#resources

Introduction to DevOps Tools

END OF UNIT 02