CSC301: Introduction to Software Engineering

Lecture 9

Wael Aboulsaadat

Software Testing

Conducting an inspection meeting

- 1. The moderator calls the meeting and distributes the documents.
- 2. The participants prepare for the meeting in advance.
- 3. At the start of the meeting, the moderator explains the procedures and verifies that everybody has prepared.
- 4. Paraphrasers take turns explaining the contents of the document or code, without reading it verbatim.
  - Requiring that the paraphraser not be the author ensures that the paraphraser say what he or she *sees*, not what the author *intended* to say.
- 5. Everybody speaks up when they notice a defect.

Inspecting compared to testing

- Both testing and inspection rely on different aspects of human intelligence.
- Testing can find defects whose consequences are obvious but which are buried in complex code.
- Inspecting can find defects that relate to maintainability or efficiency.
- The chances of mistakes are reduced if both activities are performed.

Testing or inspecting, which comes first?

- It is important to inspect software *before* extensively testing it.
- The reason for this is that inspecting allows you to quickly get rid of many defects.
- If you test first, and inspectors recommend that redesign is needed, the testing work has been wasted.
  - There is a growing consensus that it is most efficient to inspect software before *a*ny testing is done.
- Even before developer testing

Quality Assurance in General

- Root cause analysis
  - Determine whether problems are caused by such factors as
    - Lack of training
    - Schedules that are too tight
    - Building on poor designs or reusable technology

Measure quality and strive for continual improvement

- Things you can measure regarding the quality of a software product, and indirectly of the quality of the process
  - The number of failures encountered by users.
  - The number of failures found when testing a product.
  - The number of defects found when inspecting a product.
  - The percentage of code that is reused.
    - More is better, but don't count clones.
  - The number of questions posed by users to the help desk.
    - As a measure of usability and the quality of documentation.

Post-mortem analysis

- Looking back at a project after it is complete, or after a release,
  - You look at the design and the development process
  - Identify those aspects which, with benefit of hindsight, you could have done better
  - You make plans to do better next time

Process standards

- The personal software process (PSP):
  - Defines a disciplined approach that a developer can use to improve the quality and efficiency of his or her personal work.
  - One of the key tenets is personally inspecting your own work.
- The team software process (TSP):
  - Describes how teams of software engineers can work together effectively.
- The software capability maturity model (CMM):
  - Contains five levels, Organizations start in level 1, and as their processes become better they can move up towards level 5.
- ISO 9000-2:
  - An international standard that lists a large number of things an organization should do to improve their overall software process.

Difficulties and Risks in Quality Assurance

1) It is very easy to forget to test some aspects of a software system:
   - *'running the code a few times' is not enough.*
   - *Forgetting certain types of tests diminishes the system's quality.*

2) There is a conflict between achieving adequate quality levels, and 'getting the product out of the door'
   - *Create a separate department to oversee QA.*
   - *Publish statistics about quality.*
   - *Build adequate time for all activities.*

Difficulties and Risks in Quality Assurance

3) People have different abilities and knowledge when it comes to quality

- *Give people tasks that fit their natural personalities.*
- *Train people in testing and inspecting techniques.*
- *Give people feedback about their performance in terms of producing quality software.*
- *Have developers and maintainers work for several months on a testing team.*

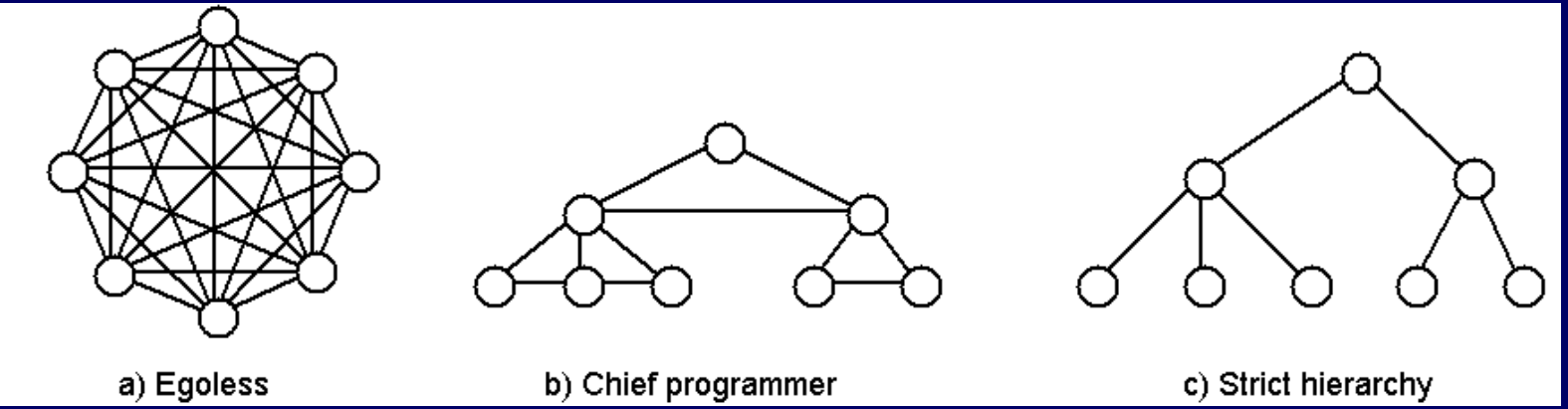
Software Project Management and Organization

Skills needed on a team

- Architect
- Project manager
- Configuration management and build specialist
- User interface specialist
- Development
  - Technology specialists
  - Hardware and third-party software specialist
  - Team Leader
- User documentation specialist
- Tester

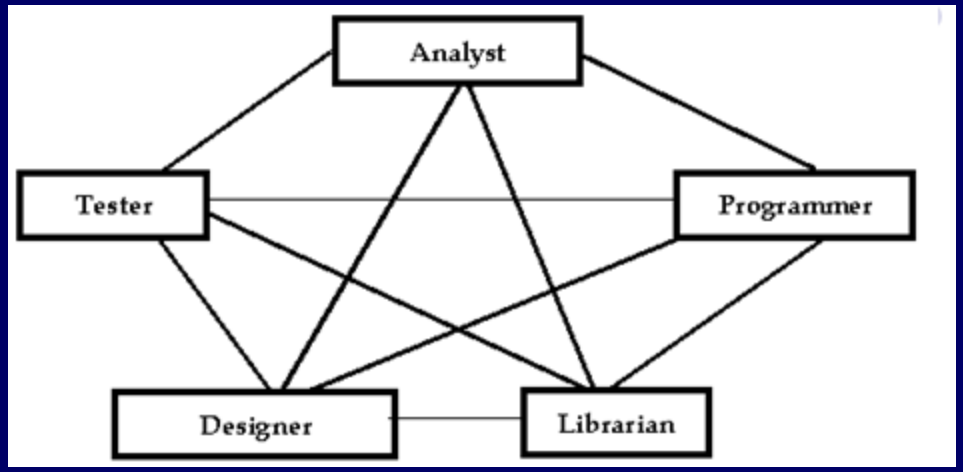
Building Software Engineering Teams

- Software engineering is a human process.
  - Choosing appropriate people for a team, and assigning roles and responsibilities to the team members, is therefore an important project management skill
  - Software engineering teams can be organized in many different ways



Software engineering teams

- Egoless team:
  - In such a team everybody is equal, and the team works together to achieve a common goal.
  - Decisions are made by consensus.
  - Most suited to difficult projects with many technical challenges.

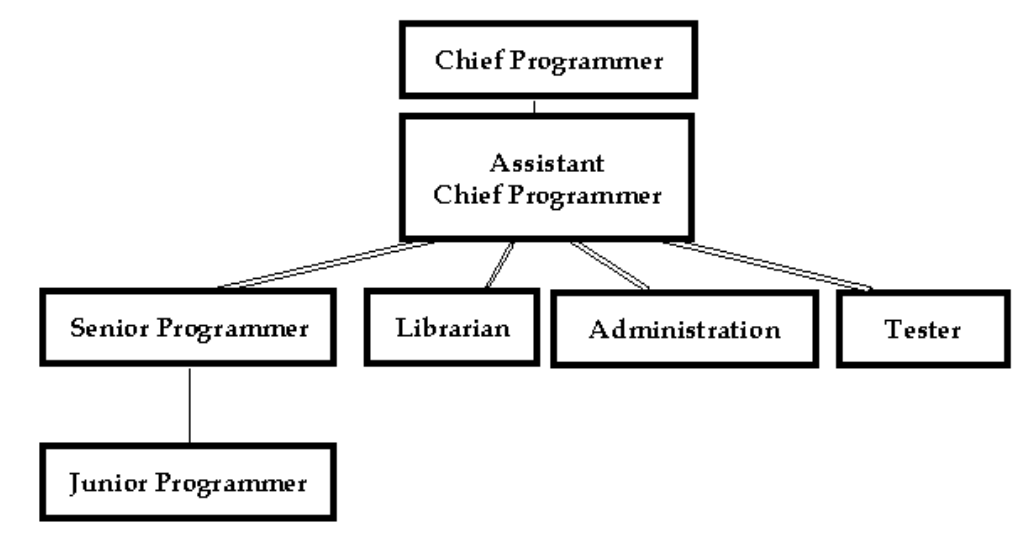


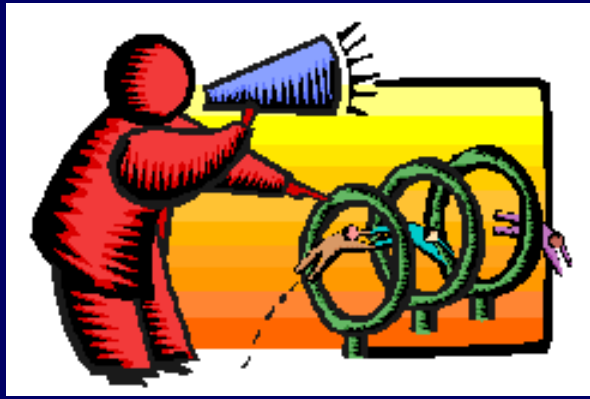
Software engineering teams

- Hierarchical manager-subordinate structure:
  - Each individual reports to a manager and is responsible for performing the tasks delegated by that manager.
  - Suitable for large projects with a strict schedule where everybody is well-trained and has a well-defined role.
  - However, since everybody is only responsible for their own work, problems may go unnoticed.

Software engineering teams

- Chief programmer team:
  - Midway between egoless and hierarchical.
  - The chief programmer leads and guides the project.
  - He or she consults with, and relies on, individual specialists.





Choosing an effective size for a team

- For a given estimated development effort, in person months, there is an optimal team size.
- Subsystems and teams should be sized such that the total amount of required knowledge and exchange of information is reduced.
- For a given project or project iteration, the number of people on a team will not be constant.

Project Scheduling and Tracking

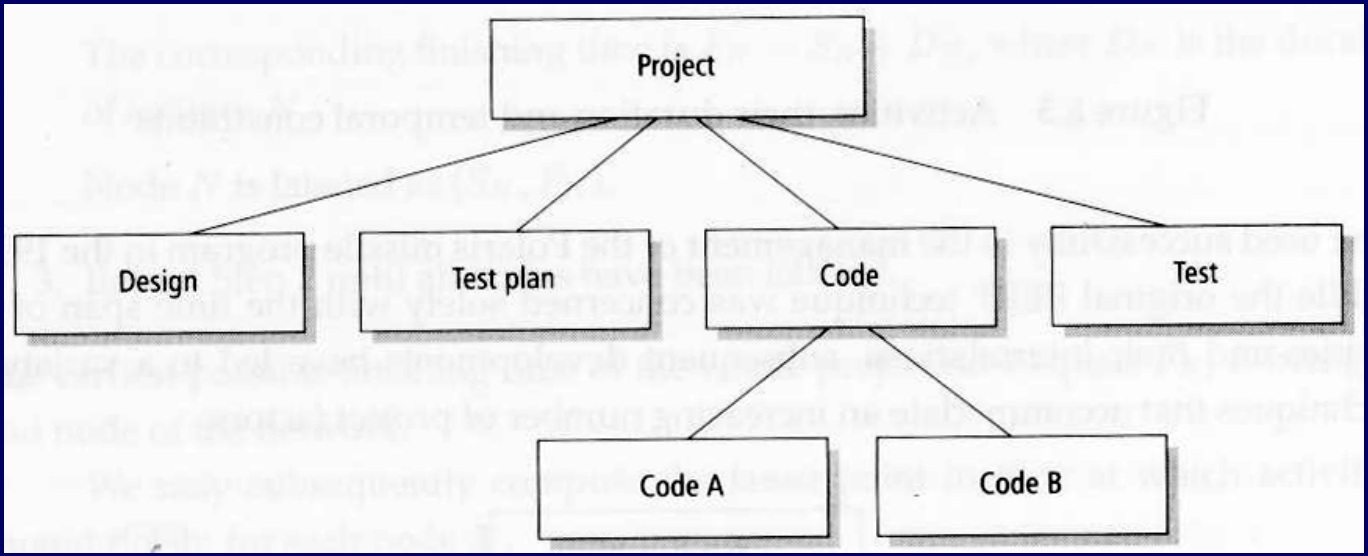
- *Scheduling* is the process of deciding:
  - In what sequence a set of activities will be performed.
  - When they should start and be completed.

- *Tracking* is the process of determining how well you are sticking to the cost estimate and schedule.

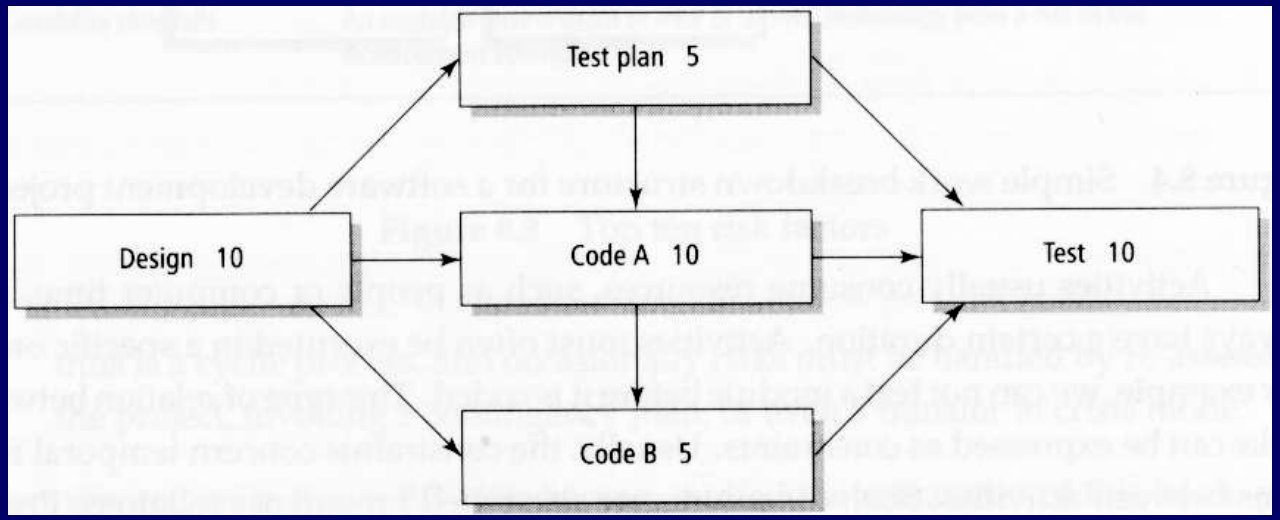
- Project Planning techniques:
  - WBS charts
  - PERT charts
  - Gantt charts

Work breakdown structure (WBS) chart

- Hierarchical decomposition of a project into subtasks
  - Shows how tasks are decomposed into subtasks
  - Does not show duration
  - Does not show precedence relations (e.g. task A must be finished before task B can start)

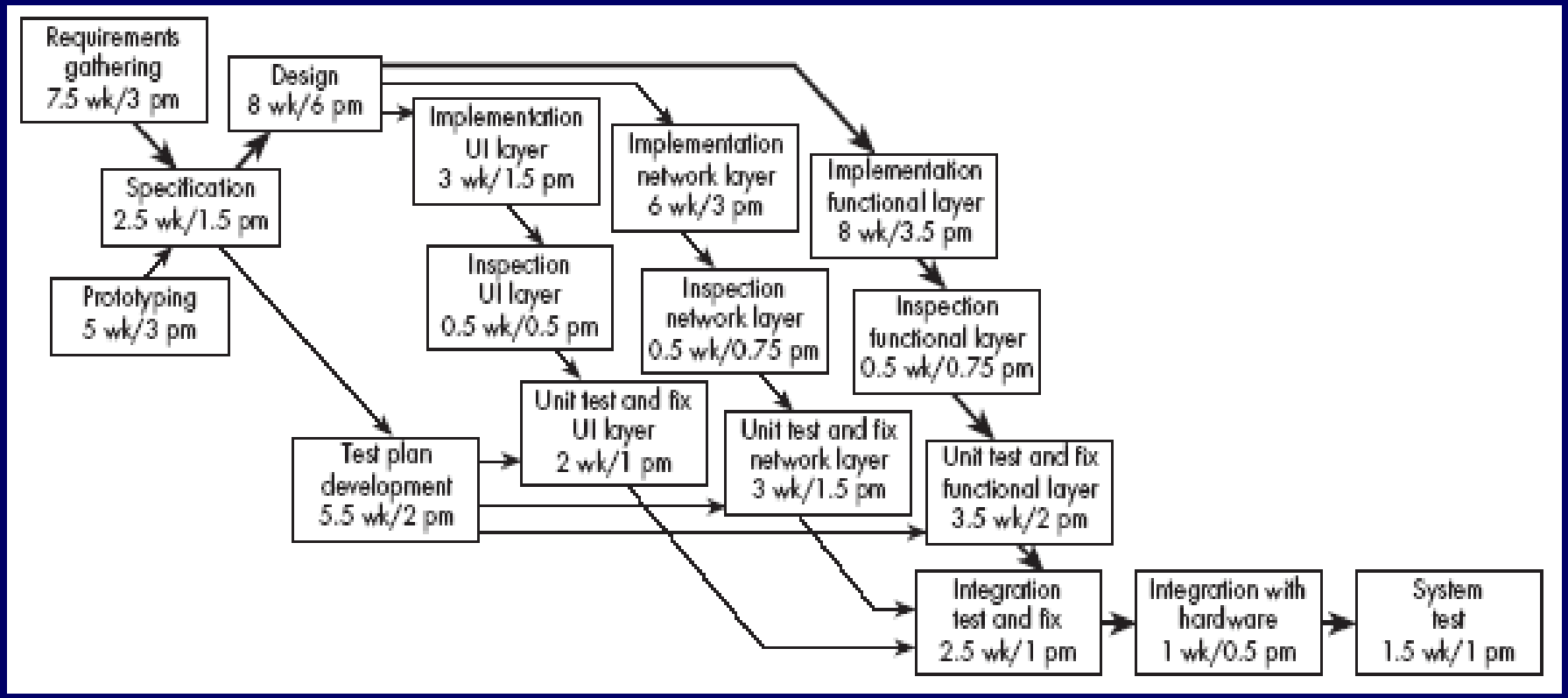


PERT charts

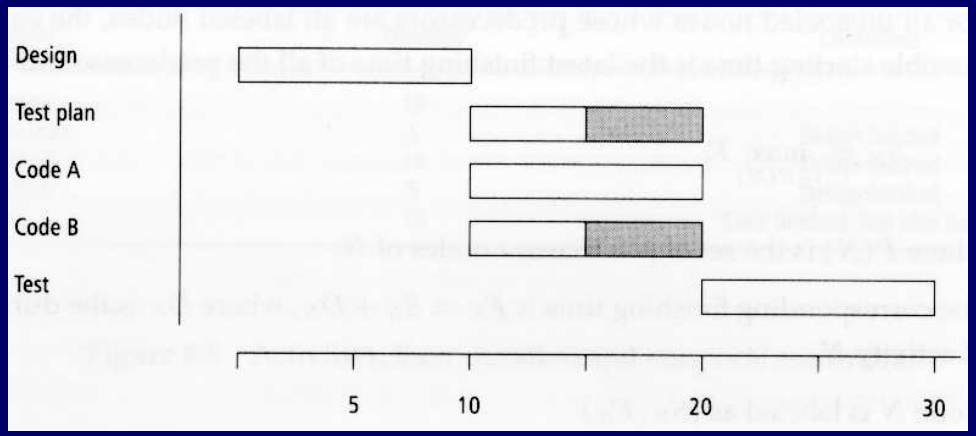


- **PERT chart** (Program Evaluation and Review Technique)
- A network (graph) where the nodes represent tasks and arrows describe precedence relations
  - Used successfully in management of Polaris missile project in 50's
  - Shows task duration (on the task node)
  - Shows precedence relations
  - Generally does *not* show task decomposition

Example of a PERT chart

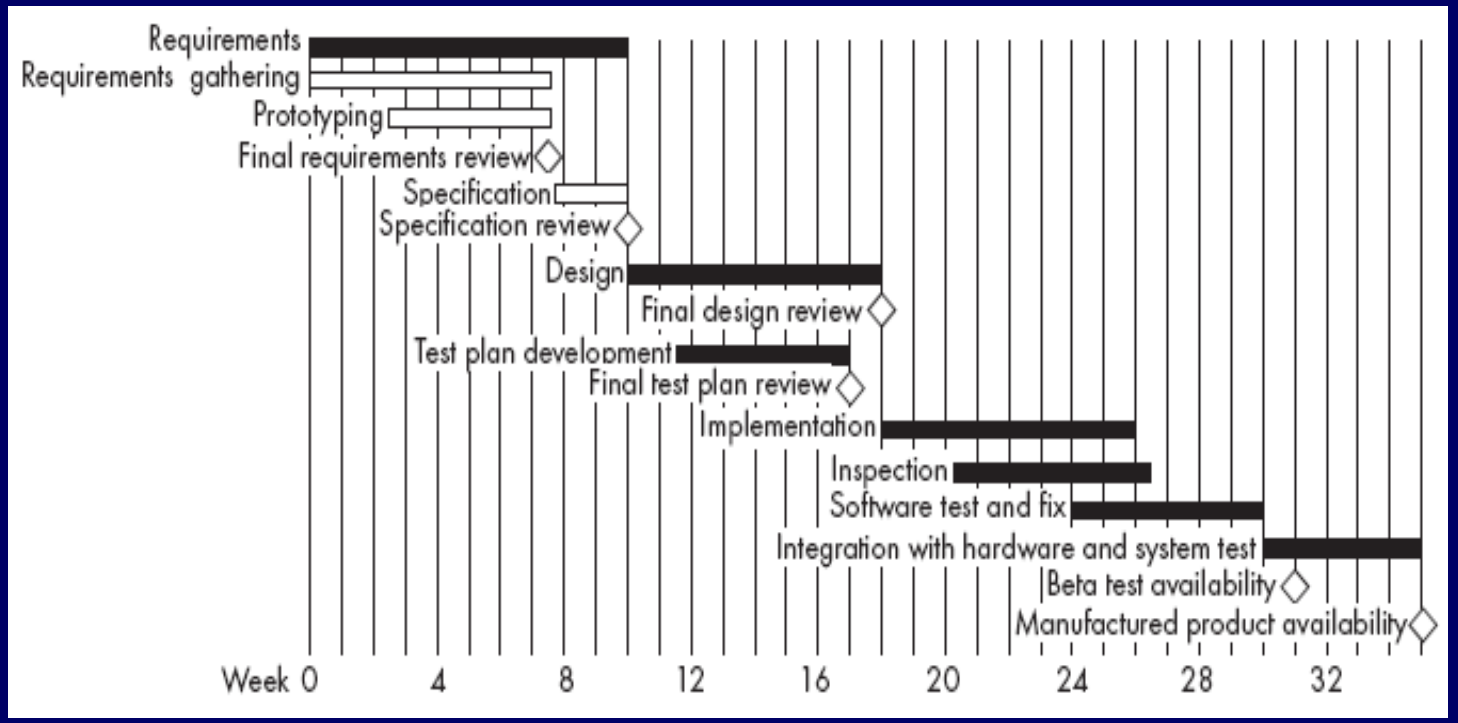


Gantt charts



- A graphical visualization of a *schedule*, where the time span for each activity is depicted by the length of a segment drawn on an adjacent calendar
  - Generally does not show task decomposition
  - Does not show duration, only the time span over which the task is scheduled
  - Does not show precedence relations
  - Can show activity of multiple developers in parallel
  - Makes it easy to monitor a project's progress and expenditures

Example of a Gantt chart



- One axis shows time.
- The other axis shows the activities that will be performed.
- The black bars are the top-level tasks.
- The white bars are subtasks
- The diamonds are *milestones*

Contents of a Project Plan

A. Purpose
B. Background information
C. Processes to be used
D. Subsystems and planned releases
E. Risks and challenges
F. Tasks
G. Cost estimates
H. Team
I. Schedule and milestones

Difficulties and Risks in Project Management

- **Accurately estimating costs is a constant challenge**
  - *Follow the cost estimation guidelines.*
- **It is very difficult to measure progress and meet deadlines**
  - *Improve your cost estimation skills so as to account for the kinds of problems that may occur.*
  - *Develop a closer relationship with other members of the team.*
  - *Be realistic in initial requirements gathering, and follow an iterative approach.*
  - *Use earned value charts to monitor progress.*

Difficulties and Risks in Project Management

- **It is difficult to deal with lack of human resources or technology needed to successfully run a project**
  - *When determining the requirements and the project plan, take into consideration the resources available.*
  - *If you cannot find skilled people or suitable technology then you must limit the scope of your project.*

Difficulties and Risks in Project Management

- **Communicating effectively in a large project is hard**
  - *Take courses in communication, both written and oral.*
  - *Learn how to run effective meetings.*
  - *Review what information everybody should have, and make sure they have it.*
  - *Make sure that project information is readily available.*
  - *Use 'groupware' technology to help people exchange the information they need to know*

Difficulties and Risks in Project Management

- **It is hard to obtain agreement and commitment from others**
  - *Take courses in negotiating skills and leadership.*
  - *Ensure that everybody understands*
    - *The position of everybody else.*
    - *The costs and benefits of each alternative.*
    - *The rationale behind any compromises.*
  - *Ensure that everybody's proposed responsibility is clearly expressed.*
  - *Listen to everybody's opinion, but take assertive action, when needed, to ensure progress occurs.*

Revisiting Software Estimation

Software Cost Estimation Methods

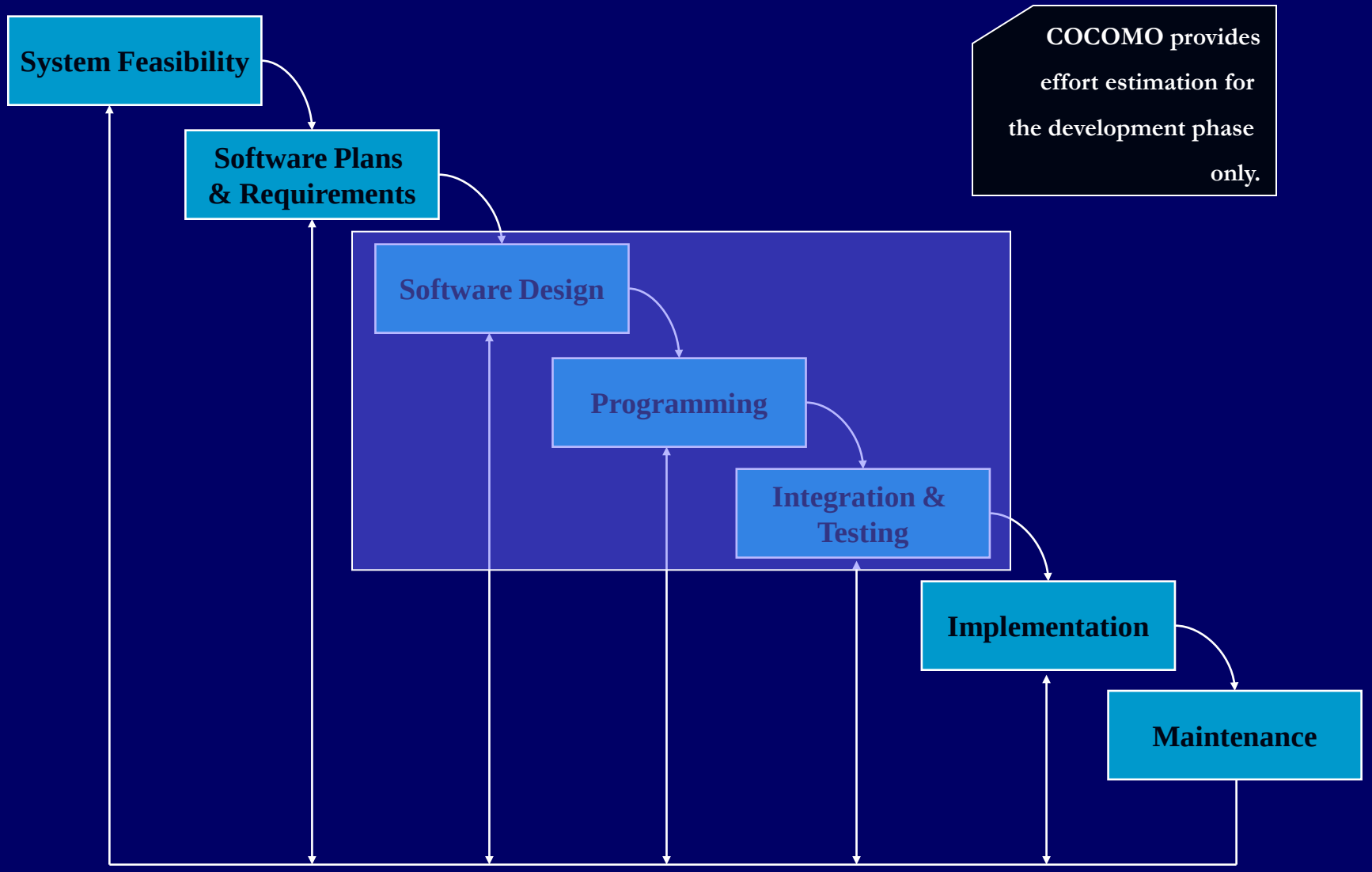
- Cost estimation: prediction of both the person-effort and elapsed time of a project

- Methods:
  - Algorithmic
  - Expert judgement
  - Estimation by analogy
  - …
- Best approach is a combination of methods
  - compare and iterate estimates, reconcile differences

- COCOMO is the most widely used, thoroughly documented and calibrated cost model

The COCOMO model

- Constructive Cost Model
- An empirical model based on project experience
- Well-documented, 'independent' model which is not tied to a specific software vendor
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2
- COCOMO 2 takes into account different approaches to software development, reuse, etc.


System Feasibility
Software Plans & Requirements
Software Design
Programming
Integration & Testing
Implementation
Maintenance
COCOMO provides effort estimation for the development phase only.

Definitions and Assumptions

- DSI (Delivered Source Instructions) *is the primary cost driver. The term 'delivered' excludes non-delivered support software.*

- PM (Person-Month) – *A COCOMO person-month consists of 19 days(152hrs) of working time (including the average time off due to sick leave, holidays, and vacation).*

COCOMO Versions

- Basic, *used mostly for rough early estimates.*

- Intermediate, *is the most commonly used version.*
  - Incorporates 15 cost drivers to account for software project cost variations that are not directly related to project size (i.e., personnel capability, use of modern tools, hardware constraints etc)

- Detailed, *Not used very often*
  - accounts for the influence of the different factors on individual project phases..

COCOMO Development Types

- Organic Type
  - Relatively small software teams develop familiar types of software in an in-house environment.
  - Most of the personnel connected with the project have previous experience working with related or similar systems in the organizations.
- Embedded Type
  - Project may require new technology, unfamiliar algorithms or an innovative new method of solving a problem.
- Semi-detached Type
  - An intermediate stage between organic and embedded types (i.e., either a mixture of organic and embedded type or an intermediate level of the project characteristics).

Cost Estimation using Basic COCOMO

Type	Effort	Schedule
Organic	PM = 2.4 $(KDSI)^{1.05}$	TD = 2.5 $(PM)^{0.38}$
Semi-Detached	PM = 3.0 $(KDSI)^{1.12}$	TD = 2.5 $(PM)^{0.35}$
Embedded	PM = 3.6 $(KDSI)^{1.20}$	TD = 2.5 $(PM)^{0.32}$

- PM – person-month
- KDSI – delivered source instructions, in thousands
- TD – number of months estimated for software development

EXAMPLE:

Assumption:

Organic Project with estimated size of 128,000 lines of code.

Effort : PM = 2.4 $(128)^{1.05}$	= 392 person-months
Productivity: 128,000 DSI/392 PM	= 327 DSI/PM
Schedule: TD = 2.5 $(392)^{0.38}$	= 24 months
Avg. Staffing: 392 PM/ 24 months	= 16 PM/TD

Cost Estimation using Intermediate COCOMO

- It is a compatible extension to basic COCOMO.
- It provides great accuracy and level of detail which makes it more suitable for cost estimations in the more detailed stages of software product definition.
- The intermediate COCOMO exponents for the three software development types are the same as Basic COCOMO, but the coefficients are different. The Intermediate development schedule is determined using the Basic COCOMO schedule equations.

Type	Effort	Schedule
Organic	$PM = 3.2 (KDSI)^{1.05}$	$TD = 2.5 (PM)^{0.38}$
Semi-Detached	$PM = 3.0 (KDSI)^{1.12}$	$TD = 2.5 (PM)^{0.35}$
Embedded	$PM = 2.8 (KDSI)^{1.20}$	$TD = 2.5 (PM)^{0.32}$

Cost Drivers

- Intermediate COCOMO incorporates 15 predictor variables called Cost Drivers
- These are grouped into 4 categories

- Software Attributes

- **Required Software Reliability (RELY)**
- **Data Base Size (DATA)**
- **Software Complexity (CPLX)**

- Personnel Attributes

- **Analyst Capability (ACAP)**
- **Applications Experience (AEXP)**
- **Programmer Capability (PCAP)**
- **Virtual Machine Experience (VEXP)**
- **Programming Language Experience (LEXP)**

- Computer Attributes

- **Execution Time Constraint (TIME)**
- **Main Storage Constraint (STOR)**
- **Virtual Machine Volatility (VIRT)**
- **Computer Turnaround Time (TURN)**

- Project Attributes

- ***Use of Modern Programming Practices (MODP)***
- ***Use of Software Tools (TOOL)***
- ***Schedule Constraint (SCED)***