# Why do projects fail so often?

- Unrealistic or unarticulated project goals
- Inaccurate estimates of needed resources
- Badly defined system requirements
- Poor reporting of the project's status
- Poor communication among customers, developers, and users
- Unmanaged risks
- Use of immature technology
- Inability to handle the project's complexity
- Sloppy development practices
- Poor project management
- Stakeholder politics
- Commercial pressures

Tip:

Must have a deadline [time-boxes]
for micro-tasks, not just macro-tasks

# Software Projects

# Core Skills of Software Engineer

- There are 5 things a software engineer should be good at:

- Programming
- Design
- Process
- Communication
- Team work
- +Tools

# Project Management

- Two over-arching inter-dependent aspects of software projects
  - Process
  - Project Management

# Project Management

- Main responsibilities of a project manager are
  - Project planning
  - Project monitoring and control

(Major activities: Software Estimation, Scheduling and Tracking)

# Software Estimation

- “Predictions are hard, especially about the future”, Yogi Berra
- Two Types of estimates:
  - Lucky or Lousy

# Estimations

- Created, used or refined during
  - Strategic planning
  - Feasibility study
  - Proposals
  - Vendor and sub-contractor evaluation
  - Project planning (iteratively)
- Basic process
  1) Estimate the **size** of the product
  2) Estimate the **effort** (man-hours/man-months)
  3) Estimate the **schedule**
  - NOTE: Not all of these steps are always explicitly performed

# Estimations

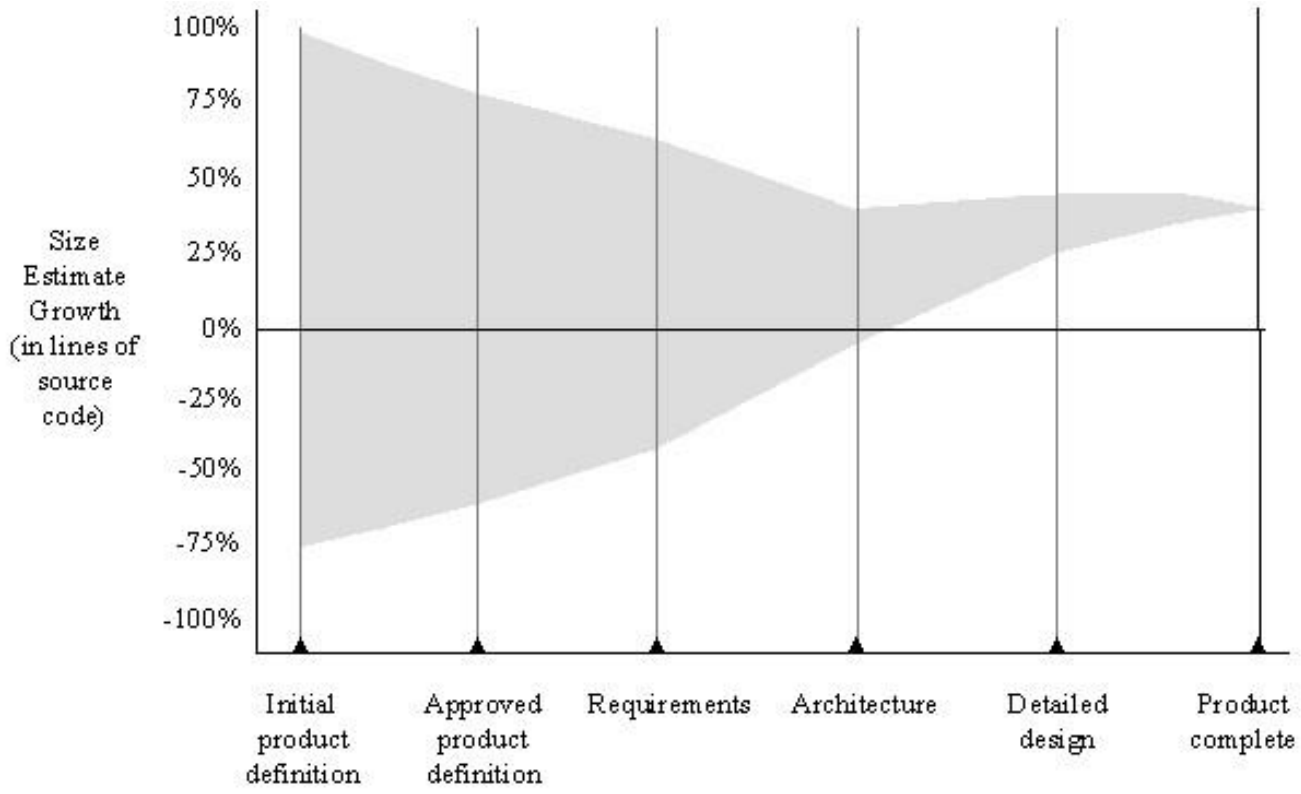
- Remember, an “exact estimate” is an oxymoron
- Estimate how long will it take you to get to dormitory or dining hall from class today-
  - On what basis did you do that?
  - Experience right? (History matters…)
  - Likely as an “average” probability
  - For most software projects there is no such ‘average’

# Estimation

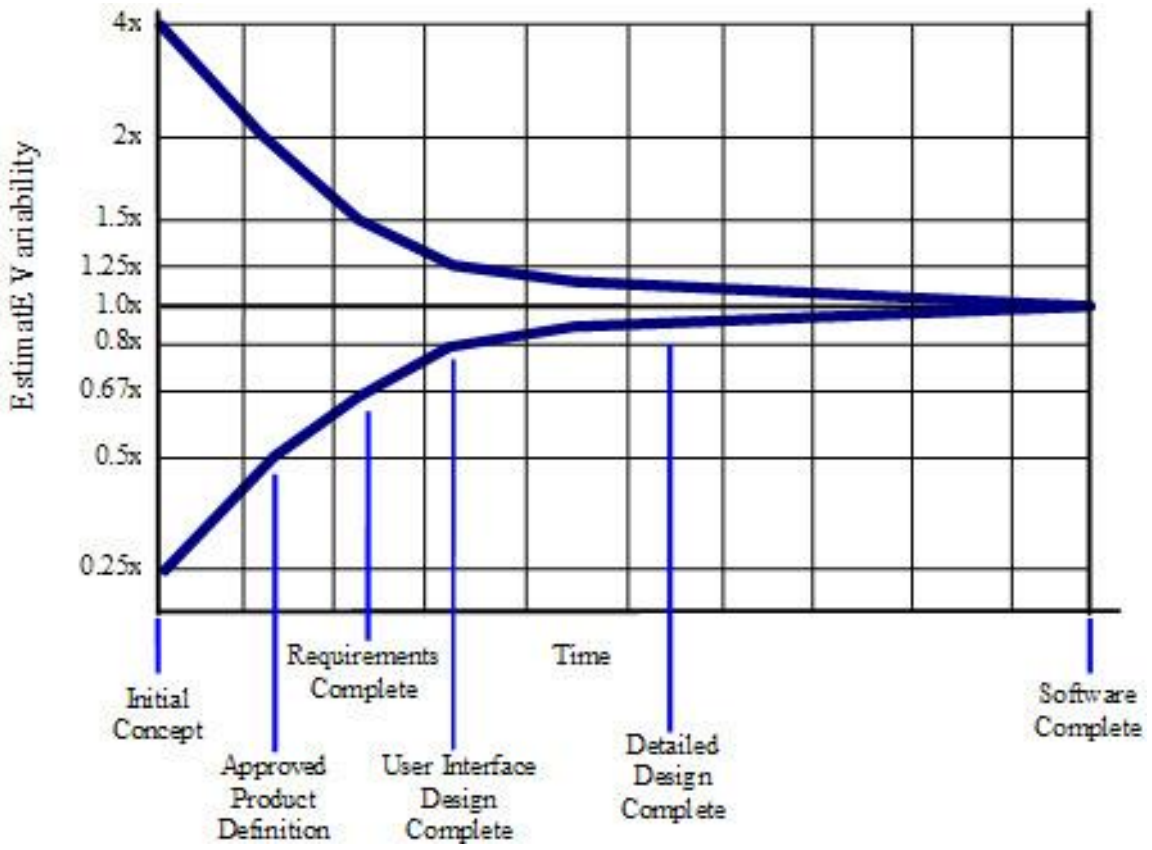
- Target vs. Committed Dates
  - Target: Proposed by business or marketing
  - Do not commit to this too soon!
  - Committed dates: Team agrees to this

- Let’s looks an assignment analogy
  - Do instructors take the various factors into consideration before assigning a deadline?

# Cone of Uncertainty

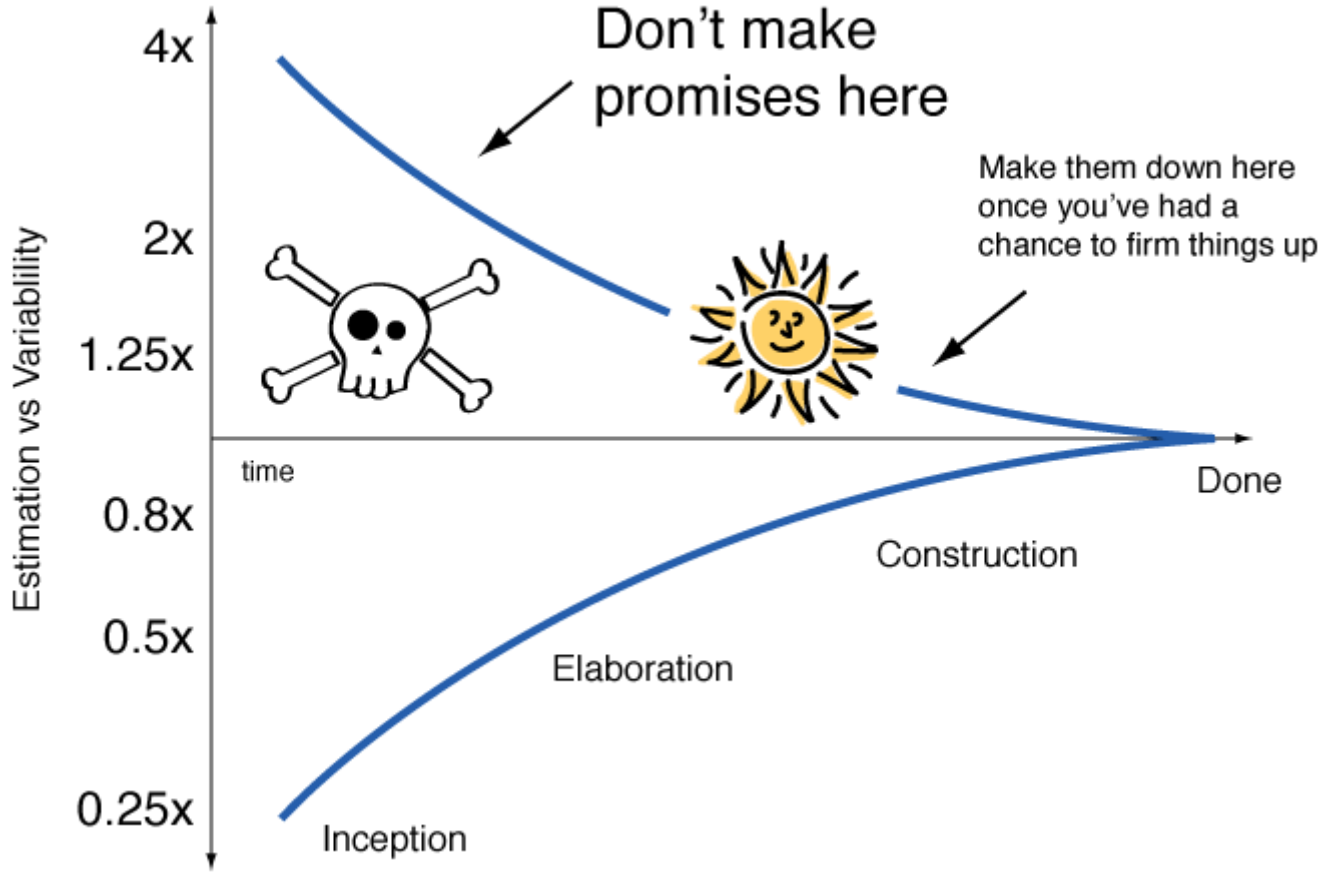


# Cone of Uncertainty






From Rapid Development by Steve McConnell


Don’t make promises here
Make them down here once you’ve had a chance to firm things up
Estimation vs Variability
4x
2x
1.25x
0.8x
0.5x
0.25x
time
Done
Construction
Elaboration
Inception

# Estimation Methodologies

- Top-down
- Bottom-up
- Analogy
- Expert Judgment
- Priced to Win (request for quote – RFQ)
- Parametric or Algorithmic Method
  - Using formulas and equations

# Top-down Estimation

- Based on overall characteristics of project
  - Some of the others can be “types” of top-down (Analogy, Expert Judgment, and Algorithmic methods)
- Advantages
  - Easy to calculate
  - Effective early on (like initial cost estimates)
- Disadvantages
  - Some models are questionable or may not fit
  - Less accurate because it doesn’t look at details

# Bottom-up Estimation

- Create WBS – Work Breakdown Structure, identify individual tasks to be done.
- Add from the bottom-up
- Advantages
  - Works well if activities well understood
- Disadvantages
  - Specific activities not always known
  - More time consuming

# Expert Judgment

- Use somebody who has recent experience on a similar project
- You get a “guesstimate”
- Accuracy depends on their ‘real’ expertise
- Comparable application(s) must be accurately chosen

# Estimation by Analogy

- Use past project
  - Must be sufficiently similar (technology, type, organization)
  - Find comparable attributes (ex: # of inputs/outputs)
- Advantages
  - Based on actual historical data
- Disadvantages
  - Difficulty ‘matching’ project types
  - Prior data may have been mis-measured
  - How to measure differences – no two exactly same

# Algorithmic Measures

- Lines of Code (LOC)
- Function points
- Feature points or object points
- LOC and function points most common
  - (of the algorithmic approaches)
- Majority of projects use none of the above

# Lines of Code (LOC) based Estimates

- LOC Advantages
  - Commonly understood metric
  - Permits specific comparison
  - Actuals easily measured
- LOC Disadvantages
  - Difficult to estimate early in cycle
  - Counts vary by language
  - Many costs not considered (ex: requirements)
  - Programmers may be rewarded based on this
    - Can use: # defects/# LOC
  - Code generators produce excess code

# LOC Estimate Issues

- How do you know how many in advance?
- What about different languages?
- What about programmer style?
- Stat: avg. programmer productivity: 3,000 LOC/yr
- Most algorithmic approaches are more effective after requirements (or have to be after)

# Wideband Delphi

- Group consensus approach
- Present experts with a problem and response form
- Conduct group discussion, collect anonymous opinions, then feedback
- Conduct another discussion & iterate until consensus
- Advantages
  - Easy, inexpensive, utilizes expertise of several people
  - Does not require historical data
- Disadvantages
  - Difficult to repeat
  - May fail to reach consensus, reach wrong one, or all may have same bias

# Function Points

- Software size measured by number & complexity of functions it performs
- More methodical than LOC counts
- House analogy
  - House's Square Feet ~= Software LOC
  - # Bedrooms & Baths ~= Function points
  - Former is size only, latter is size & function
- Six basic steps

# Function Point Process

- 1. Count # of business functions per category
  - Categories: outputs, inputs, DB inquiries, files or data structures, and interfaces
- 2. Establish Complexity Factor for each and apply
  - Low, Medium, High
  - Set a weighting multiplier for each (0 →15)
  - This results in the “unadjusted function-point total”
- 3. Compute an “influence multiplier” and apply
  - It ranges from 0.65 to 1.35; is based on 14 factors
- 4. Results in “function point total”
  - This can be used in comparative estimates

# Function point multipliers

| | Function Points | | |
|---|---|---|---|
| Program Characteristic | Low Complexity | Medium Complexity | High Complexity |
| Number of Inputs | x 3 | x 4 | x 6 |
| Number of Outputs | x 4 | x 5 | x 7 |
| Inquiries | x 3 | x 4 | x 6 |
| Logical internal files | x 7 | x 10 | x 15 |
| External interface files | x 5 | x 7 | x 10 |



From Rapid Development by Steve McConnell

# Counting the Number of Function Points

| | Function Points | | |
|---|---|---|---|
| Program Characteristic | Low Complexity | Medium Complexity | High Complexity |
| Number of Inputs | 5 x 3 = 15 | 2 x 4 = 8 | 3 x 6 =18 |
| Number of Outputs | 6 x 4 = 24 | 6 x 5 = 30 | 0 x 7 = 0 |
| Inquiries | 0 x 3 = 0 | 2 x 4 = 8 | 4 x 6 = 24 |
| Logical internal files | 5 x 7 = 35 | 2 x 10 =20 | 3 x 15 = 45 |
| External interface files | 8 x 5 = 40 | 0 x 7 = 0 | 2 x 10 = 20 |
| Unadjusted function-point total | | | 287 |
| Influence multiplier | | | 1.20 |
| Adjusted function-point total | | | 344 |



From Rapid Development by Steve McConnell

# Code Reuse & Estimation

- Does not come for free
- Code types: New, Modified, Reused
- If code is more than 50% modified, it’s “new”
- Reuse factors have wide range
  - Reused code takes 30% effort of new
  - Modified is 60% of new
- Integration effort with reused code almost as expensive as with new code

# Effort Estimation

- Now that you know the “size”, determine the “effort” needed to build it
- Various models: empirical, mathematical, subjective
- Expressed in units of duration
  - Man-months (‘staff-months’) or Man-hours

# COCOMO

- Barry Boehm – 1980's
- **CO**nstructive **CO**st **MO**del
- Input – LOC, Output - Person Months
- Allows for the type of application, size, and "Cost Drivers"
- Cost drivers using High/Med/Low & include
  - Motivation, Ability of team, Application experience, etc.
- Biggest weakness?
  - Requires input of a product size estimate in LOC

# Estimation Issues

- Quality estimations needed early but information is limited
- Precise estimation data available at end but not needed
  - Or is it? What about the next project?
- Best estimates are based on past experience
- Politics of estimation:
  - You may anticipate a “cut” by upper management
- For many software projects there is little or none
  - Technologies change
  - Historical data unavailable
  - Wide variance in project experiences/types
  - Subjective nature of software estimation

# Over and Under Estimation

- Over estimation issues
  - The project will not be funded
    - Conservative estimates guaranteeing 100% success may mean funding probability of zero.
  - Danger of feature and scope creep
  - Be aware of “double-padding”: team member + manager

- Under estimation issues
  - Quality issues (short changing key phases like testing)
  - Inability to meet deadlines
  - Morale and other team motivation issues

# Estimation Guidelines

- Estimate iteratively!
  - Process of gradual refinement
  - Make your best estimates at each planning stage
  - Refine estimates and adjust plans iteratively
  - Plans and decisions can be refined in response
  - Balance: too many revisions vs. too few

# Know Your Deadlines

- Are they ‘Real Deadlines’?
  - Tied to an external event
  - Have to be met for project to be a success
  - Ex: end of financial year, contractual deadline, Y2K
- Or ‘Artificial Deadlines’?
  - Set by arbitrary authority
  - May have some flexibility (if pushed)

# Estimation “Presentation”

- How you present the estimation can have **huge** impact
- Techniques
  - Plus-or-minus qualifiers
    - 6 months +/-1 month
  - Ranges
    - 6-8 months
  - Risk Quantification
    - +/- with added information
    - +1 month of new tools not working as expected
    - -2 weeks for less delay in hiring new developers
  - Cases
    - Best / Planned / Current / Worst cases
  - Coarse Dates
    - Q3 02
  - Confidence Factors
    - April 1 – 10% probability, July 1 – 50%, etc.

# Other Estimation Factors

- Account for resource experience or skill
  - Up to a point
  - Often needed more on the “low” end, such as for a new or junior person
- Allow for “non-project” time & common tasks
  - Meetings, phone calls, web surfing, sick days
- There are commercial ‘estimation tools’ available
  - They typically require configuration based on past data

# Summary

- Software estimation involves estimation of
  - Size
  - Effort
  - Resources

- There are various estimation techniques. For example
  - Wideband Delphi
  - CoCoMo

# Planning, Estimating, Scheduling

- What's the difference?

- Estimating: Determining the size & duration of activities.
- Plan: Identify activities. No specific start and end dates.
- Schedule: Adds specific start and end dates, relationships, and resources.

# How To Schedule

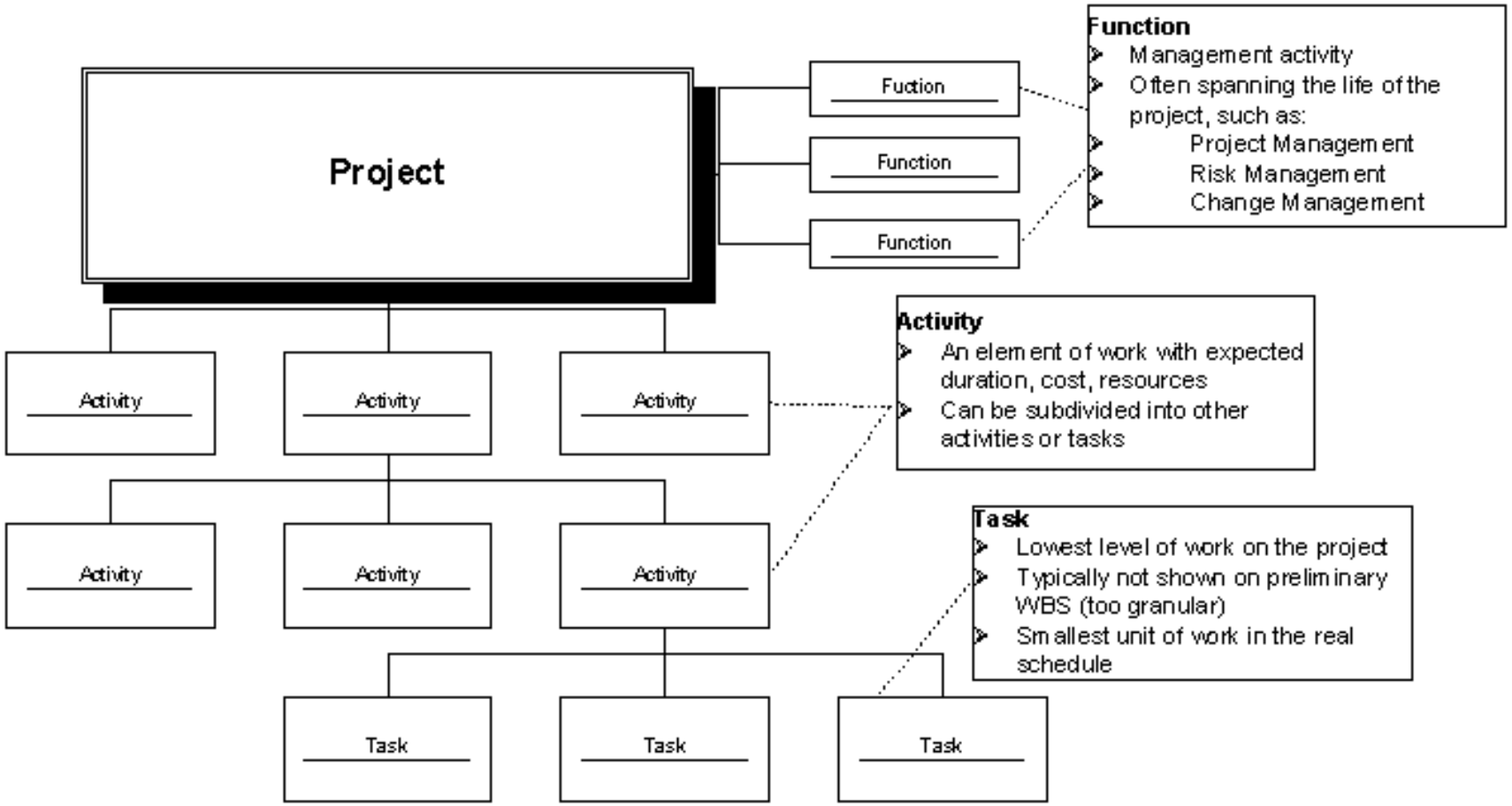
- 1. Identify “what” needs to be done
  - Work Breakdown Structure (WBS)
- 2. Identify “how much” (the size)
  - Size estimation techniques
- 3. Identify the dependency between tasks
  - Dependency graph, network diagram
- 4. Estimate total duration of the work to be done
  - The actual schedule

# Partitioning Your Project

- You need to decompose your project into manageable chunks
- ALL projects need this step
- Divide & Conquer
- Two main causes of project failure
  - Forgetting something critical
  - Ballpark estimates become targets
- How does partitioning help this?

# Project Elements

- A Project: functions, activities, tasks

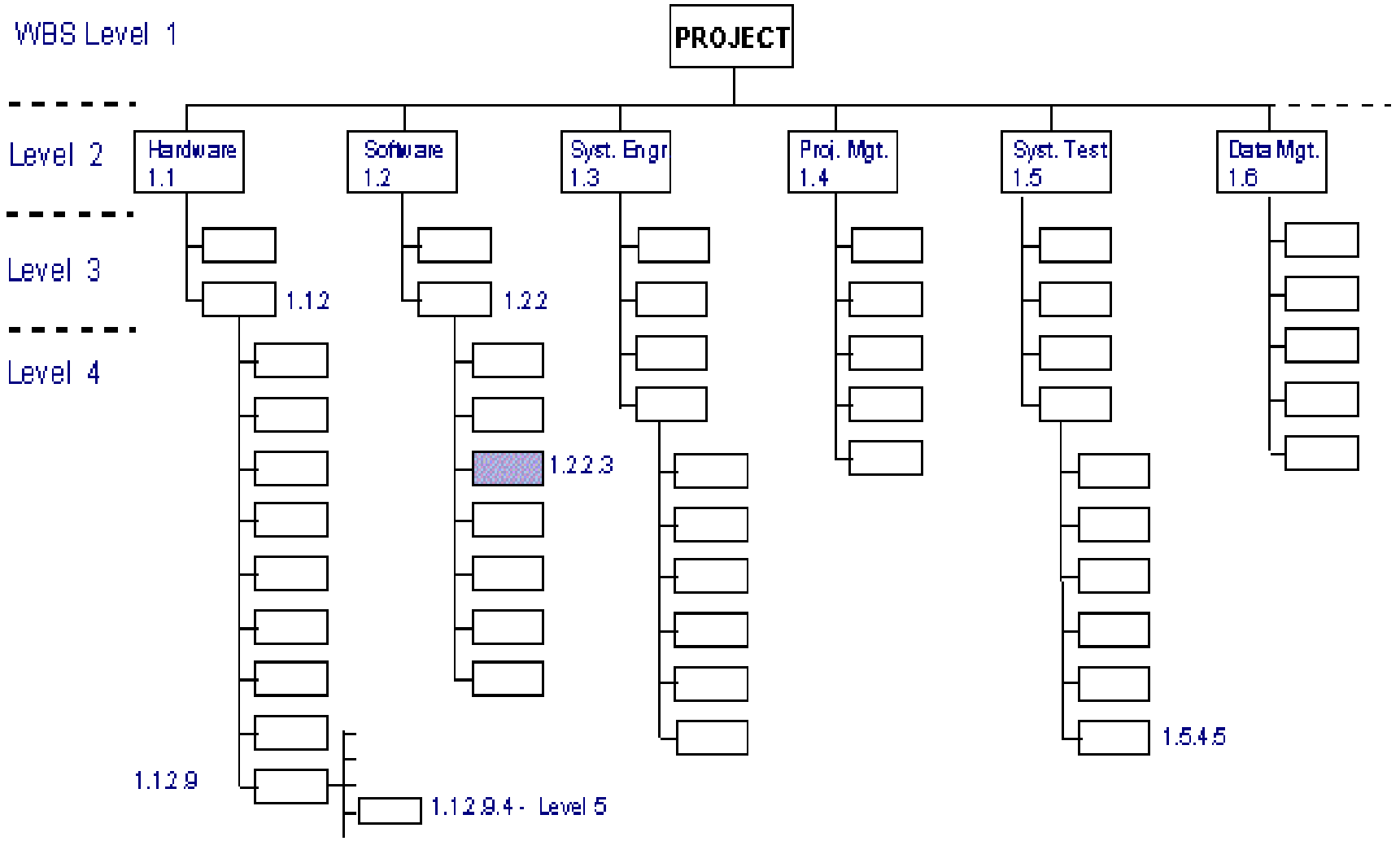


# Work Break Down Structure (WBS)

- ***Work Break Down Structure*** – a check list of the work that must be accomplished to meet the project objectives.
- The WBS lists the major project outputs and those departments or individuals primarily responsible for their completion.

# WBS Outline Example

0.0 Retail Web Site
1.0 Project Management
2.0 Requirements Gathering
3.0 Analysis & Design
4.0 Site Software Development
    4.1 HTML Design and Creation
    4.2 Backend Software
        4.2.1 Database Implementation
        4.2.2 Middleware Development
        4.2.3 Security Subsystems
        4.2.4 Catalog Engine
        4.2.5 Transaction Processing
    4.3 Graphics and Interface
    4.4 Content Creation
5.0 Testing and Production

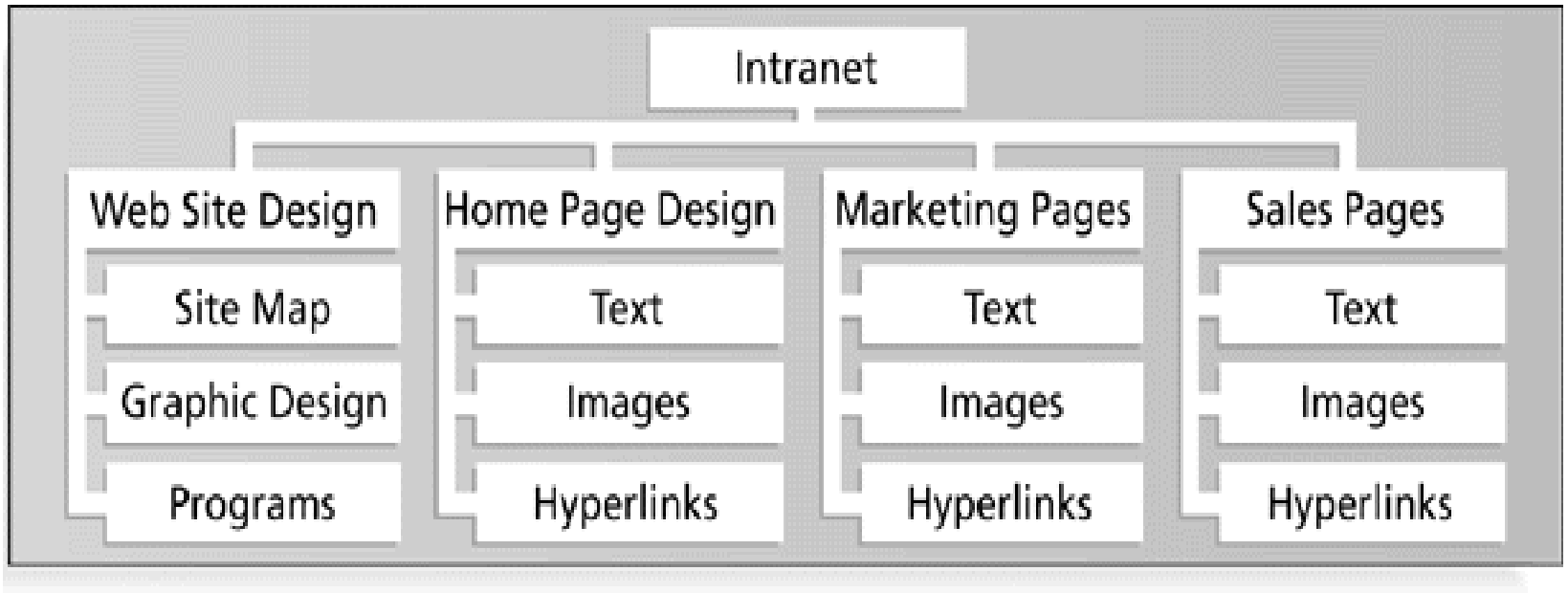



From: http://www.hyperthot.com/pm_wbs.htm

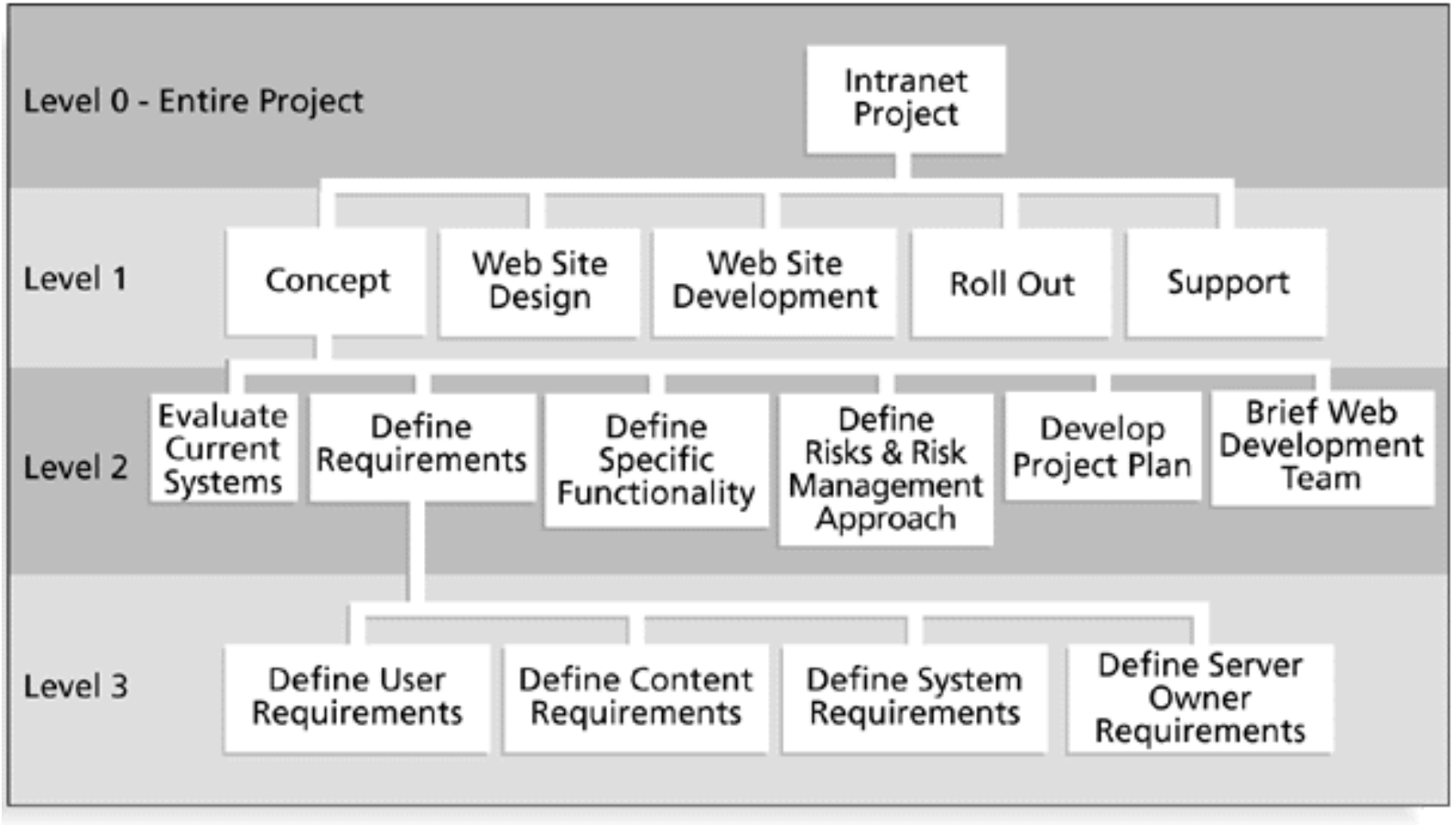
# WBS Types

- Process WBS
  - a.k.a Activity-oriented
  - Ex: Requirements, Analysis, Design, Testing
  - Typically used by PM
- Product WBS
  - a.k.a. Entity-oriented
  - Ex: Financial engine, Interface system, DB
  - Typically used by engineering manager
- Hybrid WBS: both above
  - This is not unusual
  - Ex: Lifecycle phases at high level with component or feature-specifics within phases
  - Rationale: processes produce products

# Product WBS


Intranet
Web Site Design
Site Map
Graphic Design
Programs
Home Page Design
Text
Images
Hyperlinks
Marketing Pages
Text
Images
Hyperlinks
Sales Pages
Text
Images
Hyperlinks

# Process WBS


Level 0 - Entire Project
Intranet Project
Level 1
Concept
Web Site Design
Web Site Development
Roll Out
Support
Level 2
Evaluate Current Systems
Define Requirements
Define Specific Functionality
Define Risks & Risk Management Approach
Develop Project Plan
Brief Web Development Team
Level 3
Define User Requirements
Define Content Requirements
Define System Requirements
Define Server Owner Requirements

# WBS

- List of Activities, not Things
- List of items can come from many sources
  - SOW, Proposal, brainstorming, stakeholders, team
- Describe activities using “bullet language”
  - Meaningful but terse labels
- All WBS paths do not have to go to the same level
- Do not plan more detail than you can manage

# Work Packages (Tasks)

- Generic term for discrete **tasks** with definable end results
- The “one-to-two” rule
  - Often at: 1 or 2 persons for 1 or 2 weeks
- Basis for monitoring and reporting progress
  - Can be tied to budget items (charge numbers)
  - Resources (personnel) assigned
- Ideally shorter rather than longer
  - Not so small as to micro-manage

# WBS Techniques

- Top-Down
- Bottom-Up
- Analogy
- Rolling Wave
  - 1st pass: go 1-3 levels deep
  - Gather more requirements or data
  - Add more detail later
- Post-its on a wall

# WBS Techniques

- Top-down
  - Start at highest level
  - Systematically develop increasing level of detail
  - Best if
    - The problem is well understood
    - Technology and methodology are not new
    - This is similar to an earlier project or problem
  - But is also applied in majority of situations

# WBS Techniques

- Bottom-up
  - Start at lowest level tasks
  - Aggregate into summaries and higher levels
  - Cons
    - Time consuming
    - Needs more requirements complete
  - Pros
    - Detailed

# WBS Techniques

- Analogy
  - Base WBS upon that of a “similar” project
  - Use a template
  - Analogy also can be estimation basis
  - Pros
    - Based on past actual experience
  - Cons
    - Needs comparable project

# WBS Techniques

- Brainstorming
  - Generate all activities you can think of that need to be done
  - Group them into categories
- Both Top-down and Brainstorming can be used on the same WBS
- Remember to get the people who will be doing the work involved (buy-in matters!)

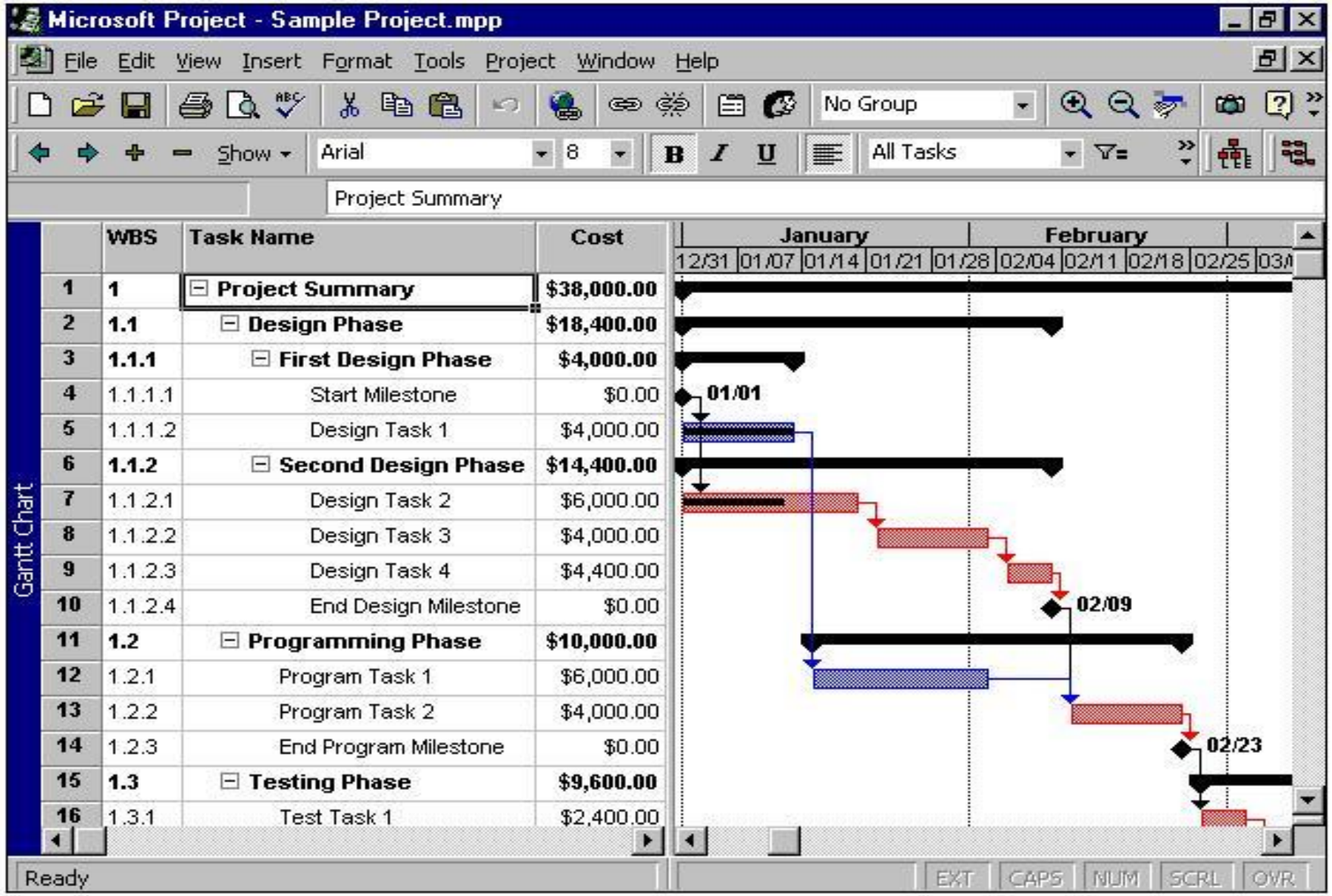
# Sequence the Work Activities

list 3 statergies of presenting software estimation

- Milestone Chart
- Gantt chart
- Network Techniques
  - CPM (Critical Path Method)
  - PERT (Program Evaluation and Review Technique)

# Gantt Chart

- Gantt chart is a means of displaying simple activities or events plotted against time or dollars
- Most commonly used for exhibiting program progress or for defining specific work required to reach an objective
- Gantt charts may include listing of activities, activity duration, scheduled dates, and progress-to-date


Microsoft Project - Sample Project.mpp
File Edit View Insert Format Tools Project Window Help
No Group
Show
Arial
8
All Tasks
Project Summary
Gantt Chart
WBS
Task Name
Cost
1 1 Project Summary $38,000.00
2 1.1 Design Phase $18,400.00
3 1.1.1 First Design Phase $4,000.00
4 1.1.1.1 Start Milestone $0.00
5 1.1.1.2 Design Task 1 $4,000.00
6 1.1.2 Second Design Phase $14,400.00
7 1.1.2.1 Design Task 2 $6,000.00
8 1.1.2.2 Design Task 3 $4,000.00
9 1.1.2.3 Design Task 4 $4,400.00
10 1.1.2.4 End Design Milestone $0.00
11 1.2 Programming Phase $10,000.00
12 1.2.1 Program Task 1 $6,000.00
13 1.2.2 Program Task 2 $4,000.00
14 1.2.3 End Program Milestone $0.00
15 1.3 Testing Phase $9,600.00
16 1.3.1 Test Task 1 $2,400.00
January
February
12/31 01/07 01/14 01/21 01/28 02/04 02/11 02/18 02/25
01/01
02/09
02/23
Ready
EXT CAPS NUM SCRL OVR

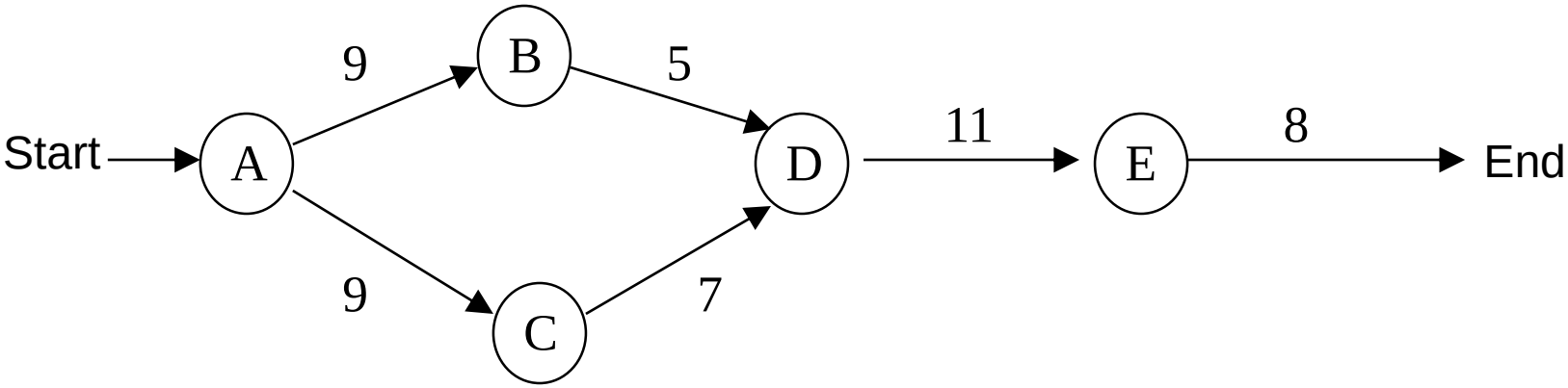
# Gantt Chart

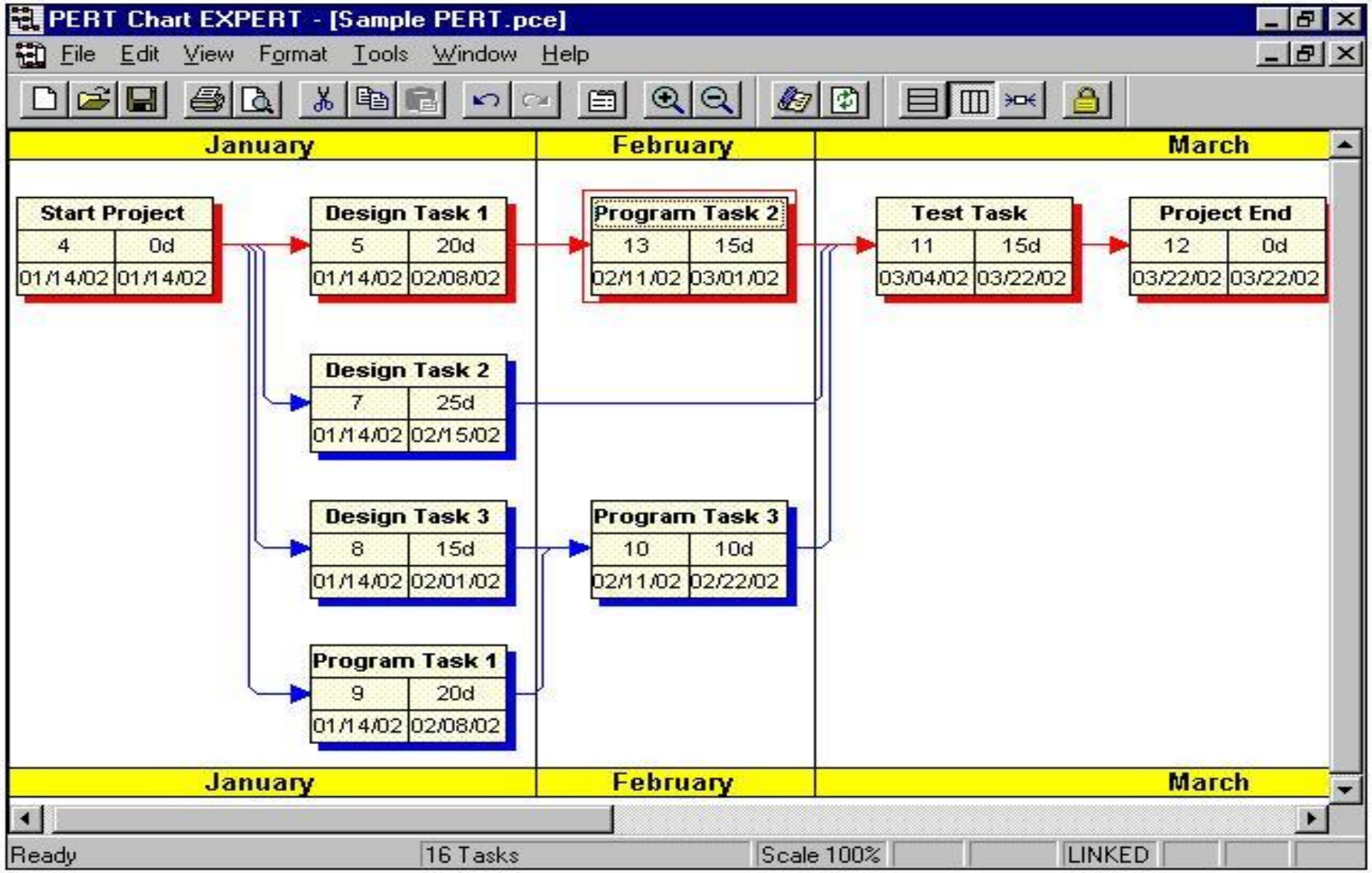
- Advantages:
  - Easy to understand
  - Easy to change
- Disadvantages:
  - only a vague description of the project
  - does not show interdependency of activities
  - cannot show results of an early or late start of an activity

# 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.

| Task | Duration | Dependencies |
|---|---|---|
| A - Architecture & design strategy | 9 | start |
| B - Decide on number of releases | 5 | A |
| C - Develop acceptance test plan | 7 | A |
| D - Develop customer support plan | 11 | B,C |
| E - Final sizing & costing | 8 | D |




PERT Chart EXPERT - [Sample PERT.pce]
File Edit View Format Tools Window Help
January
February
March
Start Project
4
0d
01/14/02
01/14/02
Design Task 1
5
20d
01/14/02
02/08/02
Program Task 2
13
15d
02/11/02
03/01/02
Test Task
11
15d
03/04/02
03/22/02
Project End
12
0d
03/22/02
03/22/02
Design Task 2
7
25d
01/14/02
02/15/02
Design Task 3
8
15d
01/14/02
02/01/02
Program Task 3
10
10d
02/11/02
02/22/02
Program Task 1
9
20d
01/14/02
02/08/02
January
February
March
Ready
16 Tasks
Scale 100%
LINKED

# CPM

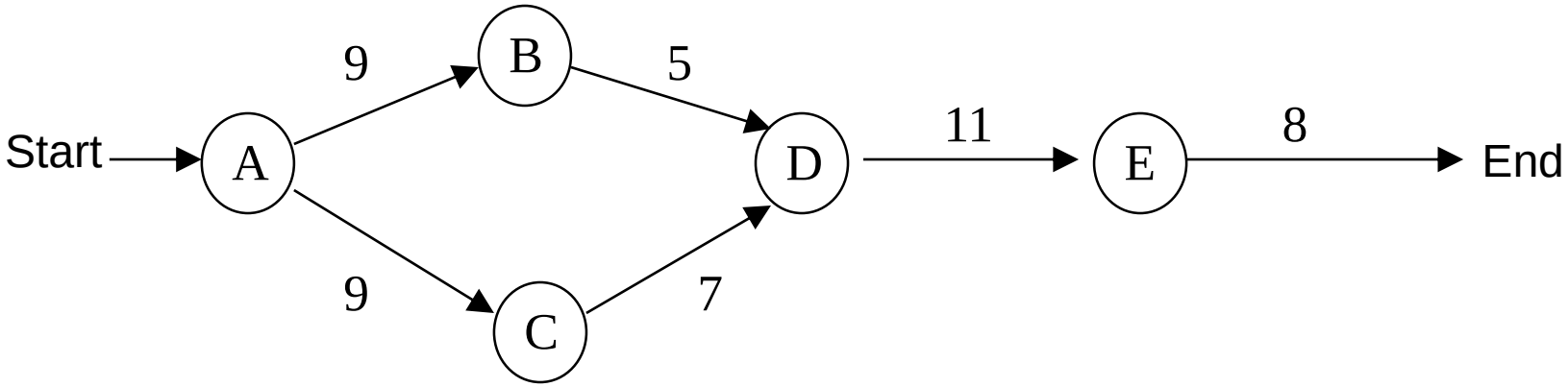
Critical Path Method (CPM) tries to answer the following questions:

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

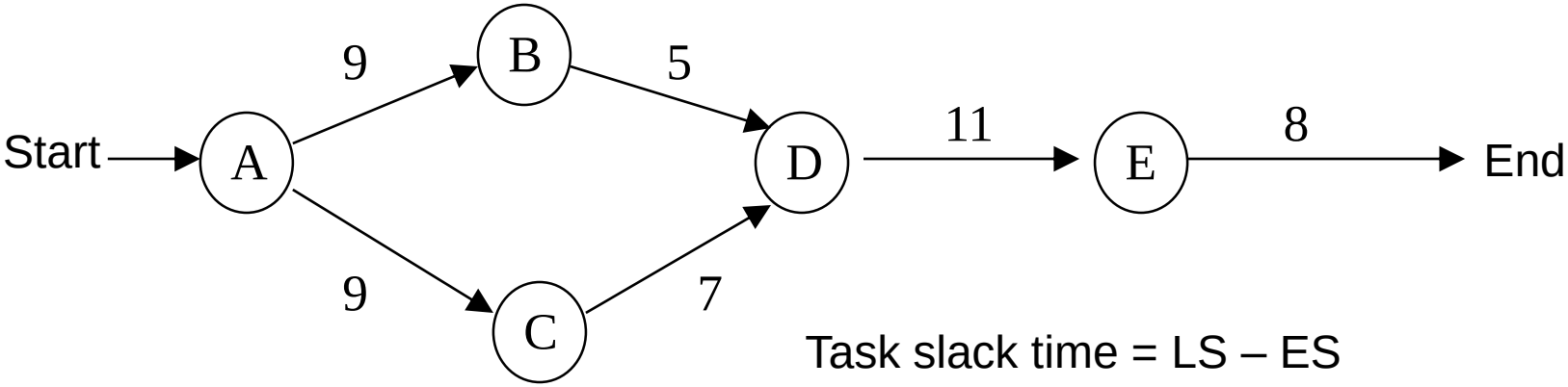



Critical Path = A – C – D – E  (35 time units)

Critical Tasks = A,C,D,E

Non-Critical Path = A-B-D-E

| Task | Duration | Depend | Earliest Start | Earliest Finish | Latest Start | Latest Finish |
|---|---|---|---|---|---|---|
| A | 9 | none | 0 | 9 | 0 | 9 |
| B | 5 | A | 9 | **14** | **11** | 16 |
| C | 7 | A | 9 | 16 | 9 | 16 |
| D | 11 | B,C | 16 | 27 | 16 | 27 |
| E | 8 | D | 27 | 35 | 27 | 35 |




**Slack time** – maximum allowable delay for a non-critical activity.

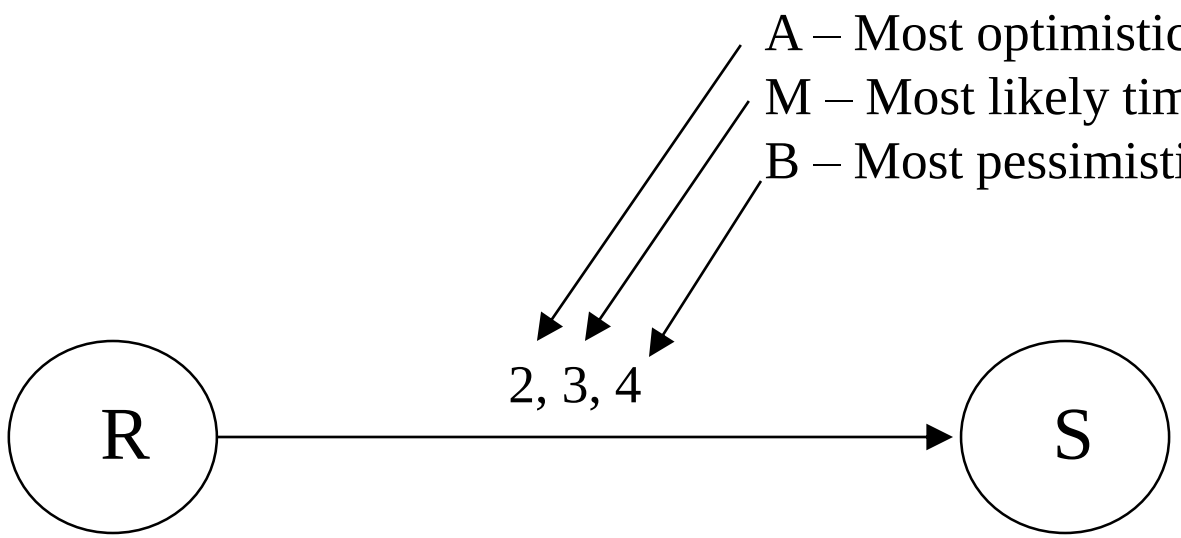
Task slack time = LS – ES

- or -

Task slack time = LF - EF

Task B has 2 time units of ***slack time***

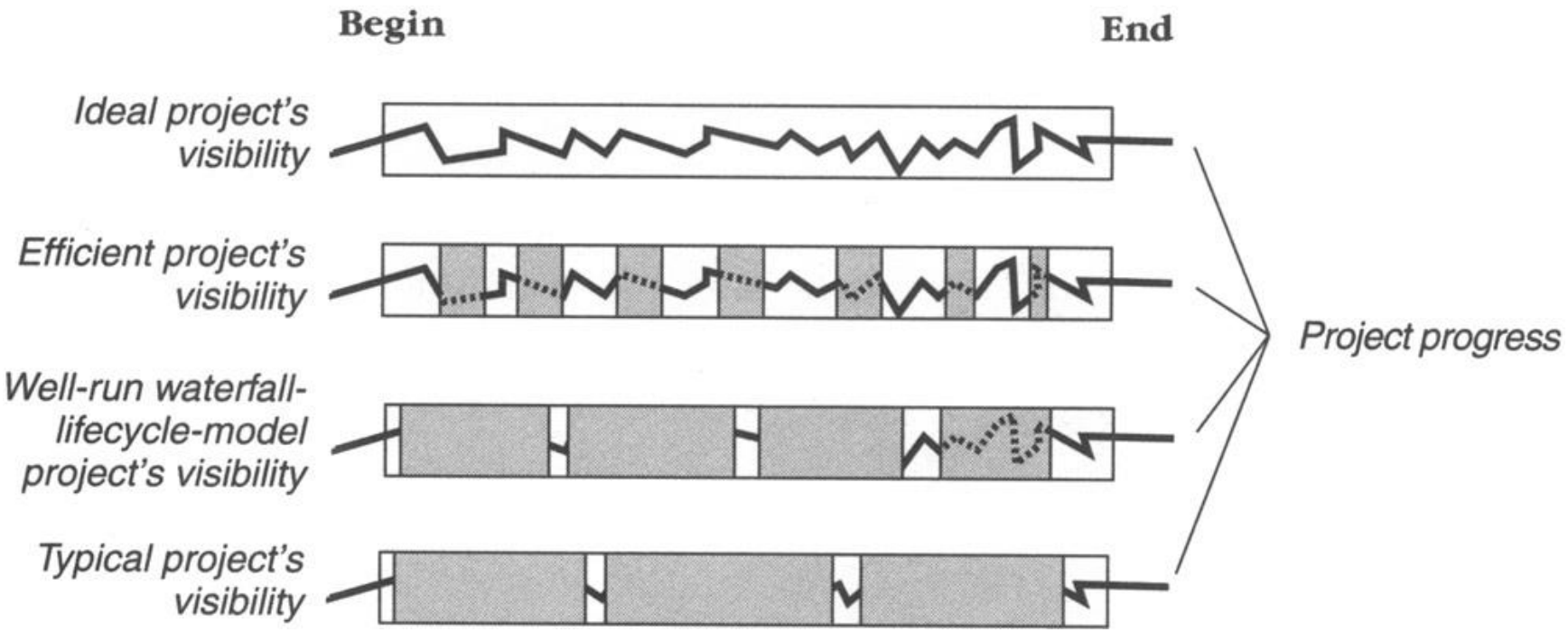
# PERT




Expected Time = (a + 4m + b)/6

Expected Time = 3

# Tracking Visibility



# Percent Complete?

1) Conceptual Design – 200/200
2) Program Specification – 300/300
3) Coding – 150/600
4) Documentation – 10/100
5) User Manual Production – 0/400
6) Testing – 0/500 hours

660 / 2100 * 100 = 31.4% complete

[Wil99]

# Version Control Systems

Some of the pictures in the slides are taken from Collabnet inc

# Why Versioning?

Members of a software development group need to:

- have access to the group source code (file sharing)
- work at the same time on the same files (concurrent editing)
- keep track of different versions of the same file (history)

A Version Control System is a special file server, designed for concurrent editing and to store history information.

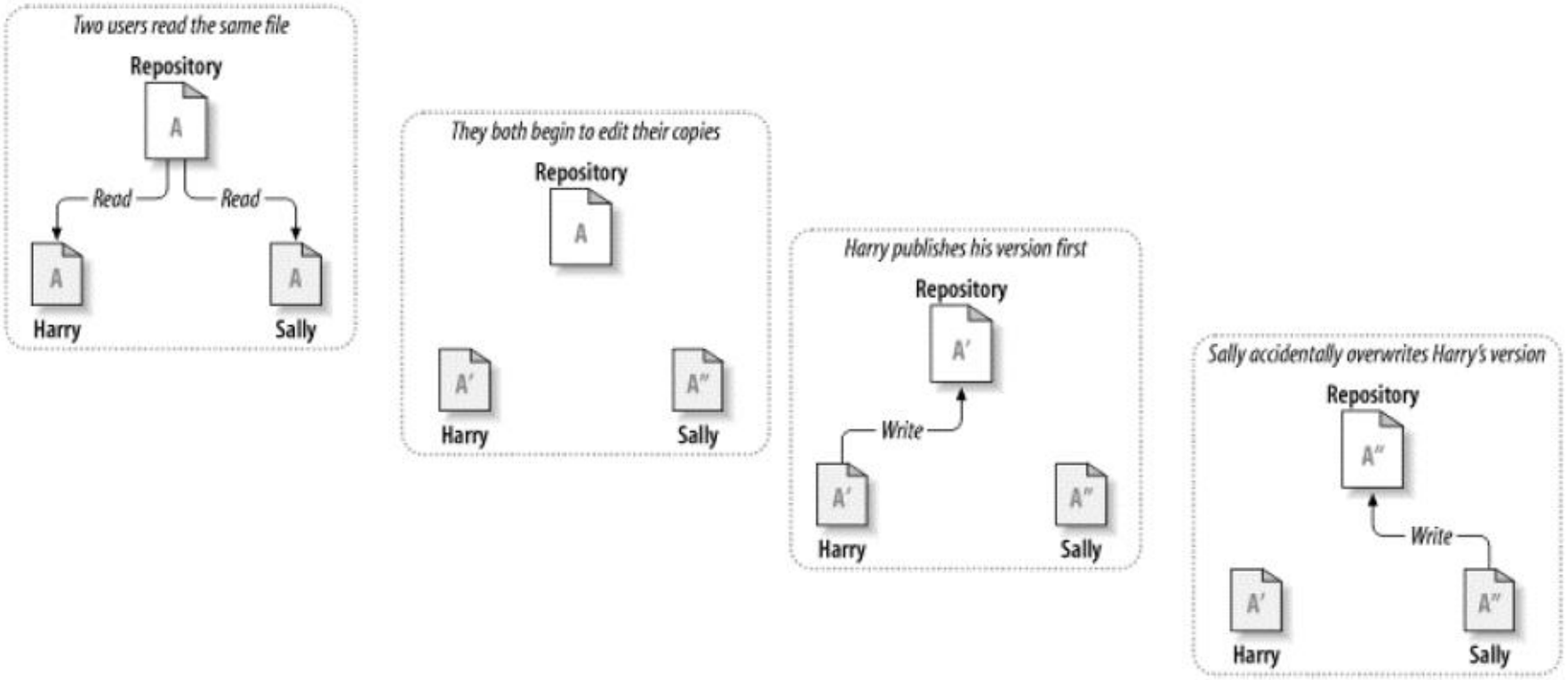
# Available Version Control systems

- CVS
- SVN
- BitKeeper
- Git
- Mercurial
- Others…

# Centralized Vs Distributed Versioning

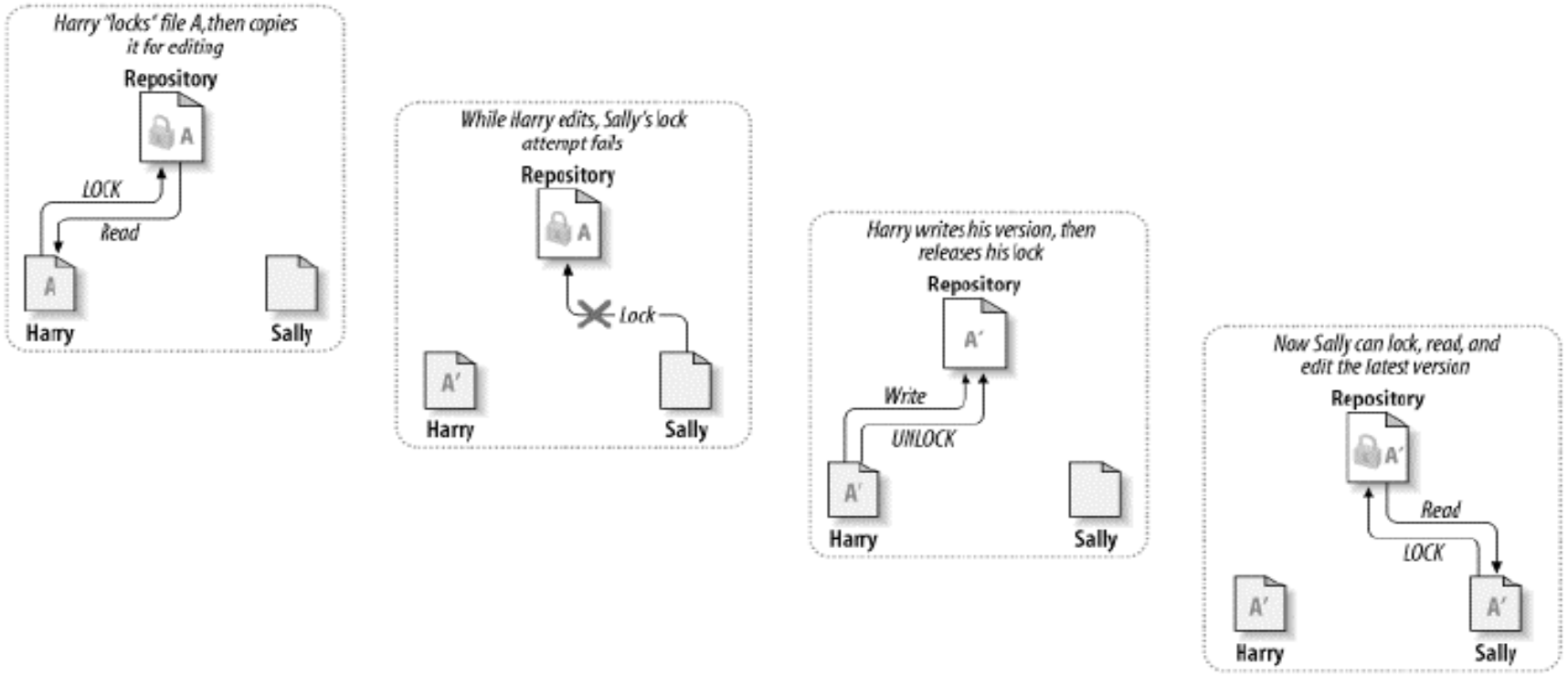
- Client-server  Vs Peer-Peer
- Single “central” repository Vs Many “central” repositories
- Many different ways to merge/branch have been proposed
- There are many other differences…

# Concurrent Editing




A normal file server (ex. NFS) can provide file sharing but maintains only one version of each file

# Lock – Modify - Unlock



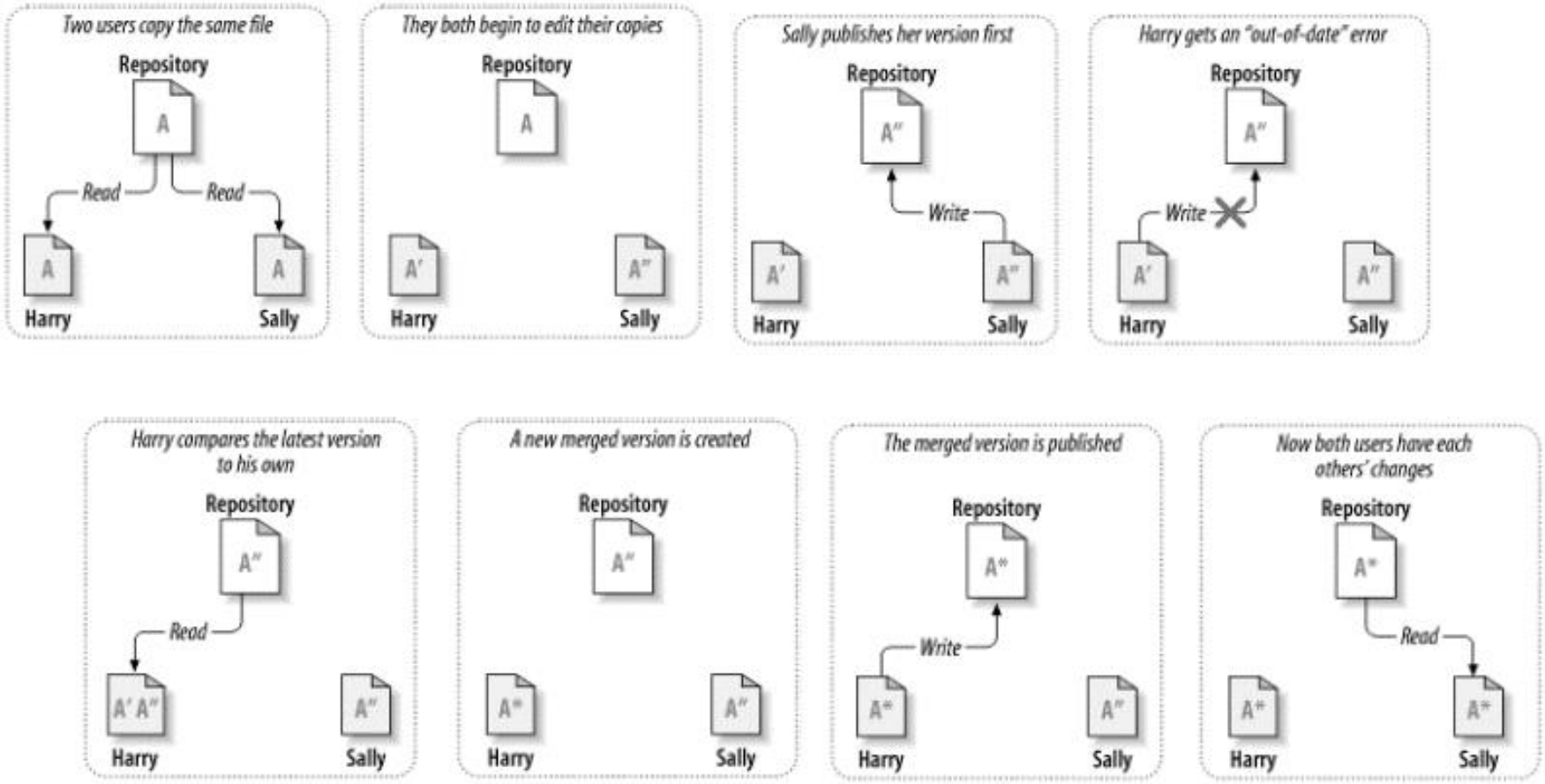

A simple mechanism to support concurrent editing

# Lock – Modify - Unlock

- Disadvantages of this scheme:
  - delays: locking a file prevents concurrent editing
  - administrative overhead: if a user forgets to release the files he has locked, an administrator has to manually remove the lock before another user can edit the files.
  - false sense of security: locking a single file is not sufficient if there are other files depending on it

# Copy – Modify - Merge

- A better mechanism



# Copy – Modify - Merge

- When merging, two types of changes to a file can occur
  - *changes that do not overlap* : in this case merging is trivial - just take the sum of changes
  - *changes that overlap* : in this case there is a conflict and merging can be difficult - users must communicate to decide which changes to propagate to the new version.
- Merging is a manual process by the user

  (No AI available yet to decide which changes to take).
- The amount of time it takes to resolve conflicts is far less than the time lost by a locking system.

# Pull based software development

# Web Frameworks

# Web applications?

- Web application is an application accessible over the internet
- Web framework is a software framework design to support development of web applications (dynamic)
  - Eliminates a lot of overhead
    - Database access
    - Session management
  - Promotes code reuse

# Is creating web applications a problem?

# Golden Hammer?

# Features of a modern web application

- MVC (Model – View – Controller Architecture)
  - Model deals with data and business logic
  - View deals with inputs and result display
  - Controller manager the flow
- ORM (Object-relational-mapping)
- Validation
- Unit-testing facility
- AJAX facility
- Easy web-service facility

# *How about Full Stack?*
# Popular Stacks [from w3 schools]

- LAMP stack: JavaScript - Linux - Apache - MySQL - PHP
- LEMP stack: JavaScript - Linux - Nginx - MySQL - PHP
- MEAN stack: JavaScript - MongoDB - Express - AngularJS - Node.js
- Django stack: JavaScript - Python - Django - MySQL
- Ruby on Rails: JavaScript - Ruby - SQLite - PHP

# What did we do till now?

- Why Software Engineering? (I*ll* vs Well-formed, Precise Vs Imprecise) – Job & Problem Solving
- What is Software Engineering Anyway?, Nature of Software Engineering, Software Engineering Discipline, Challenges
- What is Software Engineering Anyway? Programs Vs Software, Past Contributions
- Version Control and Git Tutorial
- What is Software Engineering Anyway? Current Trends, Future Predictions
- Software Quality Engineering
  - What is it?
  - How to achieve it?
  - How to assess it?
- Software Engineering Principles
- Software Life Cycle Models
- Traditional, Agile, DevOps
- Software Estimation, Project Planning, Scheduling