Managing Software Projects

What is Project Management?

- Project management encompasses all the activities needed to plan and execute a project:
  - Deciding what needs to be done
  - Estimating costs
  - Ensuring there are suitable people to undertake the project
  - Defining responsibilities
  - Scheduling
  - Making arrangements for the work
  - *continued ...*

What is Project Management?

- Directing
- Being a technical leader
- Reviewing and approving decisions made by others
- Building morale and supporting staff
- Monitoring and controlling
- Co-ordinating the work with managers of other projects
- Reporting
- Continually striving to improve the process

	Software Engineering	
	Two Sides of the Coin	
Project Management		**Software Development**
Project Planning		Software Lifecycle
Managing People and Teams		Development Process
Risk Management		Requirements Gathering
Requirements Management		Requirements Analysis
Estimation and Scheduling		Architectural Design
Project Tracking		Detailed Design
Configuration Management		Design Languages
Management Tools		Design Patterns
Quality Assurance		Implementation
Metrics and Data Gathering		Coding and Debugging
Release Management		Reviews and Inspections
Software Process Definition		Testing
Softwware Process Improvement		Software Tools

Reengineering

- Periodically project managers should set aside some time to re-engineer part or all of the system
  - The extent of this work can vary considerably:
    - Cleaning up the code to make it more readable.
    - Completely replacing a layer.
    - *Re-factoring* part of the design.
  - In general, the objective of a re-engineering activity is to increase maintainability.

Cost estimation

- To estimate how much software-engineering time will be required to do some work.
  - *Elapsed time*
    - The difference in time from the start date to the end date of a task or project.
  - *Development effort*
    - The amount of labour used in *person-months* or *person-days*.
    - To convert an estimate of development effort to an amount of money. You multiply it by the *weighted average cost* (burdened cost) of employing a software engineer for a month (or a day).

Example

In your organization, although the average salary is $4,000 /month, the weighted average salary for cost estimation purposes is $11,000/month. You have determined that a particular project will take 7 person-months to complete. How much would you estimate this project will cost financially?

You estimate that the project will cost 7×$11,000 = $77,000.

Principles of effective cost estimation

- **Principle 1: Divide and conquer**.
  - To make a better estimate, you should divide the project up into individual subsystems.
  - Then divide each subsystem further into the activities that will be required to develop it.
  - Next, you make a series of detailed estimates for each individual activity.
  - And sum the results to arrive at the grand total estimate for the project.

Principles of effective cost estimation

- **Principle 2: Include all activities when making estimates.**
  - The time required for *all* development activities must be taken into account.
  - Including:
    - Prototyping
    - Design
    - Inspecting
    - Testing
    - Debugging
    - Writing user documentation
    - Deployment.

Principles of effective cost estimation

- **Principle 3: Base your estimates on past experience combined with knowledge of the current project.**
  - If you are developing a project that has many similarities with a past project:
    - You can expect it to take a similar amount of work.
  - Base your estimates on the *personal judgement* of your experts

    or
  - Use *algorithmic models* developed in the software industry as a whole by analyzing a wide range of projects.
    - They take into account various aspects of a project's size and complexity, and provide formulas to compute anticipated cost.

Algorithmic models

- Allow you to systematically estimate development effort.
  - Based on an estimate of some other factor that you can measure, or that is easier to estimate:
    - The number of use cases
    - The number of distinct requirements
    - The number of classes in the domain model
    - The number of widgets in the prototype user interface
    - An estimate of the number of lines of code

Algorithmic models

- **COCOMO** (COnstructive COst Model) computes effort, in person-months, from an estimate of size, measured in lines of code

$$E = a + bN^c$$

Where E is the effort estimate and N is the estimate or measure being used as the basis for the effort estimate (e.g. number of use cases or lines of code). The values a, b and c are obtained by extensive analysis of past projects

- **Functions Points Analysis**: you count features of the requirements and use these to compute an estimate of the system's size

$$S = W_1F_1 + W_2F_2 + W_3F_3 + \ldots$$

The F_i represent counts of features of the requirements such as number of inputs, number of tables in the database, number of use cases etc. The W_i represent weights calculated using industry-wide experience. The results are summed to produce a system size, S.

Examples

You have been asked to estimate the cost of a project for which you developed a system domain model composed of 24 classes. In a recent similar project, the average development effort (including all testing etc.) was 10 person-days per class. What would be your cost estimate for the current project?

You might first assume from past experience that the system domain model classes represent half of the total number of classes in the final system. The other classes would be user interface and architectural classes. The development effort estimate would therefore be $24 \times 2 \times 10 = 480$ person-days = 16 person-months. Using the burdened cost from Example 11.1, this results in a cost estimate of 16x$11000 = $176,000.

Your records of previous projects show that an average class has 6 attributes, and 14 methods, averaging 6 lines of code each. The average lines of code per class is 90. How many lines of code would you expect to have to develop in the system of Example 11.2?

You would expect to develop $90 \times 48 = 4.3$KLOC.

Principles of effective cost estimation

- **Principle 4: Be sure to account for *differences* when extrapolating from other projects**.
  - Different software developers
  - Different development processes and maturity levels
  - Different types of customers and users
  - Different schedule demands
  - Different technology
  - Different technical complexity of the requirements
  - Different domains
  - Different levels of requirement stability

Principles of effective cost estimation

- **Principle 5: Anticipate the worst case and plan for contingencies.**
  - Develop the most critical use cases first
    - If the project runs into difficulty, then the critical features are more likely to have been completed
  - Make three estimates:
    - Optimistic (O)
      - Imagining a everything going perfectly
    - Likely (L)
      - Allowing for typical things going wrong
    - Pessimistic (P)
      - Accounting for everything that could go wring

Principles of effective cost estimation

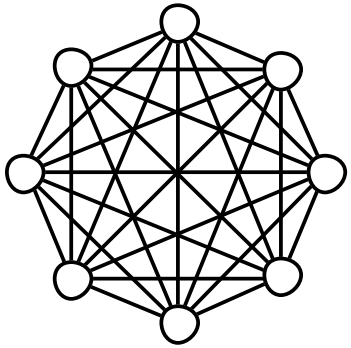
- **Principle 6: Combine multiple independent estimates.**
  - Use several different techniques and compare the results.
  - If there are discrepancies, analyze your calculations to discover what factors causing the differences.
  - Use the Delphi technique.
    - Several individuals initially make cost estimates in private.
    - They then share their estimates to discover the discrepancies.
    - Each individual repeatedly adjusts his or her estimates until a consensus is reached.

Principles of effective cost estimation

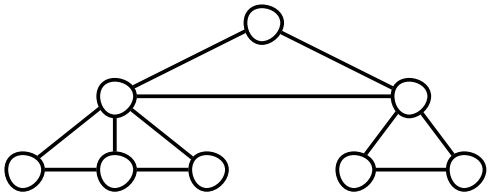
- **Principle 7: Revise and refine estimates as work progresses**
  - As you add detail.
  - As the requirements change.
  - As the risk management process uncovers problems.

Building Software Development Teams

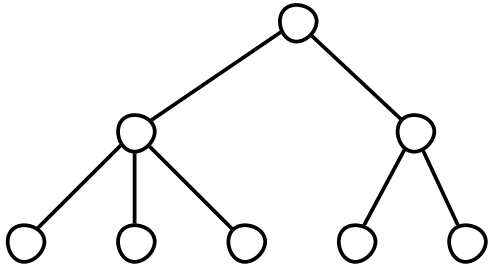
- Software development is a human process.
  - Choosing appropriate people for a team, and assigning roles and responsibilities to the team members, is therefore an important project management skill
  - Software development teams can be organized in many different ways



a) Egoless



b) Chief programmer



c) Strict hierarchy

Software Development 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 Development 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 Development 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.
  - Doubling the size of a team will not halve the development time.
- 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.
- You can not generally add people if you get behind schedule, in the hope of catching up.

Skills needed on a team

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

Project Scheduling and Tracking

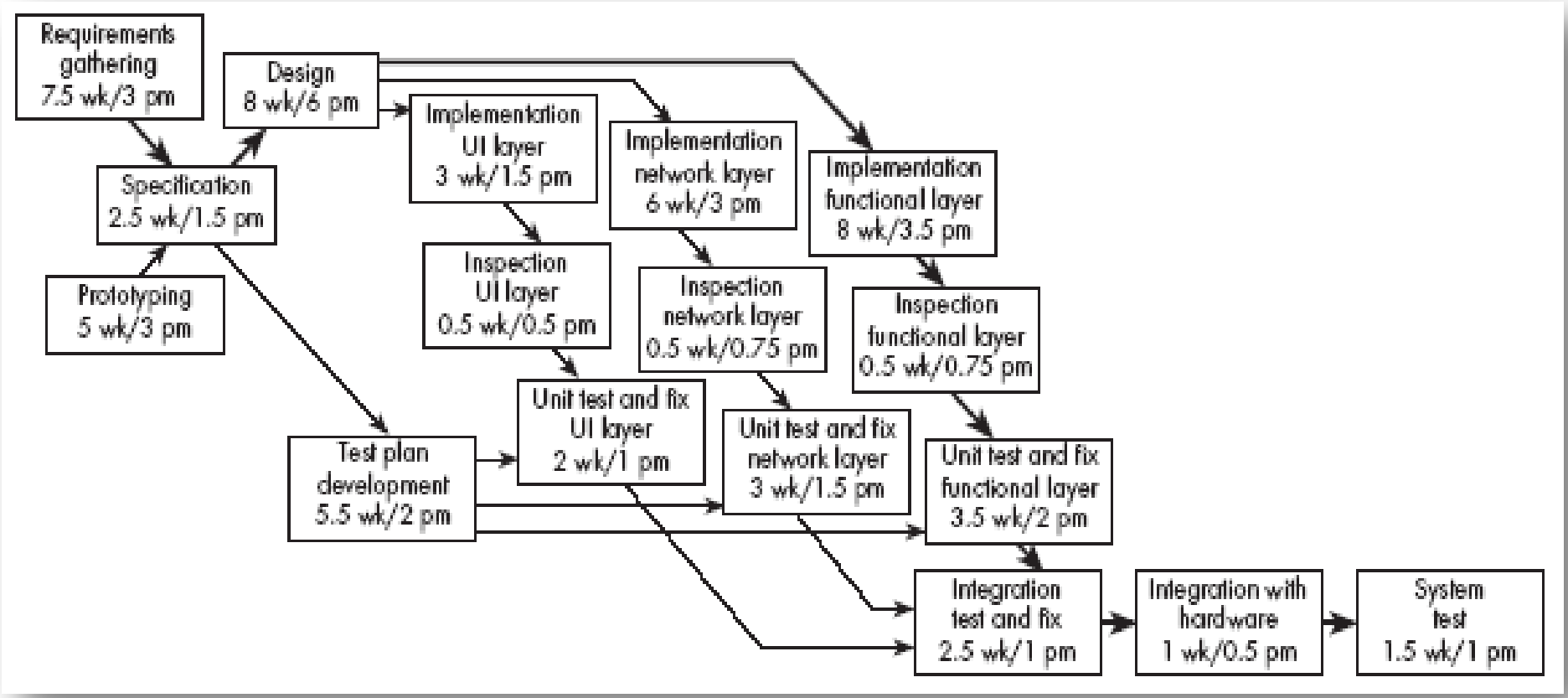
- *Scheduling* is the process of deciding:
  - In what sequence a set of activities will be performed.
  - When they should start and be completed.

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

PERT charts

- A PERT chart shows the sequence in which tasks must be completed.
  - In each node of a PERT chart, you typically show the elapsed time and effort estimates.
  - One of the most important uses of a PERT chart is to determine the critical path.The *critical path* indicates the minimum time in which it is possible to complete the project.

Example of a PERT chart

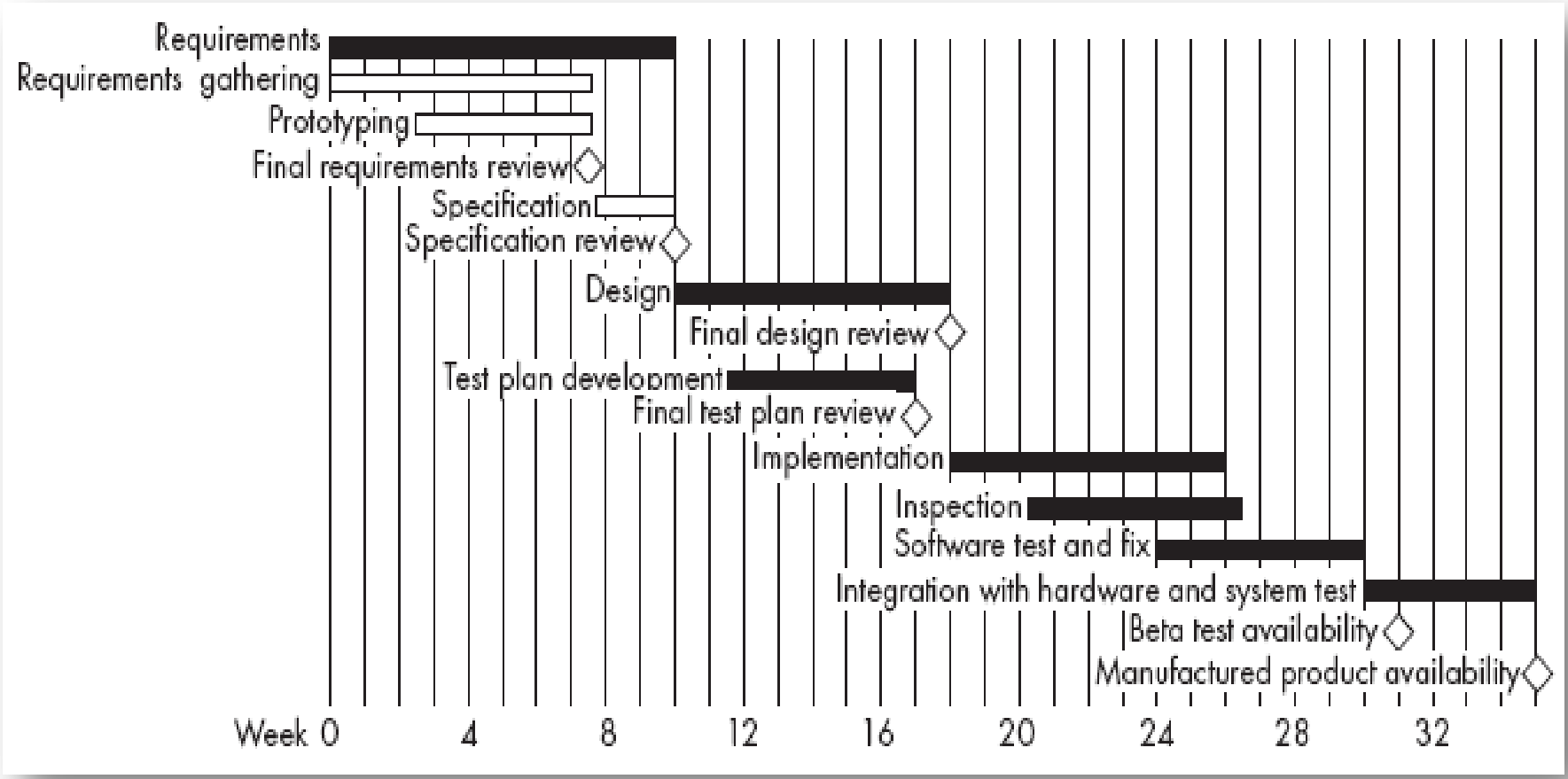


The critical path is shown in bold. You can calculate the minimum elapsed time (35 weeks) by summing the elapsed times on this path.

Gantt charts

- A Gantt chart is used to *graphically* present the start and end dates of each software engineering task
  - 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:*
    - Important deadline dates, at which specific events may occur

Example of a Gantt chart



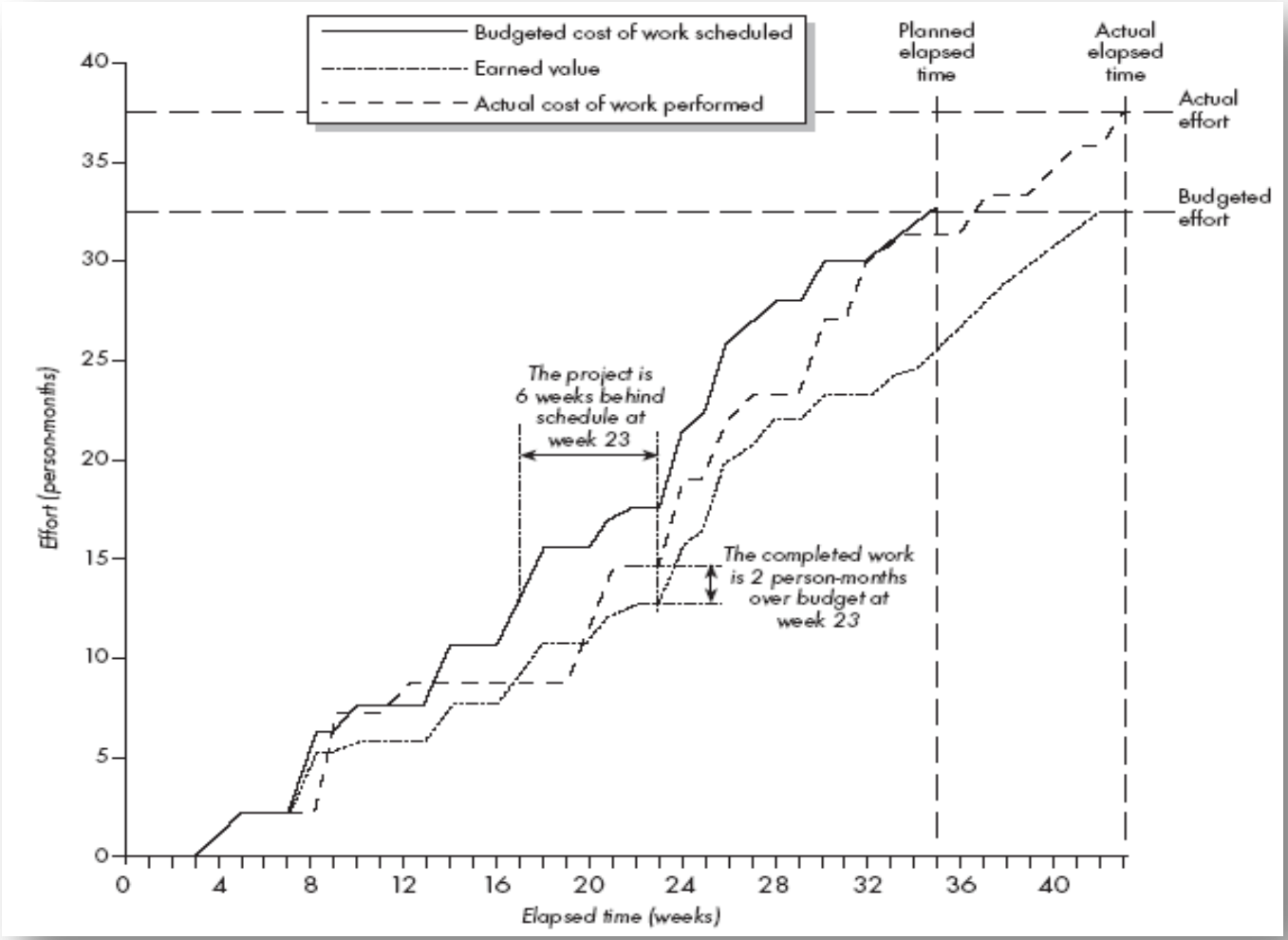
Earned value

- *Earned value* is the amount of work completed, measured according to the *budgeted* effort that the work was supposed to consume.
- It is also called the *budgeted cost of work performed*.
- As each task is completed, the number of person-months originally planned for that task is added to the earned value of the project.

Earned value charts

- An earned value chart has three curves:
  - The budgeted cost of the work scheduled.
  - The earned value.
  - The actual cost of the work performed so far.

Example of an earned value chart



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 (1)

- **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 (2)

- **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 (3)

- **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 (4)

- **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.