



Object-Oriented Software Engineering

Practical Software Development using UML and Java

Chapter 11: Managing the Software Process

Lecture 13

www.lloseng.com

11.1 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

652

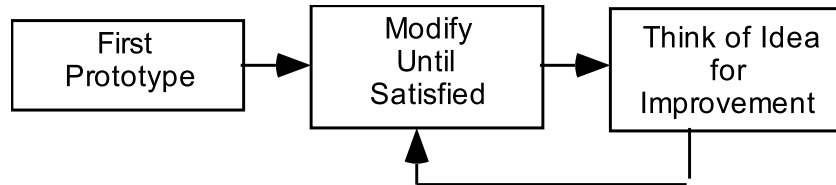
11.2 Software Process Models

Software process models are general approaches for organizing a project into activities.

- Help the project manager and his or her team to decide:
 - What work should be done;
 - In what sequence to perform the work.
- The models should be seen as *aids to thinking*, not rigid prescriptions of the way to do things.
- Each project ends up with its own unique plan.

653

The opportunistic approach



654

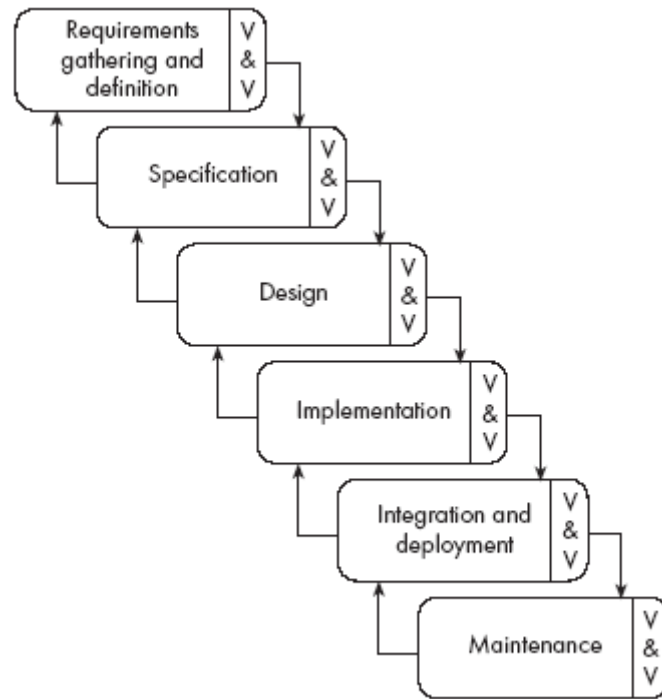
The opportunistic approach

... is what occurs when an organization does not follow good engineering practices.

- It does not acknowledge the importance of working out the requirements and the design before implementing a system.
- The design of software deteriorates faster if it is not well designed.
- Since there are no plans, there is nothing to aim towards.
- There is no explicit recognition of the need for systematic testing and other forms of quality assurance.
- The above problems make the cost of developing and maintaining software very high.

655

The waterfall model



656

The waterfall model

The classic way of looking at S.E. that accounts for the importance of requirements, design and quality assurance

- The model suggests that software engineers should work in a series of stages.
- Before completing each stage, they should perform quality assurance (verification and validation).
- The waterfall model also recognizes, to a limited extent, that you sometimes have to step back to earlier stages.

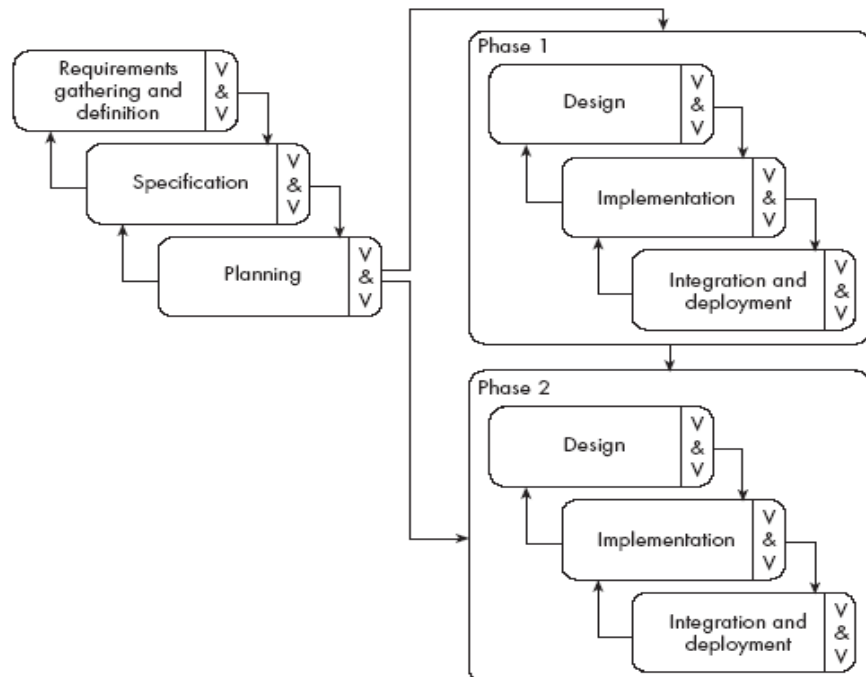
657

Limitations of the waterfall model

- The model implies that you should attempt to complete a given stage before moving on to the next stage
 - Does not account for the fact that requirements constantly change.
 - It also means that customers can not use anything until the entire system is complete.
- The model makes no allowances for prototyping.
- It implies that you can get the requirements right by simply writing them down and reviewing them.
- The model implies that once the product is finished, everything else is maintenance.

658

The phased-release model



659

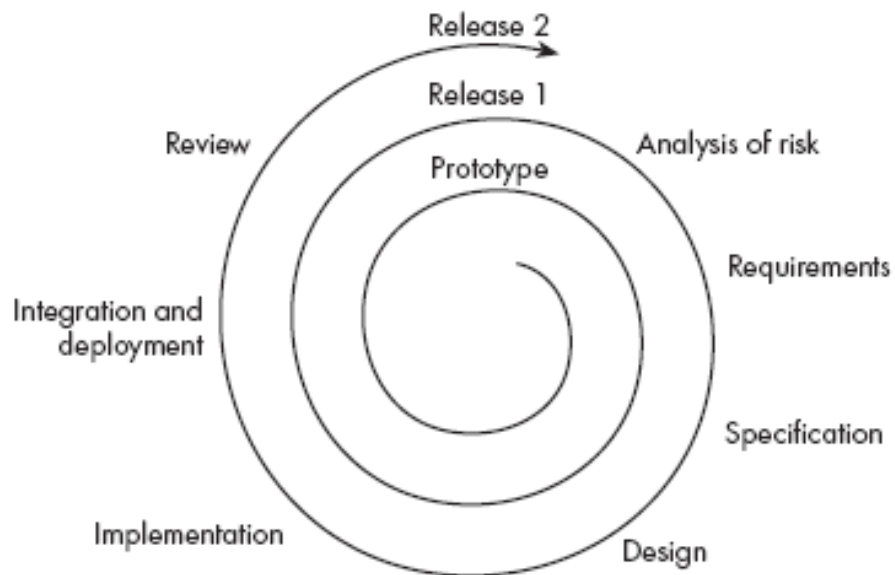
The phased-release model

It introduces the notion of *incremental* development.

- After requirements gathering and planning, the project should be broken into separate subprojects, or *phases*.
- Each phase can be released to customers when ready.
- Parts of the system will be available earlier than when using a strict waterfall approach.
- However, it continues to suggest that all requirements be finalized at the start of development.

660

The spiral model



661

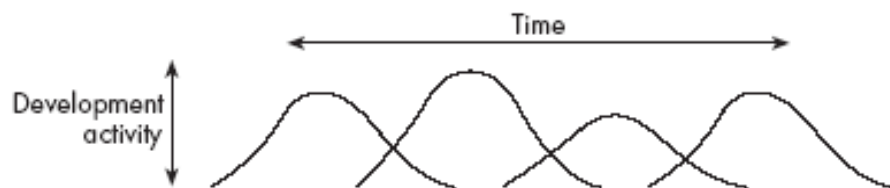
The spiral model

It explicitly embraces prototyping and an *iterative* approach to software development.

- Start by developing a small prototype.
- Followed by a mini-waterfall process, primarily to gather requirements.
- Then, the first prototype is reviewed.
- In subsequent loops, the project team performs further requirements, design, implementation and review.
- The first thing to do before embarking on each new loop is risk analysis.
- Maintenance is simply a type of on-going development.

662

The evolutionary model



663

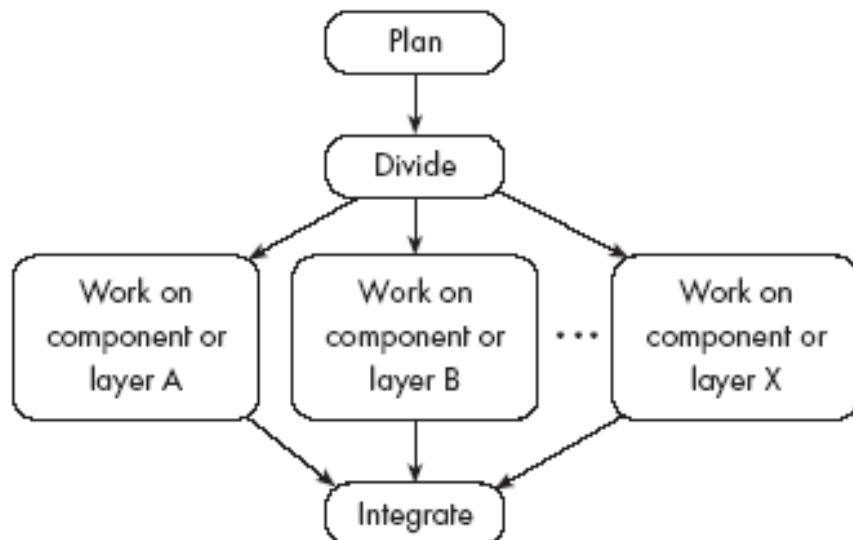
The evolutionary model

It shows software development as a series of hills, each representing a separate loop of the spiral.

- Shows that loops, or releases, tend to overlap each other.
- Makes it clear that development work tends to reach a peak, at around the time of the deadline for completion.
- Shows that each prototype or release can take
 - different amounts of time to deliver;
 - differing amounts of effort.

664

The concurrent engineering model



665

The concurrent engineering model

It explicitly accounts for the divide and conquer principle.

- Each team works on its own component, typically following a spiral or evolutionary approach.
- There has to be some initial planning, and periodic integration.

666

The Rational Unified process

This is the most widely known methodology that embraces UML

- Designed to be adaptable
- Suggests a process framework
- Adapts to the project needs
- Use-case-driven development
- Architecture-centric process

667

Agile approaches

These approaches encourage the development of particularly small iterations

- Well suited for small projects that involve uncertain, changing requirements and other high risk
- The most famous agile technique is eXtreme Programming (XP)
 - All stakeholders work closely together
 - User stories are written instead of requirement document
 - There must be a series of small and frequent releases (1 to 3 weeks)
 - The project variable are: scope, resources and time (and quality)
 - Test cases are written before the software is developed
 - A large amount of refactoring is encouraged
 - Pair programming is recommended

668

Choosing a process model

- From the waterfall model:
 - Incorporate the notion of stages.
- From the phased-release model:
 - Incorporate the notion of doing some initial high-level analysis, and then dividing the project into releases.
- From the spiral model:
 - Incorporate prototyping and risk analysis.
- From the evolutionary model:
 - Incorporate the notion of varying amounts of time and work, with overlapping releases.
- From concurrent engineering:
 - Incorporate the notion of breaking the system down into components and developing them in parallel.

669

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.

670

11.3 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).

671

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.

672

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.

673

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.

674

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

675

Algorithmic models

- A typical algorithmic model uses a formula like the following:
 - COCOMO:

$$E = a + bN^c$$

- Functions Points:

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

676

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

677

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 everything going perfectly
 - Likely (L)
 - Allowing for typical things going wrong
 - Pessimistic (P)
 - Accounting for everything that could go wrong

678

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 are 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.

679

Principles of effective cost estimation

Principle 7: Revise and refine estimates as work progresses

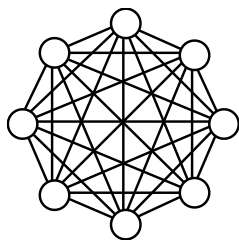
- As you add detail.
- As the requirements change.
- As the risk management process uncovers problems.

680

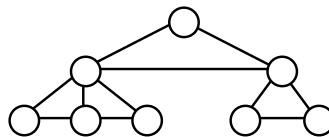
Building Software Engineering Teams

Software engineering is a human process.

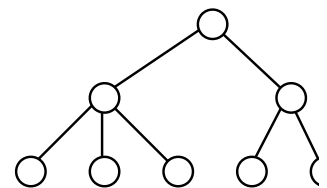
- Choosing appropriate people for a team, and assigning roles and responsibilities to the team members, is therefore an important project management skill
- Software engineering teams can be organized in many different ways



a) Egoless



b) Chief programmer



c) Strict hierarchy

681

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.

682

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.

683

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.

684

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.

685

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

686

11.5 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.

687

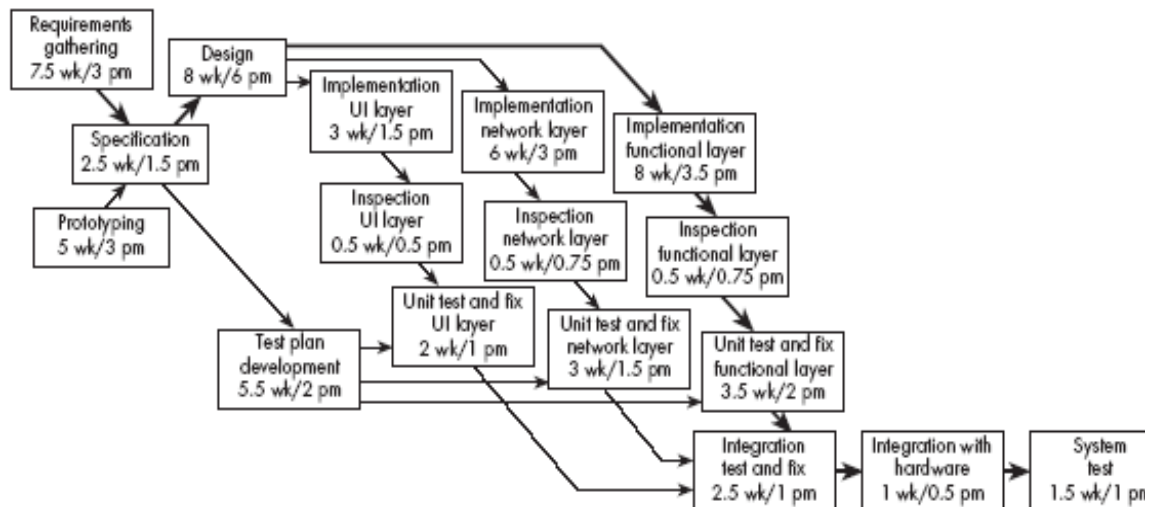
PERT charts

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.
- The *critical path* indicates the minimum time in which it is possible to complete the project.

688

Example of a PERT chart



689

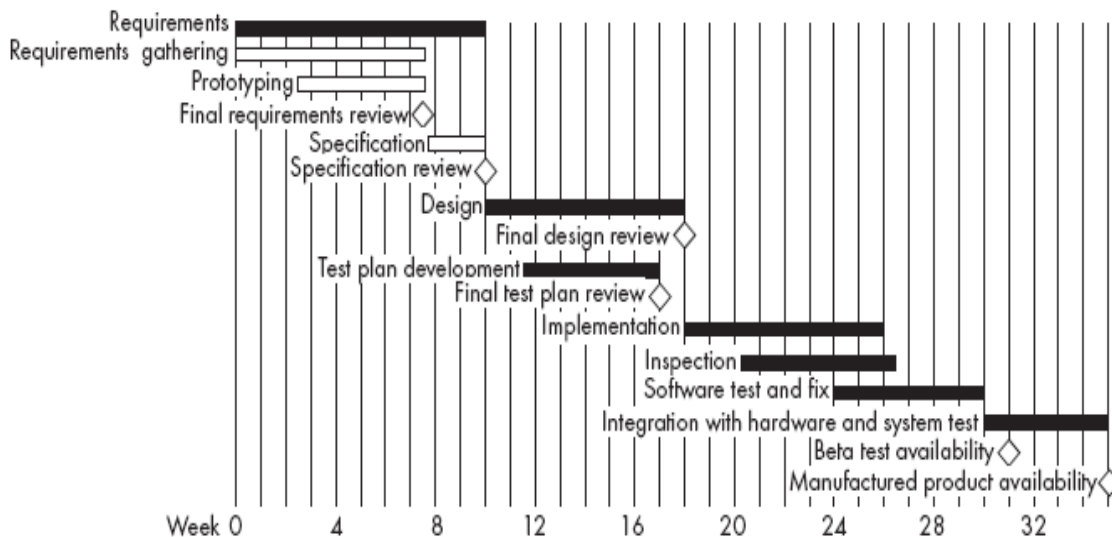
Gantt charts

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

690

Example of a Gantt chart



691

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.

692

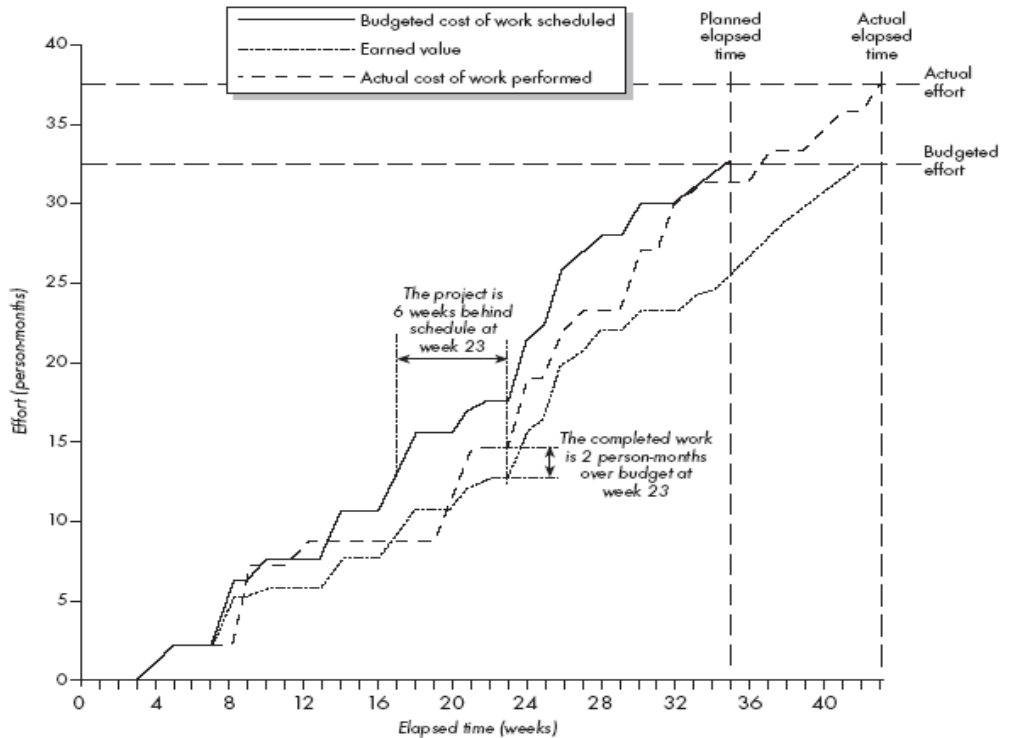
Earned value charts

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.

693

Example of an earned value chart



694

11.6 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**

695

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.*
- **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 then limit the scope of your project.* ⁶⁹⁶

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.*