Object-Oriented Software Engineering

Practical Software Development using UML and Java

Chapter 11:

Managing the Software Process

Software Engineering Projects

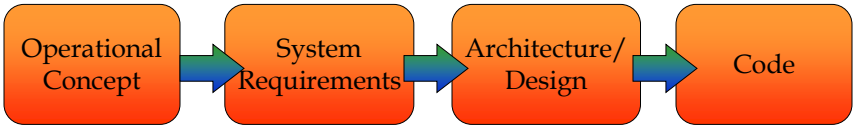
❖**The minority of projects are 'Green field' projects**

- This means - New development

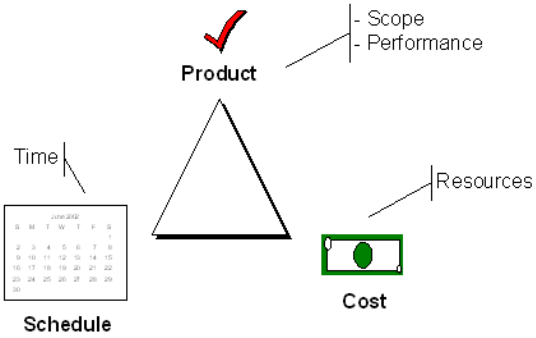
❖**Most projects are evolutionary or maintenance projects, involving work on existing systems**

- Corrective projects: fixing defects
- Adaptive projects: changes in parts of the system:
  - —Operating system
  - —Database
  - —Rules and regulations
- Enhancement projects: adding new features for users
- Reengineering or perfective projects: changing the system internally so it is more maintainable

Software development process



The Magic Triangle



3

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? (cont…)

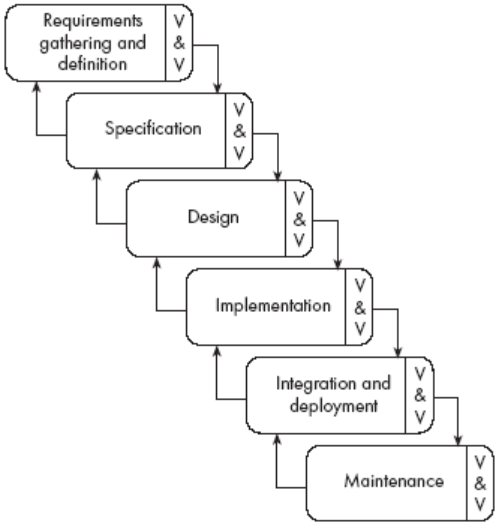
- Directing
- Being a technical leader
- Reviewing and approving decisions made by others
- Building morale and supporting staff
- Monitoring and controlling
- Co-ordinating the work with managers of other projects
- Reporting
- Continually striving to improve the process

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.

The waterfall model

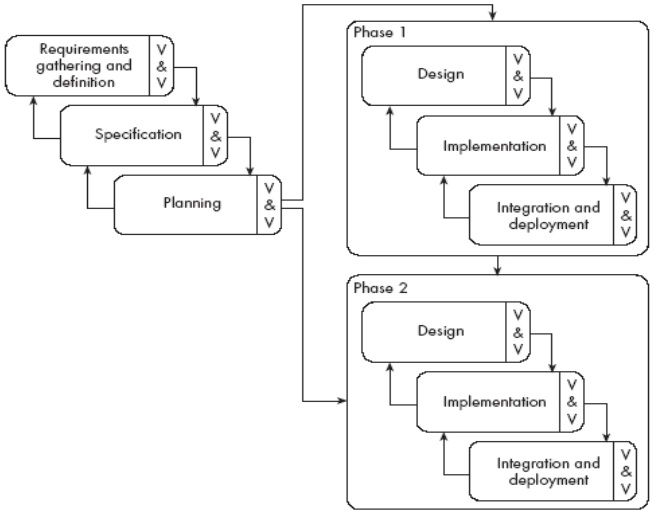


The waterfall model

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.

The phased-release model

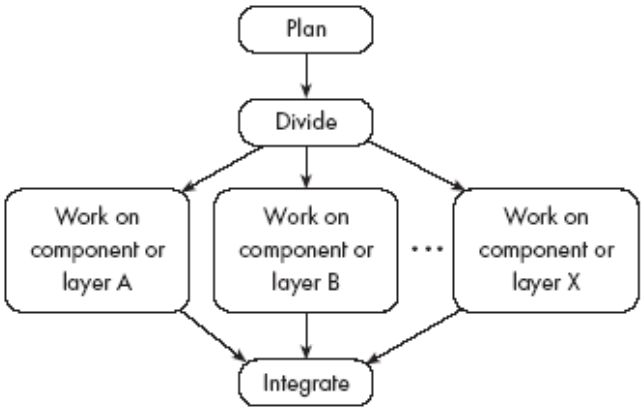


The phased-release model

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.

The concurrent engineering model



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.

Prototype

אב-טיפוס הוא גרסה חלקית פועלת של המערכת.

מטרה: לספק התנסות ממשית עם חלקים של המערכת בשלב מוקדם. הפחתת אי-ודאות ותיאום ציפיות.

סוגים:

- **Throwaway**: מקיף חלק קטן של המערכת, במטרה לבחון אספקטים מסוימים. אין לו שימוש בהמשך.
- **Evolutionary**: אב-טיפוס ראשוני, שהולך ומתפתח בשלבים עד למערכת הסופית.

יתרונות	חסרונות
• לימוד מעשיה	• איכות ירודה
• הפחתת סיכונים	• חוסר בקרה
• שיתוף משתמשים	• פער ציפיות
• חיסכון בתיעוד	• Overhead

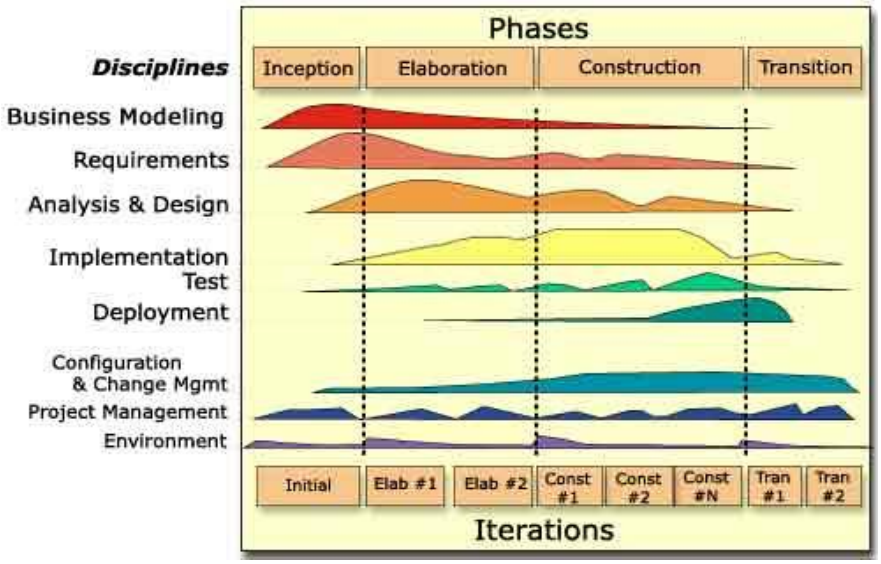
13

Unified Process

- An adaptable process framework that describes how to develop software effectively using proven techniques.
- **6 best practices:**
  - —Develop software iteratively
  - —Manage requirements
  - —Use component-based architectures
  - —Visually model software
  - —Verify software quality
  - —Control changes to software
- **4 phases:** Inception – Elaboration – Construction – Transition

14

IBM-Rational Unified Process



15

Agile Software Development

- **A conceptual framework for undertaking software engineering projects: "Lightweight", reduced documents, highly iterative**
- **Minimize risk by developing software in short time boxes, called iterations, which typically last one to four weeks.**
  - —Each iteration is like a miniature software project of its own, and includes all the tasks necessary to release the mini-increment of new functionality: planning, requirements analysis, design, coding, testing, and documentation.
- **The Agile Manifesto ("The Agile Alliance" (Feb. 2001, Snowbird, Utah))**
  - **Individuals and interactions** over **processes and tools**
  - **Working software** over **comprehensive documentation**
  - **Customer collaboration** over **contract negotiation**
  - **Responding to change** over **following a plan**

16

Principles behind the Agile Manifesto

- Satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Simplicity--the art of maximizing the amount of work not done--is essential

17

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 prototype model:
  - —Incorporate prototyping and risk analysis.
- From the Unified model:
  - —Incorporate the notion of varying amounts of time and work, with overlapping releases.
- From the concurrent engineering:
  - —Incorporate the notion of breaking the system down into components and developing them in parallel.

18

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).

Algorithmic models

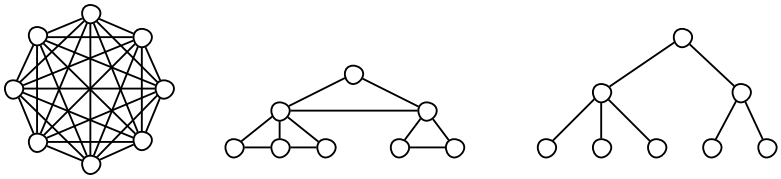
Allow you to systematically estimate development effort.

- 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
- An algorithmic model uses a formula as follows:
  - — COCOMO: $E = a + bN^c$
  - — Functions Points: $S = W_1F_1 + W_2F_2 + W_3F_3 + \ldots$

11.4 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

Software engineering teams

(a) 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.

(b) 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.

Software engineering teams

(c) 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.

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

11.5 Project Scheduling and Tracking

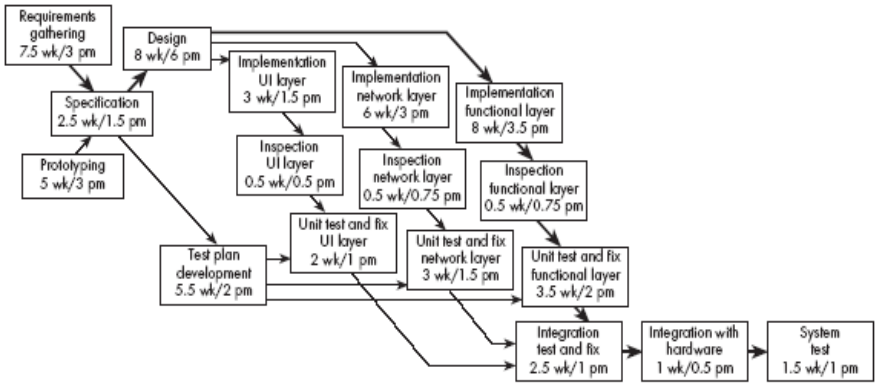
- ***Scheduling*** is the process of deciding:
  - —In what sequence a set of activities will be performed.
  - —When they should start and be completed.
- ***Tracking*** is the process of determining how well you are sticking to the cost estimate and schedule.

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.

Example of a PERT chart

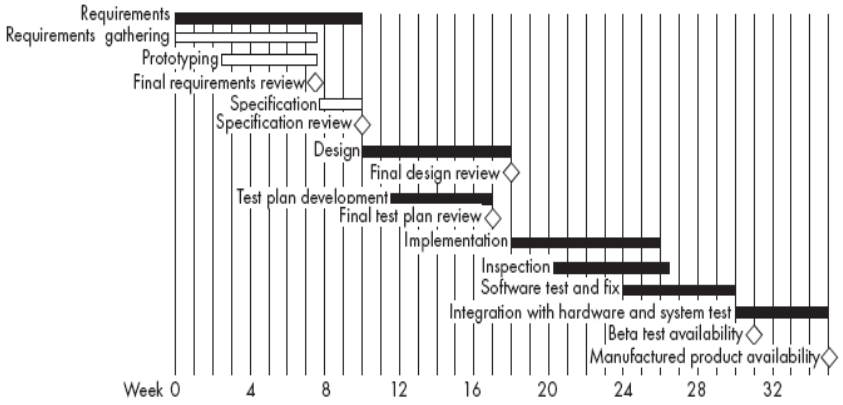


Gantt charts

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



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

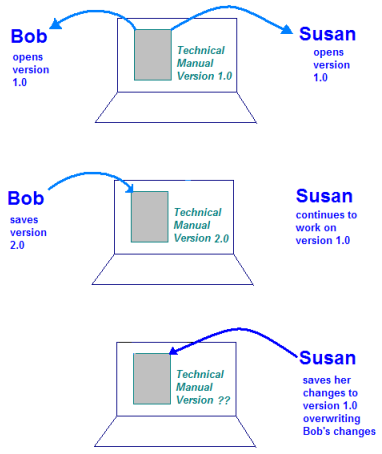
Software Configuration Management (SCM)

Used to manage and control change in project artifacts

Example: code modification scenario

1. Bob opens the document on his computer and does his work. Susan opens the document on her computer and does her work.
2. Bob completes his changes and saves the document in the repository.
3. Susan completes her changes and saves the document in the repository.

Problem?



Software Configuration Management (SCM)

Definition: A formal engineering discipline, serves as a project management support function

Description:

A set of activities designed to control software changes by

- Identifying the selected software work products and their descriptions at given points in time
- Defining mechanisms for managing different versions of these work products
- Systematically controlling and synchronizing the changes imposed
- Auditing, reporting and enabling traceability of the changes made
- Maintaining the integrity of the configuration of the software system throughout the software life cycle

Configuration Management is needed for:

— Program code artifacts
— Requirements and design artifacts
— Documentation and user manuals

SCM terminology

***Revision**:* A change in a file or set of files. A revision is one "snapshot" in a constantly changing project.

***Repository**:* The master copy where SVN stores a project's full revision history. Each project has one repository.

***Check Out**:* To request a working copy from the repository. A working copy equals the state of the project when it was checked out.

***Commit**:* To send changes from your working copy into the central repository. Also known as *check-in or submit*.

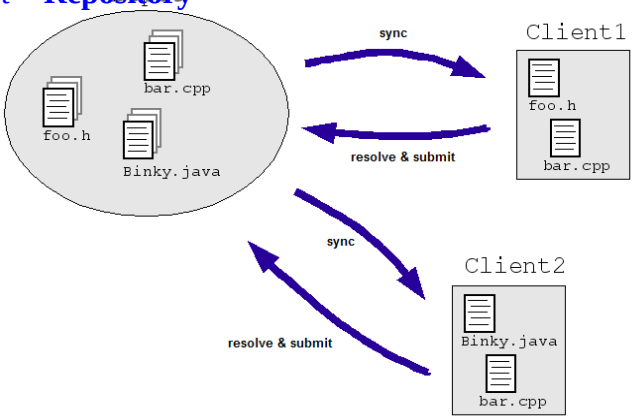
***Update**:* To bring others' changes from the repository into your working copy, or to indicate if your working copy has any uncommitted changes. This is the same as a sync, as described above. So, update/sync brings your working copy up-to-date with the repository copy.

Conflict: The situation when two engineers try to commit changes to the same area of a file. SVN indicates conflicts, but the engineers must resolve them.

***Log message**:* A comment you attach to a revision when you commit it, which describes your changes. The log provides a summary of what's been going on in a project.

SCM working scenario

Depot = **Repository**



http://subclipse.tigris.org/
http://code.google.com/edu/tools101/scm.html

Example: Synchronize and Stabilize

- By Yoffie and Cusumano. Based on Internet Explorer (Microsoft) and Communicator (Netscape) development practices.
- Combines the advantages of the spiral model with technology for overseeing and managing source code.
- Allows many teams to work efficiently in parallel.

Synchronizing: a *nightly build* (compilation) of the entire project, bringing together all the current components.

Stabilizing: change of focus before each planned code release. This includes freezing the specifications and spending the remaining time on fixing bugs.

Comprehensive verification: using an alpha release for internal testing; one or more beta releases (usually feature-complete) for wider testing outside the company, and finally a release candidate leading to a gold master, which was released to manufacturing.

35

11.7 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

- **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*