# The Human Factors and Managing People in Software Engineering

Course: Software Engineering

## Lectures 3 and 4

Lecturer: Alexander Vazhenin

E-mail: vazhenin@u-aizu.ac.jp

# Objectives

- To describe simple models of human cognition and their relevance for software managers
- To explain the key issues that determine the success or otherwise of team working
- To discuss the problems of selecting and retaining technical staff
- To introduce the people capability maturity model (P-CMM)

# Topics covered

- Human Diversity and Limits to thinking
- Management Activities
- Group working
- Choosing and keeping people
- The people capability maturity model

# Importance

- Knowledge of human factors can provide insights into the software engineering.
- Software manager must understand their staff as individuals, and how they interact with each other.
- Psychology helps understand limits and capabilities of software engineers and users.
- Can help identify possible ways of increasing productivity.

# Human Diversity

- Software development is an individual, creative task, comparable with architecture.
- Personality traits: such as extroversion or introversion, assertive or humble, trusting or suspicious, and so on.
- Personality is a dynamic combination of these traits depending on the individual itself and the environment.
- Group loyalties versus challenging status quo of each person.

# Software Engineer Personality

- There is no evidence to suggest that programming ability is related to any particular personality trait.
- However, some enjoy certain activities more than others due to certain characteristics.
- Software Engineer Personality is also in the ability to withstand a certain amount of stress as well as have an adaptive ability to new technologies and problem solving capabilities.

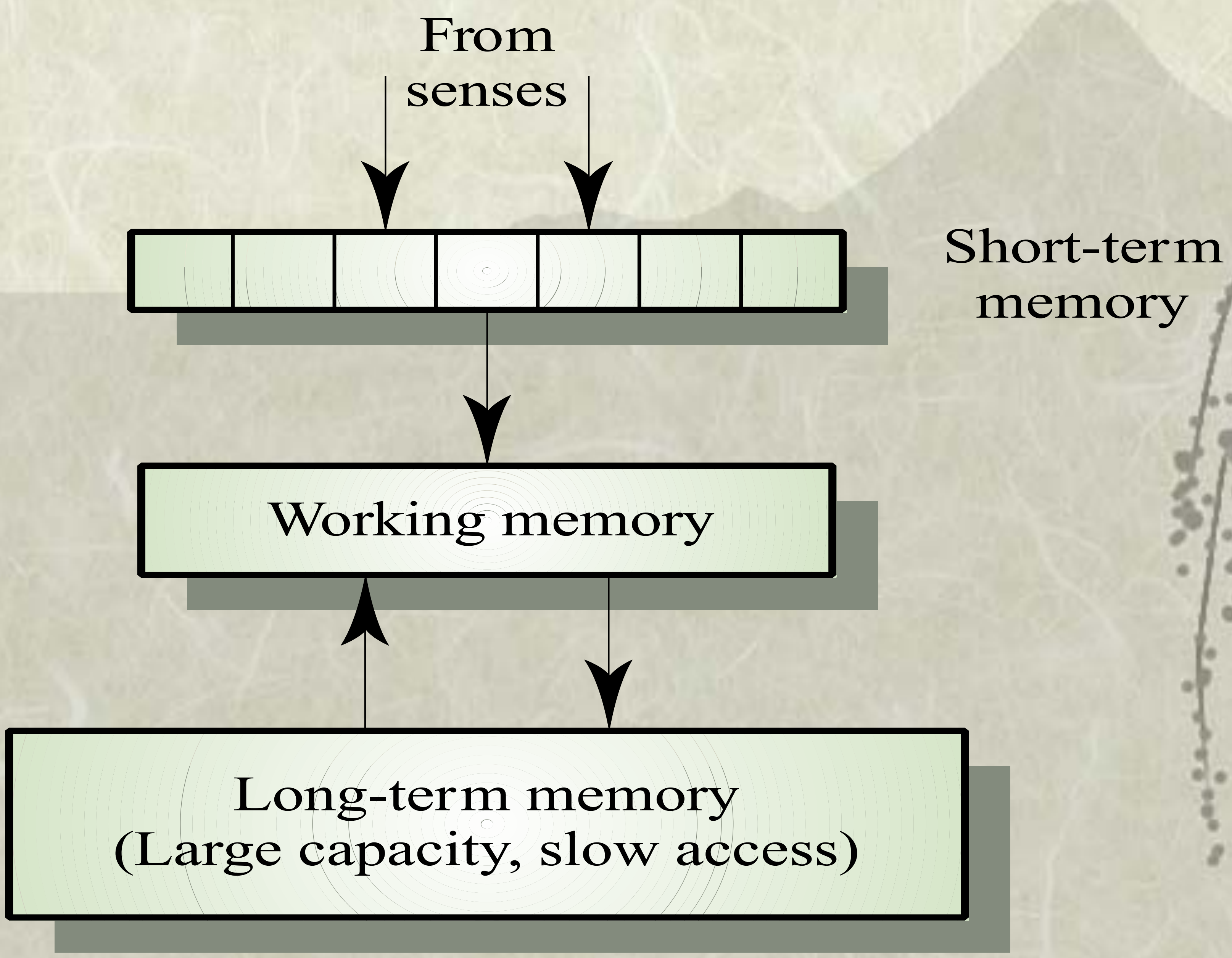
# People in the process

- People are an organisation’s most important assets
- The tasks of a manager are essentially people oriented. Unless there is some understanding of people, management will be unsuccessful
- Software engineering is primarily a cognitive activity. Cognitive limitations effectively limit the software process

# Limits to thinking

- People don’t all think the same way but everyone is subject to some basic constraints on their thinking due to
  - Memory organization
  - Knowledge representation
  - Motivation influences
- If we understand these constraints, we can understand how they affect people participating in the software process

# Memory organization



# Short-term memory

- Fast access, limited capacity
- 5-7 locations
- Holds 'chunks' of information where the size of a chunk may vary depending on its familiarity
- Fast decay time

# Working memory

- Larger capacity, longer access time
- Memory area used to integrate information from short-term memory and long-term memory.
- Relatively fast decay time.

# Long-term memory

- Slow access, very large capacity
- Unreliable retrieval mechanism
- Slow but finite decay time - information needs reinforced
- Relatively high threshold - work has to be done to get information into long-term memory.

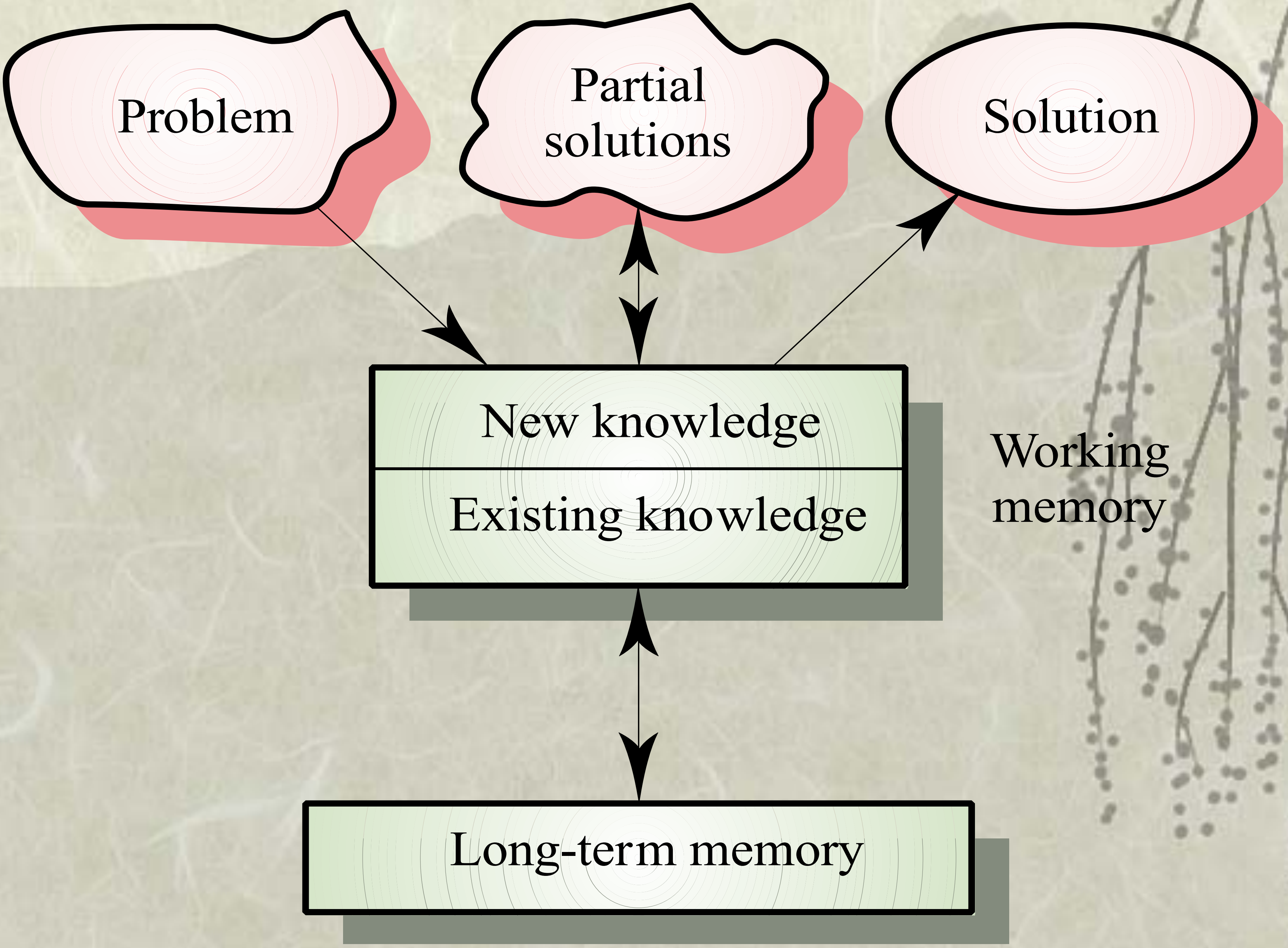
# Information transfer

- Problem solving usually requires transfer between short-term memory and working memory
- Information may be lost or corrupted during this transfer
- Information processing occurs in the transfer from short-term to long-term memory

# Problem solving

- Requires the integration of different types of knowledge (computer, task, domain, organization)
- Development of a semantic model of the solution and testing of this model against the problem
- Representation of this model in an appropriate notation or programming language

# Problem solving



# Cognitive Chunking

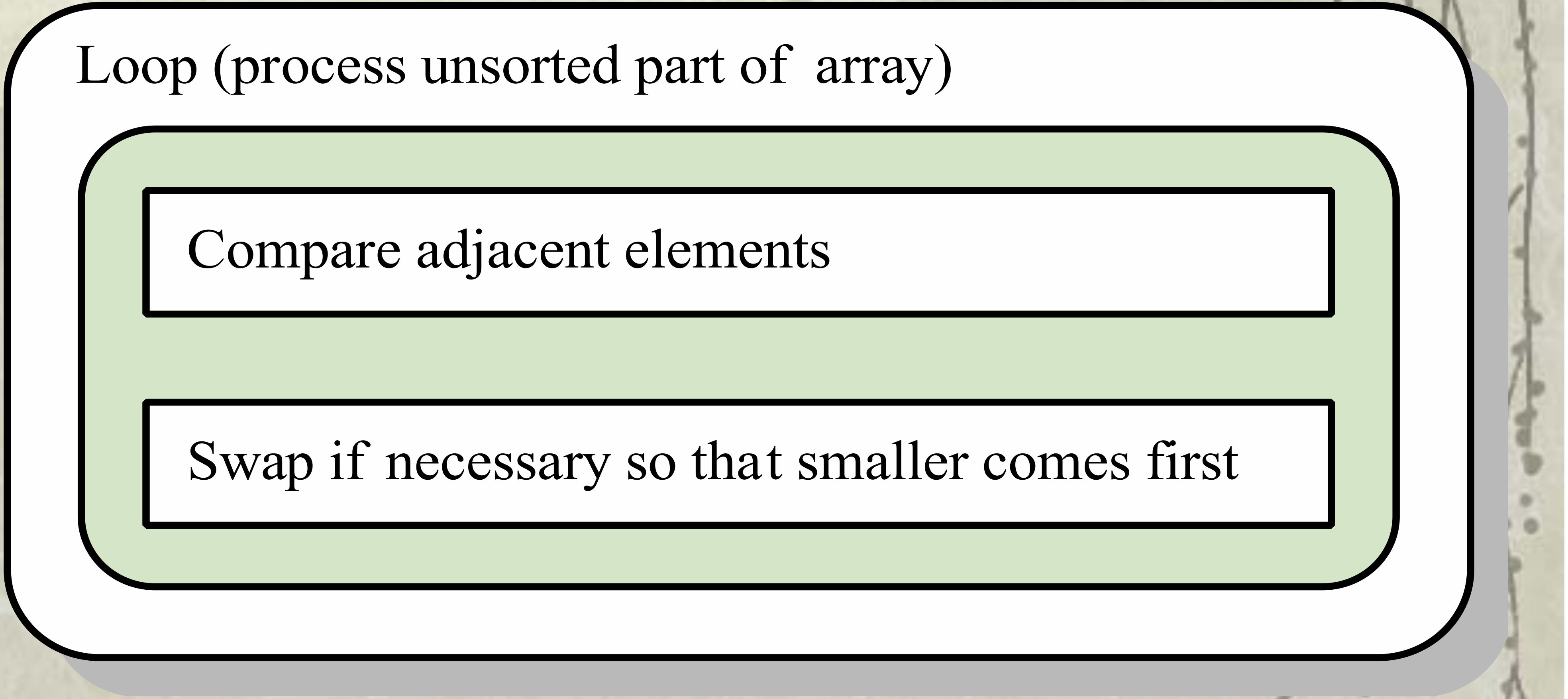
- Chunking in psychology is a process by which individual pieces of information are bound together into a meaningful whole.
- A chunk is defined as a familiar collection of more elementary units that have been inter-associated and stored in memory repeatedly and act as a coherent, integrated group when retrieved.

# Cognitive Chunking

- It is believed that individuals create higher order cognitive representations of the items on the list that are more easily remembered as a group than as individual items themselves.
- Representations of these groupings are highly subjective, as they depend critically on the individual's perception of the features of the items and the individual's semantic network.

# Cognitive chunking (Example)

Loop (process entire array)



Loop (process unsorted part of array)

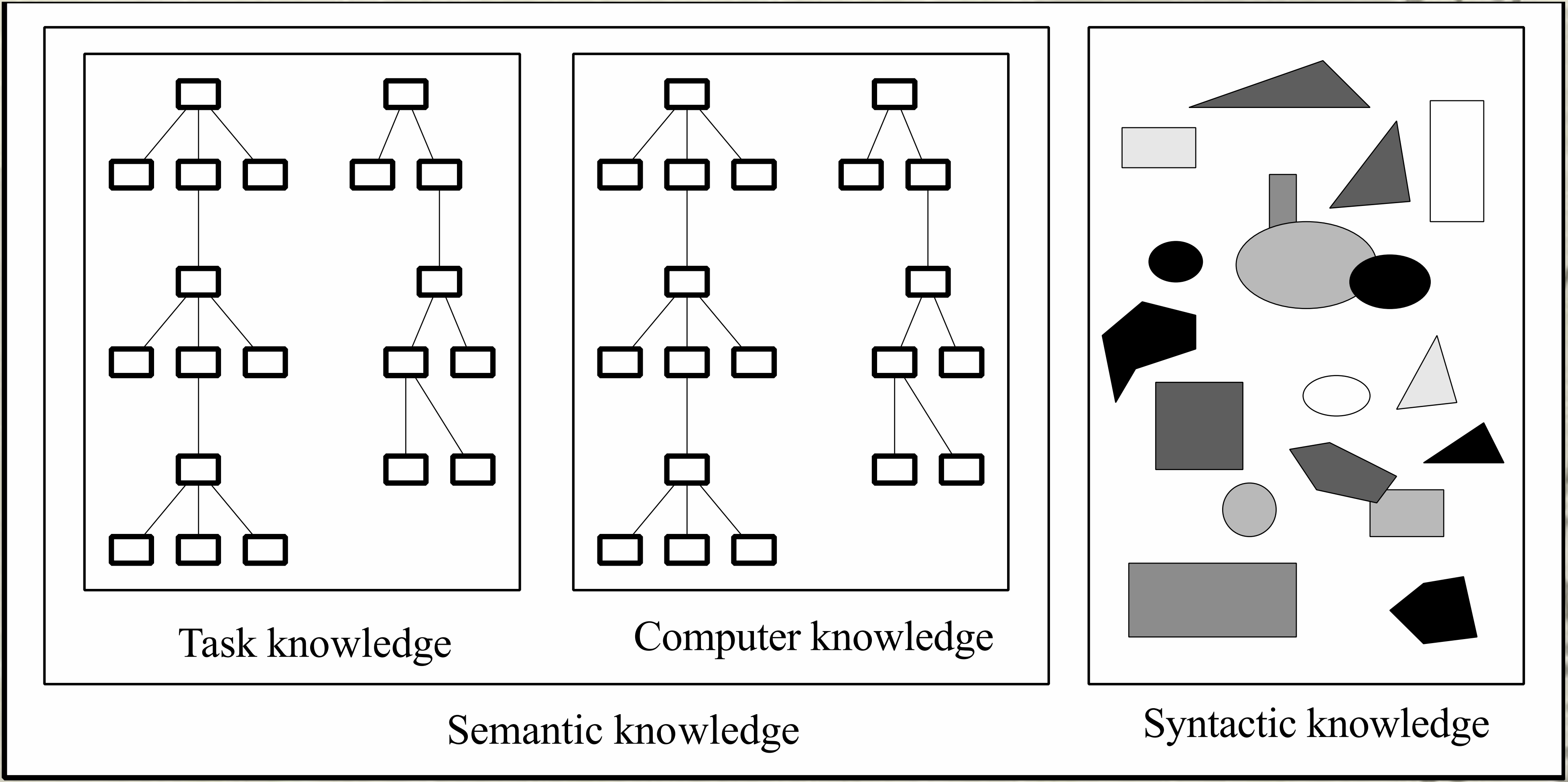
Compare adjacent elements

Swap if necessary so that smaller comes first

# Knowledge modelling

- Semantic knowledge  knowledge of concepts such as the operation of assignment, concept of parameter passing etc.
- Syntactic knowledge   knowledge of details of a representation e.g. an Ada while loop.
- Semantic knowledge seems to be stored in a structured, representation independent way.

# Syntactic/semantic knowledge



# Knowledge acquisition

- Semantic knowledge through experience and active learning - the 'ah' factor
- Syntactic knowledge acquired by memorisation.
- New syntactic knowledge can interfere with existing syntactic knowledge.
  - Problems arise for experienced programmers in mixing up syntax of different programming languages

# Semantic knowledge

- **Computing concepts** - notion of a writable store, iteration, concept of an object, etc.
- **Task concepts** - principally algorithmic - how to tackle a particular task
- Software development ability is the ability to integrate new knowledge with existing computer and task knowledge and hence derive creative problem solutions
- Thus, problem solving is language independent

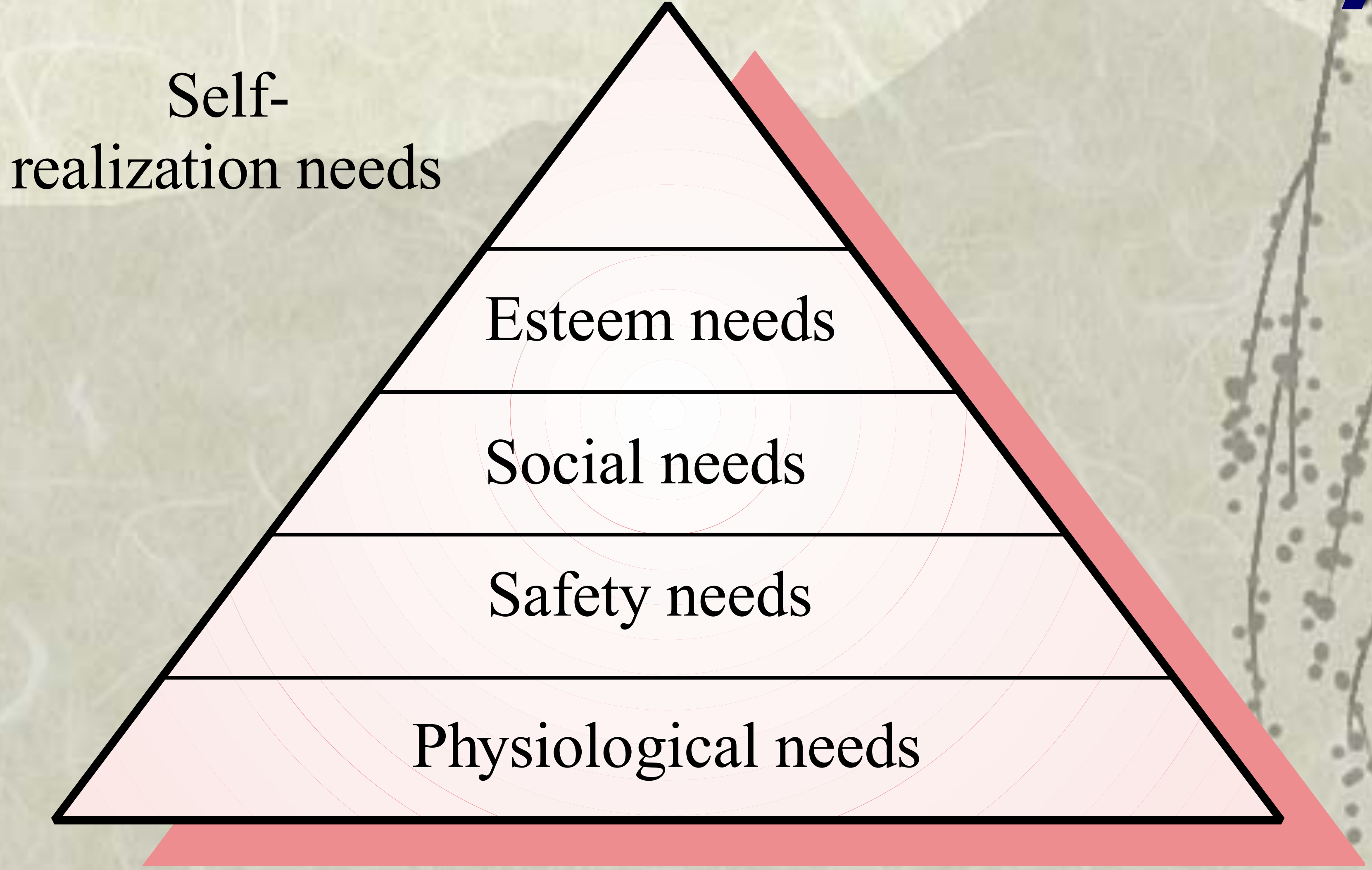
# Management activities

- Problem solving (using available people)
- Motivating (people who work on a project)
- Planning (what people are going to do)
- Estimating (how fast people will work)
- Controlling (people's activities)
- Organising (the way in which people work)

# Motivation

- An important role of a manager is to motivate the people working on a project
- Motivation is a complex issue but it appears that their are different types of motivation based on
    - Basic needs (e.g. food, sleep, etc.)
    - Personal needs (e.g. respect, self-esteem)
    - Social needs (e.g. to be accepted as part of a group)

# Human needs hierarchy



# Motivating people

- Motivations depend on satisfying needs
- It can be assumed that physiological and safety needs are satisfied
- Social, esteem and self-realization needs are most significant from a managerial viewpoint

# Need satisfaction

- Social
  - Provide communal facilities
  - Allow informal communications
- Esteem
  - Recognition of achievements
  - Appropriate rewards
- Self-realization
  - Training - people want to learn more
  - Responsibility

# Personality types

- ❑ The needs hierarchy is almost certainly an over-simplification
- ❑ Motivation should also take into account different personality types:
  - ➢Task-oriented
  - ➢Self-oriented
  - ➢Interaction-oriented

# Personality types

- Task-oriented.
  - The motivation for doing the work is the work itself
- Self-oriented.
  - The work is a means to an end which is the achievement of individual goals - e.g. to get rich, to play tennis, to travel etc.
- Interaction-oriented
  - The principal motivation is the presence and actions of co-workers. People go to work because they like to go to work

# Motivation balance

- Individual motivations are made up of elements of each class
- Balance can change depending on personal circumstances and external events
- However, people are not just motivated by personal factors but also by being part of a group and culture.
- People go to work because they are motivated by the people that they work with

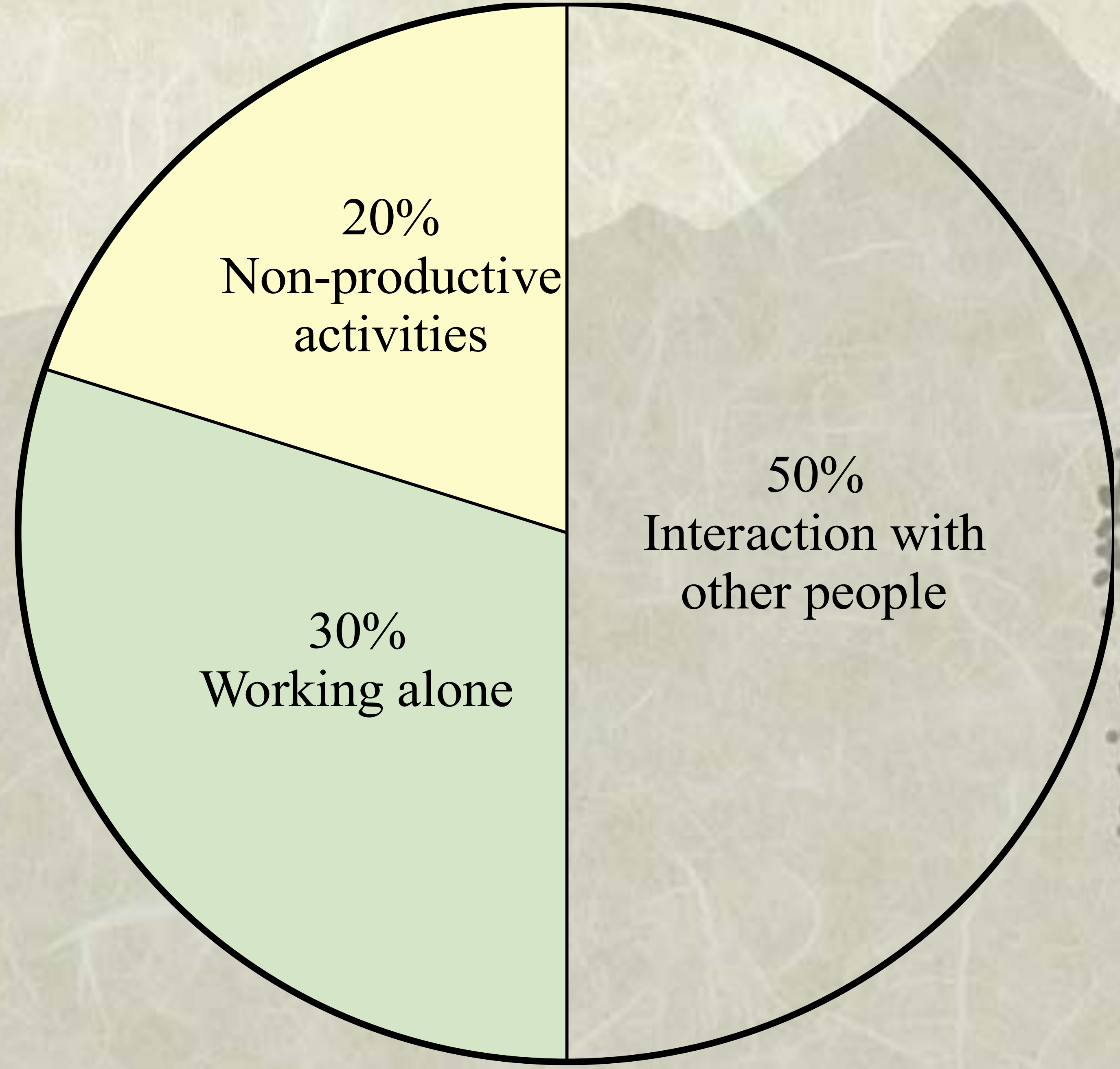
# Teams

- Software engineers frequently work in groups.
- More commonly, several software engineers share an office and other resources.
- They can work on different projects or collaborating on larger projects.
- So, software development is essentially a social activity

# Group working

- Most software engineering is a group activity
  - The development schedule for most non-trivial software projects is such that they cannot be completed by one person working alone
- Group interaction is a key determinant of group performance
- Flexibility in group composition is limited
  - Managers must do the best they can with available people

# Time distribution



# The Communication Problems

- People working on a piece of software obviously have to interact with each other. This leads to the following problems:
  - How to make module specification and module interface?
  - How to make software items names and file layouts?
  - How to work in situation when someone leaves the project temporarily or new people join the project?
  - Too many people added to a team can swamp a project due to communication overhead.
  - If a project is late, then adding further people will only make it even later.

# Group composition

- Group composed of members who share the same motivation can be problematic
  - Task-oriented - everyone wants to do their own thing
  - Self-oriented - everyone wants to be the boss
  - Interaction-oriented - too much chatting, not enough work
- An effective group has a balance of all types
- Can be difficult to achieve because most engineers are task-oriented
- Need for all members to be involved in decisions which affect the group

# Group leadership

- Leadership depends on respect not titular status
- There may be both a technical and an administrative leader
- Democratic leadership is more effective that autocratic leadership
- A career path based on technical competence should be supported

# Group cohesiveness

- In a cohesive group, members consider the group to be more important than any individual in it
- Advantages of a cohesive group are:
  - Group quality standards can be developed
  - Group members work closely together so inhibitions caused by ignorance are reduced
  - Team members  learn from each other and get to know each other's work
  - *Egoless programming* where members strive to improve each other's programs can be practised

# The Individual and the Error

- Software is seen as personal work of art, an extension of ourselves
- We do not like to see/recognize faults in ourselves and our beliefs
- People do not like people pointing out their faults either cognitive dissonance.

# Egoless Programming

- **Ego:** Individual perception of oneself; self-esteem
- Someone responsible for a software tends to defend that software against any criticism, even if it has obvious shortcomings
- Someone's ego is tied up with the software itself

# Egoless Programming

- Software production is a team effort and responsibility of the team
- Criticisms should not be taken as personal attacks
- Informal technique carried out by colleagues in a friendly manner
- Encourages team communication without regard to status, experience or sex

# Developing cohesiveness

- Cohesiveness is influenced by factors such as the organisational culture and the personalities in the group
- Cohesiveness can be encouraged through
  - Social events
  - Developing a group identity and territory
  - Explicit team-building activities
- Openness with information is a simple way of ensuring all group members feel part of the group

# Group loyalties

- Group members tend to be loyal to cohesive groups
- 'Groupthink' is preservation of group irrespective of technical or organizational considerations
- Management should act positively to avoid groupthink by forcing external involvement with each group

# Group communications

- Good communications are essential for effective group working
- Information must be exchanged on the status of work, design decisions and changes to previous decisions
- Good communications also strengthens group cohesion as it promotes understanding

# Group communications

- Status of group members
  - Higher status members tend to dominate conversations
- Personalities in groups
  - Too many people of the same personality type can be a problem
- Sexual composition of group
  - Mixed-sex groups tend to communicate better
- Communication channels
  - Communications channelled though a central coordinator tend to be ineffective

# SE Team Types

- **Functional team:** same type of work on different projects
  - *Problem:* The project manager is not the leader of the team
- **Project team:** different work on the same system
  - *Problem:* Technical competence does not necessarily imply excellence in leadership

# The Project Team Structure

- Team organization is implemented in an well-defined form
- Divide the work among skilled specialists
- Each individual role is clear
- Communication is minimized
- Hierarchical structure usually includes:
  - Senior software engineer, junior software engineer, expert software engineers, and a librarian

# Manager

- Administrative affairs like reporting to the organization
- Monitoring budgets and time-scales
- Control over the work process by effective management
- Motivate, evaluate, arbitrate
- Not really part of the team, and may deal with several teams

# Senior Software Engineer

- Highly skilled technical professional, but does not want to go into management.
- In charge of the architectural design of the software
- Produces the high level part of the system
- Determines the constituent modules and programs of the software
- Controls the crucial part of the system
- Oversees the component tests and integration of the complete system

# Junior Software Engineer

- Technical person whose potential skills are comparable to the senior
- Helps the senior with secondary tasks
- Acts as senior, when the latter is absent for any reason
- If the senior leaves, the junior can immediately take over as a back-up

# Librarian

- Maintains the documentation associated with the project
- Keeps reports and relevant technical materials to the project
- Must have computer literacy
- The current software tools allows to implement some of the librarian's work

# Expert Software Engineers

- When needed, they are brought into the team to develop subsystems specified by the senior
- Experts in a particular software area like the object-oriented paradigm, computer graphics, ...
- May be a guru, a hacker, or a tester

# Group organisation

- Software engineering group sizes should be relatively small (< 8 members)
- Small teams may be organised in an informal, democratic way
- Chief programmer teams try to make the most effective use of skills and experience

# Very Large Projects

- ❑ Break big projects down into multiple smaller projects:
  - ➢ Start the project with a TEAM of senior software engineers
  - ➢ Do the high level software architecture
  - ➢ Break the team and its members become seniors within the set of teams that develops the subsystem
  - ➢ Original team carries out system integration and validation

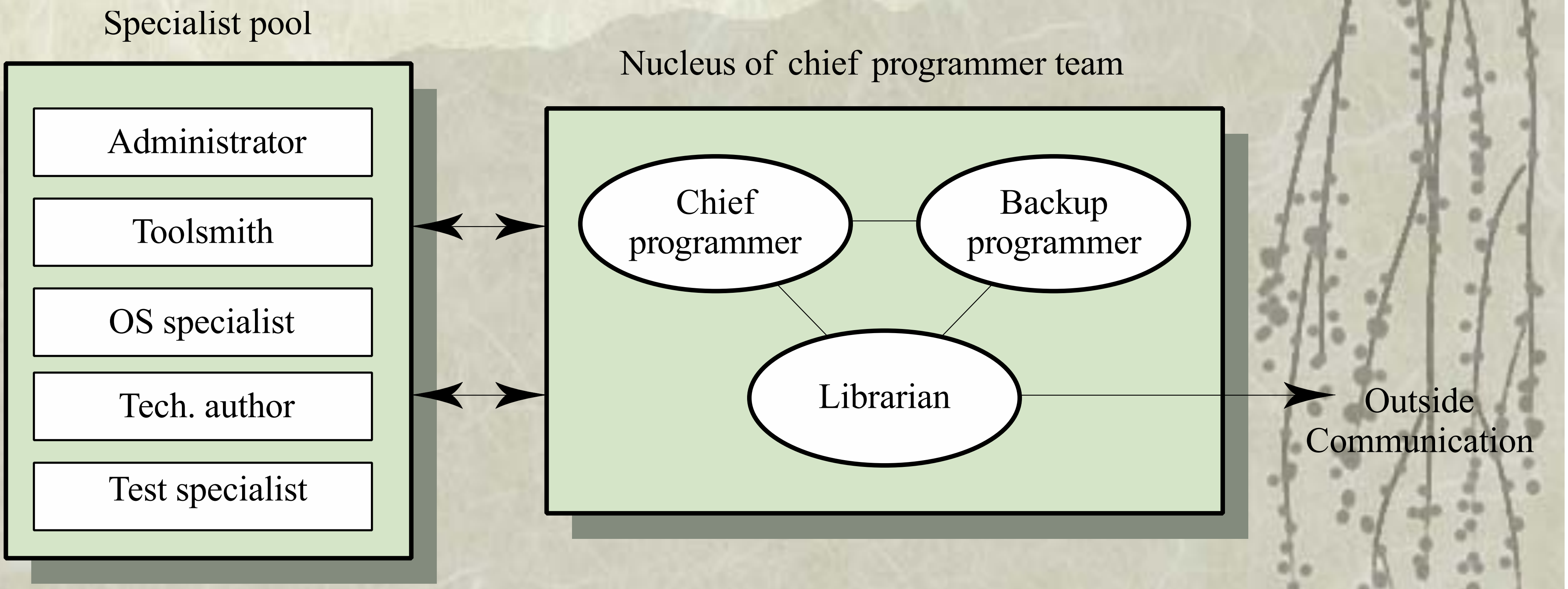
# Democratic team organisation

- The group acts as a whole and comes to a consensus on decisions affecting the system
- The group leader serves as the external interface of the group but does not allocate specific work items
- Rather, work is discussed by the group as a whole and tasks are allocated according to ability and experience
- This approach is successful for groups where all members are experienced and competent

# Extreme programming groups

- Extreme programming groups are variants of democratic organisation
- In extreme programming groups, some ‘management’ decisions are devolved to group members
- Programmers work in pairs and take a collective responsibility for code that is developed

# Chief programmer teams



# Chief programmer teams

- Consist of a kernel of specialists helped by others added to the project as required
- The motivation behind their development is the wide difference in ability in different programmers
- Chief programmer teams provide a supporting environment for very able programmers to be responsible for most of the system development

# Problems

- This chief programmer approach, in different forms, has undoubtedly been successful
- However, it suffers from a number of problems
  - Talented designers and programmers are hard to find. Without exception people in these roles, the approach will fail
  - Other group members may resent the chief programmer taking the credit for success so may deliberately undermine his/her role
  - High project risk as the project will fail if both the chief and deputy programmer are unavailable
  - Organisational structures and grades may be unable to accommodate this type of group

# Choosing and keeping people

- Choosing people to work on a project is a major managerial responsibility
- Appointment decisions are usually based on
  - information provided by the candidate (their resume or CV)
  - information gained at an interview
  - recommendations from other people who know the candidate
- Some companies use psychological or aptitude tests
  - There is no agreement on whether or not these tests are actually useful

# Staff selection factors

| Factor | Explanation |
|---|---|
| Application domain experience | For a project to develop a successful system, the developers must understand the application domain. |
| Platform experience | May be significant if low-level programming is involved. Otherwise, not usually a critical attribute. |
| Programming language experience | Normally only significant for short duration projects where there is insufficient time to learn a new language. |
| Educational background | May provide an indicator of the basic fundamentals which the candidate should know and of their ability to learn. This factor becomes increasingly irrelevant as engineers gain experience across a range of projects. |
| Communication ability | Very important because of the need for project staff to communicate orally and in writing with other engineers, managers and customers. |
| Adaptability | Adaptability may be judged by looking at the different types of experience which candidates have had. This is an important attribute as it indicates an ability to learn. |
| Attitude | Project staff should have a positive attitude to their work and should be willing to learn new skills. This is an important attribute but often very difficult to assess. |
| Personality | Again, an important attribute but difficult to assess. Candidates must be reasonably compatible with other team members. No particular type of personality is more or less suited to software engineering. |

# Working environments

- Physical workplace provision has an important effect on individual productivity and satisfaction
  - Comfort
  - Privacy
  - Facilities
- Health and safety considerations must be taken into account
  - Lighting
  - Heating
  - Furniture

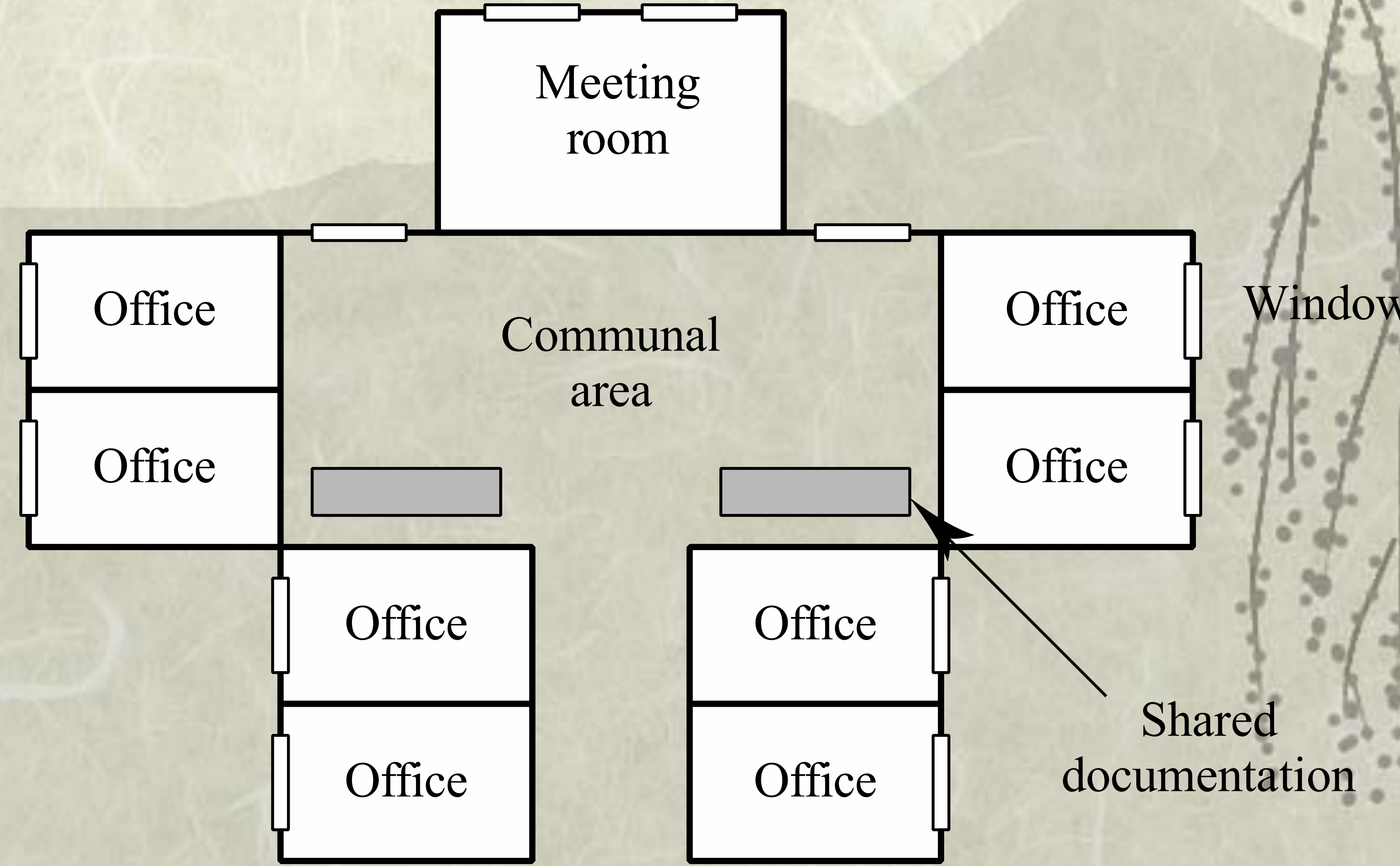
# Environmental factors

- Privacy - each engineer requires an area for uninterrupted work
- Outside awareness - people prefer to work in natural light
- Personalization - individuals adopt different working practices and like to organize their environment in different ways

# Workspace organisation

- Workspaces should provide private spaces where people can work without interruption
  - Providing individual offices for staff has been shown to increase productivity
- However, teams working together also require spaces where formal and informal meetings can be held

# Office layout

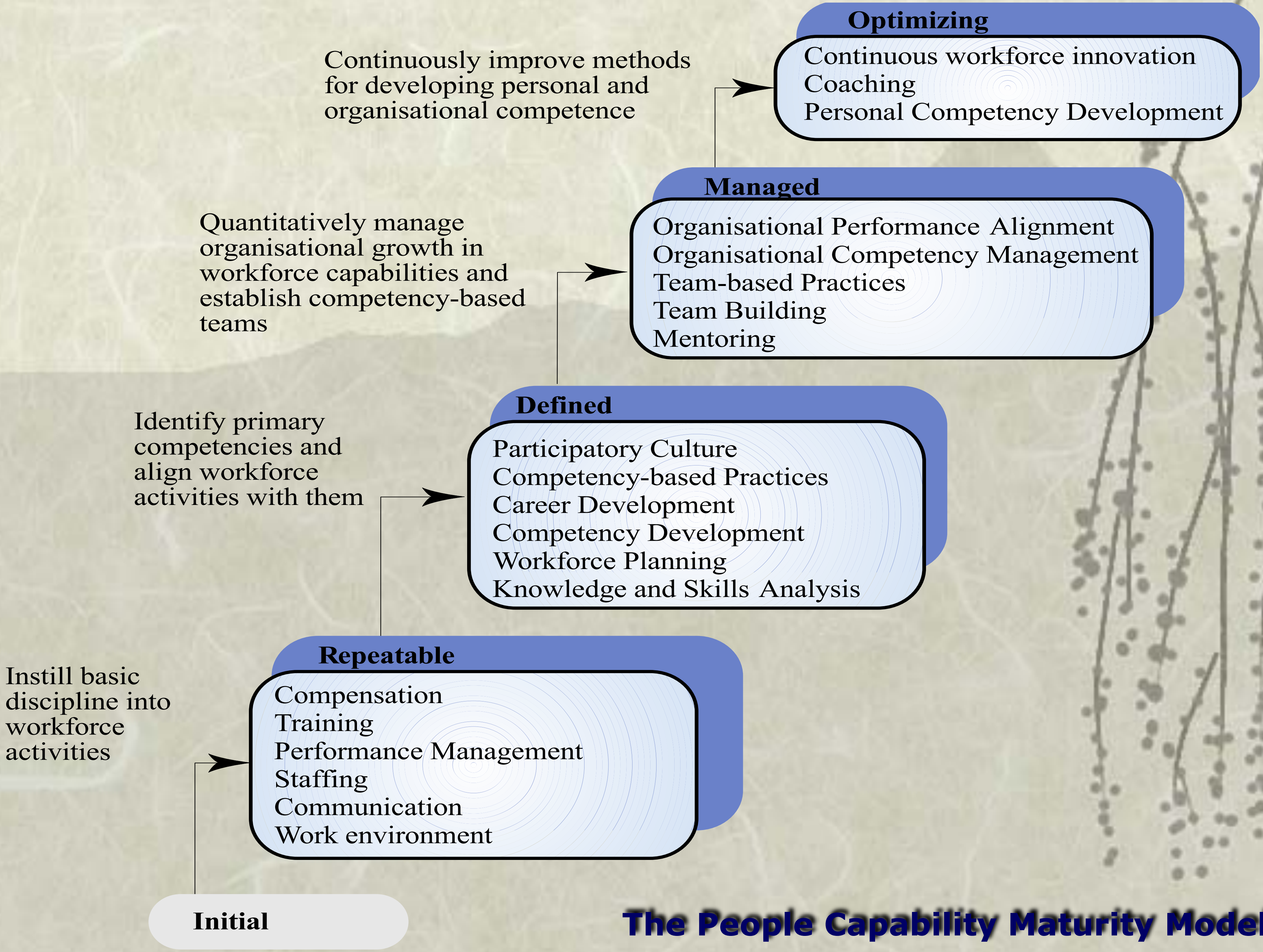


# The People Capability Maturity Model

- Intended as a framework for managing the development of people involved in software development

# The People Capability Maturity Model

- Five stage model
  - Initial. Ad-hoc people management
  - Repeatable. Policies developed for capability improvement
  - Defined. Standardised people management across the organisation
  - Managed. Quantitative goals for people management in place
  - Optimizing. Continuous focus on improving individual competence and workforce motivation



Continuously improve methods for developing personal and organisational competence

**Optimizing**

Continuous workforce innovation
Coaching
Personal Competency Development

Quantitatively manage organisational growth in workforce capabilities and establish competency-based teams

**Managed**

Organisational Performance Alignment
Organisational Competency Management
Team-based Practices
Team Building
Mentoring

Identify primary competencies and align workforce activities with them

**Defined**

Participatory Culture
Competency-based Practices
Career Development
Competency Development
Workforce Planning
Knowledge and Skills Analysis

Instill basic discipline into workforce activities

**Repeatable**

Compensation
Training
Performance Management
Staffing
Communication
Work environment

**Initial**

# The People Capability Maturity Model

# Walk-through Meetings

- at which a software item is examined by a group of colleagues
- Goal: Try to find errors and omissions
- Walk-through involves a step-by-step explanation  of the software item
- What we cannot see may be obvious to someone else
- By discussing with other people, errors will be found more quickly
- No one should threat, so no one has to become defensive

# Walk-throughs Key Points

- Everyone in the meeting must be fully involved
- Expect the participants to study the material *prior* to the meeting
- Concentrate attention on the *software* rather than the person
- The meeting should look businesslike
- An assertive moderator should coordinate activities and establish the rules
- Restrict the activity to *identifying* problems first, not solving them
- Briefly document the faults

# Walk-throughs Benefits

- **Software quality is improved:** more bugs have been eliminated
- **Programming effort is reduced:** specification are clearer before implementation
- **Meeting deadlines is improved:** major catastrophes are avoided early
- **Programmer expertise is enhanced:** everyone is learning from everyone else

# THINK

- Technical competence and administrative competence are not necessarily synonymous.
- The dangers of group-think and how it can be avoided.