Managing People and Cost


Tutorial Questions
  - Question 22.1
  - Question 22.3
  - Question 22.7
  - Question 22.9
  - Question 23.3
  - Question 23.5
  - Question 23.6

Managing people
- Managing people working as individuals and in groups

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
- Limits to thinking
- Group working
- Choosing and keeping people
- The people capability maturity model
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

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)

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 organisation
  - Knowledge representation
  - Motivation influences
- If we understand these constraints, we can understand how they affect people participating in the software process

Memory organisation

- 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

### Cognitive chunking
**Loop (process entire array)**

1. Compare adjacent elements
2. 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.

### 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

**Problem solving**

- Requires the integration of different types of knowledge (computer, task, domain, organisation)
- 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**

- Requires the integration of different types of knowledge (computer, task, domain, organisation)
- 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

**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 there 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)

**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

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
- 50% Interaction with other people
- 30% Working alone
- 20% Non-productive activities
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 than 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

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 through a central coordinator tend to be ineffective

Group organisation

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

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

• 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

<table>
<thead>
<tr>
<th>Factor</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application domain experience</td>
<td>For a project to develop a successful system, the developers must understand the application domain.</td>
</tr>
<tr>
<td>Platform experience</td>
<td>May be significant if low-level programming is involved. Otherwise, not usually a critical attribute.</td>
</tr>
<tr>
<td>Programming language experience</td>
<td>Normally only significant for short-duration projects where there is insufficient time to learn a new language.</td>
</tr>
<tr>
<td>Educational background</td>
<td>May provide an indicator of the basic fundamental knowledge the candidate should have and of their ability to learn. This factor becomes increasingly important as engineers gain experience across a range of projects.</td>
</tr>
<tr>
<td>Communication ability</td>
<td>Very important because of the need for project staff to communicate easily and to write with other engineers, managers, and customers.</td>
</tr>
<tr>
<td>Adaptability</td>
<td>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.</td>
</tr>
<tr>
<td>Attitude</td>
<td>Project staff should have a positive attitude to their work and should be willing to learn new skills. This is an important attribute that often very difficult to judge at interview.</td>
</tr>
<tr>
<td>Personality</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.
The People Capability Maturity Model

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

P-CMM Objectives

- To improve organisational capability by improving workforce capability
- To ensure that software development capability is not reliant on a small number of individuals
- To align the motivation of individuals with that of the organisation
- To help retain people with critical knowledge and skills

Key points

- Managers must have some understanding of human factors to avoid making unrealistic demands on people
- Problem solving involves integrating information from long-term memory with new information from short-term memory
- Staff selection factors include education, domain experience, adaptability and personality

Key points

- Software development groups should be small and cohesive
- Group communications are affected by status, group size, group organisation and the sexual composition of the group
- The working environment has a significant effect on productivity
- The People Capability Maturity Model is a framework for improving the capabilities of staff in an organisation
Chapter 23
Software Cost Estimation

Software cost estimation
- Predicting the resources required for a software development process

Objectives
- To introduce the fundamentals of software costing and pricing
- To describe three metrics for software productivity assessment
- To explain why different techniques should be used for software estimation
- To describe the COCOMO 2 algorithmic cost estimation model

Topics covered
- Productivity
- Estimation techniques
- Algorithmic cost modelling
- Project duration and staffing

Fundamental estimation questions
- How much effort is required to complete an activity?
- How much calendar time is needed to complete an activity?
- What is the total cost of an activity?
- Project estimation and scheduling and interleaved management activities

Software cost components
- Hardware and software costs
- Travel and training costs
- Effort costs (the dominant factor in most projects)
  - salaries of engineers involved in the project
  - Social and insurance costs
- Effort costs must take overheads into account
  - costs of building, heating, lighting
  - costs of networking and communications
  - costs of shared facilities (e.g. library, staff restaurant, etc.)
Costing and pricing

- Estimates are made to discover the cost, to the developer, of producing a software system.
- There is not a simple relationship between the development cost and the price charged to the customer.
- Broader organisational, economic, political and business considerations influence the price charged.

Software pricing factors

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market opportunity</td>
<td>A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the opportunity of more profit later. The experience gained may allow new products to be developed.</td>
</tr>
<tr>
<td>Cost estimate uncertainty</td>
<td>If an organisation is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit.</td>
</tr>
<tr>
<td>Contractual terms</td>
<td>A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.</td>
</tr>
<tr>
<td>Requirements volatility</td>
<td>If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices may be charged for changes to the requirements.</td>
</tr>
<tr>
<td>Financial health</td>
<td>Developers in financial difficulty may lower their price to gain a contract. It is better to make a small profit or be broken than to go out of business.</td>
</tr>
</tbody>
</table>

Programmer productivity

- A measure of the rate at which individual engineers involved in software development produce software and associated documentation.
- Not quality-oriented although quality assurance is a factor in productivity assessment.
- Essentially, we want to measure useful functionality produced per time unit.

Productivity measures

- Size related measures based on some output from the software process. This may be lines of delivered source code, object code instructions, etc.
- Function-related measures based on an estimate of the functionality of the delivered software. Function-points are the best known of this type of measure.

Measurement problems

- Estimating the size of the measure.
- Estimating the total number of programmer months which have elapsed.
- Estimating contractor productivity (e.g. documentation team) and incorporating this estimate in overall estimate.

Lines of code

- What’s a line of code?
  - The measure was first proposed when programs were typed on cards with one line per card.
  - How does this correspond to statements as in Java which can span several lines or where there can be several statements on one line?
- What programs should be counted as part of the system?
- Assumes linear relationship between system size and volume of documentation.
Productivity comparisons

- The lower level the language, the more productive the programmer
  - The same functionality takes more code to implement in a lower-level language than in a high-level language
- The more verbose the programmer, the higher the productivity
  - Measures of productivity based on lines of code suggest that programmers who write verbose code are more productive than programmers who write compact code

System development times

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Design</th>
<th>Coding</th>
<th>Testing</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assembly code</strong></td>
<td>3 weeks</td>
<td>5 weeks</td>
<td>8 weeks</td>
<td>10 weeks</td>
</tr>
<tr>
<td><strong>High-level language</strong></td>
<td>3 weeks</td>
<td>5 weeks</td>
<td>8 weeks</td>
<td>6 weeks</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td><strong>Effort</strong></td>
<td><strong>Productivity</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assembly code</td>
<td>5000 lines</td>
<td>23 weeks</td>
<td>714 lines/month</td>
<td></td>
</tr>
<tr>
<td>High-level language</td>
<td>1500 lines</td>
<td>20 weeks</td>
<td>200 lines/month</td>
<td></td>
</tr>
</tbody>
</table>

Function points

- Based on a combination of program characteristics
  - external inputs and outputs
  - user interactions
  - external interfaces
  - files used by the system
- A weight is associated with each of these
- The function point count is computed by multiplying each raw count by the weight and summing all values

Object points

- Object points are an alternative function-related measure to function points when 4GLs or similar languages are used for development
- Object points are NOT the same as object classes
- The number of object points in a program is a weighted estimate of
  - The number of separate screens that are displayed
  - The number of reports that are produced by the system
  - The number of 3GL modules that must be developed to supplement the 4GL code
Object point estimation

- Object points are easier to estimate from a specification than function points as they are simply concerned with screens, reports and 3GL modules.
- They can therefore be estimated at an early point in the development process. At this stage, it is very difficult to estimate the number of lines of code in a system.

Productivity estimates

- Real-time embedded systems, 40-160 LOC/P-month
- Systems programs, 150-400 LOC/P-month
- Commercial applications, 200-800 LOC/P-month
- In object points, productivity has been measured between 4 and 50 object points/month depending on tool support and developer capability.

Factors affecting productivity

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application domain experience</td>
<td>Knowledge of the application domain is essential for effective software development. Engineers who really understand a domain are likely to be the most productive.</td>
</tr>
<tr>
<td>Process quality</td>
<td>The development process used can have a significant effect on productivity. This is covered in Chapter 31.</td>
</tr>
<tr>
<td>Project size</td>
<td>The larger a project, the more time required for team communications. Less time is available for development so individual productivity is reduced.</td>
</tr>
<tr>
<td>Technology support</td>
<td>Good support technology such as CASE tools, supportive configuration management systems, etc. can improve productivity.</td>
</tr>
<tr>
<td>Working environment</td>
<td>As discussed in Chapter 28, a quiet working environment with private work areas contributes to improved productivity.</td>
</tr>
</tbody>
</table>

Quality and productivity

- All metrics based on volume/unit time are flawed because they do not take quality into account.
- Productivity may generally be increased at the cost of quality.
- It is not clear how productivity/quality metrics are related.
- If change is constant then an approach based on counting lines of code is not meaningful.

Estimation techniques

- There is no simple way to make an accurate estimate of the effort required to develop a software system:
  - Initial estimates are based on inadequate information in a user requirements definition.
  - The software may run on unfamiliar computers or use new technology.
  - The people in the project may be unknown.
- Project cost estimates may be self-fulfilling:
  - The estimate defines the budget and the product is adjusted to meet the budget.

Estimation techniques

- Algorithmic cost modelling
- Expert judgement
- Estimation by analogy
- Parkinson’s Law
- Pricing to win
Algorithmic code modelling
- A formulaic approach based on historical cost information and which is generally based on the size of the software
- Discussed later in this chapter

Expert judgement
- One or more experts in both software development and the application domain use their experience to predict software costs. Process iterates until some consensus is reached.
- Advantages: Relatively cheap estimation method. Can be accurate if experts have direct experience of similar systems
- Disadvantages: Very inaccurate if there are no experts!

Estimation by analogy
- The cost of a project is computed by comparing the project to a similar project in the same application domain
- Advantages: Accurate if project data available
- Disadvantages: Impossible if no comparable project has been tackled. Needs systematically maintained cost database

Parkinson's Law
- The project costs whatever resources are available
- Advantages: No overspend
- Disadvantages: System is usually unfinished

Pricing to win
- The project costs whatever the customer has to spend on it
- Advantages: You get the contract
- Disadvantages: The probability that the customer gets the system he or she wants is small. Costs do not accurately reflect the work required

Top-down and bottom-up estimation
- Any of these approaches may be used top-down or bottom-up
- Top-down
  - Start at the system level and assess the overall system functionality and how this is delivered through sub-systems
- Bottom-up
  - Start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate
Top-down estimation

- Usable without knowledge of the system architecture and the components that might be part of the system
- Takes into account costs such as integration, configuration management and documentation
- Can underestimate the cost of solving difficult low-level technical problems

Bottom-up estimation

- Usable when the architecture of the system is known and components identified
- Accurate method if the system has been designed in detail
- May underestimate costs of system level activities such as integration and documentation

Estimation methods

- Each method has strengths and weaknesses
- Estimation should be based on several methods
- If these do not return approximately the same result, there is insufficient information available
- Some action should be taken to find out more in order to make more accurate estimates
- Pricing to win is sometimes the only applicable method

Experience-based estimates

- Estimating is primarily experience-based
- However, new methods and technologies may make estimating based on experience inaccurate
  - Object oriented rather than function-oriented development
  - Client-server systems rather than mainframe systems
  - Off the shelf components
  - Component-based software engineering
  - CASE tools and program generators

Pricing to win

- This approach may seem unethical and unbusinesslike
- However, when detailed information is lacking it may be the only appropriate strategy
- The project cost is agreed on the basis of an outline proposal and the development is constrained by that cost
- A detailed specification may be negotiated or an evolutionary approach used for system development

Algorithmic cost modelling

- Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers
  - \( \text{Effort} = A \times \text{Size}^B \times M \)
  - \( A \) is an organisation-dependent constant, \( B \) reflects the disproportionate effort for large projects and \( M \) is a multiplier reflecting product, process and people attributes
- Most commonly used product attribute for cost estimation is code size
- Most models are basically similar but with different values for \( A, B \) and \( M \)
Estimation accuracy

- The size of a software system can only be known accurately when it is finished.
- Several factors influence the final size:
  - Use of COTS and components
  - Programming language
  - Distribution of system
- As the development process progresses, the size estimate becomes more accurate.

The COCOMO model

- An empirical model based on project experience.
- Well-documented, ‘independent’ model which is not tied to a specific software vendor.
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2.
- COCOMO 2 takes into account different approaches to software development, reuse, etc.

The COCOMO model

The COCOMO model

The COCOMO model

COCOMO 2 levels

- COCOMO 2 is a 3-level model that allows increasingly detailed estimates to be prepared as development progresses.
- Early prototyping level:
  - Estimates based on object points and a simple formula is used for effort estimation.
- Early design level:
  - Estimates based on function points that are then translated to LOC.
- Post-architecture level:
  - Estimates based on lines of source code.

Early prototyping level

- Supports prototyping projects and projects where there is extensive reuse.
- Based on standard estimates of developer productivity in object points/month.
- Takes CASE tool use into account.
- Formula is:
  - PM = (NOP x (1 - %reuse / 100)) / PROD
  - PM is the effort in person-months; NOP is the number of object points and PROD is the productivity.
Object point productivity

<table>
<thead>
<tr>
<th>Developer’s experience and capability</th>
<th>Very low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very high</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICASE maturity and capability</td>
<td>Very low</td>
<td>Low</td>
<td>Nominal</td>
<td>High</td>
<td>Very high</td>
</tr>
<tr>
<td>PROD (NOP/month)</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>

Early design level

- Estimates can be made after the requirements have been agreed.
- Based on standard formula for algorithmic models
  - PM = \( A \times \text{Size} + M \times PM_m \) where
    - \( M = \text{PERS} \times \text{RCPX} \times \text{RUSE} \times \text{PDIF} \times \text{PREX} \times \text{FCIL} \times \text{SCED} \)
    - \( PM_m = \frac{\text{ASLOC} \times (\text{AT}/100)}{\text{ATPROD}} \)
    - \( A = 2.5 \) in initial calibration, \( \text{Size} \) in KLOC, \( B \) varies from 1.1 to 1.24 depending on novelty of the project, development flexibility, risk management approaches and the process maturity.

Multipliers

- Multipliers reflect the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.
  - RCPX - product reliability and complexity
  - RUSE - the reuse required
  - PDIF - platform difficulty
  - PREX - personnel experience
  - PERS - personnel capability
  - SCED - required schedule
  - FCIL - the team support facilities
- PM reflects the amount of automatically generated code

Post-architecture level

- Uses same formula as early design estimates
- Estimate of size is adjusted to take into account
  - Requirements volatility. Rework required to support change
  - Extent of possible reuse. Reuse is non-linear and has associated costs so this is not a simple reduction in LOC

The exponent term

- This depends on 5 scale factors (see next slide). Their sum/100 is added to 1.01
- Example
  - Precedenteness - new project - 4
  - Development flexibility - no client involvement - Very high - 1
  - Architecture/risk resolution - No risk analysis - V. Low - 5
  - Team cohesion - new team - nominal - 3
  - Process maturity - some control - nominal - 3
- Scale factor is therefore 1.17
Exponent scale factors

<table>
<thead>
<tr>
<th>Scale factor</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precededness</td>
<td>Reflects the previous experience of the organisation with this type of project. Very low means no previous experience. Extra high means the organisation is completely familiar with this application domain.</td>
</tr>
<tr>
<td>Development flexibility</td>
<td>Reflects the degree of flexibility in the development process. Very low means a prescribed process is used; Extra high means that the client only sets general goals.</td>
</tr>
<tr>
<td>Architecture/risk resolution</td>
<td>Reflects the extent of risk analysis carried out. Very low means little analysis, Extra high means a complete thorough risk analysis.</td>
</tr>
<tr>
<td>Team cohesion</td>
<td>Reflects how well the development team know each other and work together. Very low means very difficult interactions. Extra high means an integrated and effective team with no communication problems.</td>
</tr>
<tr>
<td>Process maturity</td>
<td>Reflects the process maturity of the organisation. The computation of this value depends on the CMM Maturity Questionnaire but an estimate can be achieved by subtracting the CMM process maturity level from 5.</td>
</tr>
</tbody>
</table>

Multipliers

- Product attributes
  - concerned with required characteristics of the software product being developed
- Computer attributes
  - constraints imposed on the software by the hardware platform
- Personnel attributes
  - multipliers that take the experience and capabilities of the people working on the project into account.
- Project attributes
  - concerned with the particular characteristics of the software development project

Project cost drivers

<table>
<thead>
<tr>
<th>Product attributes</th>
<th>DATA</th>
<th>Project cost drivers</th>
<th>Effects of cost drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUIRED system reliability</td>
<td>Size of database used</td>
<td>System size (including factors for reuse and requirements volatility)</td>
<td>Exponent value</td>
</tr>
<tr>
<td>CPLX Complexity of system modules</td>
<td>ROSE Required percentage of remixable components</td>
<td>Initial COCOMO estimate without cost drivers</td>
<td>1.17</td>
</tr>
<tr>
<td>DOCU Extent of documentation required</td>
<td>STOR Memory constraints</td>
<td>128, 800 DSH</td>
<td>730 person-months</td>
</tr>
<tr>
<td>Computer attributes</td>
<td>TIME Execution time constraints</td>
<td>Volatility of development platform</td>
<td>Embedded spacecraft system</td>
</tr>
<tr>
<td></td>
<td>VOLATILITY</td>
<td></td>
<td>Must be reliable</td>
</tr>
<tr>
<td></td>
<td>VOLATILITY</td>
<td></td>
<td>Must minimise weight (number of chips)</td>
</tr>
<tr>
<td></td>
<td>VOLATILITY</td>
<td></td>
<td>Multipliers on reliability and computer constraints &gt; 1</td>
</tr>
<tr>
<td>Personnel attributes</td>
<td>F receptivity of project analysts</td>
<td>PCAP Programme capability analysis</td>
<td>Reliability</td>
</tr>
<tr>
<td></td>
<td>PCAP Programme capability analysis</td>
<td>AEXP Analyse experience in project domain</td>
<td>Complexity</td>
</tr>
<tr>
<td></td>
<td>PEXP Programme experience in project domain</td>
<td>LTEX Language and tool experience</td>
<td>Memory constraint</td>
</tr>
<tr>
<td></td>
<td>PERSONAL</td>
<td></td>
<td>Tool use</td>
</tr>
<tr>
<td></td>
<td>TOOR Use of software tools</td>
<td>SITE Extent of multi-site working and quality of site communications</td>
<td>Schedule</td>
</tr>
<tr>
<td></td>
<td>SCED Development schedule completeness</td>
<td></td>
<td>Adjusted COCOMO estimate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2300 person-months</td>
</tr>
</tbody>
</table>

Project planning

- Algorithmic cost models provide a basis for project planning as they allow alternative strategies to be compared
- Embedded spacecraft system
  - Must be reliable
  - Must minimise weight (number of chips)
  - Multipliers on reliability and computer constraints > 1
- Cost components
  - Target hardware
  - Development platform
  - Effort required

Management options

- A. Use existing hardware, development system and development team
- B. Processor and memory upgrades
- C. Memory upgrades only
- D. More experienced staff
- E. New development system
- F. Staff with hardware expertise
- Hardware cost increase
- Experience decrease
- Experience increase
### Management options costs

<table>
<thead>
<tr>
<th>Option</th>
<th>REEIY</th>
<th>STOR</th>
<th>TIME</th>
<th>TOOLS</th>
<th>LTEX</th>
<th>Total effort</th>
<th>Software cost</th>
<th>Hardware cost</th>
<th>Total cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.39</td>
<td>1.06</td>
<td>1.11</td>
<td>0.96</td>
<td>1</td>
<td>63</td>
<td>943793</td>
<td>190000</td>
<td>1043793</td>
</tr>
<tr>
<td>B</td>
<td>1.39</td>
<td>1</td>
<td>1</td>
<td>1.12</td>
<td>1.22</td>
<td>88</td>
<td>1311550</td>
<td>120000</td>
<td>1431550</td>
</tr>
<tr>
<td>C</td>
<td>1.39</td>
<td>1.11</td>
<td>0.86</td>
<td>1</td>
<td>64</td>
<td>845623</td>
<td>1575000</td>
<td>1000000</td>
<td>1075000</td>
</tr>
<tr>
<td>D</td>
<td>1.39</td>
<td>1.06</td>
<td>1.11</td>
<td>0.96</td>
<td>1</td>
<td>66</td>
<td>760099</td>
<td>190000</td>
<td>950099</td>
</tr>
<tr>
<td>E</td>
<td>1.39</td>
<td>1</td>
<td>1</td>
<td>0.72</td>
<td>1.22</td>
<td>56</td>
<td>644625</td>
<td>220000</td>
<td>1044625</td>
</tr>
<tr>
<td>F</td>
<td>1.39</td>
<td>1</td>
<td>1</td>
<td>1.12</td>
<td>0.84</td>
<td>57</td>
<td>851168</td>
<td>120000</td>
<td>1071168</td>
</tr>
</tbody>
</table>

### Option choice

- Option D (use more experienced staff) appears to be the best alternative
  - However, it has a high associated risk as experienced staff may be difficult to find
- Option C (upgrade memory) has a lower cost saving but very low risk
- Overall, the model reveals the importance of staff experience in software development

### Project duration and staffing

- As well as effort estimation, managers must estimate the calendar time required to complete a project and when staff will be required
- Calendar time can be estimated using a COCOMO 2 formula
  - \( T_{DEV} = 3 \times (PM)^{0.33+0.2*(B-1.01)} \)
  - \( PM \) is the effort computation and \( B \) is the exponent computed as discussed above (\( B \) is 1 for the early prototyping model).
  - This computation predicts the nominal schedule for the project
- The time required is independent of the number of people working on the project

### Staffing requirements

- Staff required can’t be computed by diving the development time by the required schedule
- The number of people working on a project varies depending on the phase of the project
- The more people who work on the project, the more total effort is usually required
- A very rapid build-up of people often correlates with schedule slippage

### Key points

- Factors affecting productivity include individual aptitude, domain experience, the development project, the project size, tool support and the working environment
- Different techniques of cost estimation should be used when estimating costs
- Software may be priced to gain a contract and the functionality adjusted to the price

- Algorithmic cost estimation is difficult because of the need to estimate attributes of the finished product
- The COCOMO model takes project, product, personnel and hardware attributes into account when predicting effort required
- Algorithmic cost models support quantitative option analysis
- The time to complete a project is not proportional to the number of people working on the project