Project management

Software project management

- Activities ensuring that:
  - software is delivered on time and on schedule
  - and in accordance with the requirements of the organisations developing and procuring the software.
- Project management is needed because software development is always subject to budget and schedule constraints that are set by the organisation developing the software.

Software management distinctions

- The product is intangible.
- The product is uniquely flexible.
- Software engineering is not recognized as an engineering discipline with the sane status as mechanical, electrical engineering, etc.
- The software development process is not standardised.
- Many software projects are 'one-off' projects.

Management activities

- Proposal writing.
- Project planning and scheduling.
- Project costing.
- Project monitoring and reviews.
- Personnel selection and evaluation.
- Report writing and presentations.

Management commonalities

- These activities are not peculiar to software management.
- Many techniques of engineering project management are equally applicable to software project management.
- Technically complex engineering systems tend to suffer from the same problems as software systems.

Project staffing

- May not be possible to appoint the ideal people to work on a project
  - Project budget may not allow for the use of highly-paid staff;
  - Staff with the appropriate experience may not be available;
  - An organisation may wish to develop employee skills on a software project.
- Managers have to work within these constraints especially when there are shortages of trained staff.

Managing Groups

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.
- Poor people management is an important contributor to project failure.

People management factors

- Consistency
  - Team members should all be treated in a comparable way without favourites or discrimination.
- Respect
  - Different team members have different skills and these differences should be respected.
- Inclusion
  - Involve all team members and make sure that people's views are considered.
- Honesty
  - You should always be honest about what is going well and what is going badly in a project.

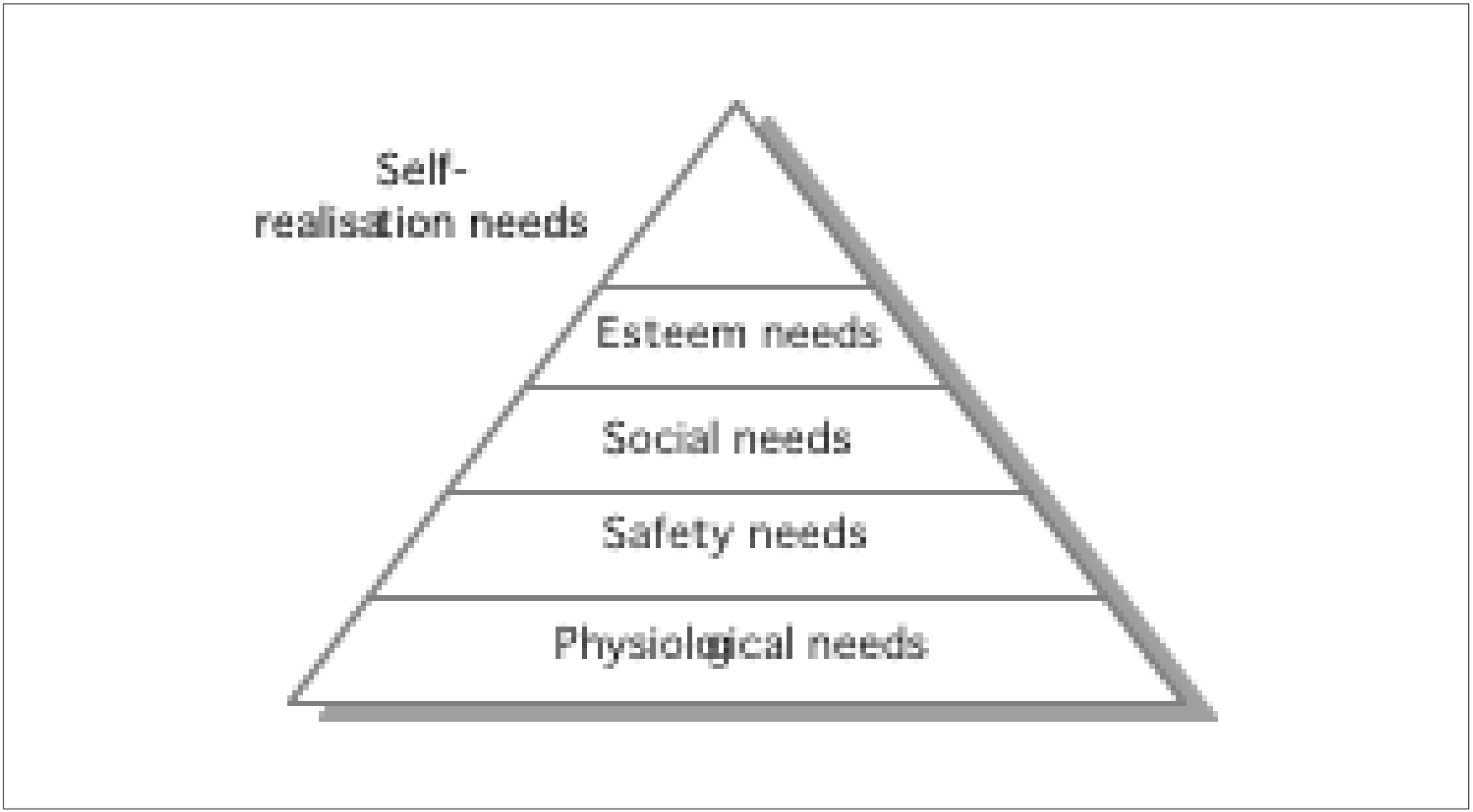
Selecting staff

- An important project management task is team selection.
- Information on selection comes from:
  - Information provided by the candidates.
  - Information gained by interviewing and talking with candidates.
  - Recommendations and comments from other people who know or who have worked with the candidates.

Motivating people

- 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



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 of motivation in practice.
- 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.
- The 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.

Managing groups

- 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.

Factors influencing group working

- Group composition.
- Group cohesiveness.
- Group communications.
- Group organisation.

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.
- This can be difficult to achieve software engineers are often task-oriented.
- Interaction-oriented people are very important as they can detect and defuse tensions that arise.

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.

Group cohesiveness

- In a cohesive group, members consider the group to be more important than any individual in it.
- The 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 organisation

- Small software engineering groups are usually organised informally without a rigid structure.
- For large projects, there may be a hierarchical structure where different groups are responsible for different sub-projects.

Informal groups

- 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 an informal, 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 been successful in some settings.
- However, it suffers from a number of problems
  - Talented designers and programmers are hard to find. Without exceptional 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;
  - There is a high project risk as the project will fail if both the chief and deputy programmer are unavailable.
  - The organisational structures and grades in a company may be unable to accommodate this type of group.

Working environments

- The 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

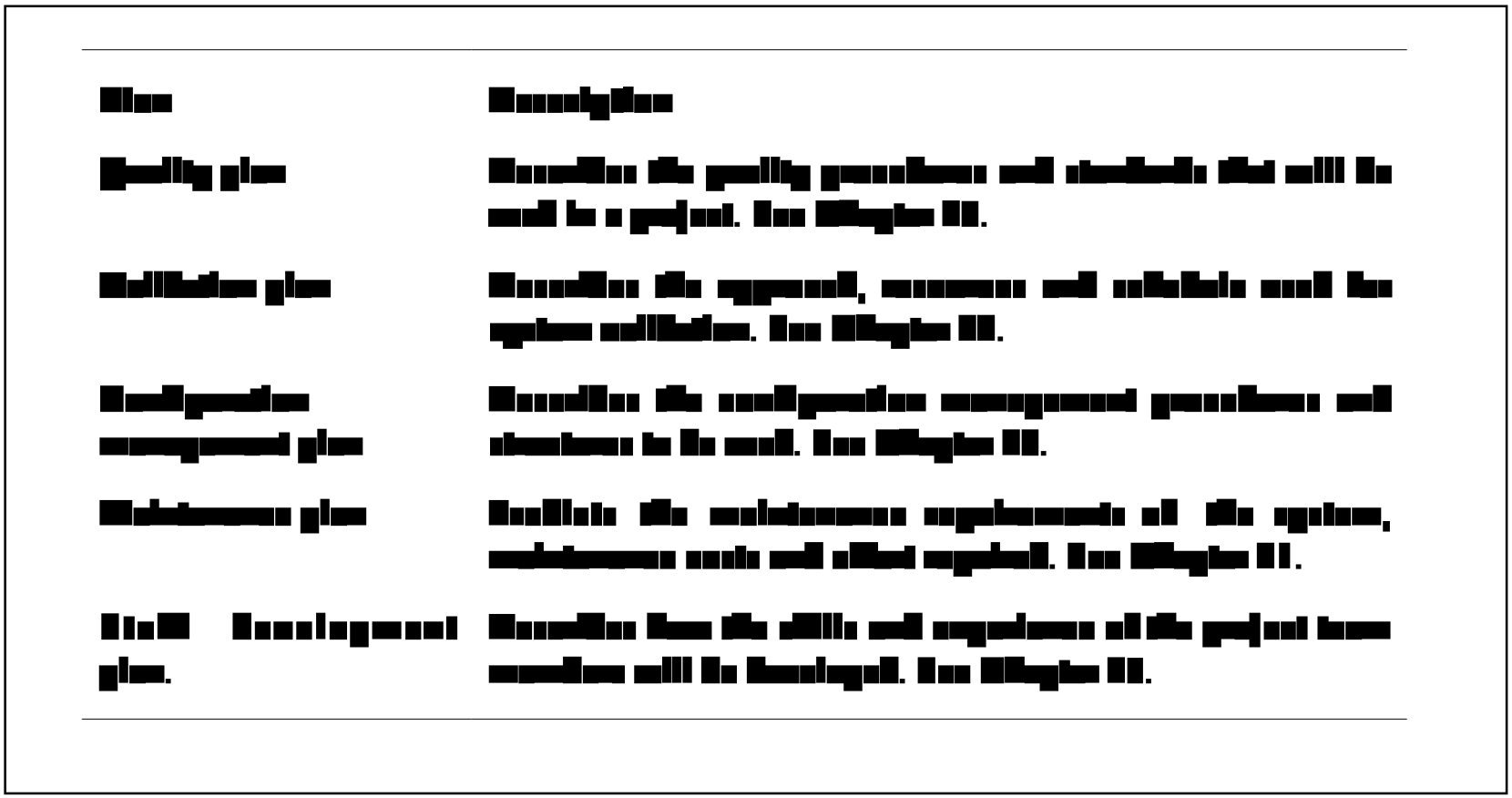
- Privacy - each engineer requires an area for uninterrupted work.
- Outside awareness - people prefer to work in natural light.
- Personalization - individuals adopt different working practices and like to organize their environment in different ways.

…other issues
Project Management

Project planning

- Probably the most time-consuming project management activity.
- Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
- Various different types of plan may be developed to support the main software project plan that is concerned with schedule and budget.

Types of project plan



The project plan

- The project plan sets out:
  - The resources available to the project;
  - The work breakdown;
  - A schedule for the work.

Project plan structure

- Introduction.
- Project organisation.
- Risk analysis.
- Hardware and software resource requirements.
- Work breakdown.
- Project schedule.
- Monitoring and reporting mechanisms.

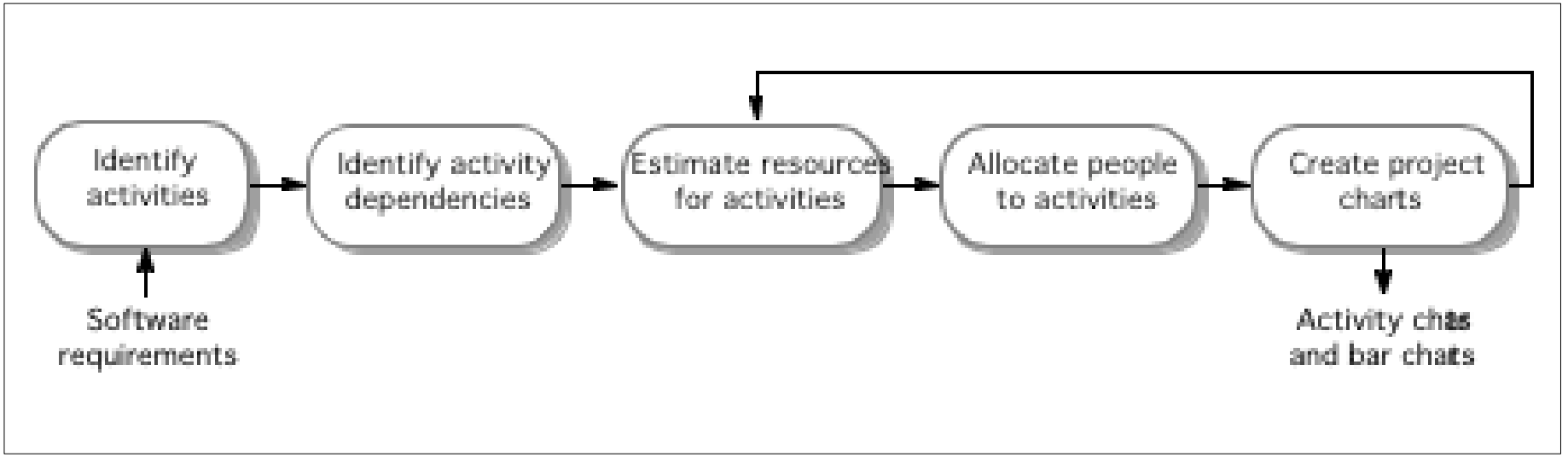
Activity organization

- Activities in a project should be organised to produce tangible outputs for management to judge progress.
- *Milestones* are the end-point of a process activity.
- *Deliverables* are project results delivered to customers.
- The waterfall process allows for the straightforward definition of progress milestones.

Project scheduling

- Split project into tasks and estimate time and resources required to complete each task.
- Organize tasks concurrently to make optimal use of workforce.
- Minimize task dependencies to avoid delays caused by one task waiting for another to complete.
- Dependent on project managers intuition and experience.

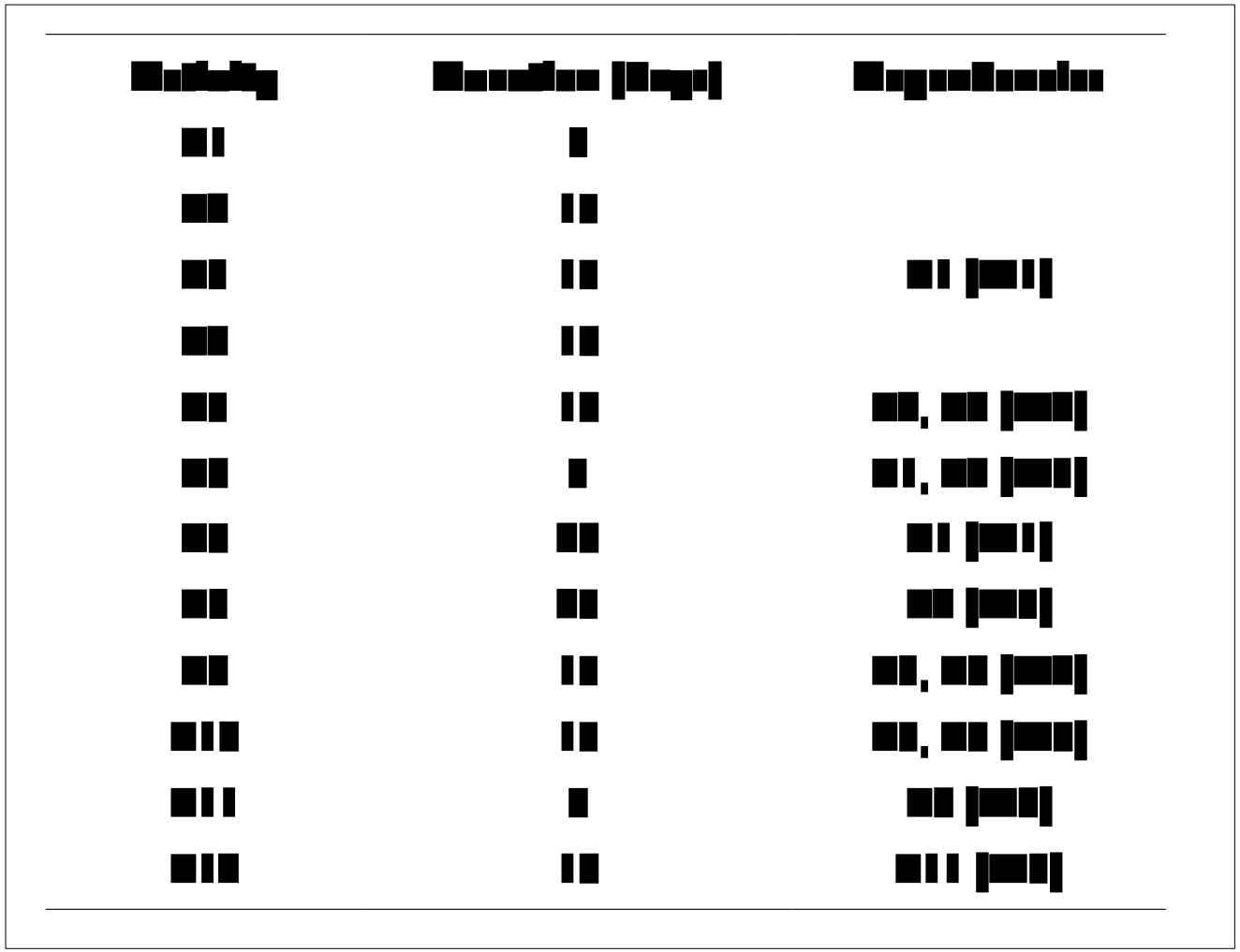
The project scheduling process



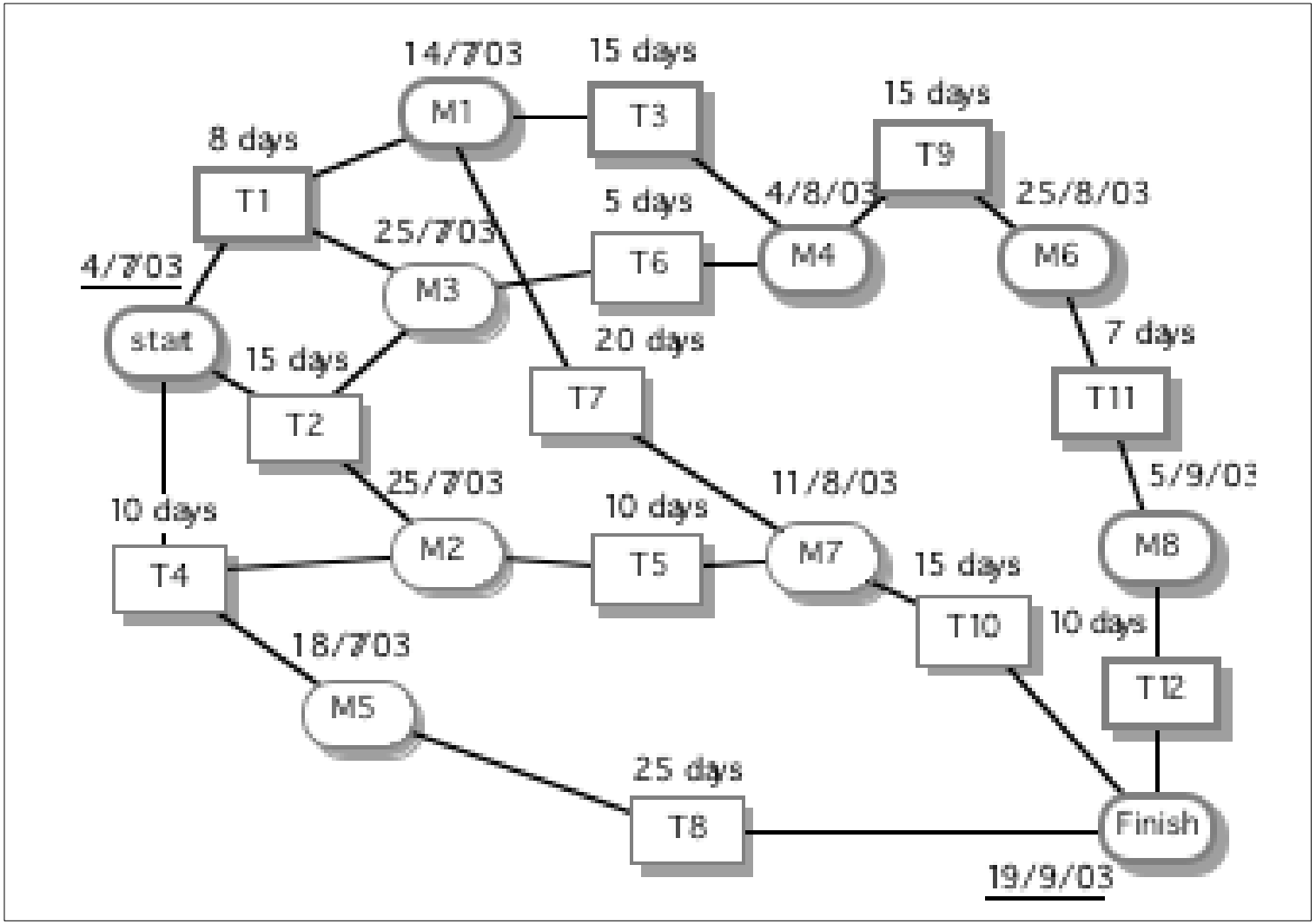
Bar charts and activity networks

- Graphical notations used to illustrate the project schedule.
- Show project breakdown into tasks. Tasks should not be too small. They should take about a week or two.
- Activity charts show task dependencies and the the critical path.
- Bar charts show schedule against calendar time.

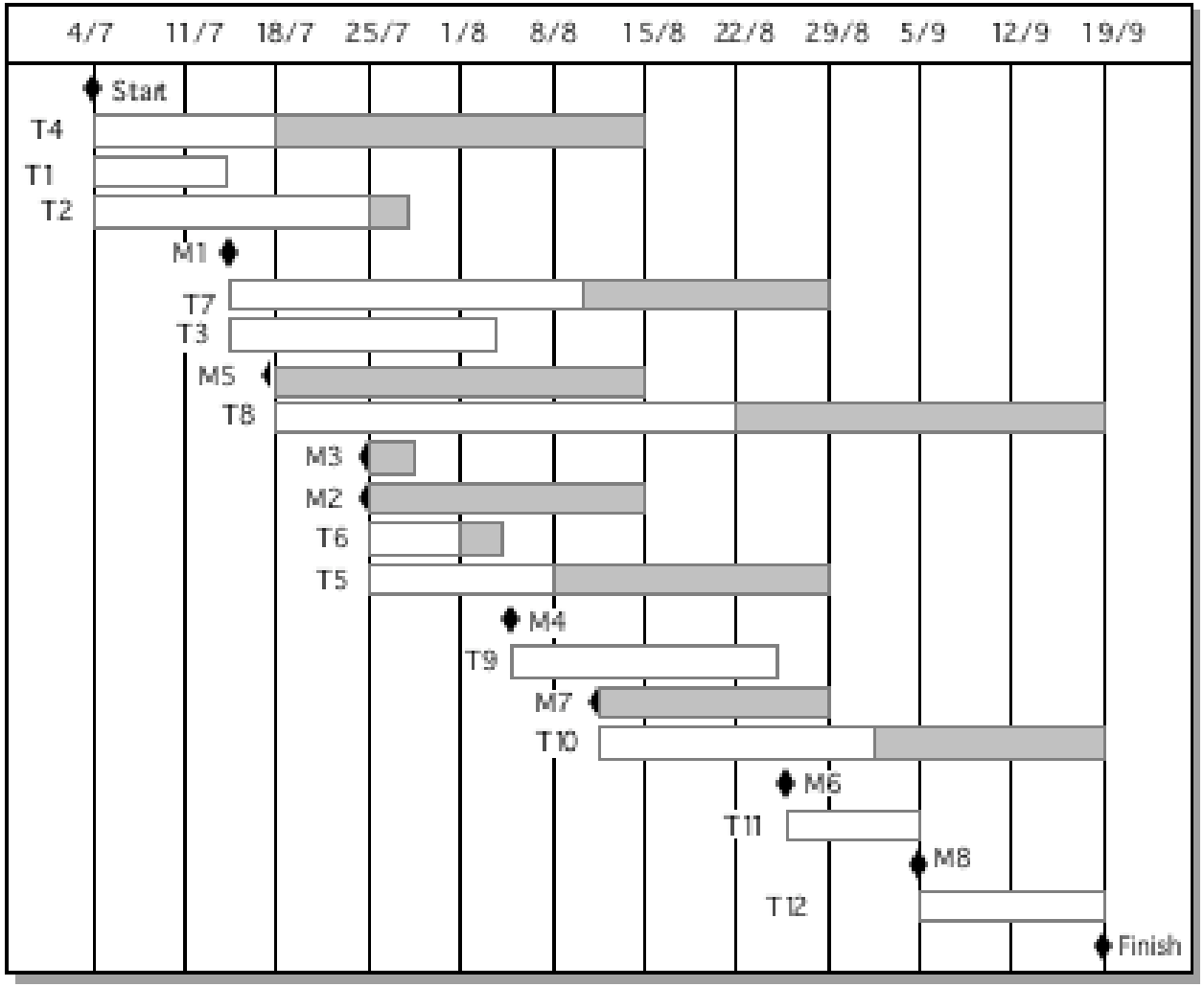
Task durations and dependencies



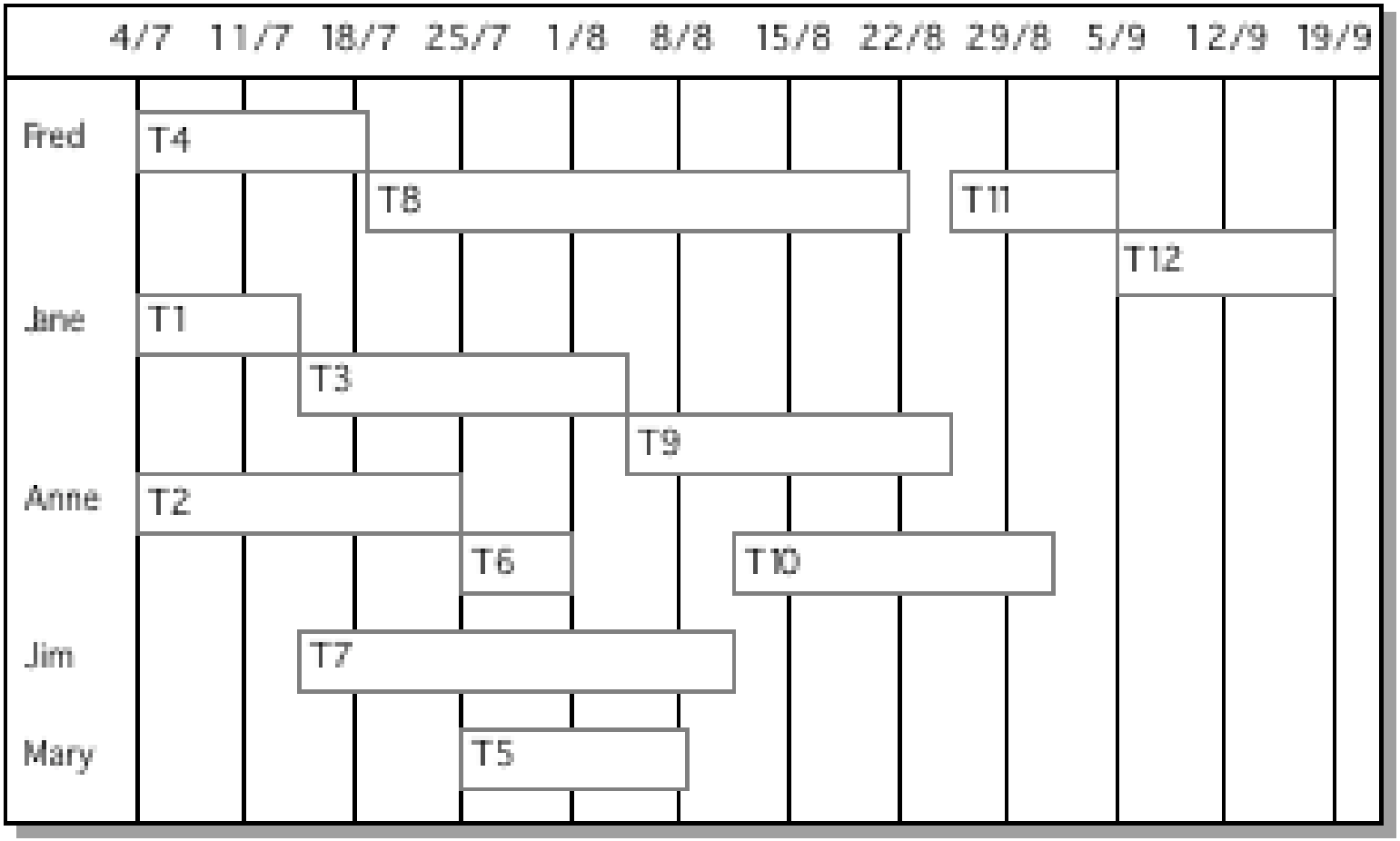
Activity network



Activity timeline



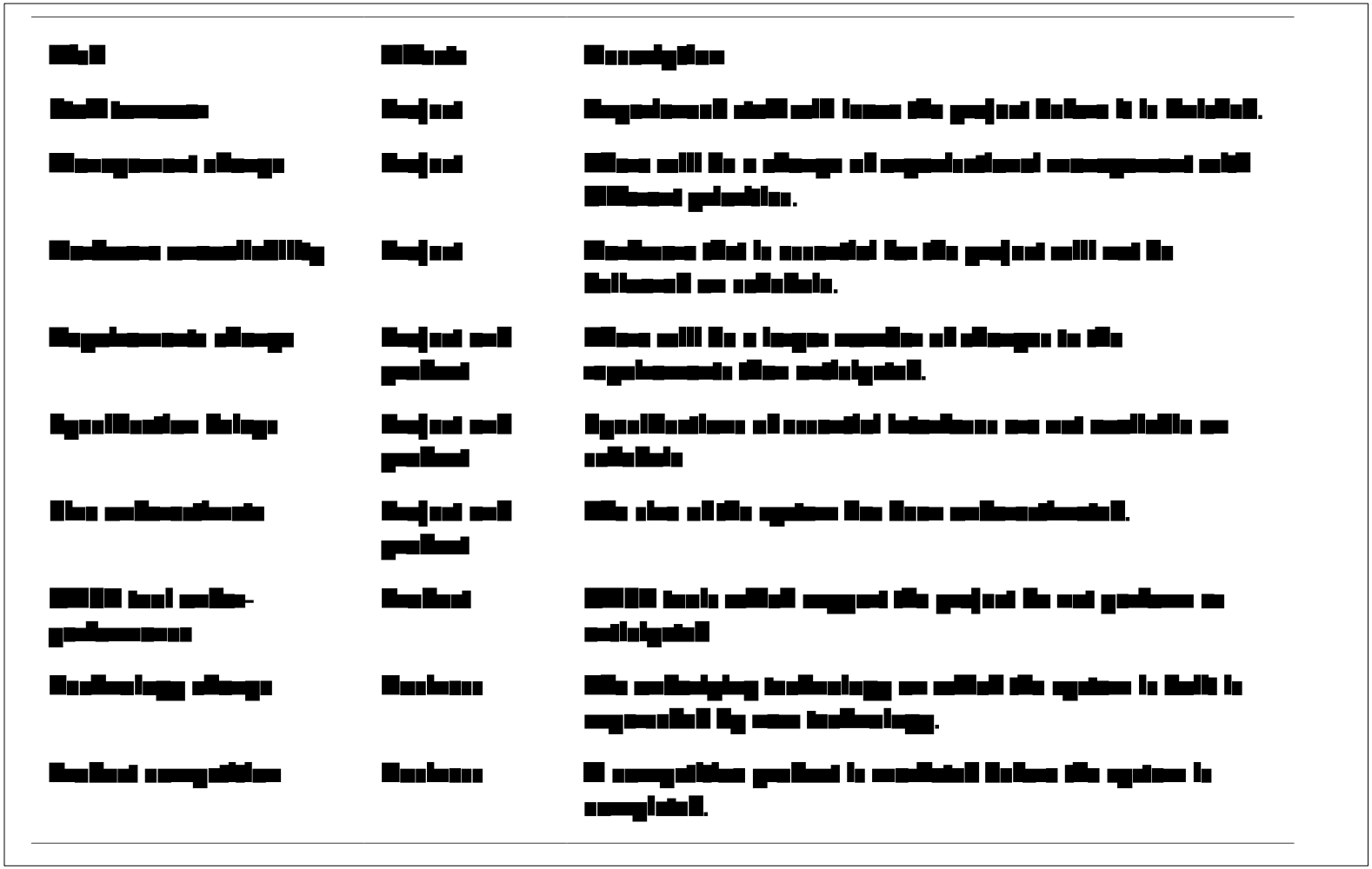
Staff allocation



Risk management

- Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.
- A risk is a probability that some adverse circumstance will occur
  - Project risks affect schedule or resources;
  - Product risks affect the quality or performance of the software being developed;
  - Business risks affect the organisation developing or procuring the software.

Software risks



Software cost estimation

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 are interleaved management activities.

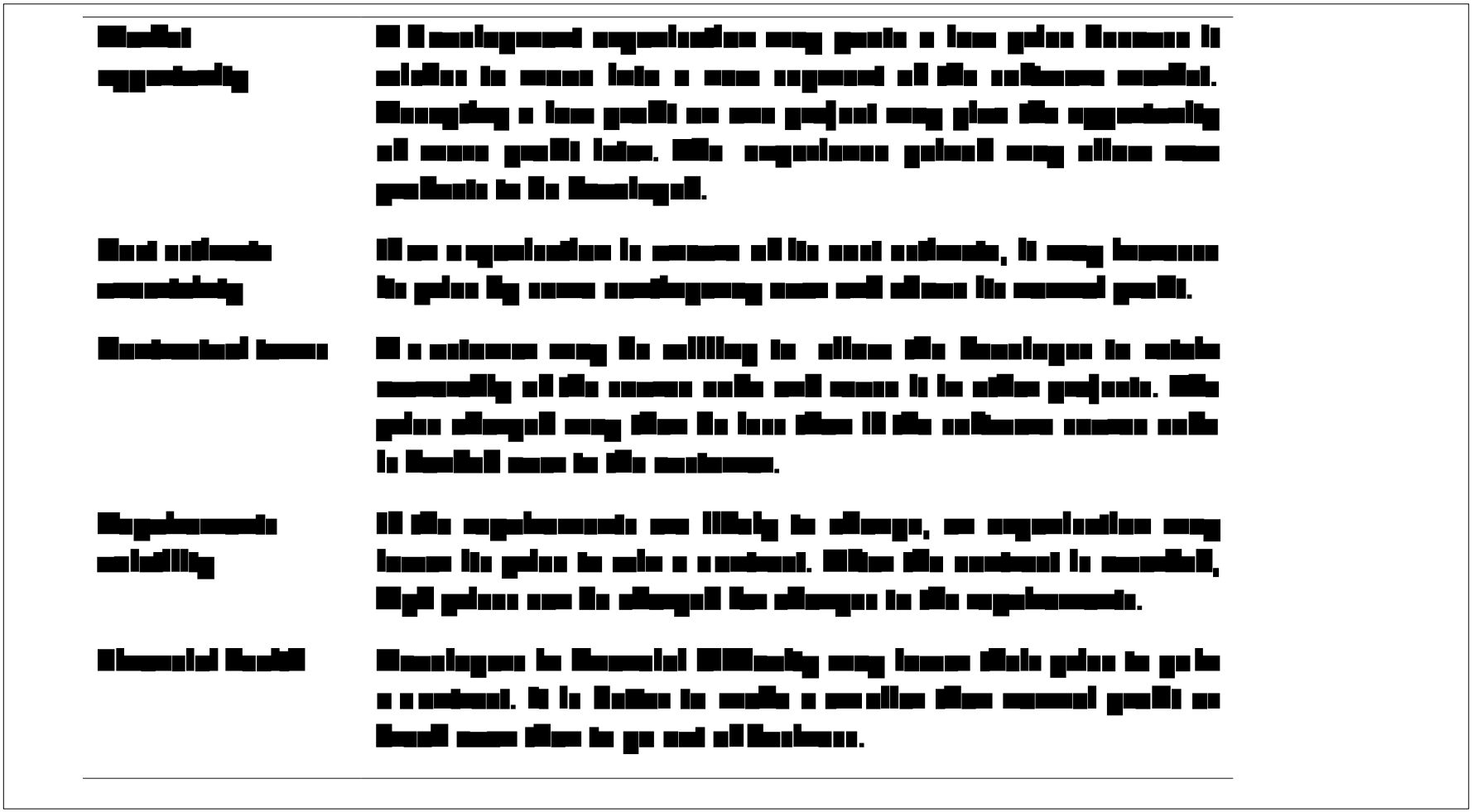
Software cost components

- Hardware and software costs.
- Travel and training costs.
- Effort costs (the dominant factor in most projects)
  - The 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



Software 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
  - (e.g. how many function points).
- Estimating the total number of programmer months that have elapsed.
- Estimating contractor productivity (e.g. documentation team) and incorporating this estimate in overall estimate.

Lines of code (LOC)

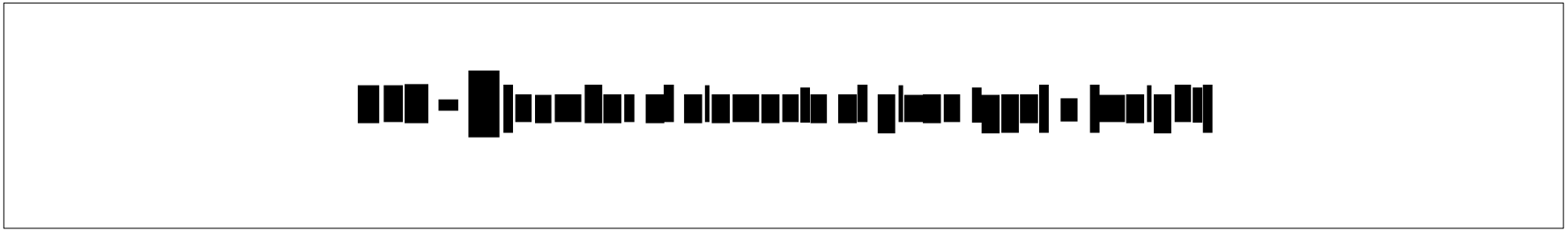
- What's a line of code?
  - The measure was first proposed when programs were typed on cards with one line per card;
  - How does this correspond to statements as in Java which can span several lines or where there can be several statements on one line.
- What programs should be counted as part of the system?
- This model assumes that there is a 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.

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 and the function point count is computed by multiplying each raw count by the weight and summing all values.



Function points

- The function point count is modified by complexity of the project
- FPs can be used to estimate LOC depending on the average number of LOC per FP for a given language
  - LOC = AVC * number of function points;
  - AVC is a language-dependent factor varying from 200-300 for assemble language to 2-40 for a 4GL;
- FPs are very subjective. They depend on the estimator
  - Automatic function-point counting is impossible.

Object points

- Object points are an alternative
  - (also named application points)
- 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 program modules that must be developed to supplement the database code;

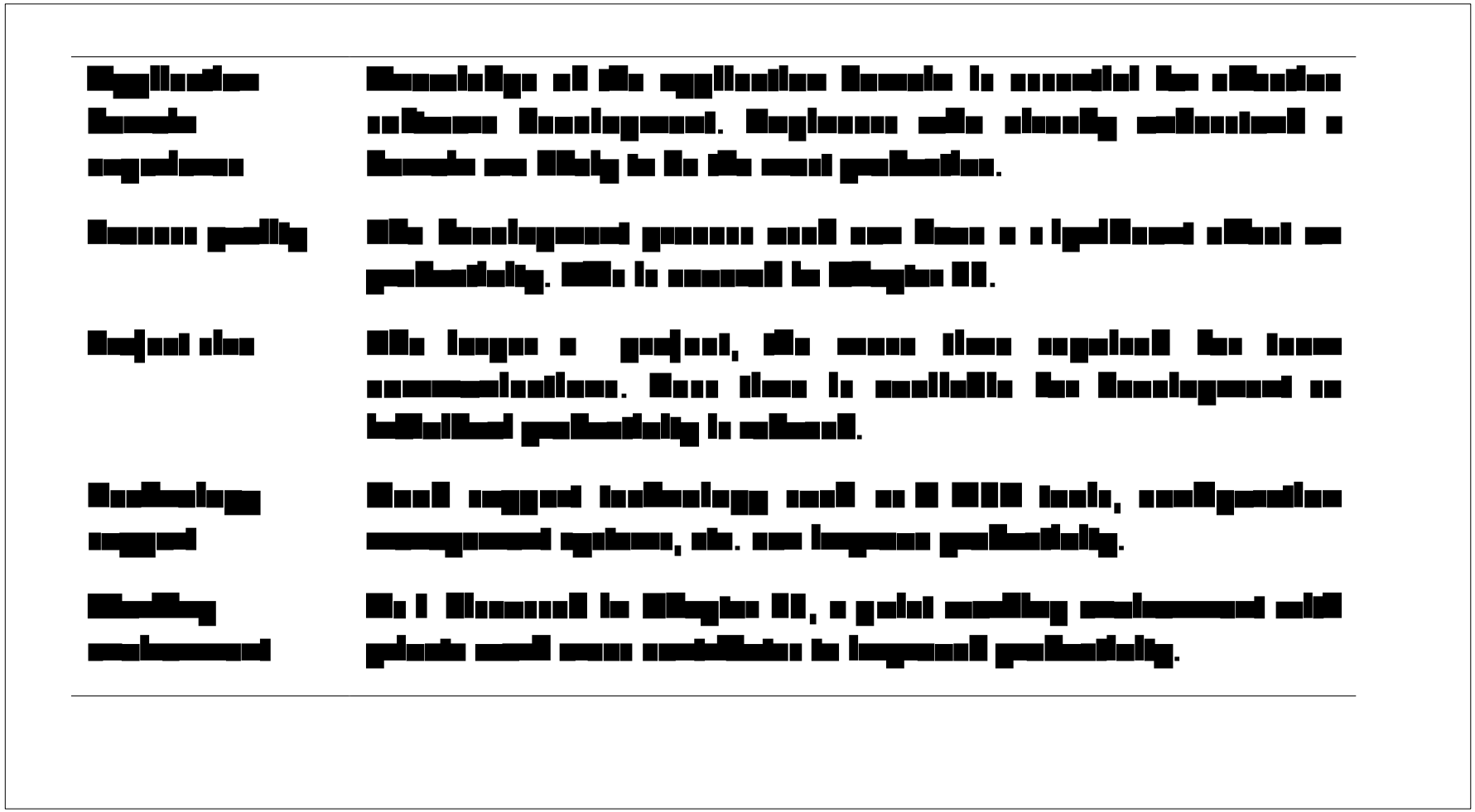
Object point estimation

- Object points are easier to estimate from a specification than function points
  - they are simply concerned with screens, reports and programming language modules.
- They can therefore be estimated at a fairly 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, 200-900 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



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 requirements are constantly changing then an approach based on counting lines of code is not meaningful as the program itself is not static;

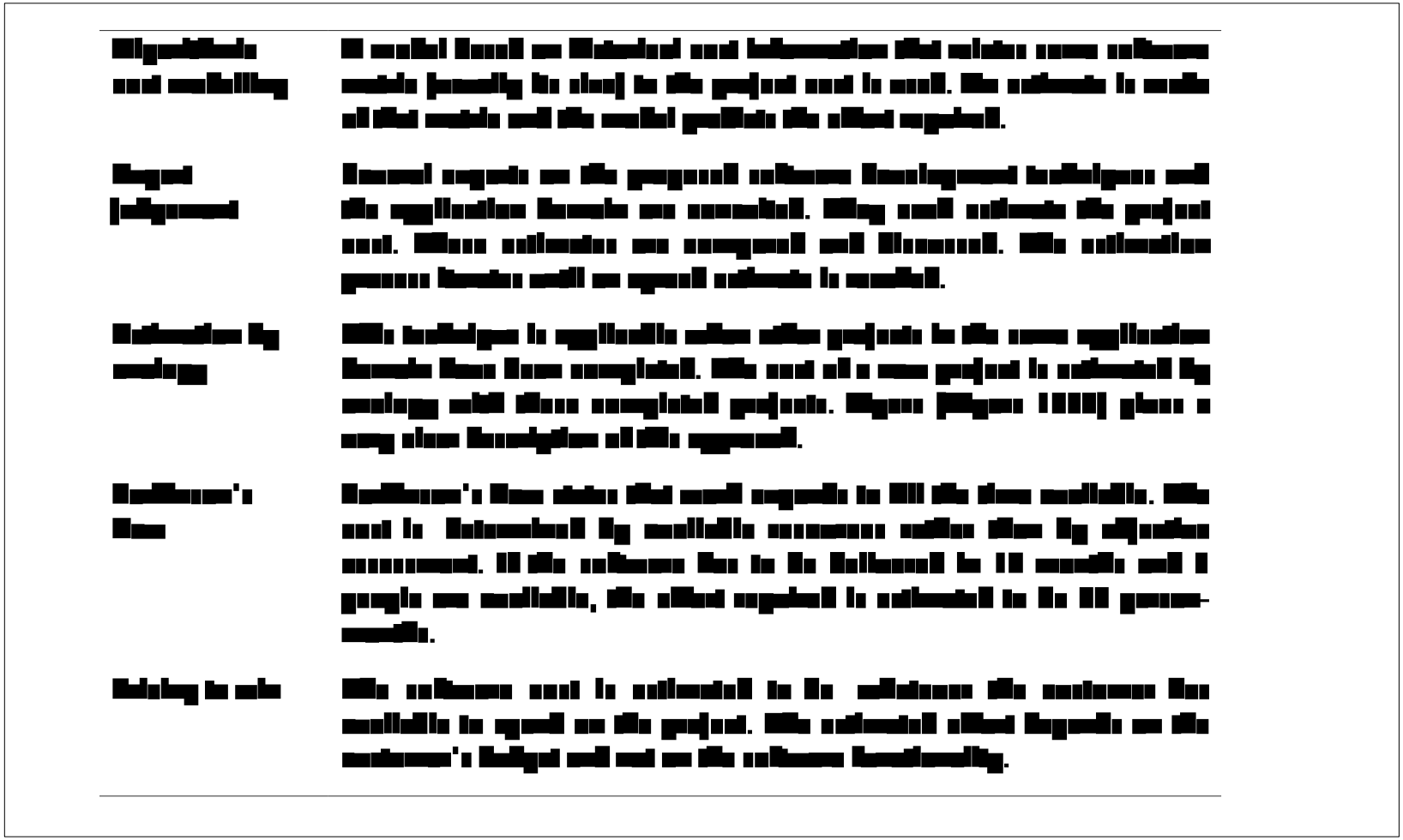
Estimation techniques

- 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.
- Changing technologies may mean that previous estimating experience does not carry over to new systems

Estimation techniques

- Algorithmic cost modelling.
- Expert judgement.
- Estimation by analogy.
- Parkinson's Law.
- Pricing to win.

Estimation techniques



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.

Pricing to win

- This approach may seem unethical and un-businesslike.
- 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.

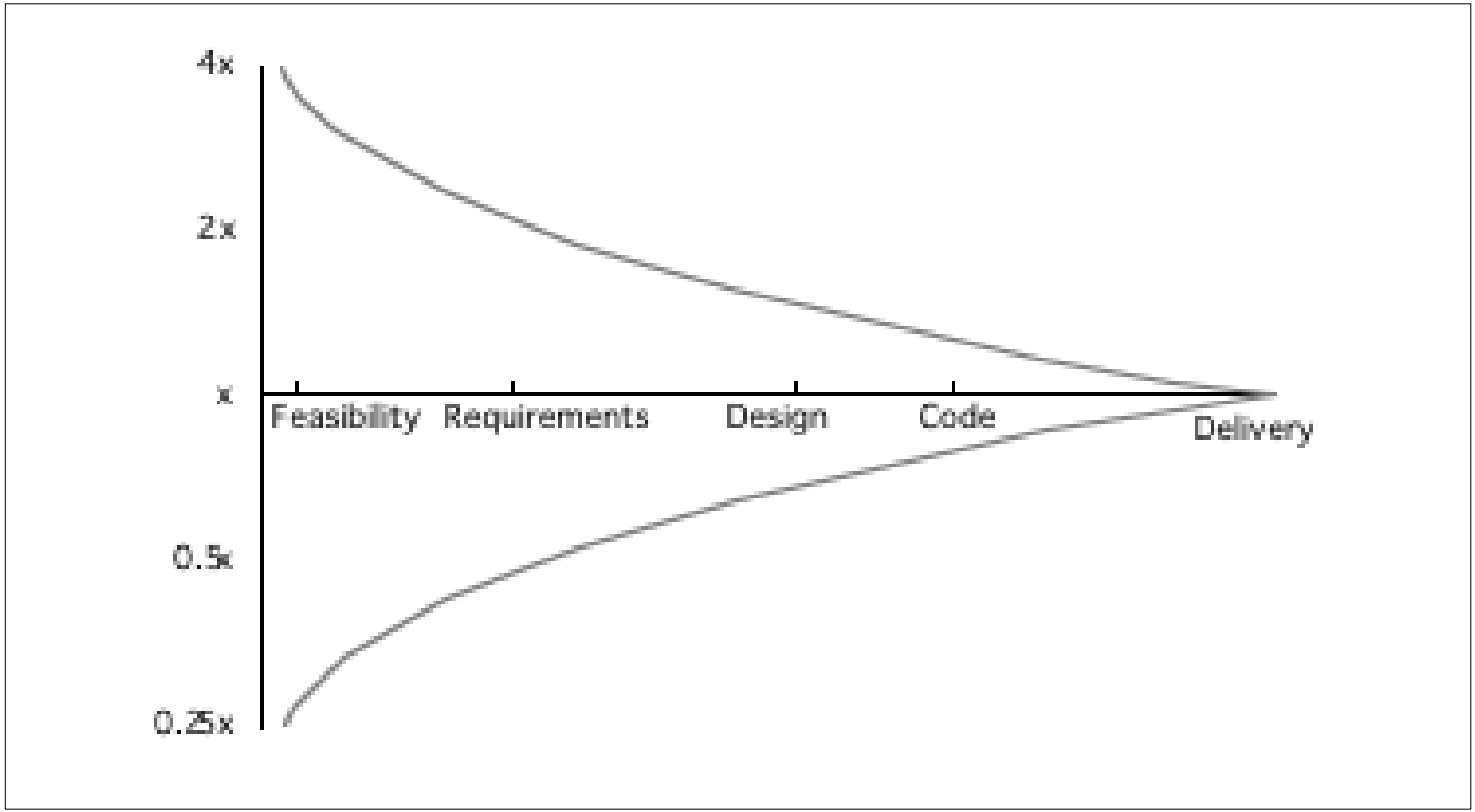
Algorithmic cost modelling

- Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers:
  - Effort = $A \times Size^{B} \times M$
  - 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.
- The most commonly used product attribute for cost estimation is code size.
- Most models are similar but they use different values for A, B and M.

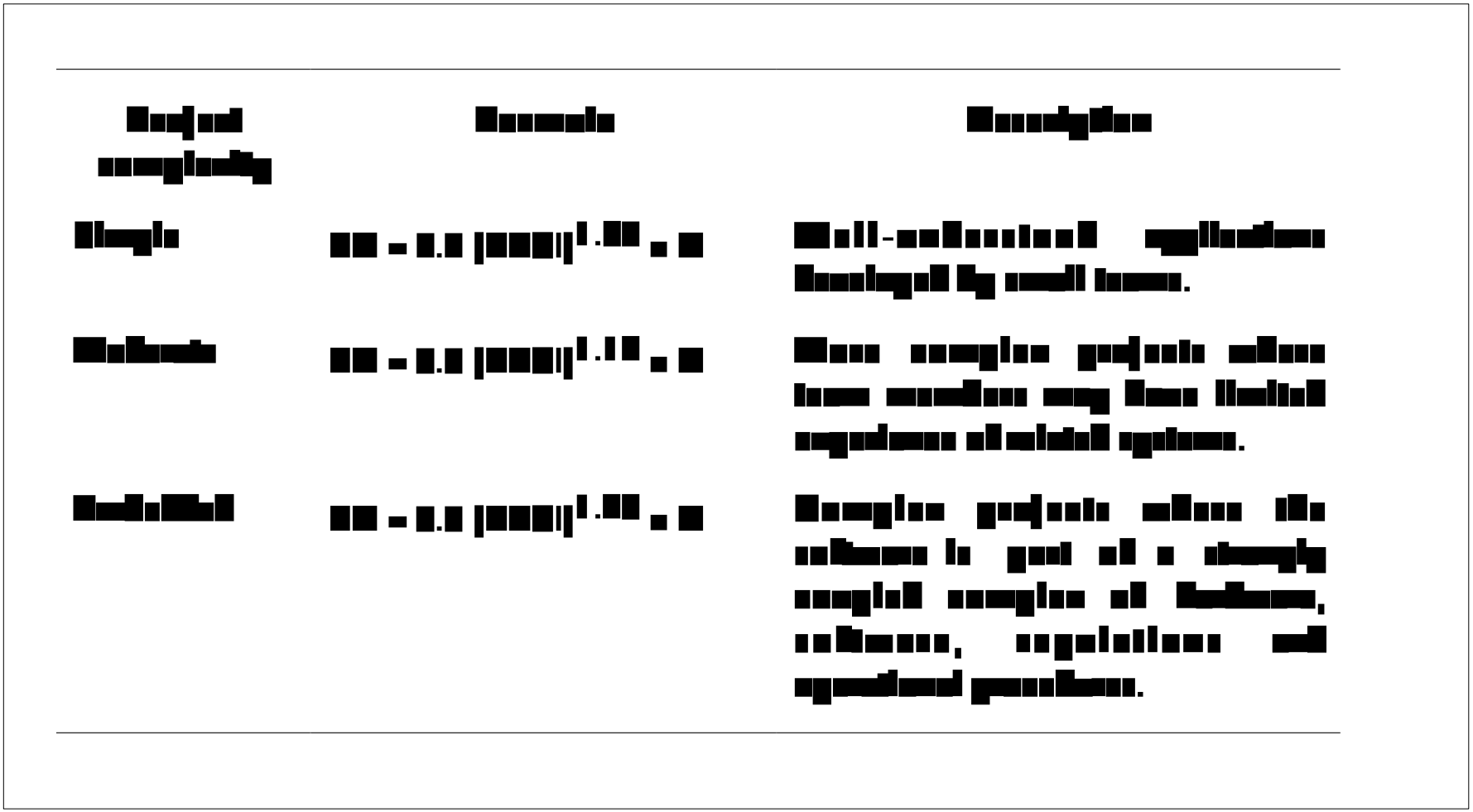
Estimation accuracy

- 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 then the size estimate becomes more accurate.

Estimate uncertainty



COCOMO 81



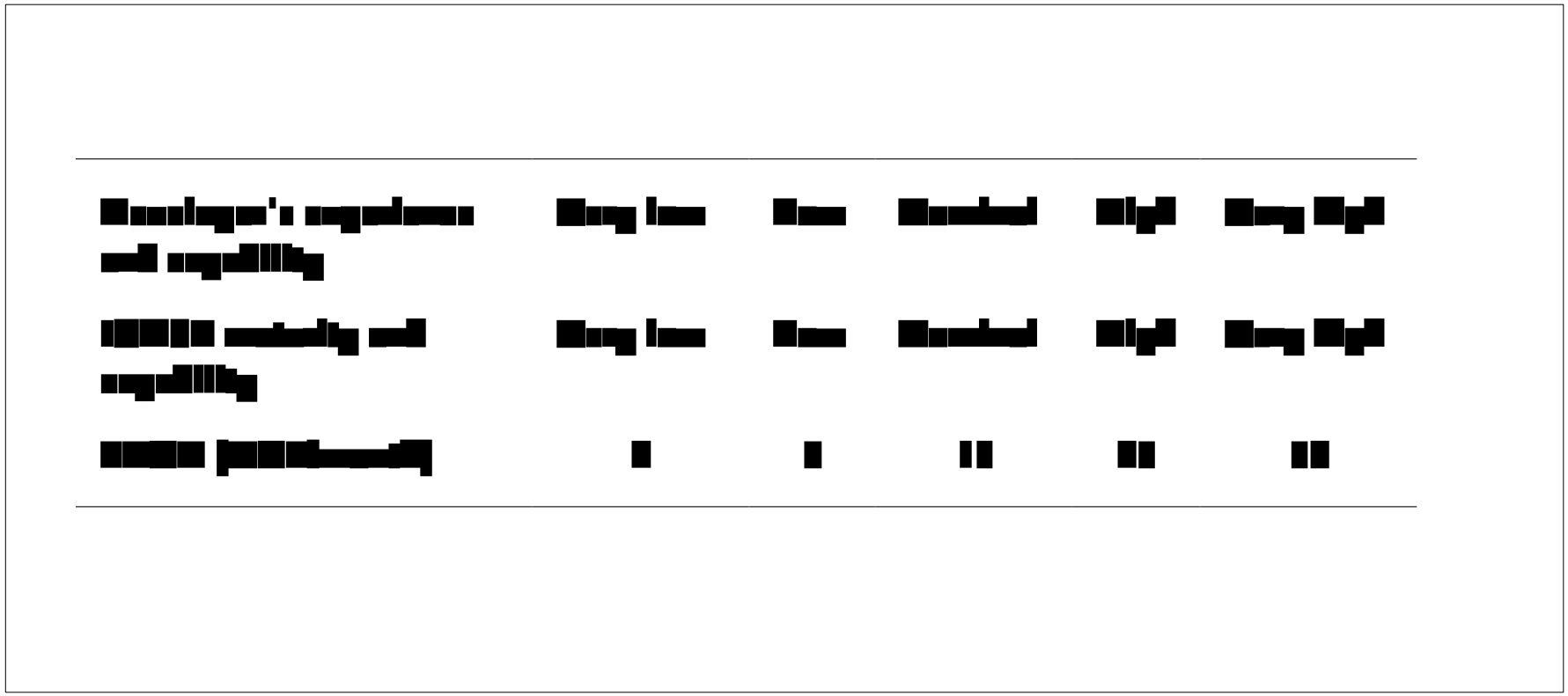
COCOMO 2 models

- COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.
- The sub-models in COCOMO 2 are:
  - Application composition model. Used when software is composed from existing parts.
  - Early design model. Used when requirements are available but design has not yet started.
  - Reuse model. Used to compute the effort of integrating reusable components.
  - Post-architecture model. Used once the system architecture has been designed and more information about the system is available.

Application composition model

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

Object point productivity



Early design model

- Estimates can be made after the requirements have been agreed.
- Based on a standard formula for algorithmic models
  - PM = A × $Size^B$ × M where
  - M = PERS × RCPX × RUSE × PDIF × PREX × FCIL × SCED;
  - A = 2.94 in initial calibration, 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.

The reuse model

- Takes into account black-box code that is reused without change and code that has to be adapted to integrate it with new code.
- There are two versions:
  - Black-box reuse where code is not modified. An effort estimate (PM) is computed.
  - White-box reuse where code is modified. A size estimate equivalent to the number of lines of new source code is computed. This then adjusts the size estimate for new code.

Reuse model estimates 1

- For generated code:
  - PM = (ASLOC * AT/100)/ATPROD
  - ASLOC is the number of lines of generated code
  - AT is the percentage of code automatically generated.
  - ATPROD is the productivity of engineers in integrating this code.

Reuse model estimates 2

- When code has to be understood and integrated:
  - ESLOC = ASLOC * (1-AT/100) * AAM.
  - ASLOC and AT as before.
  - AAM is the adaptation adjustment multiplier computed from the costs of changing the reused code, the costs of understanding how to integrate the code and the costs of reuse decision making.

Post-architecture level

- Uses the same formula as the early design model but with 17 rather than 7 associated multipliers.
- The code size is estimated as:
  - Number of lines of new code to be developed;
  - Estimate of equivalent number of lines of new code computed using the reuse model;
  - An estimate of the number of lines of code that have to be modified according to requirements changes.

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
  - $TDEV = 3 \times (PM)^{(0.33+0.2*(B-1.01))}$
  - 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.