Software cost estimation

Lecture – 9

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

Topics covered

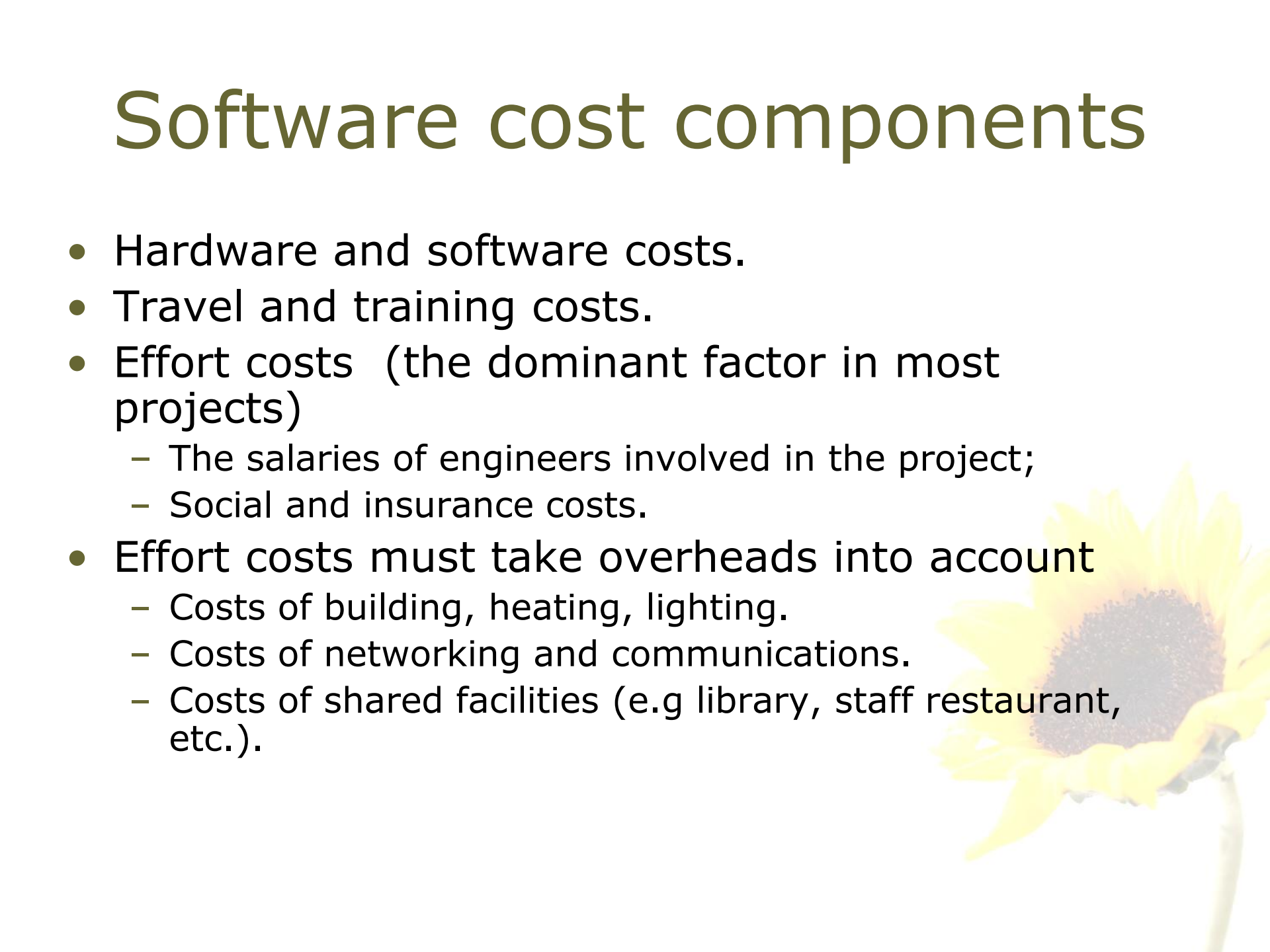
- Software productivity
- Estimation techniques
- Algorithmic cost modelling
- Project duration and staffing

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 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

Market opportunity	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.
Cost estimate uncertainty	If an organisation is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit.
Contractual terms	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.
Requirements volatility	If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements.
Financial health	Developers in financial difficulty may lower their price to gain a contract. It is better to make a smaller than normal profit or break even than to go out of business.

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

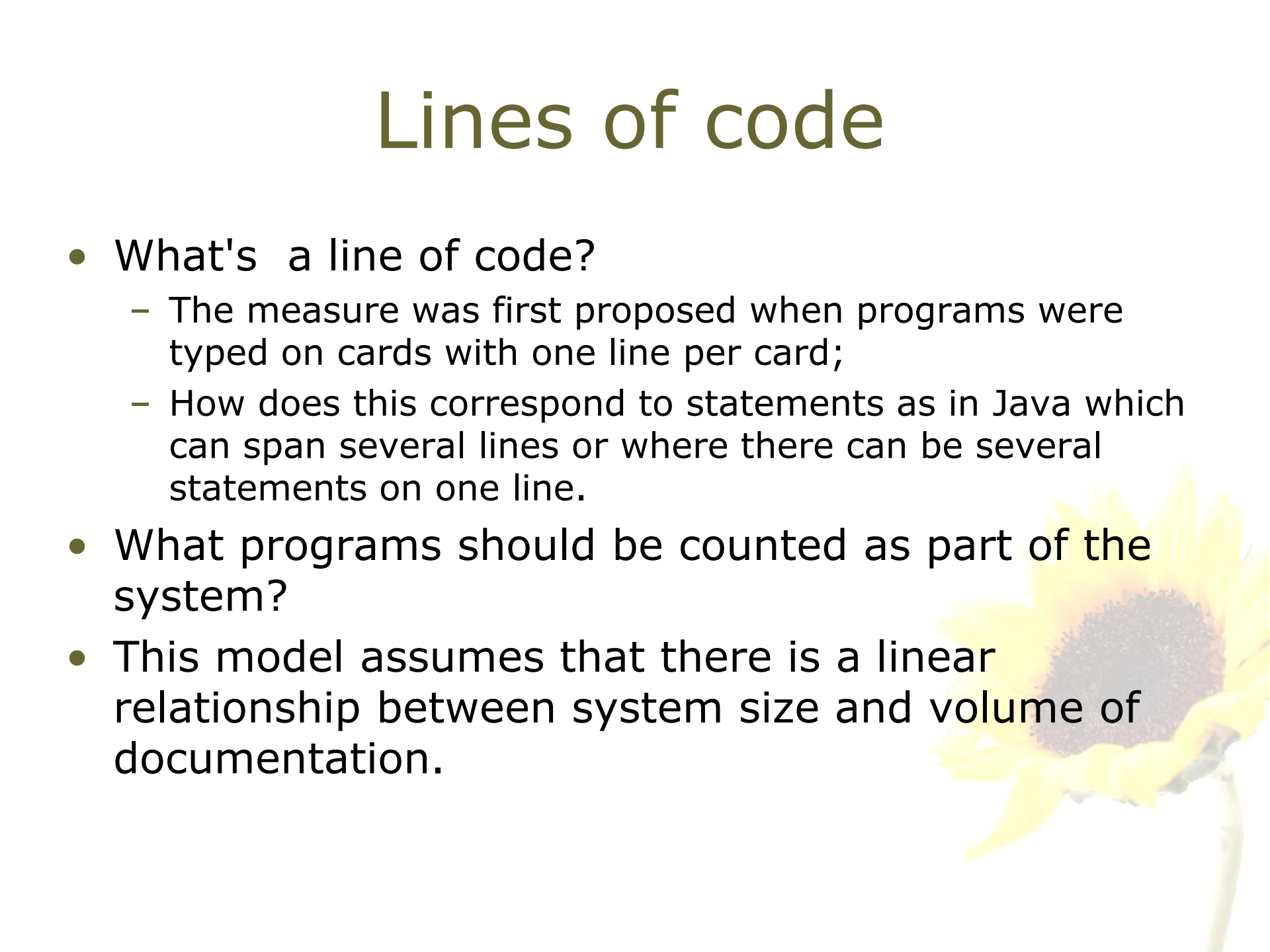
- **Size related measures** based on some output from the software process. This may be lines of delivered source code, object code instructions, etc.
- **Function-related measures** based on an estimate of the functionality of the delivered software. Function-points are the best known of this type of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

- 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.

System development times

	Analysis	Design	Coding	Testing	Documentation
Assembly code	3 weeks	5 weeks	8 weeks	10 weeks	2 weeks
High-level language	3 weeks	5 weeks	4 weeks	6 weeks	2 weeks

	Size	Effort	Productivity
Assembly code	5000 lines	28 weeks	714 lines/month
High-level language	1500 lines	20 weeks	300 lines/month

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.

$$\text{UFC} = \sum(\text{number of elements of given type}) \times (\text{weight})$$

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 (alternatively named application 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 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 as 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

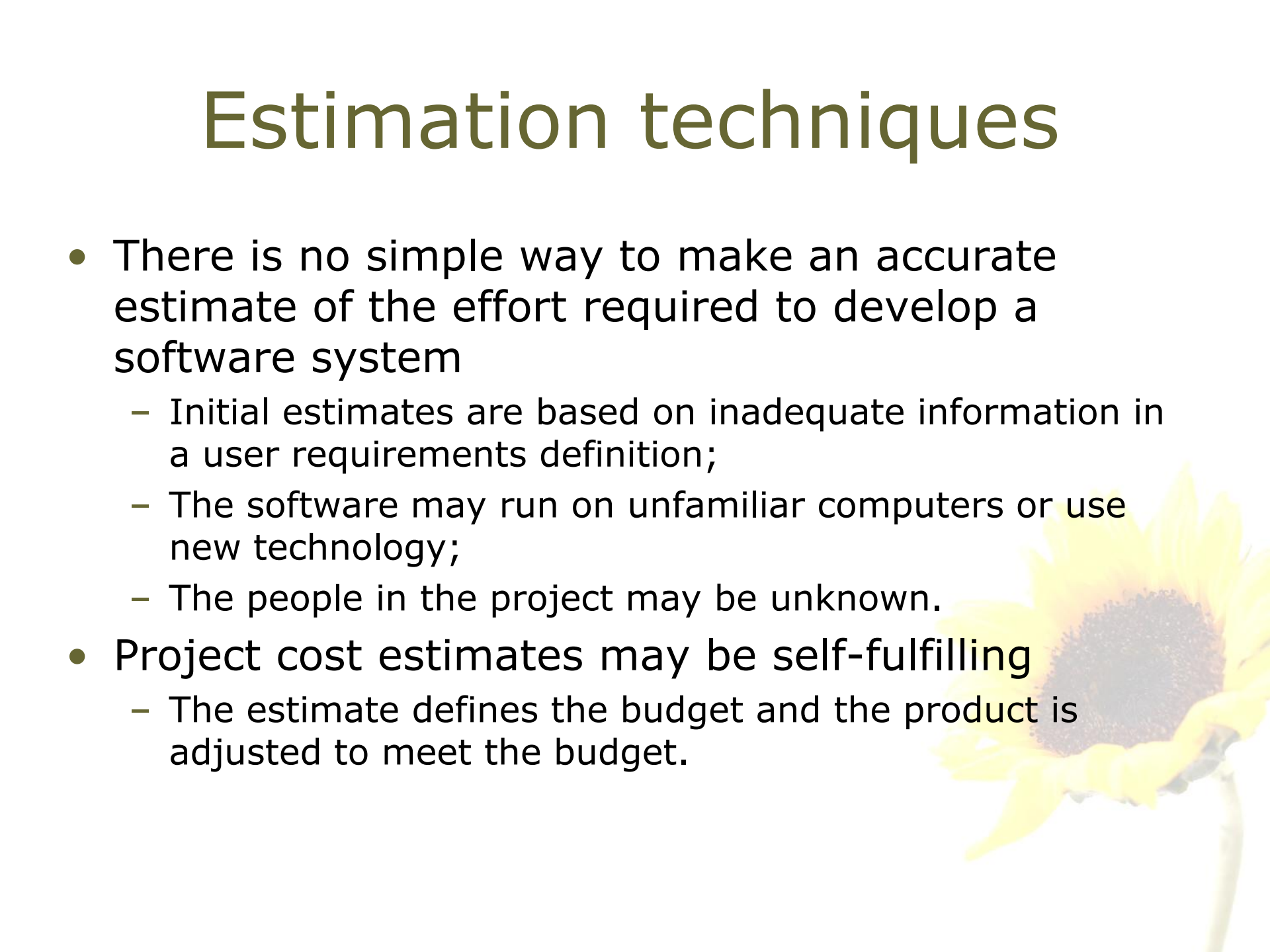
Application domain experience	Knowledge of the application domain is essential for effective software development. Engineers who already understand a domain are likely to be the most productive.
Process quality	The development process used can have a significant effect on productivity. This is covered in Chapter 28.
Project size	The larger a project, the more time required for team communications. Less time is available for development so individual productivity is reduced.
Technology support	Good support technology such as CASE tools, configuration management systems, etc. can improve productivity.
Working environment	As I discussed in Chapter 25, a quiet working environment with private work areas contributes to improved 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

- Changing technologies may mean that previous estimating experience does not carry over to new systems
  - Distributed object systems rather than mainframe systems;
  - Use of web services;
  - Use of ERP or database-centred systems;
  - Use of off-the-shelf software;
  - Development for and with reuse;
  - Development using scripting languages;
  - The use of CASE tools and program generators.

Estimation techniques

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

Estimation techniques

Algorithmic cost modelling	A model based on historical cost information that relates some software metric (usually its size) to the project cost is used. An estimate is made of that metric and the model predicts the effort required.
Expert judgement	Several experts on the proposed software development techniques and the application domain are consulted. They each estimate the project cost. These estimates are compared and discussed. The estimation process iterates until an agreed estimate is reached.
Estimation by analogy	This technique is applicable when other projects in the same application domain have been completed. The cost of a new project is estimated by analogy with these completed projects. Myers (Myers 1989) gives a very clear description of this approach.
ParkinsonÕs Law	ParkinsonÕsLaw states that work expands to fill the time available. The cost is determined by available resources rather than by objective assessment. If the software has to be delivered in 12 months and 5 people are available, the effort required is estimated to be 60 person-months.
Pricing to win	The software cost is estimated to be whatever the customer has available to spend on the project. The estimated effort depends on the customerÕs budget and not on the software functionality.

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

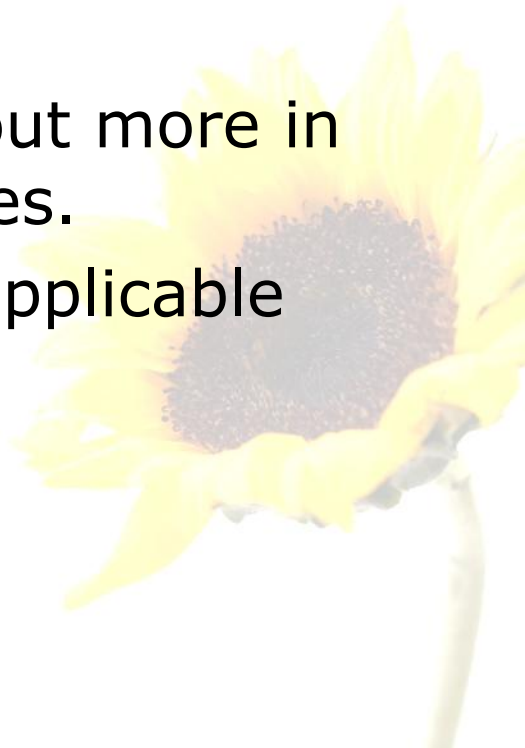
- Usable without knowledge of the system architecture and the components that might be part of the system.
- Takes into account costs such as integration, configuration management and documentation.
- Can underestimate the cost of solving difficult low-level technical problems.

Bottom-up estimation

- Usable when the architecture of the system is known and components identified.
- This can be an accurate method if the system has been designed in detail.
- It may underestimate the costs of system level activities such as integration and documentation.

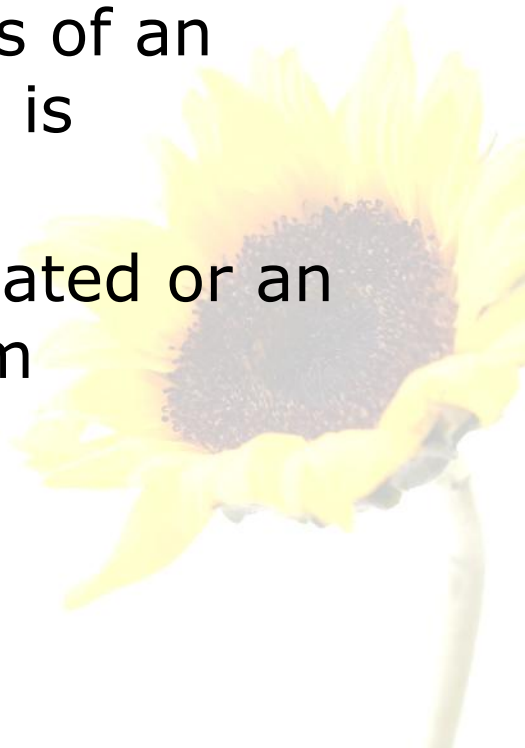
Estimation methods

- Each method has strengths and weaknesses.
- Estimation should be based on several methods.
- If these do not return approximately the same result, then you have insufficient information available to make an estimate.
- 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.



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.



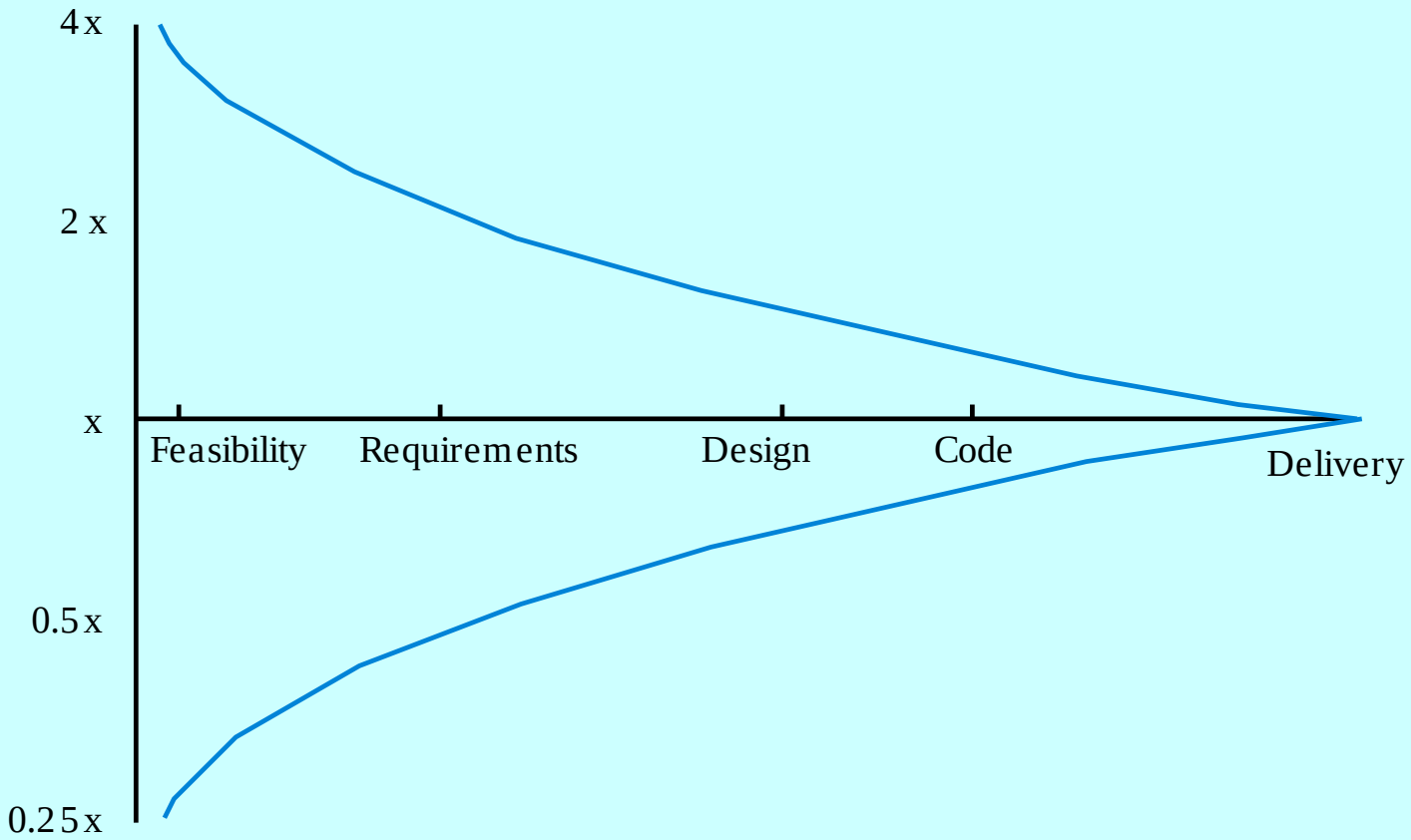
Algorithmic cost modelling

- Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers:
  - $\text{Effort} = A \times \text{Size}^B \times M$
  - 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.
- 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



Assignment

- Explain Software Cost Estimation in detail