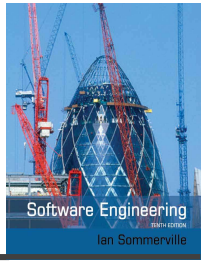
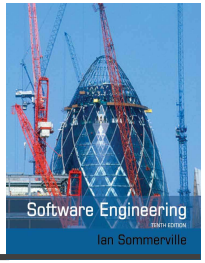


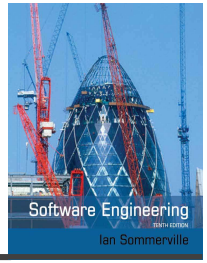
Project planning

Topics covered



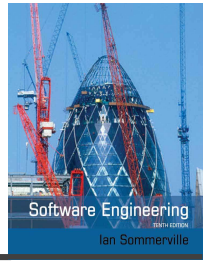
- Software pricing
- Plan-driven development
- Project scheduling
- Agile planning
- Estimation techniques
- COCOMO cost modeling

Project planning

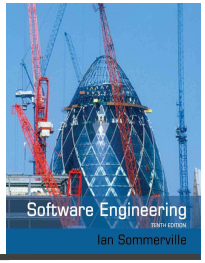


- Project planning involves breaking down the work into parts and assign these to project team members, anticipate problems that might arise and prepare tentative solutions to those problems.
- The project plan is used to communicate how the work will be done to the project team and customers, and to help assess progress on the project.

Planning stages

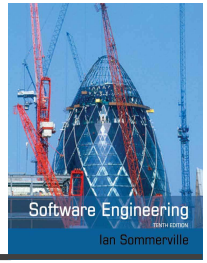


- At the proposal stage, when you are bidding for a contract to develop or provide a software system.
- During the project startup phase, for project resources allocation, for breaking down the project into increments, etc.
- Periodically throughout the project, when you modify your plan in the light of experience gained.



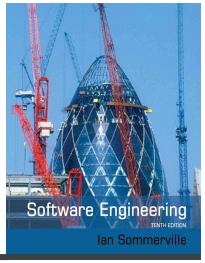
Software pricing

Software pricing



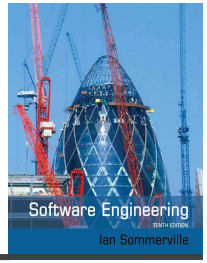
- Estimates are made to discover the cost, to the developer, of producing a software system.
  - You take into account, hardware, software, travel, training and effort costs.
- There is not a simple relationship between the development cost and the price charged to the customer.
- Broader organisational, economic, political and business considerations influence the price charged.

Factors affecting software pricing



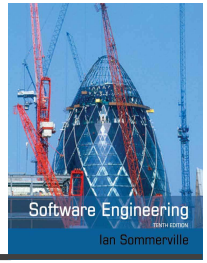
Factor	Description
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.
Cost estimate uncertainty	If an organization is unsure of its cost estimate, it may increase its price by a contingency over and above its normal profit.
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. Cash flow is more important than profit in difficult economic times.

Factors affecting software pricing



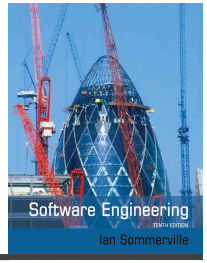
Factor	Description
Market opportunity	A development organization may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the organization the opportunity to make a greater profit later. The experience gained may also help it develop new products.
Requirements volatility	If the requirements are likely to change, an organization may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements.

Pricing strategies



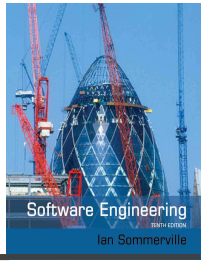
- Under pricing
  - A company may underprice a system in order to gain a contract that allows them to retain staff for future opportunities
  - A company may underprice a system to gain access to a new market area
- Increased pricing
  - The price may be increased when a buyer wishes a fixed-price contract and so the seller increases the price to allow for unexpected risks

Pricing to win



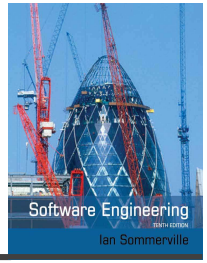
- The software is priced according to what the software developer believes the buyer is willing to pay
- If this is less that the development costs, the software functionality may be reduced accordingly with a view to extra functionality being added in a later release
- Additional costs may be added as the requirements change and these may be priced at a higher level to make up the shortfall in the original price

Software cost components



- Hardware and software costs
- Travel and training costs
- Personnel costs (the dominant factor in most projects)
  - salaries of people involved in the project
  - benefits and insurance costs
- Must also take project overhead into account
  - costs of building, heating, lighting
  - costs of networking and communications
  - costs of shared facilities (e.g library, staff restaurant, etc.)

Costs for a Proposed System



Estimated Costs for Client-Server System Alternative

DEVELOPMENT COSTS

Personnel:

2	Systems Analysts (400 hours/ea $50.00/hr)	$40,000
4	Programmer/Analysts (250 hours/ea $35.00/hr)	$35,000
1	GUI Designer (200 hours/ea $40.00/hr)	$8,000
1	Telecommunications Specialist (50 hours/ea $50.00/hr)	$2,500
1	System Architect (100 hours/ea $50.00/hr)	$5,000
1	Database Specialist (15 hours/ea $45.00/hr)	$675
1	System Librarian (250 hours/ea $15.00/hr)	$3,750

Expenses:

4	Smalltalk training registration ($3,500.00/student)	$14,000

New Hardware & Software:

1	Development Server	$18,700
1	Server software (operating system, misc.)	$1,500
1	DBMS server software	$7,500
7	DBMS client software ($950.00 per client)	$6,650

Total Development Costs: $143,275

PROJECTED ANNUAL OPERATING COSTS

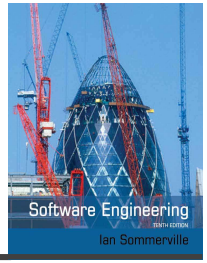
Personnel:

2	Programmer/Analysts (125 hours/ea $35.00/hr)	$8,750
1	System Librarian (20 hours/ea $15.00/hr)	$300

Expenses:

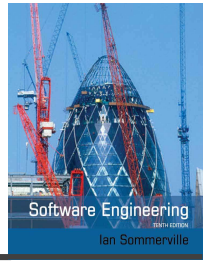
1	Maintenance Agreement for server	$995
1	Maintenance Agreement for server DBMS software	$525
	Preprinted forms (15,000/year @ .22/form)	$3,300

Total Projected Annual Costs: **$13,870**



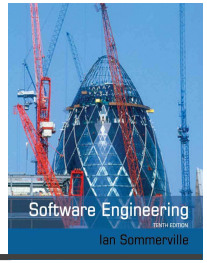
Plan-driven development

Plan-driven development



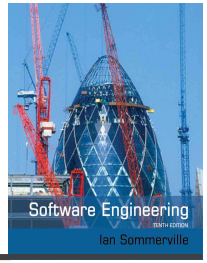
- Plan-driven or plan-based development is an approach to software engineering where the development process is planned in detail.
- A project plan is created that records the work to be done, who will do it, the development schedule and the work products.
- Managers use the plan to support project decision making and as a way of measuring progress.
- The principal argument against plan-driven development is that many early decisions have to be revised because of changes to the environment in which the software is to be developed and used.

Project plans



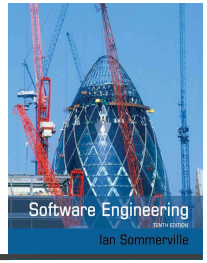
- In a plan-driven development project, a project plan sets out the resources available to the project, the work breakdown and a schedule for carrying out the work.
- Plan sections
  - Introduction
  - Project organization
  - Risk analysis
  - Hardware and software resource requirements
  - Work breakdown
  - Project schedule
  - Monitoring and reporting mechanisms

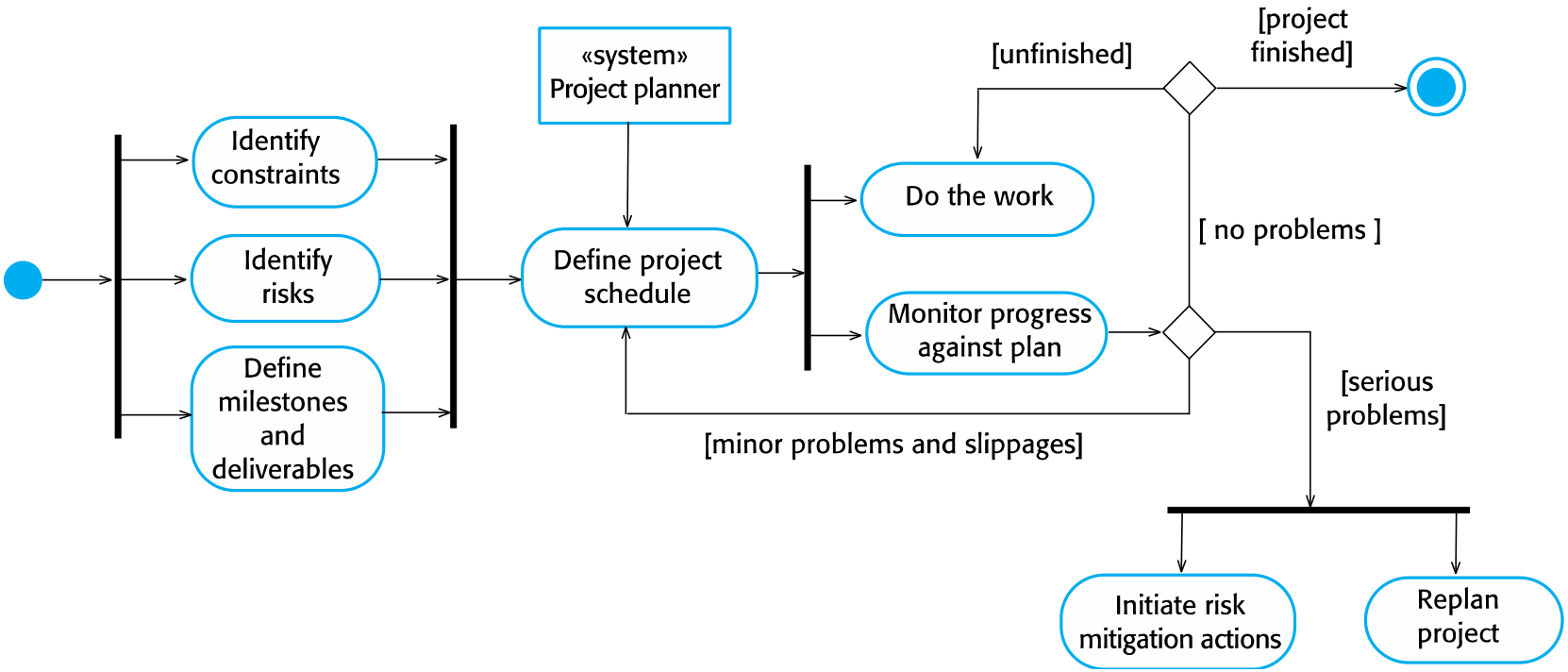
Project plan supplements

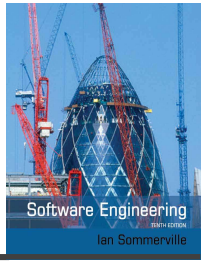


Plan	Description
Configuration management plan	Describes the configuration management procedures and structures to be used.
Deployment plan	Describes how the software and associated hardware (if required) will be deployed in the customer's environment. This should include a plan for migrating data from existing systems.
Maintenance plan	Predicts the maintenance requirements, costs, and effort.
Quality plan	Describes the quality procedures and standards that will be used in a project.
Validation plan	Describes the approach, resources, and schedule used for system validation.

The project planning process

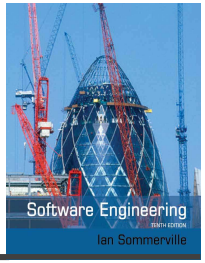






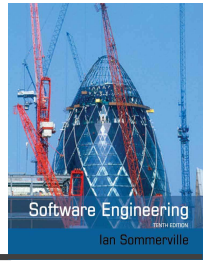
Project scheduling

Project scheduling



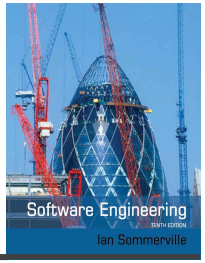
- Project scheduling is the process of deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed.
- You estimate the calendar time needed to complete each task, the effort required and who will work on the tasks that have been identified.
- You also have to estimate the resources needed to complete each task, such as the disk space required on a server, the time required on specialized hardware, such as a simulator, and what the travel budget will be.

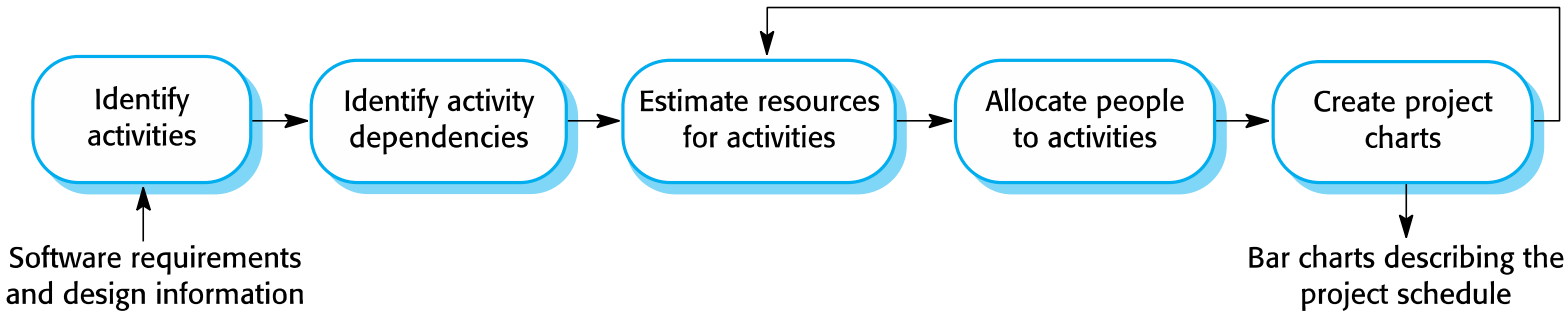
Project scheduling activities



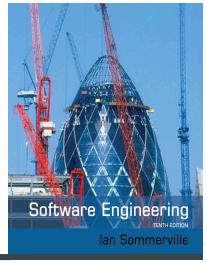
- Split project into tasks and estimate time and resources required to complete each task.
- Organize tasks concurrently to make optimal use of workforce.
- Minimize task dependencies to avoid delays caused by one task waiting for another to complete.
- Dependent on project managers intuition and experience.

The project scheduling process



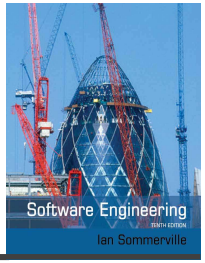


Scheduling problems



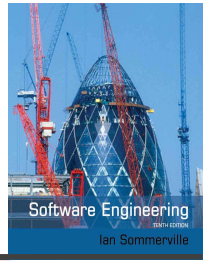
- Estimating the difficulty of problems and hence the cost of developing a solution is hard.
- Productivity is not proportional to the number of people working on a task.
- Adding people to a late project makes it later because of communication overheads.
- The unexpected always happens. Always allow contingency in planning.

Schedule presentation



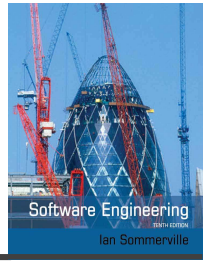
- Graphical notations are normally used to illustrate the project schedule.
- These show the project breakdown into tasks. Tasks should not be too small. They should take about a week or two.
- Calendar-based
  - Bar charts are the most commonly used representation for project schedules. They show the schedule as activities or resources against time.
- Activity networks
  - Show task dependencies

Milestones and deliverables



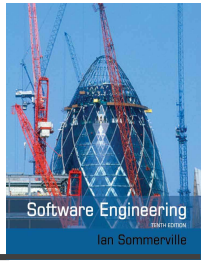
- Milestones are points in the schedule against which you can assess progress, for example, the handover of the system for testing.
- Deliverables are work products that are delivered to the customer, e.g. a requirements document for the system.

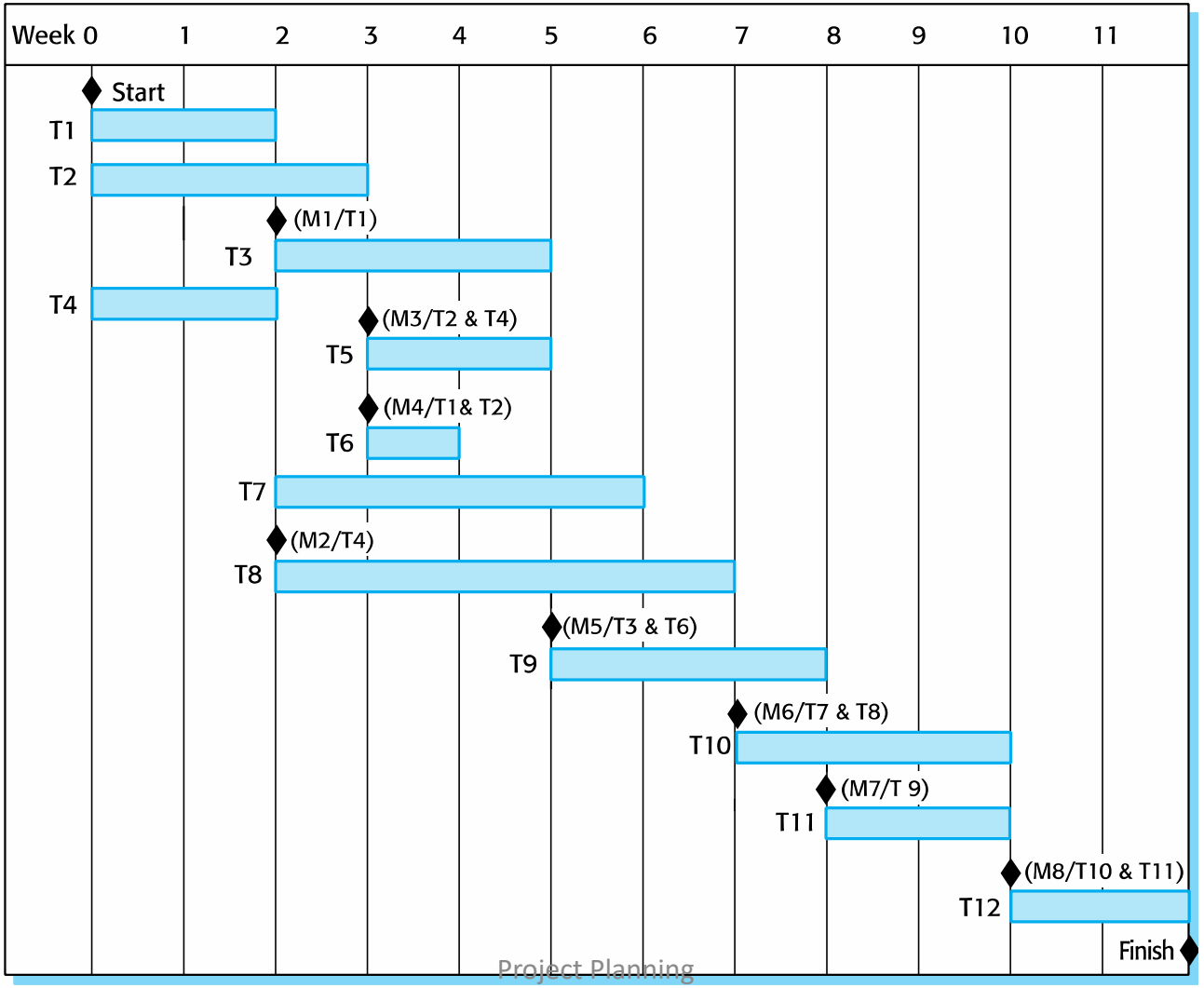
Tasks, durations, and dependencies



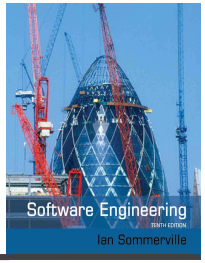
Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

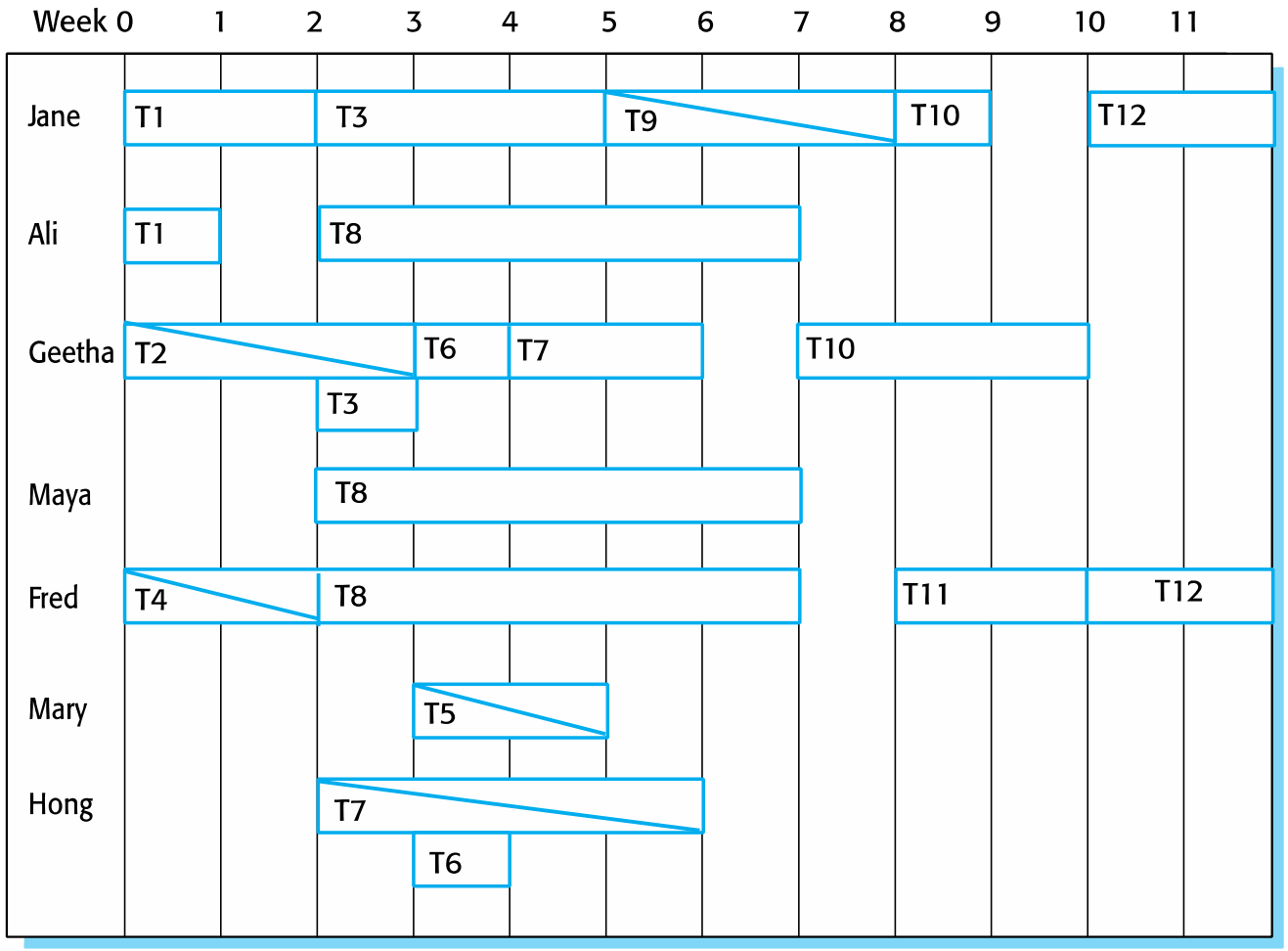
Activity bar chart (Gantt Chart)

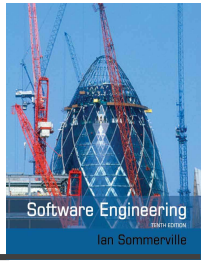




Staff allocation chart

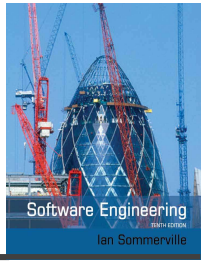






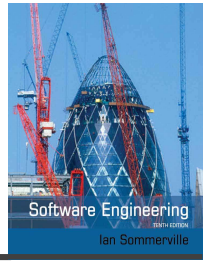
Agile planning

Agile planning



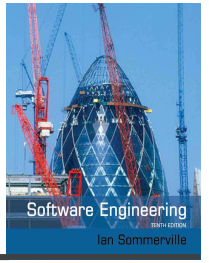
- Agile methods of software development are iterative approaches where the software is developed and delivered to customers in increments.
- Unlike plan-driven approaches, the functionality of these increments is not planned in advance but is decided during the development.
  - The decision on what to include in an increment depends on progress and on the customer's priorities.
- The customer's priorities and requirements change so it makes sense to have a flexible plan that can accommodate these changes.

Agile planning stages

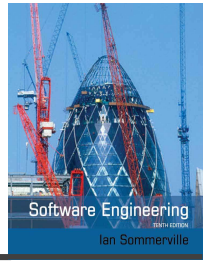


- Release planning, which looks ahead for several months and decides on the features that should be included in a release of a system.
- Iteration planning, which has a shorter term outlook, and focuses on planning the next increment of a system. This is typically 2-4 weeks of work for the team.

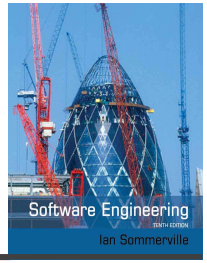
The planning game



Software delivery

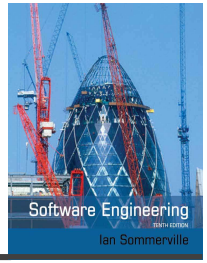


- A software increment is always delivered at the end of each project iteration.
- If the features to be included in the increment cannot be completed in the time allowed, the scope of the work is reduced.
- The delivery schedule is never extended.



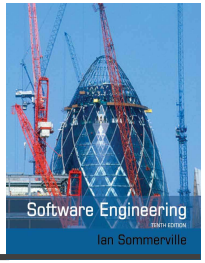
Productivity Measures

Productivity measures



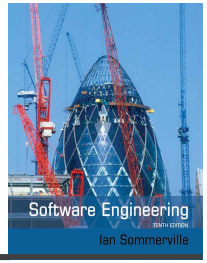
- Size related measures based on some output from the software process. This may be lines of delivered source code, object code instructions, etc.
- Function-related measures based on an estimate of the functionality of the delivered software. Function-points are the best known of this type of measure.

Lines of code



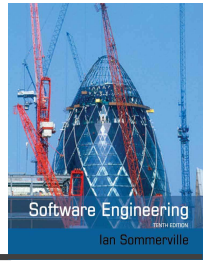
- What is a line of code?
- Productivity measures will vary from language to language – consider difference between lines of code in assembler versus Java
- Relationship to functionality must be based on past efforts in the same language

Productivity estimates - LOC



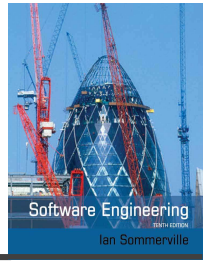
System Category	LOC/person-month
Real-time embedded systems	40-160
Systems programs	150-400
Commercial applications	200-800

Productivity estimates – Function points



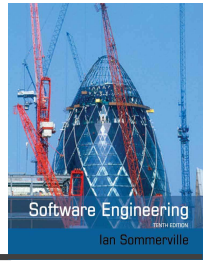
- Based on a combination of program characteristics
  - external inputs and outputs
  - user interactions
  - external interfaces
  - files used by the system
- A weight is associated with each of these
- The function point count is computed by multiplying each raw count by the weight and summing all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

Function points



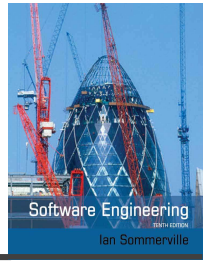
- Function point count modified by complexity of the project
- FPs can be used to estimate LOC depending on the average number of LOC per FP for a given language
  - LOC = AVC * number of function points
  - AVC is a language-dependent factor varying from 200-300 for assemble language to 2-40 for a high-level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

Application points

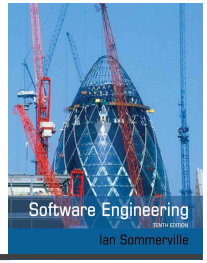


- Application points are an alternative to function points.
- They were originally called object points.
- The number of application points in a program is a weighted estimate of:
  - The number of separate screens that are displayed. Simple screens count as 1 object point, moderately complex screens count as 2 and very complex screens count as 3 object points.
  - The number of reports that are produced. For simple reports, count 2 object points, for moderately complex reports, count 5 and for reports which are likely to be difficult to produce, count 8 object points.
  - The number of modules in imperative programming languages such as Java or C++ that must be developed to supplement the database programming code. Each of these modules counts as 10 object points.

Function Points vs. Application Points

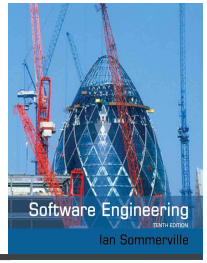


- The advantage of application points over function points is that they are easier to estimate from a high-level software specification.
- Object points are only concerned with screens, reports and modules in conventional programming languages.
- They are not concerned with implementation details and the complexity factor estimation is much simpler.



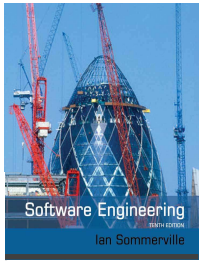
Estimation techniques

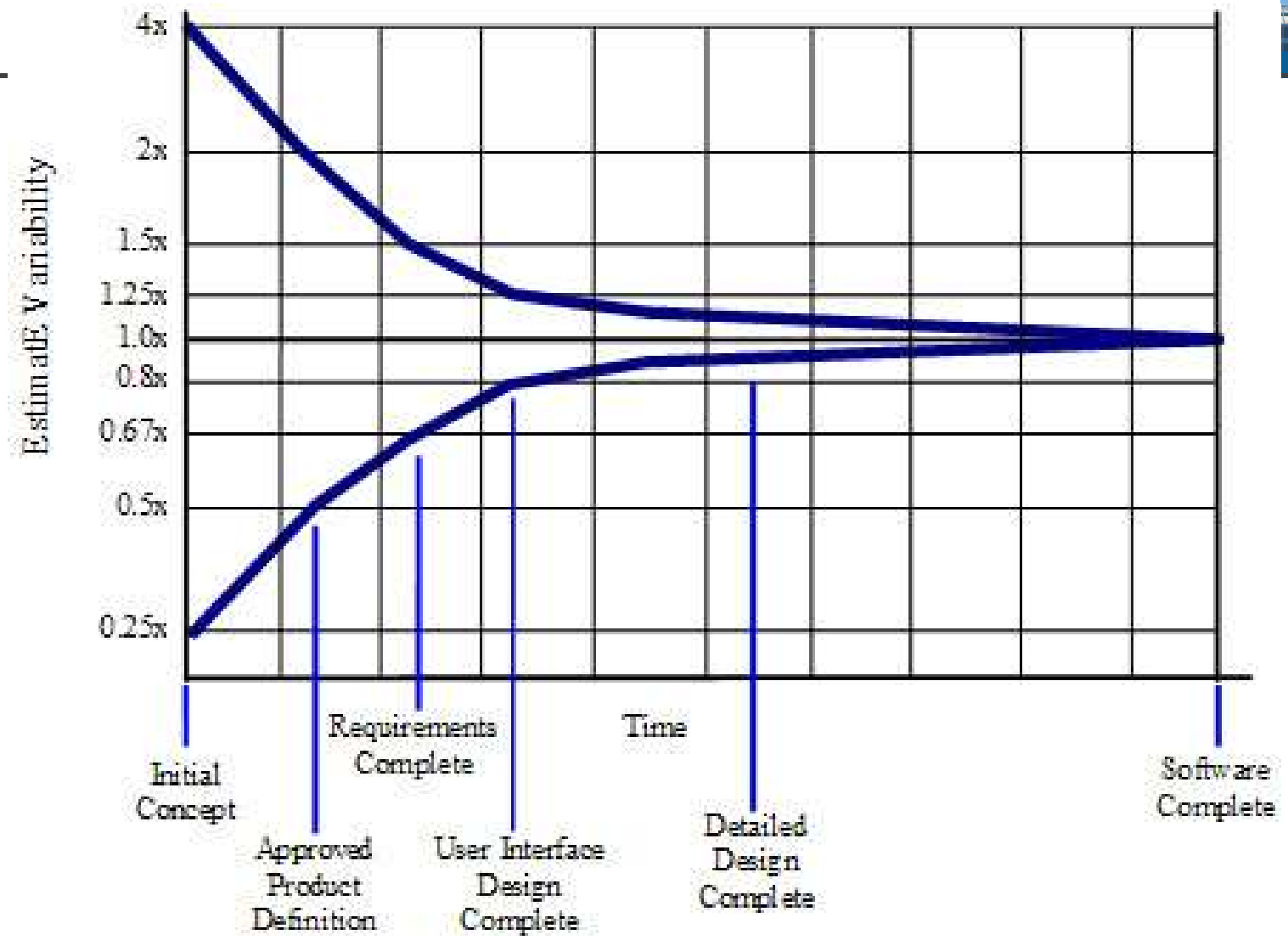
Estimation techniques



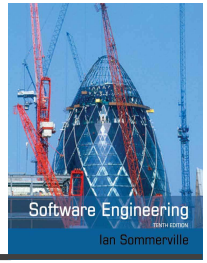
- Organizations need to make software effort and cost estimates. There are two types of technique that can be used to do this:
  - *Experience-based techniques* The estimate of future effort requirements is based on the manager's experience of past projects and the application domain. Essentially, the manager makes an informed judgment of what the effort requirements are likely to be.
  - *Algorithmic cost modeling* In this approach, a formulaic approach is used to compute the project effort based on estimates of product attributes, such as size, and process characteristics, such as experience of staff involved.

Cone of Estimation Uncertainty



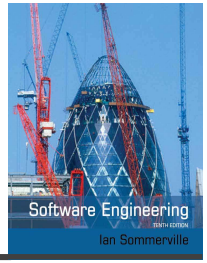


Experience-based approaches

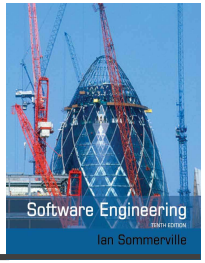


- Experience-based techniques rely on judgments based on experience of past projects and the effort expended in these projects on software development activities.
- Typically, you identify the deliverables to be produced in a project and the different software components or systems that are to be developed.
- You document these in a spreadsheet, estimate them individually and compute the total effort required.
- It usually helps to get a group of people involved in the effort estimation and to ask each member of the group to explain their estimate.

Problem with experience-based approaches

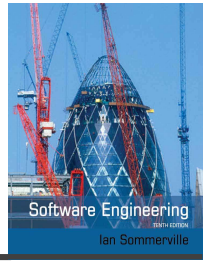


- The difficulty with experience-based techniques is that a new software project may not have much in common with previous projects.
- Software development changes very quickly and a project will often use unfamiliar techniques such as web services, application system configuration or HTML5.
- If you have not worked with these techniques, your previous experience may not help you to estimate the effort required, making it more difficult to produce accurate costs and schedule estimates.



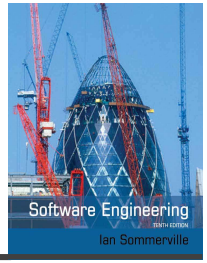
COCOMO – Constructive Cost Model

COCOMO cost modeling



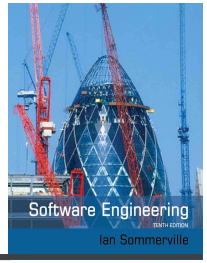
- An empirical model based on project experience.
- Well-documented, 'independent' model which is not tied to a specific software vendor.
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2.
- COCOMO 2 takes into account different approaches to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

COCOMO 81 Model



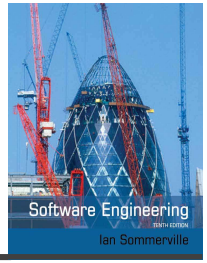
- Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers:
  - Effort = A x $Size^B$ x M
  - A is an organisation-dependent constant, B reflects the disproportionate effort for large projects and M is a multiplier reflecting 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.

COCOMO 81 Model



- Effort = A x Size^{B} x 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.
- **A** is depends in on the type of software that is being developed (simple, moderate, embedded) [will vary 2.4-3.5]
- **Size** is an estimate of the code size or other functional assessment [thousands of lines of code, ie. 5,400 LOC→ 5.4]
- **B** reflects the disproportionate effort for large projects over small projects [typically 1.0-1.5]
- **M** is a multiplier reflecting a combination of product, process and people attributes (e.g. desired reliability, reuse required, personnel capability and experience, support facilities) [will vary up from 1.0]

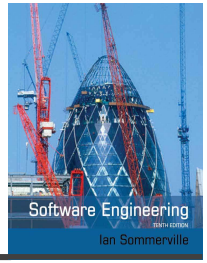
COCOMO 81 Effort (PM) = A * Size^{B} * M



Project complexity	Formula	Description
Simple	PM = 2.4 (KDSI)$^{1.05}$ × M	Well-understood applications developed by small teams.
Moderate	PM = 3.0 (KDSI)$^{1.12}$ × M	More complex projects where team members may have limited experience of related systems.
Embedded	PM = 3.6 (KDSI)$^{1.20}$ × M	Complex projects where the software is part of a strongly coupled complex of hardware, software, regulations and operational procedures.

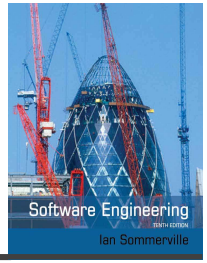
PM = person-months
KDSI = thousand of delivered software instructions

Estimation accuracy



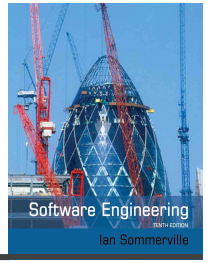
- The size of a software system can only be known accurately when it is finished.
- Several factors influence the final size
  - Use of reused systems and components;
  - Programming language;
  - Distribution of system.
- As the development process progresses then the size estimate becomes more accurate.
- The estimates of the factors contributing to B and M are subjective and vary according to the judgment of the estimator.

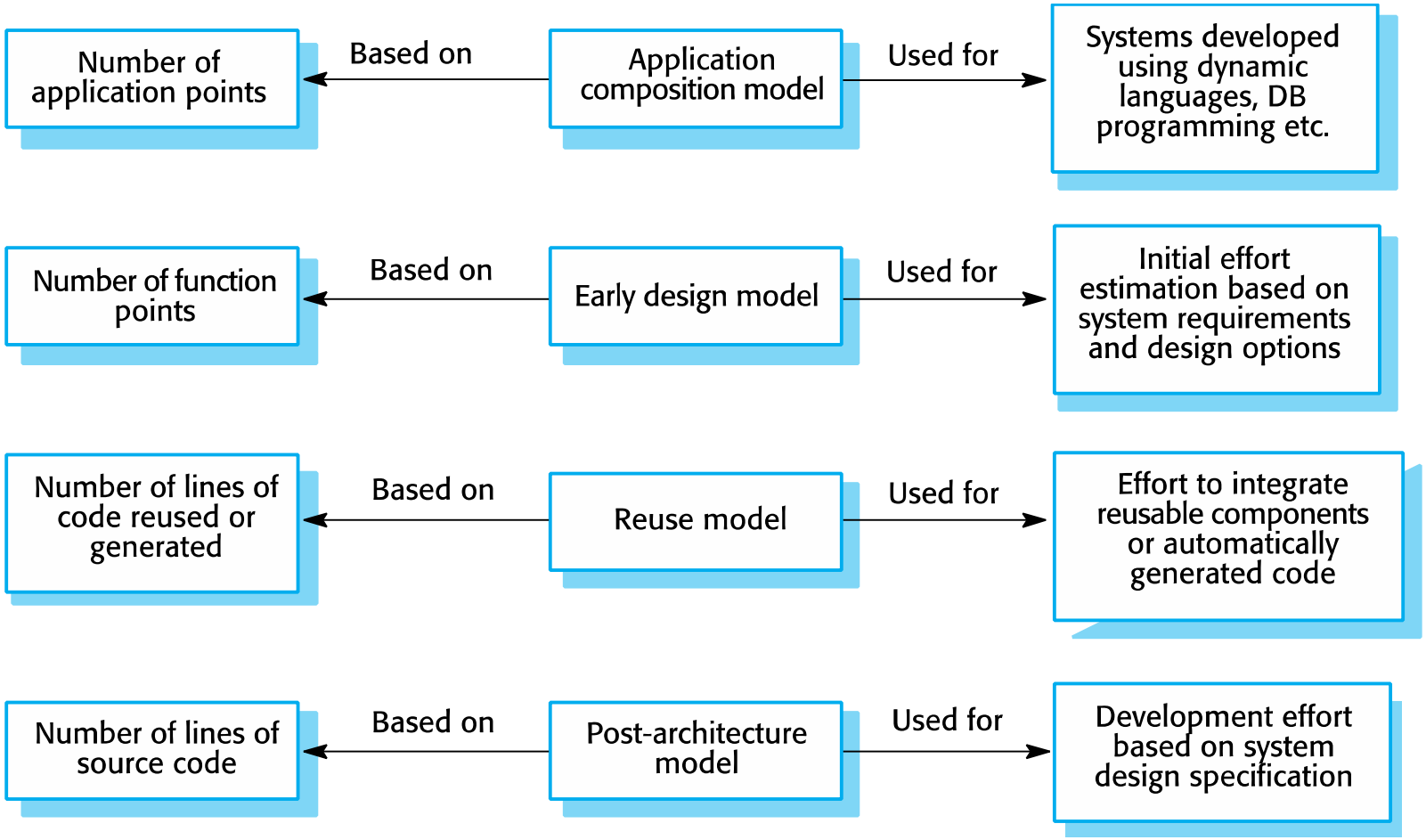
COCOMO 2 models



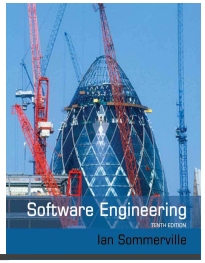
- COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.
- The sub-models in COCOMO 2 are:
  - Application composition model. Used when software is composed from existing parts.
  - Early design model. Used when requirements are available but design has not 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.

COCOMO 2 estimation models



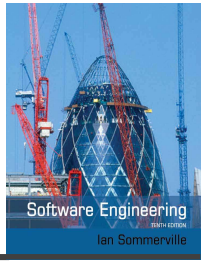


Key points



- The price charged for a system does not just depend on its estimated development costs and the profit required by the development company.
- Software is often priced to gain a contract and the functionality of the system is then adjusted to meet the estimated price.
- Plan-driven development is organized around a complete project plan that defines the project activities, the planned effort, the activity schedule and who is responsible for each activity.
- Project scheduling involves the creation of various graphical representations of part of the project plan. Bar charts, which show the activity duration and staffing timelines, are the most commonly used schedule representations.

Key points



- The agile planning game involves the whole team in project planning. The plan is developed incrementally and, if problems arise, it is adjusted so that software functionality is reduced instead of delaying the delivery of an increment.
- Estimation techniques for software may be experience-based, where managers judge the effort required, or algorithmic, where the effort required is computed from other estimated project parameters.
- The COCOMO II costing model is a mature algorithmic cost model that takes project, product, hardware and personnel attributes into account when formulating a cost estimate.