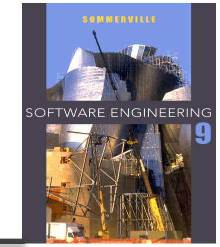
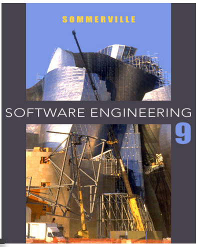


Chapter 23 – Project planning

Lecture 1



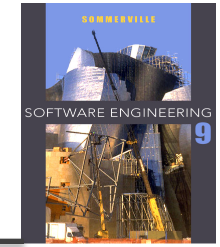
Topics covered

- ✧ Software pricing
- ✧ Plan-driven development
- ✧ Project scheduling
- ✧ Agile planning
- ✧ Estimation techniques

Project planning

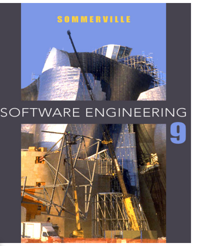
- Project planning involves breaking down the work into parts and assign these to project team members, anticipate problems that might arise and prepare tentative solutions to those problems.
- The project plan, which is created at the start of a project, is used to communicate how the work will be done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

Planning stages



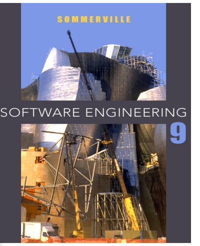
- At the proposal stage, when you are bidding for a contract to develop or provide a software system.
- During the project startup phase, when you have to plan who will work on the project, how the project will be broken down into increments, how resources will be allocated across your company, etc.
- Periodically throughout the project, when you modify your plan in the light of experience gained and information from monitoring the progress of the work.

Proposal planning



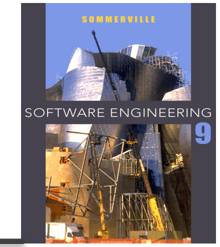
- Planning may be necessary with only outline software requirements.
- The aim of planning at this stage is to provide information that will be used in setting a price for the system to customers.

23.1 Software pricing



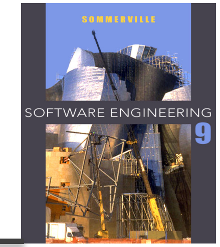
- Estimates are made to discover the cost, to the developer, of producing a software system.
  - You take into account, hardware, software, travel, training and effort costs.
- There is not a simple relationship between the development cost and the price charged to the customer.
- Broader organisational,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

Factors affecting software pricing



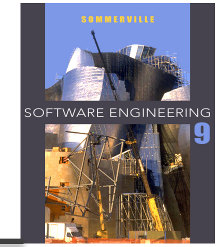
Factor	Description
Market opportunity	A development organization may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the organization the opportunity to make a greater profit later. The experience gained may also help it develop new products.
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.
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.

Factors affecting software pricing



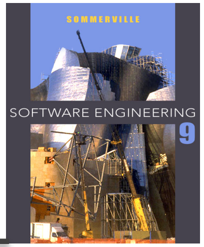
Factor	Description
Requirements volatility	If the requirements are likely to change, an organization may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements.
Financial health	Developers in financial difficulty may lower their price 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.

23.2 Plan-driven development



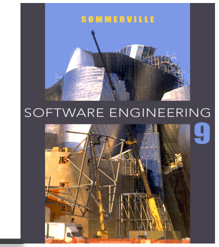
- Plan-driven or plan-based development is an approach to software engineering where the development process is planned in detail.
  - Plan-driven development is based on engineering project management techniques and is the ‘traditional’ way of managing large software development projects.
- A project 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.

Plan-driven development – pros and cons



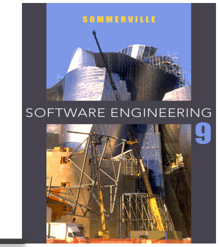
- The arguments in favor of a plan-driven approach are that early planning allows organizational issues (availability of staff, other projects, etc.) to be closely taken into account, and that potential problems and dependencies are discovered before the project starts, rather than once the project is underway.
- 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.

23.2.1 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

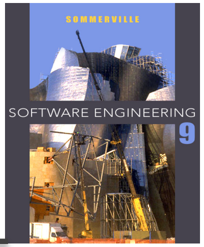


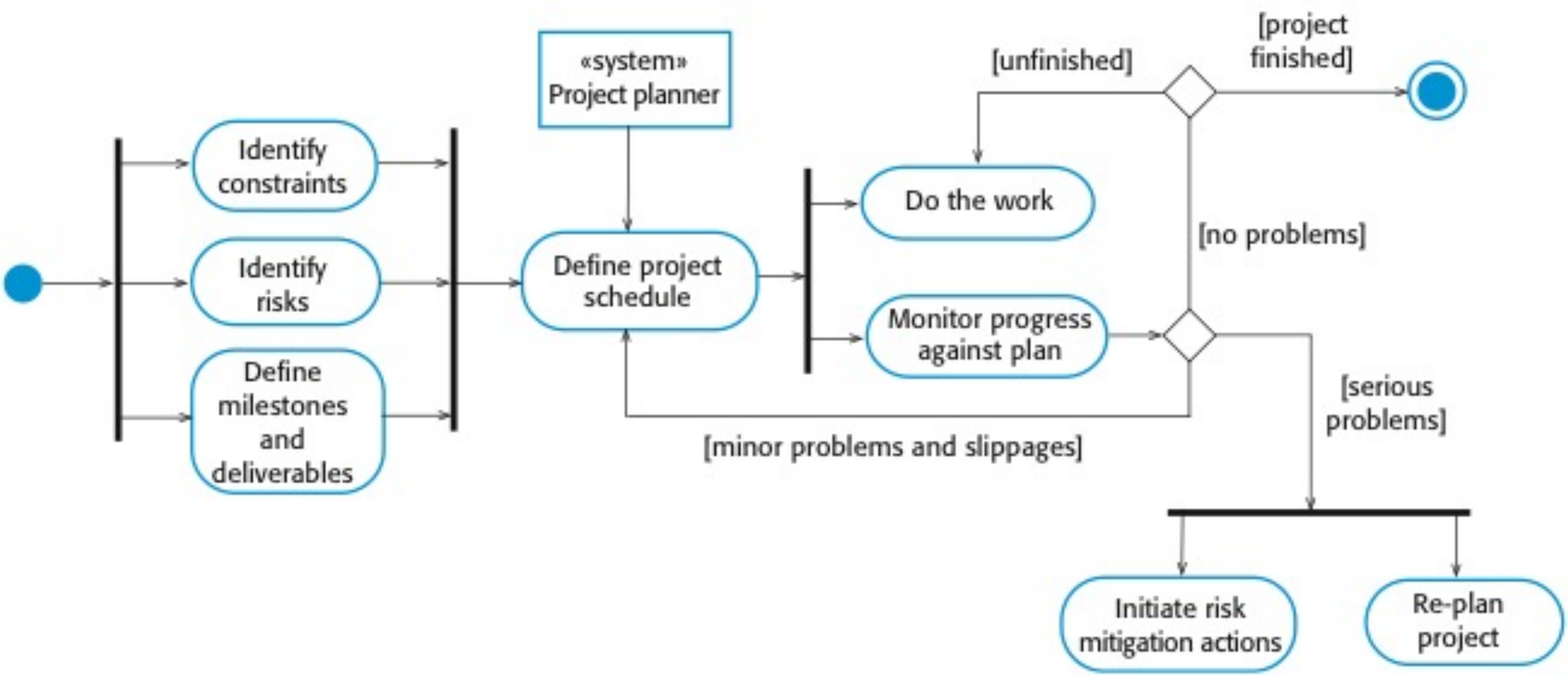
Plan	Description
Quality plan	Describes the quality procedures and standards that will be used in a project.
Validation plan	Describes the approach, resources, and schedule used for system validation.
Configuration management plan	Describes the configuration management procedures and structures to be used.
Maintenance plan	Predicts the maintenance requirements, costs, and effort.
Staff development plan	Describes how the skills and experience of the project team members will be developed.

23.2.2 The planning process

- Project planning is an iterative process that starts when you create an initial project plan during the project startup phase.
- Plan changes are inevitable.
  - As more information about the system and the project team becomes available during the project, you should regularly revise the plan to reflect requirements, schedule and risk changes.
  - Changing business goals also leads to changes in project plans. As business goals change, this could affect all projects, which may then have to be re-planned.

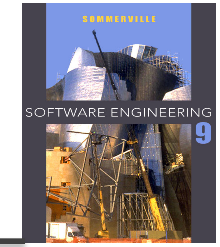
The project planning process





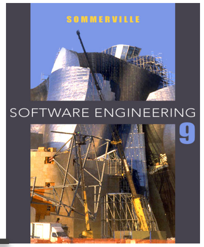
UML activity diagram - A typical workflow

23.3 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

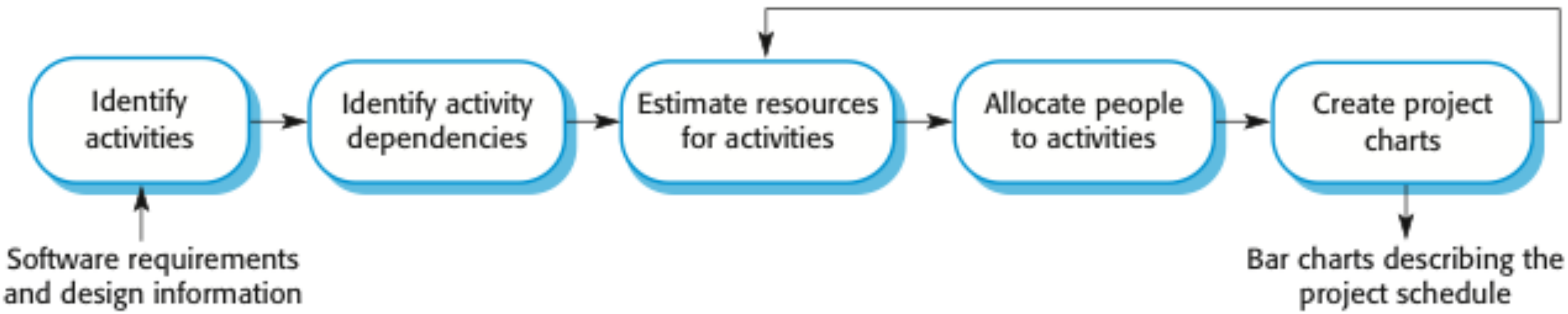


- ✧ Split project into tasks and estimate time and resources required to complete each task.
- ✧ Organize tasks concurrently to make optimal use of workforce.
- ✧ Minimize task dependencies to avoid delays caused by one task waiting for another to complete.
- ✧ Dependent on project managers intuition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

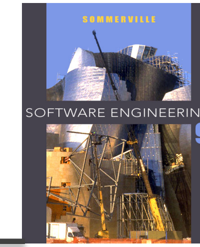
Milestones and deliverables

- 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.

The project scheduling process



Scheduling problems

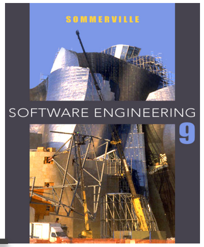


- ✧ Estimating the difficulty of problems and hence the cost of developing a solution is hard.
- ✧ Productivity is not proportional to the number of people working on a task.
- ✧ Adding people to a late project makes it later because of communication overheads.
- ✧ The unexpected always happens. Always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

Schedule representation

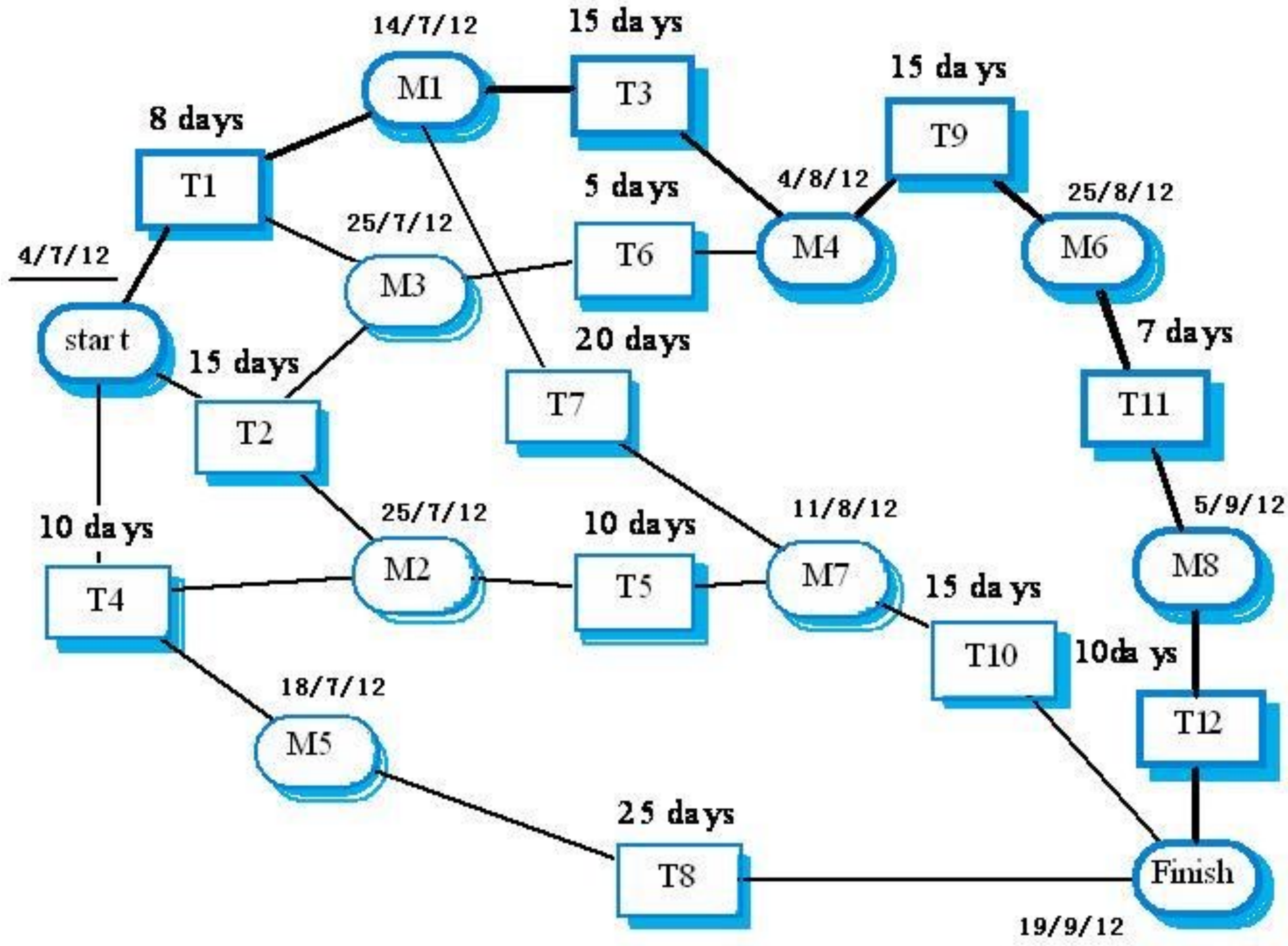
- Graphical notations are normally used to illustrate the project schedule.
- These show the project breakdown into tasks. Tasks should not be too small. They should take about a week or two.
- Bar charts are the most commonly used representation for project schedules. They show the schedule as activities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

Tasks, durations, and dependencies



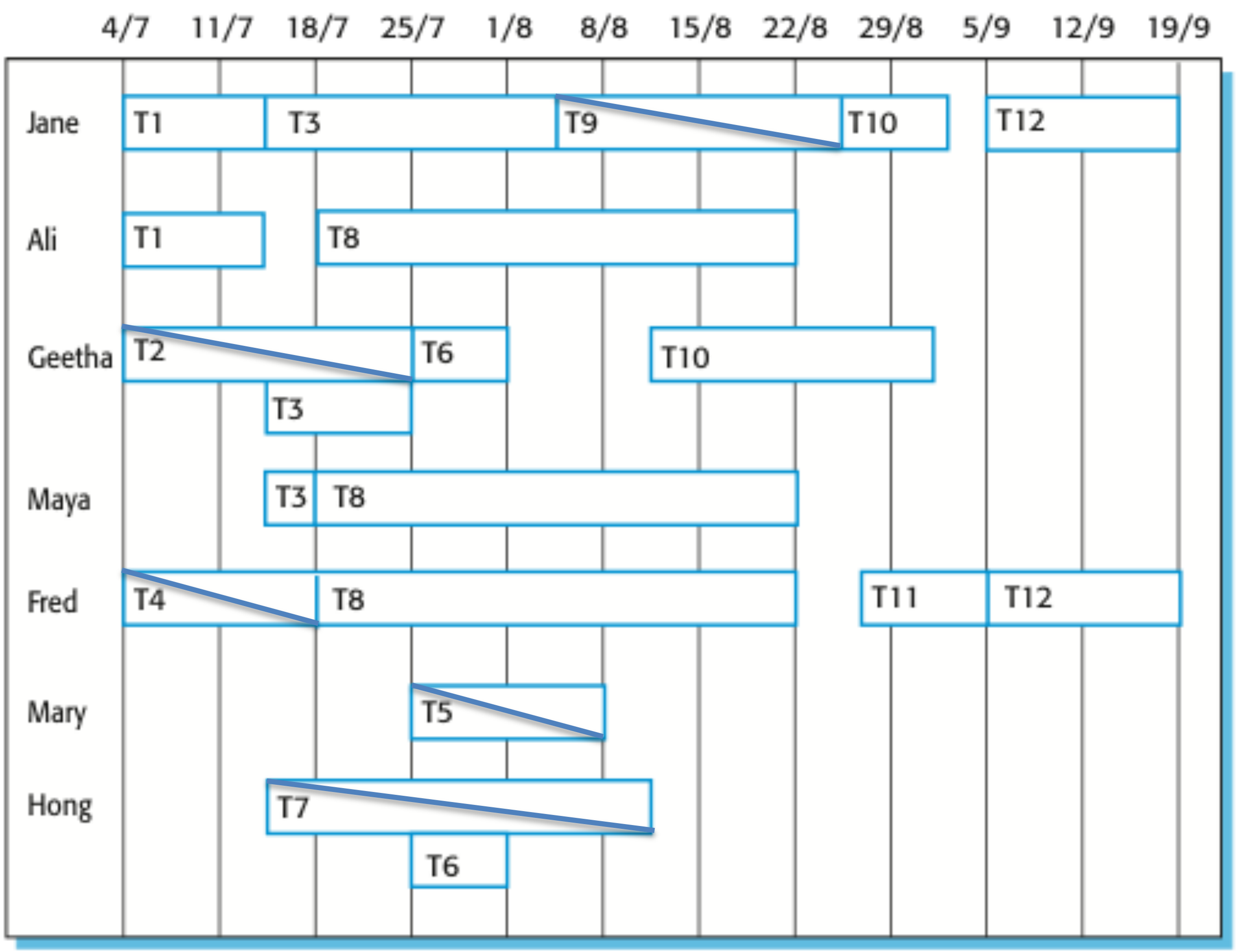
Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

Activity network



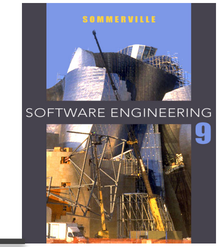
This activity network is not exactly equivalent to the last slide.

Staff allocation chart



Part-time assignments using a diagonal line crossing the bar

23.4 Agile planning

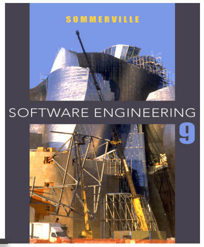


- ✧ Agile methods of software development are iterative approaches where the software is developed and delivered to customers in increments.
- ✧ Unlike plan-driven approaches, the functionality of these increments is not planned in advance but is decided during the development.
  - The decision on what 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.

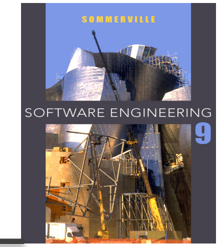
Agile planning stages

- (1) Release planning, which looks ahead for several months and decides on the features that should be included in a release of a system.
- (2) 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.

Planning in XP

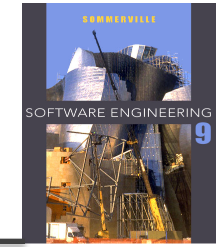


Story-based planning

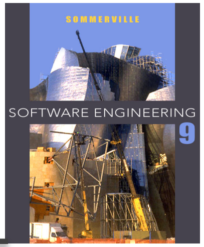


- The system specification in XP is based on user stories that reflect the features that should be included in the system.
- The project team read and discuss the stories and rank them in order of the amount of time they think it will take to implement the story.
- Release planning involves selecting and refining the stories that will reflect the features to be implemented in a release of a system and the order in which the stories should be implemented.
- Stories to be implemented in each iteration are chosen, with the number of stories reflecting the time to deliver an iteration (usually 2 or 3 weeks) and the team's velocity.

Key points



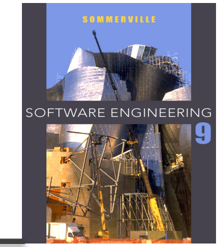
- The price charged for a system does not just depend on its estimated development costs; it may be adjusted depending on the market and organizational priorities.
- Plan-driven development is organized around a complete project plan that defines the project activities, the planned effort, the activity schedule and who is responsible for each activity.
- Project scheduling involves the creation of graphical representations of part of the project plan. Bar charts show the activity duration and staffing timelines, are the most commonly used schedule representations.
- The XP 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 delivery of an increment.



Chapter 23 – Project planning

Lecture 2

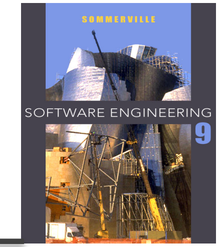
23.5 Estimation techniques



✧ 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.

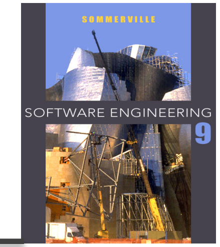
Experience-based approaches



- 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.

The difficulty with experience-based techniques is that a new software project may not have much in common with previous projects.

23.5.1 Algorithmic cost modelling



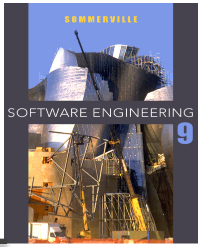
✧ Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers:

- $\text{Effort} = A \times \text{Size}^B \times M$
- A is an organisation-dependent constant, Size may be either the code size or function/application points, B reflects the disproportionate effort for large projects (usually between 1 and 1.5), and M is a multiplier reflecting product, process and development attributes.

✧ The most commonly used product attribute for cost estimation is code size(SLOC-the number of lines of source code is the fundamental size metric).

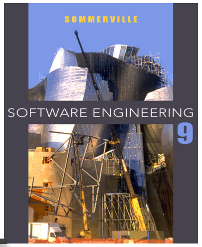
✧ Most models are similar but they use different values for A, B and M.

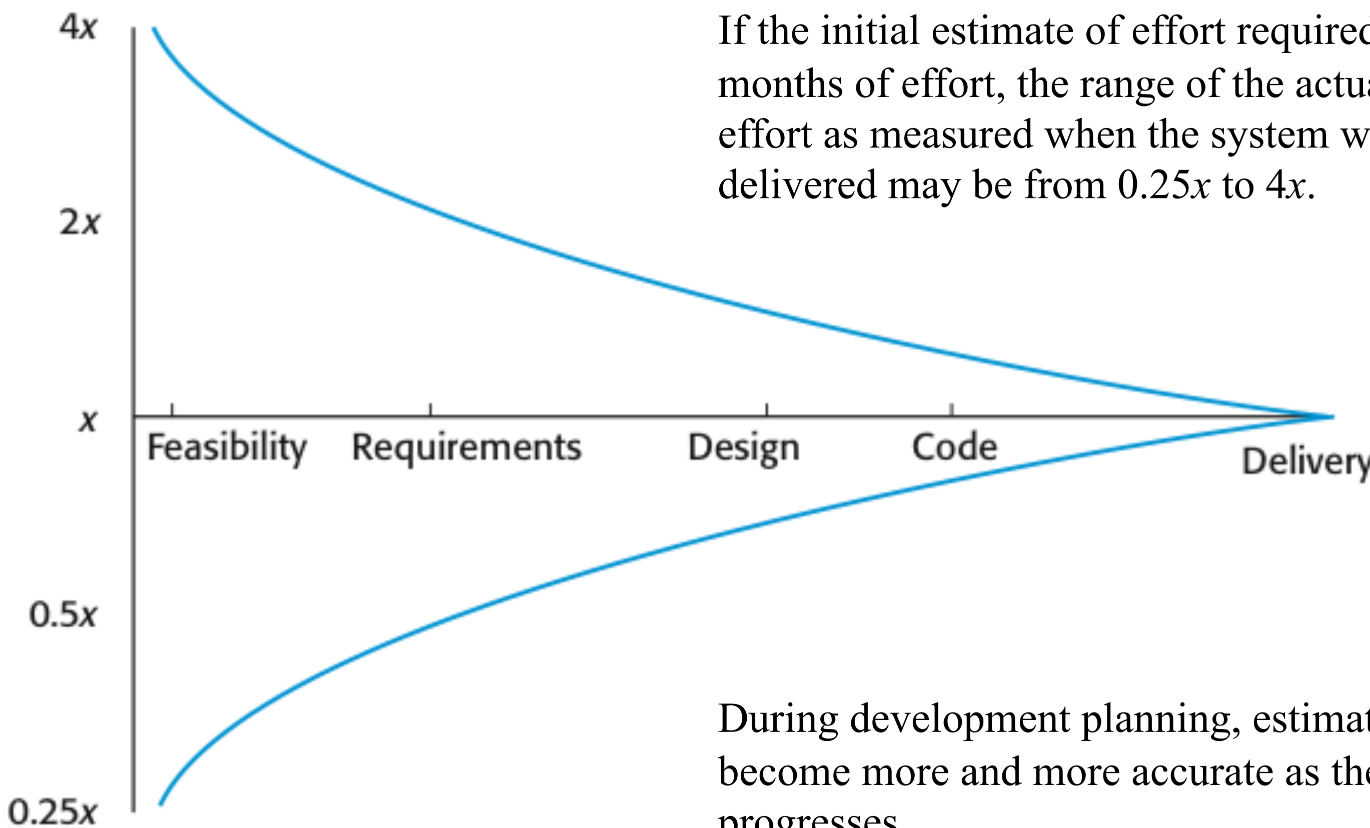
Estimation accuracy



- 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.
- The estimates of the factors contributing to B and M are subjective and vary according to the judgment of the estimator.

Estimate uncertainty

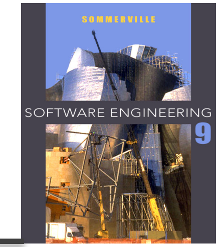




If the initial estimate of effort required is *x* months of effort, the range of the actual effort as measured when the system was delivered may be from 0.25*x* to 4*x*.

During development planning, estimates become more and more accurate as the project progresses.

23.5.2 The COCOMO 2 model

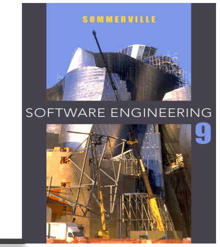


- ✧ COCOMO(Constructive cost model), an empirical model based on project experience.
- ✧ Well-documented and nonproprietary estimation model which is not tied to a specific software vendor.
- ✧ Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2.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

Modern approaches, such as:

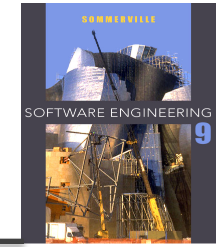
- Rapid development using dynamic languages,
- Development by component composition,
- Use of database programming.

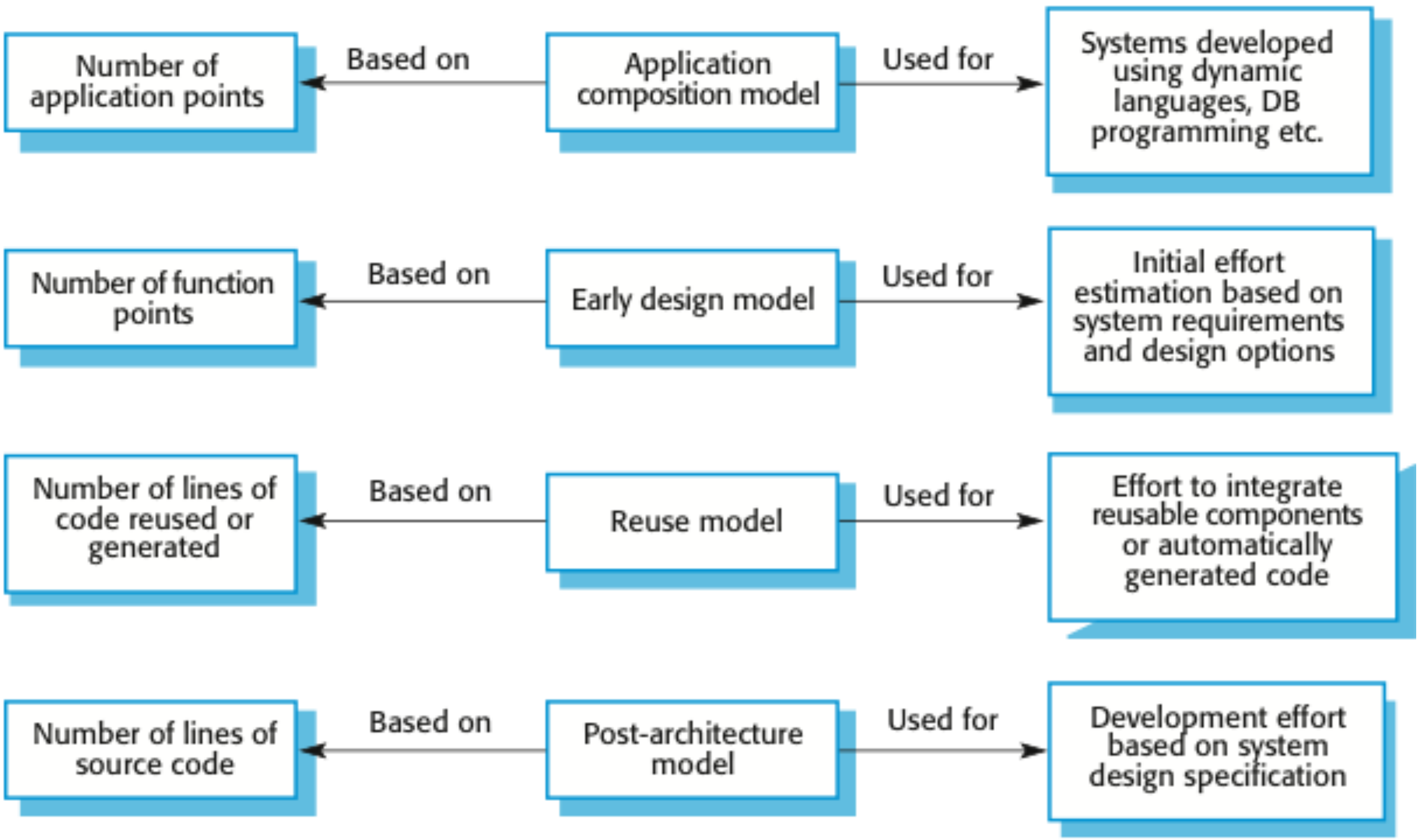
COCOMO 2 models



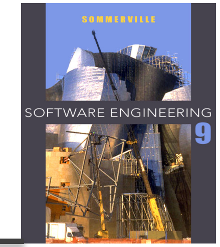
- COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.
- The sub-models in COCOMO 2 are:
  - Application composition model. Used when software is composed from existing parts.
  - Early design model. Used when requirements are available but design has not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



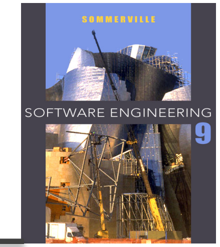


(1) Application composition model



- 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
  - $PM = (NAP \times (1 - \%reuse/100)) / PROD$
  - PM is the effort in person-months, NAP is the total number of application points in the delivered system, PROD is the productivity(as shown in next slide) and '%reuse' is an estimate of the amount of reused code in the development.

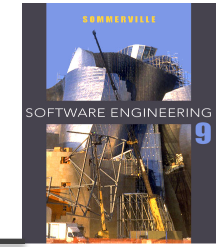
Application-point productivity



Developer's experience and capability	Very low	Low	Nominal	High	Very high
ICASE maturity and capability	Very low	Low	Nominal	High	Very high
PROD (NAP/month)	4	7	13	25	50

Application composition model produces an approximate estimate as it does not take into account the additional effort involved in reuse.

(2) 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;
    (based on seven project and process attributes-next slide)
  - **A** = 2.94 in initial calibration,

    Size in KSLOC（the number of thousands of lines of source code）,

    B varies from 1.1 to 1.24 depending on novelty of the project, development flexibility, risk management approaches and the process maturity.

This results in an effort computation as follows:

PM = 2.94 × Size$^{(1.1\sim1.24)}$ × M

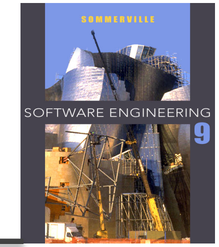
Multipliers

- 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.

(3) 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. The development effort is taken to be zero. Black-box code is code that can be reused without understanding the code or making changes to it.
  - 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



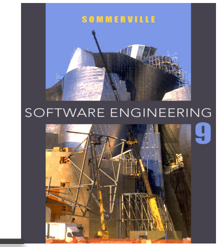
✧ For generated code:

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

Boehm, et al.(2000) have measured ATPROD to be about 2,400 source statements per month. Therefore, if there are a total of 20,000 lines of reused source code in a system and 30% of this is automatically generated, then the effort required to integrate the generated code is:

(20000 * 30/100)/2400=2.5 person-months //generated code

Reuse model estimates 2

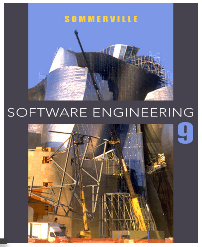


✧ If some code adaptation can be done automatically, this reduces the effort required. You therefore adjust the estimate by estimating the percentage of automatically adapted code(AT) and using this to adjust ASLOC. Then the final formula is:

ESLOC = ASLOC * (1-AT/100) * AAM.

✧ Once ESLOC has been calculated, you then apply the standard estimation formula to calculate the total effort required, where the Size parameter equals to ESLOC. You then add this to the effort to integrate automatically generated code that you have already computed, thus computing the total effort required.

(4) Post-architecture level



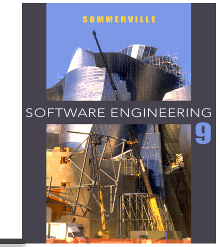
- Uses the same formula as the early design model but with 17 rather than 7 associated multipliers.

$$\textbf{PM} = A \times \mathrm{Size}^{B} \times M$$

- The code size is estimated as:
  - (1) Number of lines of new code to be developed(SLOC);
  - (2) Estimate of equivalent number of lines of new code computed using the reuse model(ESLOC);
  - (3) An estimate of the number of lines of code that have to be modified according to requirements changes.

  - We add the values of these parameters to compute the total code size, in KSLOC, that you use in the effort computation formula.

The exponent term

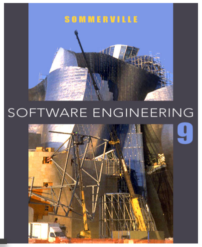


- ✧ The exponent term(B) depends on 5 scale factors (rated on a six-point scale from 0 to 5). To calculate B, we add the ratings, divide them by 100, and add the result to 1.01(sum/100+1.01).

Scale factor	Explanation
Precedentedness	Reflects the previous experience of the organization with this type of project. Very low means no previous experience; extra-high means that the organization is completely familiar with this application domain.
Development flexibility	Reflects the degree of flexibility in the development process. Very low means a prescribed process is used; extra-high means that the client sets only general goals.
Architecture/risk resolution	Reflects the extent of risk analysis carried out. Very low means little analysis; extra-high means a complete and thorough risk analysis.
Team cohesion	Reflects how well the development team knows each other and work together. Very low means very difficult interactions; extra-high means an integrated and effective team with no communication problems.
Process maturity	Reflects the process maturity of the organization. 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.

Scalefactors used in the exponent computation in the post-architecture model

The exponent term



✧ Example:

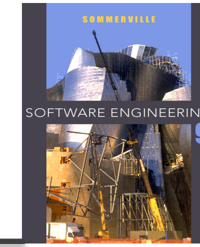
A company takes on a project in a new domain. The client has not defined the process to be used and has not allowed time for risk analysis. The company has a CMM level 2 rating.

- 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, which is calculated by:

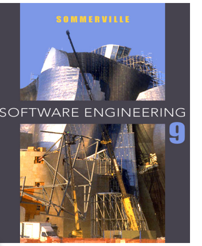
$$B=(4+1+5+3+3)/100+1.01=1.17$$

Multipliers



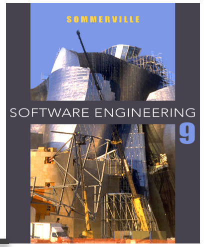
- ✧ Product attributes
  - Concerned with required characteristics of the software product being developed.
- ✧ Computer attributes
  - Constraints imposed on the software by the hardware platform.
- ✧ Personnel attributes
  - Multipliers that take the experience and capabilities of the people working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

The effect of cost drivers on effort estimates



Exponent value	**1.17**	
System size (including factors for reuse and requirements volatility)	128,000 DSI	
Initial COCOMO estimate without cost drivers	**730 person-months**	
Reliability	Very high, multiplier = 1.39	Assign maximum values to the key cost drivers
Complexity	Very high, multiplier = 1.3	
Memory constraint	High, multiplier = 1.21	
Tool use	Low, multiplier = 1.12	
Schedule	Accelerated, multiplier = 1.29	
Adjusted COCOMO estimate	**2,306 person-months**	

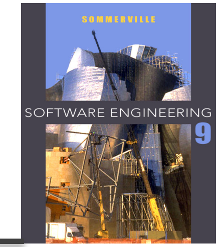
The effect of cost drivers on effort estimates



Exponent value	**1.17**	
Reliability	Very low, multiplier = 0.75	Assign minimum values to the key cost drivers
Complexity	Very low, multiplier = 0.75	
Memory constraint	None, multiplier = 1	
Tool use	Very high, multiplier = 0.72	
Schedule	Normal, multiplier = 1	
Adjusted COCOMO estimate	**295 person-months**	

We can see that high values for the cost drivers lead an effort estimate that is more than three times(2306) the initial estimate(730), whereas low values reduce the estimate to about one-third(295) of the original.
This highlights the significant differences between different types of projects and the difficulties of transferring experience from one application domain to another.

23.5.3 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 computed by COCOMO model and B is the complexity-related exponent as discussed above (B is 1 for the early prototyping model). This computation predicts the nominal schedule(TDEV) for the project, ignoring any multiplier that is related to the project schedule.
- The time required is independent of the number of people working on the project.

Staffing requirements

- ✧ Staff required can’t simply be computed through dividing the total effort by the required project 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

- 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 an algorithmic cost model that uses project, product, hardware and personnel attributes as well as product size and complexity attributes to derive a cost estimate.