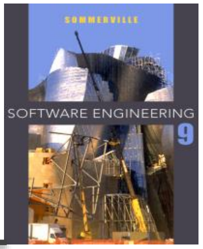
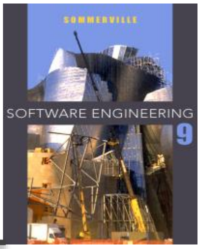


Chapter 23 – Project planning

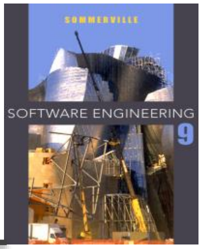
Lecture 1

Topics covered



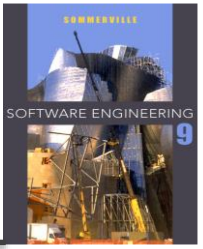
- ✧ Software pricing
- ✧ Plan-driven development
- ✧ Project scheduling
- ✧ Agile planning
- ✧ Estimation techniques

Project planning



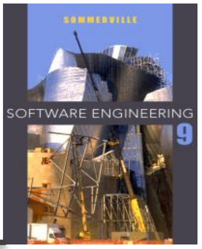
- ✧ Project planning involves **breaking down the work** into parts and **assign these** to project team members, **anticipate problems** that might arise and **prepare tentative solutions** to those problems.
- ✧ The **project plan**, which is created at the start of a project, is **used to communicate**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

Planning stages



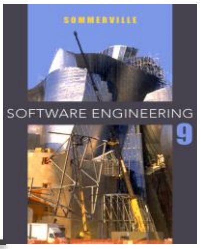
- **At the proposal stage**, when you are **bidding for a contract** to develop or provide a software system.
- **During the project startup phase**, when you have to plan **who** will work on the project, **how** the project will be **broken down** into increments, **how resources will be allocated** 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



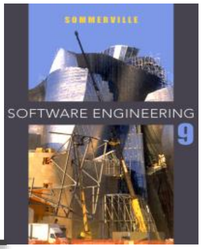
- Planning may be necessary **with only outline software requirements**.
- The aim of planning at this stage is to provide information that will be **used in setting a price** for the system to customers.

Software pricing



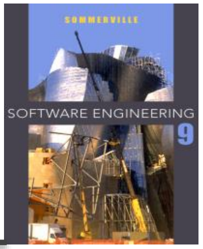
- ✧ **Estimates are made to discover the cost**, to the developer, of producing a software system.
  - You take into account, **hardware, software, travel, training** and **effort costs**.
- ✧ There is **not a simple relationship between** the development **cost and the price** charged to the customer.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

Factors affecting software pricing



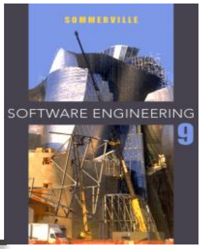
Factor	Description
Market opportunity	A development organization **may quote a low price because** it **wishes to move into a new segment** of the software market. Accepting a low profit on one project may give the organization the opportunity to make a greater profit later. The experience gained may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



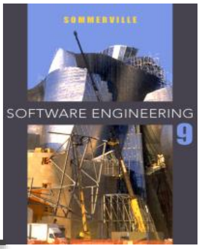
Factor	Description
Requirements volatility	If the requirements are likely to change, **an organization may lower its price to win a contract**. After the contract is awarded, **high prices can be charged for changes** to the requirements.
Financial health	Developers in financial difficulty may lower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

Plan-driven development



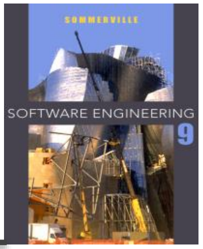
- ✧ Plan-driven **or plan-based development** is an approach to software engineering where **the development process is planned in detail.**
  - Plan-driven development is based on engineering project management techniques and is **the 'traditional' way of managing large software development projects.**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



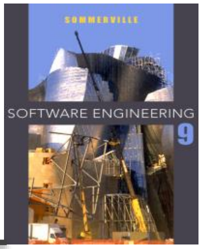
- ✧ **The arguments in favor** of a plan-driven approach are that early planning allows organizational issues (**availability of staff, other projects, etc**.) to be closely **taken into account**, and that **potential problems and dependencies are discovered** before the project starts, rather than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

Project plans



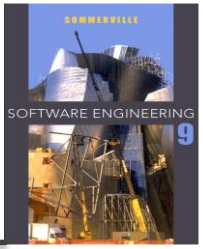
- ✧ In a plan-driven development project, a project plan **sets out the resources available to the project, the work breakdown and a schedule** for carrying out the work.
- ✧ **Plan sections**
  - Introduction
  - Project organization
  - Risk analysis
  - Hardware and software resource requirements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

Project plan supplements



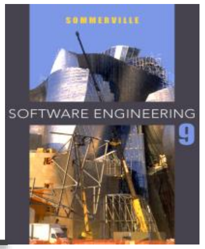
Plan	Description
Quality plan	Describes the **quality procedures and standards** that will be used in a project.
Validation plan	Describes the approach, resources, and schedule used for system validation.
Configuration management plan	Describes the configuration management procedures and structures to 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.

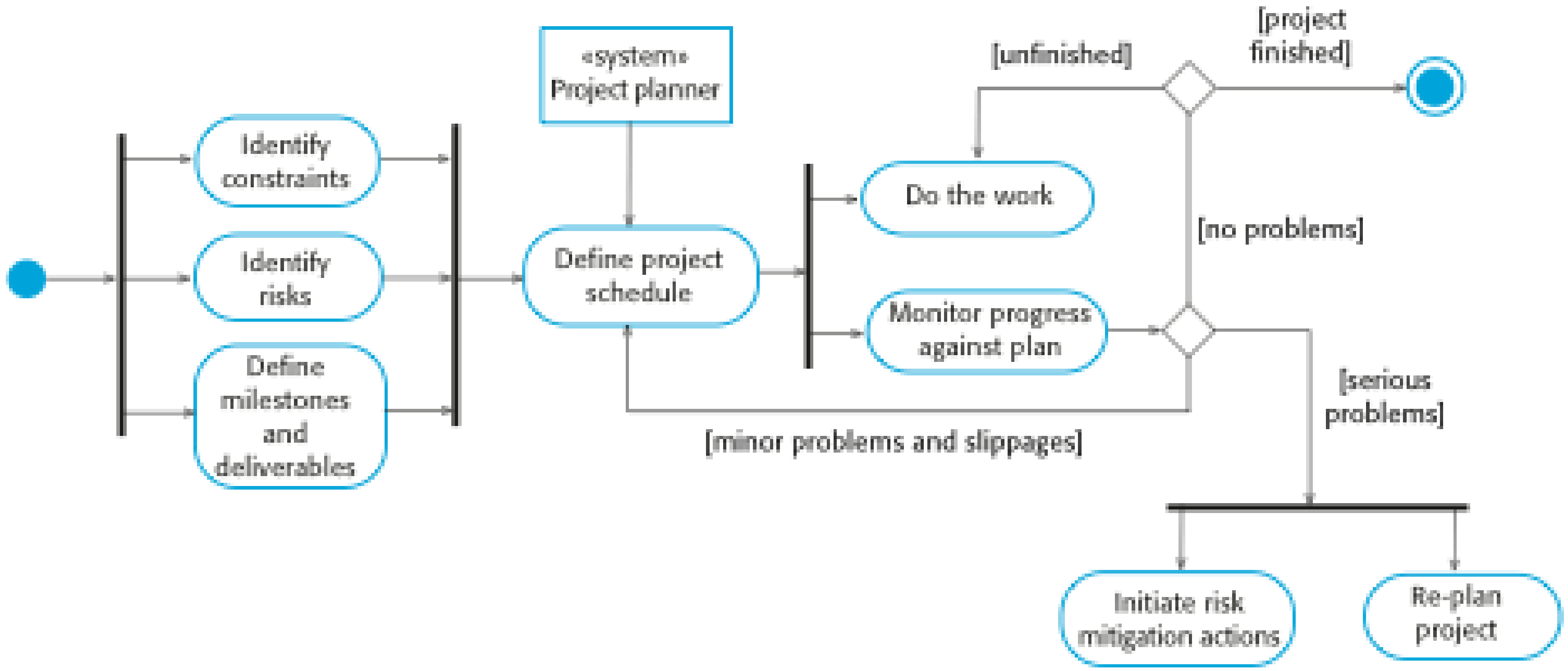
The planning process



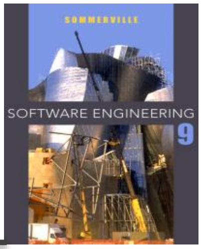
- Project planning is an **iterative process** that starts when you create an initial project plan during the project startup phase.
- Plan **changes are inevitable**.
    - **As more information** about the system and the project team becomes available during the project, **you should regularly revise 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



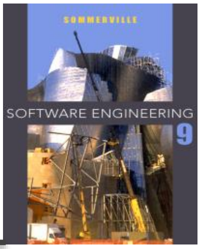


Project scheduling



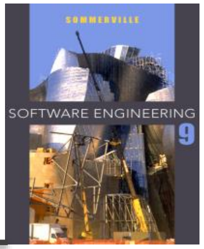
- ✧ Project scheduling is the process of deciding **how the work** in a project will be **organized as separate tasks**, and **when and how** these tasks will be **executed.**
- ✧ You **estimate the calendar time** needed to complete each task, **the effort required** and **who will work on the tasks** 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



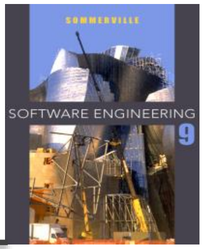
- ✧ **Split project into tasks** and **estimate time and resources** required to complete each task.
- ✧ **Organize tasks concurrently** to make optimal use of workforce.
- ✧ **Minimize task dependencies** to avoid delays caused by **one task waiting for another to complete**.
- ✧ **Dependent on project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

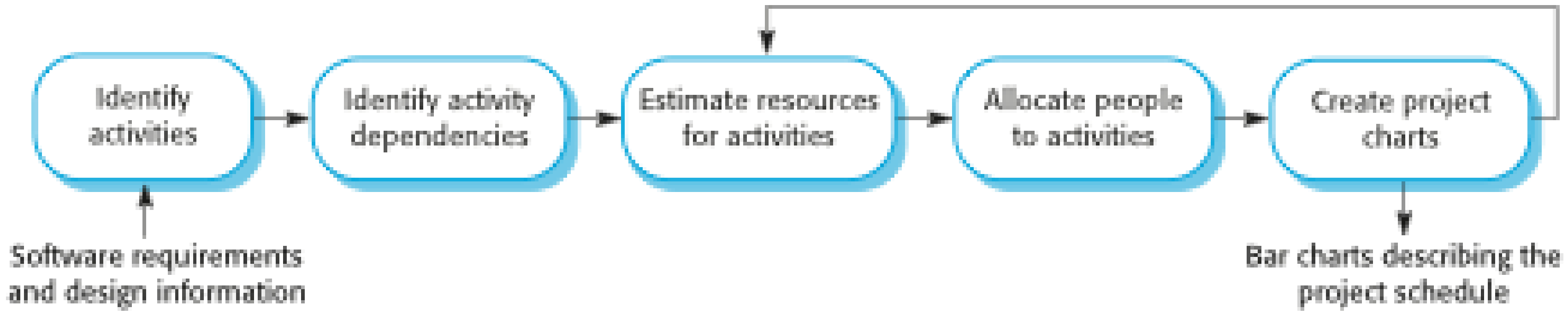
Milestones and deliverables



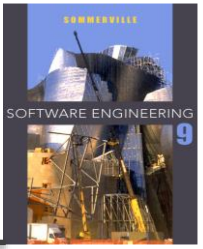
- Milestones are points in the schedule **against which you can assess progress**, for example, **the handover of the system for testing.**
- **Deliverables** are work products that are **delivered to the customer**, **e.g. a requirements document** for the system.

The project scheduling process



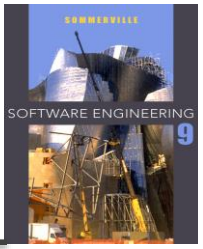

Identify activities
Identify activity dependencies
Estimate resources for activities
Allocate people to activities
Create project charts
Software requirements and design information
Bar charts describing the project schedule

Scheduling problems



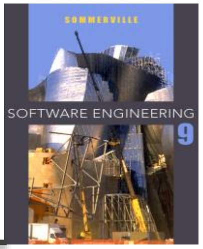
- ✧ **Estimating** the **difficulty** of problems and hence **the cost** of developing a solution **is hard**.
- ✧ **Productivity is not proportional** to the number of people working on a task.
- ✧ Adding people to a late project makes it later because of **communication overheads**.
- ✧ The unexpected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

Schedule representation



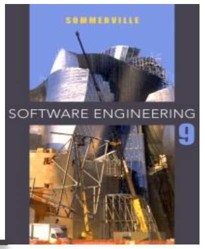
- ✧ **Graphical notations** are normally used to **illustrate the project schedule.**
- ✧ These **show the project breakdown into tasks**.
- ✧ Tasks should **not be too small**.
- ✧ They should **take about a week or two**.
- ✧ **Bar charts are the most commonly** used representation for project schedules.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

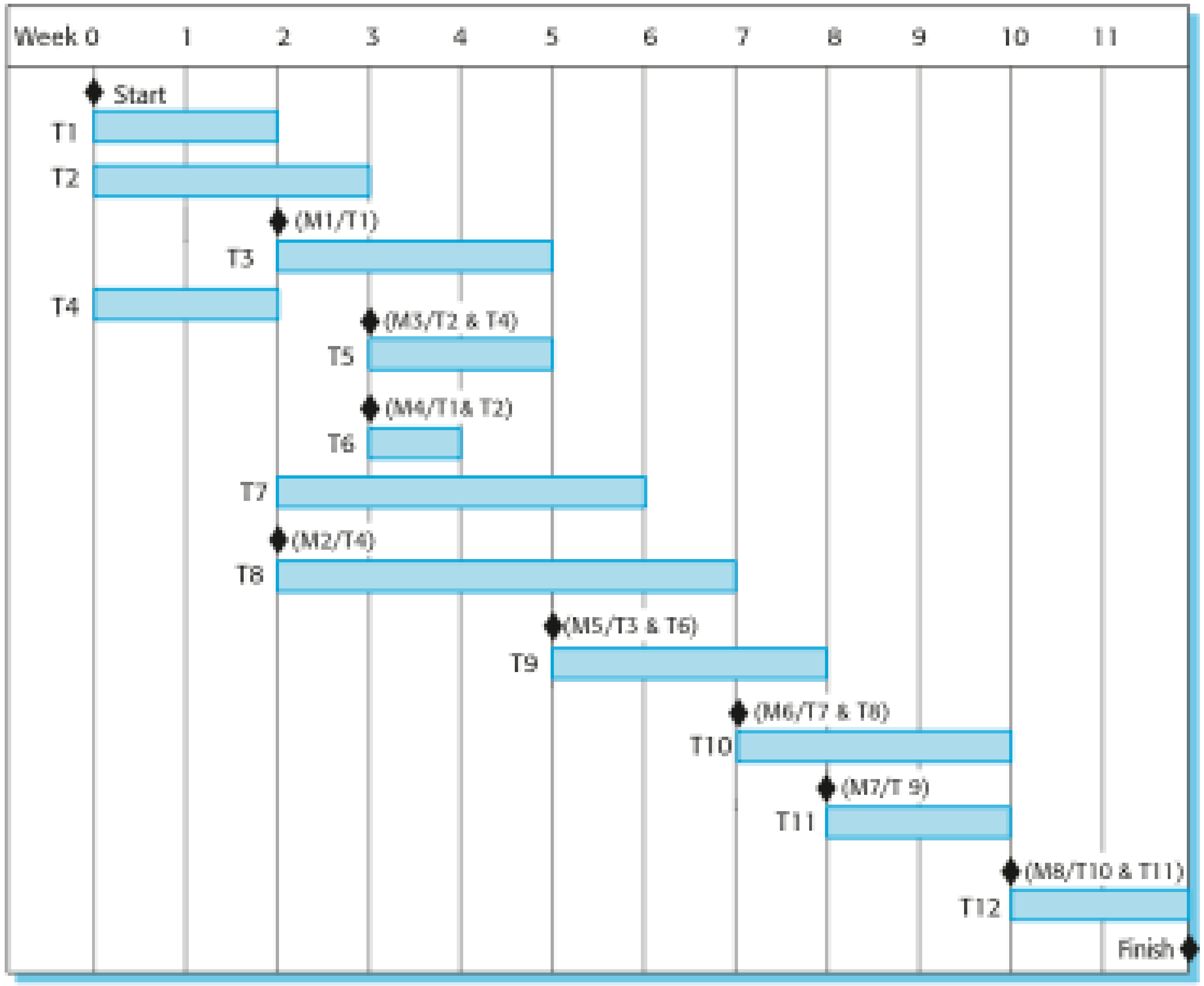
Tasks, durations, and dependencies



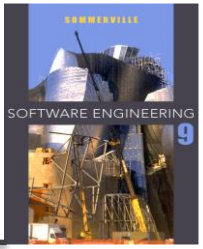
Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

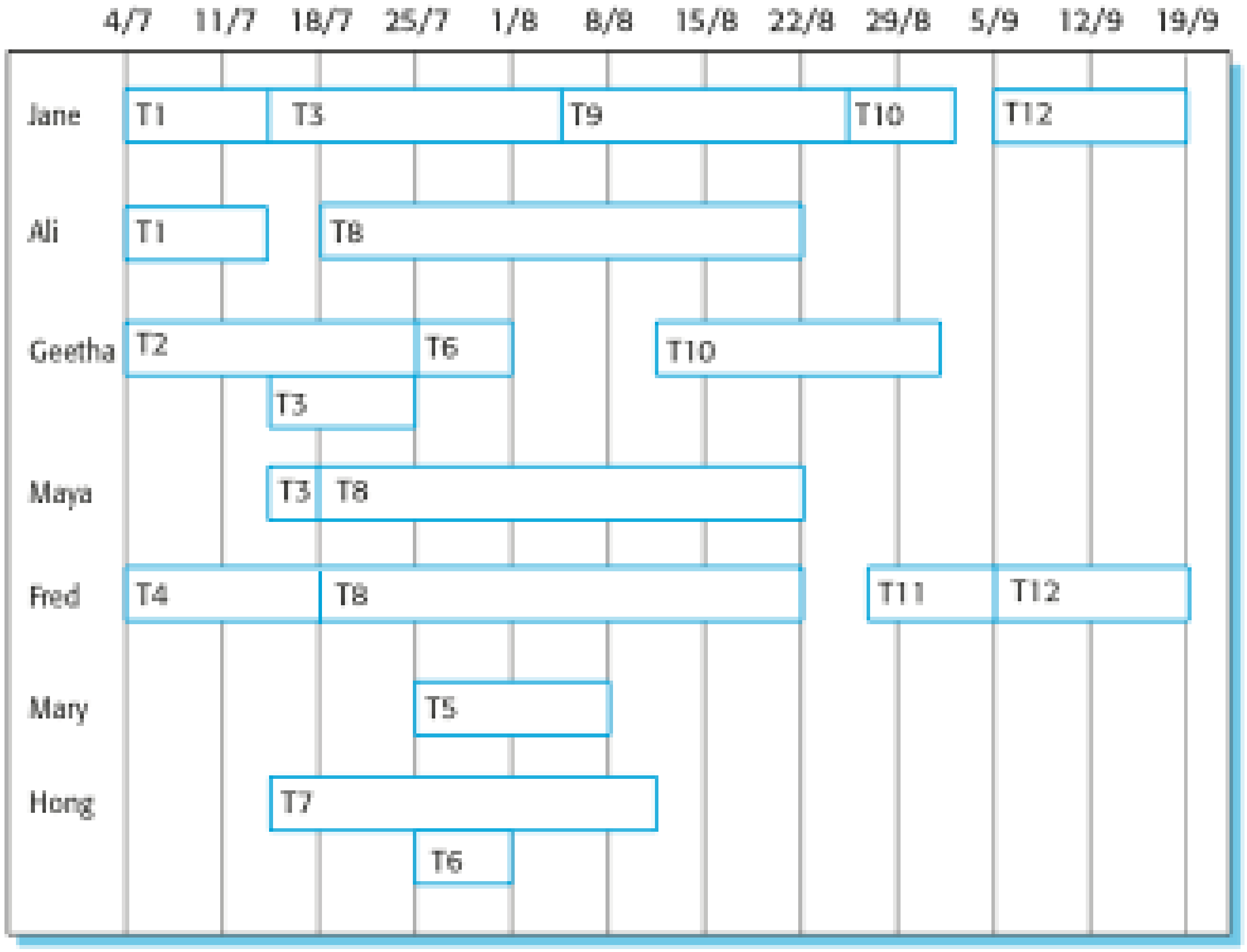
Activity bar chart



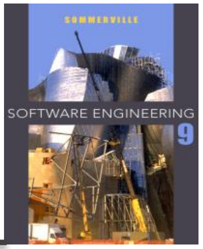


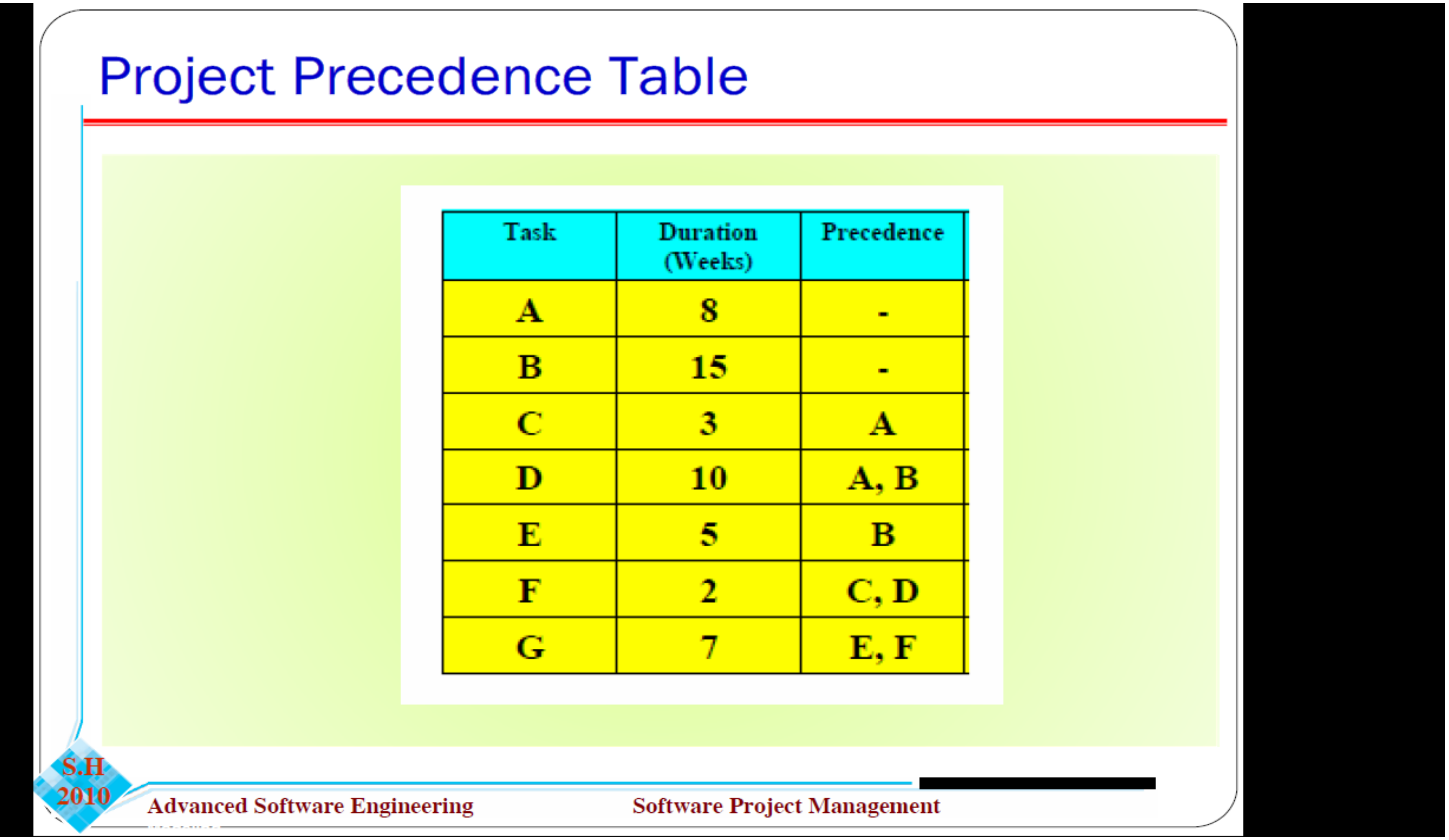
Staff allocation chart





Activity Network and Critical Path





Project Precedence Table

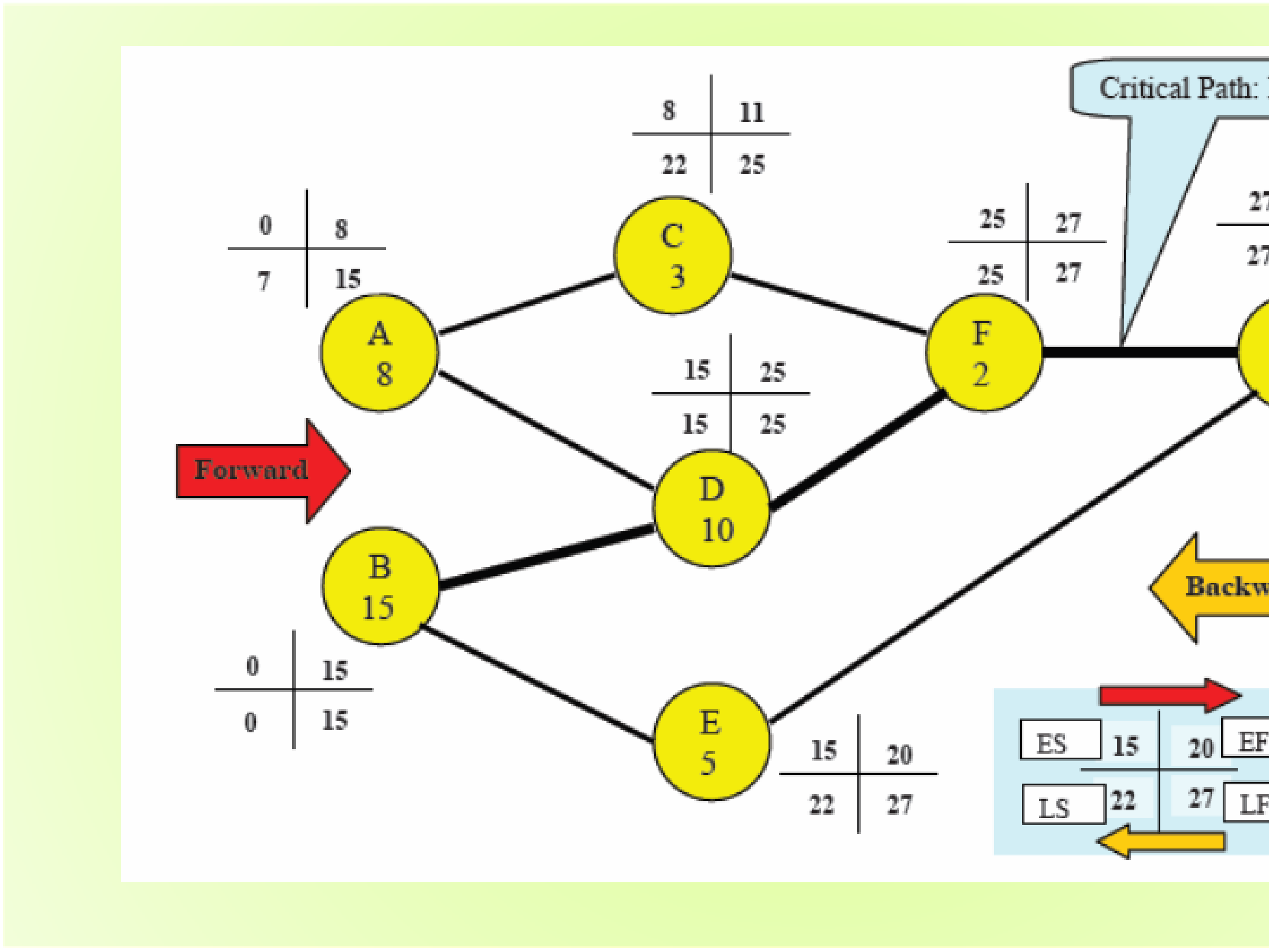
Task	Duration (Weeks)	Precedence
A	8	-
B	15	-
C	3	A
D	10	A, B
E	5	B
F	2	C, D
G	7	E, F

S.H 2010

Advanced Software Engineering

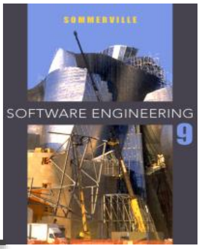
Software Project Management

Activity network – Critical Path



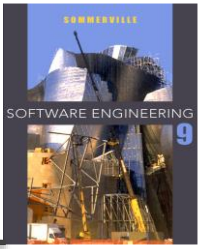
S.H

Agile planning



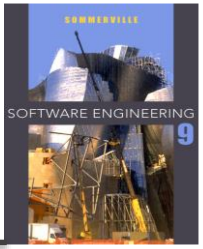
- ✧ Agile methods of **software development are iterative** approaches where **the software is developed and delivered to customers in increments**.
- ✧ Unlike plan-driven approaches, **the functionality** of these increments **is not planned in advance** but is decided during the development.
  - The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

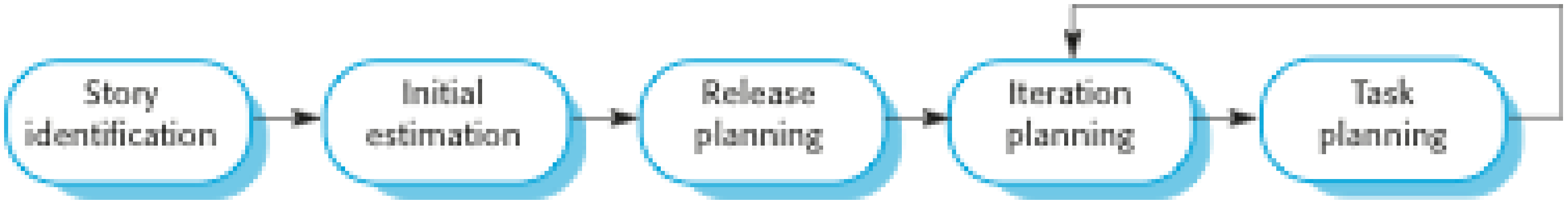
Agile planning stages



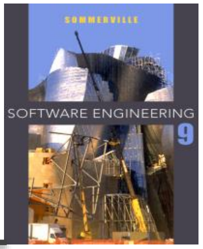
- **Release planning**, which looks ahead for several months and decides on **the features that should be included in a release of a system**.
- Iteration planning, which **has a shorter term outlook**, and focuses **on planning the next increment** of a system. This is **typically 2-4 weeks of work**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

Planning in XP (Extreme programming)



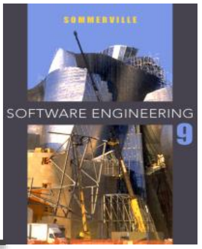


Story-based planning

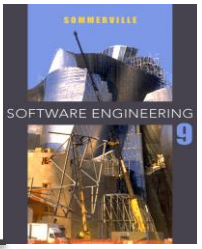


- ✧ The system specification in XP is **based on user stories** that **reflect the features that should be included in the system**.
- ✧ The project team read and discuss the stories and **rank them in order of the amount of time** they think it will take to implement the story.
- ✧ **Release planning** 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).

Key points



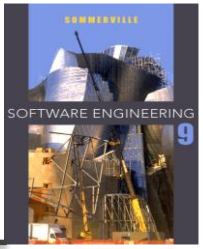
- The price charged for a system does not just depend on its estimated development costs; it may be adjusted depending on the market and organizational priorities.
- Plan-driven development is organized around a complete project plan that defines the project activities, the planned effort, the activity schedule and who is responsible for each activity.
- Project scheduling involves the creation of graphical representations 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, is adjusted. Software functionality is reduced instead of delaying delivery of an increment.



Chapter 23 – Project planning

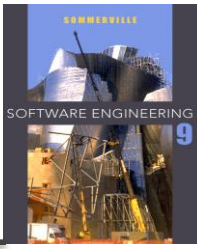
Lecture 2

Estimation is Difficult



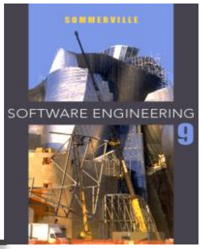
- You may have to make initial estimates on the basis of a high-level user requirements definition.
- The software may have to run on unfamiliar computers or use new development technology.
- The people involved in the project and their skills will probably not be known.
- Project estimates are often self-fulfilling

Estimation techniques



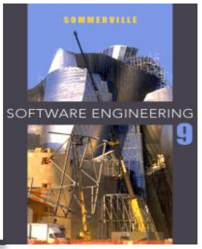
- ✧ Organizations need to make software effort and cost estimates. There are two types of technique that can be used to do this:
  - *Experience-based techniques* The estimate of future effort requirements **is based on the manager's experience** of past projects and the application domain. Essentially, 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



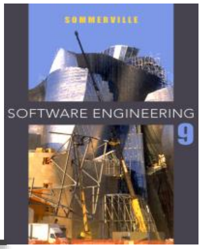
- Experience-based techniques rely on judgments **based on experience of past projects** and the effort expended in these projects on software development activities.
- Typically, **you identify the deliverables** to be produced in a project **and the different software components** or systems that are 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 (**This often **reveals factors** that others have not considered and you then **iterate towards** an agreed group estimate).

The difficulty with experience-based techniques

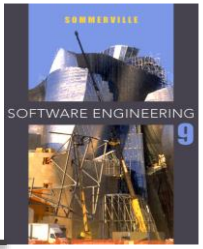


- ✧ a new software project **may not have much in common with previous projects**.
- ✧ Software development changes very quickly
- ✧ and a project will often use unfamiliar techniques such as web services, COTS-based development, or AJAX.
- ✧ If you have not worked with these techniques, your previous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

Algorithmic cost modelling



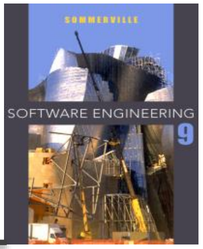
- Cost is estimated as a **mathematical function** of product, project and process attributes whose values are estimated by project managers:
  - Effort = A x Size^B x M
  - A is a constant factor which depends on local organizational practices and the type of software that is developed
  - B usually lies between 1 and 1.5
  - reflects the fact that costs do not usually increase linearly with project size.
  - As the size and complexity of the software increases, **extra costs** are incurred because of the **communication overhead** of larger teams, **more complex configuration management**, **more difficult system integration**, and so on.



- and M is a multiplier reflecting product, process and people attributes.

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

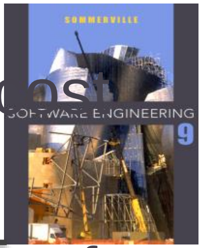
All algorithmic models have similar problems



1. It is often **difficult to estimate Size** at an early stage in a project, when only the specification is available. **Function-point** and **application-point** estimates are easier to produce than estimates of code size **but are still often inaccurate.**
2. The estimates of the **factors contributing to B and M are subjective.**
3. Model users should **calibrate their model** and the attribute values **using their own historical project** data, as this reflects local practice and experience.

However, **very few organizations have collected enough data from past projects** in a form that supports model calibration**.**

✧ **The most commonly used product attribute** for cost estimation **is code size**.

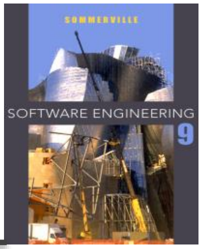


✧ Size may be either an assessment of the **code size** of the software or a **functionality estimate** expressed in function or application points.

✧ **Size estimation may involve**

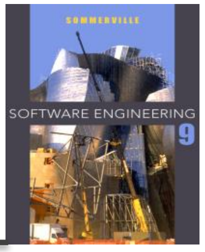
- **estimation by analogy** with other projects,
- estimation **by converting function or application points to code size**
- estimation by **ranking the sizes of system components** and using a known reference component to estimate the component size,
- or it may simply be a question of **engineering judgment**.

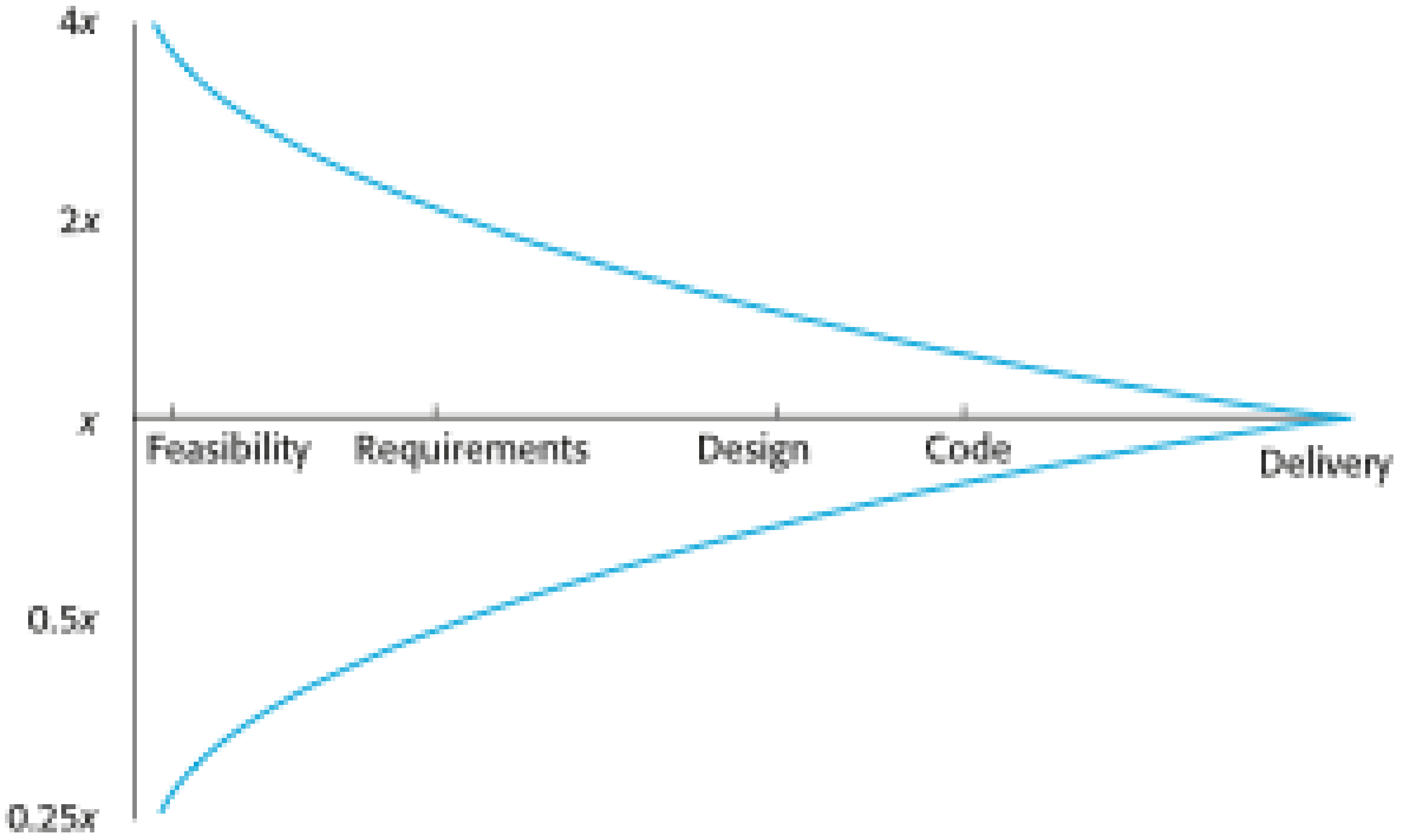
Estimation accuracy



- ✧ 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.

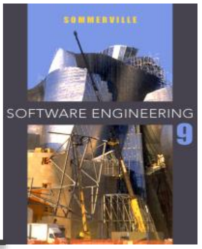
Estimate uncertainty



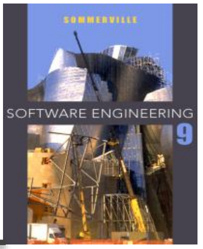


X is the initial estimates in months

The COCOMO II model

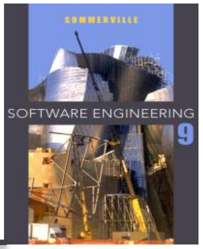


- ✧ This is an **empirical model** that was derived by collecting data from a large number of software projects.
- ✧ These data were **analyzed to discover the formulae** that were the best fit to the observations.
- ✧ These formulae **linked the size of the system and product, project and team factors to the effort to develop the system**.
- ✧ COCOMO II is a **well-documented** and **nonproprietary** (not protected by trademark or patent or copyright) estimation model.



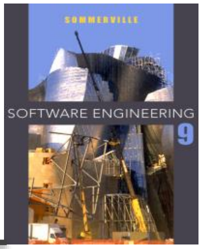
COCOMO vs COCOMO II

- ✧ COCOMO was largely based on **original code development**.
- ✧ The COCOMO II model **takes into account** more modern approaches to software development, such as **rapid development using dynamic languages**, development by component composition, and **use of database programming.**
- ✧ Also, COCOMO II **supports the spiral model** of development

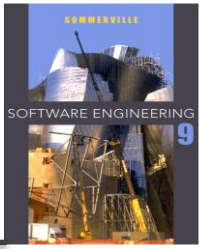


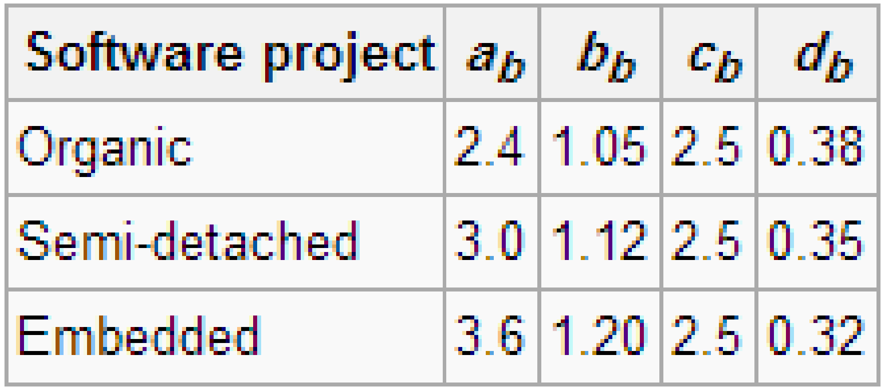
- COCOMO consists of a hierarchy of three increasingly detailed and accurate forms.
- The first level, *Basic COCOMO* is good for quick, early, rough order of magnitude estimates of software costs, but its **accuracy is limited due to its lack of factors to account for difference in project attributes** (*Cost Drivers*).
- *Intermediate COCOMO* takes these Cost Drivers into account
- and *Detailed COCOMO* additionally accounts for the influence of individual project phases.

Basic COCOMO



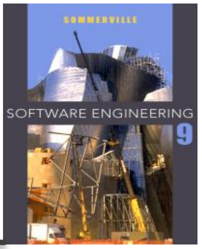
- The basic COCOMO equations take the form
- **Effort Applied (E)** = $a_b(\text{KLOC})^{b_b}$ **[man-months]**
- **Development Time (D)** = $c_b(\text{Effort Applied})^{d_b}$ **[months]**
- **People required (P)** = Effort Applied / Development Time **[count]**
- The coefficients a_b, b_b, c_b and d_b are given in the following table:





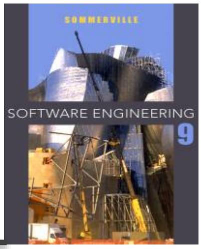
Software project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Three classes of software projects:

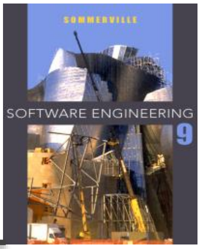


- ✧ Organic projects - "**small" teams** with "**good" experience** working with **"less than rigid" requirements**
- ✧ Semi-detached projects - **"medium" teams** with **mixed experience** working with a **mix of rigid and less than rigid requirements**
- ✧ Embedded projects - developed within a set of **"tight" constraints**. It is also **combination of organic and semi-detached projects**.(hardware, software, operational, ...)

Intermediate COCOMOs

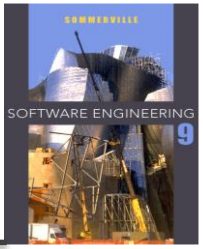


- ✧ Computes software development **effort as function of program size and a set of "cost drivers“**
- ✧ Product attributes
  - Required software reliability
  - Size of application database
  - Complexity of the product
- ✧ Hardware attributes
  - Run-time performance constraints
  - Memory constraints
  - Volatility of the virtual machine environment
  - Required turnabout time



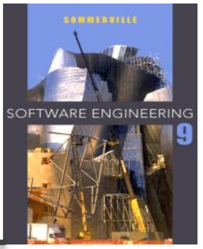
- ✧ Personnel attributes Analyst capability
  - Software engineering capability
  - Applications experience
  - Virtual machine experience
  - Programming language experience
- ✧ Project attributes
  - Use of software tools
  - Application of software engineering methods
  - Required development schedule

Detailed COCOMO



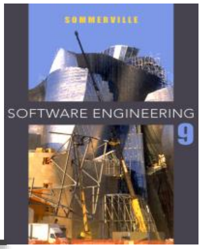
- ✧ Detailed COCOMO incorporates
- ✧ all characteristics of the intermediate version
- ✧ with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.

The COCOMO 2 model



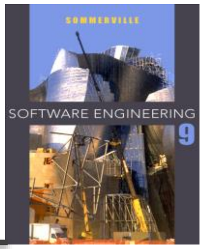
- ✧ An empirical model based on project experience.
- ✧ Well-documented, ‘independent’ model which is not tied to a specific software vendor.
- ✧ Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2.
- ✧ COCOMO 2 takes into account different approaches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

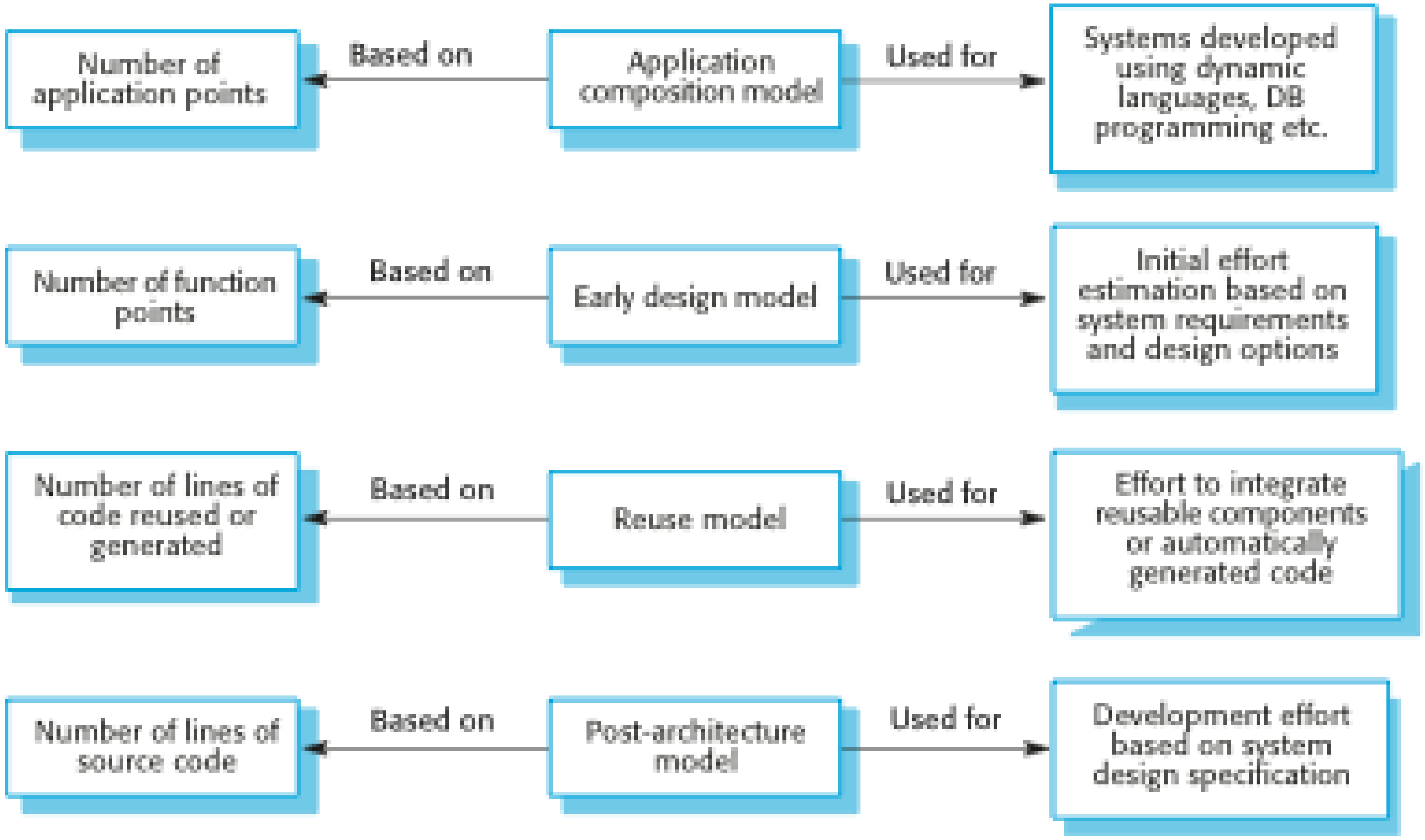
COCOMO 2 models



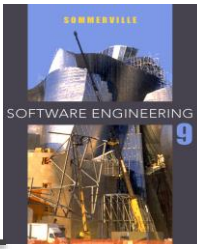
- ✧ COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.
- ✧ The sub-models in COCOMO 2 are:
  - Application composition model. Used when software is composed from existing parts.
  - Early design model. Used when requirements are available but design has 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

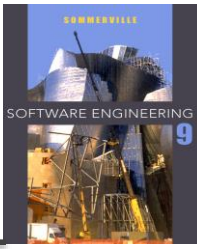



Number of application points
Based on
Application composition model
Used for
Systems developed using dynamic languages, DB programming etc.
Number of function points
Based on
Early design model
Used for
Initial effort estimation based on system requirements and design options
Number of lines of code reused or generated
Based on
Reuse model
Used for
Effort to integrate reusable components or automatically generated code
Number of lines of source code
Based on
Post-architecture model
Used for
Development effort based on system design specification

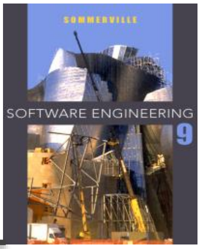
Application composition model



- ✧ This models the effort required to develop systems that are created from **reusable components, scripting, or database programming**.
- ✧ Software size **estimates are based on application points**, and **a simple size/productivity formula** is used to estimate the effort required.
- ✧ The number of **application points** in a program is a weighted estimate of the **number of separate screens** that are displayed, the **number of reports** that are produced, **the number of modules in imperative** programming languages (such as Java), and the number of lines of scripting language or database programming code.

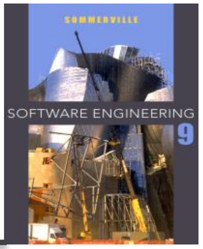


- Supports **prototyping projects** and projects where there is **extensive reuse**.
- **Based on** standard estimates of **developer productivity** in application (object) points/month.
- It is **based on an estimate** of weighted **application points** (sometimes called object points), **divided by** a standard estimate of **application point productivity**.
- The estimate is then adjusted according to the difficulty of developing each application point (Boehm, et al., 2000).

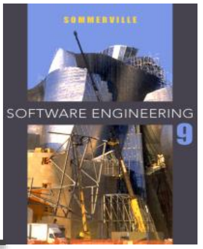


- ✧ **Productivity** depends on **the developer's experience** and **capability** as well as **the capabilities of the software tools (ICASE)** used to support development.
- ✧ Takes **CASE tool used into account**.
- ✧ See table below (next slide)

Application-point productivity

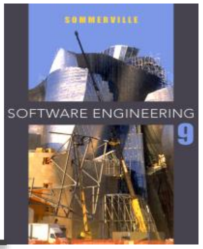


Developer’s experience and capability	Very low	Low	Nominal	High	Very high
ICASE maturity and capability	Very low	Low	Nominal	High	Very high
PROD (NAP/month)	4	7	13	25	50

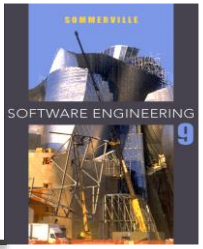


- ✧ Formula is
  - PM = (NAP X (1 - %reuse/100)) / PROD
  - PM is the effort in **person-months** (taken from the table above), NAP is the number of application points and
  - PROD is the productivity (as shown in the table above).
  - “%reuse” is an estimate of the amount of reused code in the development
- ✧ It is almost certain that some of the application points in the system will be implemented using reusable components.
- ✧ Consequently, you have to **adjust the estimate to take into account the percentage of reuse** expected.

Early design model

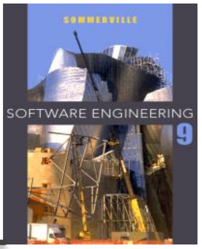


- Estimates can be made **after the requirements have been agreed.**
- **before a detailed architectural** design for the system is available
- **most useful for option exploration** where you need to compare different ways of implementing the user requirements.
- **assumes** that user **requirements have been agreed** and initial stages of the system design process are under way
- make **simplifying assumptions**, for example, that the effort involved in integrating reusable code is zero.



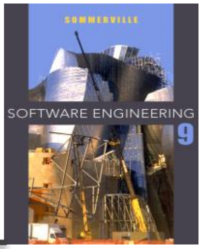
✧ Based on a standard formula for algorithmic models

- PM = A x Size^B x M
- where
- **M = PERS x RCPX x RUSE x PDIF x PREX x FCIL x SCED;**
- A = 2.94 in initial calibration,
- Size in KLOC (or KSLOC),
- You calculate KSLOC by
  - estimating the **number of function points** in the software.
  - You then **use standard tables** that **relate software size to function points for different programming languages**,



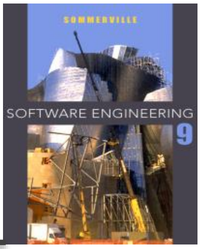
- ✧ B varies from 1.1 to 1.24 depending on **novelty of the project**, **development flexibility**, **risk management approaches** and the process maturity.
- ✧ how the value of this exponent is calculated using these parameters in the description of the COCOMO II post-architecture model
- ✧ This results in an effort computation as follows:
- ✧ PM = 2.94 x Size$^{(1.1 - 1.24)}$ x M

Multipliers:
M = PERS x RCPX x RUSE x PDIF x PREX x FCIL x SCED;

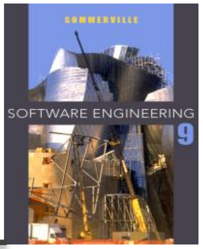


- ✧ Multipliers reflect the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.
  - RCPX - product reliability and complexity;
  - RUSE - the reuse required;
  - PDIF - platform difficulty;
  - PREX - personnel experience;
  - PERS - personnel capability;
  - SCED - required schedule;
  - FCIL - the team support facilities.
- ✧ You estimate values for these attributes **using a six-point scale**, where
- ✧ **1 corresponds to ‘very low’ and 6 corresponds to ‘very high’**
- ✧ See books web page for more details

The reuse model

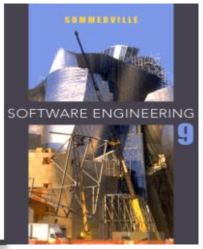


- ✧ Takes into account
  - **black-box code** that is reused without change and
  - **code that has to be adapted** to integrate it with new code.
- ✧ The reuse model is used to **estimate the effort required to integrate reusable or generated code**.
- ✧ There are two versions (two types of reused code):
  - **Black-box reuse** where code is **not modified** (code that can be reused **without understanding** the code or making changes to it).
  - **The development effort** for black-box code is **taken to be zero**.
  - An effort estimate (PM) is computed.



- ✧ White-box reuse where **code is modified** (code has to be adapted to **integrate it with new code or other reused components**).
- ✧ A size estimate equivalent to the number of lines of new source code is computed.
- ✧ This then adjusts the size estimate for new code.
- ✧ Many systems **include automatically generated code** from system models
- ✧ E.g. a model (often in UML) is analyzed and code is generated to implement the objects specified in the model.
- ✧ The COCOMO II reuse model includes a formula to estimate the effort required to integrate this generated code:

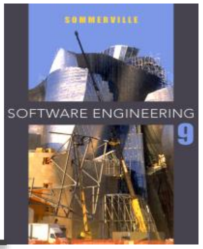
Reuse model estimates 1



✧ For generated code:

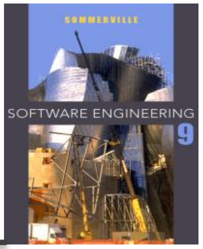
- PM = (ASLOC * AT/100)/ATPROD
- ASLOC is the number of lines of **reused code**, including code that is **automatically generated**.
- AT is the percentage of code automatically generated.
- ATPROD is the productivity of engineers in integrating this code.
- Boehm, et al. (2000) have **measured ATPROD 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:
- (20,000 x 30/100) / 2400 = 2.5 person-months // Generated code

Reuse model estimates 2



- ✧ A separate effort computation is **used to estimate the effort required to integrate the reused code** from other systems
- ✧ **When code has to be understood and integrated:**
  - ESLOC = ASLOC * (1-AT/100) * AAM.
  - ASLOC and AT as before.
  - **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

✧ Simplistically, AAM is the sum of three components:



1. **An adaptation component** (referred to as AAF) that **represents the costs of making changes to the reused code.**

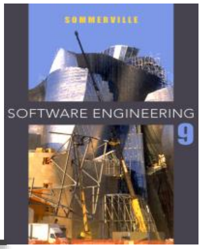
2. **An understanding component** (referred to as SU) that represents the costs of understanding the code to be reused and the familiarity of the engineer with the code.

   SU **ranges from 50** for complex unstructured code to **10 for well-written, object-oriented code**.

3. An assessment factor (referred to as AA) that represents **the costs of reuse decision making**.

   That is, **some analysis is always required** to decide whether or not code can be reused, and this is included in the cost as AA.

   AA varies from 0 to 8 depending on the amount of analysis effort required.

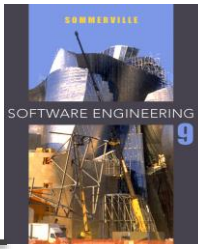


- Once ESLOC has been calculated, you then **apply the standard estimation** formula to calculate the total effort required, where **the Size parameter = ESLOC**.

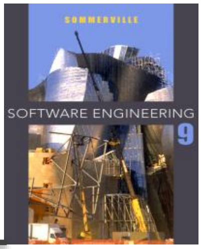
$$\mathbf{PM = A \times Size^{B} \times M}$$

- You then **add this to the effort to integrate automatically generated code** to get **the total effort** required.

Post-architecture level

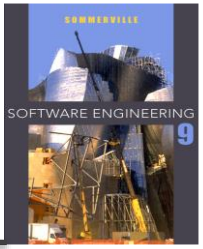


- ✧ The post-architecture model is **the most detailed** of the COCOMO II models
- ✧ **used once an initial architectural** design for the system is available
- ✧ **Once the subsystem structure is known**, you can then **make estimates for each part of the system**.
- ✧ Uses the same formula as the early design model but **with 17 rather than 7 associated multipliers (cost drivers)**.
- ✧ $PM = A \times Size^{B} \times M$



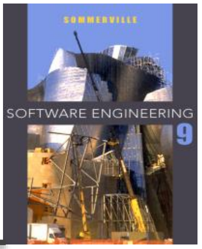
- You make this estimate of the code size using three parameters
  - Number of lines of new code to be developed;
  - Estimate of equivalent number of lines of new code computed using the reuse model;
  - An estimate of the number of lines of code that have to be modified according to requirements changes.
- You 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



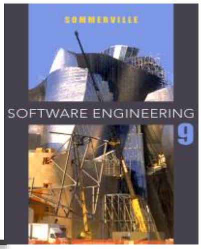
- **The exponent term (B)** in the effort computation formula is related to the **levels of project complexity**.
- **As projects become more complex**, the effects of increasing system size become more significant
- The value of the exponent B is **therefore based on five factors**, as shown in **the table below**.
- These factors are **rated on a six-point scale from 0 to 5**, where **0 means 'extra high'** and **5 means 'very low**'.
- To calculate B, **you**
  - **add the ratings,**
  - **divide them by 100, and**
  - **add the result to 1.01**

Scale factors used in the exponent computation in the post-architecture model



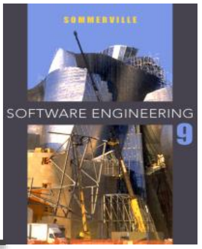
Scale factor	Explanation
Precedentedness	Reflects the **previous experience** of the organization with this type of project. **Very low means no previous experience**; **extra-high means that the organization is completely familiar** with this application domain.
Development flexibility	Reflects the 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.

Example



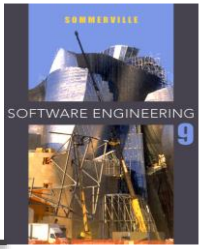
- A company takes on a project in a new domain. The client has not defined the process to be used and has not allowed time for risk analysis. The company has a CMM level 2 rating. A new development team must be put together to implement this system.
  - Precedenteness - new project (4)
  - Development 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.

Multipliers (cost drivers)



- Product attributes
  - Concerned with **required characteristics of the software product** being developed (e.g., reliability, complexity).
- Computer attributes
  - Constraints imposed on **the software by the hardware platform (e.g., memory constraints)**.
- 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 (e.g., tool use, Schedule).**

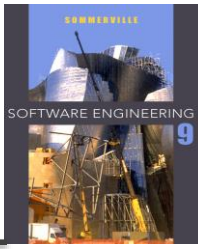
Example Continues



✧ We have assigned **maximum and minimum values** to the key cost drivers to show how they influence the effort estimate.

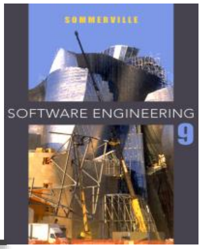
✧ The values taken are those from the COCOMO II reference manual (Boehm, 2000).

The effect of cost drivers on effort estimates

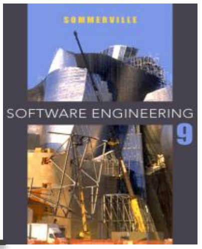


Exponent value	**1.17**
System size (including factors for reuse and requirements volatility)	128,000 DSI
Initial COCOMO estimate without cost drivers	**730 person-months**
Reliability	Very high, multiplier = 1.39
Complexity	Very high, multiplier = 1.3
Memory constraint	High, multiplier = 1.21
Tool use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

The effect of cost drivers on effort estimates

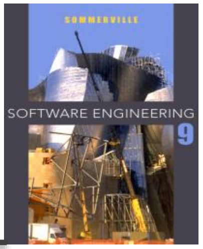


Exponent value	**1.17**
Reliability	Very low, multiplier = 0.75
Complexity	Very low, multiplier = 0.75
Memory constraint	None, multiplier = 1
Tool use	Very high, multiplier = 0.72
Schedule	Normal, multiplier = 1
Adjusted COCOMO estimate	**295 person-months**

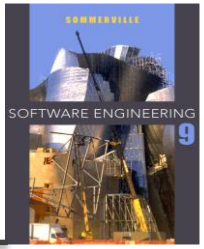


- ✧ You can see that **high values for the cost drivers lead an effort estimate that is more than three times the initial estimate,**
- ✧ **whereas low values reduce the estimate to about one-third of the original.**

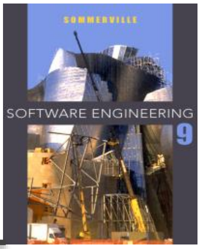
Project duration and staffing



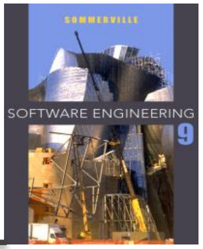
- ✧ As well as effort estimation, managers must estimate the calendar time required to complete a project and when staff will be required.
- ✧ Calendar time can be estimated using a COCOMO 2 formula
  - $\text{TDEV} = 3 \times (\text{PM})^{(0.33+0.2*(B-1.01))}$
  - PM is the effort computation and B 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.



- However, **the nominal project schedule** predicted by the COCOMO model and the **schedule required by the project plan** are **not necessarily the same thing**.
- There may be a requirement to **deliver the software earlier** or (more rarely) later than the date suggested by the nominal schedule.
- If the **schedule is to be compressed**, this **increases the effort required for the project.**
- This is **taken into account by the SCED multiplier** in the effort estimation computation.

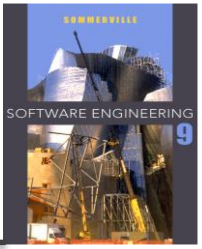


- There is a complex relationship between the number of people working on a project, the effort that will be devoted to the project, and the project delivery schedule.
- If **four peop**le can complete a project in **13 months** (i.e., 52 person-months of effort), then you might think that by adding one more person, you can complete the work in 11 months (55 person-months of effort).
- However, the COCOMO model suggests that you will, in fact, need six people to finish the work in 11 months (66 person-months of effort).



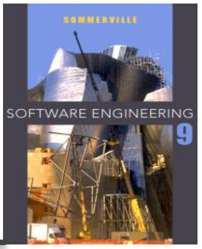
- The reason for this is that adding people actually reduces the productivity of existing team members and so the actual increment of effort added is less than one person.
- As the project team **increases in size**, team members spend more time **communicating and defining interfaces** between the parts of the system developed by other people.
- **Doubling the number of staff** (for example) therefore **does not mean that the duration of the project will be halved**

Staffing requirements



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

Key points



- ✧ 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.