Chapter 23 – Project planning

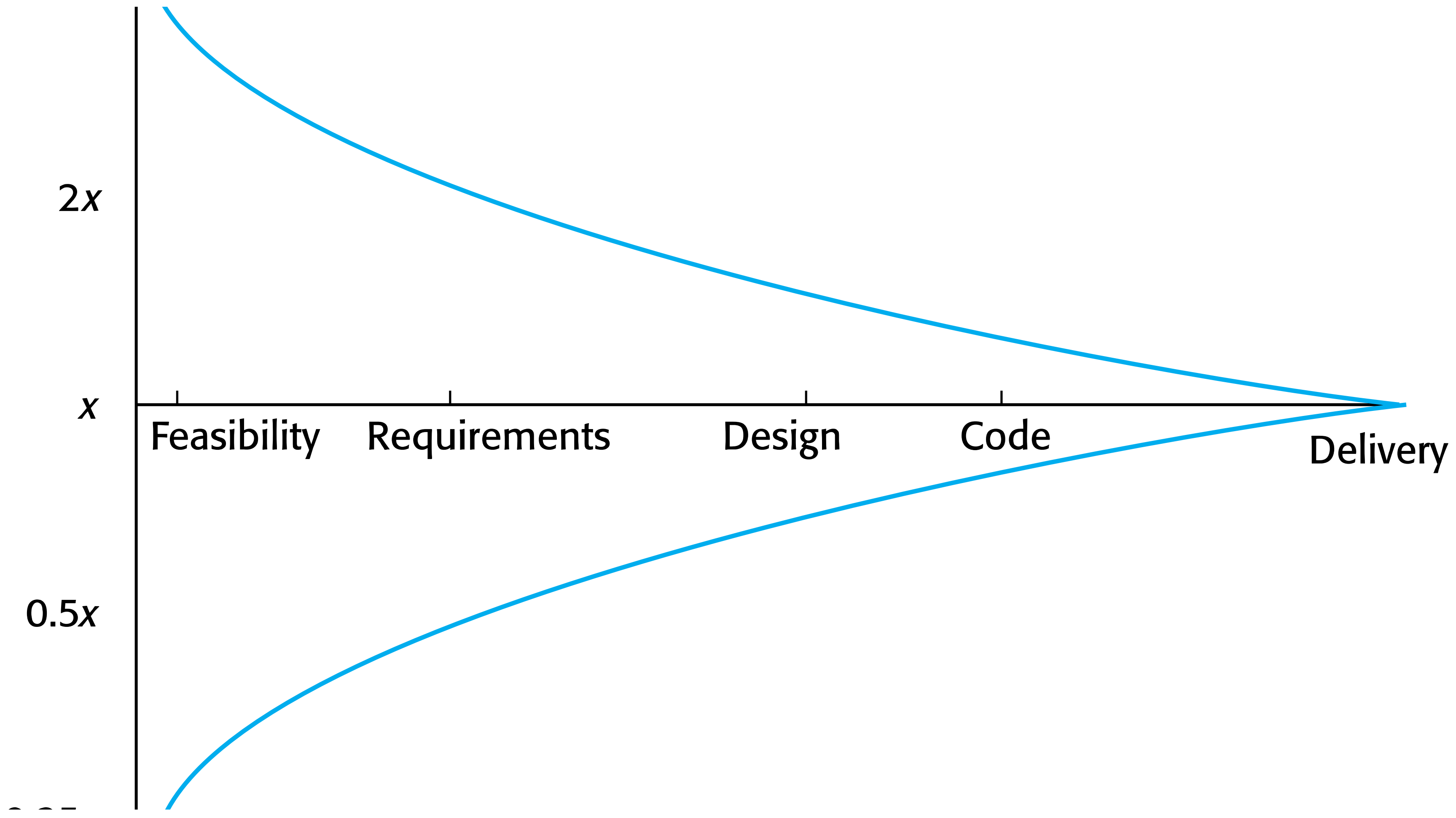
Estimation techniques

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.

Estimate 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.

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 $*$ Size^B $*$ M
  - A is an organisation-dependent constant, B reflects the disproportionate effort for large projects and M is a multiplier reflecting product, process and people attributes.
- ✧ The most commonly used product attribute for cost estimation is code size.
- ✧ Most models are similar but they use different values for A, B and M.

Estimation accuracy

- The size of a software system can only be known accurately when it is finished.
- Several factors influence the final size
  - Use of 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.

Effectiveness of algorithmic models

- ✧ Algorithmic cost models are a systematic way to estimate the effort required to develop a system. However, these models are complex and difficult to use.
- ✧ There are many attributes and considerable scope for uncertainty in estimating their values.
- ✧ This complexity means that the practical application of algorithmic cost modeling has been limited to a relatively small number of large companies, mostly working in defense and aerospace systems engineering.

COCOMO cost modeling

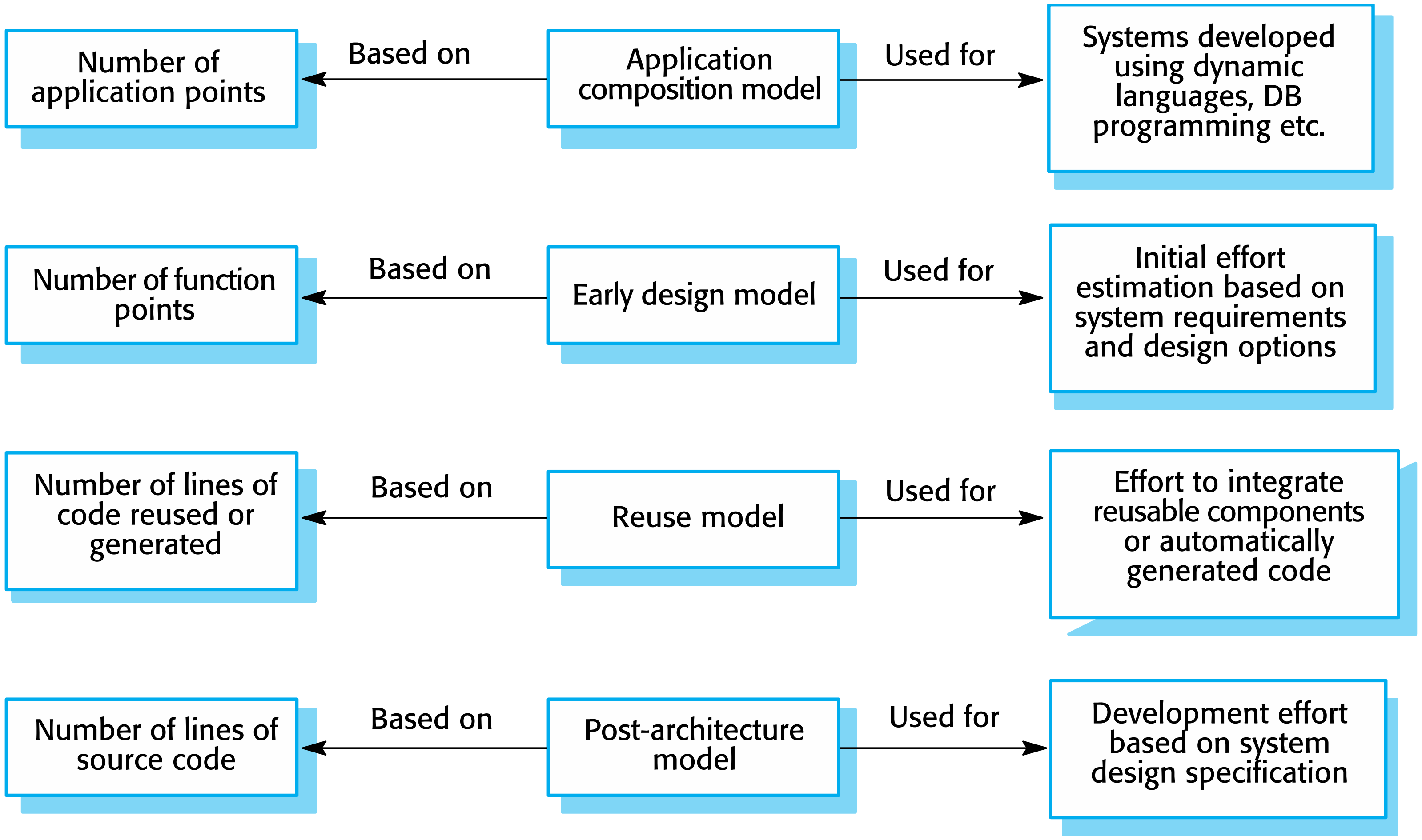
COCOMO cost modeling

- ✧ 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 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

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

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

Early design model

- ✧ Estimates can be made after the requirements have been agreed.
- ✧ Based on a standard formula for algorithmic models
- ✧ $PM = A * Size^{B} * M$ where
  - $M = PERS * RCPX * RUSE * PDIF * PREX * FCIL * SCED$;
  - A = 2.94 in initial calibration,
  - Size in KLOC,
  - B varies from 1.1 to 1.24 depending on novelty of the project, development flexibility, risk management approaches and the process maturity.
  - $PM = 2{,}94 * Size^{(1.1 - 1.24)} * M$

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.

The reuse model

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

Reuse model estimates 1

✧ To estimate the effort required to integrate this generated code:

✧ PM = (ASLOC * AT/100)/ATPROD

- ASLOC is the number of lines of generated code
- AT is the percentage of code automatically generated.
- ATPROD is the productivity of engineers in integrating this code.

✧ ATPROD was measured to be as 2400 lines of 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 :

✧ PM=(20,000 * 30/100) / 2400 = 2.5 person-months

Reuse model estimates 2

- ✧ A separate effort computation is used to estimate the effort required to integrate the reused code from other systems.
- ✧ Based on the number of lines of code that are reused, the model provides a basis for calculating the equivalent number of lines of new code (ESLOC).
- ✧ The following formula is used to calculate the number of equivalent lines of source code:
- ✧ ESLOC = ASLOC * AAM
- ✧ ESLOC is the equivalent number of lines of new source code.
- ✧ ASLOC is the number of lines of code in the components that have to be changed.
- ✧ AAM is an Adaptation Adjustment Multiplier, as discussed below.

Reuse model estimates 2

- Reuse is never free and some costs are incurred even if no reuse proves to be possible. However, reuse costs decrease as the amount of code reused increases.
- The Adaptation Adjustment Multiplier (AAM) adjusts the estimate to reflect the additional effort required to reuse code.
- 1. An adaptation component (referred to as AAF)
  - represents the costs of making changes to the reused code.
- 2. An understanding component (referred to as SU)
  - represents the costs of understanding the code to be reused and the familiarity of the engineer with the code.
- 3. An assessment factor (referred to as AA)
  - represents the costs of reuse decisionmaking.

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

Post-architecture level

- ✧ The post-architecture model is the most detailed of the COCOMO II models.
- ✧ It is used once an initial architectural design for the system is available so the subsystem structure is known.
- ✧ $PM = A * Size^B * M$
- ✧ 1. An estimate of the total number of lines of new code to be developed (SLOC).
- ✧ 2. An estimate of the reuse costs based on an equivalent number of source lines of code (ESLOC), calculated using the reuse model.
- ✧ 3. An estimate of the number of lines of code that are likely to be modified because of changes to the system requirements.

Post-architecture level

- ✧ You add the values of these parameters to compute the total code size, in KSLOC.
- ✧ The exponent term (B) in the effort computation formula is related to the levels of project complexity.
- ✧ The value of the exponent B is therefore based on five factors that are rated on a six-point scale from 0 to 5.
- ✧ 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 to get the exponent that should be used.

Post-architecture level

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

✧ The code size is estimated as:

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

The exponent term

- This depends on 5 scale factors (see next slide). Their sum/100 is added to 1.01
- 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 (0,04+0,01+0,05+0,03+0,03=0,16; 0,16+1.01=1.17).

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

Scale factor	Explanation
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.
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.
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.
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.
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.

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 2,5*(128)1,17=2,5*292=730)
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** **(=730*3,159)**

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 (=730*0,40)**

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

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

- ✧ The price charged for a system does not just depend on its estimated development costs and the profit required by the development company. Organizational factors may mean that the price is increased to compensate for increased risk or decreased to gain competitive advantage.
- ✧ 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.

Key points

- ✧ 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.
- ✧ A project milestone is a predictable outcome of an activity or set of activities. At each milestone, a formal report of progress should be presented to management. A deliverable is a work product that is delivered to the project customer.
- ✧ 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.

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 a mature algorithmic cost model that takes project, product, hardware and personnel attributes into account when formulating a cost estimate.