Systems Engineering

Lecture 3

Cost Estimation

Dr. Joanna Bryson
Dr. Leon Watts

University of Bath
Department of Computer Science

Learning outcomes

After attending this lecture you should be able to:

- Contrast approaches for estimating software project cost, and identify the main sources of cost in a software project.
- Describe three mechanisms for estimating software productivity, and their relative advantages/disadvantages.
- Describe what is meant by the term "person month" and how this metric should be used with care when planning.
- Explain the term "algorithmic cost modelling" and describe the rationale behind such approaches, giving examples.

Cost Estimation

Associating estimates of effort & time with each activity in the project plan.

Total cost of software development is governed by three dominant factors:

- Cost of hardware and software
  - including any maintenance,
- Staff travel and training
  - (can be mitigated using technology) and
- Cost of development effort.

Cost Estimation

Cost of development effort >> salary of software engineers

But also overheads such as...

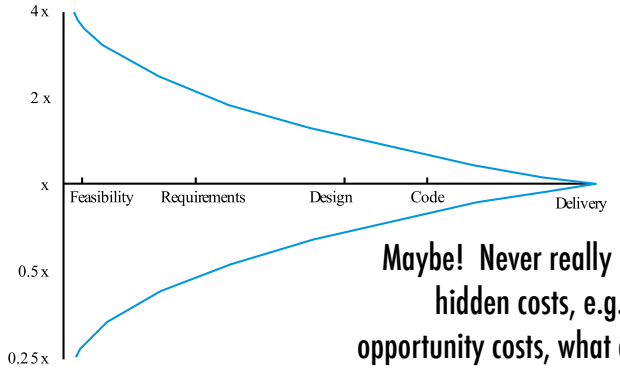
- Facility / Estates
  - Cost of providing heat, light, office space, etc.
- Network and communications
  - Network charge / bandwidth
- Support staff
  - Secretaries, technical, accountants.
- Support resources
  - Library access, software etc.
- Pensions and taxes e.g. National Insurance

Typically 2x Salary

Cost Estimation

- Cost *estimation* is ongoing throughout any software project.
  - Not exact, but must be realistic.
  - If expenditure exceeds budget, actions include:
    - Re-evaluating scope of project,
    - Applying to client for additional resources.
- Price to client may not be only a function of development issues.
  - Business strategy may play a role e.g. loss-leading.

Cost Estimate Uncertainty



Maybe! Never really know hidden costs, e.g. opportunity costs, what exact % time people work on project, human capital impact.

How to estimate Cost?

- Difficult to know what we are building early on.
- In most cases project costs are estimated based on previous, subjective experience:
  - Estimate by analogy.
  - Expert judgement.
  - Market based, e.g. self-fulfilling costs; contract-winning; quality-signalling.
  - Algorithmic approaches (more on this later...)
    - "Objective"...
- Best to employ several approaches and compare.

Software Productivity

- Productivity is typically measured using:

  Productivity = output / total development effort
- Software is intangible, so output is difficult to quantify.
- Two approaches are common:
  - Volume-related metrics
    - Material output by development team in some unit / time e.g. lines of code / week.
  - Functionality-related metrics
    - Amount of desired functionality developed / time.
- Such measures are **not** quality oriented!

1. Lines of Code (LOC/pm)

- What's a line of code?
  - First proposed with punch cards;
  - In Java a statement can span several lines or there can be several statements on one line.
- Expressiveness of language can give misleading results

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

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

Papers on Comparative Productivity

- These are linked from the moodle page.

2. Function Point Counts

- Identify "function points" as I/O operations
  - external inputs and outputs;
  - user interactions;
  - external interfaces;
  - files used by the system.
- Each point is weighted according to complexity (derived through experience)
- Weighted summation / development duration = productivity
- Advantage
  - Language independent (avoids LOC problem).
    - Can be transformed to LOC based on language specific average LOC for a func point.
- Disadvantages
  - Subjective weightings. Biased to data processing applications.

3. Object Points

- Object points (or application points)
  - 4GLs or similar – Object points are NOT the same as object classes.
- Number of object points in a program is a weighted estimate of:
  - The number of separate screens that are displayed;
  - The number of reports that are produced by the system;
  - The number of program modules that must be developed to supplement the database code.
- Higher level, so closer mapping to requirements.
  - Little dependency on implementation so easier to estimate early on in the design process.

Factors affecting Software Productivity

- Size of project, or of team.
  - Communication overhead; Integration cost.
- Prior experience of a domain.
  - Personnel make a big difference esp. in small teams.
- Support teams/resources.
- Working environment, including software & language.
- Suitability and quality of software process model adopted.

Algorithmic Cost Modelling

- Cost (effort) is estimated as a mathematical function of product, project and process attributes.
- Most algorithms are elaborations on an exponential relationship between project size and effort, e.g.:

$$\text{Effort} = A \times \text{Size}^B \times M$$

  - **A** is an organisation-dependent constant (historical adjustment)
  - **M** is a multiplier reflecting product, process and people attributes.
  - **B** is the empirical exponent: 1-1.5, relates to the complex, the size, and the difficulties of the software. The larger the system, the larger the value.

Algorithmic Cost Modelling

$$\text{Effort} = A \times \text{Size}^B \times M$$

- **A** is an organisation-dependent constant (historical adjustment)
- **M** is a multiplier reflecting product, process and people attributes.
- **B** is the empirical exponent: 1-1.5, relates to the complex, the size, and the difficulties of the software. The larger the system, the larger the value.

- How do we measure "Size"? Code size, FP, or OP.
- How do we estimate constants of proportionality?
  - Compute best, worse and average cases.
  - Subjective (still!) Based on experience.

COCOMO 81 Examples

Boehm (1981) *Software Engineering Economics*

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

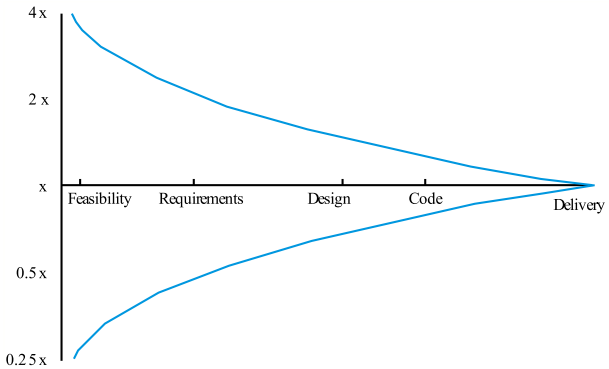
COCOMO

- COnstructive COst MOdel
  - Widely used algorithmic model. Public domain.
- Originally developed by Boehm in early 80s
  - COCOMO 81
    - Waterfall based, no OO.
    - Assumes all software developed from scratch.
  - COCOMO II (c. 2000) supports spiral (iterative) model.
- Empirically derived model
  - 63 software projects at TRW for COCOMO 81, 2-100K LoC.
- COCOMO II supports 4GLs, re-use and iterative based software processes.

COCOMO II

- Incorporates a range of sub-models that produce increasingly detailed software estimates.
- The four sub-models in COCOMO II are:
  - Application composition model. Used when software is developed using a prototyping approach, linking together existing components.
  - Early design model. Used when requirements are available and design has just 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.

Cost Estimate Uncertainty



1. Early Design Model

- A cheap (to compute) estimation method when
  - little design detail is available, but requirements are agreed.
- Based on a standard formula for algorithmic models
  - **PM = A × Size^B × M**
  - M = Product of 7 variables:
    - Product Reliability and Complexity (RCPX)
    - Reuse required (RUSE)
    - Platform difficulty (PDIF)
    - Personnel capability (PERS)
    - Personnel experience (PREX)
    - Schedule (SCED)
    - Support facilities (FCIL)

    } Score 1-6
  - A = 2.94 in initial calibration
  - Size estimated using function points

2. 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 re-use and CASE tools into account.
- Formula is:

  PM = NAP (1 - %reuse/100) / PROD

  - PM is the effort in person-months
  - NAP is the number of application (object) points
  - PROD is the productivity (NB: ref to the next slide)

3. Re-Use Model

- Takes into account black-box code and white-box code
- There are two versions:
  - Black-box reuse where code is not modified. The effort estimate (PM) is zero.
  - White-box reuse where code is modified. A size estimate equivalent to the number of lines of new source code is computed.
    - Productivity metric used to convert size to PM as in *application-composition*

4. Post-architecture model

- Most detailed COCOMO II Model
  - Used when subsystems identified (post A.D.)
- Uses the same formula as the early design model but with 17 rather than 7 associated multipliers.

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

- Size is estimated as:
  - Number of lines of new code to be developed;
  - Estimate of equivalent lines of new code computed using the reuse model (ESLOC)
  - An estimate of the number of lines of code that have to be modified according to requirements changes.
- B computed as discussed earlier (Early Design Model)

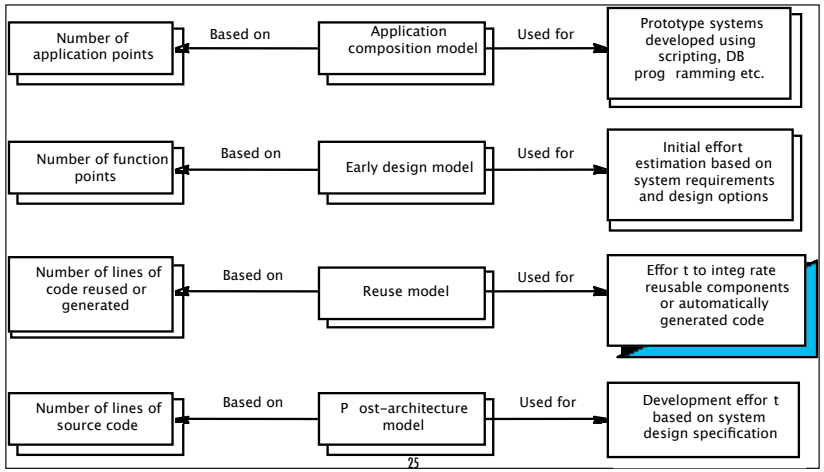
17 Post-architecture cost drivers: M

- Product
- Computer
- Personnel
- Project

[Sommerville, pp. 633]

Attribute	Type	Description
RELY	Product	Required system reliability
CPLX	Product	Complexity of system modules
DOCU	Product	Extent of documentation required
DATA	Product	Size of database used
RUSE	Product	Required percentage of reusable components
TIME	Computer	Execution time constraint
PVOL	Computer	Volatility of development platform
STOR	Computer	Memory constraints
ACAP	Personnel	Capability of project analysts
PCON	Personnel	Personnel continuity
PCAP	Personnel	Programmer capability
PEXP	Personnel	Programmer experience in project domain
AEXP	Personnel	Analyst experience in project domain
LTEX	Personnel	Language and tool experience
TOOL	Project	Use of software tools
SCED	Project	Development schedule compression
SITE	Project	Extent of multisite working and quality of inter-site communications

The four COCOMO II sub-models



Summary: with some reading you should be able to...

- Describe the activities in a typical project planning cycle.
- Identify dependencies between project tasks, and analyse these to determine critical paths in a project.
- Schedule tasks in a software project using conventional charts and notations.
- Contrast approaches to estimate software project cost, and identify the main sources of cost in a software project.
- Describe three mechanisms for estimating software productivity, and their relative advantages/disadvantages.
- Describe what is meant by the term "person month" and how this metric should be used with care when planning.
- Explain the term "algorithmic cost modelling" and describe the rationale behind such approaches, giving examples (COCOMO).

Further Reading

- "The Mythical Man Month"
  - Fred Brooks Jr. ISBN: 020-183-595-9
- "Software Engineering Economics"
  - Barry Boehm. ISBN: 013-822-122-7
- Next week: Quality management
  - Sommerville, Ch.27.