EECE 310

Software Cost Estimation

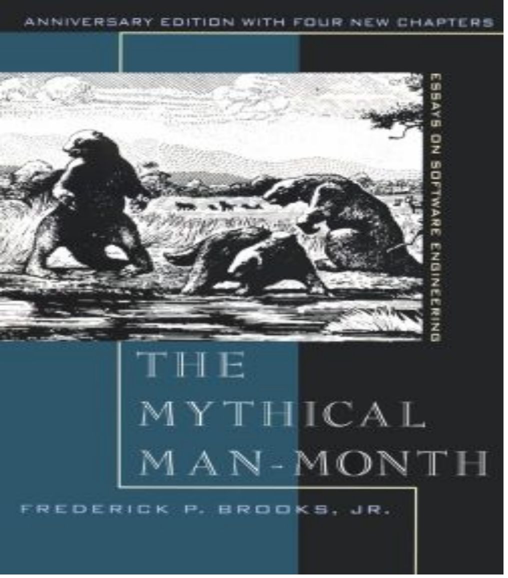
Software Hall of Shame

More than 30% of software projects fail.

e.g., Denver baggage system,
healthcare.gov

Personal experience

- Croptimizer: build an **automated system** for managing green-houses
- Control: temperature, humidity, light, water
- Calculate optimal values (complex model)


ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.

The Mythical Man-Month

- Fred Brooks' observations based on his experiences at IBM

- His book is called "**The Bible of Software Engineering**", because "*everybody quotes it, some people read it, and a few people go by it*.”

- What did he find in his observations?

Failure to meet schedule is the reason for most software project failures. Why?

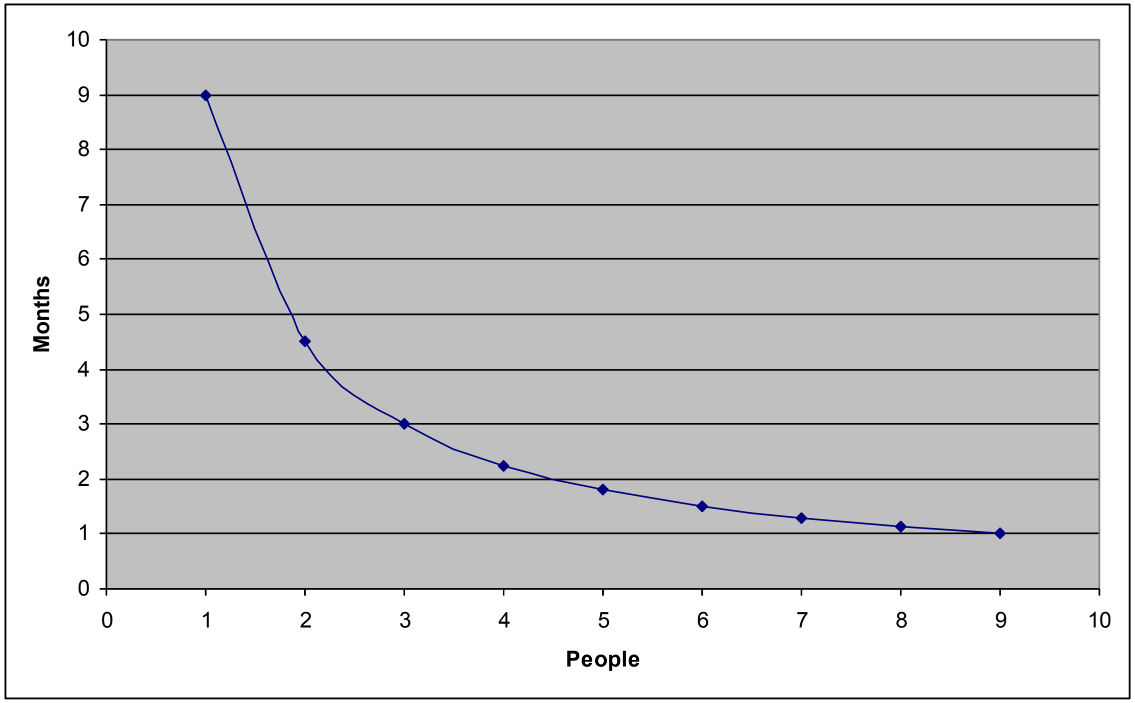
- We don’t know how to *estimate* (overly optimistic, assume all will go well).
- We give in to pressure to reduce time estimates because we are uncertain of our estimates.
- We don’t **monitor** schedule progress properly.
- We confuse effort with progress (we think men and months are interchangeable)

Question

- You are the manger of a big software project
- You are half way through the project and you know you are **late** to meet the deadline (which is in 2 months).

- What do you do?

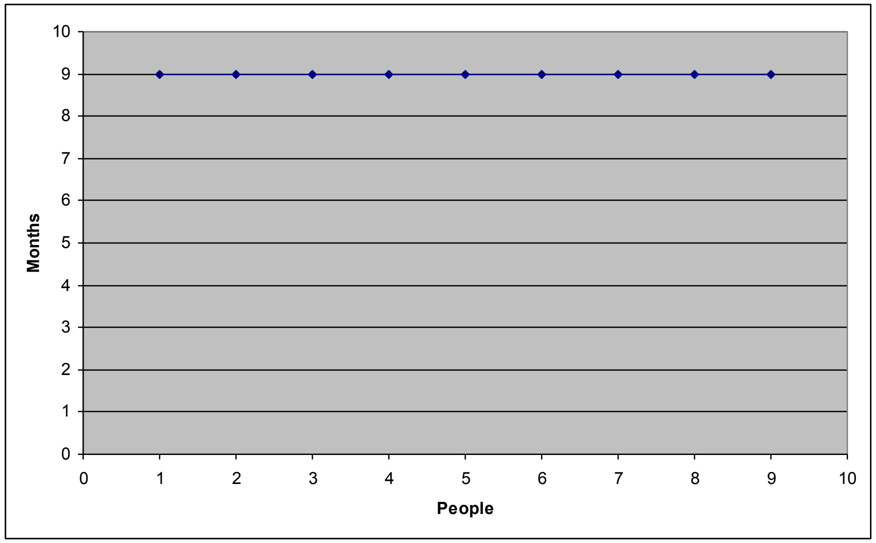
Progress vs. Cost, 1



harvest

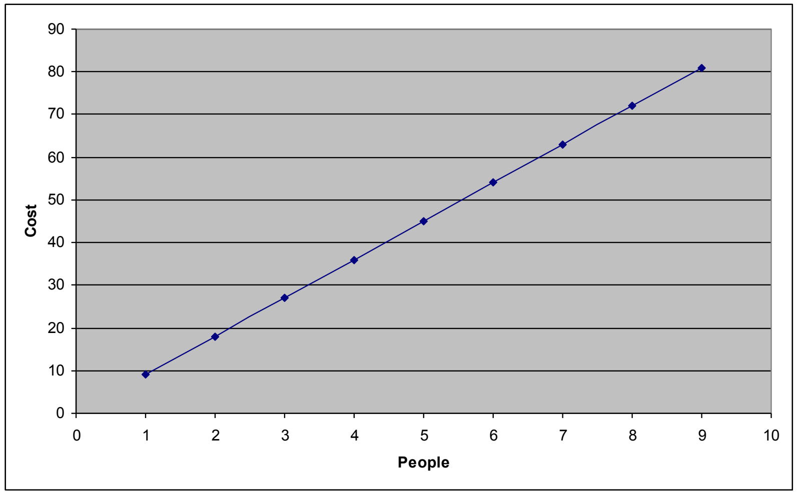
When there is no dependency among people, the amount of time to do a task diminishes with each new person. Note that the cost (people * months) is constant.

Progress vs. Cost, 2

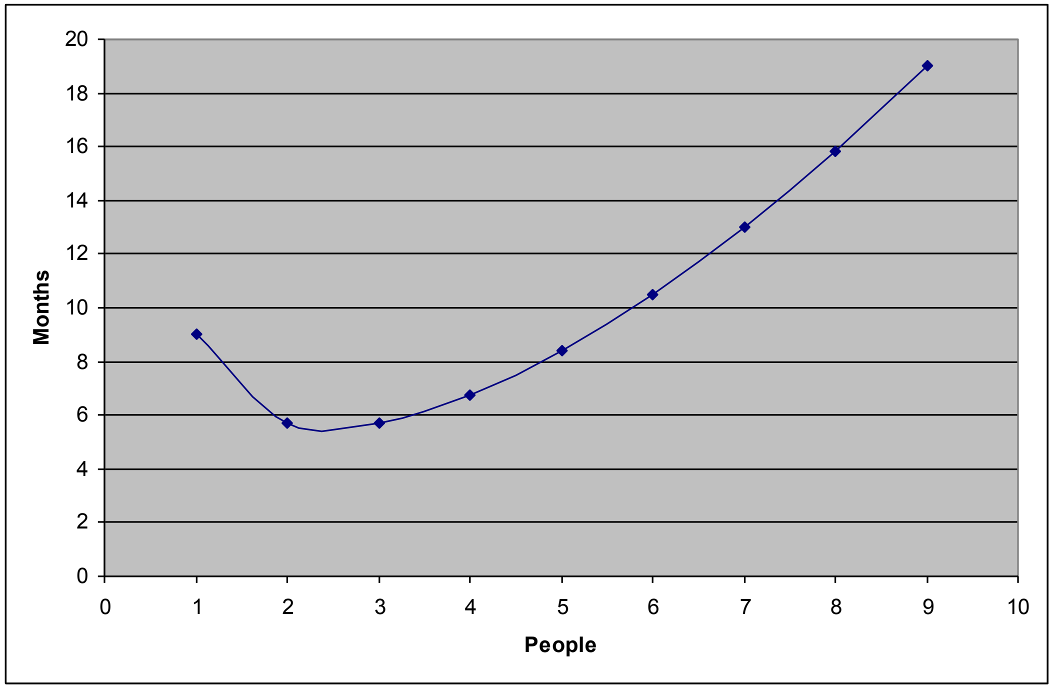


When there is a dependency and the task cannot be partitioned, adding people has no effect on time required…

…but it has a big effect on cost.



Progress vs. Cost, 3



If task can be partitioned, but requires communication, must handle training and communication as each person is added. Can cause project to be late.

Brook’s law: Late projects

- Assigning **more programmers** to a project running behind schedule will make **it even later**
- time required for the new programmers to learn about the project
- the increased **communication overhead**.
  - Group Intercommunication Formula: $n(n - 1) / 2$
  - Example: 50 developers -> 50(50 – 1) / 2 = 1225 channels of communication

What is software cost estimation?

predicting the **most realistic** amount of **effort** required to **develop** software based **on incomplete, uncertain** and **noisy input**.

Fundamental estimation questions

- How much calendar **time** is needed to complete an activity?
- How many **people** is needed?
- What is the total **cost** of an activity?
- How to make a **competitive offer** to the customer?
- How to **plan** and **manage** this project?

Why Estimate Software Cost and Effort?

- To determine (economic) feasibility
  - Costs versus benefits
- To provide a basis for agreeing to a job:
  - You must make a business case for taking on a job or set thresholds for negotiating a price for performing the job.
- To make commitments that you can meet:
  - Do we have enough man power or resources?
  - Avoid cost overruns that may cause customer to cancel project
  - Avoid cost overruns that the development team swallows

Software Cost Estimation

- How long would it take to write a test suite for a Java class with 6 methods and 80 LOC?
- How long would it take you to implement the UNDO feature in JPacman?
- How long would it take to develop a new ABS software module for Toyota?

Why is Estimating Difficult?

Unfortunately, it is very difficult to estimate the cost and effort to build a project when you don’t know very much about that project.

- We’re not estimating **repeatable**, **objective** phenomena.
- The earlier the estimate (e.g., requirements phase), the less is known about the project.
- Unlike, say, building bridges, most of the time and effort in software development is in creating new designs.

Estimate the following

(1) Developing a web-based calendar.

- (Google Calendar was built in **5 days with 4 people**)

(2) Developing a new ABS software module.

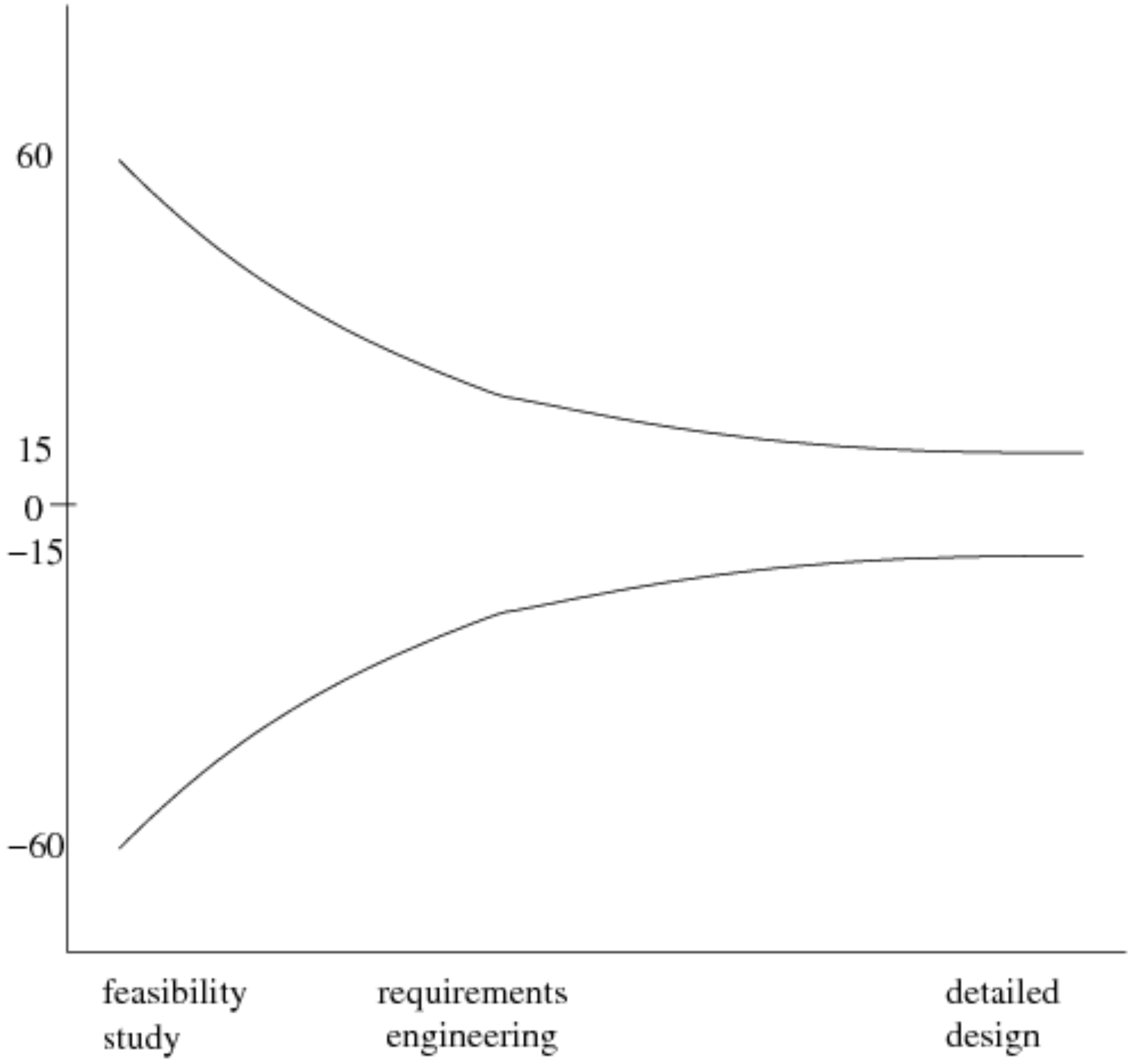
- (Honda's ABS software was created in **180 days with 26 people**)

Anchoring

(Mis)leading clues in the estimation process, which you will use as an anchor to revolve around!

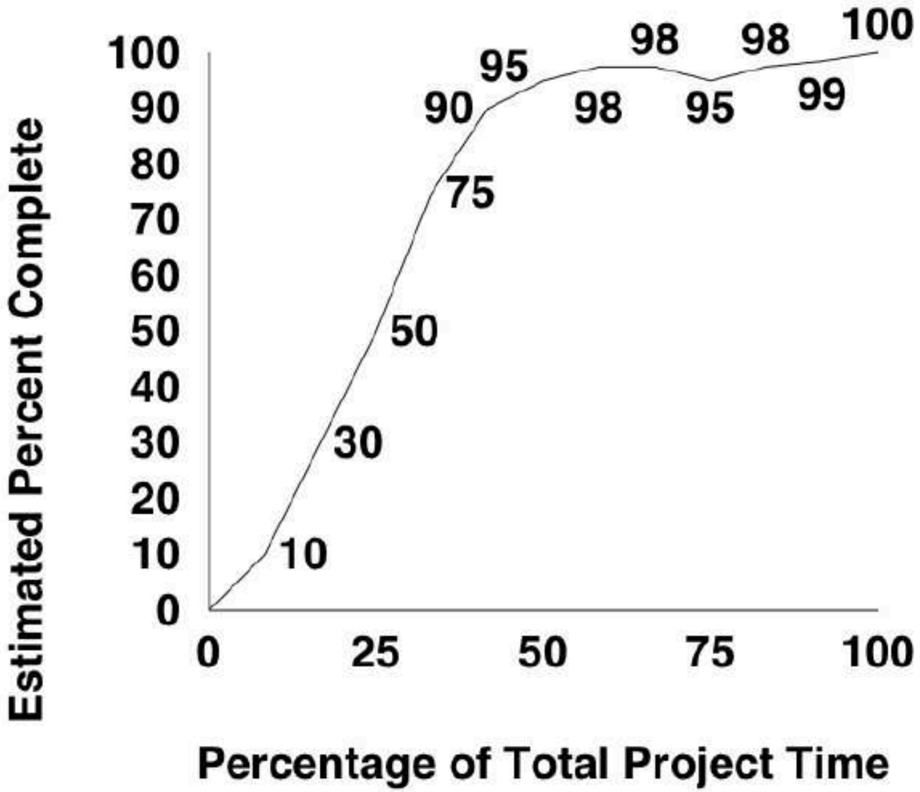
Estimates can be **biased** by business and other pressures.

Cone of uncertainty



Accuracy of Estimates

By the time we know enough to estimate a project's cost to within ***10%*** of its actual cost, the product is almost complete.



Bad Estimators

Price-to-Win

- Bidding as low as possible to beat the competition and win the contract.
- Results:
  - not enough money for resources needed for development,
  - no profit
- This can hardly be called an estimation technique.
- The idea is to price low enough to *win* the contract, but high enough to show a *profit*.

How would you estimate?

- More systematic than gut-feeling
- Techniques?

Estimation Approaches

- Empirical Estimation
- Algorithmic Estimation

Empirical Estimation

- Estimate is based on experience and historical data
- Involves experts in
  - Development techniques
  - Application domain
- Most common technique in practice
- Example: Delphi method

Delphi Method

Delphi methods are based on expert judgment.

- Each expert submits a **secret prediction**, using whatever process he or she chooses.
- The **average estimate** is sent to the entire group of experts.
- Each expert **revises** his or her prediction privately. In some variations of the Delphi method, the experts discuss their rationales before new estimates are made, justifications are circulated anonymously, or no discussion is allowed.
- **Repeat until no expert wants to revise** his or her estimate, i.e., until a fixed point is reached.

Critical Points about Delphi

- Its success depends on the experts' abilities to remember and determine which past projects are similar and in which ways.
- An expert's experience cannot be transferred to junior developers.
- Project with new technologies might not be similar to previous projects.

Planning poker

- An iterative (agile) approach to Delphi
- Steps
  - Each estimator is given a **deck of cards**, each card is filled with an **estimate**
  - Product owner reads a story and it's discussed briefly
  - Each estimator selects a card that's his/her estimate
  - Cards are turned over so all can see them
  - Discuss differences (especially outliers)
  - Re-estimate until estimates converge

Planning poker—an example

Estimator	Round 1	Round 2
Erik	3	5
Martine	8	5
Inga	2	5
Tor	5	8

Here

Estimation Approaches

- Empirical Estimation
- Algorithmic Estimation

Algorithmic Estimation

Algorithmic estimation is based on

- Cost **model**, represented by (regression) formula
  - e.g., COCOMO, Cosmic, …
- Measurement of **size**
  - function points, Lines of code, etc
- **Parameters**
  - Historical weights, constant factors, etc

1. Estimate Function Points

Idea: Predict the complexity of the system in terms of the number of functions to write, without being as specific as lines of code, which is programming language dependent.

The Basic Model is:

$$FP = a_1 I + a_2 O + a_3 Q + a_4 F + a_5 E$$

FP	= number of function points
I	= number of external inputs (input files, forms, screens, messages)
O	= number of external outputs
Q	= number of external queries (I/O queries with response)
F	= number of internal data files (files, directories, etc)
E	= number of external interfaces (libs, procedures, etc)

a_1, a_2, ..., a_5 - empirically observed weightings per function type

Estimating Code Size From FPs

LOC = Z * FP

There are tables that list for each programming language, the **number of statements (Z)** in it that are required to implement **one** function point.

Z =

• **Assembly:**	**50**
• **C:**	**15**
• **C++:**	**12**
• **Java:**	**9**
• **LISP:**	**2**

These tables are calibrated for each domain, project, etc.

Estimate Cost

- COnstructive COst MOdel (COCOMO) - used to predict the cost of a project from an estimate of its size (LOC).

$$E = a * KLOC^b * Y$$

- **E** is for Effort - estimated man-months
- **KLOC** - estimated project size (thousand lines of code)
- **a**, **b** - empirically observed weights; depend on type of system being developed
- **Y** - project attribute multipliers

Type of project	a	b
Organic (< 50 KLOC)	2.4	1.05
Semi-detached (< 300 KLOC)	3.0	1.12
Embedded	3.6	1.20

Project Attribute Multipliers (Y)

Adjust Effort estimation according to difficulty of the project:

- **product attributes**: required reliability **++**, complexity **++**, database size **++**
- **resource attributes**: execution time **++**, memory **++**, hardware volatility **++**, tight response time **++**
- **personnel attributes**: quality of analysts **--**, quality of programmers **--**, experience with product **--**,hardware experience **--**, programming language (PL) experience **--**
- **project attributes**: use of software tools (e.g., debugger) **--**, use of modern PL **--**, schedule constraints **++**

Caution!

- Models have to be **calibrated** to an organization. Accuracy is influenced by local factors: expertise, process, product type, definition of LOC.
- Be careful of model parameters based on **old** projects/technology.
- Weights and coefficients are based on empirical studies of past projects using old technology, and may be completely unlike new projects.

Caution!

- Predictions can become self-fulfilling
  - If estimates are used to generate the project plan, which is used by managers to manage the project, the project ends up having to conform to the estimate! *Parkinson's Law (PL)!*
- PL states that work expands to fill the time available. The cost is determined by available resources rather than by objective statement.

Caution!

- Cost estimation is based on intuition and experience (history). You get better at it overtime. Be wary of any method or tool vendor that claims to predict cost or effort to **unrealistic precision**.

Costing and pricing

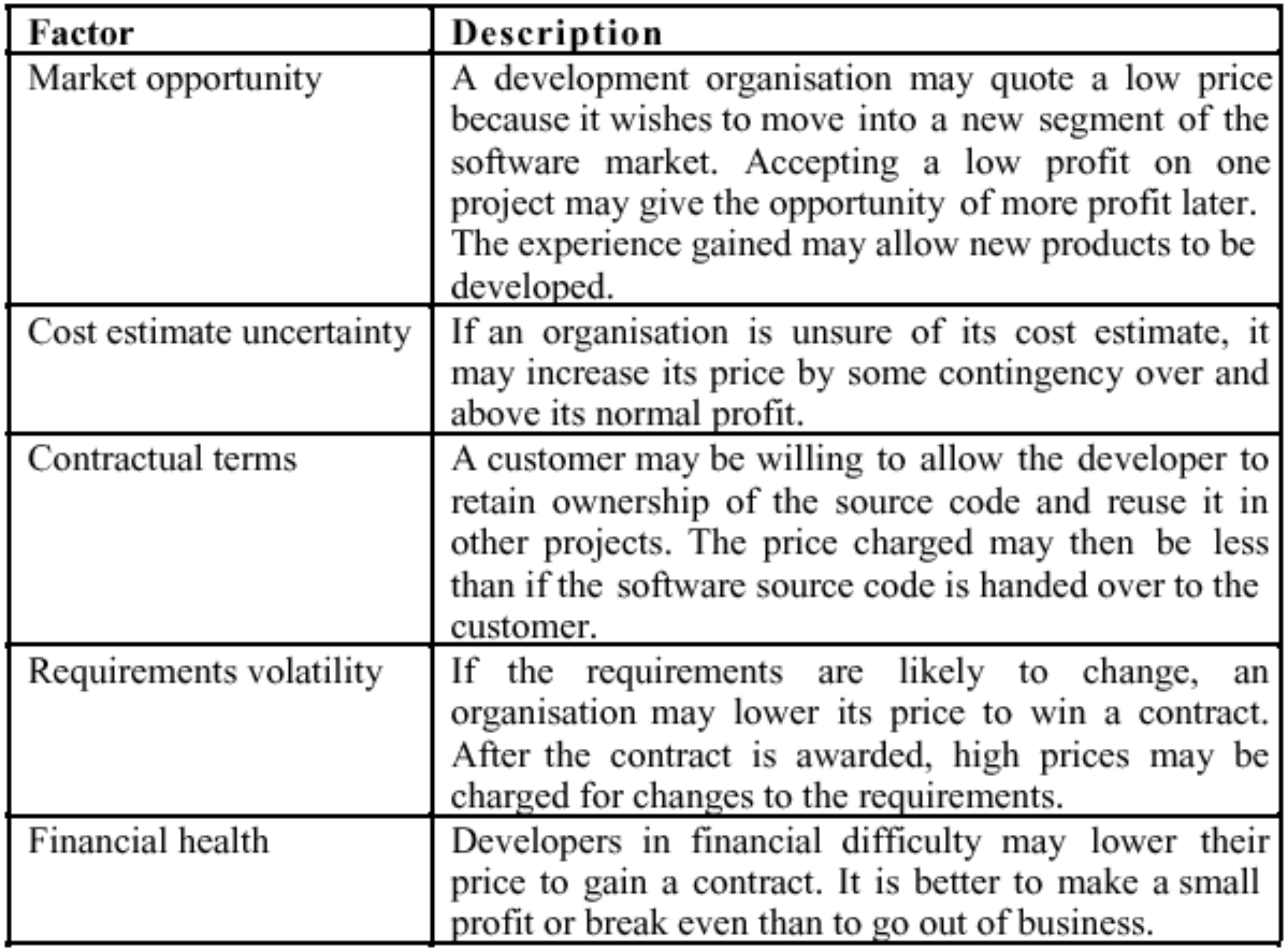
- There is not a simple relationship between the development cost and the price charged to the customer
- Broader organisational, economic, political and business considerations influence the price charged

Direct and indirect costs

- Direct costs: Costs incurred for the benefit of a specific project
  - **Salaries** of project staff
  - **Equipment** bought specifically for the project
  - **Travel** expenses

- Indirect costs: Costs incurred for the joint benefit
  - Accounting, **tax**, quality assurance department
  - **Space**, **electricity**, **heating/cooling**

Software pricing factors



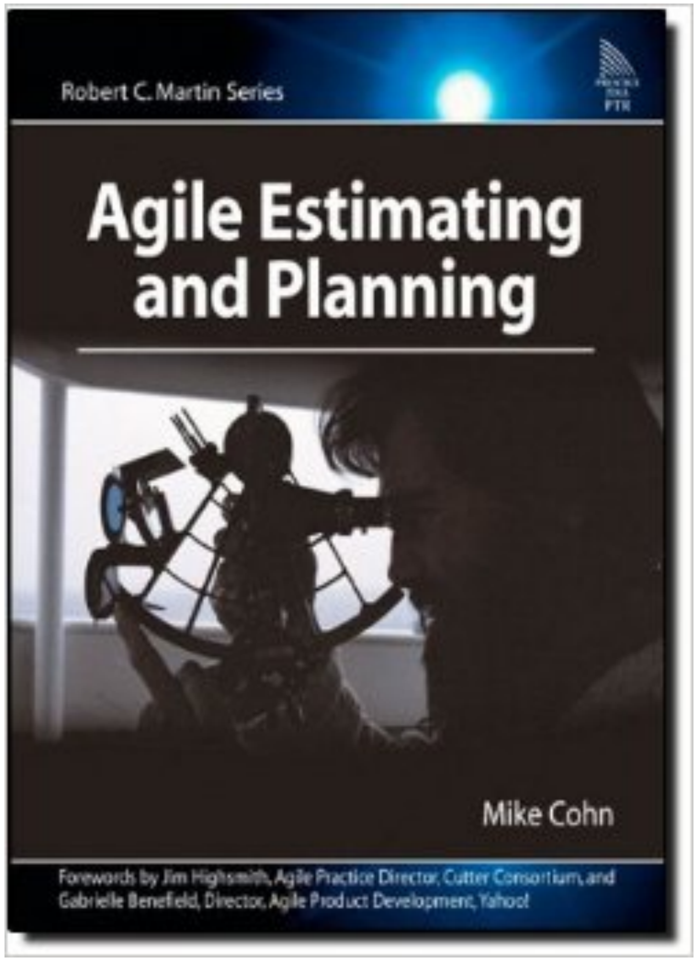
Factor	Description
Market opportunity	A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the opportunity of more profit later. The experience gained may allow new products to be developed.
Cost estimate uncertainty	If an organisation is unsure of its cost estimate, it may increase its price by some 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.
Requirements volatility	If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices may 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 small profit or break even than to go out of business.

What else should we pay attention to

- Estimate Development Costs
  - Software license fees
  - New equipment: servers, PCs,
  - Services (amazon, AppEngine)
- Estimate Testing Costs
  - Test cases, code coverage, continuous integration
- Estimate Maintenance and Enhancement Costs
  - Bugs, technical debt, refactoring, etc

In-class exercise

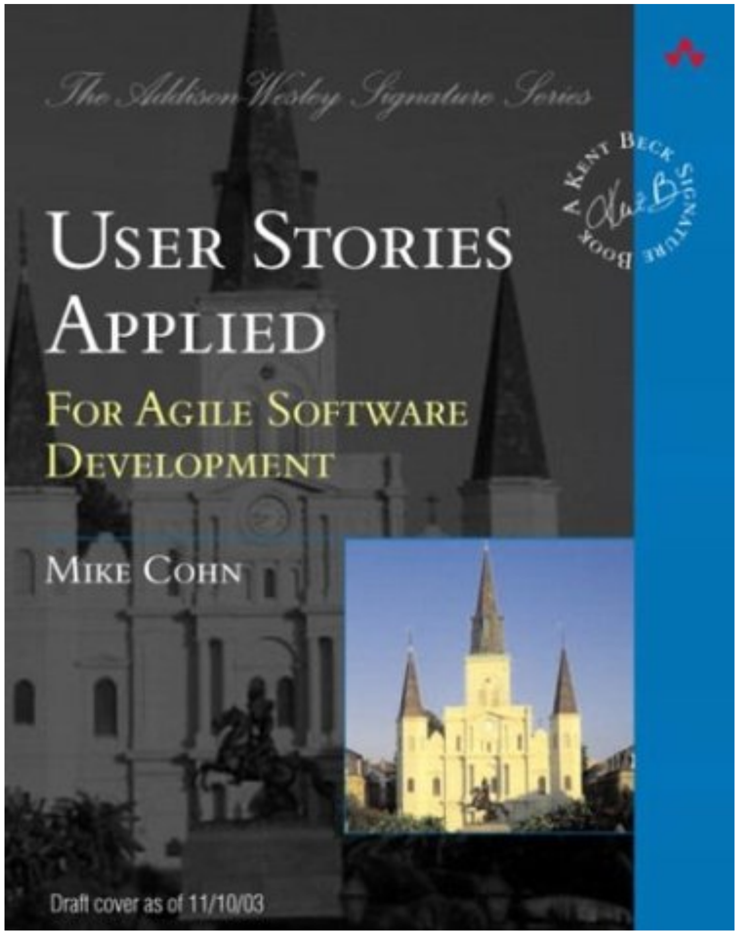
- Form your lab groups.
- Estimate the cost of implementing the UNDO feature using the COCOMO model.


Robert C. Martin Series

Agile Estimating and Planning

Mike Cohn

Forewords by Jim Highsmith, Agile Practice Director, Cutter Consortium, and Gabrielle Benefield, Director, Agile Product Development, Yahoo!


The Addison-Wesley Signature Series

A Kent Beck Signature Book

User Stories Applied

For Agile Software Development

Mike Cohn

Draft cover as of 11/10/03

Measures of size

Sequential

- Lines of code
- Function points

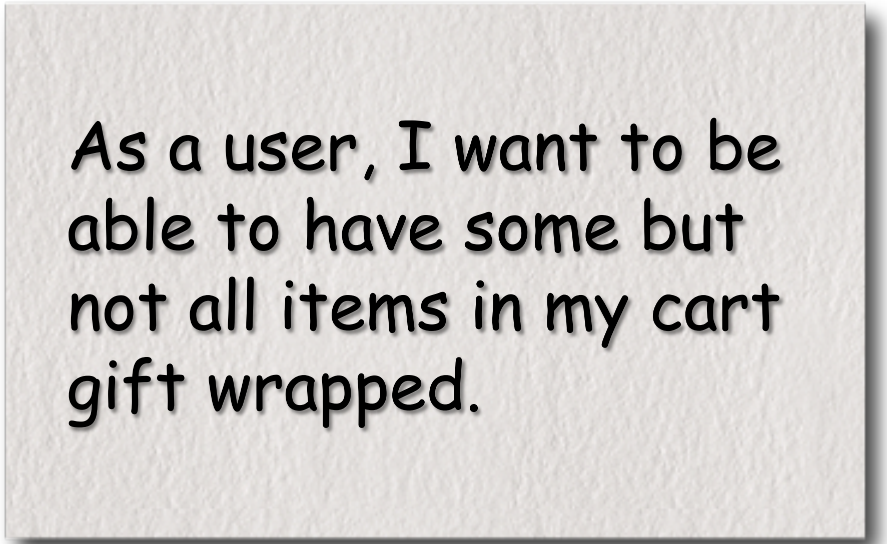
Agile

- Story points
- Ideal days

Ideal days

- How long something would take if
  - it's all you worked on
  - no **interruptions**
  - and everything you need is available
- The **ideal time** of a soccer game is **90** minutes
  - 2 X 45-minute halves
- The elapsed time is much longer (2 hours):
  - 15-minutes halftime,
  - Overtime: penalties, injuries, offside, crazy guy running naked in the field, etc
  - Total real time: 90 + 15 + overtime

Story points: relative values



As a user, I want to be able to have some but not all items in my cart gift wrapped.

- The "bigness" of a task
  - Influenced by
  - How **hard** it is
  - How **much** of it there is
- **Relative values** are what is important:
  - A login screen is a 2.
  - A search feature is an 8.
- Points are **unit-less**

Estimate by analogy

- Comparing a user story to others
  - "This story is like that story, so its estimate is what that story's estimate was."
- Don't use a single gold standard
  - Triangulate instead
- Compare the story being estimated to multiple other stories

Use the right units

- Can you distinguish a 1-point story from a 2?
  - How about a 17 from an 18?
- Use a set of numbers that make sense:
  - 1, 2, 3, 5, 8, 13
- Stay mostly in a 1-10 range
- Nature agrees:
  - Musical tones and volume are distinguishable on a logarithmic scale

Use 0 and $\frac{1}{2}$ if you like

Why Bother?

Poor estimates may be better than no estimates because:

- You need this information to **negotiate** the cost of product.
- You need this information to **plan** for the project: to determine how many developers to hire or to assign to this project, how long they'll be dedicated to this project and not to others.
- You cannot **control** what you cannot **measure**.

Your ability improves with practice and experience.

Further resources

- *The Mythical Man-Month*, by Fred Brooks
- *Rapid Development*, by Steve McConnell
  - Ch. 8, 9
  - Ch. 29, 32
- *Death March (2nd ed.)*, by Ed Yourdon
  - Critical-Chain Scheduling (pp.175-177)
- *Code Complete*, by Steve McConnell