Feasibility studies

SEng 321

Based on Material from D. Damian

Outline

- Why feasibility studies
- Types of feasibility studies
- Estimating cost
- The fallacy of the mythical man-month

Why do a feasibility study?

- To see whether the solution to the problem is both economically and technically possible
- Is the cost of development justified by the expected benefits?
- Helps management understand the alternatives possible and tradeoffs that need to be made

Types of feasibility

- Schedule
- Operational
- Technical
- Economic

Exploring schedule feasibility

- Required technical expertise available? If not, how long does it take to obtain it?
  - Training costs
  - New hires

- How reasonable are the project deadlines?

Exploring operational feasibility

- What is the likelihood of success of the software solution in the operational environment?
  - Is there management support?
  - Does it require change? Change is difficult!
  - Adoption issues
- Important to evaluate whether a system *will* work not only if it *can* work

Exploring technical feasibility

- Can the software envisaged deliver the performance asked for?
- Is the environment capable of timely delivery of the necessary data for the system?
- Is the required technology available?

Technical feasibility

Example of importance during requirements engineering

R1. “the system should provide a response time of at most 2 seconds in 80% of cases”

Or

R2. “the system should provide a response time of at most 2 seconds in 90% of cases”

The difference may be in doubling the CPU-capacity, possibly not technically feasible.

Exploring economic feasibility

- Do the gains outweigh the costs of development?

- Conducting a cost-benefit analysis
  - Difficult early in the project, both benefits and costs can be intangible
  - Estimate costs

Cost estimation

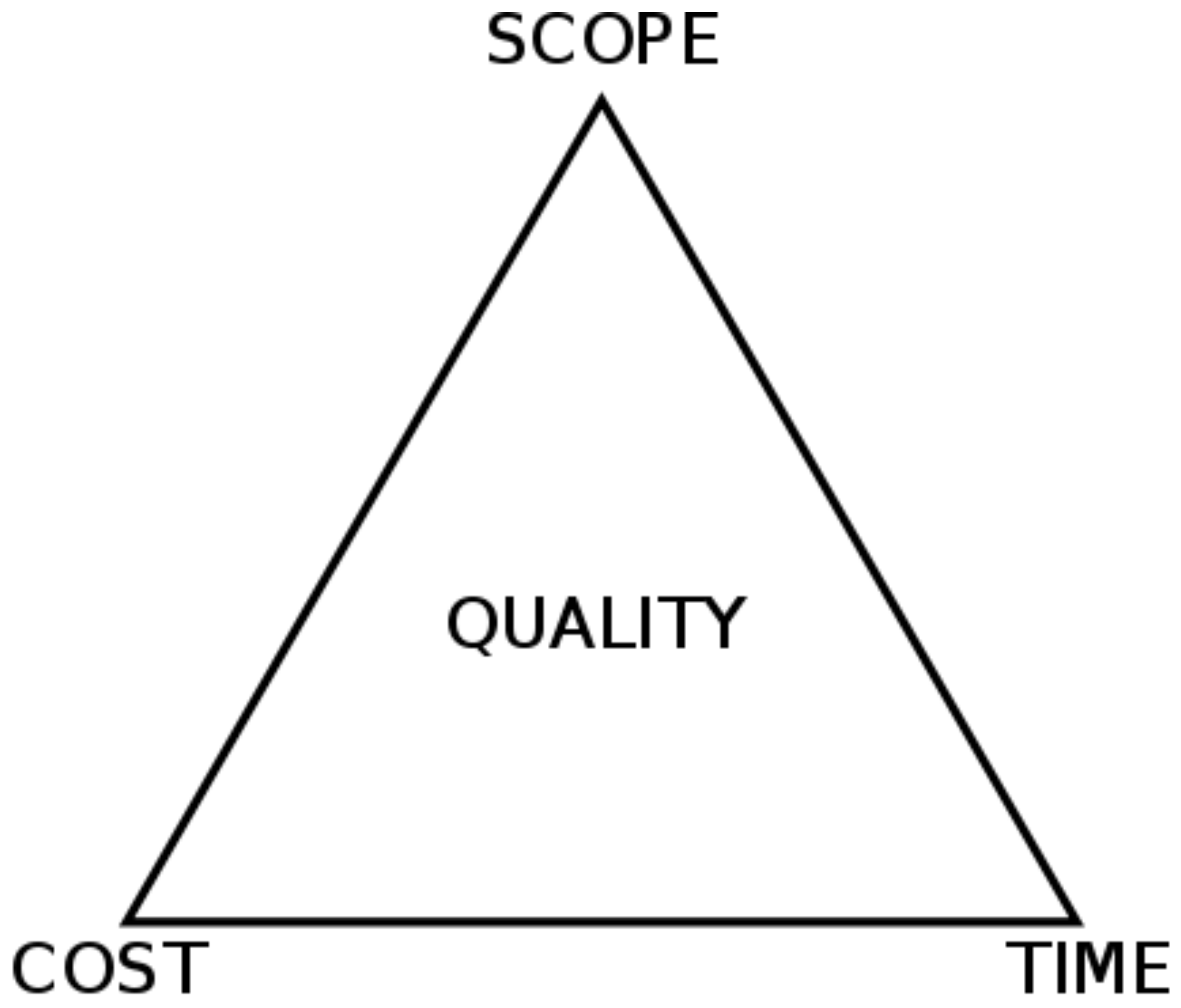
Principal components of project costs:

- Hardware costs.
- Travel and training costs.
- Effort costs (the costs of paying software engineers).

• Effort cost usually measured in staff-hours or staff-months (also referred to as man-months)

Project Management Iron Triangle

- See https://en.wikipedia.org/wiki/Project_management_triangle



Well-known and documented cost estimation techniques

- Algorithmic cost modeling
- Expert judgment
- Estimation by analogy
- Top-down estimation
- Bottom-up estimation

(how not to estimate cost:)

- Parkinson’s Law (work expands to fill the time allotted)
- Pricing to win

Algorithmic cost modeling

$$Effort = A * size^b * M$$

- Size: estimated size of the software (KLOC)
- A,b and M: parameters reflecting process and product factors

Difficulties:

- In estimating size
- More research needed to identify A,b and M

Algorithmic cost modeling, examples

- Simply LOC
- COCOMO
- Function Points

COCOMO

$$E=a*size^{b}$$

where a and b are constants that depend on the kind of project is being executed

COCOMO distinguishes three classes of project:

Organic: a relatively **small team** develops software in a known environment. The people involved generally have a **lot of experience** with similar projects in their organization. They are thus able to contribute at an early stage, since there is no initial overhead. Projects of this type will seldom be very large projects

Embedded: the product will be **embedded in an environment** which is very inflexible and poses severe constraints. An example of this type of project might be air traffic control, or an embedded weapon system

Semidetached: this is an intermediate form. The team may show a **mixture of experienced and inexperienced** people, the project may be **fairly large**, though not excessively large.

COCOMO, cont'd

e.g. if size = 200KLOC

Organic: a=2.4, b=1.05
semidetached: a=3.0, b=1.12
embedded: a=3.6, b=1.20

organic: Effort=2.4* ($200^{1.05}$) = 626 man-months (MM)
semidetached: Effort=3.0* ($200^{1.12}$) = 1133 MM
embedded: Effort=3.6* ($200^{1.20}$) = 2077 MM

Function Points

Related to functionality rather than size

Computed by counting:

- # external inputs and outputs
- # user interactions
- # external interfaces
- # files used by the system

Weights are associated with these entities

FP = weighted sum of these five entities

Well-known and documented cost estimation techniques

- Algorithmic cost modeling
- Expert judgment
- Estimation by analogy
- Top-down estimation
- Bottom-up estimation

(how not to estimate cost:)

- Parkinson’s Law
- Pricing to win

Expert judgment

Group consensus technique

- Experts come to a consensus regarding estimated cost

Delphi technique

Moderated estimation by a number of experts, iteratively

- Experts give their individual estimates
- Moderator collects the estimates and redistribute them among the experts, anonymously
- New estimates delivered
- Process continued until consensus is reached

Estimation by analogy

“This environmental impact report generator for Oregon is similar to the one we developed for Florida last year for $1,200,00. The Oregon system has about 30% more types of reports than the Florida system had, so we’ll add $360,000 to cover them. On the other hand, we’ll be using many of the same people, so we can reduce the estimate by about 20%, or $240,000, to account for the time we spent getting up the learning curve on the Florida project. Also, we can save probably another 20% by reusing some of the low level report generation modules and most of the pollution model software, for another reduction of $240,000. Thus, our cost will probably be around

$1,200K + $360K - $240K - $240K = $1,080K.”

Top-down and Bottom-up estimating

Top-down estimating:

- system level focus: doesn't miss costs of integration
- may not identify low level technical problems;

Bottom-up estimating:

- covers just the costs associated with developing individual components;
- tends to overlook many of the system level costs (integration, config. management, quality assurance, project management);

Parkinson’s Law

“This flight control software must fit on a 65,536-word machine; therefore, its size will be roughly 65,000 words. It must be done in

18 months and there are 10 people available to work on it, so the job will take roughly 180-man-months”

… by the time the project was finished, it had added another 65K on board to accommodate the 127,000 words of software actually developed, and it required a total of 32 months and 550 man-months.

Price-to-win estimating

“I know the cost model estimated $2 million for this job, and none of our experts believe we can do it for less than a million and a half. But I also know that the customer has only $1 million budgeted for this software contract, so that’s what we’re gonna bid.

… now go and fix up the cost estimate and make it look credible.

We absolutely have to announce this product at the National Computer Conference next June, and here it is September already. That means we’ve got 9 months to get the software ready…”

Summary comparison of methods

- None of these alternatives is better than the others from all aspects.
- The Parkinson and Price-to-win methods are unacceptable and do not produce sound cost estimates
- The strengths and weaknesses of the other techniques are complementary (particularly the algorithmic model versus expert judgment and top-down versus bottom-up).

The fallacy of people-month model

- People and months are interchangeable commodities
- Cost does vary as the product people x months.

PROGRESS DOES NOT

The fallacy of people-month model

People-month as a unit for measuring the size of a job is a dangerous and deceptive myth

Brooks's Law:

Adding manpower to a late software project makes it later

Why?

In all cases?

References

Boehm, B. W. (1981). **Software Engineering Economics**. Englewood Cliffs, N.J., Prentice-Hall.

Van Vliet, H. **Software Engineering: principles and practice**, 2000, Wiley

Brooks, F. **The mythical man-month**, 1995, Addison-Wesley