Universität Zürich / Wintersemester 2002/2003

Seminar on Software Cost Estimation

Introduction on Software Cost Estimation

Presented by

Rolf Hintermann

Institut für Informatik
der Universität Zürich

Prof. Dr. M. Glinz
Arun Mukhija

29. October, 2002

Content

1 Introduction

1.1 The growing Importance of Software Cost Estimation

Software cost estimating has been growing in importance up to today. When the computer era began back in the 1940's, there were few computers in use and applications were mostly small, one person projects. As time moved on, computers became widespread. Applications grew in number, size and importance; costs to develop software grew as well. As a result of that growth, the consequences of errors in software cost estimation became more severe too. Still today, a lot of cost estimates of software projects are not very accurate, mostly too low. This is not a surprising fact if we look at the various difficulties we have to face when estimating software costs. The by far greatest amount of the total costs of a project arises from the salaries of the personnel. Other costs, as license fees or new equipment for example, occur only once and are not too hard to estimate. The costs for the human workers on the other hand are highly correlated to the effort we need to perform the project. Therefore we have to get an accurate enough estimate of the total effort in order to make a reasonable estimate of the costs. The effort is estimated based on the size and complexity of the project, which both derive from the specification. Because the requirements of the software are likely to change, we have to take this into account too when estimating the effort. The big difference in productivity of software developers is one of the hardest problems to solve during the estimation process. An experienced developer will produce far more than a beginner. But because each project is unique, uses it's own tools and languages, the experience level of the development team is hard to judge. Another problem appears when humans ere estimating. We all tend to underestimate immaterial things like software. This is also a reason why software is considered to be expensive by most people, although there is nothing to compare its costs with. Today's world would not be the same if there was no software.

1.2 Birth of the Software Cost Estimation Industry

Before 1970, software cost estimating was done manually using simple rules of thumb or estimating algorithms developed through trial and error.
Around 1970, a number of researchers independently began to think about how to build automated software cost estimating tools. Those pioneers all worked for large corporations that were experiencing difficulties building large software systems.
On the early 1970's the first automated software cost estimating tools had been built. Those tools were internal estimating tools of the corporations the researchers worked for. Some of them later evolved into commercial ones.
Around 1975, Allan Albrecht and his colleagues at IBM White Plains developed the original version of the today widespread function points metric. This metric is based on five different attributes of software applications:

1) Inputs
2) Outputs
3) Inquires
4) Logical Files
5) Interfaces

It is used to express the size of an application independently from the programming language used. This independence makes it easier to estimate the size of the non-coding deliverables of a project such as user manuals and the like.

1977, the PRICE-S software-estimation model designed by Frank Freiman was the first commercial tool to be marketed in the United States.
1979, the second commercial tool was introduced to the US-Market. This was the software life-cycle management tool (SLIM) developed by Larry Putnam.
1981, Dr. Barry Boehm published his famous book “Software Engineering Economics” in which he revealed the essential algorithms of the constructive cost model (COCOMO), which is the only model whose algorithms aren’t considered trade secrets.
Also in 1981, Allan Albrechts paper on function points metric was published worldwide for the first time as a part of Capers Jones book “Programming Productivity – Issues for the Eighties”.
In 1982, Tom deMarco published his book “Controlling Software Projects” in which he describes a functional metric that duplicated some of the features of Albrecht’s function points, but was developed independently. Several of today’s tools support derived forms of Tom deMarco’s function points.
In 1983, Charles Symons, a British software-estimating researcher, published an alternative function point metric named Mark II function points, which became widely used in the United Kingdom, Canada and Hong-Kong.
In 1984, IBM made a major revision of its function point metric, which is the basis of today’s standard function points.
In 1985, Capers Jones’ SPQR/20 (for software productivity, quality and reliability) was put on the commercial market. This was the first tool to explicitly support function point metrics and bidirectional backfiring. Bidirectional backfiring is a mechanism to convert function points into an equivalent logical statements total and vice versa. It also was the first tool trying to predict a whole bunch of key software factors:

1) Sizing of all deliverables
2) Staffing by activity
3) Schedules by activity
4) Effort by activity
5) Costs by activity
6) Quality and defect severity levels
7) Defect removal efficiency
8) Reliability
9) Risks
10) Maintenance
11) Enhancements

In 1986, due to the rapidly growing usage of function point metrics, the international function points users group (IFPUG) was founded in Toronto, Canada.
From 1986 until today, a lot of commercial software cost estimating tools have been released. One new model to mention is COCOMO II, which added function point metrics and some additional features to the original version.

2 Software Cost Estimation

2.1 Time and Purpose of Estimates

Prior to a project, a rough estimate is made to help managers to decide whether to make or buy a software, perform a cost/use or break even analysis. In this estimate, the total costs and the schedule are of interest.
During the development process, software cost estimates together with measurements provide

a tool to the project manager to control the proceeding of each software phase. Those estimates require more details to be effective.

2.2 Potential Problems of Estimation

As mentioned in the introduction, we face various problems when estimating software costs. The Specification of a project builds the base of all our following estimation work. Requirements changes will lead to a specification change. This is a very serious difficulty for the estimators but also the developers.
When deriving the size of the project from the specification, humans tend to focus on the components providing the core functionality of the final product. Often those components are considered more complex than they really are. In most projects they actually are just a small part of the whole code. Another consequence of this focus is that other components such as the graphical user interface are highly underestimated. The same is true for the non coding components of the project. All the documentation that is produced besides the code makes out quite a big part of the total effort. Function point metrics are a good help to deal with those problems. Estimation tools can prevent us from underestimating or even omitting certain components.
A Situation where today's tools cannot help the human estimator is when it comes down to estimate the level of experience of the development team. The fact that in large projects the one who is estimating does not know the developers does not make this task easier.
When really estimating the costs, organisational tasks on project management level are often overlooked. This may lead to a delay of the project or higher costs.

2.3 Different Methods of Estimation

There exist various methods to perform a software cost estimate, each having it's own strengths and weaknesses:

1) Algorithmic models

In algorithmic models, cost can be looked at as a function with the major cost drivers as variables. Inside such a model, one or more algorithms are used.
The strength of algorithmic models lies in its objectiveness. Its results are repeatable. A weak point is that still some subjective input is needed

2) Expert judgement

As the name indicates, human experts perform the estimate when using this method. To make the estimates more accurate, multiple experts are estimating.
The quality of the estimates made with this method highly depends on the skills of the individual experts.

3) Analogy

The estimate is based on the data of similar previous projects.
When using this method, the biggest problem is to find representative previous projects.

4) Parkinson

The estimate is made equal to the available resources.

5) Price to win

The estimate is made as low as necessary to win the job.
The strength of this way to estimate is that you will often get the contract for the project. The weakness is obviously that you are almost sure to produce higher costs than predicted.

6) Top-down

First the total costs are estimated for the whole project. Total costs are then split up among the different components.

7) Bottom-up

First each component is estimated separately. Then the results are added up to get an estimate for the whole project.

3 Estimation Process

3.1 Key Factors and Objectives of the Estimate

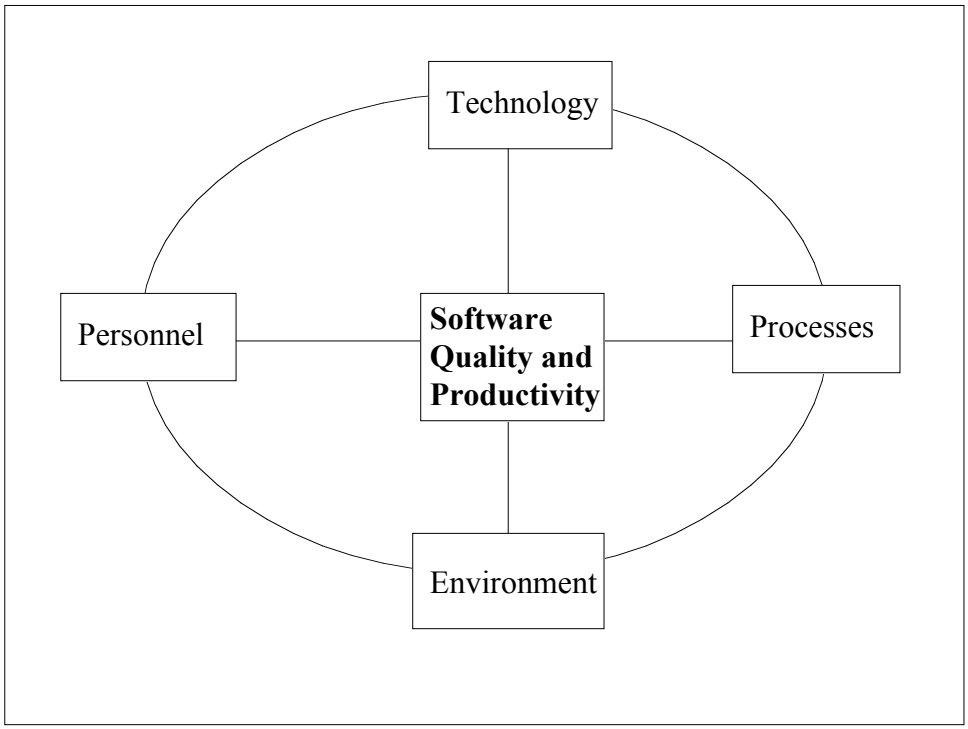

Figure 1 Key estimate Factors (Jones, 97)

There are a lot of factors that have an influence on software costs; Capers Jones puts them into four different Groups:

1) Personnel

The number of developers is in a less than linear relationship to the time span until delivery. This simply means that we cannot just employ twice as many developers to finish the project in half the time due to the super linearly growing costs of communication when doing this.

But the number of workers linearly correlates to the costs of the project. Therefore we need to find the optimal number of people to employ. We try to do this by investigating the data of previously completed projects.
The level of experience of the developers will affect both, the schedule and the total costs. High paid specialists might be needed during some phases of the development process.
In order to estimate accurately, we also have to consider the payment agreements, especially how overtime is handled.
Whenever development groups work at different geographic locations, travel costs will occur which can add up to a respectable part of the total costs, depending on the specific situation.

2) Environment

The office ergonomics will affect the total costs. Data of previous projects shows for example that the average output of developers is higher in a silent environment. Most the time, the environment is given and can only be adapted slightly.

3) Methods/Processes

The methods and processes used to develop software also have an influence on the costs. Generally it can be said that the more communication is involved in the development process, the lower the productivity will be. Data of completed project shows that especially project using some kind of matrix organization where a lot of communication is needed have a lower output than others. However one should also consider the experiences a corporation has made using a method or process to develop software.

4) Technologies

A different group of key factors is the technology involved. It's possible that new equipment has to be bought. Those costs are more or less easy to estimate. More important are the languages and tools used and how many reusable artefacts are available. It would be a great advantage if the developers have some experience with the language chosen and the tools used. The standards to be adhered are also a factor that should not be overlooked.

Additional key factors
Total costs also depend on what kind of software has to be developed. Estimates won't be the same for information systems, systems software, commercial software, military software and embedded software. For example in a military project, there is a lot more documentation work to do than if it was a commercial project.
Another key factor is the rate of changing requirements. It's obvious that this rate itself has to be estimated somehow, preferably making use of previous projects.

Objectives of the estimate
Software cost estimating performs more tasks than the name indicates. It's not just the total costs that are of interest but a lot more. Quite a few things other than costs are being estimated during the process. Of main interest are the following:

- size of all deliverables
- staff needed
- schedule
- effort
- costs to develop

- costs for maintenance/enhancement
- Quality
- Reliability

3.2 Activities of the Estimation Process

The two software cost estimating experts, Dr. Barry Boehm and Capers Jones describe the activities to perform an estimate as a sequence of a few steps. The following two sections try to summarize those descriptions

3.2.1 Estimation Steps as proposed by Jones

Because Jones sees the analysis of the requirements as a pre-estimating step, he starts his sequence with step 0.

0) Analyze the requirements

Before a meaningful estimate can be produced on the costs of a software project, it needs to be known what it will look like. Therefore it is necessary to understand the requirements before actually beginning with real estimation activities.

1) Start with sizing

The size of all deliverables of the software project needs to be estimated. Those estimates are very important, because almost all major cost drivers will later be estimated based directly or indirectly on the size. If the sizes of the key deliverables cannot be predicted accurately, the overall estimate won't be accurate either.

2) Identify the activities to be included

In order to estimate accurately, the sets of activities that will be performed for the project need to be identified. Activities include work to be done such as requirements, design, coding, reviews, document creation, testing, integration, quality assurance, project management and many others depending on the specific project.

3) Estimate software defect potentials and removal methods

The most expensive and time consuming work in the software development process is finding bugs and fixing them. Bugs may not only appear in the code, but also in the requirements, design or user documentation.

4) Estimate staffing requirements

How many workers are needed depends on the overall size of the project, and on the activities that will be performed. In this step it is essential not to forget any specialist that will be needed.

5) Adjust assumptions based on capabilities and experience

The productiveness of developers varies greatly among the different individuals. Therefore adjustments need to be made to the staffing estimate based on the level of experience and skill factors of the development team.

6) Estimate effort and schedules

Effort and schedules are estimated based on the size of the project, the activities included, the number of workers and their experience level. An interesting point here is that productivity seems to be more closely related to the number of managers engaged in a project than to the actual number of programmers involved. Productivity tends to decline when the number of managers increases.

7) Estimate development costs

The costs for personnel are dependent on the effort, the schedules, the number of workers and their average salary. If the project runs for several years, inflation rates need to be taken into account, if it is international, currency exchange rates have an influence. The above costs form a basic estimate. In addition there are several other cost factors that should not be forgotten such as license fees, new equipment costs, travel costs, marketing and advertising costs and others.

8) Estimate maintenance and enhancement costs

Estimating maintenance costs requires the knowledge of probable number of users of the application and of the probable number of bugs or defects in the software at release time. Estimating enhancements costs requires good historical data on the rate of change of similar project, once their application was being used.

3.2.2 Estimation Steps as proposed by Boehm

1) Establish Objectives

In order not to waste any effort in gathering information that has no relevance to the need for the estimate, it should be found out why this particular estimate is done prior to any other activity. There is for example no use in going into details, if the decision-maker requires a rough estimate to help him decide whether to make or buy software.

2) Plan for Required Data and Resources

Software cost estimation activity should be seen as an own mini project, which requires a project plan. This plan does not need to be a detailed document, but at least some notes on the

why, what, when, who, where, how, how much and whereas of the estimating activity should be taken.

3) Pin Down Software Requirements

Requirements should be testable, otherwise they are not very useful to estimate the costs. Where it is not possible to make a requirement testable or takes to much effort than we want to spend with regard to the objective of the estimate, any assumptions that have been made should be documented.

4) Work Out as Much Detail as Feasible

The more details we know about the software, the more accurate will be our estimate. The reasons for this are that we understand the software's technical aspects and we are less likely to miss costs of the more unobtrusive components.
However, the degree of detail should be consistent with the objectives of the estimate.

5) Use Several Independent Techniques and Sources

It is a good idea to use different estimation methods to profit from their combined strengths and to avoid the weakness of a single technique.

6) Compare and Iterate Estimates

It is important not just to perform independent estimates, but also to investigate why they produce different results.

7) Followup

Once a software project started, it's actual costs and progress should be measured and compared against the estimate.

3.3 Reasons for underestimating

It is a fact that a lot of projects have been underestimated in the past and still are today, meaning they either exceeded the planned costs, could not finish on time or in the worst case had to be stopped. The following sections try to give some possible reasons for why this is the case.

3.3.1 Fantasy Factor or "Training by Managers"

Boehm discovered, that when comparing cost estimates of humans against actual costs, their estimate usually is about one third lower than the real cost. His explanation for this fact is that people are basically optimistic, desire to please and generally are not familiar with the entire software project and all the activities it includes.
Tom deMarco has a slightly different explanation. He simply states that the managers trained us to give low estimates. In a survey on estimating techniques, most managers stated, that their software estimates were dismal, but they weren't dissatisfied with the estimation process on the whole. This makes him conclude, that the managers do not want accurate estimates, but rather underestimates, because we all tend to work harder, if we have a tight schedule.

3.3.2 Omitting some cost factors

Another explanation could be that in most cases some significant cost drivers are simply forgotten. If you take a look at how many cost drivers there are and how different the situation for every project is, this seems quite reasonable. The following tasks are often omitted or underestimated by human managers:

- effort to find and fix bugs
- production of paper documents
- travel costs
- testing
- administration
- contribution of specialists
- maintenance

4 Conclusion

Software cost estimating is a very complex process. There are a lot of factors that have an influence on the costs of a project. Therefore I believe that estimation tools are necessary to produce reliable estimates. Those tools still require some subjective inputs and I think they still will in the near future. They also make sure, that no important factor is omitted in the estimating process.
Because every project is unique and I do not believe that there is a best method to estimate software costs, Boehm's suggestion to use several estimation methods seems to be a good idea to me.

5 Literature

Boehm, B. (1981). "Software Engineering Economics" Englewood Cliffs, N.J.: Prentice Hall.
DeMarco, T. (1995). "Why Does Software Cost So Much? And Other Puzzles of the Information Age" New York: Dorset House.
Jones, C. (1998): "Estimating Software Costs" New York: McGraw-Hill.