Software Estimation

The stated goal of software engineering - delivering projects on time, within budget, and up to specifications implies set targets for these features. These targets are the product of the software estimation process.
Why Software Estimation?

• Links economic analysis and software engineering - Allows cost-benefit, breakeven, and make-or-buy analyses.

• Project planning and control needs of management - Help set staffing needs and set budget and schedule.
“Most companies and government agencies have only a hazy idea of the caliber of their software and the productivity of their programmers. Consequently, predicting the investment of money and time needed to create programs becomes such a difficult task that overruns and delays are the norm rather than the exception… this basic problem of measurement is one of the biggest obstacles now facing the software industry.”

Capers Jones (1998)
### Some Project Data

Project data - effort is in work months (as percentage of total effort in brackets)

<table>
<thead>
<tr>
<th>Project</th>
<th>Design wm (%)</th>
<th>Coding wm (%)</th>
<th>Testing wm (%)</th>
<th>Total wm</th>
<th>SLOC</th>
<th>SLOC/wm</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3.9 (23)</td>
<td>5.3 (32)</td>
<td>7.4 (44)</td>
<td>16.7</td>
<td>6050</td>
<td>362.2754</td>
</tr>
<tr>
<td>b</td>
<td>2.7 (12)</td>
<td>13.4 (59)</td>
<td>6.5 (26)</td>
<td>22.6</td>
<td>8363</td>
<td>370.0442</td>
</tr>
<tr>
<td>c</td>
<td>3.5 (11)</td>
<td>26.8 (83)</td>
<td>1.9 (6)</td>
<td>32.2</td>
<td>13334</td>
<td>414.0994</td>
</tr>
<tr>
<td>d</td>
<td>0.8 (21)</td>
<td>2.4 (62)</td>
<td>0.7 (18)</td>
<td>3.9</td>
<td>5942</td>
<td>1523.59</td>
</tr>
<tr>
<td>e</td>
<td>1.8 (10)</td>
<td>7.7 (44)</td>
<td>7.8 (45)</td>
<td>17.3</td>
<td>3315</td>
<td>191.6185</td>
</tr>
<tr>
<td>f</td>
<td>19 (28)</td>
<td>29.4 (44)</td>
<td>19 (28)</td>
<td>67.7</td>
<td>38988</td>
<td>575.8936</td>
</tr>
<tr>
<td>g</td>
<td>2.1 (21)</td>
<td>7.4 (74)</td>
<td>0.5 (5)</td>
<td>10.1</td>
<td>38614</td>
<td>3823.168</td>
</tr>
<tr>
<td>h</td>
<td>1.3 (7)</td>
<td>12.7 (66)</td>
<td>5.3 (27)</td>
<td>19.3</td>
<td>12762</td>
<td>661.2435</td>
</tr>
<tr>
<td>i</td>
<td>8.5 (14)</td>
<td>22.7 (38)</td>
<td>28.2 (47)</td>
<td>59.5</td>
<td>26500</td>
<td>445.3782</td>
</tr>
</tbody>
</table>

Avg. = 617.2002

Productivity = SLOC/wm

If the project leaders for a and d were able to accurately estimate the SLOC required and then used the average productivity of the organization to predict effort needed for the projects, how far off would their estimates have been?

For Project ‘a’ SLOC/Avg. Productivity = 9.8 wm
   Actual effort was 16.7 wm.

For Project ‘b’ SLOC/Avg. Productivity = 9.63
   Actual effort was 3.9 wm
Estimating, like testing, continues throughout the software development process. However, estimating reaches its peak after the specifications have been drawn up but before design activities begin. After the specifications are completed, meaningful duration and cost estimates are computed and a detailed plan for the project is produced. By the end of the specifications phase developers have a detailed appreciation of what is to be built.
Predicting the size of deliverables is the usual starting point for software cost estimating.

As the project proceeds the accuracy of estimates should improve as the characteristics of the required system become clearer. An increasing accuracy of size estimates characterizes the process of measurement and size estimation throughout development.

It is recommended that several size estimates be made at several milestones during development.
Estimating Cost

Before a client signs off on the construction of software, they will want to know how much it will cost.

If the development team underestimates cost they may lose money on the project.

If the development team overestimates cost the client may think producing the product is not cost-effective. Or, the client may award the project to developers with a more reasonable cost estimate.
Estimating Duration

Clients want to know when the finished product will be delivered.

If a development organization overshoots the schedule (underestimates time to completion) they will lose credibility and/or penalty clauses may be invoked.

If the developer overestimates the time to completion they may not get the work in the first place.
Human Factors Contributing to Estimation Difficulties

Several studies have found wide variations between the capabilities of programmers. Bryan (1995) found on one project: “Eighty percent of the work result was done by a single programmer out of a workforce of almost 200. A variation of 200:1 separated the top programmer from the poorest performers. The top 27% of the programmers did 78% of the work.”
When such results are considered, it is clear we cannot estimate software cost and duration unless we have information regarding the skills of members of the development organization. The presence of even a few very good (or very bad) individuals can significantly affect a team’s productivity.

Another human factor that can affect estimation is that critical staff may decide to leave an organization during the development process. This can cause a schedule to slip.
The most common metric for the size of a project is source lines of code (SLOC). There are many problems with using lines of code as a size measure for estimation.

1. Coding is only a small part of the development effort. A process including specifying, planning, designing, implementing, integrating, and testing cannot be reflected simply in the number of lines of code in the final product.

2. Counting lines of code is unreliable since different languages have such diverse structure and syntax. In VB programmers use visual controls as well as lines of code to establish how the program will function.
3. It is not clear how to count lines of code. Do you just count executable lines of code or also data definitions? And what about comments?

4. How should reused code be counted?

5. SLOC penalizes high-level languages. Only if SLOC is converted into “equivalent lines” of code in the same language can SLOC produce a valid metric across different languages.

6. The number of lines in a product that has not yet been built is uncertain.
Allan Albrecht and his developed a metric for the size of a product based upon function points. Function points reflect units of software capability and complexity. This method was designed to be independent of programming languages. Because this is a measure of a product’s size, it can be used for cost and productivity estimation.

The calculation of function points is based upon five attributes of a program:

1. Inputs
2. Outputs
3. Interactive inquiries
4. External logical files
5. Internal logical files
An Example from Jones (1998)

“Consider a simple spell checker that has one input (the name of the file that needs to be examined), three outputs (the total number of words reviewed, the number of errors found and a listing of misspelled words), one inquiry (a user can interactively obtain the number of words processed thus far), one external file (the document that needs to be inspected) and one internal file (the dictionary). For this simple program the number of elements is $1 + 3 + 1 + 1 + 1 = 7$.”
In practice, calculating function points more complicated than merely counting the five types of attributes. Various weights to account for the complexity (low, average, or high) of each element are assigned. If all components of our spell checker were of medium complexity it would have 40 function points. The function point total is either increased or decreased with a multiplier to match the perceived intricacy of the entire system. This adjustment is based upon 14 factors.

The calculation of function points is complicated and requires specialists who have passed a certification exam. Perhaps because of this, function point analysis is common mostly in very large companies.
Because function points are independent of the language used they provide an accurate way to compare different software. And the function point total for the same program coded in different languages is the same.

Function points have the advantage of being able to be assessed before any lines of code are written and before a language is chosen. Being able to estimate costs and schedules from function points gives the advantage of being able to accurately estimate sooner.
Empirical studies using function points have led to useful rules of thumb for development organizations. Jones (1998) reports that raising the function point total of a system to the 1.25 power gives a ballpark estimate of the number of errors that will have to be removed. Dividing the number of function points by 150 approximates the number of programmers, software analysts and technicians who will be needed to develop the application. Raising the function point total to the .4 power gives a rough estimate of the time in months that staff will need to complete the project.

Such rules of thumb are ballpark estimates, but do provide starting points for estimation.
Techniques of Cost and Effort Estimation

1. Expert Judgement by Analogy

A number of experts are usually consulted. Each expert arrives at an estimate by comparing the project at issue with other projects the expert has been involved in developing. Similarities and differences with the previous projects are noted. Predictions of the experts may be reconciled using the Delphi technique. Here the estimates and rationales are distributed to the other experts who then produce a second estimate either with or without group discussion.
Another form of expert estimation is bottom-up estimating associated with work breakdown structures. Here the estimator breaks the project into its component tasks and then estimates how much effort will be required to carry out each task. Breaking a project down into component subtasks that can be executed by a single person in a week or two is suggested. When probabilities are assigned to the costs associated with each individual element, an overall expected value can be determined from the bottom up for total project development cost. Expertise comes into play with this method in determining the most useful specification of the components within the structure and of the probabilities associated with each component. This type of estimating is most appropriate at the later, more detailed stages of project planning.
“Expertise-based techniques are useful in the absence of quantified, empirical data. They capture the knowledge and experience of practitioners seasoned within a domain of interest, providing estimates based upon a synthesis of the known outcomes of all past projects to which the expert is privy or in which he or she participated. The obvious drawback to this method is that an estimate is only as good as the expert’s opinion, and there is no way usually to test that opinion until it is too late to correct the damage if that opinion proves to be wrong.”

Boehm and Abts, 1999
“Accurate software estimating is too difficult for simple rules of thumb... In a comparative study of 50 manual estimates and 50 estimates produced by tools, I found two significant results. Manual estimates were wrong more than 75% of the time, and the errors were almost always on the side of excessive optimism - that is, they significantly underestimated both schedules and costs. Automated estimating tools came far closer to matching historical data, and errors tended toward conservatism - that is, they predicted slightly higher costs and longer schedules. This is the ‘fail safe’ mode of software estimation.”

Jones (1996)
2. Algorithmic Cost Estimation Models

A metric such as lines of code or function points and other factors are input into a model for determining product cost. The cost and effort needed to develop a project is related to variables mainly associated with characteristics of the final system. This approach has the advantage of being unbiased. Many algorithmic estimation models have been developed. Many of them are proprietary models and hence cannot be compared and contrasted in terms of model structure. Their form is determined either by theory or experimentation.
Examples of Algorithmic methods:
• The Software Life-cycle Model (SLIM)
  http://www.qsm.com

• Checkpoint
  http://www.spr.com/html/checkpoint.htm

• PRICE-S
  http://www.pricesystems.com

• SEER-SEM
  http://www.gaseer.com
COCOMO and COCOMOII

The Constructive Cost Model (COCOMO) is the most popular and best known cost estimation models. The Basic COCOMO model expressed development effort strictly as a function of the size (in thousands of source lines of code) and class of software being developed. The Intermediate COCOMO model enhances the effort equation of the Basic model by including 15 cost drives, known as effort adjustment factors.
COCOMOII was recently developed because COCOMO was outmoded by changes in the technologies used in developing software and by changes in the life-cycles used. For example, it used lines of code as an input but this was being made irrelevant by GUI builders. COCOMOII varies inputs according to the point in the development process when the estimate is being made. Function points form the size input for the COCOMOII Early Decision model.

COCOMOII Web sites:
http://www.softstarsystems.com
http://www.costxpert.com
http://sunset.usc.edu/research/COCOMOII
What do estimating tools do and how do they do it?

Software estimation tools require users to input a description of their project. Things the user may be asked to describe include:

- The size of the software in lines of code, function points, or some other metric
- Amount of reuse anticipated
- The type of software being developed, i.e. real time, operating systems, Web development, information system, etc.
- The software operating platform, i.e. commercial, military, ground, air, space, or desktop.
- A quantification of the organization’s software development productivity
Given these inputs the tool derives a cost and usually a schedule estimate for the project. The estimating relationships used are cost relationships derived from regression of actual data, analogies comparing input parameters to existing project databases, algorithms derived from theoretical research, or some combination of these methodologies.

Most tools also require input about the software development environment (programming language, tools) and the software development experience of the team in making cost and schedule estimates.
Estimation tools often provide information beyond cost and schedule estimates. Some tools have the capacity to predict latent defects in the delivered product and then use this information to predict maintenance costs. This allows project managers to consider total cost of product ownership rather than just development cost.
Many estimation tools also provide a risk analysis on the cost and schedule estimates so those estimates can be accompanied by a confidence level. This is done by asking the user to specify confidence levels for certain input levels (I.e. low, medium, high). The specified input uncertainties are used to replace point estimates with an output distribution. Then the developer can see the likelihood of achieving a particular cost or schedule.
Estimation Tools are Critical in Process Improvement

Software estimation technology can play an important role in an organization’s drive towards a higher CMM process maturity level (Minkiewicz, 2000). CMM standards for various levels call for standardization of estimating, data collection, quality control, measurement, and analysis.

Several CMM Key Process Areas (KPA’s) have requirements that an estimating tool will help meet.
Level 2 KPA’s

“Software estimates are documented for use in planning and tracking software estimates.” The software plan is expected to include size estimates for all work products, along with cost and schedule estimates. A software-estimating tool with the capability to estimate software size, cost, and schedule provides a way to institutionalize these estimating practices. An estimation tool provides the ability to estimate software cost and schedule consistently and logically. It puts software projects into a common framework so that information learned from one project can be applied to others.
Level 3 KPA’s

Here the focus shifts from achieving a repeatable to a defined process and from a project to an organizational level. There is a coordination of process activities at the organizational level. A key part in achieving this is building the organization’s software process database, which collects process and project data. Estimating tools require inputs to develop a framework that describes product and project characteristics (such as functionality, quality, size, complexity, and reuse) in such a way that permits comparisons across the organization.
KPA’s for Level 3 require that an organization use data in the software project database for software planning and estimating. This calibration process is automated in many estimating tools. In using an estimation tool the organization has already committed to storing data in the process database in a way that makes comparisons possible at the organizational level. The tool has the capacity to look at this historical data, which describes the past performance of the organization, and use it to improve estimates from that point onwards.
Level 4 and 5 KPA’s

Many software estimation tools have features that help meet requirements for quality management and continual process improvements. An estimation tool that provides an estimation of defects per size measure provides the basis for a process that allows for the development of a quality plan through trade-off analyses between quality, schedule, and content goals.
Suppose you have an organizational goal to reduce latent defects in your delivered software by 20%. At level 4 you have enough data in the software process database to evaluate defects delivered in the past and calibrate the estimating tool’s defect estimate. Once this is done, you can evaluate every proposed project for estimated defects and make corrections to the project plan. This makes possible a plan to extend the development schedule or reduce the amount content in a given time frame until the quality goal is met.
What Have We Learned About Software Estimation

From Putnam and Myers, 2000; Stutzke, 2000

• It takes more effort per source line of code to build larger systems than to build smaller systems. Large systems generally are more complex than small systems. The relationship between their parts are more intricate.

• At any one system size, the effort required to build it varies widely. For example collected project data at the size of 100,000 SLOC, effort ranges from a low of 10 person months to a high of 2,500 person months.
• The nonlinearities observed with effort also apply to time. Thus, development time per SLOC increases with system size. Development time varies widely at each system size. For instance, at about 10,000 SLOC, development time ranges from about two months to 80 months.

• There is a certain minimum development time. You cannot beat that minimum time by simply adding more people. Knowing what the minimum development time is for a prescribed system can aid in the decision process. For example, if you are contemplating set of bids you can throw out those that come in under the minimum time.
• You can trade off time and effort. By extending development time you can reduce effort and defects. The upper limit is about 130% of the minimum time. Beyond that point, further reductions in effort and defects are small. Within that range, you can balance time, effort, and defects to fit your business pressures.

• Small staff size is better. Other things being equal, a small staff is more efficient than a large staff. The reason is the need to communicate complex concepts and mental models between the workers, and the need to coordinate the activities of workers who are performing a set of complex, interrelated tasks. Data on 491 medium-sized projects showed that the two-to five person teams completed projects of comparable size with about one-third of the effort of the seven-14 person teams.
• Fewer defects are better. There is a time quality tradeoff. When you extend your development time beyond the minimum, you score fewer defects.

• You can replan a project midway based upon metrics. If you accumulate metrics during a project, you can use them to plan a new schedule and effort to completion. A new plan may extend the schedule or call for more staff. Perhaps the features the product offers can be reduced. Metrics and estimating can’t tell you what to do. They can provide you with information to do what your circumstances require of you.
• Don’t rely on a single model for all estimates. Because all models are approximations, we should compare estimates from two or more models.