Software Project Estimation

Effective software project estimation is one of the most challenging and important activities in software development. Proper project planning and control is not possible without a sound and reliable estimate. As a whole, the software industry doesn't estimate projects well and doesn't use estimates appropriately. We suffer far more than we should as a result and we need to focus some effort on improving the situation.

Under-estimating a project leads to under-staffing it (resulting in staff burnout), under-scoping the quality assurance effort (running the risk of low quality deliverables), and setting too short a schedule (resulting in loss of credibility as deadlines are missed). For those who figure on avoiding this situation by generously padding the estimate, over-estimating a project can be just about as bad for the organization! If you give a project more resources than it really needs without sufficient scope controls it will use them. The project is then likely to cost more than it should (a negative impact on the bottom line), take longer to deliver than necessary (resulting in lost opportunities), and delay the use of your resources on the next project.

Software Project Estimation 101

The four basic steps in software project estimation are:

1) Estimate the size of the development product. This generally ends up in either Lines of Code (LOC) or Function Points (FP), but there are other possible units of measure. A discussion of the pros & cons of each is discussed in some of the material referenced at the end of this report.
2) Estimate the effort in person-months or person-hours.
3) Estimate the schedule in calendar months.
4) Estimate the project cost in dollars (or local currency)

Estimating size

An accurate estimate of the size of the software to be built is the first step to an effective estimate. Your source(s) of information regarding the scope of the project should, wherever possible, start with formal descriptions of the requirements - for example, a customer's requirements specification or request for proposal, a system specification, a software requirements specification. If you are [re-]estimating a project in later phases of the project's lifecycle, design documents can be used to provide additional detail. Don't let the lack of a formal scope specification stop you from doing an initial project estimate. A verbal description or a whiteboard outline are sometimes all you have to start with. In any case, you must communicate the level of risk and uncertainty in an estimate to all concerned and you must re-estimate the project as soon as more scope information is determined.

Two main ways you can estimate product size are:

1) By analogy. Having done a similar project in the past and knowing its size, you estimate each major piece of the new project as a percentage of the size of a similar piece of the previous project. Estimate the total size of the new project by adding up the estimated sizes of each of the pieces. An experienced estimator can produce reasonably good size estimates by analogy if accurate size values are available for the previous project and if the new project is sufficiently similar to the previous one.

2) By counting product features and using an algorithmic approach such as Function Points to convert the count into an estimate of size. Macro-level "product features" may include the number of subsystems, classes/modules, methods/functions. More detailed "product features" may include the number of screens, dialogs, files, database tables, reports, messages, and so on.

Estimating effort

Once you have an estimate of the size of your product, you can derive the effort estimate. This conversion from software size to total project effort can only be done if you have a defined software development lifecycle and development process that you follow to specify, design, develop, and test the software. A software development project involves far more than simply coding the software – in fact, coding is often the smallest part of the overall effort. Writing and reviewing documentation, implementing prototypes, designing the deliverables, and reviewing and testing the code take up the larger portion of overall project effort. The project effort estimate requires you to identify and estimate, and then sum up all the activities you must perform to build a product of the estimated size.

There are two main ways to derive effort from size:

1) The best way is to use your organization's own historical data to determine how much effort previous projects of the estimated size have taken. This, of course, assumes (a) your organization has been documenting actual results from previous projects, (b) that you have at least one past project of similar size (it is even better if you have several projects of similar size as this reinforces that you consistently need a certain level of effort to develop projects of a given size), and (c) that you will follow a similar development lifecycle, use a similar development methodology, use similar tools, and use a team with similar skills and experience for the new project.

2) If you don't have historical data from your own organization because you haven't started collecting it yet or because your new project is very different in one or more key aspects, you can use a mature and generally accepted algorithmic approach such as Barry Boehm's COCOMO model or the Putnam Methodology to convert a size estimate into an effort estimate. These models have been derived by studying a significant number of completed projects from various organizations to see how their project sizes mapped into total project effort. These "industry data" models may not be as accurate as your own historical data, but they can give you useful ballpark effort estimates.

Estimating schedule

The third step in estimating a software development project is to determine the project schedule from the effort estimate. This generally involves estimating the number of people who will work on the project, what they will work on (the Work Breakdown Structure), when they will start working on the project and when they will finish (this is the "staffing profile"). Once you have this information, you need to lay it out into a calendar schedule. Again, historical data from your organization's past projects or industry data models can be used to predict the number of people you will need for a project of a given size and how work can be broken down into a schedule.

If you have nothing else, a schedule estimation rule of thumb [McConnell 1996] can be used to get a rough idea of the total calendar time required:

$$\text{Schedule in months} = 3.0 * (\text{effort-months})^{1/3}$$

Opinions vary as to whether 2.0 or 2.5 or even 4.0 should be used in place of the 3.0 value – only by trying it out will you see what works for you.

Estimating Cost

There are many factors to consider when estimating the total cost of a project. These include labor, hardware and software purchases or rentals, travel for meeting or testing purposes, telecommunications (e.g., long-distance phone calls, video-conferences, dedicated lines for testing, etc.), training courses, office space, and so on.

Exactly how you estimate total project cost will depend on how your organization allocates costs. Some costs may not be allocated to individual projects and may be taken care of by adding an overhead value to labor rates ($ per hour). Often, a software development project manager will only estimate the labor cost and identify any additional project costs not considered "overhead" by the organization.

The simplest labor cost can be obtained by multiplying the project's effort estimate (in hours) by a general labor rate ($ per hour). A more accurate labor cost would result from using a specific labor rate for each staff position (e.g., Technical, QA, Project Management, Documentation, Support, etc.). You would have to determine what percentage of total project effort should be allocated to each position. Again, historical data or industry data models can help.

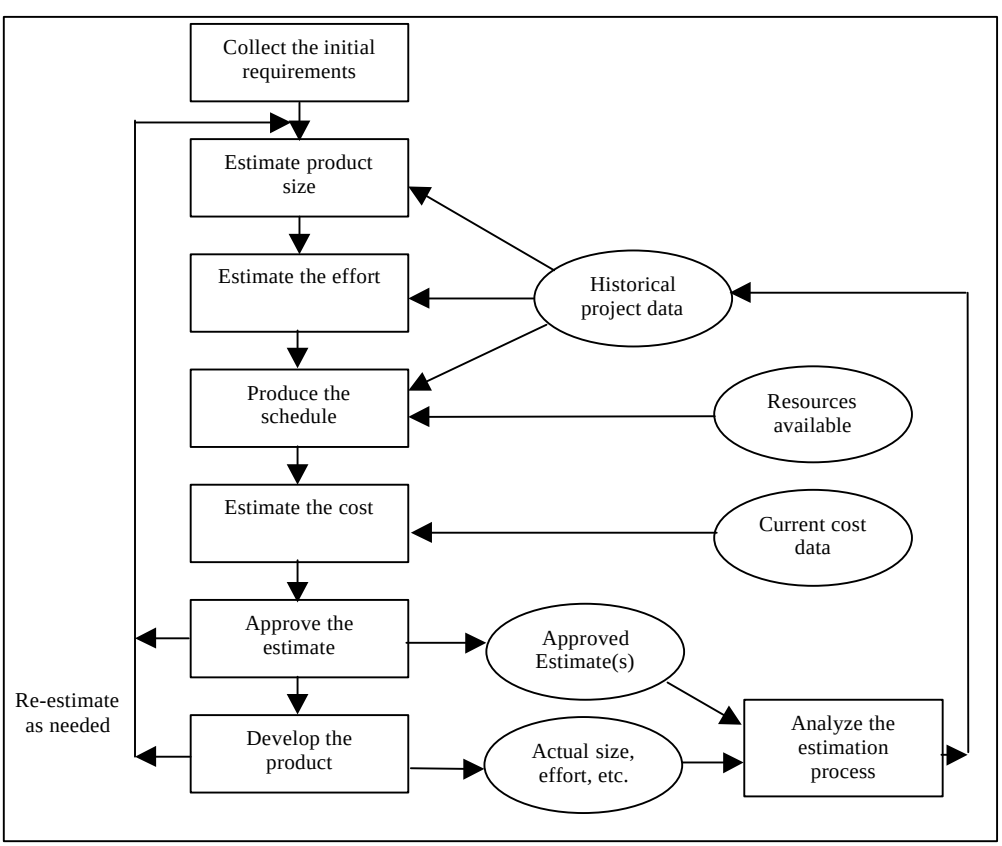

Figure 1 – The Basic Project Estimation Process

Working Backwards from Available Time

Projects often have a delivery date specified for them that isn't negotiable - "The new release has to be out in 6 months"; "The customer's new telephone switches go on-line in 12 months and our software has to be ready then". If you already know how much time you have, the only thing you can do is negotiate the set of functionality you can implement in the time available. Since there is always more to do than time available, functionality has to be prioritized and selected so that a cohesive package of software can be delivered on time.

Working backwards doesn't mean you skip any steps in the basic estimation process outlined above. You still need to size the product, although here you really do have to break it down into a number of pieces you can either select or remove from the deliverable, and you still need to estimate effort, schedule, and cost. This is where estimation tools can be really useful. Trying to fit a set of functionality into a fixed timeframe requires a number of "what if" scenarios to be generated. To do this manually would take too much time and effort. Some tools allow you to play with various options easily and quickly.

Understanding an Estimate's Accuracy

Whenever an estimate is generated, everyone wants to know how close the numbers are to reality. Well, the bottom line is that you won't know exactly until you finish the project – and you will have to live with some uncertainty. Naturally, you will want every estimate to be as accurate as possible given the data you have at the time you generate it. And of course you don't want to present an estimate in a way that inspires a false sense of confidence in the numbers.

What do we mean by an "accurate" estimate? Accuracy is an indication of how close something is to reality. Precision is an indication of how finely something is measured. For example, a size estimate of 70 to 80 KLOC might be both the most accurate and the most precise estimate you can make at the end of the requirements specification phase of a project. If you simplify your size estimate to 75000 LOC it looks more precise, but in reality it's less accurate. If you offer the size estimate as 75281 LOC, it is precise to one LOC but it can only be measured that accurately once the coding phase of the project is completed and an actual LOC count is done.

If your accurate size estimate is a range, rather than a single value, then all values calculated from it (e.g., effort, schedule, cost) should be represented as a range as well. If, over the lifetime of a project, you make several estimates as you specify the product in more detail, the range should narrow and your estimate should approach what will eventually be the actual cost values for the product or system you are developing (Figure 2).

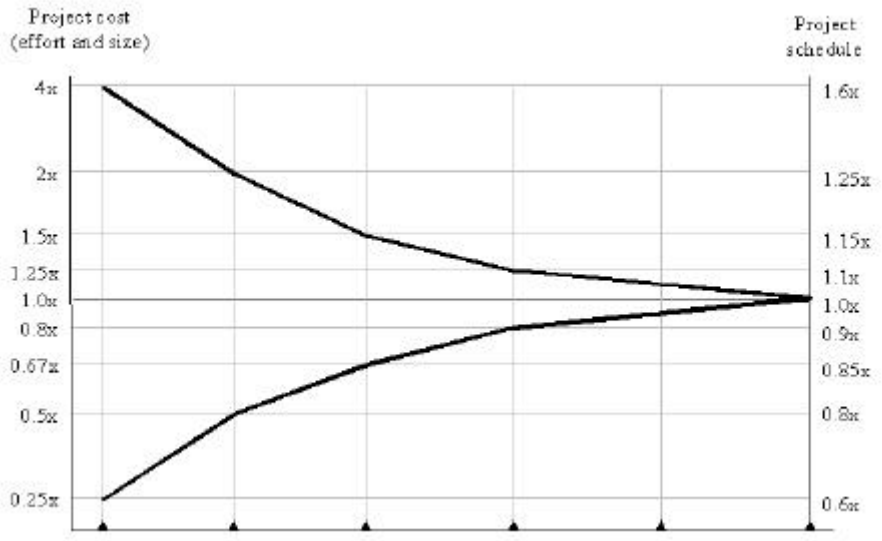

Figure 2 – Estimate Convergence Graph. Source: "Rapid Development" (McConnell 1996); Adapted from "Cost Models for Future Life Cycle Processes: COCOMO 2.0" (Boehm et al. 1995).

Of course, you must also keep in mind other important factors that affect the accuracy of your estimates, such as:

- the accuracy of all the estimate's input data (the old adage, "Garbage in, Garbage out", holds true)
- the accuracy of any estimate calculations (e.g., converting between Function Points and LOC has a certain margin of error)
- how closely the historical data or industry data used to calibrate the model matches the project you are estimating
- the predictability of your organization's software development process, and
- whether or not the actual project was carefully planned, monitored and controlled, and no major surprises occurred that caused unexpected delays.

Understanding the Tradeoffs

Once you've generated a project estimate, the real work begins – finding some combination of functionality, schedule, cost and staff size that can be accepted by management and customers! This is where a solid understanding of the relationships between these variables is so important, and where being armed with different project estimates illustrating the tradeoffs is very useful for establishing what the limits are.

Here are a few facts of life you need to remember during the estimate "adjustment" phase:

- If you lengthen the schedule, you can generally reduce the overall cost and use fewer people. Sometimes you only have to lengthen the schedule by a few weeks to get a benefit. Usually management and customers don't want a long delay, but see how much "extra" might be acceptable. Many people don't consider generating an estimate option that lengthens the schedule to see what effect it has unless they are driven to it in an attempt to reduce cost or staff size.

- You can only shorten a schedule three ways. You can reduce the functionality (reducing the effort by doing less), increase the number of concurrent staff (but only if there are tasks you could now do in parallel to take advantage of this!), or keep the number of staff constant but get them to work overtime.

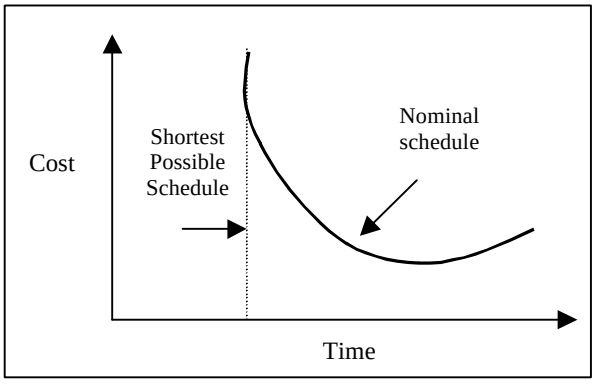

Figure 3 – Relationship between cost and schedule on a software project. Source: "Rapid Development" (McConnell 1996). The cost of achieving the nominal schedule is much less than the cost of achieving the shortest possible schedule.

If you can't reduce the functionality, choosing one of the two remaining alternatives is going to cost you. It might cost you a lot more than you can afford to pay depending on just how much you want to shrink the schedule (see Figure 3). And it might not work! Remember the "adding people to a late project only makes it later" rule? Well, the same principle applies here – you can add more people, but the amount of work also goes up because you now have additional communication and management overhead. If you rely on a lot of overtime to reduce the schedule, you have to remember that productivity may increase in the short term but will go down over the long term because people will get tired and make more mistakes.

- There is a shortest possible schedule for any project and you have to know what it is. You can only shrink a schedule so far given the functionality you are required to implement, the minimum process you have to follow to develop and test it, and the minimum level of quality you want in the output. Don't even think of trying to beat that limit!

- The shortest possible schedule may not be achievable by you. To achieve the shortest possible schedule your project team had better all be highly skilled and experienced, your development process had better be well defined and mature, and the project itself has to go perfectly. There are not many organizations that can hope to make the shortest possible schedule, so it's better not to aim for this. Instead, you need to determine what your shortest achievable schedule is (also known as the "nominal" schedule). Data from past projects is your best source of information here.

- Always keep in mind the accuracy of the estimate you are attempting to adjust. If your schedule estimate is currently "5 to 7 months" then a small change, for example 2 weeks, either way doesn't mean much yet. You can only adjust the schedule in increments that have some significance given the accuracy of the estimate.

It's interesting to observe the reactions of people learning to estimate projects who are asked to do a number of different estimates for a project using a variety of options. When they analyze the results, most people are startled by the consequences of different tradeoffs. For example, the following tables provide 3 different estimate options for a 75 KLOC project:

The difference between the nominal schedule and the shortest schedule for the project is only a little over two months, but to achieve the shortest schedule the peak staff has to increase by almost 10 people and the cost increases by over $870,000! These results should cause someone to ask if a 2-month decrease in the schedule is worth the cost, and if 10 additional people can be found in time to help achieve it. For some projects, a schedule decrease may be required at any cost; for others, it won't be.

Not all projects have such dramatic differences between estimate options, but the size-effort-schedule-staff-cost relationship follows some basic rules that you can't circumvent. Having various options available as you discuss a project estimate ensures everyone involved can see those basic rules in action and can make properly informed decisions.

Nominal Plan

Management Metric	Planning Value
Effort (staff months)	40
Schedule (calendar months)	12.4
Cost	$605,868
Peak Staff (people)	4.8
Average Staff (people)	3.2

Shortest-Schedule Plan

Management Metric	Planning Value
Effort (staff months)	97
Schedule (calendar months)	10.0
Cost	$1,479,170
Peak Staff (people)	14.6
Average Staff (people)	9.8

Least-Cost Plan

Management Metric	Planning Value
Effort (staff months)	14
Schedule (calendar months)	16.2
Cost	$212,131
Peak Staff (people)	1.3
Average Staff (people)	0.9

Figure 4 – Three different estimates for a 75 KLOC Project

The Trouble with Estimates

While effective software project estimation is absolutely necessary, it is also one of the most difficult software development activities. Why is it so hard?

The following lists some of the things that make it hard – and they're things that we need to overcome:

- Estimating size is the most difficult (but not impossible) step intellectually, and is often skipped in favor of going directly to estimating a schedule. However, if you haven't thought through what you are being asked to build you really don't have a good base from which to predict a schedule or to evaluate how scope changes may affect the schedule.

- Customers and software developers often don't really recognize that software development is a process of gradual refinement and that estimates made early in a project lifecycle are "fuzzy". Even good estimates are only guesses, with inherent assumptions, risks, and uncertainty, and yet they are often treated as though they are cast in stone. What can help is offering estimates as a range of possible outcomes by saying, for example, that the project will take 5 to 7 months instead of stating it will be complete on June 15. Beware of committing to a range that is too narrow as that's about as bad as committing to a definite date! Alternatively, you could include uncertainty as an accompanying probability value by saying, for example, that there is an 80% probability that the project will complete on or before June 15.

- Organizations often don't collect and analyze historical data on their performance on development projects. Since the use of historical data is the best way to generate estimates for new work, it is very important to establish some fundamental project metrics that you collect for every project.

- It is often difficult to get a realistic schedule accepted by management and customers. Everyone wants things sooner rather than later, but for any project there is a shortest possible schedule that will allow you to include the required functionality and produce a quality output. You have to determine what you can do in a given period of time and educate all concerned about what is and what is not possible. Yes, the impossible has been known to happen from time to time, but it's rare and very costly, and we count on it far more than it is prudent to do so!

Maintenance & Enhancement Projects vs. New Development

The software industry does far more maintenance and enhancement work on existing products than completely new development. Most maintenance projects are a combination of new development and adaptation of existing software. Although the estimation steps outlined above can still apply to maintenance and enhancement projects, there are some special issues that have to be considered such as:

- When sizing new development for a maintenance project you have to keep in mind that inserting this new functionality will only be feasible if the product's existing architecture can accommodate it. If it cannot, the maintenance effort must be increased to rework the architecture.

- It's tricky to attempt to size adaptation work in the same manner as new work. An experienced individual estimating maintenance effort by analogy is a more common approach than attempting to size adaptation work in LOC or Function Points and then converting size to effort (although approaches to this have been discussed; for example, see [Putnam 1992]).

- Estimation models that are calibrated to produce effort and schedule estimates for new development projects assume everything is created from scratch. This isn't the case for maintenance projects where you are modifying a certain amount of existing documentation, code, test cases, etc. Using these models may tend to over-estimate maintenance projects.

- Often, maintenance work has fixed delivery dates (e.g., a maintenance release every 6 months or once a year) and is done by a fixed number of people (i.e., an allocated maintenance team) so estimates have to deal with fitting work into a fixed timeframe with a constant staffing level.

Some existing estimation models do attempt to address maintenance concerns. At the moment though, there is a lot more support, guidance and discussion available regarding new development estimation than there is on maintenance and enhancement estimation. Hopefully this will change because so much help is needed in this area.

Estimating Small Projects

Many people work on small projects, which are generally defined as a staff size of one or two people and a schedule of less than six months. Existing industry-data project estimation models are not calibrated from small projects and so are of little or no use here unless they can be adequately adjusted using an organization's small project historical data.

Estimates for small projects are highly dependent on the capabilities of the individual(s) performing the work and so are best estimated directly by those assigned to do the work. An approach such as Watts Humphrey's Personal Software Process (PSP) [Humphrey 1995] is much more applicable for small project estimation.

Estimating a "New Domain" Project

How do you estimate a project in a new application domain where no one in your organization has any previous experience? If it's a leading-edge (or "bleeding-edge"!) project, no one else has any previous experience either. The first time you do something you are dealing with much more uncertainty and there is no way out of it except to proceed with caution and manage the project carefully. These projects are always high risk, and are generally under-estimated regardless of the estimation process used [Vigder 1994]. Knowing these two facts, you must (a) make the risks very clear to management and customers, (b) avoid making major commitments to fixed deadlines, and (c) re-estimate as you become more familiar with the domain and as you specify the product in more detail.

Selecting a project lifecycle which best accommodates the uncertainty of new-domain projects is often a key step that is missing from the development process. An iterative life cycle such as the Incremental Release Model where delivery is done in pieces, or the Spiral Model where revisiting estimates and risk assessment is done before proceeding into each new step, are often better approaches than the more traditional Waterfall Model.

Some Estimating Tips

- Allow enough time to do a proper project estimate – rushed estimates are inaccurate, high-risk estimates! For large development projects, the estimation step should really be regarded as a mini-project.

- Where possible, use documented data from your organization's own similar past projects. It will result in the most accurate estimate. If your organization has not kept historical data, now is a good time to start collecting it.

- Use developer-based estimates. Estimates prepared by people other than those who will do the work will be less accurate.

- Use at least one software estimation tool. Estimation tools implement complex models that would take significant time to learn to apply manually. They also make sure you don't forget anything, and allow you to tune an estimate quickly and relatively painlessly.

- Use several different people to estimate and use several different estimation techniques (using an estimation tool should be considered as one of the techniques), and compare the results. Look at the convergence or spread among the estimates. Convergence tells you that you've probably got a good estimate. Spread means that there are probably things that have been overlooked and that you need to understand better. The Delphi approach or Wideband-Delphi technique [Boehm 1981] can be used to gather and discuss estimates using a group of people, the intention being to produce an accurate, unbiased estimate.

- Re-estimate the project several times throughout its lifecycle. As you specify the product in more detail, your estimates should begin to approach what you will actually use to complete the project.

- Create a standardized estimation procedure that all involved can and do buy into. That way you can't argue about the outputs, only the inputs, and your effort is spent productively understanding the scope and cost drivers for the project.

- Focus some effort on improving your organization's software project estimation process. As each project is completed, compare actuals to estimates – how well did you do in predicting the project's effort and schedule? What did you miss? Where could you improve?

Software Project Estimation Tools

Estimation tools may be stand-alone products or may be integrated into the functionality of larger project management products. Estimation tools may just support the size estimation process, or just the conversion of size to effort, schedule and cost, or both. Project management tools have been discussed in an earlier ADS newsletter (November 1997). Tools that support just size estimation include LOC counters, Function Point analysis programs, and even requirements capture and management applications. This section of this report just focuses on estimation tools that are stand-alone products and support the conversion of size to effort etc.

No estimation tool is the "silver bullet" for solving your estimation problems. They can be very useful items in your estimation toolkit, and you should seriously consider using one (or more), but their output is only as good as the inputs they are given and they require you to have an estimation and software development process in place to support them. Beware of any vendor claiming their tool is able to produce estimates within +/- some small percentage of actuals unless they also highlight all the things you must be able to do in a predictable manner and what must go right during a project to ensure an estimate can be that accurate.

There are a variety of commercial and public domain estimation tools available. Searching for software project estimation tools on the web isn't as straightforward as one might expect. A mix of keywords and search engines needs to be used to discover about 80% of the tools and web-sites where tool lists were available identified the rest. Web-based information on the capabilities and pricing of the tools is variable and occasionally very superficial, so some phone calls and email should be used to augment what is gleaned from the web.

The following provides a summary of the important features and criteria you should consider when evaluating a software project estimation tool. Figure 3 provides a context for the discussion.

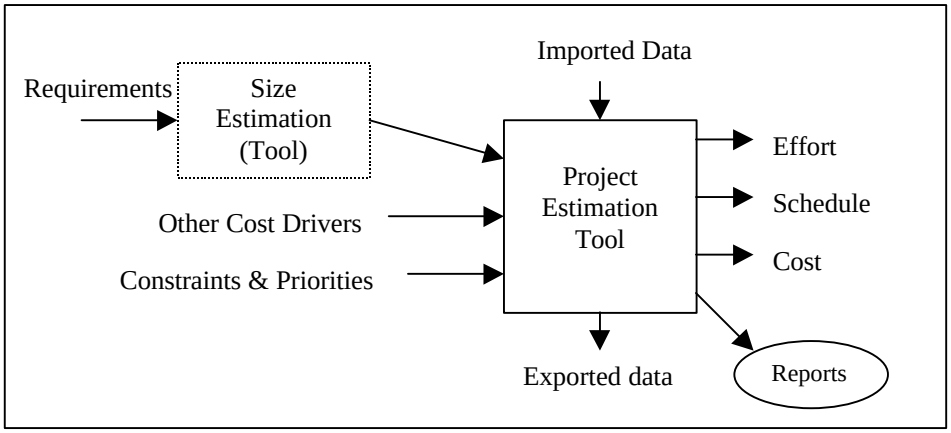

Figure 3 – Estimation Tool Context

Price

Commercial estimation tools can be categorized as either "for rent" (you pay an annual fee) or "for purchase" (one-time fee), and they come in 3 price ranges: affordable ($1000 or less), mid-range ($1001-$5000), or expensive ($5001 to $20000 or more). Probably only larger organizations or large projects would consider the mid-range or high-priced tools. The tools under $1000 implement non-proprietary models published by others (e.g., COCOMO) and may lack some of the functionality and extensive support of the expensive options, but can still generate more than adequate estimates.

Platform and Performance

Does it run on your current hardware/software? Does it run on more than one platform? How much RAM and disk space is required? Will its database handle the amount of historical data and the size and number of project estimates you will be entering?

Ease of Use and Documentation

Can you generate an initial project estimate easily the first time you use the tool, or do you have to study the underlying model in detail for days first learning all the acronyms and struggling with attribute definitions? Can you tailor your estimates easily? Do the user manual and help text provide an understanding how to use the tool to generate project estimates, as well as a simple list of what general functionality the tool possesses? Is sample project data available?

Networking Capability

Is there a common, shared database so that multiple users can access and add to the historical project data, and view or update estimates (assuming this is important to you)?

Upgrades

Model data shouldn't be static – as new programming languages and new development paradigms appear, as different kinds of development projects are studied, updates to the model data in the tool should be made. Does the vendor make model updates available to customers? Is the vendor committed to making continuous enhancements that offer new functionality and support new platforms etc.? What do these upgrades cost?

Support

It is important to understand that although estimation tools are becoming more affordable and easier to use, the model(s) they implement are quite complex and you may have questions or need some guidance now and then. Does the vendor provide technical support and a means for asking "how to" questions? Does the vendor offer estimation training courses that extend beyond just how to use the tool or can they recommend supporting courses and training/reading material?

How Scope/Size is Specified

Flexibility is the key – you may start out estimating a project's size a certain way, but as you learn more about the specification of a particular product or as you become more skilled at estimating and branch out into other sizing techniques, you want a tool that supports your needs.

What options does the tool provide re. specifying the size estimate? Can you enter either LOC or Function Points? Can you specify GUI components, number of classes/methods or modules/functions? Is the size value entered merely as a single number (e.g., 55000 LOC; 345 Function Points), a range of numbers (e.g., Low: 45000 LOC; Expected: 55000 LOC; High: 65000 LOC), or can the size estimate be divided into a number of "modules" or "work packages" that you estimate in whichever way suits that particular component?

Estimation Model(s) Supported

Some tools use one or more proprietary models where little detailed information is published; others use non-proprietary models where you can purchase a book and/or download detailed information from the web to learn more. It takes a lot of resources to develop a sophisticated model of software development so its not surprising that there are only a handful of models.

Regardless of how much you can learn about the tool's internal algorithms, what you must determine is whether or not the estimates generated by the tool are useful in estimating your organization's type of software development projects. Parametric models typically have a bias – for example, some suit a military development process while others suit commercial development. The only way to quickly become confident that a tool can give you valuable results is to obtain an evaluation or demo copy and estimate previous projects where actuals are known. Compare the estimates from the tool with what you know about previous projects and see if the results are "in the ballpark".

Assess whether the tool allows you to capture historical information on your past projects and how it requires you to enter it. Some tools can be calibrated to your projects only by modifying the underlying model data (i.e., you have to derive the values yourself); others allow you to simply enter project metrics like actual size, effort, schedule and then the tool derives the model data changes.

Does the tool support the estimation of maintenance & enhancement projects? Does it have support for object-oriented, COTS, software re-use or other issues important to your projects?

Other Cost Drivers

The models generally allow you to specify values for a number of cost, or productivity drivers (e.g., staff capabilities and experience, lifecycle requirements, use of tools, etc.) in order to tailor the estimate to your organization and your project's particular situation. What cost drivers are available and are the values you can set them to useful for your situation?

Constraints and Priorities

Does the tool allow you to specify constraints (e.g., Maximum Schedule=12 months; Peak Staff=10) when calculating an estimate? Does the tool allow you to specify priorities (e.g., "Shortest possible schedule has highest priority"; "Lowest number of staff has highest priority") when calculating an estimate?

Outputs Generated

Look for a tool that has functionality which shows options, probabilities and ranges. Tools using Monte Carlo simulation to generate estimates with different probabilities provide interesting insight into the volatility of the development process.

Reports should help you clearly present and discuss estimate options with customers and management. What kinds of reports are generated and are they useful to you? Can you obtain a softcopy of the reports so that you can add material to them or include them easily in other project documents you generate?

Import/Export Capability

Possible imports include things like module-by-module size estimates, historical project data, and updated model data. Possible exports include schedule, WBS, and staffing information to project management software like MS-Project or to a spreadsheet program like Excel.

Conclusion

There is no quick fix that will immediately make us better estimators and users of estimates. Effective estimates come about as a result of process definition and improvement, education and training, good project management, use of proper tools and techniques, measurement, sufficient resources, and sheer hard work. Depending on what the situation is when you start, and how long a typical project lasts in your organization, it could be several years before you've had enough time and project cycles to establish the basics from which better estimates are consistently made. Trying to set up everything in a week is equivalent to trying to build Rome in a day.

But don't be discouraged by this! There are things you can do right now to make a difference to your current project, and there are actions you can take to make your next project better.

Assuming you are past the planning stage of your current project and you have little time to spare, here are some important actions you can take on this project to start improving your estimation process:

- Re-estimate the project at several key stages (e.g., after completion of requirements specification, after completion of architectural design, after completion detailed design).

- At the end of the project, record the actual values (or get as close as you can) for size, effort, schedule, cost, staffing. Start your historical database.

- At the end of the project, review your estimate(s)/actuals and evaluate what you did right and how you might improve in future. Use what you learn here the next time you estimate.

Here are some important actions you can take for your next project:

- Review the current state of your software development process – is it chaotic, or does it have some order and structure that you generally follow? If it is chaotic to start with, or your process breaks down under pressure and you expect that to happen on this project, then any estimate you make had better take that into account. Even better, try to reduce the amount of chaos. Establishing a predictable development process isn't something you can do overnight, but every little bit of work you do in this area will help allow for better estimates and better project control.

- Create a first draft of an estimation procedure document and follow the procedure when estimating. See what works and what doesn't, and adjust as necessary. Note that there are templates around for creating such a document so you don't have to start from scratch.

- Allow enough time to do proper project estimates
- Decide when you should re-estimate the project and put tasks/milestones for re-estimation into the project plan
- Start the education of managers and customers re. the accuracy of estimates. Offer estimates as ranges, and explain the uncertainty and risks associated with them.
- Follow as many of the other estimation tips given earlier in this report as you can.

Try to make time to:

- work on defining, documenting and/or improving your software development process. A clearly defined, predictable development process is required so that your project estimates can be made on a firm development foundation. There are document templates and process definition guidelines, consultants and courses available to help you in this.
- investigate project estimation tools and use one (or more).

Small improvement steps, taken with care and attention, will lead you down the road to better project estimation and planning. In fact, taking small steps is often the only way to ensure permanent change occurs.

References

General Reading (includes Estimation and Project Planning)

- DeMarco, Tom, *Controlling Software Projects*, Prentice-Hall, 1982
- Goether, Wolfhart B., Elizabeth K. Bailey, Mary B. Busby, *Software Effort and Schedule Measurement: A framework for counting Staff-hours and reporting Schedule Information*, CMU/SEI-92-TR-021, 1992, http://www.sei.cmu.edu/publications/documents/92.reports/92.tr.021.html
- Humphrey, Watts, *A Discipline for Software Engineering*, Addison-Wesley, 1995
- McConnell, Steve, *Rapid Development – Taming Wild Software Schedules*, Microsoft Press, 1996
- McConnell, Steve, *Software Project Survival Guide*, Microsoft Press, 1998
- Vigder, M.R. & A.W. Kark, *Software Cost Estimation and Control*, 1994, http://wwwsel.iit.nrc.ca/abstracts/NRC37116.abs (full text available)

Sizing Lines of Code

- R. Park, *Software Size Measurement: A framework for counting source statements*, CMU/SEI-92-TR-20, 1992, http://www.sei.cmu.edu/publications/documents/92.reports/92.tr.020.html

Function Points

- Dreger, Brian, *Function Point Analysis*, Prentice-Hall, 1989
- Garmus, David & David Herron, *Measuring the Software Process*, Yourdon Press, 1996
- Jones, Capers, *Applied Software Measurement: Assuring Productivity and Quality*, McGraw-Hill, 1991
- Jones, Capers, *What are Function Points*, 1997, http://www.spr.com/library/0funcmet.htm
- Symons, Charles, *Software Sizing and Estimating: Mark II Function Point Analysis*, John Wiley, 1991
- International Function Point Users Group (IFPUG) web site: http://www.ifpug.org
- A function point FAQ: http://ourworld.compuserve.com/homepages/softcomp/fpfaq.htm

Estimation Models

- Boehm, Barry, *Software Engineering Economics*, Prentice-Hall, 1981 (original COCOMO)
- Putnam, Lawrence & Ware Myers, *Measures for Excellence: Reliable Software on Time, Within Budget*, Yourdon Press, 1992
- Putnam, Lawrence & Ware Myers, *Industrial Strength Software: Effective Management using Measurement*, IEEE Computer Society, 1997
- COCOMOII web site: http://sunset.usc.edu/COCOMOII/cocomo.html

Other Estimation Tool Summaries & Discussions

- Douglis, Charles, *Cost Benefit Discussion for Knowledge-Based Estimation Tools*, 1998, http://www.spr.com/html/cost_benefit.htm
- Giles, Alan E. & Dennis Barney, *Metrics Tools: Software Cost Estimation*, 1995, http://www.stsc.hill.af.mil/CrossTalk/1995/jun/Metrics.html
- Software Cost Estimation Web-Site (SCEW), http://www.ecfc.u-net.com/cost/index.htm
- Parametric Cost Estimating Reference Manual, http://www.jsc.nasa.gov/bu2/resources.html
- DoD Data & Analysis Center for Software, http://www.dacs.dtic.mil

About the Author

Kathleen Peters is an independent software engineering consultant with an M.Sc. in Computing Science and 15+ years of industry experience developing software and managing projects. She is currently working with Software Productivity Centre Inc. (SPC) in Vancouver, British Columbia, Canada. She also teaches software engineering courses at Simon Fraser University. Contact her at kpeters@spc.ca or petersk@istar.ca