Software estimation, enterprise-wide

Part II: Lifecycle of an estimate

Vitalie Temnenco

July 15, 2007

One thing I've learned in the course of my software project implementation work is that the number of people who can use sophisticated terminology borrowed from the estimation domain far outnumber those who can actually create accurate and precise estimates. Some of this has to do with an individual's domain expertise and biases, whether business, technical, or operational. It is also partly due to a lack of broad-based methodological knowledge and experience. There are many estimating tools and methods out there, but it is difficult to apply them effectively during the course of specific software projects.

From The Rational Edge: This article, Part 2 in a two-part series, provides advice on what estimation methods to apply at specific points in a software development project lifecycle, along with tips on how to apply these methods more effectively. Part 1 presents an overview of approaches, techniques, models, and tools used in estimating on software projects.

In Part 1 of this article I examined techniques, models, and tools that can be useful for estimating software development projects. Here, in Part 2, I will discuss when and how to make the best use of various estimation approaches in the course of a software development project lifecycle. I'll also discuss some general best practices that can help you avoid making (literally) costly mistakes.

Review of the software estimation process

Estimation was an integral part of enterprise planning long before IT arrived on the scene. As the first computers were installed and the first programs written, enterprise-savvy executives began to get a sense of what it took to develop software solutions, both cost-wise and time-wise. While past managers may have been able to know almost precisely "how much" and "how long" a software project would take, the IT industry's ongoing segmentation and increasing complexity has long since put an end to that idyll.
To reduce the risk associated with the inherent uncertainty of predicting future costs and timelines, executives attempted to build software development estimation practices around simple, tried-and-true techniques like Software Lines of Code (SLOC). These simplistic approaches worked for awhile, and are still being used here and there. But the results they produce have deteriorated steadily as actual project performance has become more and more dependent on the intake of new skills, while the growing complexity of software projects and programs has made them increasingly inapplicable.

In the software development industry today, business and technology executives are, on the whole, dissatisfied with their ability to accurately project costs and schedules. Almost every project today has cost or schedule overruns, and project failures are practically the norm. According to recent Cutter research, only 14% of companies reported good software schedule and budget estimation performance. The cost of developing business software solutions (I call it “solutioning”) is on the rise, yet the ability of many organizations to deliver quality end products on time and on budget is questionable, and even the most promising, risk-mitigating process methodologies often cannot save the day. The challenge of accurately estimating cost and schedule is certainly part of the problem.

**Estimation challenges and trends**

The issues that make it so difficult to accurately predict costs, timelines, and staffing requirements on software projects include:

- The increasing complexity and segmentation of technologies and skills
- A growing "productivity gap" between the best and worst software professionals, in terms of performance
- Ambiguity about what estimating technique to use
- The potential for misalignment between estimation techniques and organizational methodologies and best practices

These challenges are represented in Figure 1.

**Figure 1: Software development estimation challenges: These factors have prevented wide industrialization of estimation.**

**Rising technological complexity**

The IT marketplace has become increasingly segmented over time. More than a dozen hardware platforms, 25 or more operating systems, and in excess of 100 programming languages are...
now in use -- not to mention all the libraries, frameworks, middleware, etc., and all the possible combinations thereof. This bewildering complexity has rendered some estimation concepts, such as SLOC, impractical in many cases. Likewise, Function Points (FP) techniques are becoming increasingly difficult to apply as they now often require mapping between environments, significant expertise, and complex configuration of variables in order to be effectively used.

One reason for the obsolescence of these techniques is that most of today's solution implementations make use of packaged software and middleware; are at least partially generated from models and patterns; and/or are, to some extent, built from existing code. These practices have hopefully improved software quality and reliability, but they have not made estimation any easier. Instead, as the tasks of architecting, designing, and integrating software systems have become more complex, additional uncertainty has emerged in the estimating process, negatively affecting the accuracy of estimates.

**Changes in estimating assumptions**

The growing complexity of software development has changed basic estimating assumptions. In the old days, when one of a few simple programming languages were used to produce programs to run on a single platform, the task of estimating development cost and effort was relatively straightforward. One could safely assume, for example, that to develop something twice as large in terms of SLOC in a comparable time period, double the resources would be required. But the relationship between scope and effort is no longer linear. Nowadays, in order to implement a typical system 20% faster, roughly twice the effort may be required (Figure 2). For larger or specialized projects, the relationship between time and people can be even more indirect.

Additional estimation problems can arise when the pace of a project has to change in mid-stream, and, therefore, new resources must be added. In this scenario, due to the lag time required to assimilate new resources, the relationship between required resources and estimated delivery timeframe may even turn negative. In the worst case, it can follow Brooks' Law, which states that "adding manpower to a late software project makes it later."²

![Figure 2: Time/resources curves.](image-url)
The resource productivity gap

The characteristics of an effort curve such as those shown in Figure 2 depend on project-specific factors. One such factor that has a significant influence is varying individual resource productivity. By some estimates the productivity ratio, which represents the difference between the best performance and the worst, was 5:1 or lower during the early days of IT. In other words, the best staff could do as much as five times the useful work of the least productive staff. Nowadays, the productivity ratio reaches 40:1. Needless to say, with this magnitude of disparity between the productivity of the best and worst individual staff resources, accurate estimation of time and resource requirements becomes increasingly reliant on the ability to apply project resources of predictable quality.

Project results are no longer representative

As software projects increase in complexity, it appears that productivity data collected in one project often does not fully apply to similar projects, which calls the viability of learning-oriented and expertise-based estimation techniques into question. Differences in project management, development productivity, and technical particulars are among the chief reasons for this. Another aspect of the problem is that complex solutions require more sophisticated instruments; which, in turn, demand even more skill and expertise to use correctly. The offshoot is that the discipline of estimation is failing to keep pace with the growing complexity of projects, tools, and techniques.

Declining estimation accuracy

When an estimate is produced, it usually includes a confidence interval of some kind regarding its precision. This could be something like "the solution will be delivered at a cost of $1000 (+/-5%) provided that five resources are available for a period of three days." This disclaimer basically indicates that the estimator is reasonably sure that the solution won't cost more than $1050, provided appropriate resources are available.

The precision estimators are willing to commit to tends to decline as the cost of the project goes up. Estimates for larger projects may require a bigger confidence interval, such as "The solution may be delivered for the cost of $100,000 (+/-15%), given the availability of fifteen senior resources for a period of 30 days." Note that "senior resources" are specified to minimize the impact of poor resource quality on the project estimate.

On multi-million dollars projects estimates can get much messier. Not only it is painfully hard to account for all the variables that could possibly impact the project, but also the confidence interval itself can be little more than an educated guess. The recommended approach is to leverage data on industry averages and follow best practices guidelines. Most software estimating packages allow you to set these variables as configurable parameters.

Like the estimate figure itself, a value for estimate precision is often not consistent across estimates, particularly when using expert-based methods. If the estimate had to be produced again under slightly different circumstances, the numbers would probably be different. Even in algorithmic methods, which tend to produce more accurate numbers for larger projects, projected accuracy and precision can be greatly impacted by changes to only few inputs.
A single estimation method may not work
Selecting the best technique or combination of estimation methods to apply to a project has become a large problem for software teams. A single method can no longer produce reliable estimates across every step in the process. Often this is the case because the input data that a particular estimation method requires may be unavailable or inaccurate at the time when an estimate is needed. Thus, application of the preferred method may need to coincide with the application of other methods and techniques. Various methods are best used at specific points across enterprise and project lifecycles (see below for recommendations). Recently, hybrid and adaptable methods have emerged, which incorporate ideas from several methods and techniques.

Estimation is not part of the development methodology
Estimation is not explicitly part of most software development methodologies, such as the IBM® Rational® Unified Process (RUP®). The majority of software development methodologies simply assume that artifacts produced early in the lifecycle are consumed by later project planning and estimation activities. In RUP, for example, a use-case model acts as a measure of complexity and risk and is, typically, used as an input to project and iteration planning.

Whether or not a team uses RUP, The Open Group Architecture Framework (TOGAF), the IT Infrastructure Library (ITIL), or some combination of these, estimation techniques should be incorporated into the lifecycle. Similarly, estimation activities can make use of artifacts produced by software development methodologies, provided these inputs are explicitly documented.

Estimating across the software development lifecycle
Today's software development program and project estimation activities are predominately ad-hoc. When budget or schedule needs to be allocated an expert resource is assigned to perform the estimating task. Instead, estimation ought to constitute a continuous process that spans different approaches, techniques, and methods, and which can be used "à la carte" at different steps in the software development lifecycle to produce estimates in an incremental fashion, as illustrated in Figure 3.
In the sections that follow, I discuss the various points at which estimation might take place within an enterprise software development lifecycle, and what methods are most appropriate for use at various steps of the process depending on the methodology involved.

"Guesstimation" or how estimates are born

Depending on the situation, an initial estimate can originate either in the head of an enterprise planner (architect), or be produced by an assigned project specialist, such as the project architect or an experienced project manager. The estimating process typically starts with a simple question like, "What do you think it might take for us to ...?," asked by an interested executive or manager. A subject matter expert's answer to this kind of question might be something like "From my previous experience it would take anything from X to Y." This informal process, well-known in IT and beyond as "guesstimation" because it yields an "educated guess" based on the experience and analytical abilities of a single individual, is, essentially, a fast-track form of expertise-based estimation.

Although some amount of "guesstimation" is inevitable in most estimation activities, the main risk here is its reliance on the expertise, experience, and, often, the instincts of a single individual. While some people are better guesstimators than others, reliance on guesstimates is inadvisable in general as their accuracy can be very low, which can easily lead to false expectations. If you are a person who is asked to provide guesstimates, don't be afraid to say "Can I get back to you regarding this issue later?" and follow up by applying one of the standard estimation techniques.

Estimating in the Enterprise Architecture discipline

A growing role for the Enterprise Architecture (EA) discipline in the modern enterprise has been further secured with the recent availability of several EA frameworks. These frameworks include guidance for defining and implementing the enterprise architecture and a library of standard industry solution and process patterns that can be adapted to describe existing and future business and system components.

In the context of EA, "implementing" means the ability to create the necessary artifacts that define, describe, and support delivery of the next-generation enterprise business processes and systems. One such EA framework is the Open Group's TOGAF.

Estimation and TOGAF

The TOGAF Architecture Development Method (ADM) is a cyclical, phased process that helps enterprise teams build models of their existing and future enterprise architectures (see Figure 4). Each phase of the TOGAF lifecycle has a goal, and the final phase results in a complete enterprise architecture that implements the organization's vision. A full cycle of the enterprise architecture implementation process may span multiple years and govern multiple solution implementation projects.

An EA implementation using TOGAF ADM may leverage estimation several times in a cycle.
Using estimates to articulate an EA Vision

The first, high-level estimate is produced during Phase A of the TOGAF ADM implementation process. The Vision artifact created in this phase draws on a solid understanding of the current state of the Enterprise Architecture, as well as knowledge of its future direction. This phase may actively exploit expertise-based estimation, which it applies in a top-down manner to determine how much work falls into the current organizational change cycle, which can span anything from two to ten years or longer.

Estimation for business cases

It is typical for many organizations to produce a business case for their largest projects, also often called "programs" or "initiatives." A business case document, which is an end result of Phase B of the process, usually includes a list of assumptions and high-level statements regarding the program implementation. The difference between a business case and a TOGAF Vision artifact is that the business case may call for specific, albeit high-level, actions. For example, the business case may cite the need for improved turnaround across corporate business systems, request the addition of a new business service, or recommend improvements in business systems availability. Often, a business case includes statements like these without specifying the means, specific projects, or technologies that will deliver on these assertions; however, it does normally contain high-level cost and duration analysis.

It is common for "business case estimation" to use learning-oriented techniques, and obtain information from previous internal and external projects. Sometimes, business case estimates are based on quotations provided by consultants who have had previous experience with similar
activities, which are summarized in order to produce the ballpark figure. Business case estimation accuracy is usually very low, with the actual confidence interval often higher than 50%, as no detailed functional and project analysis is usually performed. It is largely impractical to use algorithmic methods for business case estimation, as the majority of project factors cannot be accounted for.

**Estimation and EA implementation planning**

Estimation is resorted to again in TOGAF Phase E when identified opportunities and their proposed solutions are outlined and discussed. Since the Project List artifact produced during this phase contains recommendations to commence specific implementations, more rigorous estimation than in phases A and B is required. Among the recommended solutions can be the creation or procurement of a new software component, the upgrading of an existing software application, improvements to the network infrastructure, or a mandate to perform more detailed analysis.

A combination of Function Count-based analysis and learning-oriented estimation could be a natural choice for use during this phase. Since there are no use cases or equivalent units of functional composition available yet, an algorithmic method such as the COnstructive COst MOdel (COCOMO) may use data from business process activities or business services as its inputs. An estimating application might also adapt the productivity rate from the previous completed EA implementation cycle or, in the case of the initial cycle, consult industry averages.

Algorithmic estimates produced in TOGAF Phase E can be further refined during Phase E, in which implementation projects are usually dedicated to performing business and system analysis of their scope. In Figure 5, RUP is used as an example of the solution implementation methodology. Here, project analysis may bring to light new details about the implementation sequence and project and component dependencies, which are often based on essential use cases, scenarios, features, tasks, or other units available as part of the solution lifecycle methodology. Project analysis may often be supported by expertise-based estimates produced by project leaders who are experts in their respective business and technical disciplines.

The use of expertise-based methods may last well into the Elaboration or even the Construction phase of RUP, at which point it can be dropped in favor of using algorithmic estimation, usually after the majority of requirements (generally more than 90%) are known.
Figure 5: A combined TOGAF ADM and RUP lifecycle.

Estimation and future EA planning

By the end of TOGAF Phase G, when all solutions are implemented, it becomes time to set targets for a new Enterprise Architecture implementation cycle. At this stage an EA team may use estimation for the last time in the current enterprise architecture implementation cycle. Learning-oriented and expertise-based techniques may be useful here to study the feasibility of proposed new activities, and to shape the parameters of the next round of EA round.

Estimating for software implementations

The saying goes that “An enterprise is a sum of its projects.” Small or large, every change in enterprise operations, products, or services requires a project in order to be delivered. While enterprises vary in terms of their products and services, their projects are fairly similar as regards the manner in which they analyze, modernize, or reorganize existing applications and processes, or create new ones.

A wide range of methods and techniques exist today that deal with project execution, from those with domain-neutral perspectives, like the Project Management Body of Knowledge (PMBOK) to those geared to certain activities, such as software development (RUP, Feature-Driven Development (FDD), Extreme Programming (XP)), or IT Service Management (ITIL, proprietary ITSM methodologies).
Estimates in the waterfall method

Notorious weaknesses in the waterfall development method have been widely criticized, yet the method still remains very popular. Management likes it because it pretends to provide more answers earlier in the project (notwithstanding whether they are correct or not), while developers follow it because they are often pressured to provide more all-encompassing answers up-front than they realistically can. The waterfall method also seems to be intuitive to get started with, as it does not require any methodological training.

Fundamentally, estimation in a waterfall environment is treated as if the entire solution will be delivered in a single shot. In a properly set-up waterfall project there is a lengthy period in the beginning dedicated to a comprehensive feasibility study and requirements analysis. Many project managers sincerely believe that by performing a thorough analysis of the solution it is possible to obtain robust solution designs. In the beginning of a project this assumption often proves right, and the project runs as expected, which can be attributed to the thorough requirements analysis that was performed at its inception. In the long run, however, this strategy almost always proves ineffective, as unforeseen and often cascading complexity (discussed above) creeps in, impacts the scope, and renders an estimate produced at the beginning of the project inaccurate or invalid (see Figure 6).

Waterfall methods may work on smaller projects and for well-coordinated teams. The method is expected to produce a single estimate, usually after feasibility analysis and requirements gathering activities. A rigorous initial analysis and planning phase is capable of producing enough parametric data to feed the algorithmic methods, such as COCOMO and the Software Lifecycle Model (SLIM).

Estimates in iterative software development

In today's environment of growing hardware and software complexity and constantly changing requirements, iterative and cyclical methodologies prevail in terms of their success rate. Iterative methodologies significantly improve overall efficiency and time/cost savings by sacrificing some initial visibility into requirements (see Figure 6).
Here are some of the advantages to a project of using an iterative methodology over the waterfall method:

- Since the riskiest use cases and issues are found and addressed earlier, project risk is reduced.
- It is possible to mitigate the impact of changing requirements by describing and implementing the riskiest and those most dependent upon use cases before others. Since use cases are written incrementally, those that were affected as a result of new findings do not have to be rewritten, which results in time and cost savings.
- Implementation of some requirements before all others are described in full detail allows for gradual and more effective distribution of labor, resulting in improved efficiency and reduced costs.

The key differences in estimation activities on an iterative software development project versus one governed by a waterfall approach are:

- A series of iteration estimates is produced in addition to a single project estimate.
- Unlike on waterfall projects, an overall project estimate is adjusted after each iteration to improve its accuracy.
• Project estimates can be produced very early in the iterative development lifecycle, which offers improved support for project planning.

Estimating on RUP projects

RUP is the leading iterative software development methodology. RUP comes in multiple "form factors" (RUP for Systems Engineering, for instance). However, they all share the same fundamental principles, such as an iterative approach and early risk mitigation. The iterative approach implemented in RUP is not specific to software development and may be applied in other types of projects and engagements, such as commercial off-the-shelf (COTS) software implementations, systems analysis, and infrastructure management.

A typical RUP project consists of multiple iterations, which are established within the projected timeframe in order to implement groups of pre-selected use cases that are identified at project inception. RUP contains guidance for how to elicit, select, and group use cases, and how to allocate them among iterations. With the exception of use cases that are implemented in the current iteration, other use cases and groups are fully dynamic; that is, the lists and groups of use cases produced at the beginning of a project may be modified as result of ongoing analysis in subsequent iterations.

A part of the RUP compile_software_development_plan task is to build an implementation plan for the whole project. The same task also generates as an output an iteration plan for the subsequent iteration. During execution of this task, project decisions are made based on available empirical data, such as the number of essential use cases and their complexity, the estimated total number of use cases, or the projected number of SLOCs. This same data is used to establish how many project iterations are required given the actual resources available, and the use cases to be implemented in an iteration. As you can see in Figure 6, estimation in iterative lifecycle methodologies is both iterative (for project planning) and incremental (for iteration planning), and it takes place multiple times during the project.

The importance of accurate estimation to a RUP project cannot be overstated. Unlike the waterfall approach, RUP relies more heavily on expertise-based estimation, especially during the Inception phase. When a draft project plan is created, it is based on essential use cases as the only structural units of measure available at the time. A project manager may use essential use cases to determine how many iterations will be required. Expertise-based estimation can come in handy at that point, as very little detailed data is available about the system being built, which limits the opportunity for using algorithmic estimation methods. At this early stage, it may be useful to apply learning-oriented estimation to assess the resources required for the first few iterations. By the end of the first Elaboration iteration, algorithmic methods become more applicable as more structured requirements are developed to feed them.

In the second and subsequent Elaboration iterations, even broader and deeper requirements become available. Then, a selected algorithmic method can be configured using actual achieved performance data, and reapplied to improve the project estimate and the project plan. Methods like COCOMO and SLIM begin to shine, while expertise-based techniques gradually become less useful.
Although RUP itself contains limited advice about estimation, as a broad platform of best practices IBM Rational Method Composer contains guidance for selecting estimation techniques and weaving estimation activities in with RUP lifecycle activities.\textsuperscript{5}

**Estimating on Agile projects**

The attitude toward estimation in Agile methodologies and among Agile practitioners is about the same as toward any other regulatory work, which might be characterized as "ignore as much of it as possible, as much as possible." The difference, however, is that even the most progressive executives and most Agile-influenced project teams must commit to specific delivery dates and request specific budgets and resources for their projects, which naturally brings estimation into the project.

A perennial question asked by all Agile methods is "How much estimation is too much?" The consensus opinion is that estimation has to take the form of an inventory measure, where units are inexpensive to produce, and are a logical part of the process flow. The units can be features for Feature-Driven Development (FDD), stories for Extreme Programming (XP), or use cases for light-weight RUP and Unified Process (UP) methods.

It is clear that the Agile community gives preference to expertise-based estimation techniques over algorithmic methods; however, it does not fully exclude using FP-based analysis. The bottom line for estimation on Agile projects is to use whatever techniques project teams have the most expertise with to estimate their chunk of work, rather than to dedicate a single expert to producing an estimate for everyone or for the entire project. Agile estimation should measure shorter cycles rather broad deliverables, and should be applied in multiple increments, which can correspond to sprints, iterations, and other methodology-specific intervals.

Putting together a reasonably good, enterprise-wide estimate in an Agile manner can be more difficult than producing an estimate for a project; however, all the same rules and principles just discussed still apply. Large elements like projects, subsystems, and business processes can be used as units of measure instead of smaller units, such as use cases, scenarios, or features. Expertise-based and learning-oriented techniques may also prove very handy during enterprise Agile estimation, as can the top-down approach.

**Agile COCOMO**

Agile COCOMO is a method and accompanying tool that are geared for lightweight incremental cost and schedule estimation. Created by Cyrus Fakharzadeh and Gunjan Sharmanby from the USC Center for Software Engineering\textsuperscript{6}, the method is based on COCOMO II 2000 and incorporates elements of learning-oriented estimation.

The traditional COMOMO II method requires an estimator to manipulate all twenty-two parameters. In Agile COCOMO an estimator can choose and set an anchoring parameter, such as the Total Cost in Dollars or Total Effort in Person/Months and then incrementally add cost or schedule drivers and scale factors that differentiate the reference solution from the actual system being built. The method allows you to apply factors to an estimate as they emerge on the course of implementation, which makes it suitable for projects that employ Agile and iterative methodologies.
Estimation and other solution methodologies and best practices

Enterprise "solutioning" does not stop with software development and includes, among other areas, Application Support, Business Intelligence (BI), Data Warehousing (DW), Network Infrastructure Management, Database Management and Administration, and Security Administration. Although each one of these areas may be governed by its own methodologies and seemingly require a unique approach to its cost and effort estimation, most of the estimation techniques and methods discussed above, such as expertise-based and learning-oriented techniques, are universal and can be applied in any of these areas. Other estimation techniques, such as FP-based methods, may only require some tailoring in order to be used.

What differentiate enterprise solution areas from an estimation point of view is their Units of Measure (UM). While use cases, use case steps, actors, and alternate scenarios are traditionally used as UMs during software development, other valid UMs can include data sources, schemas, cubes, reports for BI and DW; databases, database structure and connectivity objects for Database Maintenance and Administration; servers, server clusters, workstations, and network segments for Network Management; or domains, profiles, groups, pages, and certificates for Security Administration (see Figure 7).

Figure 7: Application of Function Point techniques in different solution domains.

Improving consistency and quality of estimates

This section contains advice on how to improve estimating efficiency and accuracy. It explores technical and methodological issues as well as cultural and social aspects to provide some essential recommendations.
Method selection

Despite the existence of guidelines for method selection it has been proven on many projects that there is no single correct answer to the question of what method to use. The choice often depends on many factors, some of which found their way into algorithmic methods as configurable parameters. The limitation, however, is that in most cases there is not enough information to specify those parameters, or the available information is not representative, which results in poorer estimates than those produced by expert-based techniques.

Expertise-based and learning-oriented techniques are very good at predicting code or programming effort. Their weakness is in predicting requirements growth, design effort, documentation and management, testing, repair, and rework. Human experts are consistently too optimistic at predicting support work. This happens because humans fail to fully recognize the extent of project inertia and administrative and support overhead activities that are part of complex implementations. Algorithmic techniques are perfect for estimating on large projects as they are capable of accounting for measured, organization-specific, and industry averages; and of automatically adding calculated overhead costs into the produced estimates.

All in all, large software implementation projects tend to devote more effort to producing paper documents and fixing bugs than they spend on producing software deliverables. This surplus project effort is harder to account for in expertise-based and learning-oriented techniques as there is no perceived entity or measure it can be attached to, so it slips through the fingers. Most algorithmic methods excel in this respect as they force a user through a complete process of configuring them.

While it is true that, for all methods and techniques, estimate accuracy is a product of complexity, completeness, and quality of requirements -- as well as project factors -- it is also worth noting that given identical initial conditions, different techniques produce estimates of varying accuracy. In expertise-based methods accuracy is a product of the estimator's expertise and experience, which cannot be easily measured and validated. In learning-oriented methods, accuracy depends on a degree of variance between the reference and the desired solutions and projects, the amount of knowledge about the reference solution that is available, and the expertise of the estimator. The accuracy of algorithmic methods may vary; however, it can also be improved by calibrating the method to the target environment.

At different stages in the development process some techniques produce better results than others. Close to the beginning of a project, expertise-based techniques usually beat formula-based models. This quickly changes, however, as algorithmic models are able to be configured and tailored to the project environment.

Promoting a positive attitude towards estimation

Reliable software cost estimation is not only about using the right tools and techniques. It is also about promoting a realistic view of what can be done and when, and what cannot be done. Many software engineers start off by fooling themselves and their supervisors about deliverables and their possible impacts. This most often happens due to their inexperience dealing with technical and project complexities.
Anyone involved in enterprise project planning has to keep in mind that accurate estimation is first and foremost about the right attitude. It concerns not only software developers providing realistic numbers, but also senior management understanding the challenges that surround estimation of software solutions, among which is the difficulty in producing accurate estimates in the beginning of the implementation. While developers and estimators alike have to be conservative, thorough, and avoid over-promising, management in turn has to accept the incremental nature of the software development and estimation processes and support their staff on this path.

Adjusting for organizational culture and politics

Do not fool yourself by thinking that you can change your organizational culture. Instead, study it and try to embrace it or, at very least, don't fight it too hard. Estimation tasks have to adapt to the realities of the culture in terms of how much you can realistically promise and when. Adapting to the organization culture is not about bending under something you do not believe in. It is about understanding the values and rhythms of the real people that are part of the business, in order to make intelligent adjustments to estimates that you produce, for the benefit of all.

Part of estimating is to make sure that groups involved in a project commit to delivering on your promises. Internal politics and bureaucracy may play a role in whether or not this happens, as can the cultural and social interests of groups and individuals.

For obvious reasons, these factors are generally not included as parameters in estimation methods and tools. Political and cultural issues are hard to quantify, and they may be very sensitive to discuss. Thus, their inclusion in estimation methods could stir up more problems than it would solve. Some organizational factors can be accounted for on an ad hoc basis -- as they should be, since their impact can exceed the contribution of more readily quantified factors like technological risks. In some organizations the weight of cultural factors is so huge, the organization can barely move forward. Typically, large corporations and government bodies are more affected than smaller private shops and medium-size companies. A rule of thumb is to secure a commitment from diverse groups as well as to adjust for the culture in the estimates you produce.

Defending your estimates

Nowadays, estimating is sometimes akin to horse trading, when you negotiate rather than prove your estimate’s validity. Top managers, project sponsors, and clients tend to exert various pressures on middle managers and estimating personnel, usually in the direction of optimistic estimates.

It is crucial to be able to produce estimates that can be defended in a face of arguments. One strategy is to present a collection of historical data from similar projects along with an estimate. A better strategy, however, is to instill trust in management and peer groups. Once you have achieved a state of mutual understanding and trust, your estimates will be respected and defended by your colleagues and bosses alike as their own.

Effective presentation of an estimate to the stakeholders is also very important. Don't just present raw numbers. Stakeholders will want to see a description of the method you used, at least a partial
sample of the data the estimate is based on, and, hopefully, a reference or two to successful
past implementations that resulted in comparable actual cost and delivery figures -- which your
stakeholders can, in turn, leverage for justification when presenting your estimate to their bosses.
It is also worthwhile to create a graph showing the high-level breakdown of costs and time, as well
as a link or two to external papers and documents that support the project assumptions you made
and the techniques you used.

**Basing estimates on structured requirements**

An estimate is only as good is its weakest link, which usually resides in the requirements it is
based on. Changing, incomplete, incorrect, or poorly presented requirements are perhaps the
single greatest reason for faulty estimates, which translate into project failures and frustration.

Some of the best source material for both algorithmic methods and expertise-based techniques is
the structured requirements. The reason they are called "structured" is because their structure is
dictated by an agreed schema. Using a pre-defined schema for requirements ensures that their
content is consistent, even if they are produced by different individuals. From an estimation point
of view, the beauty of a structured requirement is that it is measurable and scalable. For example,
a use case is a structured requirement that consist of steps, pre- and post- conditions, and other
parameters and outputs. Such a requirement can be accurately estimated upon based on number
of steps, alternate scenarios, and references to other requirements it contains. Likewise, its growth
can be predicted, and the impact of changes to it on other requirements and the project can be
calculated.

**Combining techniques to yield reliable results**

Although hybrid estimation methods that include elements and best practices from different
methods, approaches, and techniques are already available, an estimator should be aware that
applying multiple techniques to test estimates is often still the best way to go. Applying multiple
methods may yield some unexpected results, which are not always relevant or right; however, the
exercise is usually highly worthwhile. It is akin to eliciting a second opinion about the problem,
which is almost always the right thing to do.

**Configuring the estimating model**

Expertise-based and learning-oriented techniques can act as a useful tool for configuring
algorithmic methods. When an original estimate figure is produced using a formula, it can be
compared against similar projects that took place in the past. Based on the results of such a
comparison, considerations can be established and parameter values that were used to produce
an estimate can be questioned and adjusted as required to align the estimate with the reference
figure. This side-by-side comparison can also help you to find new, often unforeseen, factors and
parameters.

**Keeping estimates current**

An estimate should be kept as current as possible after it is created. RUP and other iterative
methodologies recommend that artifacts are kept up-to-date throughout the project. Waterfall
methods and some others procedures that emphasize advanced planning are not as stringent
about this rule. While, in general, the estimating procedure should be in harmony with a specific methodology's principles, not keeping estimates up-to-date throughout the project may result in a ballooning "expectations gap" between the implementation team and their management, leading to misunderstandings and dissatisfaction on both ends, and even partial or complete project failure.

If the project for some reason is unable to keep estimates current, then the estimation deliverables should include a clause indicating the conditions on which they remain valid. When put in place, these conditions should help to "insure" the estimate and link it with requirements and the environment it was created for. A conditional clause can be essential for preserving the integrity of an estimate and to help defend it.

**Encouraging development best practices**

To excel at estimation it is not enough to master methodologies and tools. The dependability of estimates also depends on the ability of an organization to consistently deliver results; i.e., its maturity. It is quite true that teams that follow best practices and master good support tools are, usually, more predictable and consistent at delivering on their promises. In order to produce an estimate, most methods will ask the estimator questions about what methodologies and design and development practices are standard in his or her environments. Such questions may range from "What software development lifecycle (SDLC) experience does the implementation team have?" to "What design best practices and tools are standard in the organization?"

While applying the right estimation methods and tools may help to improve the quality of estimates, you should also encourage the application of best practices and tools in the development organization, as their adoption directly affects the dependability of estimates.

**Conclusion**

Apply rigorous techniques to create sound estimates -- do not rely on instincts. Discourage expertise-based estimation as non-repeatable in comparison with algorithmic methods, but only when you have enough requirements to feed these methods, and where this does not add undesired friction to the process. Look at hybrid methods and do not hesitate to mix different techniques. Adjust estimates for the realities of your organizational culture, and be prepared to defend you estimates if necessary. Cultivate a positive attitude towards formal estimation practices and focus on building trust within your team. Encourage the use of software development best practices and tools, and strive for increased repeatability of the delivery processes.

**Further reading**

- A decent estimation methods comparison paper by Liming Wu is available at [http://www.computing.dcu.ie/~renaat/ca421/LWu1.html](http://www.computing.dcu.ie/~renaat/ca421/LWu1.html)
- This article by Amit Bhagwat from *The Rational Edge* provides a perspective on iterative, use-case driven estimation of "fixed-cost" projects: [http://www.ibm.com/developerworks/rational/library/content/RationalEdge/oct03/f_estimate_b.pdf](http://www.ibm.com/developerworks/rational/library/content/RationalEdge/oct03/f_estimate_b.pdf)
• Another interesting perspective on predicting iterative projects, by Bill Walton, is available at
  http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&Objectld=6781
• This research article by Chris F. Kemerer (http://portal.acm.org/citation.cfm?id=22906&dl=)
  contains empirical validation of software cost estimation models.

Notes

1 For more information see http://www.cutter.com/content/alignment/fulltext/advisor/2006/
   bit061220.html (this link no longer works). Requires a paid subscription.

2 From Frederik Brooks' classic 1975 book The Mythical Man-Month. For more information see
   http://en.wikipedia.org/wiki/Brooks'_law

3 For a good summary of incremental estimating practices see http://public.dhe.ibm.com/software/
   dw/racionaleedge/oct03/f_estimate_b.pdf

4 For more information see my recent Rational Edge article at http://www.ibm.com/developerworks/
   rational/library/jan07/temnenco/index.html

5 Consult the Rational Method Composer content database (http://www.ibm.com/developerworks/
   views/rational/library.jsp) for advice.

6 See http://sunset.usc.edu/

© Copyright IBM Corporation 2007
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)