from The Rational Edge: This article, Part 1 in a two-part series, presents an overview of approaches, techniques, models, and tools used in estimating costs, schedules, and other factors on software projects. Emerging innovations are discussed in addition to well-established methods. Part 2 provides advice on what methods to apply in specific circumstances.

Estimation has always been one of the riskiest aspects of project or program planning. This is not because estimators are regularly unqualified or poorly informed -- it is primarily because of the large and growing number of complexities and dependencies that must be factored into software project estimates. Inevitably, as software projects, software products, and IT environments all become more and more complex, so, too, does the task of estimating what they will cost and how long they will take. To compound the challenge, established parameters that form the basis for many estimation techniques are not as universally applicable, or as straightforward to calculate, as they once were. Many estimators are thus left searching for methods that can yield more accurate results.

In this, Part 1 of a two-part article, I'll examine approaches, techniques, models, and tools that have gained momentum and support among estimation experts, as well as some intriguing innovations that seem to have bright futures. In Part 2 I will discuss different estimation scenarios, what methods are best under what circumstances, and how to most efficaciously apply them.

Why we estimate

We human beings tend to estimate constantly in our everyday lives. We estimate how long something will take, how much something will cost, how many calories are in that dessert, and
so on and on. Estimation is just as vitally important to an organization, as its economic viability depends in great part on the quality of the decisions made by its executives. Which, in turn, are driven to a large if not a primary extent by estimates. Business decision-makers estimate for reasons like:

- **Budgeting.** Budgeting is not only about how to spend cash sitting in a bank account, but also about how to strategically allocate available and future resources for the greatest benefit to the organization. Some of the cash you’re planning to spend may not yet be readily available; and, indeed, may never materialize -- factors an estimator will try to predict.

- **Project planning.** For software projects, estimation is part of predicting costs, schedules, and resources, and striking a balance among them that best meets enterprise objectives and goals. Estimates are also very important for internal project planning and execution, as they influence project lifecycles (iterations, increments, etc.), and thus ultimately project results and even individual productivity.

- **Risk management and trade-off analysis.** Estimation is a major factor in enterprise risk analysis and risk management because every enterprise decision makes assumptions about the flow of events, which is largely based on estimates.

- **IT infrastructure and process improvement analysis.** Estimation is an essential part of the enterprise architecture implementation and governance, which includes (among other things) assessing enterprise process improvement alternatives and their impacts on other processes, as well as considering options for building versus buying software, hardware, and services.

### Estimating approaches, techniques, models, and tools

In the early days of IT, people invented straightforward ways of estimating software development work. Back then, software estimating was by-and-large a matter of applying a linear equation with variables for lines of code and staff headcount. As the nature and role of IT has become vastly more complex and diverse, so have the estimating techniques applied on software projects. Something that worked in an environment with a single programming language may no longer work in today’s heterogeneous environments. Other aspects that have contributed to the evolution of the estimation discipline and prompted introduction of non-linear factors include the emergence of multiple platforms and frameworks, commoditization of IT services, and increased complexity and size of IT projects.

To cope with this growing estimating challenge, innovators came up with numerous approaches and techniques, some based on mathematical models and others drawing upon human factors. Figure 1 presents an attempt to categorize some of the many available estimation resources.
Figure 1: Illustration of estimation resources

Top-down versus bottom-up estimation

Perhaps the most fundamental aspect of estimation, not only in software development but for almost any project, is the choice regarding one of the two principle estimation strategies to adopt: bottom-up and top-down (see Figure 2). The bottom-up approach calculation is based on the sum of the various, individual project activity estimates, while the top-down approach bases the overall estimate on properties of the project as a whole, which are then applied across estimates encompassing all project activities.

Figure 2: Illustration of bottom-up and top-down estimation

Bottom-up approach

In the early days of IT programming, most software estimates were produced in a bottom-up manner. Lines of manually written source code, which acted as a key variable in estimates, were counted to calculate the total size of the project. The other key variable, which was equally easy to quantify, was developer productivity. The situation started to change with the introduction of diverse programming languages (C, Pascal, Ada, Lisp, etc.), programming paradigms (object-oriented, template-based, etc.), hardware architectures and platforms, and code generation capability.
The bottom-up approach is generally considered to be more intuitive and less error-prone than the top-down approach (discussed below). In a case of well-isolated components it can produce very precise estimates. However, by using only a bottom-up approach, it can be easy to overlook significant, system-level constraints and costs, especially when this approach is used in the absence of sufficient requirements data.

**Top-down approach**

The top-down approach, also known as the Macro Model, is based on partitioning project scope and then using averages (either specific to the organization or industry averages) to estimate cost and complexity associated with implementing the parts and their interconnections.

The distinctive feature of the top-down approach is its focus on overall system properties, such as integration, change management, and incremental delivery, combined with its relative ease of application. However, as mentioned above, by nature the top-down approach is not as good as the bottom-up approach at capturing lower-level factors, such as project-specific issues, application design features, and implementation-specific details, which tend to accumulate rapidly and can affect the estimate greatly.

**Popular estimating techniques and models**

Although they are generally approach-agnostic, each of the techniques discussed below is normally used in combination with either a top-down or bottom-up approach.

All estimating techniques can be grouped into one of three categories: experience-based, learning-oriented, and those based on algorithmic (statistical or mathematical) models.

**Experience-based techniques**

Experience- or expertise-based techniques rely on the subjective judgment of a subject matter expert or group of experts (see Figure 3).

Unlike more stringent, parametric models, these techniques rely primarily on the experience and insight of human beings, which constitutes their main weakness. This weakness can be mitigated by expanding the size of the group of estimators.
The Delphi technique

In the Delphi technique, a group of subject matter experts is asked to make an assessment of an issue. During the initial estimation round, each expert provides his or her individual assessment without consulting others. After the results of the initial round are collected and sorted, the participants engage in a second review round, this time with information about the assessments made by other participants. This round is supposed to further narrow the spread of numbers, as some concerns are taken into more detailed consideration, while other are dismissed. After a few rounds, the team arrives at a mutually agreed-upon estimate.

Work Breakdown Structure

A more structured form of expert-based estimation is the Work Breakdown Structure (WBS) technique. In order to produce an estimate, this technique models the structure of the solution and/or the breakdown of the process to be used to implement the solution. These structures can be devised based on the high-level solution architecture and the project plan, respectively, by decomposing them into granular units that can be described and measured (see Figure 4). Weights are then assigned to each identified unit, which can be summarized to calculate the total solution cost.

Figure 3: Illustration of expertise-based estimation

Figure 4: Illustration of work breakdown structures
Click to enlarge
Learning-oriented techniques

Various studies indicate that more than three-quarters of software estimates are built using some form of analogy or comparison with previously completed solutions -- that is, they utilize the technique known as learning-oriented estimation. Although intuitively very similar to expertise-based techniques, learning-oriented techniques take a different angle. Their objective is to find a similar system produced elsewhere and, through knowing how the properties of the new system vary from the existing one, extrapolate the estimate (see Figure 5).

The most evident advantage of learning-oriented techniques is that the estimate is based on proven characteristics and not just empirical evidence, as in the case of expertise-based estimation. A limitation of this technique is the need to determine the most important variables to be used for describing the solution, including those that distinguish it from the baseline. These objectives may be difficult, or very time-consuming, to achieve.

Figure 5: Illustration of learning-oriented estimation

Case studies

In case studies techniques, estimators apply lessons and heuristics learned from examining specific examples that resemble the system that is being built. The process consists of several basic steps that should result in an estimate:

- In Step 1, the principle characteristics of the solution are agreed upon. These can include essential solution features and key actions to be taken to deliver it.
- In Step 2, a reference solution is selected from the organization's knowledge database (or elsewhere), whose core characteristics match or come close to those agreed to in Step 1.
- In Step 3, unique characteristics of the new solution are determined by comparing the planned and existing solution architectures.
- In Step 4, all necessary adjustments are made to the estimating model to account for the unique properties of the solution.
- In Step 5, an estimate is produced using the adjusted model developed in Step 4.
Algorithmic methods

Algorithmic methods use mathematical (i.e., formula-driven) models. They take historical, calculated, or statistical metrics -- such as lines of code and the number of functions -- as well as known environmental factors -- such as the programming platform, framework, hardware platform, and design methodology -- as their inputs to produce an estimate with a known degree or range of accuracy. For their ability to accept various metrics these methods are also sometimes called metric-based (see Figure 6).

Unlike non-algorithmic techniques, algorithmic methods may produce repeatable estimates, and they are comparatively easy to fine-tune. These methods, however, are also more sensitive to inaccurate input data and may produce very poor estimates if not calibrated and validated properly. Also, these methods cannot efficiently deal with exceptional conditions, such as highly creative or lazy staff or exceptionally strong or poor teamwork.

Figure 6: Illustration of algorithmic estimation

Cost is, arguably, the most important factor in any enterprise decision. Therefore, many algorithmic methods are geared for cost estimation. Cost itself may be a product of other factors, including the project duration, resources, and environment.

COCOMO and COCOMO 2.0

The COnstructive COst MOdel (COCOMO) uses simple a formula that links Required Man-Months Of Effort (MMOE) to Thousands Of Delivered Source Code Instruments (TODSI). The formula is:

\[
\text{MMOE} = K_1 \times (\text{TODSI})^{K_2}
\]

where K1 and K2 depend on environment and design constraints and factors, such as size, reliability, and performance, as well as project constraints, such as team expertise and experience with technologies and methodologies. Creators of the model envisaged that it would be used multiple times throughout the project, each time producing more accurate estimates.

With the emergence of numerous process models, techniques, approaches, and technologies, the original COCOMO model, which was based solely on Software Lines of Code (SLOC), rapidly lost favor with estimators. This led to the creation of COCOMO 2.0.

COCOMO 2.0 eliminated a principle drawback of the original COCOMO -- its dependence on SLOC as the only valid unit of measure. It included an adjustable and dynamic collection of
sizing models. This second release encompassed object and method points, function points, and lines of code; and the model could be calibrated based on the type of an intended design (i.e., object-oriented, component-based, COTS-based, prototype, etc.), development methodology to be used (RUP, XP, SCRUM, waterfall, etc.), and project type (maintenance, "green field" development, re-engineering, etc.). COCOMO 2.0 also included an expanded set of size and effort characteristics among potential calibration factors. These additions can help to increase the accuracy of estimates; however, a higher level of expertise is required for their application.

The Software Lifecycle Model

Another popular cost model is Putnam's Software Lifecycle Model (SLIM).

It relies on SLOC and the Raleigh model of project personnel distribution (Figure 7). The basic Raleigh equation links Total Person Months (TPM) and size in SLOC with the Required Development Time (RDT) coefficient through the Technical Constant (TC):

\[ TC = SLOC \times TPM^{1/3} \times RDT^{4/3} \]

TC here is a centerpiece of the method, as it helps to track environment factors. After being calculated, TC may range from 200 for "poor" to 12000 for "excellent."

From the previous expression can be derived:

\[ TPM = \frac{1}{RDT^4 \times (SLOC/TC)^3} \]

After being calculated using the above formula, the estimate may be fine-tuned using the Raleigh curve (see Figure 7), which an estimator can adjust by manipulating two parameters: the slope, which represents initial manpower buildup (MBI); and a productivity factor PF. The method provides two ways of setting values for these parameters, either by applying the average calculated during past projects (iterations, cycles), or by answering a series of basic questions.

This model suffers from the same weakness as the original COCOMO: it requires an ability to extrapolate SLOC for the software to be implemented. Since this is not an easy thing to do, especially at the beginning of the project, and because in almost every modern system, a large part of the executable code is either generated or reused through standalone components or middleware, it may be difficult to achieve acceptable accuracy using this method.
Function point analysis

In order to calculate Man Months of Effort and Total Person Months, respectively, COCOMO and SLIM require an estimator to calculate SLOC, which can be an impossible task in a modern, heterogeneous IT environment. Besides, isn't calculating the number for SLOC inherently part of the same process as calculating MMOE or TDM? This is why bottom-up and top-down approaches, and the expertise, learning, and model-based techniques may -- and should -- be used together, as discussed below.

Function point analysis is a vital alternative that allows basing estimates on the more technology-neutral concept of function points. In practical terms, function points can be represented with anything from use cases and business processes to services and Web pages.

In function point-based methods, counting function points involves enumerating external inputs, external outputs, external inquiries, internal files, and external interfaces, and assigning each of them one of three complexity factors: simple, moderate, or complex. The weighted count of functions, also called a function count (FC), is then multiplied by a complexity factor previously assigned to functions to arrive at the total number of function points (see Figure 8). The calculated total is then adjusted for environment complexity, which is determined by weighting fourteen different aspects of the environment. As a result, the final FC may be affected by a factor of .35.

Figure 7: Illustration of the Raleigh model of project personnel distribution

Figure 8: Illustration of function point analysis-based estimation
The main advantage of using FC instead of SLOC is that it can be derived from functional requirement or analysis and design specifications. Additionally, an FC method allows abstracting from specific language, methodology, or technology, and is easier to understand and interpret to non-technical and external stakeholders and users.

Some of the most popular function point estimation methods include ESTIMACS and SPQR/20.

SPOR/20 expands the basic technique by grouping function points into classes based on the complexity of Algorithms, Code, and Data, with each class having a distinct weight in the estimating process. After calculating the total cost based on function points, the method takes an estimator through a series of multiple-choice questions that may range from "What is the objective of the project: a) Prototype, b) Reusable Module" to "What is the desired response time", in order to calculate the risk factor to be used to adjust the total cost figure.

ESTIMACS, in turn, focuses on key "business factors." It introduced the concept of so-called "estimating dimensions," which are "things" that are most important to the business. These "things" include Hours of Effort, Staff Count and Cost, Hardware, Risks, and Portfolio Impact. ESTIMACS also includes the classification of "project factors," such as "Customer Complexity" and "Business Function Size" for the "Effort Hours" categories. This allows you to link "estimating dimensions" with the major functional parts of the system expressed using function points. Besides providing this classification ability, ESTIMACS supports an incremental estimation approach that breaks the estimation process into three stages, enclosed in a closed loop (see Figure 9).

![Figure 9: Three stages of ESTIMACS approach](image-url)
Emerging techniques and models

Although the COCOMO, SLIM, and FC techniques are among the most frequently used estimation methods, it is important to mention some emerging and promising newer models and techniques.

Neural Networks

As the complexity of IT architectures grows, so does the sophistication of estimation techniques and methods. One of the latest ideas adapted for estimating is Neural Networks. The technique is capable of "wrapping" a function point or other equation-based algorithm. It is especially promising as it allows you to dynamically adjust the underlying algorithm based on historical data (see Figure 10).

![Illustration of neural networks estimation](Click to enlarge)

It is interesting to note that Neural Networks is an evolution of learning-oriented estimation, in which the method algorithm is trained to behave like a human expert.

Today's trend is that estimating tools are moving closer to process tools (an example being the inclusion of an estimation component in the IBM® Rational® Method Composer process framework). Similarly, process tools now integrate increasingly tightly with design and portfolio management suites. For estimators these trends mean a more consistent and timely supply of historical data, and they point toward a bright future for Neural Networks.

Dynamics-based techniques

Dynamics-based techniques emphasize that project conditions constantly fluctuate over the duration of implementation, affecting software development productivity, which is a central factor in most estimating methods. Therefore, estimates should not be based on a limited number of snapshots of the situation, but rather on the continuous, dynamic intake of project data.

One way to model dynamic behavior like this is to use a graph to represent the estimating model's structure. In a graph-based model, nodes act as variable project factors, while edges represent information flows that may affect those factors. Using such a graph, it is possible to tell how change in one project factor affects other project factors and, eventually, the Development Rate (DR). (See Figure 11).
Regression-based techniques

Regression-based techniques are most helpful when the amount of data available for analysis is high. These techniques, including "Standard Regression" and "Robust Regression," employ statistical analysis of significant volumes of data to adjust their models. As the category name indicates, these methods use changing experimental (project) data to calibrate their base models, which could be either COCOMO, COMOMO 2, or SLIM (see Figure 12).
Hybrid techniques

To make of the best of all estimating techniques and models, a preferred solution is to combine
algorithmic, statistical, expertise-based, and learning-oriented estimation in a single approach.
Such an approach would be adaptable to solution and project characteristics, and yet provide the
flexibility to select a method that is best suited for a particular situation.

A good example of a hybrid method is Bayesian Analysis technique, which allows an estimator to
use both project data and expert analysis in a consistent and predictable fashion. At the heart of
the technique lie statistical and expertise-based procedures that endeavor to predict parameters
of the project, based on observed data and prior experience. At first, one of the standard methods
or techniques is used to produce the baseline estimate. When the baseline number is known, an
estimator calculates the probability of the ultimate, actual number matching the baseline number.
Calculated probability is then applied to the base number to arrive at the final estimate (see Figure
13).

![Figure 13: Illustration of probability-based estimation](Click to enlarge)

While still in their infancy, hybrid estimation methods are expected to actively develop in the near
future.

Estimating tools

A large and growing number of estimation techniques and methods has made possible the
emergence of a wide spectrum of estimating tools. Initially, estimating tools focused on a single
method or technique, but no longer. Today’s rugged IT landscapes mean that a single technique
can no longer be used universally with any hope of success.

Tools that are currently used by many organizations world-wide include, among others, COCOMO
II, CoStar, CostXpert, ESTIMATE Professional, KnowledgePlan, PRICE-S, SEER, SLIM, and
SoftCost. Other tools that were popular in the past and may be still in use are COCOMO,
CheckPoint, ESTIMACS, REVIC, and SPQR/20. Table 1 provides an overview of the scope and
focus of some popular estimating tools.

Table 1: Scope and focus of popular estimating tools

<table>
<thead>
<tr>
<th>N</th>
<th>Tool</th>
<th>Features</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>USC COCOMO II</td>
<td>Cost, effort, and schedule estimation using FC and SLOC</td>
<td>USC Centre for Software Engineering</td>
</tr>
</tbody>
</table>
2. CoStar  
   Estimation using COCOMO 2.0 through all project phases  
   Softstar Systems

3. CostXpert  
   COCOMO-based estimation adaptable to different lifecycle processes and software size models  
   Cost Expert Group

4. ESTIMATE professional  
   Organization-tailored estimation with COCOMO 2, Putnam, and other techniques  
   Software Productivity Center

5. KnowledgePlan  
   FC and learning-oriented project scheduling with feedback loop  
   Software Productivity Research

6. PRICE-S  
   Suite of industry-specific cost models that implements a top-down approach to enterprise planning  
   Price Systems

7. SEER-SEM  
   Analysis and measurement of resources, staff, schedules, and costs  
   Galorath

8. SLIM-ESTIMATE  
   Cost, schedule, and reliability estimation using Putnam’s Software Lifecycle Model  
   Quantitative Software Management

### Conclusion

An awareness of the spectrum of available techniques and tools is not sufficient to make you an estimation expert. Accurate estimation also takes experience in the target business, technology, and operations domains, as well as a clear understanding of what techniques can reasonably be used when (which I'll explain more about in Part 2 of this article). Estimation is a very risky business when the margin for error can fluctuate significantly. This is thanks to many factors, some of which are hard to even predict, let alone account for.

Nevertheless, it is important to understand estimation dependencies and the need for timely data, in order to avoid over- and under-estimation and to deliver estimates with a reasonable degree of actual accuracy. More about this in Part 2.

### Acknowledgements

The author wants to express gratitude to Larry Simon as well as Cal Rosen of ActionInfo Consulting for sharing highlights of their work in their respective areas of expertise.

### Notes and references

2. This article from *The Rational Edge* provides a good summary of the incremental estimating practice: [http://public.dhe.ibm.com/software/dw/rationaledge/oct03/f_estimate_b.pdf](http://public.dhe.ibm.com/software/dw/rationaledge/oct03/f_estimate_b.pdf)

5. 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)


8. This research article by Chris F. Kemerer contains empirical validation of software cost estimation models: [http://portal.acm.org/citation.cfm?id=22906&dl=](http://portal.acm.org/citation.cfm?id=22906&dl=)


10. This is one of the many articles about estimation written by Cutter experts. It is worth taking a look, but that may require a paid subscription: [http://www.cutter.com/content/alignment/fulltext/advisor/2006/bit061220.html](http://www.cutter.com/content/alignment/fulltext/advisor/2006/bit061220.html) (link no longer works).