

FEATURED PAPER
Effective Software Sizing
By Dan Galorath

IT departments are under increasing pressure each year to deliver projects on time and on budget. Finance teams closely monitor project budgets to ensure ROI, while marketing teams keep a close eye on schedules to ensure software will be delivered when promised.

Despite all this monitoring, project overruns are alarming. The Standish Group –well-known for its 12+ years of IT project research – reports budget overruns for projects since 2000 have averaged 50%, while schedule overruns occur in more than 75% of all projects.

The key to consistent project success is effective software sizing, but sizing is often the most difficult part of the project estimate. Project managers grapple with being able to accurately estimate the amount of code needed to satisfy product requirements, and they struggle to quantify risks and uncertainties associated with their estimates.

Why is sizing important?

Software size directly relates to development effort. An accurate size estimate therefore, will yield accurate cost and schedule estimates.

In reviewing software program estimates for the past 20 years, Galorath has found estimates commonly fall short because:

Not enough time and/or effort is spent up front estimating projects. The project size is not fully defined – or not commonly understood – by members of the project team.

Size is treated as an independent variable. The size estimate is changed to achieve desired cost or schedule estimates, without considering the impact of the size on the project.

Data from previous projects is not used as input into a sizing estimate because it is believed the differences are too great.

These mistakes cause inconsistent estimates that are many times simply unbelievable. At worst, everything about the project is wrong, resulting in projects that can't be executed. Typically the model is blamed, when in fact the methodology surrounding the estimation process is at fault.

The key to achieving accurate size estimates is understanding:

- What are you trying to estimate?
- How will your project be measured?
- How will your project be estimated?

What Are You Trying to Estimate?

The lifecycle of a software project begins with an idea; an identified need in the marketplace that can be satisfied with software.

In order for the idea – or product – to be successful, the product manager should have an understanding of the product's ROI. To formulate ROI, the product manager will make assumptions regarding the project size.

The product manager communicates the idea to the project team via product requirements. These requirements in essence reflect the product manager's viewpoint; the 'what' for the project.

The project team's focus is on 'how' to design and build the product. Through their analysis, the project team also forms an opinion of project size.

Typically, the different viewpoints into the project cause differing opinions of size. Both viewpoints are important however, and should be well understood before sizing a project. Measuring project size from both viewpoints will resolve differing opinions of size.

How Will You Measure Your Project?

Function points and lines of source code (SLOC) are two common methods for software sizing. While the following document will focus primarily on SLOC, many of the observations are just as applicable to function point methodology.

While SLOC is a reasonable form of measurement, it's not without pitfalls. Measuring lines of code is easy after a project is complete, but developers struggle to estimate how many lines of code it will take to complete a project before coding begins.

During the requirements phase, size is still a fuzzy concept. Metrics that mirror the product manager's viewpoint, e.g. number of requirements, comparisons to previous projects, and function points, help bring the 'what' into focus.

As the project progresses to the design phase, metrics that mirror the developers' viewpoint can be used. Object-oriented metrics such as use cases can be counted at this stage.

Once implementation has begun, precise, technical measurements such as SLOC, executable lines of code, or other similar 'grammatical metrics' can begin to be used.

Total versus Effective Size

The two project viewpoints – ‘what’ versus ‘how’ – bring into play two perspectives on size: total size and effective size.

The project’s total size reflects the total functionality of the program: the ‘what’ viewpoint.

Effective size takes into account the amount of effort required to build the ‘what’. Since many programs include a combination of new and pre-existing software, effective size factors in:

- Re-design, re-implementation, and/or re-testing of pre-existing code.
- Reverse-engineering of existing code required to discover how it works.
- Design, development, and testing of new code.
- Use of code generators, GUI builders, or off-the-shelf products.

Total size will remain constant over the course of the project if requirements remain unchanged. Effective size however, changes as the mix of pre-existing and new lines of code are refined. It is not uncommon for the project team at the start to assume they’ll be able to re-use a large portion of existing code. As development proceeds, the team discovers that their ability to re-use is less than anticipated. More new code needs to be written. The mix of new versus pre-existing code raises the level of effort required for the project, impacting project cost and schedule.

More often than not, requirements also grow over time. Sharp growth periods occur due to added functionality and better understanding of requirements. Effective software sizing requires that you understand size, and factor code growth, into your estimates.

Expressing Uncertainty

The final size of a finished program can be expressed as a single point value. However when estimating, single point values don’t tell the whole story.

Project teams need to be able to express their:
Confidence (what is the probability of not exceeding this value?)
Certainty (how wide is the probability distribution?)

Providing three estimates accounts for project uncertainties:

Least Estimate (1% probability of success)
‘I can’t imagine the result being any smaller than this.’

Likely Estimate (best guess)
‘If I were forced to pick one value, this would be it.’

Most Estimate (99% probability of success)

'I can't imagine the result being any larger than this.'

Project managers miss this point when they size a project and only provide one point estimate. By providing a range of values, the project team's accuracy and confidence in the estimate improves.

How Will You Estimate Your Project?

Some software size metrics can be directly measured at certain stages of development. Once the requirements are created, the number of requirements and function points can accurately be counted. Once the software is designed, the number of objects, use cases, and pre-existing software in the project lines of code can be counted.

Other metrics require estimation. Until the new source code has been written, you can only estimate the number of lines of code that will be needed. Until the design is complete, you can only estimate the number of use cases.

Fortunately, virtually any metric that can be measured can be estimated. Examples of estimation techniques include:

- Expert Judgment,
- Delphi Analysis,
- Analogy, and
- Database Comparison.

(each technique is described in subsequent sections.)

Which technique(s) you choose will depend on the expertise, data, and time, available when sizing your project. Techniques chosen will also depend on the level of accuracy required.

What's the bottom line? There is no single best method, because estimation has no single correct answer. The Galorath size study methodology recommends that all sources of information and multiple estimation methods be used to achieve an estimate with a high degree of confidence. This approach helps identify outlier estimates, and as the results of each technique are charted, clusters of estimates tend to emerge.

Expert Judgment

This technique can be described as 'What does the development team think?'

You rely on an expert – not historical data – to estimate size. It can be very accurate and effective, especially if the expert has a great deal of domain knowledge and experience.

The challenge lies in the ability to find a true expert that can provide judgment. And if the playing field changes – i.e. a different, unfamiliar application type is needed – the expert may have difficulty providing an accurate estimate.

Delphi Analysis

Delphi analysis uses a panel of knowledgeable persons; each providing anonymous, independent answers to a series of questions. Each panelist is provided a summary of opinions before each series of questions, and so with each series, it is expected that the range of estimates will decrease and the median will approach the 'correct' answer.

Similar to expert judgment, historical data is not required. But unlike judgment, a range of expertise is incorporated into the final estimate, thus providing a statistically derived estimate with a known confidence level. This technique does take additional time, effort, and coordination to conduct the study, and may be difficult to assemble a panel of experts to participate.

Analogy

Analogy applies information learned from past projects. A specific program for which actual size information is known is chosen, and compared to the project to be estimated. Judgments are made as to whether the unknown program is bigger, smaller, or comparable to the known program. Data points such as platform, application, acquisition method, development method, development standard, language, and functionality are used for comparison.

The advantage of this approach is that you are using history and familiarity with a past project to estimate. Choosing an analogous program can be tricky, though. In general, the program chosen should have data points in common with the program to be estimated. In all cases, be sure to choose analogies that you can explain.

Database Comparison

Proprietary sizing databases can be powerful for estimation. Relevant data points for past projects in your company are captured along with normalized size metrics.

Like analogy, data points for these projects are compared to data points for the project to be estimated. However with database comparison, statistical significance is possible because more data can be used to derive the estimate.

Development and maintenance of the sizing database can be daunting. Fortunately off-the-shelf tools (e.g. SEER Repository) can be used to minimize effort.

A Case Study in Sizing

To demonstrate sizing methodology in practice, assume your task as project manager is to develop a size estimate for a particular program.

Resources available for estimation include the development team, other development teams in the organization, and the organization's software process improvement group. You also have available a firm software design, and a history of performance for similar projects.

The first step towards effective software sizing is to understand what it is you are trying to estimate. The software design provides the answer to this question.

Next, define how the project will be measured by establishing the estimation methods and ground rules.

Five estimation techniques are chosen based on the available resources: expert judgment, Delphi analysis, analogy, functional measurement, and database comparison.

Two ground rules are established:

All methods of sizing will be normalized to the C++ development language.
SLOC estimates will be normalized to logical lines of code.

The estimation process can begin. Each estimation technique, and the resulting range of estimates are described below.

Expert Judgment

The lead developer is consulted. The opinion is that this project will be about 50,000 lines of C++, +/- 10,000 lines.

Delphi Analysis

A panel of six experts from several development teams and the process improvement group are assembled. Five iterations were conducted and results tallied.

Analogy

Several analogous programs are selected based on similar platform, application, and development environment. Estimates are drawn by comparing applications.

Functional Measurement

Use cases are counted based on available design documentation. Metrics were used to convert use cases into C++ SLOC.

Database Analysis

Thirty data points were selected from the repository based on similarity in application

and development environment. Standard language conversion factors were used to normalize to C++.

The resulting estimates include:

Estimation Technique	Normalized Lines of C++		
	Least	Likely	Most
Expert Judgment	40,000	50,000	60,000
Delphi Analysis	32,865	53,720	75,940
Analogy	45,320	59,450	78,950
Functional Measurement	24,721	27,468	32,962
Database Analysis	23,150	72,350	115,670

Next, plot all estimation values and look for clusters of values to determine the composite estimates.

Estimation Technique	Normalized Lines of C++		
	Least	Likely	Most
Expert Judgment	40,000	50,000	60,000
Delphi Analysis	32,865	53,720	75,940
Analogy	45,320	59,450	78,950
Functional Measurement	24,721	27,468	32,962
Database Analysis	23,150	72,350	115,670
Composite	24,721	54,015	78,950

By following Galorath's sizing study methodology, the composite estimates can be assured to have a high degree of accuracy because:

Project size was measured from both viewpoints: 'what' and 'how'.

Multiple measurement techniques were used to eliminate outlier and gut-feel estimates. Uncertainties associated with the project have been accounted for with the range of estimates.

The most difficult part of a project estimate has been accomplished: effective software sizing.



Dan Galorath
Author



Daniel D. Galorath has over 35 years of experience in the software industry where he has solved a variety of management, costing, systems, and software problems, and performed all aspects of software development and management. Mr. Galorath is founder and president of Galorath Incorporated, maker of the SEER® suite of estimation tools. Mr. Galorath is one of the principal developers of the SEER-SEM™ Software Estimation Model. Mr. Galorath completed his undergraduate work and MBA from California State Universities. He is a member of the International Society of Parametric Analysis (ISPA), Society of Cost Estimation and Analysis (SCEA), IEEE, the International Function Point Users Group (IFPUG), and the Association of Computing Machinery (ACM). He was honored with the Freiman Award, recognizing his long-term contributions to the field of parametric analysis. Mr. Galorath teaches courses in software cost, schedule, and risk analysis; software project management; software engineering; systems architecture, and other related topics. He has lectured internationally and is the author of many papers about software project management. Mr. Galorath is also the co-author, with Michael W. Evans, of the book, "Software Sizing, Estimation, and Risk Management." Mr. Galorath can be reached at info@galorath.com. His website is www.galorath.com