

# System Level Metrics for Software Development Estimation

Ricardo Valerdi, Yue Chen, Ye Yang  
Center for Software Engineering, University of Southern California  
{rvalerdi, yuec, yey}@usc.edu

## Abstract

*Previous empirical observations on the development of software systems indicate that the effort required for developing software is primarily driven by the size of the software which is measured by either the source line of code (SLOC) or function based metrics, such as Function Points. However, since most large scale systems today are more software intensive, where with the growing complexity and coupling nature between hardware and software, estimating development effort for these systems requires metrics that go beyond software oriented sizing techniques. In this spirit we propose a set of four system level metrics that provide a better indication of software complexity and in turn software development effort. These metrics are (1) requirements, (2) interfaces, (3) algorithms, and (4) operational scenarios.*

## 1. Motivation

Accurately predicting the size of a project will result in a better estimate of the software development effort. However, researchers have argued that current SLOC and function point estimates are difficult to perform at the early stages of a project because of the lack of reliability [1]. SLOC-based estimates are usually based on expert judgment or comparison with similar previous projects, it is simple and widely used, however greatly dependent on particular programming language and very difficult to visualize early. The Function Points approach helps estimate the size of software in terms of functionalities from the user's point of view, it is language independent and easier to make early estimates, however, FP counting is hard to automate and greatly relies on estimator's experience. Hence, we propose using metrics at a higher level of abstraction enables the early estimation of software systems with a higher degree of certainty.

## 2. Methods

We obtained the four system level metrics by surveying industry practitioners in the domain of large-scale military systems. Our initial results, entirely expert-based, evolved from a list of eight system level metrics and were consolidated to four after two rounds of a Delphi survey. The Delphi technique is a proven method for validating expert opinion and has been used in previous software engineering studies [2]. Existing software development estimation models such as COCOMO, PRICE-S, and SEER-SEM are strictly based on SLOC and function based metrics as inputs. The proposed system level metrics approach can complement these existing tools and provide a system level perspective to the estimate. By system level we refer to a higher level of abstraction that can be characterized by the system being delivered. For example, a satellite ground station may contain a number of software modules that need to be developed. The effort that corresponds to these software modules can be estimated by the aforementioned software development models. The acquisition or developing organization needs to also have an idea of what the total cost of the satellite ground station will be. A percentage of the cost can be estimated by the software tools and another part can be estimated by hardware tools. A third category of effort that involves integrating the hardware and software also needs to be estimated. This layer can be estimated by the four proposed metrics.

## 3. Results

A survey of 40 industry practitioners helped us identify the most relevant system level metrics, or drivers, for software systems. The average experience with software systems of the respondents was 18 years. Table 1 obtains the names and definitions of these drivers.

Table 1. System Level Drivers for Software Systems

<p><i># of System Requirements</i>                  This driver represents the number of requirements for the system-of-interest at a specific level of design. Requirements may be functional, performance, feature, or service-oriented in nature depending on the methodology used for specification. They may also be defined by the customer or contractor. System requirements can typically be quantified by counting the number of applicable “shall’s” or “will’s” in the system or marketing specification. Do not include a requirements expansion ratio – only provide a count for the requirements of the system-of-interest as defined by the system or marketing specification.</p>
<p><i>Number of Major Interfaces</i>                  This driver represents the number of shared major physical and logical boundaries between system components or functions (internal interfaces) and those external to the system (external interfaces). These interfaces typically can be quantified by counting the number of interfaces identified in either the system’s context diagram and/or by counting the significant interfaces in all applicable Interface Control Documents.</p>
<p><i>Number of Critical Algorithms</i>                  This driver represents the number of newly defined or significantly altered functions that require unique mathematical algorithms to be derived in order to achieve the system performance requirements. As an example, this could include a complex aircraft tracking algorithm like a Kalman Filter being derived using existing experience as the basis for the all aspect search function. Another example could be a brand new discrimination algorithm being derived to identify friend or foe function in space-based applications. The number can be quantified by counting the number of unique algorithms needed to support each of the mathematical functions specified in the system specification or mode description document.</p>
<p><i>Number of Operational Scenarios</i>                  This driver represents the number of operational scenarios that a system must satisfy. Such threads typically result in end-to-end test scenarios that are developed to validate the system and satisfy all of its requirements. The number of scenarios can typically be quantified by counting the number of unique end-to-end tests used to validate the system functionality and performance or by counting the number of high-level use cases developed as part of the operational architecture.</p>

In addition to identifying the drivers, it was important to obtain their relative weights in order to compare the influence of each driver on system size. The Delphi respondents also provided their opinions in the survey. We used an approach similar to the Function Points technique in software development [3]. Each driver contains “Easy”, “Nominal”, and “Difficult” which provides for three different weight levels for each of the four drivers. Table 2 includes the survey results from the four system level drivers.

Table 2. Relative weights for System Level Drivers

	Easy	Nominal	Difficult
# of System Requirements	0.49	1.00	4.23
# of Interfaces	1.50	3.92	8.21
# of Critical Algorithms	3.01	5.84	16.64
# of Operational Scenarios	10.31	24.58	53.85

This approach uses “Nominal” # of System Requirements as the baseline for the driver weights. For example, the effort required for “Easy” Critical Algorithms is three times compared to “Nominal” # System Requirements.

The results from the Delphi survey show that the number of operational scenarios have the most influence on software system size. The weights provided in Table 2 also provide an initial comparison of the relative difficulties of drivers. For example, easy requirements take approximately half the effort as nominal ones, as seen by the 0.49 and 1.0 weights. Similar logic applies for the rest of the drivers.

#### 4. Future work

The next steps in this experiment are to (1) collect historical project data to validate the results from the Delphi survey; and (2) build these drivers and weights into a model that can help developers estimate system level costs.

#### Acknowledgements

We would like to thank the respondents of the Delphi survey and the corporate affiliates of the Center for Software Engineering for supporting this research.

#### 5. References

- [1] Kemerer, C. F., Porter, B. S., Improving the Reliability of Function Point Measurement: An Empirical Study, IEEE Transactions on Software Engineering, Vol. 18, No. 11, Nov 1992.
- [2] Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B., Horowitz, E., Madachy, R., Reifer, D. J. and Steece, B. (2000). Software Cost Estimation With COCOMO II, Prentice Hall.
- [3] Albrecht, A. J., Gaffney, J., Software function, source lines of code, and development effort prediction: A software Science validation, IEEE Transactions on Software Engineering, Vol. SE-9, pp. 639-648, 1983.