

An Empirical Study of eServices Product UML Sizing Metrics

Yue Chen, Barry W. Boehm, Ray Madachy, Ricardo Valerdi
Center for Software Engineering, University of Southern California
 {yuec, boehm, madachy, rvalerdi}@sunset.usc.edu

Abstract

Size is one of the most fundamental measurements of software. For the past two decades, the source line of code (SLOC) and function point (FP) metrics have been dominating software sizing approaches. However both approaches have significant defects. For example, SLOC can only be counted when the software construction is complete, while the FP counting is time consuming, expensive, and subjective. In the late 1990s researchers have been exploring faster, cheaper, and more effective sizing methods, such as Unified Modeling Language (UML) based software sizing. In this paper we present an empirical 14-project-study of three different sizing metrics which cover different software life-cycle activities: requirement metrics (requirement), UML metrics (architecture), and SLOC metrics (implementation). Our results show that the software size in terms of SLOC was moderately well correlated with the number of external use cases and the number of classes. We also demonstrate that the number of sequence diagram steps per external use case is a possible complexity indicator of software size. However, we conclude that at least for this 14-project eServices applications sample, the UML-based metrics were insufficiently well-defined and codified to serve as precise sizing metrics.

1. Introduction

Software size is important in the management of software development because it is a generally reliable predictor of project effort, duration, and cost. According to the Software Engineering Institute's (SEI) Capability Maturity Model (CMM), size is recommended as one of the most fundamental measurements beginning as early as Level 2. [15] In the past two decades, counting number of Source Line of Code (SLOC) [14] and Function Points [1] have been dominating software sizing approaches. According to Daniel V. Ferens' survey in 1999, "software

size is a key input to all software cost estimation models." [8]. For example, SLOC has been used as the primary size input in many cost estimation tools such as COCOMO, COCOMO II, SLIM, SEER, and Price/S. To convert the FP into SLOC (sometimes vice versa), backfiring has been used as "a standard function for many commercial software-estimating tools." [11]

Unfortunately, there are significant drawbacks in SLOC and FP sizing for software estimation. For example, SLOC can only be accurately counted when the software construction is complete, while the most critical software estimations need to be performed before construction. The FP can only be manually counted, and the estimator has to have special expertise and experience to do so. Furthermore, "FP counting involves a degree of subjectivity." [7] The FP counting rules are sufficiently well-defined and codified to enable experienced FP counters to be consistent between roughly 15-20% in FP counts [12]. Even then, though, manual FP counting and recounting processes are unavoidably time consuming and expensive relative to automated counting.

Facing these challenges, researchers are looking for faster, cheaper, and more effective methods to estimate software size. In 1999, Fischman presented "UML-Based Software Sizing" to estimate software size from its Unified Modeling Language (UML) design. [9] John Smith at IBM Rational Rose Software Inc. proposed a framework of "The Estimation of Effort Based on Use Case", to estimate software effort from use cases. [18] Takuya Uemura, Shinji Kusumoto, and Katsuro Inoue built an automated tool to estimate software size in terms of FP from the UML class diagrams and sequence diagrams which are created with Rational Rose. [20] In 2002, Galorath and Ferens reported a sizing approach by counting Textual Specification (TS) such as requirements. [10]

However, to date, few empirical studies have been conducted to contrast these new approaches. Just as John Smith (Rational Software) mentioned in his paper, "the paper is full of bold (or should that be bald) conjecture because I can see no other way forward given the lack of work and data in this area." [18]

In our paper we present a set of empirical data from the Center for Software Engineering (CSE) at USC to analyze software sizing metrics which cover different life-cycle activities. The data was collected from USC CSCI 577 class eService projects over a period of three years. Currently these projects are organized in the MBASE Archives database at the NSF Center for Empirically-Based Software Engineering. [21] Our results show that the software size in terms of SLOC is moderately well correlated with the number of external use cases in use case diagrams and the number of classes in class diagrams.

Section 1 provides a background of software sizing approaches and research trends, section 2 discusses the advantages and challenges in UML sizing, section 3 presents our analysis approach, section 4 illustrates the empirical results we observed, section 5 analyzes the experiment results, and section 6 summarizes the conclusions.

2. UML Sizing

As an emerging industrial standard for object-oriented software analysis and design, UML has been widely used in presenting and visualizing software architecture. There are three categories of UML diagrams which describe the software system from different perspectives: structural, dynamic, and model management. These diagrams are summarized in Table 1. [17]

Table 1. UML Diagrams

Category	Diagram
Structural	Use case diagram
	Component diagram
	Class diagram
	Deployment diagram
Dynamic	Sequence diagram
	Collaboration diagram
	Activity diagram
Model Management	State transition diagram
	Class diagram describing packages, sub-systems and models

We believe UML metrics have a great potential to be a promising sizing method for the next generation of software estimation tools because they can be used earlier in the project phase, and their electronic representation makes them cheaper and easier to use. Table 2 compares the UML sizing approach with the SLOC, FP approaches

from three dimensions: difficulty level for automated counting, countable phase, and counting rules.

Table 2. Sizing approaches comparison

Sizing Approach	Difficulty level for automated counting	Countable Phase	Counting Rules
UML	Low	Elaboration Construction	Exact, but Uncodified
SLOC	Low	Construction	Exact, codified
FP	High	Inception Elaboration Construction	Codified, but Subjective

Compared to SLOC and FP, UML sizing has the following advantages:

- Automatic UML element counting tools can be developed to count the number of classes in class diagrams, the number of steps in sequence diagrams, etc. [20] This makes the method faster and cheaper than FP
- Usually, the UML design is done before software construction; therefore it enables the estimator to perform software estimation before starting construction
- UML design describes not only the software capabilities and use cases from the user's perspective, but also explains the implementation blueprints from the developer's perspective. Furthermore, UML sizing is different from either FP which is "initially designed to measure requirements" [5] or SLOC which "attempt to characterize software size in terms of the number of software instructions" [14]. Therefore, it can provide a much more complete information base for better size estimation

However, there are no codified definitions and counting rules for the UML elements being counted, and different projects use UML to different extents and to describe to different detail levels, while the source code represents the final project. This is particularly true at different points in the life cycle. UML constructs (and to some extent FP constructs) tend to be fewer and more coarse-grained at the beginning of a project, and more numerous and fine-grained by the end. Thus a model calibrated to UML data collected at the end will tend to underestimate a new project whose size is based on UML constructs available at the beginning. There are no mature processes or guidelines for software organizations to follow for collecting, organizing and maintaining project UML sizing

data. Also, as an emerging new sizing approach little empirical data on UML sizing has been collected to accommodate the related research.

3. Analysis Approach

In order to measure the effectiveness of different software sizing metrics such as UML sizing, we analyzed data from the USC eServices Software Engineering project course experience base over a period of three years.

3.1 Sizing Metrics

The purpose of the study was to measure the relationship between different sizing metrics and identify significant sizing indicators. In this spirit we analyzed three types of sizing metrics which cover different life-cycle activities: requirement metrics (requirement), UML metrics (architecture), and source line of code (implementation).

UML Metrics

We derived the sizing metrics for the system design activity from three types of UML diagrams: Use Case Diagram, Class Diagram, and Sequence Diagram because these diagrams capture system information from all three major perspectives that UML focuses on: operational, structural and behavioral. The metrics were arrived at from the following sources:

- The Operational Concept Description (OCD) document provided the number of external use cases. (The external use cases refer to those high-level use cases whose actors are outside the system boundary)
- The System and Software Architecture Description (SSAD) document provided the:
 - Number of decomposed use cases
 - Number of steps in sequence diagram
 - Number of classes
 - Average number of methods per class
 - Average number of attributes per class

Source Code Metrics

As the sizing metrics for the implementation activity, we counted the total logical SLOC of each project because the SLOC metric has the following advantages:

- It is sensitive to programming language type and level, thus capturing the project implementation characteristics better than FP metrics
- Compared to FP counting, the subjective factors are

- reduced, thus making the process more repeatable
- Sufficient automatic counting tools are available to efficiently perform SLOC counting

Requirement Metrics

As the sizing metrics for the requirements activity, we counted the following types of requirements from the SSRD documents of each project:

- Number of project requirements, which are general constraints and mandates placed upon a project team, as well as the global constraints, such as budget, schedule, etc.
- Number of capability requirements, which define system capabilities that are implemented and tested.
- Number of interface requirements, which describe how the software will interfaces with other systems or users
- Number of level of service requirements, which describe the desired levels of service of the software, for example, define how well the software system should perform a capability requirement

3.2 USC eServices Projects

The CSCI 577 software engineering class at USC is a 2 semester course which provides the opportunities for graduate students to experience the software life-cycle by working on real projects with real clients. Most projects are eService applications providing web-based services to campus users. Each project is typically assigned to a 5-graduate-student team. The student developers are required to follow the Model Based (System) Architecting Software Engineering (MBASE) Guideline [4] as a process guideline to complete their project. Table 3 summarizes the important features of typical eServices projects as follows:

Table 3. Typical eService Project

Feature	Description
Domain	eServices
Team Size	5 graduate students
Duration	24 weeks
Process Guideline	MBASE
Architecture Description Language	UML
Average Number of External Use Cases	5.86
Average # of Logical SLOC	5262

Currently all the project data is maintained in the MBASE Archives database at the NSF Center for

Table 4. eServices Project List

#	PROJECT	TITLE	LANGUAGE
1	Spring 03, Team 03	UML2Web	Python
2	Spring 03, Team 06	Pilot Web Based Geotechnical Virtual Data Center	JSP
3	Spring 03, Team 09	Conference Trip Planning System	HTML, PHP, SQL
4	Spring 03, Team 13	Quality Information Management System for 577 Course	JSP
5	Spring 03, Team 14	Caroline's Closet	ASP
6	Spring 02, Team 01	Dental Library New Booklist	Perl
7	Spring 02, Team 06	ISD Interactive Web Based Contract Management System	PhP, HTML
8	Spring 02, Team 15	Strategic Risk-Value Assessment Tool	Java, JSP
9	Spring 02, Team 19	Opportunity Tree Framework	JavaScript
10	Spring 01, Team 01	Station Data Project for the Web	HTML/JAVA
11	Spring 01, Team 03	Pathology Image Search Engine	JSP/JAVA
12	Spring 01, Team 08	Full-text Titles Database	HTML/JAVA
13	Spring 01, Team14	Access & Display Archive Image Composer	JSP/ JAVA/ HTML/ SQL
14	Spring 01, Team17	Web Mail	JAVA/ HTML/ JavaScript/ JSP

Empirically-Based Software Engineering. [21] The project artifacts such as documents, prototypes, and source code are organized as “packages” and provide a snapshot of important project life-cycle milestones: the Life-Cycle Objectives (LCO), Life Cycle Architecture (LCA) Rebased Life-Cycle Architecture (RLCA), and the Initial Operational Capability (IOC). [4] For purposes of this study we analyzed all project sizing data from the following three documents in the project “Final Deliverables” packages which were delivered at the IOC milestone:

- Operational Concept Description (OCD), which describes how a proposed new system will operate within its environment
- System and Software Requirement Description (SSRD), which defines software system requirements
- System and Software Architecture Description (SSAD), which presents software system architecture design in UML

3.3 Project Selection

We studied 14 eServices projects from the USC project archive from 2001 to 2003. To maximize the data comparability, all the selected projects have:

- the same application domain: eService products
- the same project type: development intensive (instead of COTS intensive projects)
- the same process to follow: MBASE/RUP [4]

We also considered the project documentation quality and data availability when we made the selection. The list of projects is provided in Table 4.

4. Results

This section presents the empirical results we observed from our research. Note that the following adjustments were made during our counting process:

- Extraneous elements were not counted
- Physical SLOC was adjusted into logical SLOC by multiplying the physical SLOC to a (Logical SLOC)/(Physical SLOC) ratio which was calculated from the sample code files of each project
- Duplicated elements in different diagrams were counted only once

We believe that these adjustments improved the reliability of our measurements and provided a cleaner representation of what actually happened on the projects.

4.1 SLOC vs. External Use Cases

The first set of metrics that we compared was SLOC and the number of external use cases. This metric was extracted from the OCD document provided by each project team.

Figure 1 shows the relationship between the number of external use cases and the number of SLOC.

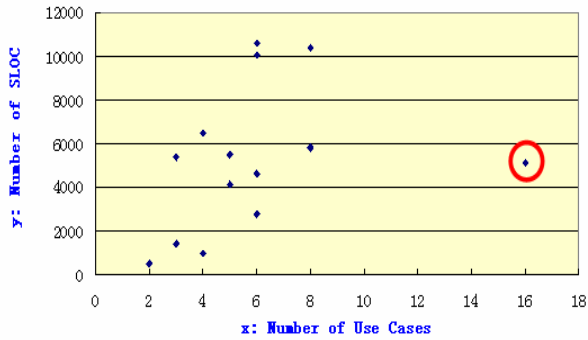


Figure 1. SLOC vs. External Use Case –with Outlier

It is clear that the number of SLOC increases as the number of external use cases increases despite the outlier which is marked with a circle.

The excessive number of use cases (16) in the outlier project results from an over detailed OCD document. Their use case diagrams were decomposed one level lower providing a level of detail of their high-level system capabilities more suited to detailed design. As a result, we removed this project from the sample set in our use case study.

This example also illustrates the point mentioned in Table 2 that counting rules for UML sizing metrics are not well defined and codified, and can be affected by differences in levels of decomposition.

Figure 2 shows the data set without project No. 6. Intuitively the trend line shows that external use cases and SLOC have a linear relationship supported by the coefficient of determination (R^2) value of 0.401. The linear equation provides an additional piece of information: approximately, for every additional external use case, there was an increase of around 1055 SLOC, at least for this sample of eServices projects.

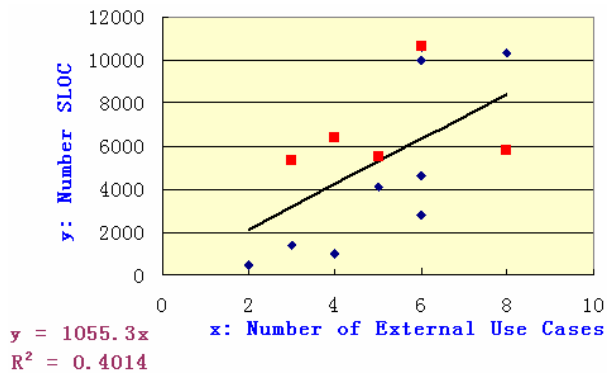


Figure 2. SLOC vs. External Use Case without Outlier

Furthermore, Figure 2 illustrates another interesting phenomenon: the number of sequence steps per use case

is likely to be a *complexity indicator* which affects the software size. We grouped the projects by the number of external use cases. In each group, the projects which have the above average number of sequence diagram steps per use case are marked with square dots. For example, in the 6-use-case group, the project represented by a big square dot is the only project which has a greater *complexity indicator* than the average of the four projects in that group. Figure 2 shows that, in general, projects which have greater number of sequence diagram steps per external use case also have larger size if their numbers of external use cases are the same.

Thus, the number of sequence steps may be a useful modifier to the number of use cases, although the pattern of square dots indicates that it would still have considerable variability.

Following this observation, we further plotted the relationship between the number of SLOC and the total number of steps in sequence diagrams as shown below in Figure 3. However, no clear trend is observed. For the linear trend line the coefficient of determination (R^2) is only 0.0021, indicating that the total number of sequence diagram steps has much less capability to predict the number of SLOC than does the total number of external use cases.

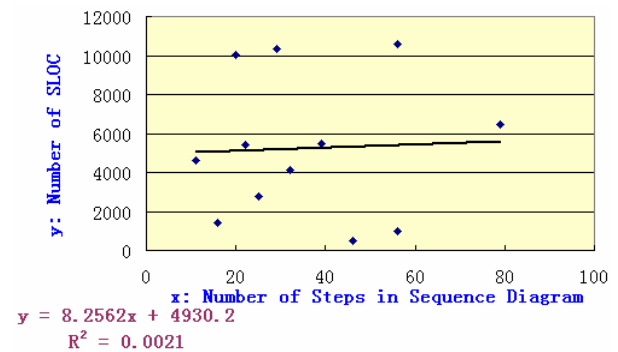


Figure 3. SLOC vs. number of steps in Sequence Diagrams

Therefore, synthesizing the analysis of Figure1, Figure 2, and Figure 3, the observations indicate that the number of steps per external use may be a minor complexity indicator or modifier in addition to external use cases which has a major impact on size.

4.2 SLOC vs. Decomposed Use Cases

The second set of metrics compared was SLOC and Decomposed Use Cases. In the eServices projects, high-level external use cases in OCD were analyzed and decomposed further in the SSAD (architecture) document.

So we derived the number of Decomposed Use Cases directly from the SSAD documents.

Figure 4 illustrates the relationship we observed between the number of SLOC and the number of decomposed use cases.

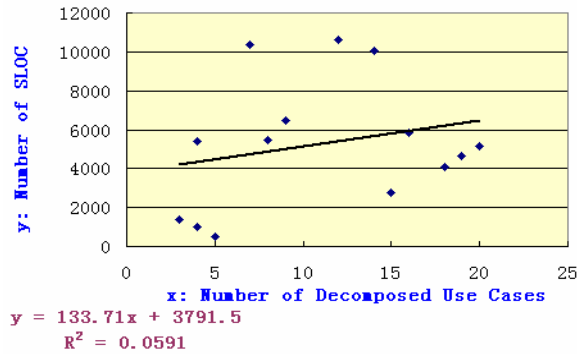


Figure 4. SLOC vs. Decomposed Use Case

No clear trend is observed in Figure 4. For the linear trend line the coefficient of determination (R^2) is only 0.059, indicating a weak correlation between the Number of Decomposed Use Cases and the number of SLOC.

As mentioned before, the UML sizing metrics are very sensitive to the level of design decomposition. Although the student developers worked from common UML usage guidelines and education, differences in interpretations, skills, and learning curves can produce significant differences in use case decomposition from team to team, therefore adding further sources of dispersion in software sizing. The challenge behind the observation above is to develop well defined and thoroughly-practiced UML levels of detail and counting rules for each stage of software development. This is discussed further in section 6.

4.3 SLOC vs. Classes

The third set of metrics compared was SLOC and classes defined in Class Diagrams. Figure 5 shows the relationship between the number of SLOC and the number of “implementation classes” which are counted from the class diagrams in the SSAD documents. These classes are named as “implementation classes” in MBASE Guidelines because they are directly mapped to the coding classes.

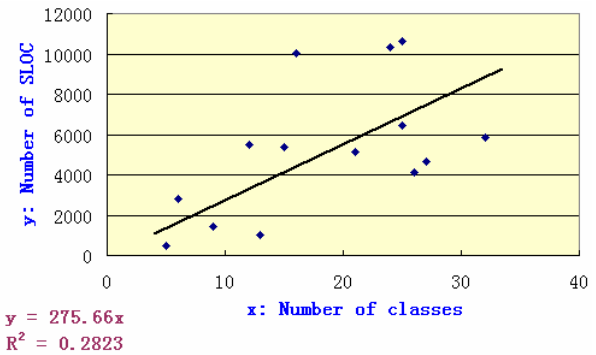


Figure 5. SLOC vs. Number of Classes

There is a clear relationship observed between the number of SLOC and the number of classes. The coefficient of determination (R^2) of the trend line is 0.283. The linear equation shows that roughly the increasing slope is around 276 SLOC/External Use Case for the eService projects.

4.4 SLOC vs. Requirements

The final set of metrics compared were SLOC and Capability Requirements (as shown in Figure 6) and SLOC and Total Requirements (as shown in Figure 7). We specifically studied the capability requirement metrics because capability requirements essentially have a close relationship with use cases which have demonstrated a strong correlation with software size in our study.

Both the number of capability requirements and the number of total requirements were manually counted from the SSRD documents. The quantity for Total Requirements included: project requirements, capability requirements, interface requirements, and level of service requirements. The capability requirements were a subset of the total requirements.

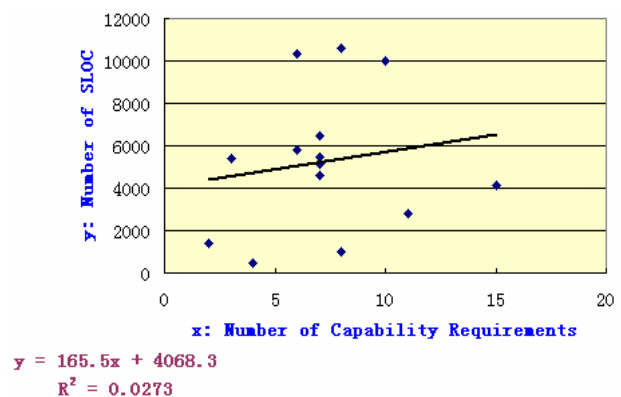


Figure 6. SLOC vs. Capability Requirements

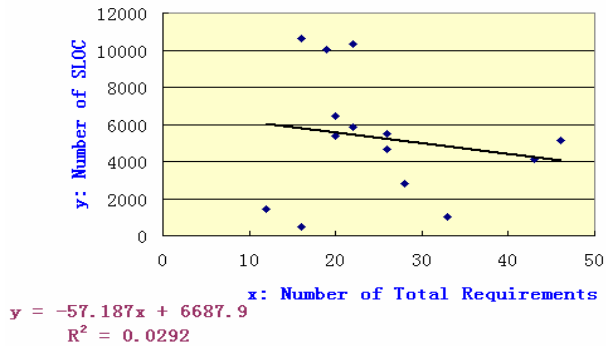


Figure 7. SLOC vs. Number of Total Requirements

Unfortunately, no clear trend is observed in either Figure 6 or 7: the R^2 value for linear trend lines is only 0.027 and 0.029 respectively, indicating weak correlations. Some possible explanations are:

- The requirement decomposition level for the projects can be different
- Difficulty of requirements can vary for different type projects
- Some requirements can be implemented by non-developed portions of COTS or other off-the-shelf products
- In Figure 7, the correlation is actually negative. This is likely the case because not all the requirements have significant impact on software size, particularly for the project requirements and the level of service requirements

5. Metrics Correlation Analysis

The five sizing metrics that were compared to SLOC provided a basis for understanding their relationships to

software size. Further exploration of these metrics was performed to develop a simple function for software size with one of the sizing metrics as an input. Four different formulas were compared and judged by their ability to fit the data in the sample. In Table 5, each row presents the type of relationship and each column shows the corresponding R^2 value in terms of linear, logarithmic, exponential, and polynomial equations for each relationship. The greatest value of R^2 for each row is underlined, and the greatest value of R^2 for each column is marked with italic font.

As shown in Table 5, the SLOC vs. External Use Cases is the strongest correlation because it has the highest value of R^2 . Furthermore, the polynomial form of the size function provides the best fit in all of the cases but one. However, there is no strong behavioral explanation for a polynomial relationship.

The equation form that best represents the relationship between SLOC and Number of External Use Cases is:

$$\text{SLOC} = 272.11 (\# \text{ External Use Cases})^{1.709} \quad (1)$$

where $R^2 = 0.538$, indicating the ratio of the goodness of fit of the regression equation to the sample data. Again, though, we have not found a good behavioral explanation for an exponent of 1.709 vs. 1.00 (linear) in the relationship.

The second strongest correlation observed was between the SLOC size and the number of classes. The form is shown in equation (3):

$$\text{SLOC} = 167.07 (\# \text{ Classes})^{1.14} \quad (2)$$

where $R^2 = 0.526$.

Table 5. R^2 Comparison

R^2	$y = a * x + b$ (Linear)	$y = a * \log(x)$ (Logarithmic)	$y = a * b^x$ (Exponential)	$y = a * x^b$ (Polynomial)
SLOC vs. External Use Cases	<i>0.401</i>	<i>0.417</i>	<i>0.467</i>	<u>0.538</u>
SLOC vs. Steps in Sequence Diagrams	0.002	0.003	0.003	<u>0.007</u>
SLOC vs. Decomposed Use Cases	0.059	0.138	0.182	<u>0.273</u>
SLOC vs. Classes	0.283	0.380	0.430	<u>0.526</u>
SLOC vs. Capability Requirements	0.027	0.081	0.066	<u>0.119</u>
SLOC vs. Total Requirements	<u>0.029</u>	0.015	0.001	0.009

6. Conclusions

Our conclusions are specific to the sample of 14 small eServices projects for which we have data on number of requirements, number of UML artifacts, and numbers of SLOC in various languages. However, they serve at least as a caution in calibrating expectations of the accuracy of UML-based sizing metrics at this stage in UML's definition and usage.

UML sizing metrics have significant potential advantages in both early availability and automatic countability. But their lack of uniform usage definitions and counting rules made them immature as precise software sizing metrics.

However, some of the UML metrics are moderately well correlated with SLOC counts. The Number of External Use Cases had R^2 value in the 0.401-0.538 range; the Number of Classes had R^2 value in the 0.283-0.526 range. Thus there is some hope of producing reasonably precise composite UML sizing metrics once more mature UML usage definitions and counting rules are defined and used. Even today, this can happen for organizations that are able to do this internally.

The other early metrics such as Number of Requirements performed considerably less well as SLOC estimators. Again, their lack of uniform definition, particularly with respect to level of detail, is likely a major cause of this weakness.

Acknowledgments

We are grateful to Mr. Ashish Agarwal, Mr. A. Winsor Brown, Mr. Jesal Bhuta, Mr. Ed Colbert, Mr. Steve Meyers, Ms. Yinghua Liu, Mr. Donald J. Reifer, and Mrs. Ye Yang for their comments and discussions presented in this paper. The research has been supported by NSF ITR Grant CCR0086078 and the USC-CSE Affiliates.

References

- [1] Albrecht, A.J. and J. Gaffney, Jr., "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation", IEEE Trans. Software Eng., vol. SE-9, no.6, pp. 639-648, Nov. 1983.
- [2] Barry W. Boehm, "Software Engineering Economics", ISBN 0-13-822122-7, Prentice-Hall, Englewood Cliffs, N.J., 1981
- [3] Barry W. Boehm, "A Spiral Model for Software Development and Enhancement", Computer, May 1988, pp 61-72
- [4] Barry Boehm, A. Winsor Brown, et al, "Guidelines for Model Based (System) Architecting Software Engineering", Center for Software Engineering, 2003
- [5] Bonnie Brown, et al., "Framework for Functional Sizing", IFPUG online article, 2003
- [6] Carol Dekkers and Ian Gunter, "Using 'Backfiring' to accurately size software: More Wishful Thinking than Science?" IT Metrics Strategies, Nov. 2000 vol. VI, no. 11
- [7] Norman E Fenton, "Software Metrics, A Rigorous Approach", Chapman & Hall, London, 1991, pp 143
- [8] Ferens, D.V., "The conundrum of software estimation models"; Aerospace and Electronics Conference, 1998. NAECON 1998. Proceedings of the IEEE 1998 National, 13-17 Jul 1998
- [9] Fischman Lee, "UML-Based Software Sizing", 14th International Forum on COCOMO and Software Cost Modeling, 1999
- [10] Daniel D. Galorath, Daniel V. Ferens, "A Software Model Based On Architecture", 17th International Forum on COCOMO and Software Cost Modeling, 2002
- [11] Jones, Capers. "Backfiring: Converting Lines of Code to Function Points." IEEE Computer 28, 11 (November 1995): 87-8.
- [12] Chris Kemerer, "Reliability of Function Point Measurement, A Field Study", Comm. ACM 36(2), pp.85-97, 1993
- [13] Arlene F. Minkiewicz, "Measuring Object Oriented Software with Predictive Object Points", PRICE Systems, L.L.C, 1997
- [14] R. Park, "Software Size Measurement: A Framework for Counting Source Statements." CMU/SEI-92-TR-20, Software Engineering Institute, Pittsburgh, PA, 1992.
- [15] Paulk et al., The Capability Maturity Model: Guidelines for Improving the Software Process. Addison Wesley, 1995. pp. 29-79.
- [16] Mike Ross, "Managing Software Size", ISPA/SCEA Conference, 2003
- [17] James Rumbaugh, Ivar Jacobson, Grady Booch, "The Unified Modeling Language Reference Manual", Addison Wesley Longman Inc, ISBN 0-201-30998-X, 1999
- [18] John Smith, "The Estimation of Effort Based on Use Case", IBM Rational Software White Paper, 1999
- [19] Richard D. Stutzke, "Using UML Elements to Estimate Feature Points", International Workshop on Software Measurement (IWSM'99), Sep., 1999
- [20] Uemura, T., Kusumoto, S., Inoue, K., "Function point measurement tool for UML design specification", Software Metrics Symposium, 1999. Proceedings. Sixth International, 4-6 Nov. 1999
- [21] http://www.cebase.org:444/usc/mbase_projects_archive/archive.html
- [22] http://sunset.usc.edu/classes/cs577a_2003/index.html
- [23] <http://www.qsm.com/CodeCounters.html>