Function Points

Yegor Bugayenko

Lecture #17 out of 24
80 minutes

All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as web sites. Copyright belongs to their respected authors.

"From the start of the software era in the 1950s until roughly 1970, software cost estimating was performed manually, using simple rules of thumb or local estimating algorithms developed through trial and error methods."

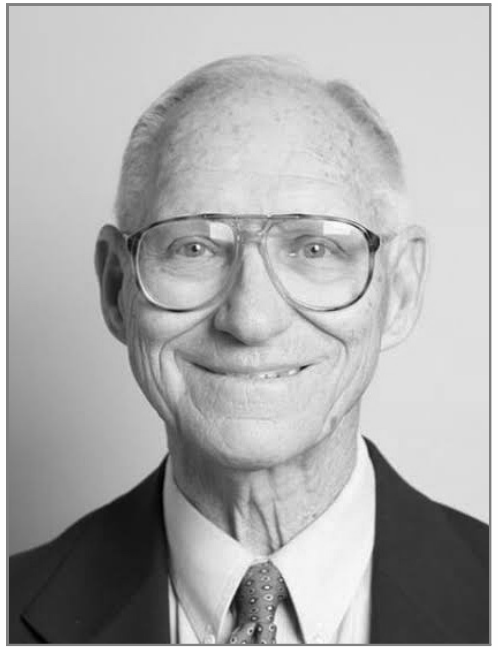
— Capers Jones. *Estimating Software Costs: Bringing Realism to Estimating.* McGraw-Hill, 2007

"Our techniques of estimating are poorly developed. More seriously, they reflect an unvoiced assumption which is quite untrue, i.e., that all will go well."

— Frederick P. Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering.* Pearson Education, 1995. doi:10.5555/540031

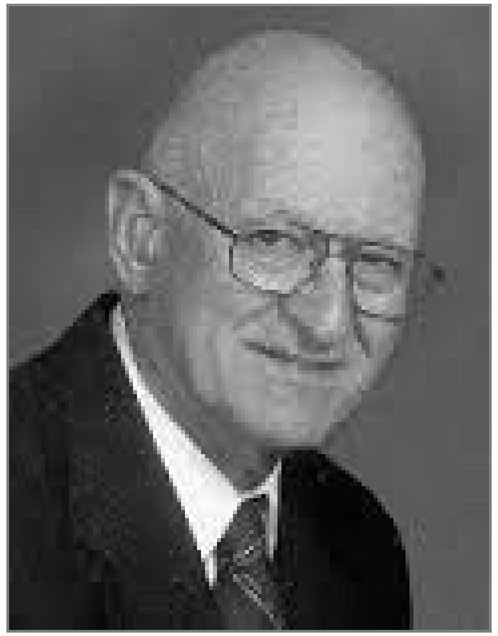
"**SLIM**: In general, the size of the product in source statements is $S = C \times \overline{K^{1/3}} \times t^{4/3}$, where C is a productivity constant, K is development effort, and t is time."

— Lawrence H. Putnam. A General Empirical Solution to the Macro Software Sizing and Estimating Problem. *IEEE Transactions on Software Engineering*, (4): 345–361, 1978. doi:10.1109/tse.1978.231521



"**COCOMO** (**Co**nstructive **Co**st **Mo**del): We compute the estimated development effort as the nominal development effort times the product of the effort multipliers for the 15 cost driver attributes... A nominal development effort is estimated as a function of the product's size in delivered source instructions in thousands (KDSI) and the project's development mode."

— Barry W. Boehm. Software Engineering Economics. *IEEE Transactions on Software Engineering*, (1):4–21, 1984. doi:10.1109/tse.1984.5010193



"**FPA**: The general approach is to count the number of external user inputs, inquiries, outputs, and master files delivered by the development project. These factors are outward manifestations of any application. They cover all the functions in an application."

— Allan J. Albrecht. Measuring Application Development Productivity. In *Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium*, pages 83–92, 1979

These counts are weighted by numbers designed to reflect the function value to the customer. The weights used were determined by debate and trial. These weights have given us good results:

- Number of Inputs X 4
- Number of Outputs X 5
- Number of Inquiries X 4
- Number of Master Files X 10

Then we adjust that result for the effect of other factors.

"If the inputs, outputs, or files are extra complicated, we add 5%. Complex internal processing can add another 5%. On-line functions and performance are addressed in other questions. The maximum adjustment possible is 50%, expressed as ±25% so that the weighted summation is the average complexity."

Source: Allan J. Albrecht. Measuring Application Development Productivity. In *Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium*, pages 83–92, 1979

Table 6.7 Function Complexity Table

Function Type	*Low*	*Average*	*High*
External Input (EI)	×3	×4	×6
External Output (EO)	×4	×5	×7
Internal Logical Files (ILF)	×7	×10	×15
External Interface Files (EIF)	×5	×7	×10
External Inquiry (EQ)	×3	×4	×6

"In order to find the adjusted function point (AFP) value, the UFP (the raw function count weighted by the appropriate complexity shown in the Table) is multiplied by the VAF."

Source: Daniel D. Galorath and Michael W. Evans. *Software Sizing, Estimation, and Risk Management: When Performance Is Measured Performance Improves.* CRC Press, 2006. doi:10.1201/9781420013122

"The entity-relationship (ER) model adopts the more natural view that the real world consists of entities and relationships. It incorporates some of the important semantic information about the real world."

— Peter Pin-Shan Chen. The Entity-Relationship Model — Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976. doi:10.1145/320434.320440

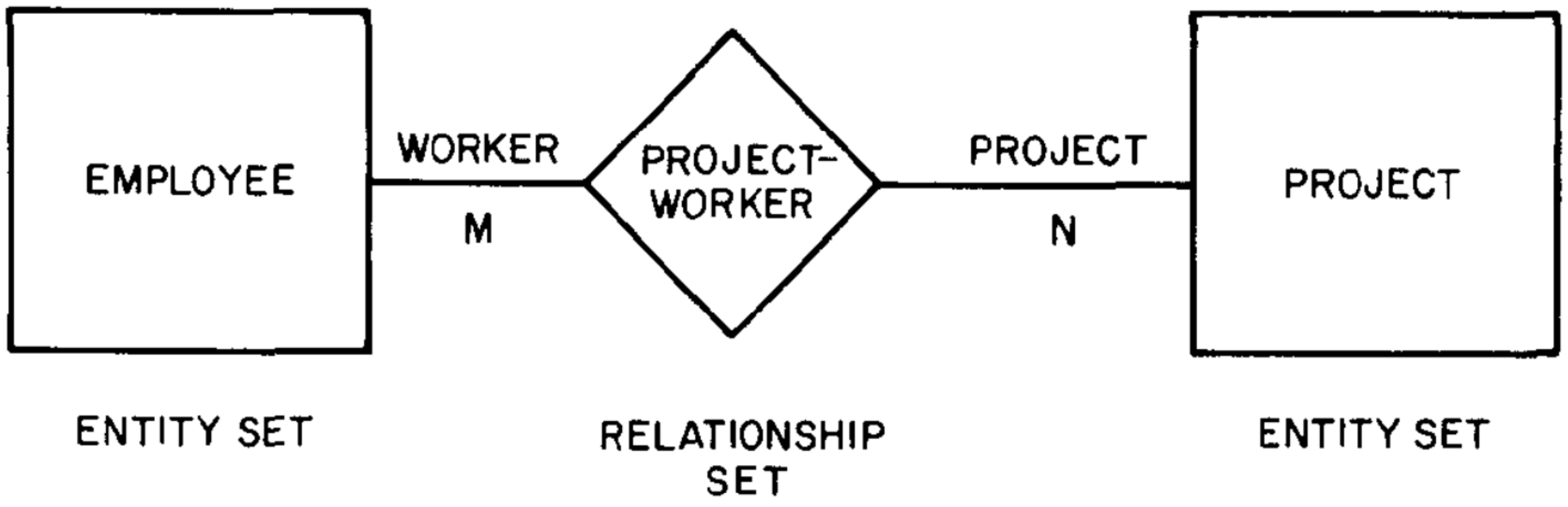
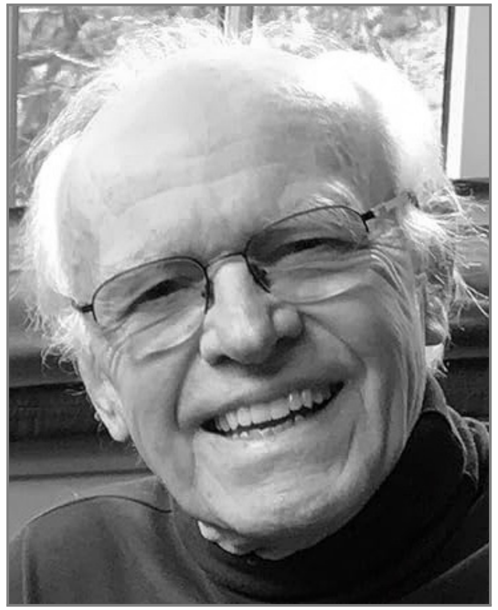


Fig. 10. A simple entity-relationship diagram

Source: Peter Pin-Shan Chen. The Entity-Relationship Model — Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976. doi:10.1145/320434.320440



"The Data Flow Diagram shows flow of data, not of control. This is the difference between Data Flow Diagrams and flowcharts. The Data Flow Diagram portrays a situation from the point of view of the data, while a flowchart portrays it from the point of view of those who act upon the data."

— Tom DeMarco. *Structure Analysis and System Specification.* Prentice Hall, 1978. doi:10.1007/978-3-642-48354-7_9

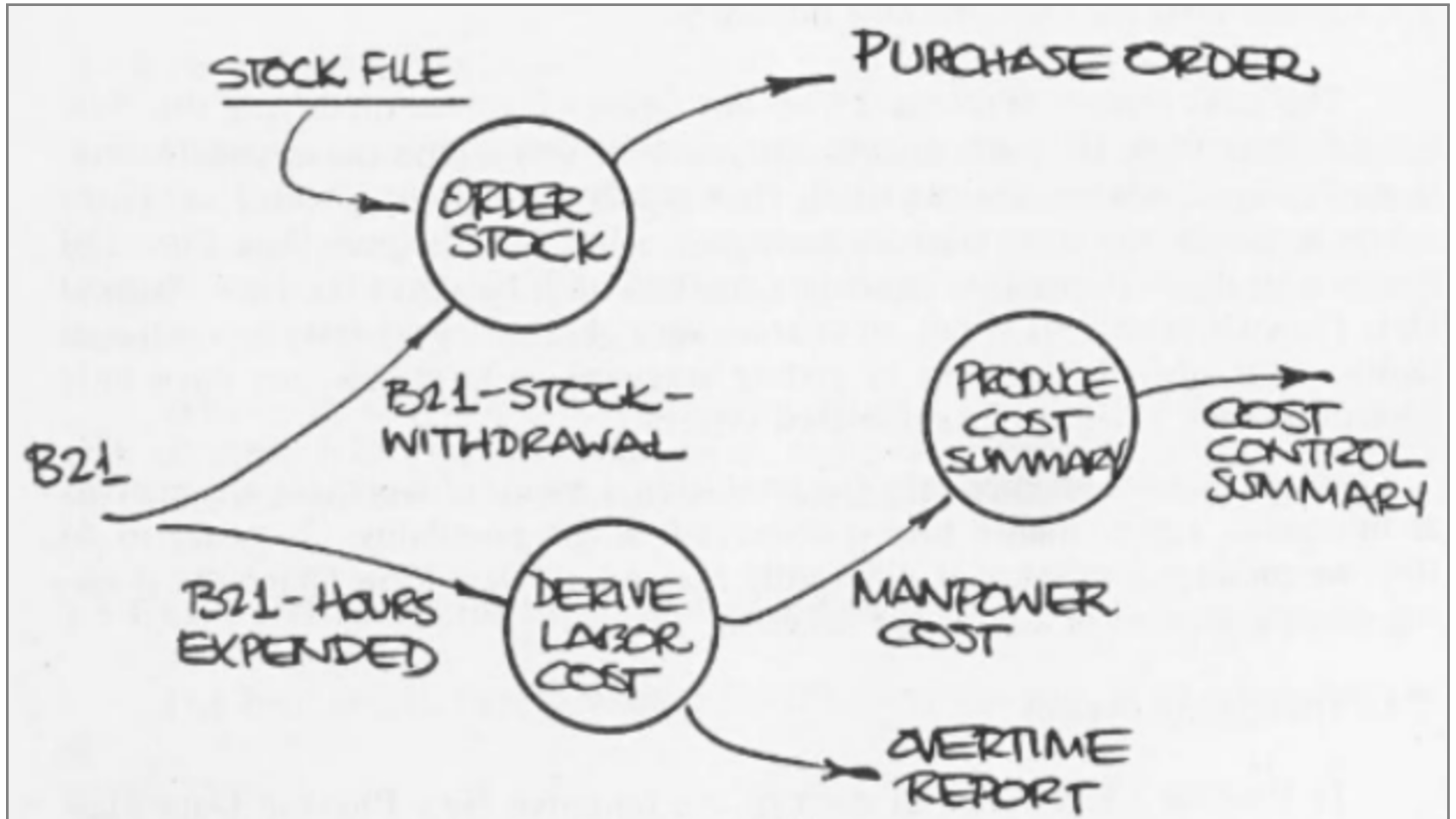
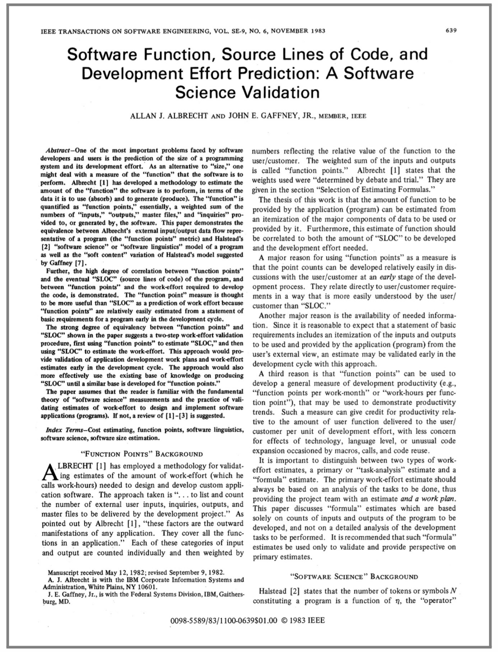


Figure 8

Source: Tom DeMarco. *Structure Analysis and System Specification.* Prentice Hall, 1978. doi:10.1007/978-3-642-48354-7_9



IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-9, NO. 6, NOVEMBER 1983 639

Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation

ALLAN J. ALBRECHT AND JOHN E. GAFFNEY, JR., MEMBER, IEEE

Abstract—One of the most important problems faced by software developers and users is the prediction of the size of a programming system and its development effort. As an alternative to "size," one might deal with a measure of the "function" that the software is to perform. Albrecht [1] has developed a methodology to estimate the amount of the "function" the software is to perform, in terms of the data it is to use (absorb) and to generate (produce). The "function" is quantified as "function points," essentially, a weighted sum of the numbers of "inputs," "outputs," master files," and "inquiries" provided to, or generated by, the software. This paper demonstrates the equivalence between Albrecht's external input/output data flow representative of a program (the "function points" metric) and Halstead's [2] "software science" or "software linguistics" model of a program as well as the "soft content" variation of Halstead's model suggested by Gaffney [7].

Further, the high degree of correlation between "function points" and the eventual "SLOC" (source lines of code) of the program, and between "function points" and the work-effort required to develop the code, is demonstrated. The "function point" measure is thought to be more useful than "SLOC" as a prediction of work effort because "function points" are relatively easily estimated from a statement of basic requirements for a program early in the development cycle.

The strong degree of equivalency between "function points" and "SLOC" shown in the paper suggests a two-step work-effort validation procedure, first using "function points" to estimate "SLOC," and then using "SLOC" to estimate the work-effort. This approach would provide validation of application development work plans and work-effort estimates early in the development cycle. The approach would also more effectively use the existing base of knowledge on producing "SLOC" until a similar base is developed for "function points."

The paper assumes that the reader is familiar with the fundamental theory of "software science" measurements and the practice of validating estimates of work-effort to design and implement software applications (programs). If not, a review of [1]-[3] is suggested.

Index Terms—Cost estimating, function points, software linguistics, software science, software size estimation.

"FUNCTION POINTS" BACKGROUND

ALBRECHT [1] has employed a methodology for validating estimates of the amount of work-effort (which he calls work-hours) needed to design and develop custom application software. The approach taken is ". . . to list and count the number of external user inputs, inquiries, outputs, and master files to be delivered by the development project." As pointed out by Albrecht [1], "these factors are the outward manifestations of any application. They cover all the functions in an application." Each of these categories of input and output are counted individually and then weighted by numbers reflecting the relative value of the function to the user/customer. The weighted sum of the inputs and outputs is called "function points." Albrecht [1] states that the weights used were "determined by debate and trial." They are given in the section "Selection of Estimating Formulas."

The thesis of this work is that the amount of function to be provided by the application (program) can be estimated from an itemization of the major components of data to be used or provided by it. Furthermore, this estimate of function should be correlated to both the amount of "SLOC" to be developed and the development effort needed.

A major reason for using "function points" as a measure is that the point counts can be developed relatively easily in discussions with the user/customer at an *early* stage of the development process. They relate directly to user/customer requirements in a way that is more easily understood by the user/customer than "SLOC."

Another major reason is the availability of needed information. Since it is reasonable to expect that a statement of basic requirements includes an itemization of the inputs and outputs to be used and provided by the application (program) from the user's external view, an estimate may be validated early in the development cycle with this approach.

A third reason is that "function points" can be used to develop a general measure of development productivity (e.g., "function points per work-month" or "work-hours per function point"), that may be used to demonstrate productivity trends. Such a measure can give credit for productivity relative to the amount of user function delivered to the user/customer per unit of development effort, with less concern for effects of technology, language level, or unusual code expansion occasioned by macros, calls, and code reuse.

It is important to distinguish between two types of work-effort estimates, a primary or "task-analysis" estimate and a "formula" estimate. The primary work-effort estimate should always be based on an analysis of the tasks to be done, thus providing the project team with an estimate *and a work plan*. This paper discusses "formula" estimates which are based solely on counts of inputs and outputs of the program to be developed, and not on a detailed analysis of the development tasks to be performed. It is recommended that such "formula" estimates be used only to validate and provide perspective on primary estimates.

"SOFTWARE SCIENCE" BACKGROUND

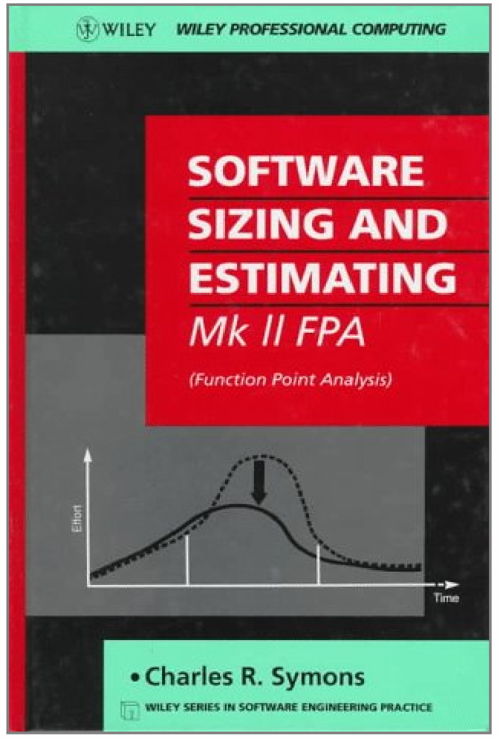
Halstead [2] states that the number of tokens or symbols N constituting a program is a function of η, the "operator"

Manuscript received May 12, 1982; revised September 9, 1982.
A. J. Albrecht is with the IBM Corporate Information Systems and Administration, White Plains, NY 10601.
J. E. Gaffney, Jr., is with the Federal Systems Division, IBM, Gaithersburg, MD.

0098-5589/83/1100-0639$01.00 © 1983 IEEE

"At least for the applications analyzed, both the development work-hours and application size in "SLOC" are strong functions of "function points" and "input/output data item count." Further, it appears that basing applications development effort estimates on the amount of function to be provided by an application rather than an estimate of "SLOC" may be superior."

— Allan J. Albrecht and John E. Gaffney. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering*, (6):639–648, 1983. doi:10.1109/tse.1983.235271



"The major difference is that **Mk II FPA**, with its finer granularity, is a continuous measure whereas IFPUG limits component size once a threshold is reached."

— Charles R. Symons. *Software Sizing and Estimating: Mk II FPA (Function Point Analysis)*. John Wiley & Sons, Inc, 1991. doi:10.5555/120462

The Functional Size (Function Point Index) is the weighted sum over all Logical Transactions, of the Input Data Element Types (Ni), the Data Entity Types Referenced (Ne), and the Output Data Element Types (No).

So the Function Point Index (FPI) for an application is:

$$FPI = W_i * \Sigma N_i + W_e * \Sigma N_e + W_o * \Sigma N_o,$$

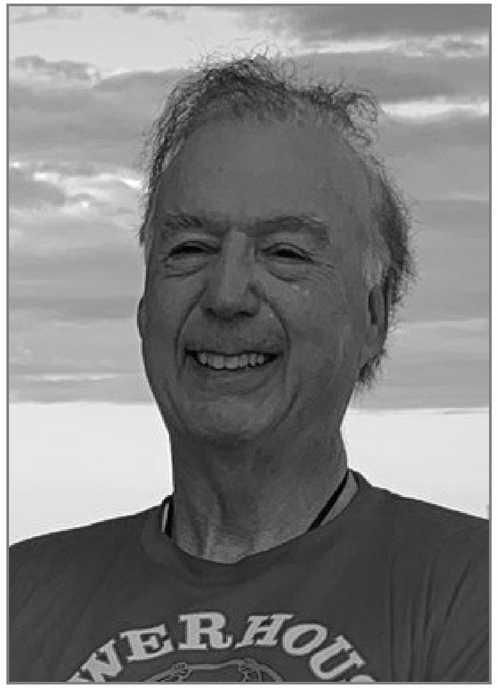
where 'Σ' means the sum over all Logical Transactions, and the industry average weights per Input Data Element Type, Data Entity Type Reference and Output Data Element Type are, respectively:

$$W_i = 0.58$$

$$W_e = 1.66$$

$$W_o = 0.26$$

Source: Charles R. Symons. *Software Sizing and Estimating: Mk II FPA (Function Point Analysis).* John Wiley & Sons, Inc, 1991. doi:10.5555/120462



"**SEER-SEM** is based on the concept that if a user can describe the essential characteristics of a project and range of size, SEER-SEM can provide estimates of schedules, efforts, staffing, risks, uncertainties, and defects, characterizing each as a most likely estimate or a risk estimate."

— Daniel D. Galorath and Michael W. Evans. *Software Sizing, Estimation, and Risk Management: When Performance Is Measured Performance Improves.* CRC Press, 2006. doi:10.1201/9781420013122

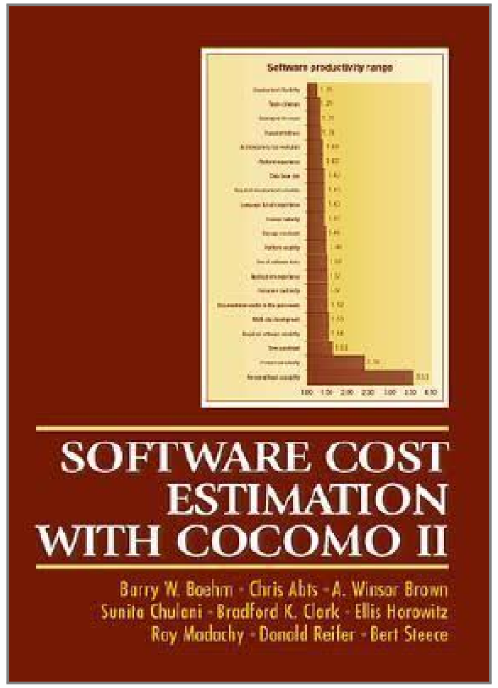
Table 6.8 Comparison SEER-SEM Function Modes: IFPUG and SEER-SEM

Functions	*IFPUG Compatible Mode*	*SEER-SEM Extended Mode*
External inputs (EIs)	*X*	*X*
External outputs (EOs)	*X*	*X*
External inquiries (EQs)	*X*	*X*
External interface files (EIFs)	*X*	*X*
Internal logical files (ILFs)	*X*	*X*
Internal functions		*X*

Note: SEER-SEM, the cost model containing SEER-FBS, will also accept unadjusted function point counts performed by traditional counting.

"SEER-FBS ("function-based sizing"), introduced in 1992, is consistent with IFPUG counting rules, but adds a sixth category (internal functions) that allows users to account for highly algorithmic processes of systems such as real-time and embedded-type systems."

Source: Daniel D. Galorath and Michael W. Evans. *Software Sizing, Estimation, and Risk Management: When Performance Is Measured Performance Improves.* CRC Press, 2006. doi:10.1201/9781420013122



"**COCOMO-II**: Success in all types of organization depends increasingly on the development of customized software solutions, yet more than half of software projects now in the works will exceed both their schedules and their budgets by more than 50%."

— Barry Boehm, Chris Abts, Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Steece Bert. *Software Cost Estimation With COCOMO II.* Englewood Cliffs, Prentice-Hall, 2000. doi:10.5555/1795822

Table 6.17 Conversion Ratios: Lines of Code per Function Point

Source Code Language	*DCG Likely*	*Capers Jones*			*Galorath Likely*
		Low	*Mean*	*High*	
Basic Assembly	575	200	320	450	320
C	225	60	128	170	61
FORTRAN	210	75	107	160	58
C++	80	30	53	125	59

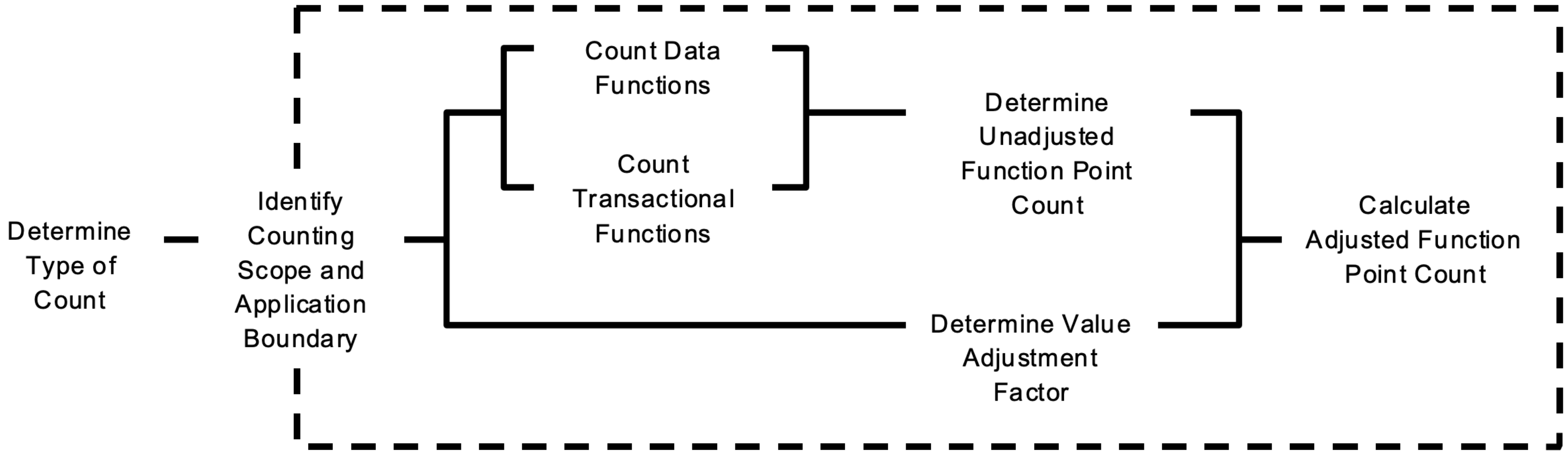
"Backfiring is converting lines of code to function points by dividing the line count by a conversion ratio. The author does not recommend backfiring as an approach to generating function points."

Source: Daniel D. Galorath and Michael W. Evans. *Software Sizing, Estimation, and Risk Management: When Performance Is Measured Performance Improves.* CRC Press, 2006. doi:10.1201/9781420013122

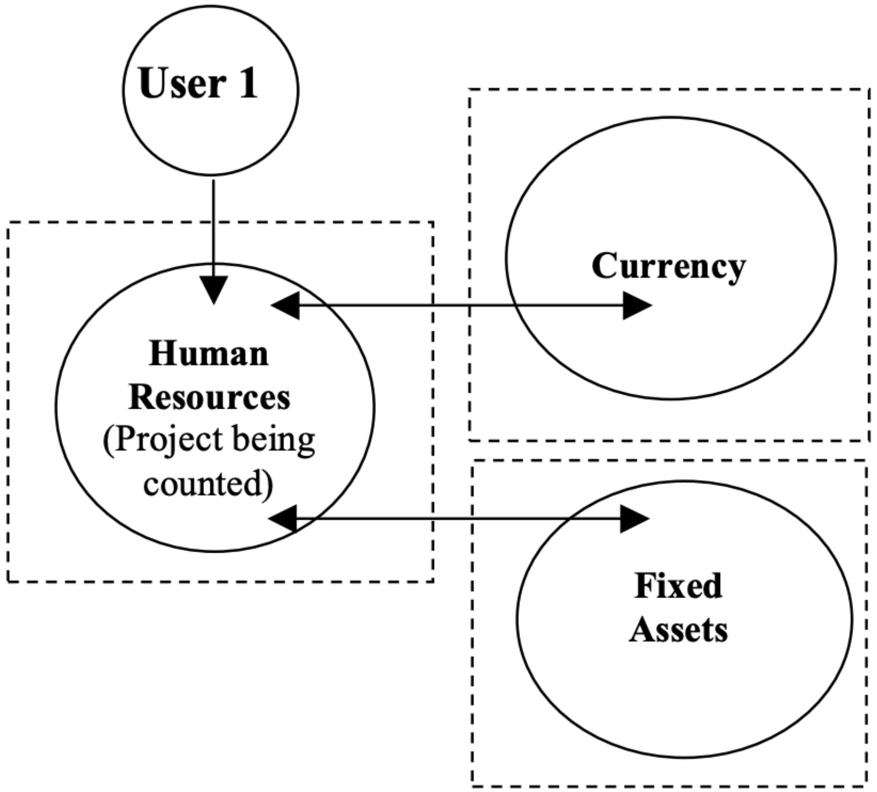
"The reliability of function point analysis is good enough to allow function points to serve as the basis for contracts, for carrying out scholarly research, for cost estimating, and for creating reliable benchmarks. In fact, function points are now used for more business purposes than any other metric in the history of software."

— Capers T. Jones. *Foreword to IFPUG Functional Size Measurement Method.* ISO/IEC 20926:2009, 2009

IFPUG Procedure



Source: International Stardardization Organization ISO. ISO/IEC 20926:2009, IFPUG Functional Size Measurement Method, 2009



"The application boundary indicates the border between the software being measured and the user."

Source: International Stardardization Organization ISO. ISO/IEC 20926:2009, IFPUG Functional Size Measurement Method, 2009

The 14 general system characteristics are:

1. Data Communications
2. Distributed Data Processing
3. Performance
4. Heavily Used Configuration
5. Transaction Rate
6. Online Data Entry
7. End-User Efficiency
8. Online Update
9. Complex Processing
10. Reusability
11. Installation Ease
12. Operational Ease
13. Multiple Sites
14. Facilitate Change

"The 14 general system characteristics are summarized into the value adjustment factor (VAF). When applied, the value adjustment factor adjusts the unadjusted function point count $\pm$35 percent to produce the adjusted function point count."

Source: International Stardardization Organization ISO. ISO/IEC 20926:2009, IFPUG Functional Size Measurement Method, 2009

Function Type	Functional Complexity		Complexity Totals		Function Type Totals
ILFs	4	Low	X 7 =	28	
	0	Average	X 10 =	0	
	0	High	X 15 =	0	
					28
EIFs	4	Low	X 5 =	20	
	0	Average	X 7 =	0	
	0	High	X 10 =	0	
					20
EIs	4	Low	X 3 =	12	
	2	Average	X 4 =	8	
	1	High	X 6 =	6	
					26
EOs	4	Low	X 4 =	16	
	2	Average	X 5 =	10	
	0	High	X 7 =	0	
					26
EQs	5	Low	X 3 =	15	
	0	Average	X 4 =	0	
	0	High	X 6 =	0	
					15
	Unadjusted Function Point Count				115

"The formula calculates the development project function points: `DFP = (UFP + CFP) * VAF`. Where UFP is the unadjusted function points for the functions that will be available after installation, and CFP is the unadjusted function points added by the conversion unadjusted function point count."

Source: International Stardardization Organization ISO. ISO/IEC 20926:2009, IFPUG Functional Size Measurement Method, 2009

"The function point metric, like LOC, is relatively controversial... Opponents claim that the method requires some 'sleight of hand' in that computation is based on subjective, rather than objective, data."

— Roger S. Pressman and Bruce Maxim. *Software Engineering: A Practitioner's Approach.* McGraw Hill, 2014

"Function Points and Function Counts (adjusted FPs) can be used as predictors of KSLOC. In particular, Function Counts correlated with KSLOC at the level of 75.1 percent, which is similar to the correlations published by Albrecht and Gaffney [1983], and is likely to be good enough to be of use to the software manager."

— Chris F. Kemerer. An Empirical Validation of Software Cost Estimation Models. *Communications of the ACM*, 30(5):416–429, 1987. doi:10.1145/22899.22906

"Function point counts appear to be a more consistent a *priori* measure of software size than source lines of code. As such it is recommended that function point estimates be used in preference to lines of code estimates as the measure of system size."

— Graham C. Low and D. Ross Jeffery. Function Points in the Estimation and Evaluation of the Software Process. *IEEE Transactions on Software Engineering*, 16(1):64–71, 1990. doi:10.1109/32.44364

Function Point Standards

- Mark-II — ISO/IEC 20968:2002
- IFPUG — ISO/IEC 20926:2009
- FiSMA — ISO/IEC 29881:2010
- COSMIC — ISO/IEC 19761:2011
- Nesma — ISO/IEC 24570:2018
- OMG — ISO/IEC 19515:2019

Some Other Function Points

- Early and easy function points
- Engineering function points
- Object-Oriented Function Points (OOFP)
- Weighted Micro Function Points
- Fuzzy Function Points

References

Allan J. Albrecht. Measuring Application Development Productivity. In *Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium*, pages 83–92, 1979.

Allan J. Albrecht and John E. Gaffney. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering*, (6): 639–648, 1983. doi:10.1109/tse.1983.235271.

Barry Boehm, Chris Abts, Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Steece Bert. *Software Cost Estimation With COCOMO II*. Englewood Cliffs, Prentice-Hall, 2000. doi:10.5555/1795822.

Barry W. Boehm. Software Engineering Economics. *IEEE Transactions on Software Engineering*, (1): 4–21, 1984. doi:10.1109/tse.1984.5010193.

Frederick P. Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering*. Pearson Education, 1995. doi:10.5555/540031.

Peter Pin-Shan Chen. The Entity-Relationship Model — Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976. doi:10.1145/320434.320440.

Tom DeMarco. *Structure Analysis and System Specification*. Prentice Hall, 1978. doi:10.1007/978-3-642-48354-7_9.

Daniel D. Galorath and Michael W. Evans. *Software Sizing, Estimation, and Risk Management: When Performance Is Measured Performance Improves*. CRC Press, 2006. doi:10.1201/9781420013122.

International Stardardization Organization ISO. ISO/IEC 20926:2009, IFPUG Functional Size Measurement Method, 2009.

Capers Jones. *Estimating Software Costs: Bringing Realism to Estimating*. McGraw-Hill, 2007.

Capers T. Jones. *Foreword to IFPUG Functional Size Measurement Method*. ISO/IEC 20926:2009, 2009.

Chris F. Kemerer. An Empirical Validation of

Software Cost Estimation Models. *Communications of the ACM*, 30(5):416–429, 1987. doi:10.1145/22899.22906.

Graham C. Low and D. Ross Jeffery. Function Points in the Estimation and Evaluation of the Software Process. *IEEE Transactions on Software Engineering*, 16(1):64–71, 1990. doi:10.1109/32.44364.

Roger S. Pressman and Bruce Maxim. *Software Engineering: A Practitioner's Approach*. McGraw Hill, 2014.

Lawrence H. Putnam. A General Empirical Solution to the Macro Software Sizing and Estimating Problem. *IEEE Transactions on Software Engineering*, (4):345–361, 1978. doi:10.1109/tse.1978.231521.

Charles R. Symons. *Software Sizing and Estimating: Mk II FPA (Function Point Analysis)*. John Wiley & Sons, Inc, 1991. doi:10.5555/120462.