Function Points

Yegor Bugayenko

Lecture #17 out of 24
80 minutes

"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."

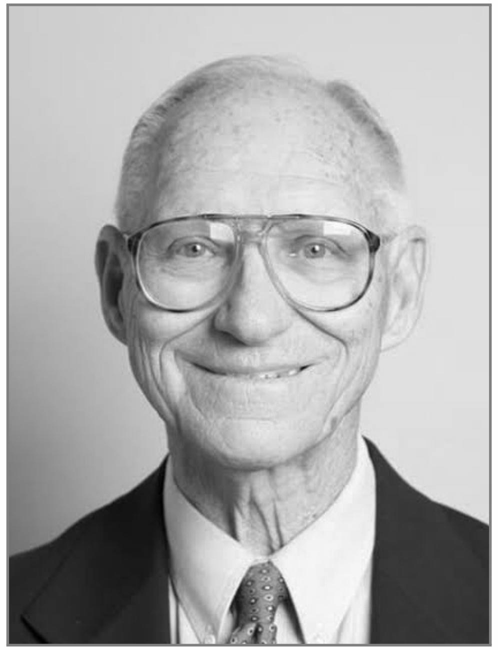
— Capers Jones. *Estimating Software Costs: Bringing Realism to Estimating.* McGraw-Hill, 2nd edition, 2007

"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

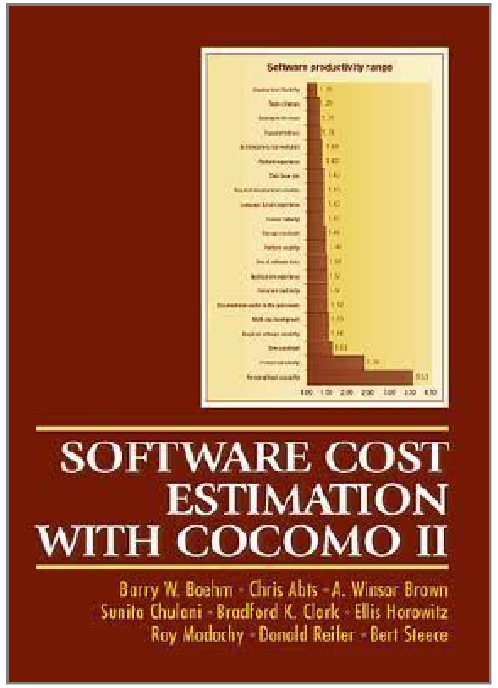
"In general, the size of the product in source statements is $S = C \times K^{1/3} \times t^{4/3}$, where C is a productivity constant, K is development effort, and t is time."

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



"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



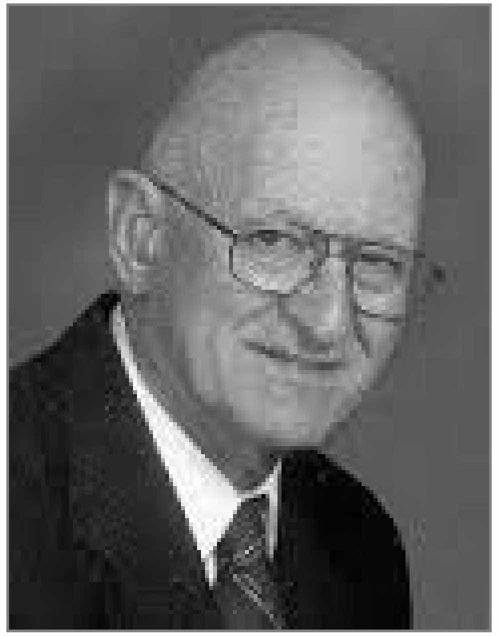
"Success in all types of organization depends increasingly on the development of customized software solutions, yet more than half of software projects now in the works will exceed both their schedules and their budgets by more than 50%."

— 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

Parametric Review of Information for Costing and Evaluation – Software (PRICE-S)

Software Evaluation and Estimation of Resources – Software Estimating Model (SEER-SEM).

Doty Model



"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

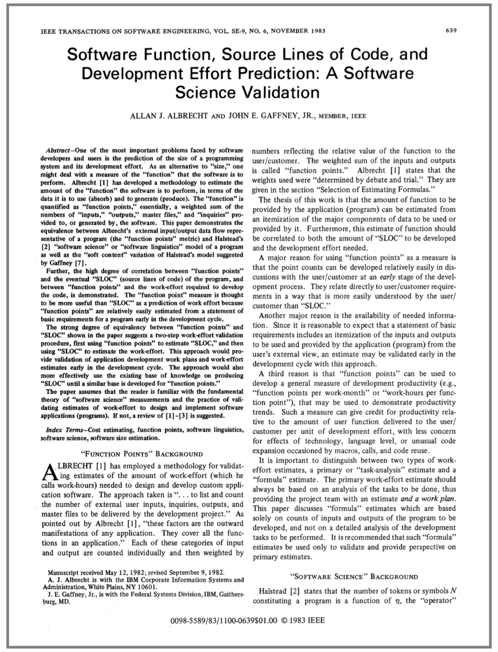
These counts are weighted by numbers designed to reflect the function value to the customer. The weights used were determined by debate and trial. These weights have given us good results:

- 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



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

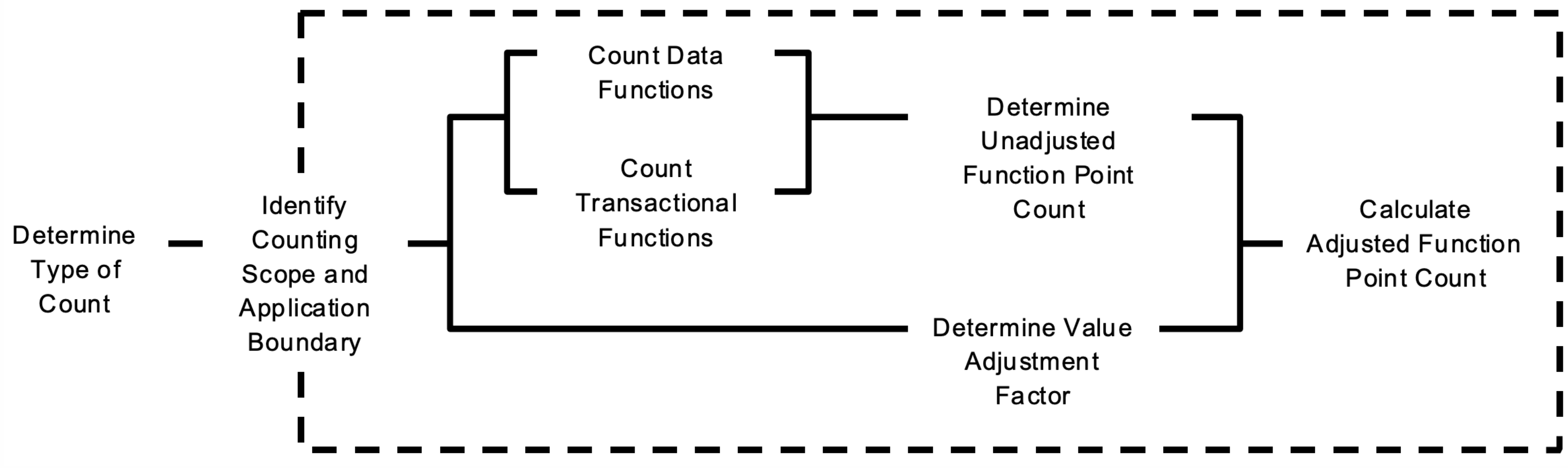
"At least for the applications analyzed, both the development work-hours and application size in "SLOC" are strong functions of "function points" and "input/output data item count." Further, it appears that basing applications development effort estimates on the amount of function to be provided by an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

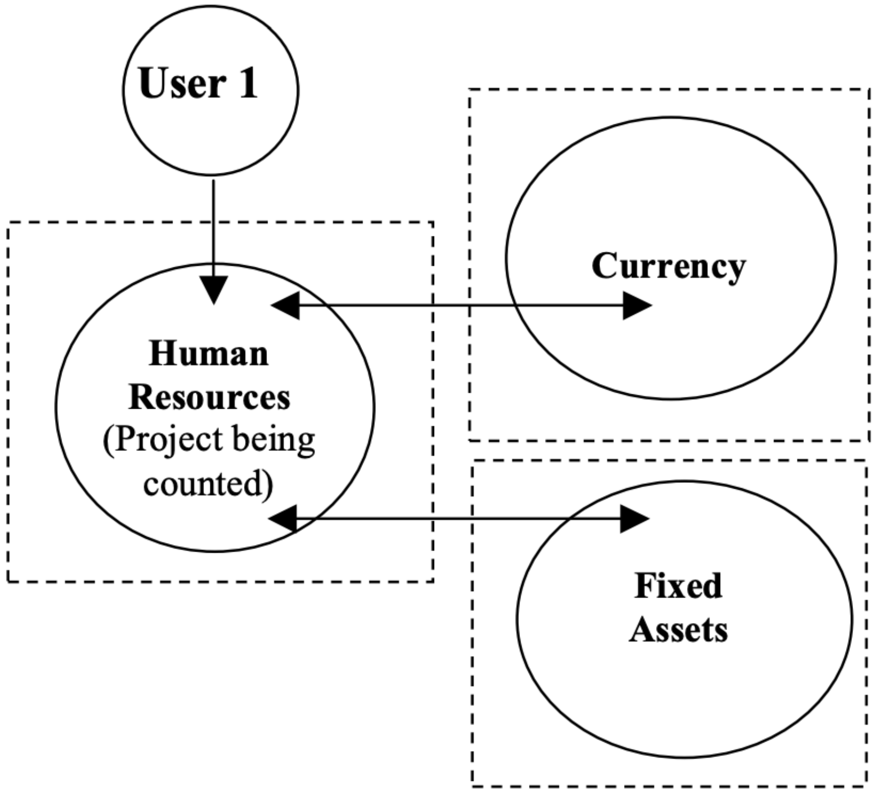
"The reliability of function point analysis is good enough to allow function points to serve as the basis for contracts, for carrying out scholarly research, for cost estimating, and for creating reliable benchmarks. In fact, function points are now used for more business purposes than any other metric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

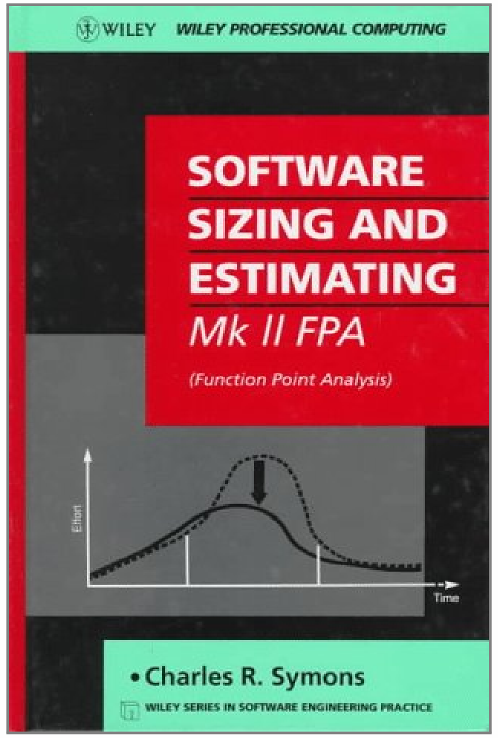
"The 14 general system characteristics are summarized into the value adjustment factor (VAF). When applied, the value adjustment factor adjusts the unadjusted function point count ±35 percent to produce the adjusted function point count."

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



"…"

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

Function Points

- Early and easy function points
- COSMIC Function Points
- Mk II Function Points
- Nesma Function Points
- FiSMA Function Points
- Engineering function points
- Object-Oriented Function Points (OOFP)

- Weighted Micro Function Points
- Fuzzy Function Points

Function Points may be measured by these tools:

- ...

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.

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.

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

Frederick P. Brooks Jr. *The Mythical Man-month: Essays on Software Engineering*. Pearson Education, 1995.

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, 2nd edition, 2007.

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

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

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