

# Function Points


YEGOR BUGAYENKO

Lecture #17 out of 24

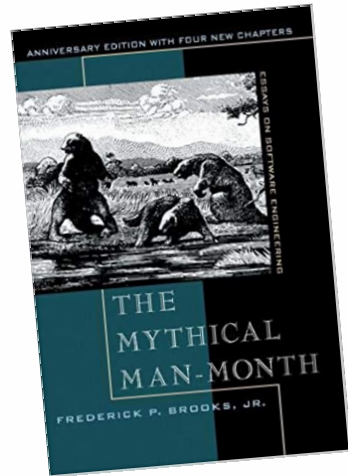
80 minutes

The slidedeck was presented by the author in this [YouTube Video](#)

All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as web sites.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

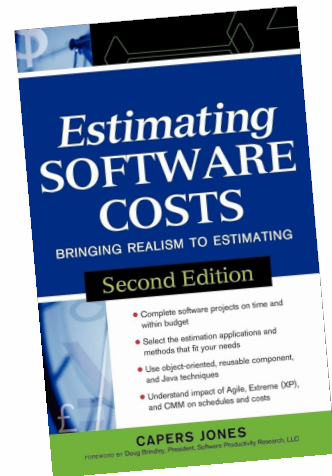


1. We don't know how to estimate the effort required to develop a software system.



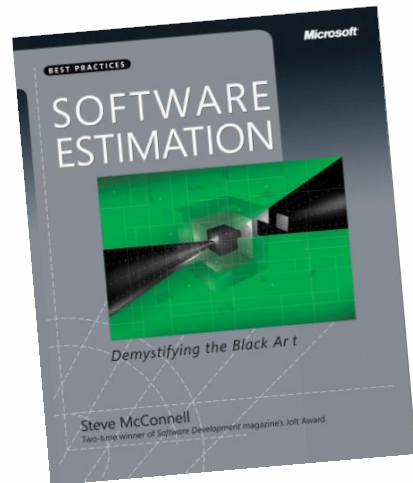
“Our techniques of estimating are poorly developed. More seriously, they reflect an unvoiced assumption which is quite untrue, i.e., that all will go well.”

— Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1978. doi:[10.5555/540031](https://doi.org/10.5555/540031)



“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.”

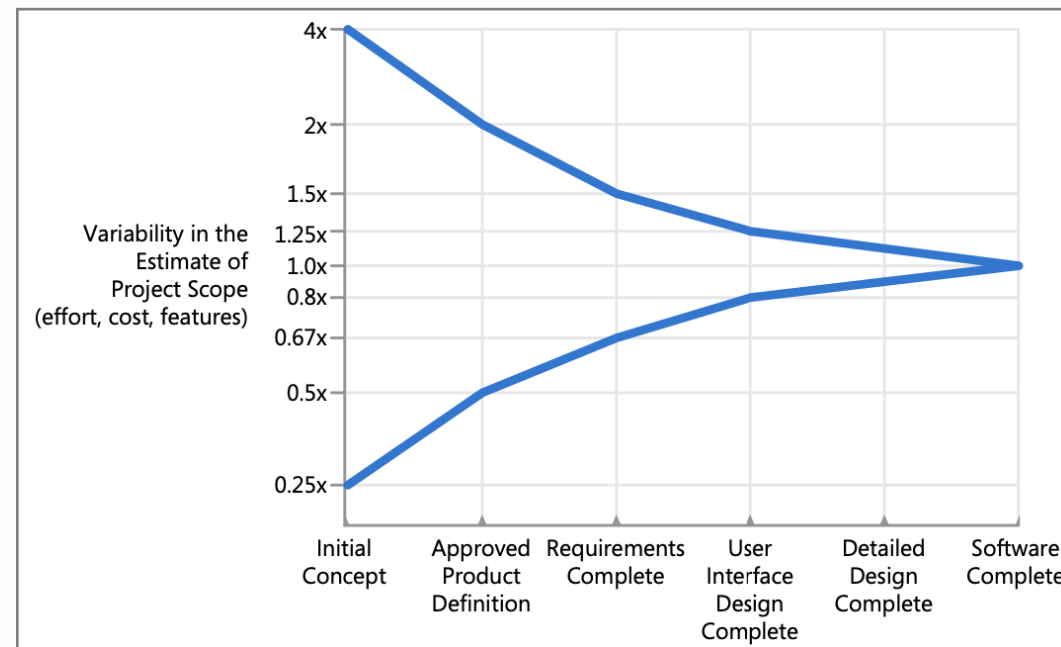
— Capers Jones. *Estimating Software Costs: Bringing Realism to Estimating*. McGraw-Hill, 2007



“The largest driver in a software estimate is the size of the software being built, because there is more variation in the size than in any other factor.”

— Steve McConnell. *Software Estimation: Demystifying the Black Art*. Microsoft Press, 2006. doi:[10.5555/1204642](https://doi.org/10.5555/1204642)

## Cone of Uncertainty



Source: Steve McConnell. *Software Estimation: Demystifying the Black Art*. Microsoft Press, 2006.  
doi:[10.5555/1204642](https://doi.org/10.5555/1204642)



2. However, we can speculate, with the help of function points.



“**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(4): 345–361, 1978. doi:[10.1109/tse.1978.231521](https://doi.org/10.1109/tse.1978.231521)





“**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, 1979

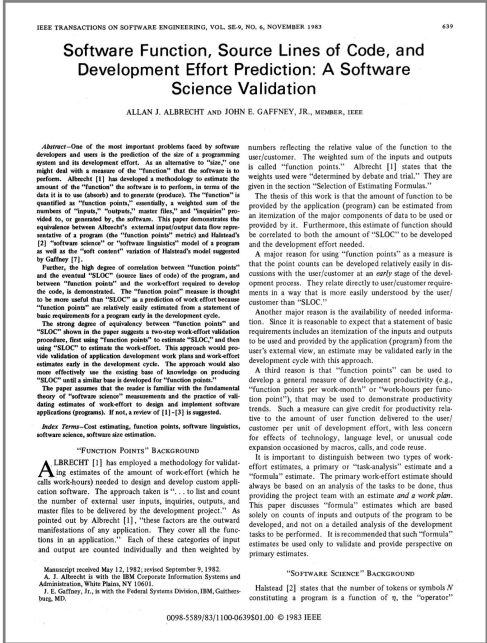
These counts are weighted by numbers designed to reflect the function value to the customer. The weights used were determined by debate and trial. These weights have given us good results:

- 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.

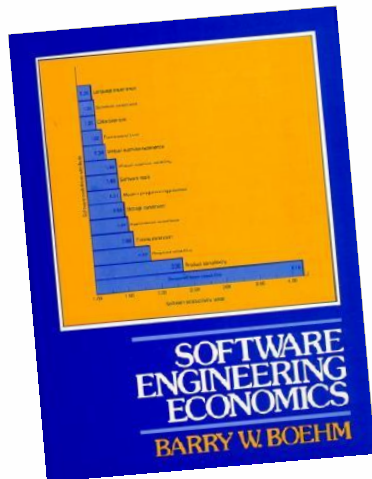
“If the inputs, outputs, or files are extra complicated, we add 5%. Complex internal processing can add another 5%. On-line functions and performance are addressed in other questions. The maximum adjustment possible is 50%, expressed as  $\pm 25\%$  so that the weighted summation is the average complexity.”

Source: Allan J. Albrecht. Measuring Application Development Productivity, 1979



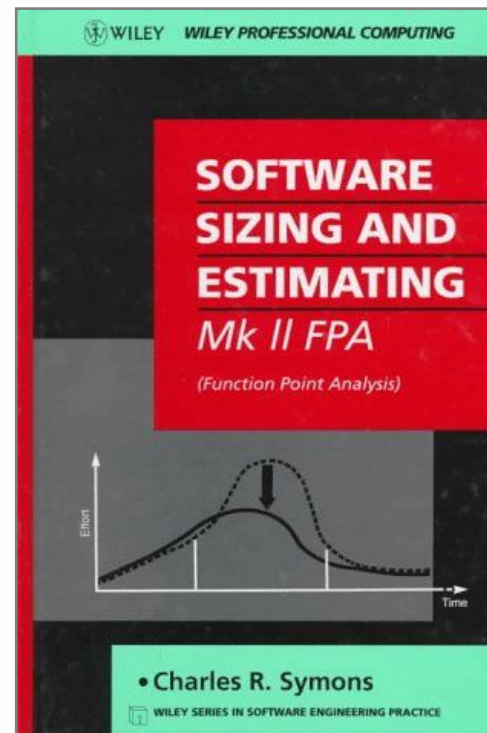
“At least for the applications analyzed, both the development work-hours and application size in “SLOC” are strong functions of “function points” and “input/output data item count.” Further, it appears that basing applications development effort estimates on the amount of function to be provided by an 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*, 9(6):639–648, 1983. doi:10.1109/tse.1983.235271



“**COCOMO (Constructive Cost Model)**: 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*, 10(1):4–21, 1984. doi:[10.1109/tse.1984.5010193](https://doi.org/10.1109/tse.1984.5010193)



“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)*. Wiley, 1991. doi:[10.5555/120462](https://doi.org/10.5555/120462)

The Functional Size (Function Point Index) is the weighted sum over all Logical Transactions, of the Input Data Element Types ( $N_i$ ), the Data Entity Types Referenced ( $N_e$ ), and the Output Data Element Types ( $N_o$ ).

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

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

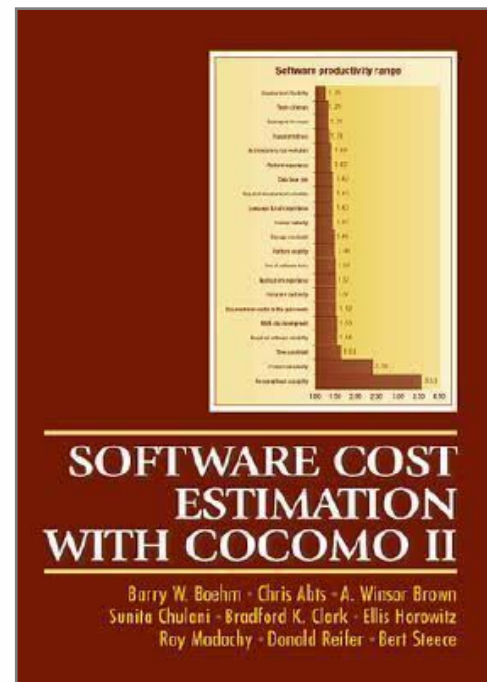
where ' $\Sigma$ ' means the sum over all Logical Transactions, and the industry average weights per Input Data Element Type, Data Entity Type Reference and Output Data Element Type are, respectively:

$$W_i = 0.58$$

$$W_e = 1.66$$

$$W_o = 0.26$$

Source: Charles R. Symons. *Software Sizing and Estimating: Mk II FPA (Function Point Analysis)*. Wiley, 1991.  
doi:[10.5555/120462](https://doi.org/10.5555/120462)



“**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*. Prentice Hall, 2000. doi:[10.5555/1795822](https://doi.org/10.5555/1795822)

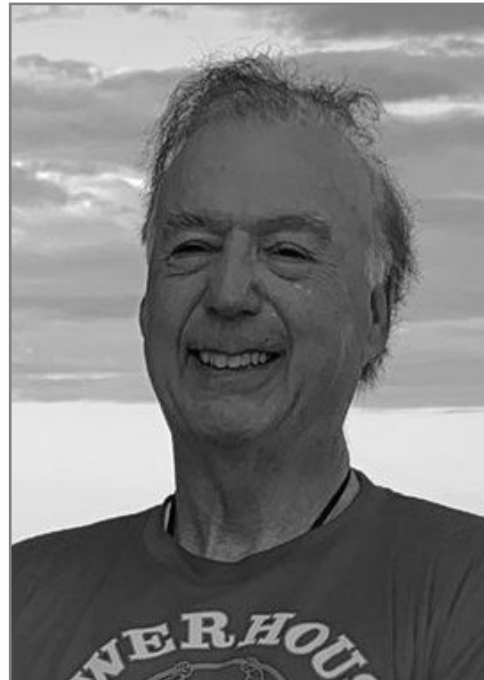
**Table 6.7 Function Complexity Table**

<i>Function Type</i>	<i>Low</i>	<i>Average</i>	<i>High</i>
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](https://doi.org/10.1201/9781420013122)





“**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](https://doi.org/10.1201/9781420013122)

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

<i>Functions</i>	<i>IFPUG Compatible Mode</i>	<i>SEER-SEM Extended Mode</i>
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
<i>Note:</i> 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](https://doi.org/10.1201/9781420013122)

**Table 6.17 Conversion Ratios: Lines of Code per Function Point**

<i>Source Code Language</i>	<i>DCG Likely</i>	<i>Capers Jones</i>			<i>Galorath Likely</i>
		<i>Low</i>	<i>Mean</i>	<i>High</i>	
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](https://doi.org/10.1201/9781420013122)

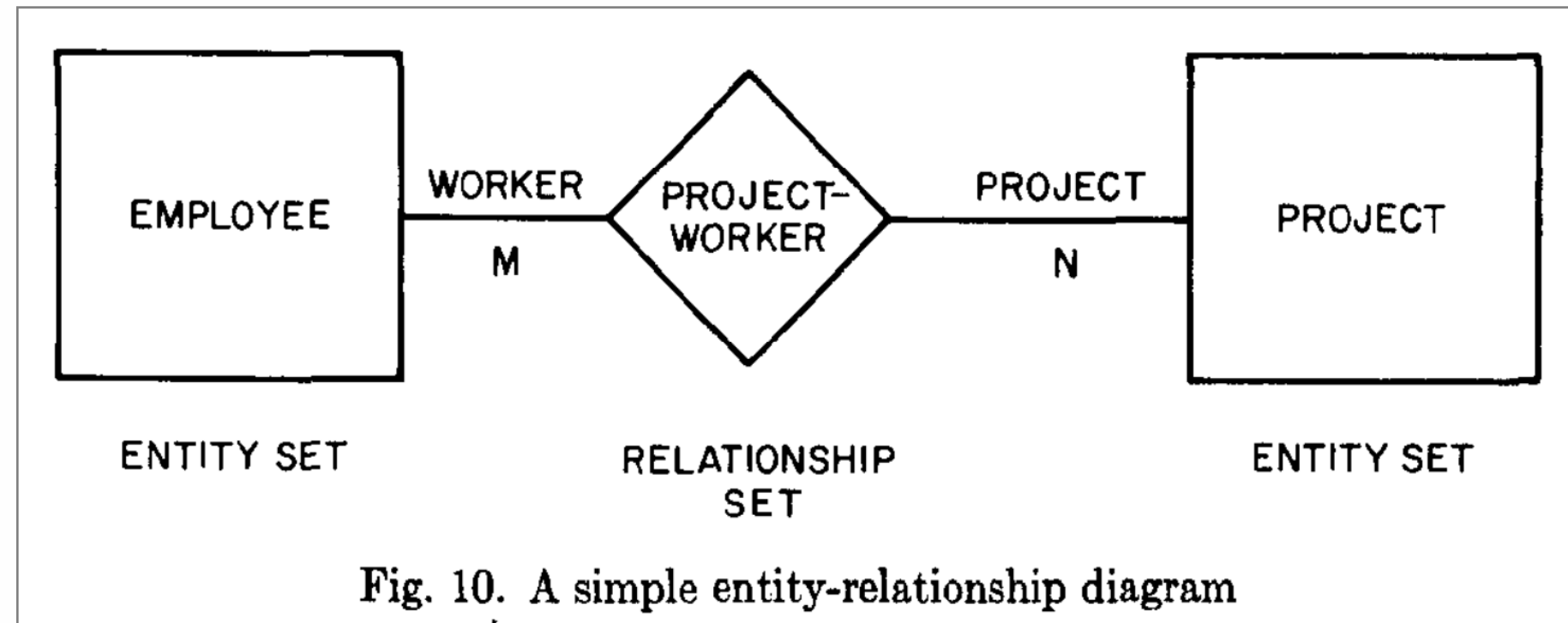


3. We can also use ER Model and Data Flow Diagrams.

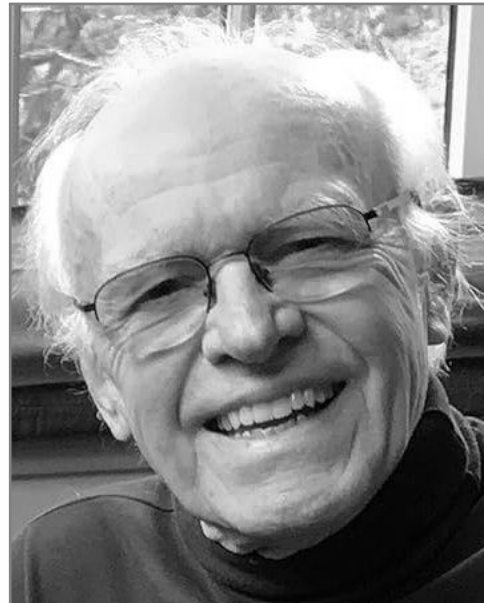


“The **Entity-Relationship** (ER) model adopts the more natural view that the real world consists of entities and relationships. It incorporates some of the important semantic information about the real world.”

— 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](https://doi.org/10.1145/320434.320440)

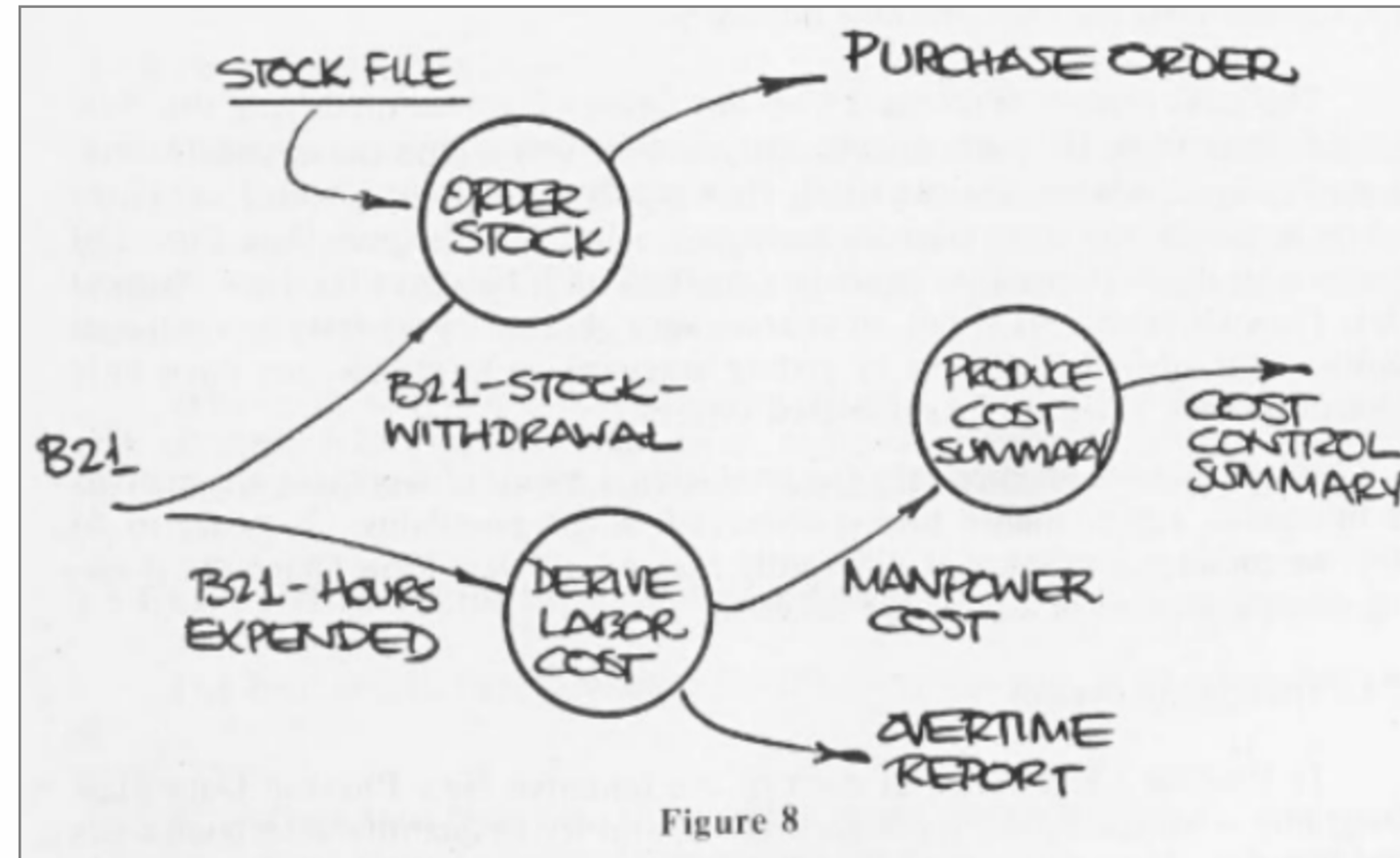


Source: Peter Pin-Shan Chen. The Entity-Relationship Model — Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976. doi:[10.1145/320434.320440](https://doi.org/10.1145/320434.320440)



“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](https://doi.org/10.1007/978-3-642-48354-7_9)



Source: Tom DeMarco. *Structure Analysis and System Specification*. Prentice Hall, 1978.  
 doi:[10.1007/978-3-642-48354-7\\_9](https://doi.org/10.1007/978-3-642-48354-7_9)





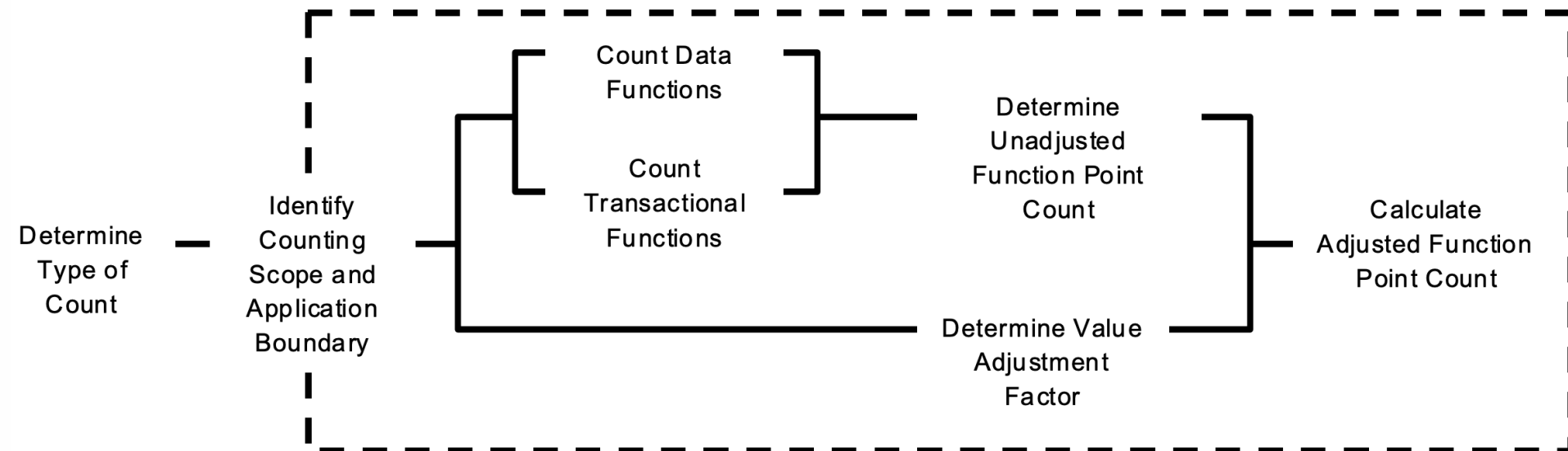
4. We can even standardize function points.



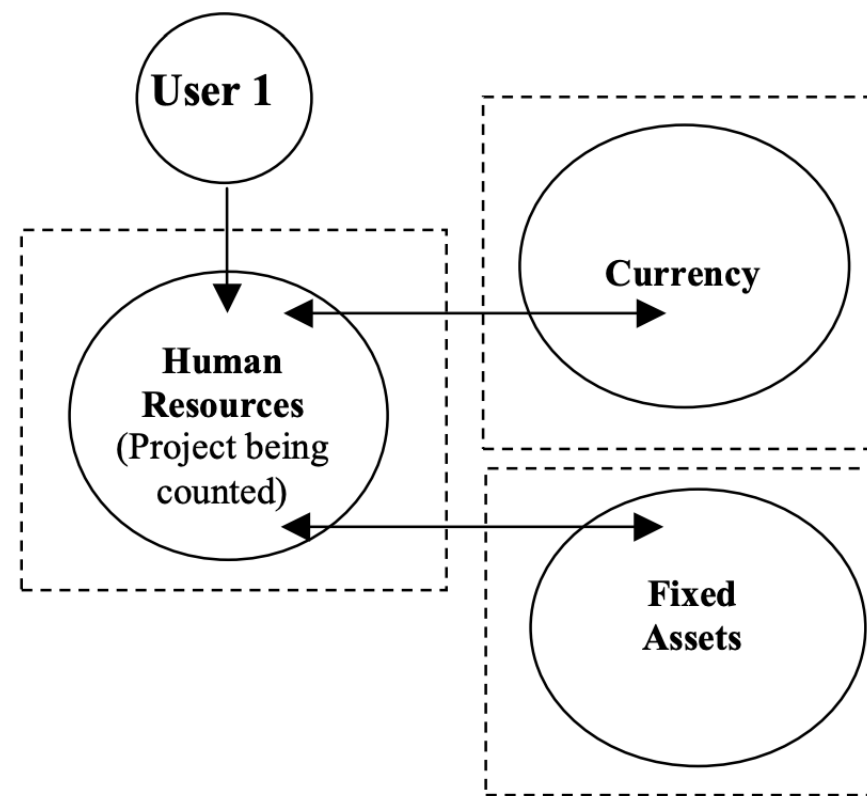
“The reliability of function point analysis is good enough to allow function points to serve as the basis for contracts, for carrying out scholarly research, for cost estimating, and for creating reliable benchmarks. In fact, function points are now used for more business purposes than any other metric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

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

## IFPUG Procedure



Source: International Standardization 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 Standardization 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 Standardization 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	28
	0	Average	X 10 =	0	
	0	High	X 15 =	0	
EIFs	4	Low	X 5 =	20	20
	0	Average	X 7 =	0	
	0	High	X 10 =	0	
EIs	4	Low	X 3 =	12	26
	2	Average	X 4 =	8	
	1	High	X 6 =	6	
EOs	4	Low	X 4 =	16	26
	2	Average	X 5 =	10	
	0	High	X 7 =	0	
EQs	5	Low	X 3 =	15	15
	0	Average	X 4 =	0	
	0	High	X 6 =	0	
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 Standardization Organization ISO. ISO/IEC 20926:2009, IFPUG Functional Size Measurement Method, 2009


## 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 (OOFPP)
- Weighted Micro Function Points
- Fuzzy Function Points





5. Apparently, a correlation exists between function points and some other metrics.



“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](https://doi.org/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](https://doi.org/10.1109/32.44364)



“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

# Bibliography

- Allan J. Albrecht. Measuring Application Development Productivity, 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*, 9(6):639–648, 1983. doi:[10.1109/tse.1983.235271](https://doi.org/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*. Prentice Hall, 2000. doi:[10.5555/1795822](https://doi.org/10.5555/1795822).
- Barry W. Boehm. Software Engineering Economics. *IEEE Transactions on Software Engineering*, 10(1):4–21, 1984. doi:[10.1109/tse.1984.5010193](https://doi.org/10.1109/tse.1984.5010193).
- Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1978. doi:[10.5555/540031](https://doi.org/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](https://doi.org/10.1145/320434.320440).
- Tom DeMarco. *Structure Analysis and System Specification*. Prentice Hall, 1978. doi:[10.1007/978-3-642-48354-7\\_9](https://doi.org/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](https://doi.org/10.1201/9781420013122).
- International Standardization 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](https://doi.org/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](https://doi.org/10.1109/32.44364).
- Steve McConnell. *Software Estimation: Demystifying the Black Art*. Microsoft Press, 2006. doi:[10.5555/1204642](https://doi.org/10.5555/1204642).
- 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(4):345–361, 1978. doi:[10.1109/tse.1978.231521](https://doi.org/10.1109/tse.1978.231521).
- Charles R. Symons. *Software Sizing and Estimating: Mk II FPA (Function Point Analysis)*. Wiley, 1991. doi:[10.5555/120462](https://doi.org/10.5555/120462).