The Origins of Function Point Measures

June 2016

Note of Thanks

We're pleased to share this month's Trusted Advisor, which was written by Capers Jones. Capers is a well-known author and speaker on topics related to software estimation. He is the co-founder, Vice President, and Chief Technology Officer of Namcook Analytics LLC, which builds patent-pending advanced risk, quality, and cost estimation tools.

Many thanks to Capers for participating in Trusted Advisor and allowing us to publish his report!

Scope of Report

The 30th anniversary of the International Function Point User's Group (IFPUG) is approaching. As such, this report addresses a brief history of the origin of function points. The author, Capers Jones, was working at IBM in the 1960's and 1970's, observing the origins of several IBM technologies, such as inspections, parametric estimation tools, and function point metrics. This report discusses the origins and evolution of function point metrics.

Introduction

In the 1960's and 1970's IBM was developing new programming languages, such as APL, PL/I, PL/S etc. IBM executives wanted to attract customers to these new languages by showing clients higher productivity rates.

As it happens, the compilers for various languages were identical in scope and had the same features. Some older compilers were coded in assembly language, while newer compilers were coded in PL/S, which was a new IBM language for systems software.

When we measured productivity of assembly-language compilers versus PL/S compilers using "lines of code (LOC)," we found that even though PL/S took less effort, LOC per month favored assembly language.

This problem is easiest to see when comparing products that are almost identical but merely coded in different languages. Compilers, of course, are very similar. Other products, besides compilers, that are close enough in feature sets to have their productivity negatively impacted by LOC metrics are PBX switches, ATM banking controls, insurance claims handling, and sorts.

A Better Metric

To show the value of higher-level languages, the first IBM approach was to convert high-level languages into "equivalent assembly language." In other words, we measured productivity against a synthetic size based on assembly language instead of against true LOC size in the actual higher level languages. This method was used by IBM from around 1968 through 1972.

An IBM Vice President, Ted Climis, said that IBM was investing a lot of money into new and better programming languages. Neither he nor clients could understand why we had to use the old assembly language as the metric to show productivity gains for new languages. This was counter-productive to the IBM strategy of moving customers

to better programming languages. He wanted a better metric that was language independent and could be used to show the value of all IBM high-level languages.

This led to the IBM investment in function point metrics and to the creation of a function-point development team under Al Albrecht at IBM White Plains.

He wanted a better metric that was language dependent and could be used to show the value of all IBM high-level languages.

The Origin of Function Points

Function point metrics were developed by the IBM team by around 1975 and used internally and successfully. In 1978, IBM placed function point metrics in the public domain and announced them via a technical paper given by Al Albrecht at a joint IBM/SHARE/Guide conference in Monterey, California.

Table 1 shows the underlying reason for the IBM function point invention based on the early comparison of assembly language and PL/S for IBM compilers.

Table 1 shows productivity in four separate flavors:

1. Actual lines of code in the true languages.
2. Productivity based on "equivalent assembly code."
3. Productivity based on "function points per month."
4. Productivity based on "work hours per function point."

Table 1: IBM Function Point Evolution Circa 1968-1975 (Results for two IBM compilers)			
	Assembly Language		**PL/S Language**
Lines of code (LOC)	17,500.00		5,000.00
Months of effort	30.00		12.50
Hours of effort	3,960.00		1,650.00
LOC per month	583.33		400.00
Equivalent assembly	17,500.00		17,500.00
Equiv. Assembly MO	583.33		1,400.00
Function points	100.00		100.00
Function points/month	3.33		8.00
Work hours per function point	39.60		16.50

The creation and evolution of function point metrics was based on a need to show IBM clients the value of IBM's emerging family of high-level programming languages, such as PL/I, APL, and others. This is still a valuable use of function points, since there are more than 2,500 programming languages in 2016 and new languages are being created at a rate of more than one per month.

Another advantage of function point metrics vis a vis LOC metrics is that function points can measure the productivity of non-coding tasks, such as creation of requirements and design documents. In fact, function points can measure all software activities, while LOC can only measure coding.

Up until the explosion of higher-level programming languages occurred, assembly language was the only language used for systems software (the author programmed in assembly for several years when starting out as a young programmer).

With only one programming language, LOC metrics worked reasonably well. It was only when higher-level programming languages appeared that the LOC problems became apparent. It was soon realized that the essential problem with the LOC metric is really nothing more than a basic issue of manufacturing economics that had been understood by other industries for over 200 years.

This is a fundamental law of manufacturing economics: "*When a manufacturing process has a high percentage of fixed costs and there is a decline in the number of units produced, the cost per unit will go up*."

The software non-coding work of requirements, design, and documentation acts like fixed costs. When there is a move from a low-level language, such as assembly, to a higher-level language, such as PL/S, the cost-per-unit will go up, assuming that LOC is the "unit" selected for measuring the product. This is because of the fixed costs of the non-code work and the reduction of code "units" for higher-level programming languages.

Function point metrics are not based on code at all, but are an abstract metric that defines the essence of the features that the software provides to users. This means that applications with the same feature sets will be the same size in terms of function points no matter what languages they are coded in. Productivity and quality can go up and down, of course, but they change in response to team skills.

The Expansion of Function Points

Function points were released by IBM in 1978 and other companies began to use them, and soon the International Function Point User's Group (IFPUG) was formed in Canada.

Today, in 2016, there are hundreds of thousands of function point users and hundreds of thousands of benchmarks based on function points. There are also several other varieties of function points, such as COSMIC, FISMA, NESMA, etc.

Overall, function points have proven to be a successful metric and are now widely used for productivity studies, quality studies, and economic analysis of software trends. Function point metrics are supported by parametric estimation tools and also by benchmark studies. There are several flavors of automatic function point tools. There are function point associations in most industrialized countries. There are also ISO standards for functional size measurement.

(There was never an ISO standard for code counting, and counting methods vary widely from company to company and project to project. In a benchmark study performed for a "LOC" shop, we found four sets of counting rules for LOC that varied by over 500%).

Table 2 shows countries with increasing function point usage circa 2016, and it also shows the countries where function point metrics are now required for government software projects.

TABLE 2: COUNTRIES EXPANDING USE OF FUNCTION POINTS 2016

1	Argentina	
2	Australia	
3	Belgium	
4	**Brazil**	**Required for government contracts 2008**
5	Canada	
6	China	
7	Finland	
8	France	
9	Germany	
10	India	
11	**Italy**	**Required for government contracts**
12	**Japan**	**Required for government contracts**
13	**Malaysia**	**Required for government contracts**
14	Mexico	
15	Norway	
16	Peru	
17	Poland	
18	Singapore	
19	**South Korea**	**Required for government contracts**
20	Spain	
21	Switzerland	
22	Taiwan	
23	The Netherlands	
24	United Kingdom	
25	United States	

Several other countries will probably also mandate function points for government software contracts by 2017. Eventually most countries will do this.

In retrospect, function point metrics have proven to be a powerful tool for software economic and quality analysis.

Trusted Advisor Guest Author

Again, our sincere thanks are extended to our guest author, Capers Jones.