Function Points -- A Productivity Measure Benefits APL

Kevin R. Weaver
Consultant
Avery Road, Box 319
Garrison, NY 10524

As APL users and developers we all know that APL is a productivity language. We've known this since the 1960s. The original marketing literature promoting APL shouted to the computing world that APL was akin to Superman: faster (to program) than a speeding bullet, more powerful (for applications) than a locomotive, and able to leap tall buildings (of systems) in a single bound. When STSC and IPSA began marketing APL in 1969, the claim was that APL was "10 times more powerful" than other traditional high level languages. Hundreds and even thousands around the world were convinced of the potential and climbed on the bandwagon. Most of us on this bandwagon have continued to promote the Superman-like qualities of APL. We slowly but surely convince/convert a few more each year. More often than not, however, we leave behind hundreds wondering who dazzled them and how it was done. Worse than that, we leave behind tens of thousands who simply don't believe in miracles, never aspire to Superman heights, or who feel that Superman is only a fantasy.These tens of thousands never seem to give up hope that a real Superman will eventually come along and save the computing world. They continue to search, and the APL community continues to hide or be ignored. The searching uncovers new easy-to-use and easy-to-learn software, Nth-generation applications and/or languages, problem/solution specific application software, and always new methods for evaluating, justifying, and proving that Superman has been uncovered. The APL community manages to bypass these searches, quietly goes about its business, saves a few souls from time to time, and sometimes gets a mention in the Daily Planet, page 2.

The time has come (or is well overdue) for the APL community to *join* Lois Lane and the computing professionals in the search for Superman. If we *join* this effort rather than ignore it or run head-long against it, we all will win. We need to take time out from writing new prime number generators, postpone the next implementation of quad-ding-dong, stop bending steel with our *bare* hands, and be measured for the Superman suit. Too many pretenders have taken credit where credit isn't due.

The Superman suit is already made-to-measure. What have the computing professionals accepted as their measure of productivity? For nearly 10 years, Function Points have been measuring productivity for major corporations, including IBM. Function Points have been measuring application development, development groups, and languages. And, almost without exception, APL has been ignored in this effort -- probably because the APL community has been hiding (for what reasons?) from the measurement tools. Let's join the measurement effort and see how we stack up against the big players. Fighting application by application only wins small battles. We need to win the war. Sure, we're a different sort of folk (haven't you heard or said that yourself?), proud of our accomplishments, convinced that our productivity method is or should be obvious to the common person, and we may be a real exception to this measurement business anyway. We're in a different domain. Right, Domain Error. Function Point Analysis may be our ticket to legitimately joining the data processing community without being accused of printing the tickets. They have accepted this measurement technique; we should be willing to be treated like all the mortals, take our lumps, if necessary, but expose ourselves to this careful evaluation and scrutiny.

Function Points were developed in 1979 by A.J. Albrecht of IBM. While they're an "abstract metric similar in concept to the Dow Jones industrial average," they are becoming a standard measurement tool adopted by hundreds of major corporations world-wide. The International Function Point Users Group lists in its membership hundreds of major corporations from various industries:

Manufacturing: IBM, GE, GTE, DuPont, Amoco, Exxon, 3M, Nissan, Xerox

Insurance: CNA, USAA, Mutual of Omaha, Travellers, Blue Cross, Aetna

Utilities: Pacific Bell, AT&T, Bell Canada, Ontario Hydro, Ohio Edison

Banks: Royal Bank of Canada, Harris Bank, First National Bank of Chicago, Chemical Bank, Manufacturers Hanover, Marine Midland.

Transportation: American Airlines, Air Canada, Quantas, Canadian National Railway.

Function Points are made up of a weighted combination of the number of inputs, outputs, inquiries, logical data files, and interfaces associated with an application. These are characteristics which provide functionality in an application for the USER. To measure a program, the developer counts these items and multiplies the total by weighting factors that adjust for complexity. The adjusted function point then becomes the unit of measurement for that application. Since function points are independent of the language used to implement the application, it is free of the paradox associated with counting lines of code as a productivity measure. Function points also consider the efficiency of operation versus development. And, as I noted above, function points consider and measure the functionality delivered to the USER. Isn't that what it's all about?

The first step in function point analysis is to measure system complexity by examining the following application characteristics:

1. **Data Communications:** to what extent are communications facilities used?

2. **Distributed Functions:** does the application prepare data for end-user processing on another component of the system?

3. **Performance:** do application performance objectives influence the design, development, installation, and support of the system?

4. **Heavily Used Configuration:** are special design considerations a characteristic of this application -- e.g., does the application run on a heavily used system?

5. **Transaction Rate:** is it high; does it influence the design, development, installation and support of the application?

6. **On-Line Data Entry:** what percent of the application?

7. **Design for End-User Efficiency:** was this considered in the initial planning and design of the system because of a user requirement?

8. **On-Line Update:** is volume high or low; recovery easy or difficult or not a concern?

9. **Complex Processing:** logical or math processes?

10. **Usable in Other Applications:** was the code developed specifically to be used in future applications?

11. **Installation Ease:** were tools provided for and tested during the system test phase?

12. **Operation Ease:** effective start-up, back-up, recovery procedures; minimal need for manual activities?

13. **Multiple Sites:** will the application be used by multiple users from different locations?

14. **Facilitate Change:** the application has been specifically designed, developed, and supported to facilitate change such as flexible query capability, business control data grouped in tables maintained by the user, etc. Is this a default characteristic of development in APL?

Once these points are evaluated, the video screens are evaluated for the level of information processing by each of five components:

1. External Inputs (i.e., the user)
2. External Output (what does the user see on the screen)
3. External Inquiry (what questions are asked)
4. Logical Internal Files
5. External Interface Files (other applications)

Simple weights are given all of the above, trivial computation is performed, and a function point count is the result. Other levels of complexity can and are factored into the application, but most of this detail is the subject of a full course on counting function points (an introductory course takes a day). One of the reported major benefits from using function points is the ability to easily estimate software development effort, i.e., productivity rate. Over 150 businesses use function points this way. Simply stated, one merely finds the ratio of function points to working months for the development of an application. There are a number of published articles and references describing how to count function points (see bibliography). Therefore, I will only address statistics and results rather than go into more detail than I have about specifically counting the function points in an application. Let's examine some of the industry information, and try to put APL into the picture.

The International Function Point Users Group measured 375 development projects from many different companies. None of the projects was developed in APL. They found a wide variation in the resulting rates partly due to large differences in counting the effort actually spent on each project. The median of their rates for large projects was 8 function points (FPs) per work month (WM). For small projects of 10 WMs or less, it was 26 FPs/WM.

Charles Behrens, a consultant and researcher, found that productivity rates vary also by the size of the system being developed. He analyzed 26 business systems developed in various languages (again, none written in APL). Behrens concluded that productivity rates varied not only by the system size but also that the language used and the development environment were major determinants of the rate. His findings are shown in Figure 1.

Figure 1

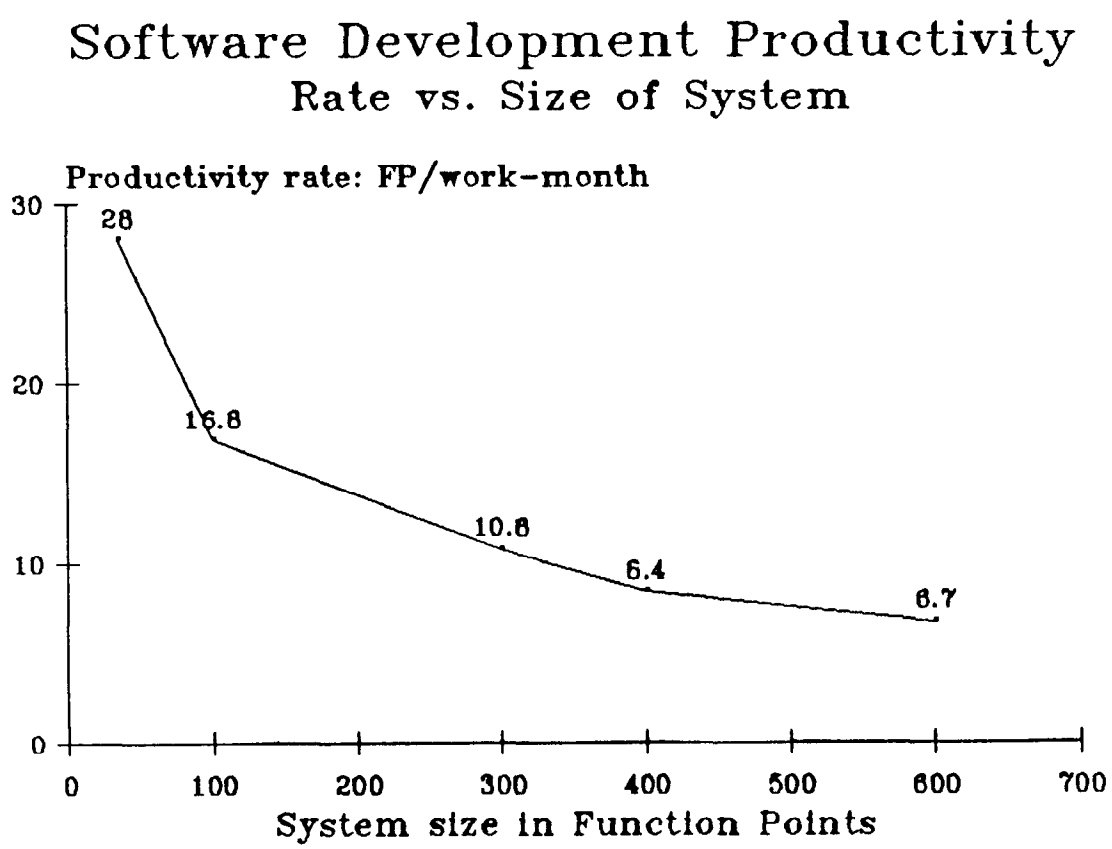
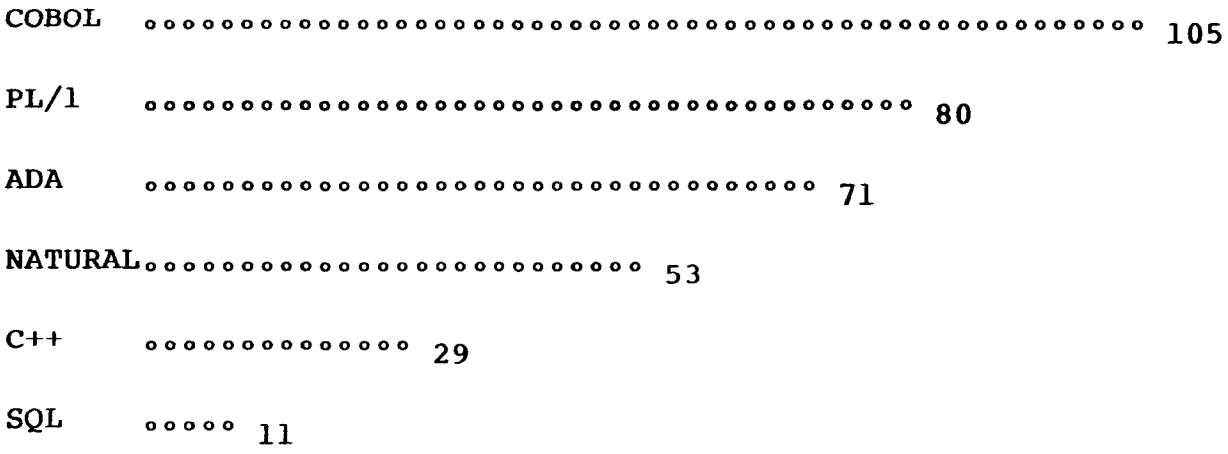


Figure 2

NON-COMMENTARY SOURCE CODE STATEMENTS

PER FUNCTION POINT



GE's Medical Systems measured five recent development projects. Some were developed using IBM's DMS Application Generator for CICS IBM applications and some were batch COBOL. Both used TSO ISPF as a major development tool. Their productivity rates varied between 27 and 47 FP/WM.

With these results in mind, let's now look at two APL-based applications and the function point analysis. First, STSC reported at its October 1988 Client Conference a function point delivery rate by their consulting group of approximately 300 FP/WM. Second, GE's financial reporting and consolidation system, LEX, is an APL-based, multi-site, multi-user system. The development and support team for this application has a productivity rate of 450 FP/WM.

Cost of development (per function point) might also be a serious consideration. One of the largest applications at GE is CPARS, their corporate payroll system. Development cost has been estimated between $18-25 million. The CPARS function point is roughly 102,000. LEX is estimated at costing $3 million to develop, and has a function point count of 60,000. CPARS development cost per function point is approximately $200, while the LEX cost per function point is approximately $50.

Capers Jones, chairman of Software Productivity Research, Inc. in Cambridge, MA, has taken yet another look at function points in relationship to the productivity of programming languages. He has observed that languages have varying but characteristic levels. He defines "level" as the average number of statements required to implement one function point. While this form of research is new and the findings are preliminary, the results are being considered by many in the industry. He reports that COBOL seemed to require about 105 noncommentary source code statements to implement one FP. PL/1 seemed to require approximately 80. Others are noted in Figure 2. Jones ranked APL at 10. Feeling this APL ranking was worthy of more careful examination, I examined some of the components of the LEX application at GE. The system administrator module required about 4 non-commentary code statements, the PC version of LEX required 3, and the LEX reporting module required 2. The LEX reporting module also contains the code for the LEX input module. When combined these two components of the system contain approximately 7000 lines of code and have been counted with 5000 function points, or 1.4 code statements per FP.

What does this mean for the APL community? Didn't we believe this all along? Now, using an accepted industry metric, we have an opportunity to re-introduce APL as a productivity tool. Rather than changing into our Superman outfit in the phone booth, we can now approach the MIS and traditional data processing groups on their terms, mere mortals. We may not fully believe in this measurement approach, but it's the best tool available today and the only one that is widely accepted. It might take some of us longer to count the function points in our application than it does to write the application, but the end result will accomplish the Superman feat. I encourage APLers to learn more about function points, and have this technique play a role in evaluating your APL application development for users versus the development of an application in another language or software product. We can become mild-mannered, stand up against the alternatives facing our users, and APL will take over as we always expected it would.

BIBLIOGRAPHY

1. *Function Point Mathematics* -- GUIDE 68, Session No. MP-5000E, Speaker: Allan Albrecht, IBM Corporation, July 20, 1987.

2. *Proving Productivity - The Fun in Function Points*, William Hufschmidt, Development Support Center, Inc. 16435 Tia Court, Brookfield, WI 53005

3. *Computing Technology NEWS* -- Corporate Information Technology, GE Corporation.

4. **Counting Practices Manual**, International Function Point Users Group, Release 2.0, April 4, 1988.

5. Albrecht, A.J., *Measuring Application Development Productivity*, **Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium**, pp 83-92, October 1979.

6. Jones, T. Capers, *Productivity Gauge Changing*, **Computerworld**, November 9, 1984.

7. Behrens, C.A., *Measuring the Productivity of Computer Systems Development Activities with Function Points*, **IEEE Transactions on Software Engineering**, Vol. SE-9 No. 6, Introduction