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.

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.

"SOFTWARE SCIENCE" BACKGROUND

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

TABLE I
DP SERVICE PROJECT DATA

Custom Application Number	Language	Input/Output Element Counts IN	OUT	FILE	INQ	Function Points	Source Lines of Code (SLOC)	Work-Hours
1	COBOL	25	150	60	75	1750	130K	102.4K
2	COBOL	193	98	36	70	1902	318K	105.2K
3	COBOL	70	27	12	--	428	20K	11.1K
4	PL/I	40	60	12	20	759	54K	21.1K
5	COBOL	10	69	9	1	431	62K	28.8K
6	COBOL	13	19	23	--	283	28K	10.0K
7	COBOL	34	14	5	--	205	35K	8.0K
8	COBOL	17	17	5	15	289	30K	4.9K
9	COBOL	45	64	16	14	680	48K	12.9K
10	COBOL	40	60	15	20	794	93K	19.0K
11	COBOL	41	27	5	29	512	57K	10.8K
12	COBOL	33	17	5	8	224	22K	2.9K
13	COBOL	28	41	11	16	417	24K	7.5K
14	PL/I	43	40	35	20	682	42K	12.0K
15	COBOL	7	12	8	13	209	40K	4.1K
16	COBOL	28	38	9	24	512	96K	15.8K
17	PL/I	42	57	5	12	606	40K	18.3K
18	COBOL	27	20	6	24	400	52K	8.9K
19	COBOL	48	66	50	13	1235	94K	38.1K
20	PL/I	69	112	39	21	1572	110K	61.2K
21	COBOL	25	28	22	4	500	15K	3.6K
22	DMS	61	68	11	--	694	24K	11.8K
23	DMS	15	15	3	6	199	3K	0.5K
24	COBOL	12	15	15	--	260	29K	6.1K

Note: (1) "IN" = No. of inputs; "OUT" = No. of outputs; "FILE" = No. of master files; "INQ" = No. of inquires

vocabulary size, and η, the "operand" vocabulary size. His software length equation is

$$N = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2.$$

This formula was originally derived to apply to a small program (or one procedure of a large program) or function, that is, to apply to the program expression of an algorithm. Thus, the number of tokens in a program consisting of a multiplicity of functions or procedures is best found by applying the size equation to each function or procedure individually, and summing the results.

Gaffney [3] has applied the software length equation to a single address machine in the following way. A program consists of data plus instructions. A sequence of instructions can be thought of as a string of "tokens." At the machine code level, for a single address machine, "op. code" tokens, generally alternate with "data label" tokens. Exceptions do occur due to instructions that require no data labels. The "op. codes" may be referred to as "operators" and the "data labels" as "operands." Thus, in an instruction of the form "LA X", meaning, load accumulator with the content of location X, "LA" is the operator, and "X" is the operand. For single address machine level code, the case Gaffney [3] analyzes, one would expect to have approximately twice as many tokens (N) as instructions (I). That is, $I = 0.5N$. Gaffney [3] applied the Halstead software length equation to object code for the AN/UYK-7 military computer (used in the Trident missile submarine's sonar system, as well as other applications). He determined a value for the coefficient "b" in the equation "$I = bN$." It was $b = 0.478$, and the correlation between the estimate $I = 0.478N$ (where the estimate $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$), and the actual instruction count I, was 0.916. Thus, the data correlated closely with the estimate from the software length equation.

Gaffney's work presumed that the number of unique instruction types (η_1), or operator vocabulary size employed, as well as the number of unique data labels (η_2) or operand vocabulary size used, was known. However, η_1 need not be known in order for one to estimate the number of tokens N or the number of instructions I. An "average" figure for η_1 (and thus for $\eta_1 \log_2 \eta_1$) can be employed, or the factor $\eta_1 \log_2 \eta_1$ can be omitted, inducing some degree of error, of course. Indeed, Christiansen *et al.* [4] have observed that "... program size is determined by the data that must be processed by the program." Thus one could take several different approaches to estimating software (code) size. The data label vocabulary size (η_2) could be estimated. Alternatively, it is suggested that (η_2^*), the number of conceptually unique inputs and outputs, can be used as a surrogate for (η_2). The estimate for (η_2^*) should be relatively easy to determine early in the design cycle, from the itemization of external inputs and outputs found in a complete requirements definition or external system design.

Some data by Dekerf [5] supports the idea that I (and N) can be estimated as multiples of the variates $\eta_2 \log_2 \eta_2$ and $\eta_2^* \log_2 \eta_2^*$ (for example: $I = A\ \eta_2 \log_2 \eta_2$, where ($A$) is some constant). Dekerf counted tokens (N), operands (η_2), and conceptually unique inputs and outputs (η_2^*) in 29 APL programs found in a book [6] by Allen of the IBM System Science Institute in Los Angeles. Using Dekerf's data, we have found that the sample correlation between N and $\eta_2^* \log_2 \eta_2^*$ is 0.918, and between N and $\eta_2 \log_2 \eta_2$ is 0.988.

In the next sections, we demonstrate that these (and other) software science formulas originally developed for (small) algorithms only can be applied to large applications (programs), where (η_2^*) is then interpreted to mean the sum of overall external inputs and outputs to the application (program). "Function points" [1] can be interpreted as a weighted sum of the top level input/output items (e.g., screens, reports, files) that are equivalent to (η_2^*). Also, as is shown subsequently, a number of variates based on "function points" can be used as the measure of the function that the application (program) is to provide.

TABLE II
ESTIMATING VARIATES EXPLORED

Independent Variate	Formula Basis	Dependent Variable Explored: Source Lines of Code: PL/I	Source Lines of Code: COBOL	Work-Hours
1. Function Points	F	X	X	X
2. Function Sort Content	$(F/2)\log_2(F/2)$	X		X
3. Function Potential Volume	$(F+2)\log_2(F+2)$	X		
4. Function Information Content	$F\log_2 F$	X		X
5. I/O Count	V			X
6. Sort Count	$(V/2)\log_2(V/2)$			X
7. Count Information Content	$V\log_2 V$			X
8. Source Lines of COBOL	SLOC			X
9. Source Lines of PL/I	SLOC			X

DP SERVICES DATA

Table I provides data on 24 applications developed by the IBM DP Services organization. The language used in each application is cited. The counts of four types of external input/output elements for the application as a whole are given. The number of "function points" for each program is identified. The number of "SLOC," including comments (all the "SLOC" was new) that implemented the function required is identified. Finally, the number of work-hours required to design, develop, and test the application is given.

SELECTION OF ESTIMATING FORMULAS

Using the DP Services data shown in Table I, 13 estimating formulas were explored as functions of the 9 variates listed in Table II. A basis for their selection is now provided. "Function points" count (F) is the variable determined using Albrecht's methodology [1]. Albrecht uses the following average weights to determine "function points": number of inputs X 4; number of outputs X 5; number of inquiries X 4; number of master files X 10. Interfaces are considered to be master files. As stated in [1], the weighted sum of inputs, outputs, inquiries, and master files can be adjusted within a range of +/- 25 percent, depending upon the estimator's assessment of the complexity of the program. As an example of the calculation of the number of "function points," consider the data for "custom application one" in Table I. The number of function points is equal to

$$F = (25 \times 4) + (150 \times 5) + (75 \times 4) + (60 \times 10) = 1750.$$

The current "Function Points Index Worksheet" being used in the IBM I/S organization is shown in Appendix A. The major changes from [1] are as follows.

1) Interfaces are separately identified and counted.

2) Provision is made for above average and below average complexities of the elements counted.

3) A more objective measure of processing complexity is provided.

"I/O count" (V) is the total program input/output count without the weights and processing complexity adjustment applied in "function points." Both function points and I/O count are treated as equivalent to Halstead's η_2^*, the unique input/output (data element) count. The "potential volume" formula $(F+2)\log_2(F+2)$ was developed by Halstead [2]. The "information content" formula ($F\log_2 F$), also used as a variate here, corresponds to $\eta_2^*\log_2\eta_2^*$, an approximation to the factor $\eta_2\log_2\eta_2$ in Halstead's software length equation.

The origin of the "sort count" variate $(F/2)\log_2(F/2)$ is as follows. Gaffney [7] estimated the number of conditional jumps in a program to be

$$J = (\eta_2^*/2)\log_2(\eta_2^*/2)$$

if it is assumed that the η_2^* (total number of conceptually unique inputs and outputs) are equally divided between inputs and outputs. This is in keeping with the following observations: if $\eta_2^*/2$ were to symbolize the number of items to be sorted (by a data processing program), then the number of comparisons (and hence conditional jumps) required would be on the order of J, as just defined [8]. This form is used subsequently as the "sort content" where either the variable "F" ("function point") or "V" (I/O count) is employed in the place of η_2^*.

DEVELOPMENT AND APPLICATION OF ESTIMATING FORMULAS

This section provides a number of formulas for estimating work hours and "SLOC" as functions of "function points" (F), "input/output count" (V), and several of the variates cited in Table II, which themselves are functions of (F) and (V), as described in the previous section).

To demonstrate the equivalency of the various measures and also to show their effectiveness as estimators, correlations were performed on the combinations of variates checked in Table II. Table III summarizes the results of using the variates checked to estimate "SLOC" in the Cobol and PL/1 applications (see Table I) as indicated. The estimating model relating

TABLE III
SUMMARY COMPARISON OF THE SLOC ESTIMATION APPROACHES

Estimators of SLOC (1) Variables Used Were:	Relative Error (2) Avg.	Standard Deviation	Sample Correlation Between the Variables And Actual SLOC
1. Function Points (COBOL)	.229	.736	.854
2. Function Points (PL/I)	.003	.058	.997
3. Function Sort Content (PL/I)	.007	.057	.997
4. Function Potential Volume (PL/I)	-.002	.057	.997
5. Function Information Content (PL/I)	-.002	.057	.997

Notes:

(1) The formulas used were:

1. $\hat{S} = 118.7\ (F) - 6{,}490$
2. $\hat{S} = 73.1\ (F) - 4{,}600$
3. $\hat{S} = 13.9\ (F/2)\ \log_2\ (F/2) + 5{,}360$
4. $\hat{S} = 6.3\ (F+2)\ \log_2\ (F+2) + 4{,}370$
5. $\hat{S} = 6.3\ (F \log_2 F) + 4{,}500$

(2) $\frac{\hat{S}-S}{S}$, where $\hat{S}$ = estimate and S = actual SLOC

TABLE IV
SUMMARY COMPARISON OF THE WORK-HOURS ESTIMATION APPROACHES

Estimators of Work-hours (1) Variables Used Were:	Relative Error (2) Avg.	Standard Deviation	Sample Correlation Between the Variables And Actual Work-Hours
1. Function Points	.242	.848	.935
2. Function Sort Content	.192	.759	.945
3. Function Information Content	.194	.763	.944
4. I/O Count	.206	.703	.945
5. I/O Count Sort Content	.195	.630	.954
6. I/O Count Information Content	.195	.637	.954
7. Source Lines of Code (PL/I)	.023	.213	.988
8. Source Lines of Code (COBOL)	.323	.669	.864

Notes:

(1) The formulas used were:

1. $\hat{W} = 54\ (F) - 13{,}390$
2. $\hat{W} = 10.75\ (F/2)\ \log_2\ (F/2) - 8{,}300$
3. $\hat{W} = 4.89\ (F \log_2 F) - 8{,}762$
4. $\hat{W} = 309\ (V) - 15{,}780$
5. $\hat{W} = 79\ (V/2)\ \log_2\ (V/2) - 8{,}000$
6. $\hat{W} = 35\ (V \log_2 V) - 8{,}900$
7. $\hat{W} = 0.6713\ (S) - 13{,}137$, PL/I
8. $\hat{W} = 0.3793\ (S) - 2{,}913$ COBOL

(2) $\frac{\hat{W}-W}{S}$, where $\hat{W}$ = estimated and W = actual work-hours

"function points" to PL/1 "SLOC" was found to be quite different from the model for Cobol. Significantly more Cobol "SLOC" are required to deliver the same amount of "function points" than are required with PL/1 "SLOC." The PL/1 data in particular closely approximate a straight line for all the measures. Any of the measures shown should be a good estimator for PL/1 "SLOC." In the next section, these measures are further validated using additional data from three other application development sites.

Table IV summarizes the results of using the variables checked in Table II and SLOC to estimate work-effort. The correlations and standard deviations of the data for the estimating formulas, using I/O count (V) and function point count (F), shown suggest that any of them would be an effective "formula" estimate. The measures based on "I/O count" show slightly, but not significantly, better statistics than those based on "function points." The last two rows of Table IV summarize the results of using "SLOC" to estimate work-effort. It is

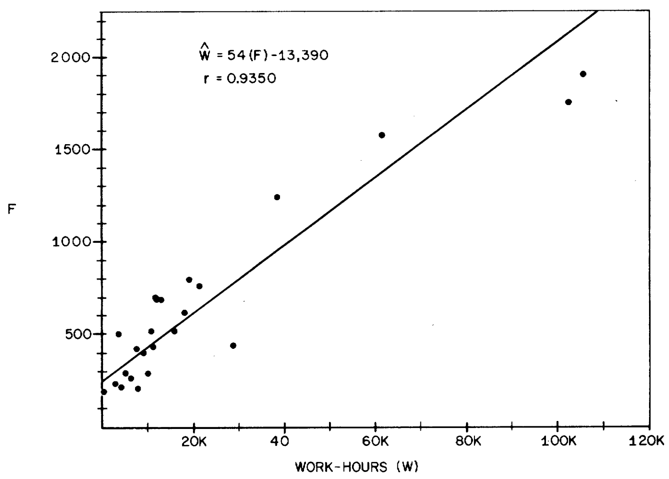


Fig. 1. Function points versus work-hours.

TABLE V
SOME VALIDATION STATISTICS

SLOC Estimating Formula (2)	(3) Relative Error Avg.	Standard Deviation	Sample Correlation Between $\hat{S}$ and S
$\hat{S}_1$	-.0753	.5438	.9367
$\hat{S}_2$	.2406	.5174	.9367
$\hat{S}_3$	-.0182	.4151	.9374
$\hat{S}_4$	.0629	.3983	.9289

Notes:
(1) For 'validation sites' 2, 3, 4, see detailed data in Table 6.

(2) $\hat{S}_1$ = 73.1F - 4600 (based on the PL/I cases)
$\hat{S}_2$ = 53.2F + 12773 (based on all 24 cases)
$\hat{S}_3$ = 66F (a simplified model)
$\hat{S}_4$ = 6.3Flog F + 4500 (based on the PL/I cases)
} derived from the DPS data

(3) $\frac{\hat{S}-S}{S}$, where S = actual SLOC and $\hat{S}$ = estimated SLOC

shown that the estimating model based on the Cobol data is quite different from the model based on PL/1 data. Almost twice as much work-effort is required to produce a "SLOC" of PL/1 as is required to produce a "SLOC" of Cobol. However, Table III shows that almost twice as much "function" is estimated to be delivered by an "SLOC" of PL/1 as is estimated to be delivered by an "SLOC" of Cobol. Therefore, it is advisible to keep these languages separated in estimating models based on "SLOC."

Fig. 1 is a scatter plot of actual "function points" and work-hours data and estimation formula number 1 from Table IV plotted on the same graph.

The correlations and standard deviations of linear models of "function points," "function sort content," "function information content," "I/O count," "I/O count," "I/O count sort content," and "I/O count information content," and "SLOC" are shown in Table IV. Each model could be an effective tool for validating estimates. The early availability of elements that comprise "function points" and "I/O count" for an application suggest that this validation could be done earlier in the development schedule than validations based on estimated "SLOC."

VALIDATION

The previous section, and the related figure and tables, developed several estimating formulas and explored their consistency within the DP Services data used to develop the formulas. This section validates several SLOC estimation formulas developed from the DP Services data presented in Table I against three *different* development sites. While it is *interesting* to note relations between "function points" and SLOC, it is *significant* to know that these relations hold also on a *different* set of data than that employed to develop them originally. The excellent degree of fit obtained would tend to support the view that these (and the other) formulas not validated here have some degree of universality. Table V presents four formulas developed from the DP Services data and the statistics of their validation on the data form the other three sites. Table VI provides the data from the three sites from which the statistics in Table V are derived. The very high values of sample correlation between the estimated and actual SLOC for the 17 validation sites, listed in Table V (i.e., >0.92) are most encouraging.

TABLE VI
VALIDATION OF SOME SLOC ESTIMATING EQUATIONS

Application Number	Function Points F	PL/I SLOC S	$\hat{S}_1$ KSLOC	$\hat{S}_2$ KSLOC	$\hat{S}_3$ KSLOC	$\hat{S}_4$ KSLOC
1. DPS - 4	759	54K	50.9	53.2	50.1	50.2
2. 14	682	42K	45.2	49.1	45.0	44.9
3. 17	606	40K	39.7	45.0	40.0	39.8
4. 20	1,572	110K	110.3	96.4	103.8	109.7
5. Site 2-1	803	31.0K	54.1	55.5	53.0	53.3
6. 2	335	31.4K	19.9	30.6	22.1	22.2
7. 3	685	23.3K	45.5	49.2	45.2	45.2
8. 4	1,119	126.6K	77.2	72.3	73.9	75.9
9. 5	712	40.9K	47.4	50.7	47.0	47.0
10. 6	261	19.9K	14.5	26.7	17.2	17.7
11. Site 3-1	1,387	120K	96.8	86.6	91.5	95.7
12. 2	1,728	120K	121.7	104.7	114.0	121.6
13. 3	2,878	150K	205.8	165.9	190.0	212.8
14. Site 4-1	2,165	123.2K	153.7	128.0	142.9	155.6
15. 2	236	16.3K	12.7	25.3	15.6	16.2
16. 3	3,694	195.0K	265.4	209.3	243.8	280.3
17. 4	224	41.0K	11.8	24.7	14.8	15.5
18. 5	42	6.5K	-1.5	15.0	2.8	5.9
19. 6	1,629	102.0K	114.5	99.4	107.5	114.0
20. 7	105	9.8K	3.1	18.4	6.9	8.9
21. 8	581	45.9K	37.9	43.7	38.3	38.1
Average Relative Error			-.060	.186	-.024	.051
Std. Deviation of Relative Error			.488	.480	.372	.358
Correlation With S			.938	.938	.938	.997

CONCLUSION

The "function point" software estimation procedure appears to have a strong theoretical support based on Halstead's software science formulas. Apparently, some of Halstead's formulas are extremely robust and can be applied to the major inputs and outputs of a software product at the top level. 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.

The observations suggest a two-step estimate validation process, which uses "function points" or "I/O count" to estimate, early in the development cycle, the "SLOC" to be produced. The work-effort would then be estimated from the estimated "SLOC." This approach can provide an early bridge between "function points," software science," and "SLOC," until "function points" and "software science" have a broader supporting base of productivity data.

APPENDIX

A. Function Points Definitions

This section provides the basic definitions supporting the measurement, recording, and analysis of function points, work-effort, and attributes.

B. General

The following considerations are generally applicable to the specific definitions of function points, work-effort, and attributes in later paragraphs in this section.

1) Development Work-Product versus Support Work-Product: Development productivity should be measured by counting the function points added or changed by the development or enhancement project. Therefore, we have the following.

Development Work-Product: The absolute value sum of all function points added or changed by the development or enhancement project. (Deleted function points are considered to be changed function points.)

Support productivity should be measured by counting the total function points supported by the support project during the support period. Therefore, we have the following.

Support Work-Product: The original function points of the application, adjusted for any changes in complexity introduced, plus any function points added, minus any function points deleted by subsequent enhancement projects.

2) Measurement Timing: To provide the work-product, work-effort, and attributes measures needed for each development project, enhancement project, and support project to be measured, the indicated measures should be determined at the following times in the application life cycle.

- The development work-product and attributes measures should be determined at the completion of the *external design phase* for each development and enhancement project (when the user external view of the application has been documented).
- The development work-product, work-effort, and attributes measures should be determined at the completion of the *installation phase* for each development and enhancement project (when the application is ready for use).
- The support work-product, work-effort, and attributes measures should be determined at the end of *each year* of support and use for each support project.

3) Application Boundaries: Normally, as shown in Fig. 1, a single continuous external boundary is considered when counting function points. However, there are two general situations where counting function points for an application in parts, is necessary.

a) The application is planned to be developed in multiple stages, using more than one development project.

This situation should be counted, estimated, and measured as *separate projects*, including all inputs, outputs, interfaces, and inquiries crossing *all* boundaries.

b) The application is planned to be developed as a single application using one development project, but it is so large that it will be necessary to divide it into subapplications for counting function points.

The internal boundaries are arbitrary and are for counting purposes only. The subapplications should be counted separately, but *none* of the inputs, outputs, interfaces, and inquiries crossing the *arbitrary internal* boundaries should be counted. The function points of the subapplications should then be summed to give the total function points of the application for estimation and measurement.

4) Brought-In Application Code: Count the function points provided by brought-in application code (reused code), such as: an IBM, IUP, PP, or FDP; an internal shared application; or a purchased application if that code was selected, modified, integrated, tested, or installed by the project team. However, do *not* count the function points provided by the brought-in code that provided user function beyond that stated in the approved requirements.

Some examples are the following.

a) Do count the function points provided by an application picked up from another IBM site, or project, and installed by the project team.

b) Do *not* count the function points provided by software, such as IMS or a screen compiler, if that software had been made available by another project team.

c) Do *not* count ADF updates of *all* files if the user only required updates of *three* files, even though the capability may be automatically provided.

5) Consider All Users: Consider *all* users of the application, since each application may have provision for many specified user functions, such as:

- end user functions (enter data, inquire, etc.)
- conversion and installation user functions (file scan, file compare discrepancy list, etc.)
- operations user functions (recovery, control totals, etc.).

C. Function Points Measure

After the general considerations described in the preceding paragraphs have been decided, the function points measure is accomplished in three general steps:

a) classify and count the five user function types;
b) adjust for processing complexity;
c) make the function points calculation.

The paragraphs in this section define and describe each of these steps. The first step is accomplished as follows.

Classify, to three levels of complexity, the following user functions that were made available to the user through the design, development, testing, or support efforts of the development, enhancement, or support project team:

a) external input types;
b) external output types;
c) logical internal file types;
d) external interface file types;
e) external inquiry types.

Then *list* and *count* these user functions. The counts should be recorded for use in the function points calculation, on an appropriate work-sheet. Examples of the useful function points work-sheets are provided in Section IV, function points work-sheets.

The definitions of each of the user functions to be counted, and the levels of complexity, are provided in the following paragraphs.

1) External Input Type: Count each unique user *data* or user *control* input type that enters the external boundary of the application being measured, and *adds* or *changes* data in a logical internal file type. An external input type should be considered unique if it has a different *format*, or if the external design requires a *processing logic* different from other external input types of the same format. As illustrated in Fig. 1, include external input types that enter directly as transactions from the user, and those that enter as transactions from other applications, such as, input files of transactions.

Each external input type should be classified within three levels of complexity, as follows.

- *Simple*: Few data element types are included in the external input type, and few logical internal file types are referenced by the external input type. User human factors considerations are not significant in the design of the external input type.
- *Average:* The external input type is not clearly either simple or complex.
- *Complex:* Many data element types are included in the external input type, and many logical internal file types are referenced by the external input type. User human factors considerations significantly affect the design of the external input type.

Do *not* include external input types that are introduced into the application only because of the technology used.

Do *not* include input files of records as external input types, because these are counted as external interface file types.

Do *not* include the input part of the external inquiry types as external input types, because these are counted as external inquiry types.

2) External Output Type: Count each unique user *data* or *control* output type that leaves the external boundary of the application being measured. An external output type should be considered unique if it has a different *format*, or if the external design requires a *processing logic* different from other external output types of the same format. As illustrated in Fig. 1, include external output types that leave directly as reports and messages to the user, and those that leave as reports and messages to other applications, such as, output files of reports and messages.

Each external output type should be classified within three levels of complexity, using definitions similar to those for the external input types [paragraph C1)]. For reports, the following additional complexity definitions should be used.

- *Simple:* One or two columns. Simple data element transformations.
- *Average:* Multiple columns with subtotals. Multiple data element transformations.
- *Complex:* Intricate data element transformations. Multiple and complex file references to be correlated. Significant performance considerations.

Do *not* include external output types that are introduced into the application only because of the technology used.

Do *not* include output files of records as external output types, because these are counted as external interface file types.

Do *not* include the output response of external inquiry types as external output types, because these are counted as external inquiry types.

3) Logical Internal File Type: Count each major logical group of user *data* or *control* information in the application as a logical internal file type. Include each logical file, or within a data base, each logical group of data from the viewpoint of the user, that is *generated, used,* and *maintained* by the application. Count logical files as described in the external design, not physical files.

The logical internal file types should be classified within three levels of complexity as follows.

- *Simple:* Few record types. Few data element types. No significant performance or recovery considerations.
- *Average:* The logical internal file type is not clearly either simple or complex.
- *Complex:* Many record types. Many data element types. Performance and recovery are significant considerations.

Do *not* include logical internal files that are *not* accessible to the user through external input, output, interface file, or inquiry types.

4) External Interface File Type: Files *passed* or *shared* between applications should be counted as external interface file types within *each* application. Count each major logical group of user *data* or *control* information that enters or leaves the application, as an external interface file type. External interface file types should be classified within three levels of complexity, using definitions similar to those for logical internal file types [paragraph C3)].

Outgoing external interface file types should also be counted as logical internal file types for the application.

5) External Inquiry Type: Count each unique input/output combination, where an input causes and generates an immediate output, as an external inquiry type. An external inquiry type should be considered unique if it has a *format* different from other external inquiry types in either its input or output parts, or if the external design requires a *processing logic* different from other external inquiry types of the same format. As illustrated in Fig. 1, include external inquiry types that enter directly from the user, and those that enter from other applications.

The external inquiry types should be classified within three levels of complexity as follows.

a) Classify the input part of the external inquiry using definitions similar to the external input type [paragraph C1)].

b) Classify the output part of the external inquiry type using definitions similar to the external output type [paragraph C2)].

3) The complexity of the external inquiry type is the greater of the two classifications.

To help distinguish external inquiry types from external input types, consider that the input data of an external inquiry type is entered only to direct the search, and no update of logical internal file types should occur.

Do *not* confuse a query facility as an external inquiry type. An external inquiry type is a direct search for specific data, usually using only a single key. A query facility provides an organized structure of external input, output, and inquiry types to compose many possible inquiries using many keys and operations. These external input, output, and inquiry types should *all* be counted to measure a query facility.

6) Processing Complexity: The previous paragraphs define the external input, external output, internal file, external interface file, and external inquiry types to be listed, classified, and counted. The function points calculation [paragraph C7)] describes how to use these counts to measure the standard processing associated with those user functions. This paragraph describes how to apply some general application characteristics to adjust the standard processing measure for processing complexity.

The adjustment for processing complexity should be accomplished in three steps, as follows.

a) The *degree of influence* of each of the 14 general characteristics, on the value of the application to the users, should be estimated.

b) The 14 degree of influence(s) should be summed, and the total should be used to develop an *adjustment factor* ranging from 0.65 to 1.35 (this gives an adjustment of +/- 35 percent).

c) The *standard processing measure* should be multiplied by the adjustment factor to develop the work-product measure called function points.

The first step is accomplished as follows.

Estimate the degree of influence, on the application, of each of the 14 general characteristics that follow. Use the degree of influence measures in the following list, and record the estimates on a work-sheet similar to Fig. 2.

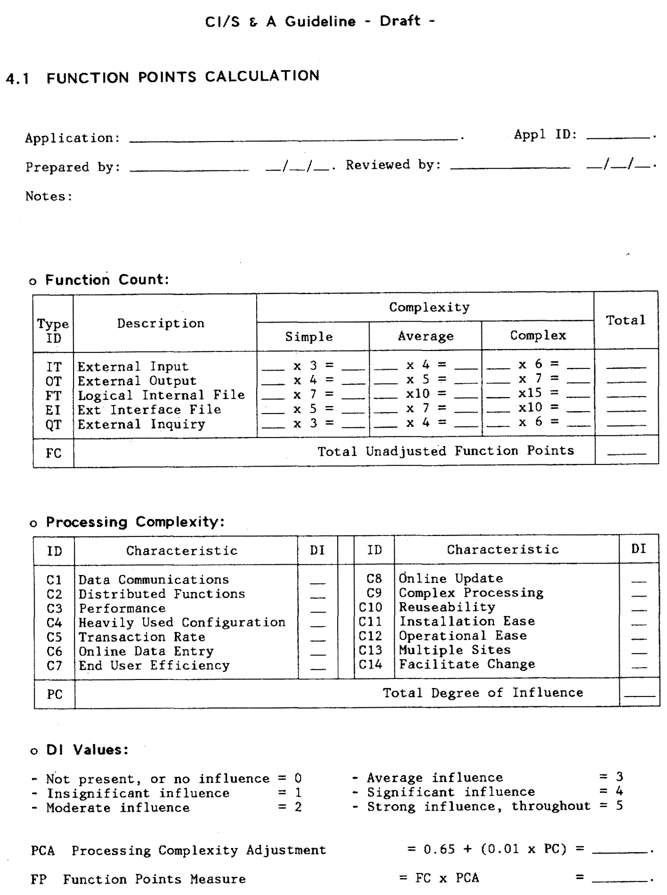
Degree of Influence Measures:

• Not present, or no influence if present	= 0
• Insignificant influence	= 1
• Moderate influence	= 2
• Average influence	= 3
• Significant influence	= 4
• Strong influence, throughout	= 5.

General Application Characteristics:

a) The *data* and control information used in the application are sent or received over *communication* facilities. Terminals connected locally to the control unit are considered to use communication facilities.

b) *Distributed* data or processing *functions* are a characteristic of the application.



CI/S & A Guideline - Draft -

4.1 FUNCTION POINTS CALCULATION

Application: ______________________. Appl ID: ______.

Prepared by: __________ __/__/__. Reviewed by: __________ __/__/__.

Notes:

o Function Count:

Type ID	Description	Complexity: Simple	Complexity: Average	Complexity: Complex	Total
IT	External Input	___ x 3 = ___	___ x 4 = ___	___ x 6 = ___	___
OT	External Output	___ x 4 = ___	___ x 5 = ___	___ x 7 = ___	___
FT	Logical Internal File	___ x 7 = ___	___ x10 = ___	___ x15 = ___	___
EI	Ext Interface File	___ x 5 = ___	___ x 7 = ___	___ x10 = ___	___
QT	External Inquiry	___ x 3 = ___	___ x 4 = ___	___ x 6 = ___	___
FC	Total Unadjusted Function Points				___

o Processing Complexity:

ID	Characteristic	DI		ID	Characteristic	DI
C1	Data Communications	___		C8	Online Update	___
C2	Distributed Functions	___		C9	Complex Processing	___
C3	Performance	___		C10	Reuseability	___
C4	Heavily Used Configuration	___		C11	Installation Ease	___
C5	Transaction Rate	___		C12	Operational Ease	___
C6	Online Data Entry	___		C13	Multiple Sites	___
C7	End User Efficiency	___		C14	Facilitate Change	___
PC	Total Degree of Influence					___

o DI Values:

- Not present, or no influence	= 0	- Average influence	= 3
- Insignificant influence	= 1	- Significant influence	= 4
- Moderate influence	= 2	- Strong influence, throughout	= 5

```
PCA  Processing Complexity Adjustment      = 0.65 + (0.01 x PC) = ______.
FP   Function Points Measure               = FC x PCA           = ______.
```

Fig. 2. Function points calculation worksheet.

c) Application *performance* objectives, in either response or throughput, influenced the design, development, installation, and support of the application.

d) A *heavily used* operational *configuration* is a characteristic of the application. The user wants to run the application on existing or committed equipment that will be heavily used.

e) The *transaction rate* is high and it influenced the design, development, installation, and support of the application.

f) *On-line data entry* and control functions are provided in the application.

g) The on-line functions provided, emphasize *end user efficiency*.

h) The application provides *on-line update* for the logical internal files.

i) *Complex processing* is a characteristic of the application. Examples are:

- many control interactions and decision points
- extensive logical and mathematical equations
- much exception processing resulting in incomplete transactions that must be processed again.

j) The application, and the code in the application, has been specifically designed, developed, and supported for *reusability* in other applications, and at other sites.

k) Conversion and *installation ease* are characteristics of the application. A conversion and installation plan was provided, and it was tested during the system test phase.

l) *Operational ease* is a characteristic of the application. Effective start-up, back-up, and recovery procedures were provided, and they were tested during the system test phase. The application minimizes the need for manual activities, such as, tape mounts, paper handling, and direct on-location manual intervention.

m) The application has been specifically designed, developed, and supported to be installed at *multiple sites* for multiple organizations.

n) The application has been specifically designed, developed, and supported to *facilitate change*. Examples are:

- flexible query capability is provided
- business information subject to change is grouped in tables maintainable by the user.

7) Function Points Calculation: The previous paragraphs described how the function types are listed, classified, and counted; and how the processing complexity adjustment is determined. This paragraph describes how to make the calculations that develop the function points measures.

Using the definitions in paragraph C1), two equations have been developed to more specifically define the *development work-product* measure and the *support work-product* measure:

$$\text{Development Work-Product FP Measure} = (\text{Add} + \text{ChgA})\ \text{PCA2} + (\text{Del})\ \text{PCA1} = \underline{\quad\quad}.$$

$$\text{Support Work-Product FP Measure} = \text{Orig FP} + (\text{Add} + \text{ChgA})\ \text{PCA2} - (\text{Del} + \text{ChgB})\ \text{PCA1} = \underline{\quad\quad}.$$

Orig FP	= adjusted FP of the application, evaluated as they were before the project started.
Add	= unadjusted FP added to the application, evaluated as they are expected to be at the completion of the project.
ChgA	= unadjusted FP changed in the application, evaluated as they are expected to be at the completion of the project.
Del	= unadjusted FP deleted from the application, evaluated as they were before the project started.
ChgB	= unadjusted FP changed in the application, evaluated as they were before the project started.
PCA1	= the processing complexity adjustment pertaining to the application before the project started.
PCA2	= the processing complexity adjustment pertaining to the application after the project completion.

REFERENCES

[1] A. J. Albrecht, "Measuring application development productivity," in *Proc. IBM Applications Develop. Symp.*, Monterey, CA, Oct. 14-17, 1979; GUIDE Int. and SHARE, Inc., IBM Corp., p. 83.

[2] M. H. Halstead, *Elements of Software Science.* New York: Elsevier, 1977.

[3] J. E. Gaffney, "Software metrics: A key to improved software development management," presented at the Conf. Comput. Sci. Statist., 13th Symp. on Interface, Pittsburgh, PA, Mar. 1981; also Proceedings, Springer-Verlag, 1981.

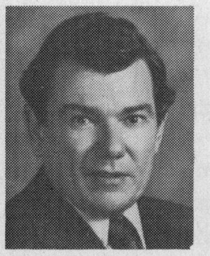
[4] K. Christensen, G. P. Fitsos, and C. P. Smith, "A perspective on software science," *IBM Syst. J.*, vol. 20, no. 4, pp. 372-387, 1981.

[5] J.L.F. Dekerf, "APL and Halstead's theory of software metrics," in *APL81 Conf. Proc. ACM* (APL Quote Quad), vol. 12, Sept. 1981, pp. 89-93.

[6] A. O. Allen, "Probability, statistics, and queueing theory–With computer science applications," in *Computer Science and Applied Mathematics Series.* New York: Academic, 1978.

[7] J. E. Gaffney, "A comparison of a complexity–Based and Halstead program size estimates," presented at the 1979 ACM Comput. Sci. Conf., Dayton, OH, Feb. 1979.

[8] D. F. Stanat and D. F. McAllister, *Discrete Mathematics in Computer Science.* Englewood Cliffs, NJ: Prentice-Hall, 1977, p. 265.
[9] N. Pippenger, "Complexity theory," *Scientific Amer.*, p. 120, June 1978.



Allan J. Albrecht received the B.S.E.E. degree from Bucknell University, Lewisburg, PA, in 1949.

He is currently the Program Manager of the Application Development and Maintenance (AD/M) Measurement Program for IBM. The program objective is to measure, establish, and improve the effectiveness and efficiency of the development of application programs in the company. Before this assignment, he was on the staff of the Director of IBM's DP Services Organization (now known as Information System Services). In addition to providing software project management advice and consultation to the Director, he was responsible for reviewing, developing, and revising the processes used to propose, manage, and review their contracts. He has had about 25 years of project management experience. The software projects he has managed have included missile range control systems, an insurance medical claims processing system, and a motor freight company management system. His experience has included development of hardware, development of software for internal use, development of software for license to customers, and development of software under contract to customers.

John E. Gaffney, Jr. (M'55) received the A.B. degree from Harvard University, Cambridge, MA, in 1955 and the M.S. degree from Stevens Institute of Technology, Hoboken, NJ, in 1957. He has done additional graduate work at The American University, Washington, DC.

He joined IBM, Ossining, NY, in 1957 in the Research Division, working on the SABRE airlines reservation system. Subsequently, he had assignments in Poughkeepsie, NY, San Jose, CA, Stockhom, Sweden, Bethesda and Gaithersburg, MD, and Manassas, VA. He has worked in process control, image processing, artificial intelligence, sonar systems, and in recent years in software and cost engineering. He spent the period July 1982 to July 1983 on sabbatical with the National Weather Service, U.S. Department of Commerce, Silver Spring, MD, working on systems requirements and the application of artificial intelligence to weather forecasting. Currently, he is working in the Advanced Technology Department at IBM, FSD, Gaithersburg, MD.

Mr. Gaffney is a Registered Professional Engineer (electrical engineering) in the District of Columbia.

Measuring the Productivity of Computer Systems Development Activities with Function Points

CHARLES A. BEHRENS

Abstract—**The function point method of measuring application development productivity developed by Albrecht is reviewed and a productivity improvement measure introduced. The measurement methodology is then applied to 24 development projects. Size, environment, and language effects on productivity are examined. The concept of a productivity index which removes size effects is defined and an analysis of the statistical significance of results is presented.**

Index Terms—**Development, function points, measurement, productivity.**

Introduction and Background

THE subject of this paper is a description of the results obtained by implementing the function point method of productivity measurement in the data processing organization of a large financial institution.[1] The systems' development activities of this particular group are quite varied, in that multiple customers are served and various levels of language are employed (e.g. Cobol, Fortran, Focus). Further, the organization has made a substantial investment in productivity improvement by providing its programmers with an on-line development environment. Additional enhancements to this environment are planned.

In order to track the cost-benefit impact of its productivity improvement program, a measurement methodology was needed. It was decided not to develop such a methodology, but rather to adopt one that has been used with success in another organization. Discussions with other data processing groups and academic investigators, as well as a search of the literature, suggested that the function point method as developed by A. J. Albrecht had the most potential for success in our particular environment.

Albrecht developed his measurement technique in his capacity as Program Manager of the Application and Maintenance Measurement Program for IBM. His methodology, which was developed over a five year period, was reported at the October 1979 Guide/Share Meeting in Monterey, CA [1].

Albrecht's function point method can be summarized as

Manuscript received September 10, 1981; revised March 12, 1982.
The author is with Cooper, Behrens and McMullen, Inc., 129 E. 32nd St., Beach Haven, NJ 08008.

[1] This research project was undertaken under the auspices of the Application Productivity Group of the Equitable Life Assurance Society of the United States.