Automated Calculation of Function Points

Edward L. Jones, Ph.D.
Harris Corporation Visiting Professor
Florida A & M University
ejones@cis.famu.edu

ABSTRACT

Function points were introduced in 1979 as a line-of-code independent measure of software size which could be determined early in the life cycle. The function point count of a system is computed by counting and weighting "essential" information entities that flow across the boundary of the system or that are retained within the system. Until recently, the counting process appeared to be inherently manual because the counter is required to assign complexity weights to each information entity and an overall application complexity weight. Fortunately, evidence from industry is growing that these weights do not significantly improve the usefulness of function points as a predictor of (i.e., basis for estimating) eventual software size measured in lines of code. It is becoming clearer that the essence of function points is in the choice of which information entities to count, not in the methods used to weight the counts. This paper proposes a variant method of "function" counting which is applied to the information entities defined in the data dictionary resulting from a formal requirements analysis or design process. The argument is made that the proposed method captures the intent of the classical function point definition. A tool which automates the proposed counting rules is described and recommended as a must-have tool for facilitating the arduous process of building and validating function-points based estimation databases.

KEYWORDS: function points, bang, metrics, project management, CASE, estimation databases.

1 INTRODUCTION

In this section we introduce the function point definition proposed by Albrecht [Albrecht79, Albrecht83], discuss some variants of Albrecht function points, and summarize some results from field studies that show that the power of function points to predict lines of code is statistically independent of the variant method used. These findings are used to justify the simplified counting rules presented in section 2. Section 3 discusses the potential benefits to an organization of automating the function counting process. Section 4 presents conclusions and proposes areas of future research.

Albrecht's Counting Rules

The Albrecht function point definition is based on counting and individually weighting the complexity of the following five "function types" (essential information entities):

Input — unique user input that enters the boundary of the software system to cause processing to occur (e.g., transactions from a user or another application, input data, or control inputs).

Inquiry — unique query expressed as an input/output combination, where the input causes the immediate generation of an output (e.g., interactive inputs for which an immediate response is required) but does not require a file update.

Output — unique output that exits through the boundary of the software system targeted for a user or another application (e.g., data, control, or transactions to other applications).

Interface — unique data stores shared by the software system and other applications

Logical Internal File — unique group of data or a file whose data are accessible to the user via inquiries, inputs or outputs.

1. "unique" in that a different data format or processing logic is required.
The Albrecht function point counting process involves the following steps:

1. Identify all instances of each function type in the system.
2. Assign a complexity weight to each instance of a function type.
3. Calculate a weighted sum for each function type.
4. Calculate the sum over the five function types (the result is termed the unadjusted function points or function count).
5. Determine an overall application complexity measure by rating the complexity in fourteen application characteristics. The resulting weight is termed the application adjustment factor.
6. Multiply the unadjusted function points by the application adjustment factor to get the function points measure.

\[ FP = \text{APPLICATION\_ADJUSTMENT\_FACTOR} \times \text{FUNCTION\_COUNT}. \]

The function point counting guidebook in [IBM84] gives detailed instructions for performing these steps. Figure 1 and Tables 1–2 have been adapted from this guidebook. Step 1 of the process requires either a written specification of the system requirements or design, or someone experienced with the system. Classification of the function types involves a subjective judgment by the estimator, since there is some overlap in the function types.

The assignment of a complexity weight also requires judgment. As shown in Figure 1, the complexity of a function type instance is based on its composition (data element types) and its involvement in information processing (number of files or record types referenced). For each complexity rating (Low, Average, High) of a function type instance, a numeric weight is determined. Table 1 shows the complexity weights for each function type. Logical internal files have the highest weights; inputs and inquiries have the same (lowest) weights. Table 1 is used to record the results of Steps 2–4.

In Step 5, the overall application complexity is determined by rating the degree of influence of each of the fourteen application characteristics shown in Table 2. Finally, in Step 6 the function count is weighted by the application adjustment factor, also termed the value adjustment factor [Albrecht83]. Table 2 depicts this computation, which can adjust the function point count up or down by up to 35%.

### Table 1. Calculation of Albrecht Function Counts

<table>
<thead>
<tr>
<th>FUNCTION TYPE</th>
<th>Complexity</th>
<th>LOW</th>
<th>AVERAGE</th>
<th>HIGH</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inquiry</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interface</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical File</td>
<td>7</td>
<td>10</td>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the table, `DET` stands for Data Element Types.

**Data Element Types (DET)**

<table>
<thead>
<tr>
<th>File Types</th>
<th>1–4</th>
<th>5–15</th>
<th>&gt;15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–1</td>
<td>L</td>
<td>L</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>L</td>
<td>A</td>
<td>H</td>
</tr>
<tr>
<td>&gt;2</td>
<td>A</td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>

**Legend:**

- L = low
- A = average
- H = high

**Additional Factors**

Consider the following factors, relative to the average, to adjust the result up or down not more than one level:

- automatic cursor movement
- other human factors
- data conversion
- application performance
Table 2. Calculation of Application Adjustment Factors

<table>
<thead>
<tr>
<th>Application Characteristic</th>
<th>DI Application Characteristic</th>
<th>DI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data communication</td>
<td>On-line updates</td>
<td></td>
</tr>
<tr>
<td>Distributed functionality</td>
<td>Complex processing</td>
<td></td>
</tr>
<tr>
<td>Performance constraints</td>
<td>Reusability</td>
<td></td>
</tr>
<tr>
<td>High use configuration</td>
<td>Installation base</td>
<td></td>
</tr>
<tr>
<td>Transaction rate</td>
<td>Operation base</td>
<td></td>
</tr>
<tr>
<td>On-line data entry</td>
<td>Multiple sites</td>
<td></td>
</tr>
<tr>
<td>End-user efficiency</td>
<td>Adaptability</td>
<td></td>
</tr>
<tr>
<td><strong>SUBTOTAL</strong></td>
<td><strong>SUBTOTAL</strong></td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL Degree of Influence (TDI) ...</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

APPLICATION ADJUSTMENT FACTOR = 0.65 * (0.01 * TDI)

LEGEND:

<table>
<thead>
<tr>
<th>Degree of Influence (DI) Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – Not present, or no influence</td>
</tr>
<tr>
<td>1 – Insignificant</td>
</tr>
<tr>
<td>2 – Moderate</td>
</tr>
<tr>
<td>3 – Average</td>
</tr>
<tr>
<td>4 – Significant</td>
</tr>
<tr>
<td>5 – Strong, throughout</td>
</tr>
</tbody>
</table>

Variant Counting Methods

Over the years, numerous variants of the classical method have appeared. Even within the domain of information systems, at least 14 variant definitions of function points have been identified [Kemerer93]. Some variants have been introduced to remove ambiguities [Symons88, Jones88], others to simplify the counting [Gaffney91], or to adapt function points to new analysis/design technology [Kemerer93]. Still other variants seek to extend the domain of applicability beyond information systems into computation-intensive applications [Jones88, Whitmire95].

An important question is: "Does it matter which of the variants one uses?" Recent findings suggest that when the goal of function point counting is to get an early estimate of eventual size in lines of code, nearly all the variants have a significant statistical correlation with lines of code. A discussion of several results follows.

Table 3 compares four variants on the bases of number of function types, number of functional complexity levels, and number of application characteristics used to compute the overall application adjustment factor. These three represent the places where the estimator judgment is required. A trend suggested by the table is the reduction in the number of places in the estimation process where estimator judgment is required.

The SPQR/20 fp variant includes the same five function types as Albrecht, but includes a single level of functional complexity—Albrecht's average weight. SPQR/20 fp reduces the number of application characteristics from 14 to two—data complexity and logical complexity.

The Mark II variant of Symons maps the five Albrecht function types into three types: input transactions, output transactions and entities. Albrecht interfaces and inquiries are counted as transactions, internal logical files as entities. Each count is weighted by an empirically determined weight specific to the function type. The application adjustment factor is determined using 19 application characteristics instead of the 14 used by Albrecht.

Gaffney's 3/4 Externals variant counts either three (Inputs, Outputs, Inquiries) or four (Inputs, Outputs, Inquiries, Interfaces) of the five Albrecht function types. The Gaffney variant totally eliminates all weights. Gaffney claims that, despite the ultimate simplicity, the resulting measure is as good a predictor of size (measured in lines of code) as is Albrecht's function points.

The 3-D function points variant [Whitmire95] extends function points into the domain of computation/control intensive applications by explicitly adding the two dimensions of function (transformations) and control (state transitions) to the dimension treated by the Albrecht method.

Transformations within the function dimension are identified and weighted in a similar manner as are the Albrecht's Inputs and Outputs (see Figure 1). Only one level of weights applies to transitions. Because function and control are included in the 3-D counts, no application adjustment factor is used.

Does It Matter?

Field results to date point to a startling conclusion: most of the variants, when used consistently by an organization, have nearly identical ability to predict the eventual size of a software system. In this section we present some of these conclusions and offer an explanation.

Jeffrey, et. al. [Jeffrey93] conclude that the SPQR/20 fp method [Jones88], which simplifies the application adjustment factor determination, is statistically equivalent to the Albrecht method:
Another study [Kemerer93] on the relative estimating power of two variant function point counting methods which simplified the complexity weighting of function types, reach a similar conclusion: the power of function point variants to estimate lines of code is statistically independent of various weighting schemes applied to the primitive counts.

Results from Gaffney's [Gaffney91] study of radical simplifications, the elimination of weights altogether, show that: (1) using raw counts of function types correlates as well with lines of code as do Albrecht function points; and (2) using raw counts of only Inputs, Outputs and Inquiries (i.e., ignoring Interfaces and Logical Files) correlates as well with line of code as do Albrecht function points. Moreover, his results appear to be comparable for both information-intensive and aerospace applications.

The evidence does not state that the numerical values obtained by the variant counting methods are statistically equivalent, but that their utility for early estimation of project size and effort are effectively equivalent.

2 AUTOMATING COUNTING

In this section we define the function counting method which lends itself readily to automation. We present the counting rules, then relate the rules to published results cited in section 1.

Data Volume Counting Rules

The proposed counting rules produce a function count which measures the volume of data flow at a point (e.g., at a boundary) in a system. The volume is measured in number of atomic data elements.

The basis for automation is the use of formalisms (such as DeMarco notation) used in CASE tool data dictionaries to express the composition of data entities. The counting rules are summarized below.

- If data item X is atomic,
  \[ \text{count}(X) = 1. \]
- If data item \( X = A + B + C \),
  \[ \text{count}(X) = \text{count}(A) + \text{count}(B) + \text{count}(C). \]
- If data item \( X = \{ A | B | C \} \),
  \[ \text{count}(X) = \text{count}(A) + \text{count}(B) + \text{count}(C). \]
- If data item X = { A },
  \[ \text{count}(X) = \text{count}(A). \]
- If data item X = { A },
  \[ \text{count}(X) = \text{count}(A). \]

The proposed function counting process involves the following steps:

1. Identify and tag all data dictionary entries corresponding to "function types" to be included in the counts.
2. Apply the counting rules above to compute function counts for each tagged element.
3. Sum the counts over all the tagged elements.
4. Apply an application adjustment factor of choice.

At minimum this approach requires a data dictionary which contains the results from an analysis/design process, along with a designation of which entries in the dictionary correspond to the function types to be counted. As step 4 implies, the counts can be used as-is, or adjusted for application complexity.

Features of the Volume Counting Approach

Note that when no application adjustment is applied, DVC is closest to the Gaffney variant, where all subjective weighting is eliminated. The method essentially replaces estimation of the decomposition (Albrecht's Data Element Types) with exact counting. Whereas the Gaffney method weights all externals by one, this approach permits the volume of a data item to serve as its weight.

Readers familiar with DeMarco's bang metric [Demarco82] will recognize that DVC, when applied to all entries in the system data dictionary, computes a value similar to DeMarco's TC, the raw token counts associated with DFD transformations. Note also that DVC avoids DeMarco's quagmire of weighting the TCs to correct for variations in complexity; complex weighting schemes nullify the practical usefulness of an elegant concept.

The reader should be clear on this point: DVC merely provides a tool which may support any variant method in which the individual function types instances are not weighted. With DVC, the human estimator is still responsible for making the critical decision of which function types to include in the counting.
Table 3. Representative Variants to Albrecht Function Point Counting

<table>
<thead>
<tr>
<th>Albrecht Feature Modified</th>
<th>VARIANT FUNCTION POINT COUNTING METHOD</th>
<th>3-D FP vs. 2-D FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Function classifications (types)</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>#Function complexity categories</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>#Application characteristics used in application adjustment factor</td>
<td>14</td>
<td>2</td>
</tr>
</tbody>
</table>

An Example

Figure 2 contains a specification of a system in terms of a data flow diagram and supporting data dictionary. (For the purposes of this example, the details of the system are irrelevant.) In the diagram, the encircled numbers beside data flows and stores are the corresponding data volume counts. Table 4 contains the results of applying DVC and the variants given in section 1 to compute function counts for this system. The functional complexity for all the Albrecht function types is LOW, as are the transformations used by the Whitmire variant. Note that the Whitmire 3-D function points include transformations and state transitions, in addition to the Albrecht function types.

This example shows that the numerical values produced by the variants differ, but all methods look at elements of the data (and process) dictionary. Compared to the Albrecht function counts some methods routinely yield lower counts (Gaffney, Mark II), some higher counts (Whitmire), and others intermediate counts (SPQR/20 fp and DVC).

A Case for Data Volume Counting

We conclude from the studies reported in section 1 that the essential predictive property of Albrecht function points is preserved in variant methods which reduce the number of subjective decisions (weight assignment) made during counting. We claim that data volume is an essential aspect of these function point variants.

Although the DVC counting approach has not been validated, we believe that it will have the same predictive properties as the variants discussed in the previous sections, for the following reasons:

1. DVC incorporates concept of data flow volume, which is present in most function point variants.

Table 4. Comparison of Function Counts for Variants

<table>
<thead>
<tr>
<th>Data Dictionary Entry</th>
<th>Albrecht Function Type</th>
<th>Function Weight (type)</th>
<th>Albrecht</th>
<th>SPQR/20</th>
<th>Mark II</th>
<th>Gaffney</th>
<th>Whitmire</th>
<th>DVC</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>Input</td>
<td>3</td>
<td>4</td>
<td>0.44 (IT)</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>IN3</td>
<td>Input</td>
<td>3</td>
<td>4</td>
<td>0.44 (IT)</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>OUT1</td>
<td>Output</td>
<td>4</td>
<td>5</td>
<td>0.38 (OT)</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>IN2</td>
<td>Inquiry(I)</td>
<td>3</td>
<td>4</td>
<td>0.44 (IT)</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>OUT2</td>
<td>Inquiry(O)</td>
<td>3</td>
<td>4</td>
<td>0.38 (OT)</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>DS1</td>
<td>Interface</td>
<td>5</td>
<td>7</td>
<td>0.44 (IT)</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>DS2</td>
<td>Interface</td>
<td>5</td>
<td>7</td>
<td>0.44 (IT)</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>DS4</td>
<td>Logical File</td>
<td>7</td>
<td>10</td>
<td>1.67 (E)</td>
<td>-</td>
<td>7</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>DS3</td>
<td>Logical File</td>
<td>7</td>
<td>10</td>
<td>1.67 (E)</td>
<td>-</td>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Function 1–4</td>
<td>Transform</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7 x 4</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(not shown in Figure)</td>
<td>Transition</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2 x 3</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>TOTALS ...</td>
<td></td>
<td>40</td>
<td>55</td>
<td>6.4</td>
<td>7</td>
<td>74</td>
<td>28</td>
<td></td>
</tr>
</tbody>
</table>
2. DVC eliminates subjective functional weights in favor of the direct measurement of data volume.

3. Empirical results show that simplifying functional weighting does not diminish the predictability of eventual software size.

DVC is modular. DVC can be used in a standalone mode, or incorporated into some other variant counting method. As an example of the latter case, DVC can be used to perform steps 2–3 of the Albrecht function point counting process.

DVC is broadly applicable to various analysis technologies, including structured and object-oriented methods, so long as a data dictionary is produced. Most CASE environments provide the necessary DVC inputs.

3 FUNCTION–BASED ESTIMATION

This section addresses the questions: "How might my organization get started using function–based estimation?" and "How can the DVC tool help my organization?"

Function Point Technology Resources

As a first step an organization should consider joining the International Function Point Users Group (IFPUG). Members participate in its semi-annual conferences and tutorials.

2. IFPUG has over 550 corporate members from 28 countries. Spring and Fall conferences are held. Annual corporate membership is less than $300. IFPUG supports evolving standards and is a clearinghouse for industry experiences. Address: 5008–28 Pine Creek Drive / Westerville, OH 43081–4899/ (614) 895–7130.
IFPUG members also have access to instructional materials, including a course leading to certification as a function points analyst.

A second step should be the investigation of function–point oriented project cost estimation models and supporting toolsets. ESTIMACS [Rubin83] and SPQR/20 fp [Jones88] are examples of such tools. IFPUG members are likely to have experience with these and other estimation tools.

Function–Based Estimation Database

An organization should establish a testbed of on–going and completed projects for the purpose of validating and calibrating function–based models used by the organization. A useful estimation database requires the following: a sufficient number of projects; comparable function counts across all projects; metrics on the completed project, e.g., effort and source lines of code. The cost of the database is proportional to the human effort required to select projects, to locate relevant project artifacts (including metrics), to ensure that function counts are comparable. Automation of function counting ensures comparability while reducing the cost of establishing the database.

The simplest model to build from the estimation database is one which relates function counts to software size measured in source lines of code (SLOC). Validation should focus initially on whether function counts are good predictors of SLOC and under what circumstances the quality of the prediction is acceptable. The model can be expressed mathematically as a regression equation; statistical methods can be applied to determine the quality of the regression fit [as in Gaffney91]). The advantage of a model relating function to SLOC is that the organization can continue to use SLOC–based estimation models such as COCOMO, and use the function counting as an alternative source for line–of–code estimates.

More sophisticated models are possible, either via commercial estimation tools or by developing custom models in–house.

Function Counting for Emerging Technology

The move away from structured analysis methods to more object–oriented methods is reflected in variant “function point” definitions such as the Entity–Relationship approach given in [Kemerer93]. DVC is applicable to this approach since the information about entity decomposition and relationships are routinely stored in CASE tool data dictionaries. In fact, we expect DVC to be applicable to nearly any analysis methods which produces a data dictionary, with appropriate modification to the counting rules.

CASE tool vendors are likely to provide variants of function counting that fall between DeMarco’s bang and DVC.

How Can DVC Help?

Finally, as organizations respond to their need for historical project estimation databases, a tool such as DVC facilitates the retrospective capture of function–oriented software size data from old projects.

The Data Volume Counter is a research tool. It is offered to organizations wishing to begin or accelerate the move toward function counting. DVC ensures consistency of counting, hence the comparability of function counts across projects in the estimation database.

For an organization which has completed a number of projects which were developed using CASE tools, DVC is an economical way to add those projects to the estimation database.

4 CONCLUSIONS AND FUTURE WORK

The original intent of function point analysis appears to be the estimation of the volume of data flow across the boundary of a system and the volume of “essential” data flow within the system (i.e., logical files). Without the support of CASE tools, the analyst resorted to using weights to estimate the volume of “atomic” data elements associated with a system entity. The prevalence of CASE tools renders the estimation of entity weights unnecessary: they can be counted outright, given a valid data dictionary defining the set of system entities to be included in the analysis.

A variant counting method, data volume counting (DVC), which lends itself to automation has been defined. Results from studies of other variant methods lead the author to believe that DVC will also preserve the correlation to SLOC that is common across popular variants of the Albrecht function point measure.

Because DVC is readily adaptable to accept data dictionary reports from any CASE tool, numerous opportunities for using and validating this variant can be exploited, at low cost. The DVC tool not only ensures consistency and efficiency of counting, but it also permits function point growth to be tracked and managed during analysis and design in much the same way line–of–code growth is managed during implementation. Moreover, DVC provides a quantitative means of measuring convergence of analysis/design processes.

Future work includes the validation of DVC with respect to SLOC estimation. Because validation requires an estimation
database, validation will need to be performed in non-academic setting, or with projects of non-academic origin. Other issues to be investigated include tracking the growth of DVCs over the development lifecycle as a quantitative indicator of analysis/design progress.

REFERENCES


