Measure Size, Complexity of Algorithms Using Function Points

Nancy Redgate
PRI Automation

Charles B. Tichenor
Defense Security Cooperation Agency

Software developers and software metrics analysts have long known that algorithms can be of important functionality in software applications. Examples of these algorithms' functions are: controlling a nuclear reactor, calculating complex pricing arrangements, optimizing production levels in manufacturing, and finding the shortest routes through transportation networks. These algorithms can require considerable effort to develop, and can contribute significantly to the size and complexity of the software. There have been a number of attempts to quantify the size and complexity of these algorithms using function point metrics; however, no such method is generally recognized as satisfactory. This leads to situations where functionality is only partly accounted for, or the current function point methodology is "patched up" in academically controversial ways. In contrast, our research shows that no "patches" are needed to account for many of these algorithms. We show how to identify them, disassemble them into functional components, and measure their size and complexity while remaining within the strict interpretation of current counting rules. Complete accounting of these algorithms leads to better software sizing accuracy, which produces better forecasts of costs, schedules, and measures of quality in terms of defects per function point delivered.

An algorithm is a set of equations that is executed in a logical sequence to produce an external output. In the fields of function point analysis and operations research, an algorithm can be seen as a critical tool to reduce effort needed to solve a complex series of calculations. We have written this paper focusing on intermediate function point analysis theory, and use linear programming to exemplify our points. Some readers may want to refer to the Definition of Terms at the end of this paper, or to [1, 2, and 3] for detailed references describing function point analysis, linear programming, and operations research, respectively.

An algorithm we counted recently was one used to compute pay. This required solving a set of equations to calculate such pay subcomponents as base pay rate, holiday pay, temporary duty (travel) expenses, and foreign exchange rates. All subtotals were added to yield the final pay amount. Using the method of the International Function Point Users Group (IFPUG), beginning function point counters would probably recognize the input screen needed to enter a traveler's pay/expense data and count six or fewer function points. They would probably recognize the resulting earnings/expense statement as an external output and count it as seven or fewer function points. However, they may overlook other, more substantial functionality inherent in this algorithm.

Conditions for Sizing an Algorithm Using Function Points*

- An algorithm must represent a step-by-step procedure based on mathematical calculations and perhaps logic statements. It contains a set of equations that are executed according to business rules. The solutions to the equations are stored until later combined to produce an external output (EO) the user can recognize.
- It must be complete, solvable, feasible, and should not contain redundant constraints. (If it contains redundant constraints, they are considered nonunique and, therefore, not countable.)
- It must have a logical storage area to hold solutions to intermediate equation calculations until they are finally combined into a meaningful EO. This logical storage area is counted as one or more internal logical files (ILFs). We count the number of data element types (DETs) as the number of unique algorithm variables that must be populated plus the number of unique instances of DET control information that must be used to operate the algorithm. We also count the number of record elements types (RETs) if the algorithm contains logical subgroups of data and/or control information.
- The ILFs must be maintained by externally inputting values of variables and of stored control information; therefore, there must be at least one external input (EI) for each algorithm. We count one DET for each ILF variable maintained and one for each instance of maintained control information. The file types referenced (FTRs) by the EIs include the algorithm's ILFs.
- Data in the algorithm's ILF must be used to perform one or more intermediate calculations. The result of this calculation sequence must be recognizable by the user. There must be at least one EO in the application, which contains the DETs resulting from this calculation sequence. We count the ILFs of the algorithm when determining the number of EO FTRs; moreoever, we include the number of unique DETs output from the algorithm in the number of EO DETs.
- As an option, the application may have an external inquiry (EQ) capability to permit the user to view the values of the algorithm's variables and control information. If so, we count the number of viewable variables and control information in the EQ DET count and count the number of ILFs touched by the EQ process as the number of FTRs.
- We also recognize that algorithms may exert a greater degree of functionality influence than software without algorithms. The function point general system characteristics (GSCs) must therefore also be considered during the analysis. Examples include:
  - GSC 5, On-Line Data Entry, may be influenced if the algorithm's variables are populated on-line.
  - Since there must be at least one algorithm ILF updated during real-time processing, GSC 8, On-Line Update, may need to be considered.
  - GSC 9, Complex Processing, is likely to be a characteristic affected by complex algorithms. The function point counter should examine the algorithm for functionality such as extensive logical and/or mathematical processing and adjust the function point count accordingly.
  - GSC 14, Facilitate Change, may need to be examined—especially the fifth item, "Business control data is kept in tables that are maintained by the user with on-line interactive processes and the changes take effect immediately."

* All definitions and conditions are as defined by International Function Point Users Group.

Example: Linear Programming

Formulating the Problem

To illustrate the algorithm function point counting process, a linear programming algorithm can serve as an example. This is because linear programming fits the criteria for an algorithm as previously described. Linear programming is a repeatable, step-by-step process based on mathematical calculations and logic statements depending on how the problem is solved.

Formulating and solving a linear programming problem requires externally inputting and storing the values of certain variables and instances of control information until they are needed, to perform intermediate calculations according to a logical sequence, and to externally output the solution. It is complete, solvable, feasible, and should not contain redundant constraints when well formulated (if a formulation contains redundant constraints, these are not counted).

We illustrate by performing a function point analysis of the Joseph Ecker's and Michael Kupferschmid's Brewery Problem [3]. Microbrewers Inc. makes four products called Light, Dark, Ale, and Premium. These products are made using the resources of water, malt, hops, and yeast. Microbrewers Inc. has a free supply of water, so it is the amount of other resources that restricts production capacity. Table 1 shows how these resources are used to produce each corresponding gallon of beer, the available amount of resources, and the revenue realized from selling each type. The problem is to calculate the product mix to brew to maximize revenue.

	Light	Dark	Ale	Premium	Available (lb.)
Malt	1	1	0	3	50
Hops	2	1	2	1	150
Yeast	1	1	1	4	80
Revenue($)	6	5	3	7	

Table 1. *Pounds of Resource Needed to Produce 1 Gallon of Beer*

We can formulate this problem using the Simplex linear programming algorithm methodology from the field of Operations Research. Following this methodology, we define each of the products as follows:

- X1 – Gallons of Light.
- X2 – Gallons of Dark.
- X3 – Gallons of Ale.
- X4 – Gallons of Premium.

We can also define the objective function as: Max: 6X1 + 5X2 + 3X3 + 7X4.

The resources are constrained by the available amount of each resource. Therefore, the objective function is subject to the following:

- X1 + X2 + 3X4 <= 50 (Malt)
- 2X1 + X2 + 2X3 + X4 <= 150 (Hops).
- X1 + X2 + X3 + 4X4 <= 80 (Yeast).

Also, each variable must be greater than or equal to 0, or, X1, X2, X3, X4 >= 0.

In its traditional Simplex form, then, this algorithm appears as follows:

Max: 6X1 + 5X2 + 3X3 + 7X4

Subject to:

X1 + X2 + 3 X4 <= 50 (Malt)
2X1 + X2 + 2 X3 + X4 <= 150 (Hops)
X1 + X2 + X3 + 4X4 <= 80 (Yeast)
X1, X2, X3, X4 >= 0

This formulation of the linear program could be considered the primal formulation. However, every linear program can be formed in both a primal and a dual formulation and each method produces identical results. Suppose we set the following as the resource variables:

- Y1 – Pounds of Malt.
- Y2 – Pounds of Hops.
- Y3 – Pounds of Yeast.

Then, the dual can be formulated as follows:

Min: 50Y1 + 150Y2 + 80Y3
Subject to:
Y1 + 2Y2 + Y3 >= 6 (Light)
Y1 + Y2 + Y3 >= 5 (Dark)
2Y2 + Y3 >= 3 (Ale)
3Y1 + Y2 + 4Y3 >= 7 (Premium)
Y1, Y2, Y3 >= 0

Counting the Unadjusted Function Points

Solving linear programs is an algorithmic procedure because a set of equations is executed in a logical sequence to produce an EO. Reaching the solution using the graphical method requires constructing several constraint lines, determining the feasible region, and then finding the corner point of the feasible region that optimizes the objective function. Solving using the Simplex method requires building *tableaus* according to a certain procedure until the optimal solution is found. Each data element type (DET) must be stored in a logical storage area until it is needed.

In this example, the logical data storage area consists of one ILF. This ILF is the set of equation variables and control information containing the objective function and constraints. This meets the test of an ILF because it will be demonstrated that it is a logical, user-identifiable group of data or control information; and the group of data is maintained through an elementary process within the counted application boundary. ILFs have record element types (RETs) and DETs as components. Here is our approach for determining their number in this algorithm.

RETs

In a linear program, there are several logically distinct subgroups of data. The first is the objective function. It contains the control information telling us to either maximize or minimize. It also contains the coefficients for the decision variables that, in this example's primal formulation, indicate the revenue corresponding to each product. The other subgroups are represented by the constraint equations. Graphically, each constraint equation represents a unique set of data points in the plane (or space) that are feasible contributions to the solution of the linear program.

Either the primal or dual formulation of a linear program is mathematically sound. However, one may have fewer constraint equations. Extending the notion of the *elementary process* we always choose the smallest unit of activity meaningful to the user. In principle this means counting the formula-

tion containing the fewest constraint equations. In this example, it is the primal:

- We count five RETs in this algorithm.
- We count one RET for the optimization function.
- We count the minimum number of variables and number of constraints. This guarantees that the same number of RETs is counted regardless of the primal or dual formulation of the problem. This is three RETs since the primal formulation is the smaller unit of activity according to our reasoning.
- We count one RET for the nonnegativity constraint.

DETs

- We count 24 DETs in the algorithm.
- We count one DET for the maximum (or minimum) function, which is an instance of control information telling us how to optimize the objective function.
- We count one DET for each nonnegative variable or constraint. This is analogous to the number of variables in Table 1. In this example we count 18 such DETs.
- We count one DET for the number of variables and number of constraints in the primal or dual formulation used to count RETs, and we add one for the zero. This accounts for the nonnegativity condition. In this case there are five such DETs (X1, X2, X3, X4, and zero) for the nonnegativity constraint.

Solving linear programs is an algorithmic procedure because a set of equations is executed in a logical sequence to produce an external output.

The unadjusted function point count of this ILF of five RETs and 24 DETs is 10, and is therefore an *average* ILF.

To maintain the data in the ILF, we must use an EI. In this example, there is one FTR and there are 24 DETs (one for each data element plus one for minimum/maximum function). If this were a simple EI, there would probably be one further DET representing invoking the *enter* key to initiate the EI process and perhaps another DET if there were an associated error/confirmation message. This would be an average EI and would contribute four unadjusted function points.

The number of EOs will vary depending on the user requirements. Suppose the user wanted an output screen showing the optimal value of the objective function for this linear program, the optimal assignment for each of the variables, and the shadow price of each variable:

- We count one DET for the optimal value of the objective function, or one.
- We count one DET for each optimal variable assignment, or four.
- We count one DET for each shadow price or three.

To show all three of these aspects we count eight DETs. Since there is one FTR this would be a low EO of four unadjusted function points.

In this example, we count the total unadjusted function points as 18:

10 (ILF) + 4 (EI) + 4 (EO) = 18

Counting Large Algorithms

Some algorithms contain a few variables, like the preceding example; however, algorithms can contain many hundreds of variables. If the function point counter believes that there is more functionality inherent when counting these types of algorithms, then the counter may want to consider the super file rule (SFR).

A *super file* is defined as an ILF or ELF that contains more than 100 DETs if it contains multiple, countable RETs. If this is the case, each RET is considered a unique ILF (or EIF) and is counted as such. Although the SFR is not recognized by IFPUG, it is statistically significant. Some organizations informally adopt the SFR as part of their own local counting practice resolutions, and footnote their counting documentation accordingly.

Conclusion

An algorithm is a set of equations that are executed in a logical sequence to produce an external output. Algorithms appear in a variety of software applications. They can be used to help production planning in a brewery, perform many sets of calculations in sales applications, control nuclear reactors, schedule training, and determine the shortest route for telephone calls through a city telephone line network. The widely accepted IFPUG function point methodology can be used to count many algorithms. The paradigm described in this paper requires breaking down an algorithm into its functional components. These include its ILFs, EIs, and EOs. It also includes examining the corresponding GSCs. This paradigm is repeatable and reliable.

Recognizing and counting these kinds of algorithms is important. Their function point count helps quantify the work effort required to develop them that otherwise would have been overlooked. It also better portrays the size and complexity of the software. Finally, it helps quantify better cost and schedule forecasts, and can improve software quality measurement for overall software development.◆

References

1. International Function Point Users Group (IFPUG), *Function Point Counting Practices Manual Release 4.1.*, 1999
2. *Function Point Counting Practices Manual 3.4*, 1991.
3. Ecker, Joseph G. and Kupferschmid, Michael, *Introduction to Operations Research*, Malabar, Fla., Krieger Publishing Company, 1988.
4. Ibid., pp. 16-17.
5. Sedgewick, Robert, *Algorithms*, Addison-Wesley Publishing Co., Reading, Mass., 1983.

Additional Reading

- Garmus, David and Herron, David, *Measuring the Software Process: A Practical Guide to Functional Measurements,* Upper Saddle River, N.J., Prentice Hall PTR, 1996.
- Jones, Capers, *Applied Software Measurement: Assuring Productivity and Quality,* New York, McGraw-Hill, Inc., 1991.
- Monks, Joseph G. *Operations Management,* New York, McGraw-Hill Publishing Company, 1985.
- Shapiro, Roy D., *Optimization Models for Planning and Allocation: Text and Cases in Mathematical Programming,* New York: John Wiley & Sons, 1984.

Definition of Terms

Note: Some of these terms have two definitions. We have provided explanations in layman's terms. The italicized definitions are the precise ones from the IFPUG Counting Practices Manual.

Algorithm. An algorithm is the set of rules that must be completely expressed in order to solve a significant computational problem [4].

Application. This is a software package, such as a word processing, spreadsheet, or checkbook package.

Application User (simply referred to as "user"). A user is someone who needs a software application to perform his or her duties. For example, a user set might include data entry clerks, managers who need certain reports, customers who receive bills, system administrators who need to query the software's databases, et al. A user set does not normally refer to those whose role is software production such as programmers, database designers, or release managers; their role is to develop the software, not to use it after its market implementation.

Data Element Type (DET). Usually a DET is a field of data. It can also be an element of control information, such as the Enter key when it is needed to initiate the process of data input into an internal data file. In general, the more DETs in a function type (such as an external input), the higher its function point size. *"A unique, user-recognizable, nonrecursive field. The number of DETs is used to determine the complexity of each function type and the function type's contribution to the unadjusted function point count."*

Dual. This is a certain perspective of defining the resources available to reach the stated objective in linear programming. It is essentially a reflection of the primal perspective.

External Input (EI). EI is the process of adding, changing, and/or deleting data from an internal database. An example would be entering check numbers and amounts into a checkbook software package. An EI has three, four, or six unadjusted function points depending on whether it is of low, average, or high size/complexity. The textbook definition includes *"... processes data or control information that comes from outside the application's boundary. The external input itself is an elementary process. The processed data maintains one or more [internal logical files] ILFs. The processed control information may or may not maintain an ILF."*

External Inquiry (EQ). The process that allows the user to simply read or retrieve existing data from a database using certain criteria, much like an automated card catalog system in a public library. An EQ has three, four, or six unadjusted function points depending on whether it is of low, average, or high size and complexity. The textbook definition includes *"... an elementary process made up of an input-output combination that results in data retrieval. The output side contains no derived data. No ILF is maintained during processing."*

External Interface File (EIF). A database maintained in another application, but accessed by the application being counted on a read-only basis. An EIF has five, seven, or 10 unadjusted function points depending on whether it is of low, average, or high size/complexity. The textbook definition includes *"... a use-identifiable group of logically related data or control referenced by the application, but maintained within the boundary of another application. This means an EIF counted for an application must be an ILF in another application."*

External Output (EO). The process that yields a completed report, output file, or any other type of message set, which is sent to users. The report often contains data in fields that require calculations to derive. Examples could include credit card bills, completed spreadsheet reports, or state tax refunds. An EO has four, five, or seven unadjusted function points depending on whether it is of low, average, or high size and complexity. The textbook definition includes *"... is an elementary process that generates data or control information sent outside the application's boundary."*

Definition of Terms for this article is continued on page 30.

About the Authors

Nancy Redgate has a bachelor's degree in industrial engineering/operations research from the University of Massachusetts at Amherst. She received master's degrees in operations research, statistics, and business administration from Rensselaer Polytechnic Institute (RPI). She was the primary author of this paper, submitted as the requirement for an independent study in operations research at RPI.

PRI Automation
805 Middlesex Turnpike
Billerica, Mass. 01821
Voice: 978-670-4270
E-mail: nancy.redgate@prodigy.net

Charles B. Tichenor has a bachelor's degree in business administration from Ohio State University, a master's degree in business administration from Virginia Polytechnic and State University, and a doctorate degree in business from Berne University. He serves as an information technology operations research analyst for the Department of Defense, Defense Security Cooperation Agency. Tichenor holds part-time positions as a senior consultant for Development Support Center in Elm Grove, Wis., and as an adjunct faculty member at Strayer University's Anne Arundel, Md. campus. He served as technical advisor for this paper.

Defense Security Cooperation Agency
1111 Jefferson Davis Hwy., Suite 303
Arlington, Va. 22202-4306
Voice: 703-601-3746
Fax: 703-602-7836
E-mail: tichenor@erols.com
Internet: tichenor@erols.com