Measuring Calculus Integration Formulas Using Function Point Analysis

Nancy Redgate
American Express Risk Management

Dr. Charles Tichenor
Defense Security Cooperation Agency

Function point counters, software developers, and others occasionally need to measure the size and complexity of calculus integration formulas embedded in engineering and scientific applications. Sizing these formulas using function point analysis can result in more accurate measures of application size and improved quality in forecasting costs, schedule, and quality. It can also improve the confidence of those new to the function point methodology as they see that all of their calculus work is recognized and measured. This article shows an approach to sizing these formulas. This methodology is in full compliance with the International Function Point Users Group procedures and does not require any additional counting rules or patches.

Measuring the size and complexity of software is critical to the development of business models that accurately forecast the development cost, duration, and quality of future software applications. One way to measure software size is through function point analysis, especially using the methodology of the International Function Point Users Group (IFPUG) as published in their "Function Point Counting Practices Manual" (CPM) version 4.1 [1]. Readers unfamiliar with function point analysis are referred to the CPM as the primary reference for this methodology.

As good as the IFPUG function point methodology has been, it has historically had a perceived gap regarding the sizing of software algorithms, especially those embedded in real-time software. A proposed solution to the function point community was a general procedure to measure the size of those algorithms, which appeared in CROSSTALK February 2001 [2].

Since publishing that article, the authors were asked to focus this procedure for certain types of algorithms – calculus integration formulas – that can appear in engineering and scientific applications. This article, therefore, focuses the general procedure for sizing algorithms into a specific procedure for sizing calculus integration formulas. This is suitable for experienced function point counters who are also familiar with the fundamentals of calculus.

Description of an Algorithm

An algorithm is a series of equations solved in a logical sequence to produce an external output (EO). Real-time software sometimes contains embedded algorithms. Examples of these can include algorithms for controlling a nuclear reactor, calculating complex pricing agreements, or optimizing production levels in manufacturing. Calculus integration formulas fit this description of an algorithm because, as we will show, their solution requires solving a series of equations in a logical sequence before a solution can be reached.

Every algorithm must have one external input (EI), internal logical file (ILF), and EO with at least the following:

- Data and/or control information must be externally input into the algorithm.

"The authors suggest that ... [those] in mathematical academia or those with strong mathematical skill sets can further this theory."

- Data and/or control information must be logically stored. This storage area is not necessarily shown on an entity relationship diagram or other diagram depicting physical data files.
- The results of the algorithm's execution must be identifiable to the user as an EO.

Integration Formulas as Algorithms

Let us consider the following as an example for using function point analysis to measure integration formula size and complexity.

$$\int_{4}^{10} 6\,dx \qquad (1)$$

The solution of this formula can be expressed as the area under the curve f(x) = 6, as bounded by the x-axis, and within the domain four through 10. Figure 1 shows this graphically as the shaded area.

The Fundamental Theorem of Calculus gives us the algorithm to determine this area. The general formulation of the theorem is as follows.

$$\int_{a}^{b} f(x)\,dx = F(b) - F(a) \qquad (2)$$

In this example, this formulates as follows.

$$\int_{4}^{10} 6\,dx = F(10) - F(4) \qquad (3)$$

This algorithm can be solved graphically. First, calculate the area of the shaded region in Figure 2. This is the region bounded by the points (0,0), (10,0), (10,6), and (0,6). This is F(b), or F(10), and its area is 60 square units.

Then, calculate the area of the striped region in Figure 3. This is the region bounded by the points (0,0), (4,0), (4,6), and (0,6). This is F(a), or F(4) and its area is 24 square units.

Finally, subtract the striped region from the shaded region. This is F(b) - F(a), or F(10) - F(4). The remaining shaded region is shown in Figure 4. Its area is 60 - 24, or 36 square units.

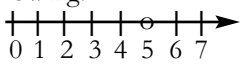
Measuring Size and Complexity

Counting the ILF

Sometimes an ILF is often identified with a physical data table. Here we need to be more specific. According to the IFPUG's CPM, an ILF is "... a user identifiable group of logically related data or control information maintained within the boundary of the application [1]." In mathematical terms, we suggest that an ILF could be described using the follow-

ing set theory: a set of data or control information maintained within the boundary of the application.

A mathematical set can be represented in several ways. For example, we might represent the set of *all numbers greater than five* as the following:

- {all numbers greater than five}, or
- using a number line such as the following:
- 

In this example, therefore, we have an ILF, which is the set of data points in the Cartesian plane required to solve this equation using the Fundamental Theorem of Calculus.

Sometimes ILFs have subgroups of data. These are called record element types (RETs). According to the CPM [1], a RET is a "user recognizable subgroup of data elements within an ILF ..." Using set theory, we could describe an RET as *a subset of data*.

Each ILF is measured by considering the number of data element types (DETs) it logically contains and the number of RETs it contains. A complexity matrix in the CPM then shows how to convert the combination of counted DETs and RETs into function points.

Counting the RETs

The Fundamental Theorem of Calculus states that the set of points (or the area) in the Cartesian plane, which represent the solution to this integration formula, is found by subtracting the area of the striped region of Figure 3 from the area of shaded region of Figure 2. This resulting region has the area shown in Figure 4.

For this example, this entire region has three subsets. The first subset of all points in the Cartesian plane affected by this algorithm is the area of Figure 2, that is, F(b), or F(10). The second subset is the area in Figure 3: F(a), or F(4). The third subset is the area in Figure 4: F(b) - F(a), or F(10) - F(4). This is the solution subset. Therefore, this integration formula has one ILF, with three RETs.

Counting the DETs

The integration formula tells us how to partition the Cartesian plane into the areas F(b) and F(a). Let us recall the formula.

$$\int_{4}^{10} 6\,dx \qquad (4)$$

The ILF is the region of the Cartesian plane bounded by the f(x) on the *north* side, the x-axis on the *south* side, x = 10 on the *east* side, and x = 0 on the *west* side. The solution RET is bounded by x = 4 on the west side. This gives us the information we need to count this formula's DETs.

- The first DET is f(x), the northern boundary of the ILF. In this case, the DET is six.
- The second DET is dx, which tells us that the integration is with respect to x and is therefore bounded in this case by the x-axis on the southern side.
- The third DET is 10, which gives us the "b" in F(b).
- The fourth DET is four, which gives us the "a" in F(a).

Therefore, this ILF has three RETs and four DETs. According to the CPM complexity matrix, this is a low complexity ILF and is worth seven function points.

To generalize this method, we suggest the following key points shown in Figure 5.

- The shaded area on this graph represents an ILF. It is "... a user identifiable group of logically related data or control information maintained within the boundary of the application."
- The RETs are the "blocks" or areas needed to graphically depict F(b), F(a), and F(b) - F(a).
- The number of DETs represents the instances of data and/or control information to define the solution area F(b) - F(a).
- This ILF does not appear in an ER diagram.

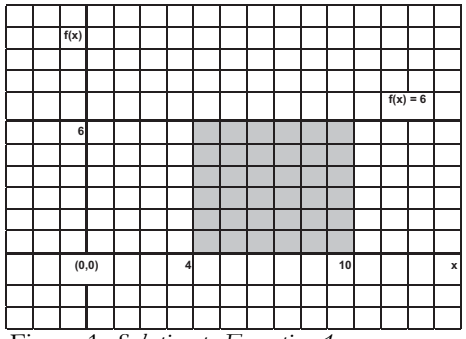


Figure 1: *Solution to Equation 1*

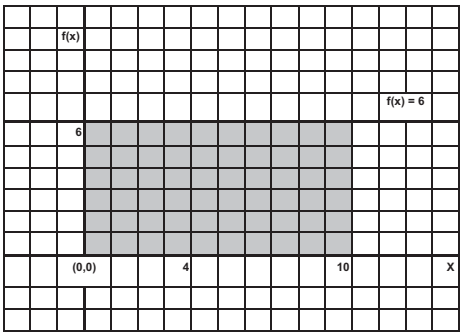


Figure 2: *F(b)*

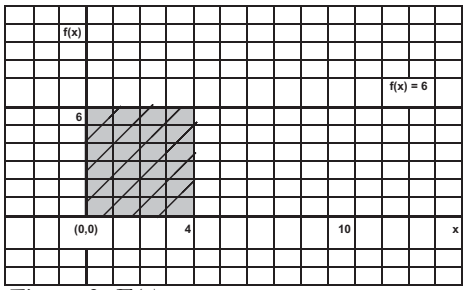


Figure 3: *F(a)*

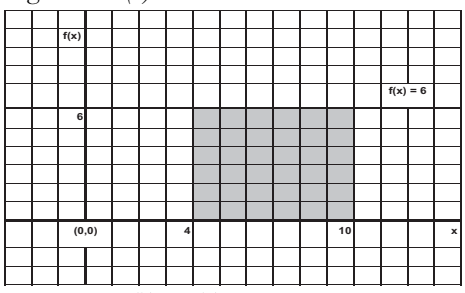


Figure 4: *F(b) - F(a)*

Counting the EI

One can imagine a simple version of an input screen having a layout as follows:

f(x) = ☐
dx or dy ☐
b = ☐
a = ☐
ENTER

There are four DETs as stated, plus the control information ENTER key to affect the input process. There is one ILF being referenced (File Types Referenced [FTR]), which is the graph of Figure 5. The CPM complexity matrix shows that an EI with four DETs and one ILF is a Low complexity EI worth three function points.

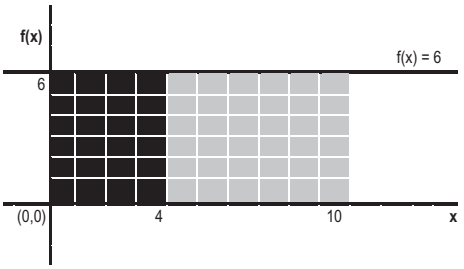
Counting the EO

Imagine a simple report with the solution of the integration formula calculations – perhaps a paper report that looks like this.

The solution is ☐

In this case, there is one DET on the report, and one FTR – the graph of Figure 5. Therefore, the CPM complexity matrix is used to show that this is a Low EO worth four function points.

Figure 5: *Equation 1's Internal Logic File*



Coming Events

July 18-20
Shareware Industry Conference
St. Louis, MO
www.sic.org

July 22-25
Joint Advanced Weapons Systems Sensors, Simulation, and Support Symposium (JAWS S3)
Colorado Springs, CO
www.jawswg.hill.af.mil

July 22-26
6th Annual PSM Users' Group Conference
Keystone, CO
wwwpsmsc.com

August 19-22
The 2nd Software Product Line Conference
San Diego, CA
www.sei.cmu.edu/SPLC2/

September 9-13
International Conference on Practical Software Quality Techniques (PSQT) 2002 North and International Conference on Practical Software Testing Techniques (PSTT) 2002 North
St. Paul, MN
www.psqtconference.com

November 18-21
International Conference on Software Process Improvement
Washington, DC
www.software-process-institute.com

April 28-May 1, 2003
Software Technology Conference 2003

Salt Lake City, UT
www.stc-online.org

Other Examples and Their Solutions

Consider taking a function point count of the following integration formula.

$$\int_4^{10} x^2\,dx \qquad (5)$$

In principle, this formula's ILF is counted the same way as the previous example. There are four DETs (x^2, dx, 10, and 4). There are three RETs in the ILF (F(10), F(4), and F(10) - F(4)). This is a low complexity ILF, worth seven function points. We assume that the EI input screen and EO are similar to the first example, so the total unadjusted function point count is 14.

Consider a slightly different formula.

$$\int_4^{10} (x^2 - x)\,dx \qquad (6)$$

We count the function points using the same logic.

The solution to this formula requires us to break this into two parts. We first use the Fundamental Theorem of Calculus to find the following.

$$\int_4^{10} (x^2)\,dx \qquad (7)$$

Then we subtract from it the solution to the second part.

$$\int_4^{10} (x)\,dx \qquad (8)$$

This ILF therefore contains six RETs – three regions from the first part and three regions from the second part. There are four DETs needed to solve the first part (x^2, dx, 10, and 4) and four to solve the second part (x, dx, 10, and 4). Therefore, this ILF of six RETs and eight DETs is an average complexity ILF worth 10 function points.

One point needs to be clarified here for the advanced function point counter. Even though the dx, 10, and four might appear to be counted twice in this ILF, this does not violate the CPM rule that each DET must be unique. They are actually unique here because the Fundamental Theorem of Calculus must be executed twice – once for each part. Each instance of dx, 10, and four must be used in order to solve the formula.

Finally, consider this formula.

$$\int_4^{10} 3x\,dx \qquad (9)$$

The solution to this requires first simplifying to the following.

$$3\int_4^{10} x\,dx \qquad (10)$$

This formula's ILF still has its three RETs, but its DET count increases from four to five. This is because the solution area of the following is multiplied by three.

$$\int_4^{10} x\,dx \qquad (11)$$

Further Notes for Function Point Counters

The above examples were intended to be simplistic to illustrate the procedure. In more complex cases, we might want to consider whether, for example, there is an add, change, and delete capability associated with the EI. If so, count up to three EIs accordingly (one for the add capability, one for the change capability, and one for the delete capability.) The EO may be more complex, or several EOs might be required. There might be an EQ capability to view what data and control information is currently in the ILF. Finally, an external interface file could be involved.

If an integration formula can be formulated in several ways, extend the concept of the elementary process and choose the smallest unit of activity meaningful to the user. For example, perhaps choose the ILF formulation having the smallest number of RETs.

Algorithms may influence the general systems characteristics. You may need to check, for example, the following:

- GSC5 Online Data Entry.
- GSC8 Online Update.
- GSC9 Complex Processing.
- GSC14 Facilitate Change.

Areas for Future Research

The authors suggest that readers in mathematical academia or those with strong mathematical skill sets can further this theory. The authors suggest that a variety of integration techniques can be counted and that research should be conducted to expand this methodology.

Conclusion

Function point analysis can be used to measure the size and complexity of algorithms in general, and integration formulas in particular. Advanced function point counters need to recognize all algorithms in the software they count, to include

those embedded integration formulas. The IFPUG function point methodology can be used to measure the size and complexity of these integration formulas without the need for additional patches or counting rules.

Reference

1. International Function Point Users Group. Function Point Counting Practices Manual Release 4.1, 1999. Refer to <www.ifpug.org/publications/manual.htm>.
2. Redgate, Nancy, and Charles B. Tichenor. "Measure Size, Complexity of Algorithms Using Function Points." CROSSTALK Feb. 2001: 12-15.

Additional Reading

1. International Function Point Users Group. "Function Points as Assets." (Provides direction for using function point analysis to develop cost, schedule, and quality forecasts.)
2. Garmus, David, and David Herron. Measuring the Software Process: A Practical Guide to Functional Measurements. Upper Saddle River, N.J.: Prentice Hall PTR, 1996.
3. Jones, Capers. Applied Software Measurement: Assuring Productivity and Quality. New York: McGraw-Hill, 1991.

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.

5 Wood Hallow Drive
34-02-01
Parsippany, NJ 07054
Phone: (973) 526-6602
Fax: (973) 526-3635
E-mail: nancy.redgate@prodigy.net

Charles B. Tichenor, Ph.D., serves as an information technology operations research analyst for the Department of Defense, Defense Security Cooperation Agency. Dr. Tichenor holds a part-time position as an adjunct faculty member at Strayer University's Anne Arundel, Md. campus. He 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.

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

WEB SITES

Software Cost Estimation Web Site

www.ecfc.u-net.com/cost/index.htm

The Software Cost Estimation Web site presents a review of current cost estimation techniques to help industry and academia choose the appropriate methods when preparing software cost estimates. The site covers both traditional and state-of-the-art methods identifying advantages and disadvantages of each and the underlying aspects in preparing cost estimates. The site also provides links to other software cost estimation sites that are involved in this area and details the research that has been undertaken at Bournemouth University.

International Function Point Users Group

www.ifpug.org

The International Function Point Users' Group (IFPUG) is a non-profit organization committed to increasing the effectiveness of its members' information technology environments through the application of function point analysis (FPA) and other software measurement techniques. IFPUG endorses FPA as its standard methodology for software sizing and maintains the "Function Point Counting Practices Manual," the recognized industry standard for FPA.

Practical Software and Systems Measurement Support Center

www.psmsc.com

The Practical Software and Systems Measurement (PSM) Support Center is sponsored by the Department of Defense (DoD) and the U.S. Army. It provides project managers with the objective information needed to successfully meet cost, schedule, and technical objectives on programs. PSM is based on actual measurement experience with DoD, government, and industry programs. The Web site also has the most current version of the PSM Guidebook.

The Software Productivity Consortium

www.software.org/default.asp

The Software Productivity Consortium is a nonprofit partnership of industry, government, and academia. It develops processes, methods, tools, and supporting services to help members and affiliates build high-quality, component-based systems, and continuously advance their systems and software engineering maturity pursuant to the guidelines of all of the major process and quality frameworks. Its Technical Program builds on current best practices and information technologies to create project-ready processes, methods, training, tools, and supporting services for systems and software development.