Measuring Software Size, Without Function Points

Raymond E Boehm

Software Composition Technologies

Abstract- IFPUG function points have long been the principle method for measuring the functionality of a system. There are other methods. Use case points have developed a following in agile circles. This method is described in some detail. Users of COCOMO II have been exposed to application/object points. There have been other, more object oriented measures proposed. Web objects and internet points have been used in that domain. These measures are compared and contrasted.

Introduction

One of the principle methods of measuring software size is through the use of functional sizing measures. As the name implies, functional measures are based on the functionality of an application.

This functionality is usually evaluated in logical terms. For example, adding a customer is a piece of functionality. It may be irrelevant that it is accomplished with two screens. Other technical aspects of the application, such as the quality of the user interface, may also be irrelevant. The manner that the application is developed is almost always considered irrelevant.

Function points are the best known example of functional measures. Some people would further restrict this to unadjusted function points. In any case, this presentation introduces other functional measurements.

Use Case Points

About 20 years after Alan Albrecht introduced the notion of function points, Gustav Karner described a measure called use case points. It, too, was intended to be an estimating technique.

Use case points were strongly influenced by the work of Ivar Jacobson and other object oriented methodologists.[8] The technique is primarily driven by the actors and use cases identified for the application. Following the steps under Actor Weight and Use Case Weight, below, will yield unadjusted weights for both. Adding these weights together will yield the number of unadjusted use case points.

The unadjusted use case points are multiplied by technical and environmental weights. This yields the total number of use case points. The significance and method of establishing these weights is described below, under Technical Complexity and Environmental Complexity,.

There is a tool that automatically calculates the use case points from descriptions of the actors and use cases.[9] The tool does not always match the value arrived at by human experts. In addition, the use cases must be written in Japanese. There are other tools that require the user to evaluate the complexity but that automate the calculations. In any case, the existence of these tools is an indication of the level of interest that exists regarding this technique.

Like function points, use case points were originally developed for estimating. Originally, a use case point was thought to take 30 hours to implement. Later studies have changed this number or made it a function of additional cost drivers.

Benta Anda has conducted several studies comparing the accuracy of use case point based estimates with actual results and with estimates generated by experts.[2] The use case point based estimates were fairly close to the actual development effort. They were usually closer than the estimates presented by the experts.

Actor Weight

Use the following criteria to assign a complexity and weight to each of the actors:

- A simple actor might be another application that accesses this application through an API. Its weight is 5.
- An average actor might be a user accessing the application through a text-based user interface. Its weight is 10.
- A complex actor might access the application through a graphical user interface. Its weight is 15.

These weights are summed to arrive at the unadjusted actor weight.

Use Case Weight

Evaluating the use cases requires a fair amount of knowledge about use cases. A course on use cases is beyond the scope of this article. However, the calculation of use case weight will be described for those who are interested.

Each use case consists of one or more transactions. Each step in the main success scenario is a transaction. Some extensions are also transactions;

those that are a continuation of another transaction are not counted.

Use (Table 1) to assign weights to each use case based on their complexity. The complexity is established by the number of transactions. Sum the weights to arrive at the unadjusted use case weight.

Table 1. Use Case Weights

Complexity	Number of transactions	Weight
Simple	3 or less	1
Average	4 to 7	2
Complex	7 or more	3

Technical Complexity

The way that use cases are implemented have an impact on the cost, and therefore on the use case points. For example, an application that is designed to be portable between several different platforms will probably take longer to develop than one that only works on one platform. This would be the case even though the actors and use cases were exactly the same.

Take each of the attributes in (Table 2) and assign a value between 0 (for no impact) and 5 (for very high impact). Multiply that value by the weight and sum up the values. Multiply the sum by .01 and add .6 to get the technical complexity factor.

Table 2. Technical Complexity

Factor	Desription	Weight
T1	Distributed system	2
T2	Performance objectives	2
T3	End-user efficiency	1
T4	Complex processing	1
T5	Resuable code	1
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portable	2
T9	Easy to change	1
T10	Concurrent use	1
T11	Security	1
T12	Access for third parties	1
T13	Training needs	1

Environmental Complexity

The team that performs the implementation obviously has an impact on the cost. For example, an application development team that is familiar with the development process would be able to perform that implementation more quickly. The environmental complexity factor accounts for this.

Take each of the attributes in (Table 3) and assign a value between 0 (not the case) and 5 (very much the case). Multiply that value by the weight and sum up the values. Multiply the sum by -.03 and add 1.4 to get the environmental complexity factor.

Table 3. Environmental Complexity

Factor	Desription	Weight
E1	Familiar with the development process	1.5
E2	Application experience	0.5
E3	Object-oriented experience	1
E4	Lead analyst capability	0.5
E5	Motivation	1
E6	Stable requirements	2
E7	Part-time staff	-1
E8	Difficult programming language	-1

Application/Object points

Object points were originally introduced as a sizing measure for ICASE environments.[4] The objects had nothing to do with object oriented development. They were work items, like screens and reports, which CASE tools might produce.

Barry Boehm made them part of the COCOMO II model. They are used to estimate software size in an ICASE environment where development is done through application composition. He renamed them application points so as not to confuse the object oriented development community into thinking this measure was for them

A good description of counting application points is from a presentation given by Barry Boehm.[3] It shown in Figure 1. Note that it still refers to object points instead of the newer name, application points. Also, the procedure recognizes a difference in the source of data tables: srvr tables are on the mainframe, while clnt tables are on the personal workstation.

Object Oriented Measures

It has been observed that in object oriented systems development, there is a "natural correspondence between specification and implementation."[10] Because of this, it can be argued that object oriented size measures are functional by their very nature.

The CK metrics suite is one of the most commonly quoted sets of object oriented measures. One of the principle sizing measures is Weighted Modules per Class (WMC).[7] There are several variants on this measure.

Figure 1. Application Point Estimation Procedure

Step 1: Assess Object-Counts: estimate the number of screens, reports, and 3GL components that will comprise this application. Assume the standard definitions of these objects in your ICASE environment.

Step 2: Classify each object instance into simple, medium and difficult complexity levels depending on values of characteristic dimensions. Use the following scheme:

For Screens				For Reports			
# and source of data tables				# and source of data tables			
Number of Views Contained	Total < 4 (<2 srvr, <3 clnt)	Total <8 (<3 srvr, 3 - 5 clnt)	Total 8+ (>3 srvr, >5 clnt)	Number of Sections Contained	Total < 4 (<2 srvr, <3 clnt)	Total <8 (<3 srvr, 3 - 5 clnt)	Total 8+ (>3 srvr, >5 clnt)
<3	simple	simple	medium	0 or 1	simple	simple	medium
3-7	simple	medium	difficult	2 or 3	simple	medium	difficult
>8	medium	difficult	difficult	4+	medium	difficult	difficult

Step 3: Weigh the number in each cell using the following scheme. The weights reflect the relative effort required to implement an instance of that complexity level.

Object Type	Complexity-Weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL Component			10

Step 4: Determine Object-Points: add all the weighted object instances to get one number, the Object-Point count.

Step 5: Estimate percentage of reuse you expect to be achieved in this project. Compute the New Object Points to be developed *NOP =(Object-Points) (100-%reuse) / 100.*

By itself, WMC has been useful in predicting maintenance and testing effort. It has also been used as a component of such measures as Predictive Object Points, Object-oriented Function Points. These measures have evolved in another measure called Class Points.[6]

Class Points are a Function Point-like measure conceived to estimate the size of object oriented software. The class point calculation worksheet, shown in Figure 2. Function point counters will find that this worksheet is very similar to ones that they use in their work.

While function point counters begin by identifying data and transactions, use case point counters begin by classifying their user classes into one of the following types:

- Problem Domain containing entities in the application domain of the system,
- Human Interaction containing screen related widgets,
- Data Management containing data handling functionality, and
- Task Management, responsible for communication between subsystems.

Just like function point analysis, the next step is to assign complexity to each of these components. Like function points, there are tables that assign low, average and high complexity based on certain attributes. Unlike function points, use case points allows for two different levels of detail.

The first method, CP_1, is intended for use early in the life cycle. It assigns complexity based on the number of external methods and the number of services required. The assignment is made through the use of a simple three-by-three table that function point counters are familiar with.

The second method, CP_2, is intended for use later in the life cycle. It assigns complexity based on the number of attributes, in addition to the same items as in CP_1. This assignment is made through the use of three different tables. Each table is set up for a different range of services requested.

The technical complexity factor is a function of 18 system characteristics. The first 14 are the same as those used in function point analysis. The four new ones are:

1. User Adaptivity provided,
2. Rapid Prototyping required,
3. Multiuser Interactivity supported, and
4. Multiple Interfaces for different users.

The guidelines for these four new characteristics are in the paper that has been referenced. The calculation is what would be expected for 18 characteristics, as opposed to 14.

Figure 2. Class Point Calculation Worksheet

Application: _____ Appl. ID: _____

Prepared by: _____ __/__/__

Reviewed by: _____ __/__/__

- Class Point Count:

System Component Type	Description	Complexity			
		Low	Average	High	Total
PDT	Problem Domain	…* 3 =…	…* 6 =…	…* 10 =…	…
HIT	Human Interaction	…* 4 =…	…* 7 =…	…* 12 =…	…
DMT	Data Management	…* 5 =…	…* 8 =…	…* 13 =…	…
TMT	Task Management	…* 4 =…	…* 6 =…	…* 9 =…	…
TUCP		*Total Unadjusted Class Point*			

- Processing Complexity:

ID	System Characteristic	DI	ID	System Characteristic	DI
C1	Data Communications	…	C10	Reusability	…
C2	Distributed Functions	…	C11	Installation ease	…
C3	Performance	…	C12	Operational ease	…
C4	Heavily used configuration	…	C13	Multiple sites	…
C5	Transaction rate	…	C14	Facilitation of change	…
C6	Online data entry	…	C15	User Adaptivity	…
C7	End-user efficiency	…	C16	Rapid Prototyping	…
C8	Online update	…	C17	Multiuser Interactivity	…
C9	Complex processing	…	C18	Multiple Interfaces	…
TDI				**Total Degree of Influence**	…

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

TCF Technical Complexity Factor = 0.55 + (0.01 * TDI) = _____

CP Class Point Measure = TUCP * TCF = _____

Web Related Measures

There are a few measures specifically designed for use on web applications but the best known are web-points, web objects and internet points.

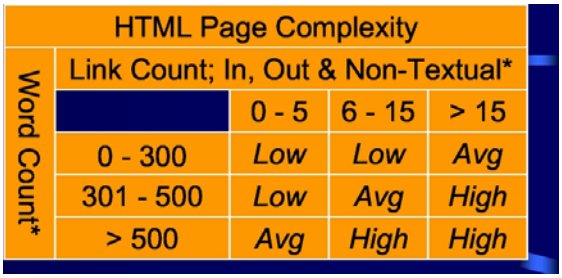
Web-Points

Web-points were developed by David Cleary of Charismatek.[5] They were designed to measure static web sites. They measure the size of the HTML pages. They are not designed to capture the size or effort of other content development, such as the preparation of a movie for the site.

The first step in calculating web points is to use the assign each static page a complexity based on Figure 2. It is from Cleary's presentation. It show how to assign complexity based on the word count and the following three type of links:

1. Links into the web site,
2. Links out of the web site, and
3. Links to pictures, movies, etc.

Figure 3. Web-Points Page Complexity



The next step is to calculate the weighted sum of all of the static web pages in the site. Use the weightings in Table 4. Remember that the web-points developed by this calculation is only for the static portion of a web site. Conventional function points are advocated for what Cleary calls "Information System-Structured Web-Sites" or portions of web sites.

Table 4. Web-Points Weighting

Complexity	*Web-Points*
Low	*4*
Average	*6*
High	*7*

Web Objects

In 2000, Don Reifer published a description of web objects.[11] At that time, he identified nine web object predictors. He suggested the use of object or application points as one of them. He based his calculations on Halstead's Software Science volume equation.

About a year later, he posted a update to this method as a white paper on his web site (www.reifer.com). The paper showed the use of function points as the primary predictor. It was augmented by the following four types of objects:

1. Multi-media files,
2. Web building blocks,
3. Scripts, and
4. Links (XML, HTML and query language lines).

There are some counting conventions in the white paper. The conventions and a weighting table basically hide the Halstead equation. A COCOMO-like model called WEBMO is used to estimate schedule and effort from the web object count.

Internet Points

Internet points are widely mentioned, but not as often described as the other web related measures. Silvia Abrahao and her co-authors explains that for internet points, "a web site is sized by counting seven types of functions: files, RDB tables, APIs, messages sent by the system, number of static HTML pages, number of dynamic HTML pages and number of interactive pages."[1] The technique is used by the Cost Xpert estimating tool. The company's web site (www.costxpert.com) is probably the best source of additional information.

Comparing and Contrasting Measures

It has been claimed that both use case points and application points have a time benefits over function points. They each require less training than function point counting. They are also faster to apply.

In the case use case points, both time savings are really displacements of effort. If a person has already spent the time to learn about use cases, learning use case points requires minimal effort. Otherwise, it is probably about as involved as learning function points. The same is true of applying the technique. If the use cases are available, calculating use case points is relatively quick. However, developing use cases for requirements expressed in another fashion may take longer than performing a function point count.

Calculating application points is both easier to learn and faster to apply than function points. Unfortunately, their relationship to the ICASE environment means they are not standard. Few organizations have a single enterprise wide ICASE environment. Furthermore, ICASE tools often change over time. Thus, the application points are probably not consistent between organizations or over time in the same organization.

Various object oriented techniques have similar advantages and disadvantages of use case points. However, the biggest disadvantage is that these techniques are not widely practiced. It will be some time before any object oriented technique enters the mainstream.

The web related measures are mostly extensions to function points. Web-points provide a separate measure of static web pages. For most real web sites, function point must still be used to measure the dynamic portions. Web objects are similar, but they result in a blended measure. Internet points seem to have little following beside users of the Cost Xpert estimating tool.

References

[1] S. Abrahao, G. Poels, and O. Pastor, "Evaluating a Function Size Measurement Method for Web Applications: An Empirical Analysis," presented at 10th International Symposium on Software metrics (METRICS'04), 2004.

[2] B. Anda, "Empirical Studies of Construction and Application of Use Case Models," in *Department of Informatics*: University of Oslo, 2003.

[3] B. Boehm, "COCOMO II Overview," presented at 14th International COCOMO Forum, 1999.

[4] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, *Software Cost Estimation With COCOMO iI*. Upper Saddle River, New Jersey: Prentice-Hall, Inc., 2000.

[5] D. Cleary, "Web-Based Development and Function Size Measurement," presented at IFPUG Annual Conference, San Diego, CA, 2000.

[6] G. Costagliola, F. Ferrucci, G. Tortora, and G. Vitiello, "Class Point: An Approach for the Size Estimation of Object Oriented Systems," *IEEE Transactions on Software Engineering*, vol. 31, pp. 52-74, 2005.

[7] D. P. Darcy and C. F. Kemerer, "OO Metrics in Practice," *IEEE Software*, vol. 22, pp. 17-19, 2005.

[8] I. Jocobson, M. Christerson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach.* Wokingham, England: Addison-Wesley, 1992.

[9] S. Kusumoto, F. Matukawa, K. Inoue, S. Hanabusa, and Y. Maegawa, "Estimating Effort by Use Case Points: Method, Tool and Case Study," presented at 10th International Symbosium on Software Metrics, Chicago, Illinois, 2004.

[10] L. A. Laranjeira, "Software Size Estimation of Object-Oriented Systems," *IEEE Transactions on Software Engineering*, vol. 16, pp. 510-522, 1990.

[11] D. J. Reifer, "Web Development: Estimating Quick-To-Market Software," *IEEE Software*, vol. 17, pp. 57-64, 2000.

About the Author

Raymond Boehm is Software Composition Technologies's principal consultant. He is an IFPUG CFPS, a QAI CSQA and a member of the ACM and the IEEE. Contact him at rayboehm@softcomptech.com.