How to Determine Your Application Size Using Function Points

By: Alvin Alexander

Abstract: This session describes the technique of counting function points to determine application size, including presentation of basic counting rules, a sample count, and time/cost estimating tips and tricks.

Alvin Alexander is a managing partner of Mission Data, a software development firm in Louisville, Ky. Al, who has been a consultant to many Fortune 1000 firms, is a trained aerospace engineer, a former author for Inside JBuilder, the creator of devdaily.com, and has been an entertaining instructor and lecturer since 1993.

ala@missiondata.com

Introduction

Objectives of Function Point Analysis

What is a "Function Point"?

Important introductory FPA notes

Other useful information

A brief history

The benefits of Function Point Analysis

Overview

A simple five step counting process

Five standard "functions"

Details on the Five Data and Transactional Functions

(1) Data Functions -- Internal Logical Files (ILFs)

(2) Data Functions -- External Interface Files (EIFs)

(3) Transaction Functions -- External Inputs (EI’s)

(4) Transaction Functions -- External Outputs (EO’s)

(5) Transaction Functions -- External Inquiries (EQ’s)

Summary

A sample count

An introduction to our example

The counting process

Step 1: The type of count

Step 2: Identify the scope and boundary of the count

Step 3: Determine the unadjusted function point count

Step 3a: Determine the count resulting from ILF’s

Step 3b: Determine the count resulting from EIF’s

Step 3c: Determine the count resulting from EI’s

Step 3d: Determine the count resulting from EO’s

Step 3e: Determine the count resulting from EQ’s

Step 4: Determine the Value Adjustment Factor (VAF)

Step 5: Calculate the Adjusted Function Point Count

The Value of Counting Function Points

Bonus - Application development repositories and two magic formulas

FP repositories

Three magic formulas

Summary

Introduction

Although Function Points (FPs) were introduced by Alan Albrecht of IBM 25 years ago, they remain largely unknown by most developers today. For instance, I personally know over 120 software developers at 20 different companies, and nobody had ever heard of Function Points before I did. The closest I ever came was earlier this year, when a friend started telling me that he really wished there was some good metric for determining the size of software applications, because he sure was tired of making up estimates all the time. (Actually, I think he referred to this process as "pulling an estimate out of the air"). He was looking at something called Use Case Points, which is a very non-formal, non-standard approach I had tossed aside years ago. He was trying to get this to work, and I suggested that he take a look at FPs instead.

As I’ve learned over the last 18-24 months, being aware of FPs and what they can add to your overall understanding of the software development process is very important. Understanding FPA helps take the fear out of tasks like software development cost estimating. Additionally, it can also serve as an aid in measuring and improving product quality as you begin to understand critical metrics such as project velocity and defect ratio.

Objectives of Function Point Analysis

Most practitioners of Function Point Analysis (FPA) will probably agree that there are three main objectives within the process of FPA:

1. Measure software by quantifying the functionality requested by and provided to the customer.

2. Measure software development and maintenance independently of technology used for implementation.

3. Measure software development and maintenance consistently across all projects and organizations.

In working towards objectives 2 and 3 above, several organizations have created large repositories of FP counts that cross projects, technologies, and organizations. These repositories can be an invaluable tool for your first estimation efforts, because it lets you compare your project to similar projects that have been developed by other organizations around the world.

What is a "Function Point"?

One of the first questions I'm always asked is a logical one: "What is a function point?"

Simply stated, function points are a standard unit of measure that represent the functional size of a software application. In the same way that a house is measured by the square feet it provides, the size of an application can be measured by the number of function points it delivers to the users of the application.
A good example is when I had my house built two years ago. I worked with a very straightforward home builder and he basically said "Al, you have two choices here. First, how many square feet do you want to build? Second, what quality of materials do you want to use?" He continued "Let's say that you want to build a house that is 2,000 square feet. If you want to use cheap materials we can build it for $80 per square foot. That's $160,000. If you want to go to top of the line then you're looking at more like $110 per square foot, and that's $220,000. What would you like?"

Don't read into this example that building a software application is like building a house. We all know that there are a lot of other variables, and it's not quite this simple. But function points do get you a lot closer. For example, although it's not like building a house, I'm currently working on a Java software development project where we are building a Swing-based application that runs on Windows and Mac computer platforms. We're building this application on a fixed-price basis, at a rate of approximately $250/FP. So, is it like building a house? No. But can I bid projects on a fixed-price basis now? Yes, I can. And I couldn't do that 18 months ago.

### Important introductory FPA notes

There are several other important notes about the FPA process that need to be introduced at this time, so I'm including them here:

- Measured from the user's perspective
- The size of the application being measured is based on the user's view of the system. It is based on what the user asked for, not what is delivered. It's based on the way the user interacts with the system, including the screens that the user uses to enter input, and the reports the users receive as output. Finally, it's also based on their understanding of the data that needs to be stored and processed by the system.
- Technology-independent
- As mentioned in the objectives section, FPA is also technology-neutral. As a Certified Function Point Specialist (CFPS) it does not matter to me what technology you are using to implement your application. It doesn't matter if it's a Web application written in Java, PHP, ColdFusion, or .Net; or a client-server app written in Delphi, VB; or even an AS/400 RPG application. Just show me your screens and your data tables and I'll derive "number of function points" from there.
- Low cost
- Adding FPA to your software development portfolio is also very easy. Historically, adding the process of counting FPs to your development process results in a cost increase of only 1%
- Repeatable
- Studies have shown that multiple function point counters can independently count the same application to within 10% accuracy of each other. Repeatability is very important, because without it we could not begin to trust the data from the hundreds of applications that are stored in repositories around the world.
- Work well with use cases
- This process works extremely well with use cases, and can even work with the concept of "stories" in Extreme Programming.

### Other useful information

Before we get into the practice of counting FPs, there are a few other background points you should also know:

#### Large user group

A large user group known as IFPUG ([http://ifpug.org](http://ifpug.org)) is responsible for carrying the FP torch. IFPUG is a non-profit, member-governed organization, consisting of over 1,200 members in 30 countries around the world. As the time of this writing version 4.2 of the IFPUG specifications for counting FPs (referred to as the Counting Practices Manual) has just been released.

#### ISO Standard

The "Unadjusted FP Count" of IFPUG v4.1 is now an ISO standard. In this paper you'll learn some of the basics of performing an Unadjusted FP Count.

#### De-facto standard

In addition to being an ISO standard, FPs are used as the de facto standard for cost estimating applications like Cocomo II, Construx Estimate, and other estimating packages.

#### Certified Function Point Specialist, or CFPS

A CFPS is a person who has passed the official IFPUG certification test. The CFPS designation must be renewed every three years.

#### Counting Practices Manual, or CPM

The CPM is the official manual created and distributed by IFPUG. It details the official counting rules used by CFPS practitioners. These rules help to keep counts consistent from one CFPS to another. Version 4.1 of this manual is over 300 pages in length.

#### FP data repositories

Because many companies have been using FP information for quite some time, there are several large repositories of project data, where companies have combined FP counts along with other information, such as tools used, man hours, and overall project cost. With accurate counts and other accurate data, you don't have to feel so alone when making those all-important project estimates.

#### A brief history

The following table shows a brief history of function points, beginning with the introduction of the concept by Alan Albrecht in 1979.

<table>
<thead>
<tr>
<th>Year</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1979</td>
<td>FPs introduced by Alan Albrecht</td>
</tr>
<tr>
<td>1984</td>
<td>First FP guidelines</td>
</tr>
<tr>
<td>1986</td>
<td>First IFPUG Board of Directors</td>
</tr>
<tr>
<td>1994</td>
<td>CPM Release 4.0</td>
</tr>
<tr>
<td>2003</td>
<td>ISO standard</td>
</tr>
</tbody>
</table>

| Table 1: A brief history of Function Point Analysis |

#### The benefits of Function Point Analysis

Now that you have a little understanding of what FPA is, we can discuss the important things that they bring to your overall software development process.

From my experience, I've found that with a small amount of experience, understanding the functional size of your applications leads to a gold mine of other information that will help you run a successful software development business, including:

1. The ability to accurately estimate:
   - project cost
   - project duration
   - project staffing size
2. An understanding of other important metrics, such as:
   - Project defect rate
   - Cost per FP
   - FPs per hour (what I refer to as "velocity")
   - The productivity benefits of using new or different tools

As an example of what FPs can do for you, my company, Mission Data, can now tackle projects on a fixed-price basis, whereas in the last five years we've had only one other fixed price effort. This gives us a significant competitive advantage against our competition, because most people think it's impossible to develop software on a fixed price basis.

#### Overview

In this section I'll provide a brief overview of the FP counting process, and then we'll dig more into the nitty-gritty details of the process.

#### A simple five step counting process

To start at a high level, there are five steps in the process of counting FPs. They are:

1. Determine the type of count.
2. Identify the scope and boundary of the count.
3. Determine the unadjusted FP count.
4. Determine the Value Adjustment Factor.
5. Calculate the Adjusted FP Count.

I'll introduce steps 1, 2, 4, and 5 during our sample count, because they are most easily introduced by using an example. At this point I'll get into...
I'll introduce steps 1, 2, 4, and 5 during our sample count, because they are most easily introduced by using an example. At this point I'll get into the heart of step 3 in our process, because this is where the actual FP counting takes place. At this point FP practitioners look at a software application in terms of five standard functions.

**Five standard “functions”**

In counting FPs there are five standard “functions” that you count. The first two of these are called Data Functions, and last three are called Transaction Functions. The names of these functions are listed below.

1. Data Functions:
   1. Internal logical files
   2. External interface files

2. Transactional Functions:
   1. External Inputs
   2. External Outputs
   3. External Inquiries

Using this terminology, when a person that counts FPs looks at a software system, they see something like this:

![Diagram of a software application from the eyes of a Function Point practitioner.](image)

These five functions will be discussed in greater depth in the sections that follow.

**Details on the Five Data and Transactional Functions**

This section provides more detailed information and definitions on the five Data and Transactional Functions. Before getting into the details of the five functions there are several terms that you need to understand, because they’ll be used in each of the subsequent definitions. These are taken directly from the CPM.

**Important terms and definitions used in describing the five functions**

- **User identifiable**
  This term refers to defined requirements for processes and/or groups of data that are agreed upon, and understood by, both the users and software developers.

- **Control information**
  This is data that influences and elementary process of the application being counted. It specifies what, when, or how data is to be processed.

- **Elementary process**
  An elementary process is the smallest unit of activity that is meaningful to the user. An elementary process must be self-contained and leave the business of the application being counted in a consistent state.

- **Data Element Type, or DET**
  A data element type is a unique, user recognizable, non-repeated field. This definition applies to both analyses of data functions and transactional functions.

- **Record Element Type, or RET**
  A record element type is a user recognizable subgroup of data elements within an Internal Logical File or External Interface File.

**1) Data Functions -- Internal Logical Files (ILFs)**

ILF stands for “Internal Logical File”. In my words, ILFs represent data that is stored and maintained within the boundary of the application you are counting. When counting ILFs you are basically counting the data functions that your application is being built to maintain.

The more precise IFPUG definition of an ILF is:

> "An ILF is a user-identifiable group of logically related data or control information maintained within the boundary of the application. The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted."

Furthermore, for data or control information to be counted as an ILF, both of the following IFPUG counting rules must also apply:

1. The group of data or control information is logical and user identifiable.
2. The group of data is maintained through an elementary process within the application boundary being counted.

**Examples of ILFs**

Samples of things that *can* be ILFs include:

1. Tables in a relational database.
2. Flat files.
3. Application control information, perhaps things like user preferences that are stored by the application.
4. LDAP data stores.

This isn't to say that all these things are ILFs, just that they can be.

**Function point counts resulting from ILFs**

When you're counting ILFs you will constantly be referring to the two tables that follow. The purpose of the first table is to help you determine whether the ILF you're currently looking at has a complexity level of Low (L), Average (A), or High (H). You do this by going into the table knowing the number of DETs and number of RETs you have counted in the ILF, and then finding the resulting Low, Average, or High value.

For instance, suppose I counted 5 DETs and 1 RET; that would be a Low complexity table. Conversely, if I had a table with 21 DETs and 2 RETs, that would be an Average complexity table.

<table>
<thead>
<tr>
<th>RETS</th>
<th>Data Element Types (DETs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-19</td>
<td>20-50</td>
</tr>
<tr>
<td></td>
<td>51+</td>
</tr>
</tbody>
</table>

Figure 1: The view of a software application from the eyes of a Function Point practitioner.
Now that you know whether the ILF under consideration has a complexity of Low, Average, or High, you’ll come to this next table and determine the number of FPs that should be counted for this ILF. A Low complexity ILF is worth 7 points, an Average ILF is worth 10 points, and a High is worth 15.

Weights:

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>7</td>
</tr>
<tr>
<td>Average</td>
<td>10</td>
</tr>
<tr>
<td>High</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 3: ILF weights

(2) Data Functions -- External Interface Files (EIFs)

EIF stands for “External Interface File”. In my words, EIFs represent the data that your application will use/reference, but data that is not maintained by your application.

The official IFPUG definition of an EIF is:

“An external interface file (EIF) is a user identifiable group of logically related data or control information referenced by the application, but maintained within the boundary of another application. The primary intent of an EIF is to hold data referenced through one or more elementary processes within the boundary of the application counted. This means an EIF counted for an application must be in an ILF in another application.”

Again, think of this as data that your application needs and uses, but does not maintain.

Examples of things that can be EIFs are identical to the list for ILFs, but again, the big difference here is that EIFs are not maintained by the application under consideration, but ILFs are.

Function point counts resulting from EIFs

Assigning an FP value to an EIF is the same as assigning one to an ILF. First, determine the number of DETs and RETs in the EIF, then do a lookup in the following table to determine whether the EIF has a complexity of Low, Average, or High.

<table>
<thead>
<tr>
<th>RETS</th>
<th>Data Element Types (DETs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-19</td>
<td>20-50</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2 to 5</td>
<td>A</td>
</tr>
<tr>
<td>6 or more</td>
<td>A</td>
</tr>
</tbody>
</table>

Table 4: EIF complexity matrix

Then, once you know whether the EIF is considered Low, Average, or High, look in the following table for the number of FPs to count for this particular EIF.

Weights:

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>5</td>
</tr>
<tr>
<td>Average</td>
<td>7</td>
</tr>
<tr>
<td>High</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 5: EIF weights

You’ll notice that the first two lookup tables are identical, but that more FPs are assigned to ILFs than EIFs. Here’s some homework for you: Can you guess why?

(3) Transaction Functions -- External Inputs (EI’s)

EI stands for “External Input”. Here the official IFPUG definition of an EI is as follows:

An external input (EI) is an elementary process that processes data or control information that comes from outside the application boundary. The primary intent of an EI is to maintain one or more ILFs and/or to alter the behavior of the system.

Examples of EIs include:

1. Data entry by users.
2. Data or file feeds by external applications.

Function point counts resulting from EIs

Allocating FPs to EIs is similar to the process we covered for ILFs and EIFs. However, in this case, instead of doing a lookup based on DETs and RETs to determine a Low/Average/High complexity, the lookup is performed using DETs and FTRs. As you’ll recall from the earlier definition, an FTR is a “file type referenced”, so it can be either an ILF or an EIF.

As an example, suppose that you have a process that has 5 DETs, and during the processing it references an EIF named Users and an ILF named Process. You would go into the following table looking for the complexity of an EI that has 5 DETs and 2 FTRs. As you’ll see from the table below, this EI is considered an “Average” complexity EI.

<table>
<thead>
<tr>
<th>FTRs</th>
<th>Data Element Types (DETs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>2-15</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>3 or more</td>
<td>A</td>
</tr>
</tbody>
</table>

Table 6: EI complexity matrix

To carry our example forward, as you can see from the following table, an Average complexity EI is worth 4 FPs.

Weights:

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>3</td>
</tr>
<tr>
<td>Average</td>
<td>4</td>
</tr>
<tr>
<td>High</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 7: EI weights

(4) Transaction Functions -- External Outputs (EO’s)

External Outputs are referred to as EO’s. The IFPUG definition of an EO is as follows:

An external output (EO) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external output is to present information to a user through processing logic other than, or in addition to, the retrieval of data or control information. The processing logic must contain at least one mathematical formula or calculation, create derived data maintain one or more ILFs or alter the behavior of the system.

Examples of EOs include:
1. Reports created by the application being counted, where the reports include derived information.

**Function point counts resulting from EO's**

Allocating FPs to EOs is very similar to the process for EIs. Again, you perform your lookup using DETs and FTRs, with a resulting Low/Average/High complexity.

As an example, suppose that you have a process that you've identified as being an EO, and it has 10 DETs and references two FTRs. You would go into the following table looking for the complexity of an EI that has 10 DETs and 2 FTRs. As you'll see from the table below, this EO is considered an "Average" complexity EO.

<table>
<thead>
<tr>
<th>FTRs</th>
<th>Data Element Types (DETs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-5</td>
</tr>
<tr>
<td>0-1</td>
<td>L</td>
</tr>
<tr>
<td>2-3</td>
<td>A</td>
</tr>
<tr>
<td>4 or more</td>
<td>A</td>
</tr>
</tbody>
</table>

Table 8: EO complexity matrix

Weights:

To carry our example forward, using the table that follows, you'll see that an Average complexity EO has a value of 5 FPs.

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>4</td>
</tr>
<tr>
<td>Average</td>
<td>5</td>
</tr>
<tr>
<td>High</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 9: EO weights

**5) Transaction Functions -- External Inquiries (EQ's)**

The last transactional function is referred to as an EQ, or External Inquiry. The IFPUG definition of an EQ is as follows:

_An external inquiry (EQ) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external inquiry is to present information to a user through the retrieval of data or control information from an ILF of EIF._

The processing logic contains no mathematical formulas or calculations, and creates no derived data. No ILF is maintained during the processing, nor is the behavior of the system altered.

Examples of EQs include:

1. Reports created by the application being counted, where the report does not include any derived data.
2. Other things known as "implied inquiries", which unfortunately, are a little out of scope for this paper.

**Function point counts resulting from EQ's**

Allocating an FP count to EQs is very similar to the process for EIs and EOs. Again, you perform your lookup using DETs and FTRs, with a resulting Low/Average/High complexity.

As an example, suppose that you have a process that you've identified as being an EQ, and it has 20 DETs and references 4 FTRs. You would go into the following table looking for the complexity of an EI that has 20 DETs and 4 FTRs. As you'll see from the table below, this EQ is considered a "High" complexity EQ.

<table>
<thead>
<tr>
<th>FTRs</th>
<th>Data Element Types (DETs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-5</td>
</tr>
<tr>
<td>0-1</td>
<td>L</td>
</tr>
<tr>
<td>2-3</td>
<td>A</td>
</tr>
<tr>
<td>4 or more</td>
<td>A</td>
</tr>
</tbody>
</table>

Table 10: EQ complexity matrix

Weights:

Carrying our EQ example forward, you'll find from the table below that a High complexity EQ is worth 6 FPs.

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>3</td>
</tr>
<tr>
<td>Average</td>
<td>4</td>
</tr>
<tr>
<td>High</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 11: EQ weights

**Summary**

Taken together, these two data functions (ILFs and EIFs) and three transactional functions (EIs, EOs, EQs) represent the five functions that are counted in a FP count.

**A sample count**

An introduction to our example

Okay, I've given you some background information and theory. The best thing to do now is to jump into the process of counting a sample application.

To demonstrate how the FP process works, I'm going to use a sample application that I've created. The application is named "FPTracker", and as its name implies, it's a tool that I use when I perform FP counts on other applications. For instance, if I came to your company to perform FP counts on your applications, I would record the data from the functions in your applications using this tool. As output, it provides some convenient reports and an analysis of the data to help keep me from making errors in my counts.

The FPTracker application consists of the following primary process areas:

1. Project management, including creating, editing, and deleting projects.
2. Entity management, including creating, editing, and deleting ILFs and EIFs.
3. Process management, including creating, editing, and deleting EIs, EOs, and EQs.
4. Process group management, which is a mechanism for grouping processes.
5. Reporting, which includes several useful FP reports.

For the purposes of our example, I'm only going to cover a subset of the application. In a few moments you'll learn more about the FPTracker application by seeing the data it stores, the screens it offers, and two output reports it provides.

**The counting process**

As a quick review, the five steps in the process of counting FPs are as follows:

1. Determine the type of count.
2. Identify the scope and boundary of the count.
3. Determine the unadjusted FP count.
4. Determine the Value Adjustment Factor.
5. Calculate the Adjusted FP Count.

In our example we're going to follow these steps precisely.

**Step 1: The type of count**

The first step in our FPA process is determining the type of count for the application at hand. Function point counts can be one of three different types:

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development Project FP Count</td>
<td>Measures the functions provided to the users with the first installation of the software being delivered.</td>
</tr>
</tbody>
</table>
Step 1: Identify the source of the count

The first step of the FPA process is identifying the source of the count, which will be the project that we are counting. Since we already have a project in mind, this step is relatively straightforward.

Table 12: Types of FP counts

<table>
<thead>
<tr>
<th>Count Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development Project</td>
<td>Measures the initial development of a new application or enhancement of an existing application.</td>
</tr>
<tr>
<td>Enhancement Project</td>
<td>Measures the modifications to an existing application.</td>
</tr>
<tr>
<td>Application Count</td>
<td>Measures the functionality provided to users in an existing application.</td>
</tr>
</tbody>
</table>

Because my FPTracker application already exists and is in production, the type of our count is an "Application FP Count".

That's all we have to do for Step 1. Note that this does not have anything to do with assigning points at this time. We are simply recording what type of project we are counting, because there are different summation rules at the end of the count, and they vary depending on the type of count you are performing.

Step 2: Identify the scope and boundary of the count

The second step of the FPA process is identifying the scope and boundary of our count. The scope of this count will be defined by the data, screens, and reports that I’m about to show you. You should not make any assumptions about any behavior that may appear to be missing.

Regarding the boundary of this application, for our purposes FPTracker should be thought of as a simple, standalone software application. Unlike a suite of applications like Microsoft Office, or a combination of applications that can be found on an Intranet or Extranet site, the FPTracker is not tied to any other applications in any way. Therefore, if I were drawing a boundary line to set off our application from others, it would be similar to the boundary line shown in the sample of Figure 1.

It’s also worth noting that the purpose of this particular count is to serve as a sample count for this paper. In the real world the purpose of our counts vary, but typical examples include:

1. Serving as the basis to determine the cost of this software application.
2. Helping to determine the productivity rate of the development team on this project.
3. Helping to determine the retail value of this software asset to our company.

Of course there could be any number of other reasons for pursuing a count like this, but this is a good list of the typical reasons.

Step 3: Determine the unadjusted function point count

Although the first two steps are very important in setting the groundwork for your count, and specifying the scope of your count, they are usually performed very quickly, and it is the third step that really takes the majority of the FPA time, and is the heart of the count.

In this step we’ll be counting the data functions and transactional functions that yield the unadjusted FP count. As you’ve seen, to this point we haven’t been dealing with any numbers, but we’re now ready to start adding them up.

Step 3a: Determine the count resulting from ILFs

In the FPTracker application the data is stored in a series of relational database tables, so we’ll use those as the basis of our analysis. The following is a list of the database table names used in the FPTracker application:

1. Project
2. Entity
3. Process Group
4. Process
5. ProcessDETs
6. ProcessEntities

The full listing of the fields stored in each table are shown in the following tables. These tables provide each field name, a field description, and indicator as to whether the field should be counted as a DET, and other notes/comments regarding each field. The last row of each table provides a count of the total number of DETs in the corresponding database table.

Table 13: The "Project" database table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Count as a DET?</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>project_id</td>
<td>Sequential id, system-assigned.</td>
<td>No</td>
<td>This is a technical artifact. It is not user-recognizable, and therefore not counted.</td>
</tr>
<tr>
<td>project_name</td>
<td>The name a user assigns to a given project.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>project_type</td>
<td>The type of project.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>A description of the project.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Total DETs:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 14: The "Entity" database table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Count as a DET?</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity_id</td>
<td>Sequential id, system-assigned.</td>
<td>No</td>
<td>System-generated sequence number. Not user-recognizable.</td>
</tr>
<tr>
<td>project_id</td>
<td>Foreign key.</td>
<td>Yes</td>
<td>Do count a DET for pieces of data that are required by the user to establish a relationship with another ILF or EIF. Foreign keys usually fit this definition.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the entity.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Type of entity (ILF or EIF).</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>no. RETs</td>
<td>Number of RETs in the entity.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>no. DETs</td>
<td>Number of DETs in the entity.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>complexity</td>
<td>Calculated complexity (Low, Average, or High).</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Total DETs:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 15: The "Process" database table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Count as a DET?</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_id</td>
<td>Foreign key.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>Name of the process.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Type of process (EI, EO, or EQ).</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>sort_order</td>
<td>Order of appearance when displayed.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>no. FTRs</td>
<td>Number of FTRs.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>no. DETs</td>
<td>Number of DETs.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>complexity</td>
<td>Calculated complexity (Low, Average, or High).</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>Description of the process.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Total DETs:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 16: The "Process Group" database table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Count as a DET?</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>process_group_id</td>
<td>Sequential id, system-assigned.</td>
<td>No</td>
<td>System-generated sequence number. Not user-recognizable.</td>
</tr>
<tr>
<td>project_id</td>
<td>Foreign key.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>Name of the process group.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Total DETs:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In this particular application there is not a one-to-one relationship between the database tables and the way a user would logically view this data being stored. As an FP counter, and also the primary user of this application!), I think of the data logically being stored like this:

1. Project
2. Entity
3. Process Group
4. Process
5. ProcessDETs
6. ProcessEntities

In this case what I'm saying is that, as the user of this application, I think of the Process DETs and the Process Entities as being logical subgroups of the overall Process data. Okay, you say, but what does that matter to me as an FP counter? Let's look at how our counts would differ if we count the tables first as being separate, then second as being logically merged.

First, if I think of the these six tables as separate ILFs, because of the number of DETs they contain, each table would be considered to be a "Low" complexity table. Because each Low table is worth 7 FPs, the FP count for these six tables would be 42 FPs (6x7=42).

However, if I look at the tables from a "logical" perspective instead of the physical RDBMS layout, what I really have are four logical groups. Because of their DET and RET counts, all four of the groups are considered "Low" in complexity. So, in this case, the total count is 28 FPs (4x7=28).

Therefore, had I counted the physical implementation of the tables as opposed to a logical grouping, I would have over-counted this portion of the application by 14 FPs, or 50%. This is a good example of where a physical implementation differs from a correct, logical grouping for an FP count.

### Total Count Due to ILFs

To tally up the FP count due to ILFs, I've identified four ILFs, with the number of RETs and number of DETs as shown in the table below. The first three ILFs are very straightforward, and as discussed above, I believe that I have one Process ILF with three RETs: (1) Process, (2) Process DETs, and (3) Process Entities. Using the ILF lookup table shown earlier in Table 2, I see that each ILF is considered "Low" in complexity. Looking up Low Complexity ILFs in Table 3, I see that a Low Complexity table is counted as seven FPs. This gives me the total of 28 FPs as a result of ILFs.

<table>
<thead>
<tr>
<th>ILF</th>
<th>No. RETs</th>
<th>No. DETs</th>
<th>Complexity</th>
<th>Function Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>1</td>
<td>3</td>
<td>Low</td>
<td>7</td>
</tr>
<tr>
<td>Entity</td>
<td>1</td>
<td>6</td>
<td>Low</td>
<td>7</td>
</tr>
<tr>
<td>Process</td>
<td>3</td>
<td>13</td>
<td>Low</td>
<td>7</td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td></td>
<td></td>
<td>28</td>
</tr>
</tbody>
</table>

Table 19: Total FP counts due to ILFs

### Step 3b: Determine the count resulting from EIF's

In this particular application there are no EIFs, so EIFs contribute zero FPs to the overall FP count.

However, in many applications there definitely are EIFs. Suppose, for example, that this application required the use of a file named Users, and this file was maintained by another application, but was accessed by our application. In this case the file would be counted as an EIF, using the EIF rules discussed earlier.

At this point we've finished counting data functions, and it's time to move on to transaction functions.

### Step 3c: Determine the count resulting from EI's

To begin counting the transactional functions we first need to look at the user interface screens in the application. Let's look at all of these screens first, then determine which of these are EI's, which are EO's, and finally which are EQ's.

![Screen shots from the application](image-url)
Figure 3: This screen shows the Entities tab. From this location users can View, Add, Edit, and Delete entities.

Add Entity screen lets the user add new entities (ILFs and EIFs) to the project.
Figure 5: The “Edit Entity” screen lets the user edit entities in the project.

Figure 6: The “Delete Entity” screen asks the users to confirm their attempt to delete an entity in the application.

Figure 7: This screen shows the Process Groups tab. From this screen users can Add, Edit, and Delete Process Groups.
Figure 8: The "Add Process Group" screen lets the user add new process groups to the application.

Figure 9: The "Edit Process Group" lets the user assign a new name to a process group.

Figure 10: This screen lets the user delete a selected process group.
Figure 11: This figure shows the Processes tab. From this location users can Add, Clone, Edit, and Delete processes.

Figure 12: This screen lets the user define a new process. Note the "Description" tab. Those details are shown in the next figure.
Figure 13: This screen shows the details of the Comments/Description tab. Users can optionally enter information here when creating a new Process.

Figure 14: Users can use this screen to clone a Process in the application. They simply select an existing Process, then select a Clone option that leads to this screen.

Unadjusted Function Point Count

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Functional Complexity</th>
<th>Complexity Totals</th>
<th>Function Type Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILF</td>
<td>Low</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>High</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>EIF</td>
<td>Low</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>High</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>EI</td>
<td>Low</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>High</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>EQ</td>
<td>Low</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>High</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>EQ</td>
<td>Low</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>High</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

Unadjusted Function Point Count: 66

Figure 15: This screen shows a basic Unadjusted Function Point Count report.
Now that we've seen the screens in the application we have to determine which of these correspond to EI's, which are EO's, and which are EQ's. We'll do that in the next sections.

**Brief process descriptions**

Because you haven't seen this application before, it may be hard to understand the processes in this application. Therefore, I'll provide a brief name and description of each process in this application.

<table>
<thead>
<tr>
<th>Process Name</th>
<th>Process Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a Project</td>
<td>This screen is displayed in Figure 2. This lets the user create a new project. Note that for the purposes of this paper I am skipping other project-related processes, including the ability to list, edit, delete, and clone projects.</td>
</tr>
<tr>
<td>Display Entity List</td>
<td>The list is displayed in Figure 3. This is a list of every Entity that the user has identified for the application they are counting.</td>
</tr>
<tr>
<td>Add Entity</td>
<td>Shown in Figure 4, this dialog lets the user add a new Entity.</td>
</tr>
<tr>
<td>Edit Entity</td>
<td>Shown in Figure 5, this dialog lets the user edit an existing Entity. This dialog appears when the user highlights a row in the Entity list, then selects the Edit button. Alternatively, they can double-click on the desired row.</td>
</tr>
<tr>
<td>Delete Entity</td>
<td>Shown in Figure 6, this confirmation dialog is displayed when a user selects a row, then presses the Delete button.</td>
</tr>
<tr>
<td>Display Process Group List</td>
<td>This list is shown in Figure 7. This is a list of every Process Group the user has identified.</td>
</tr>
<tr>
<td>Add Process Group</td>
<td>Shown in Figure 8, this dialog lets the user define a new Process Group.</td>
</tr>
<tr>
<td>Edit Process Group</td>
<td>Shown in Figure 9, this dialog lets the user edit an existing Process Group.</td>
</tr>
<tr>
<td>Delete Process Group</td>
<td>Shown in Figure 10, this confirmation dialog is displayed when a user selects a row in the Process Group table, then presses the Delete button.</td>
</tr>
<tr>
<td>Display Process List</td>
<td>The list is displayed in Figure 11. This is a list of every Process that the user has identified for the application they are counting.</td>
</tr>
<tr>
<td>Add Process</td>
<td>Shown in Figures 12 and 13, this dialog lets the user define a new Process.</td>
</tr>
<tr>
<td>Edit Process</td>
<td>Although not shown, this dialog is identical to screens shown in Figures 12 and 13, other than the title of the dialog. These screens let the user edit an existing Process.</td>
</tr>
<tr>
<td>Delete Process</td>
<td>Although not shown, this confirmation dialog is displayed when a user selects a row, then presses the Delete button. For the sake of our count, assume that it is a simple confirmation dialog, similar to the dialog for deleting a Process Group.</td>
</tr>
<tr>
<td>Clone Process</td>
<td>Shown in Figure 14, this dialog lets the user make a duplicate copy of a Process. This makes it easy to create new processes which are very similar to an existing process.</td>
</tr>
<tr>
<td>Display UPPC Report</td>
<td>Shown in Figure 15, this is a typical report that totals up the number of FPs in the application you are recording. The report is displayed after the user selects the report type from a drop-down list of reports in the Reports tab. The list of data shown in the drop-down list is hard-coded into the application.</td>
</tr>
<tr>
<td>Display ILF/EIF Report</td>
<td>Shown in Figure 16, this report shows the ILFs and EIFs (i.e., all the FTRs) in the application, along with their associated number of DETs and RETs, as well as their complexity level. It is selected from the same drop-down list as the UPPC Report.</td>
</tr>
</tbody>
</table>

**Function points resulting from EI’s**

The table below lists the External Inputs in the application. It also lists the number of DETs and FTRs for each process, and the complexity that results from the number of DETs and FTRs.

<table>
<thead>
<tr>
<th>Process</th>
<th>No. DETs</th>
<th>FTR Names</th>
<th>No. FTRs</th>
<th>Resulting Complexity</th>
<th>No. FPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Project</td>
<td>5</td>
<td>Project</td>
<td>1</td>
<td>Low</td>
<td>3</td>
</tr>
<tr>
<td>Add Entity</td>
<td>7</td>
<td>Project, Entity</td>
<td>2</td>
<td>Average</td>
<td>4</td>
</tr>
<tr>
<td>Delete Entity</td>
<td>4</td>
<td>Project, Entity</td>
<td>2</td>
<td>Low</td>
<td>3</td>
</tr>
<tr>
<td>Add Process Group</td>
<td>1</td>
<td>Project, ProcessGroup</td>
<td>2</td>
<td>Low</td>
<td>3</td>
</tr>
<tr>
<td>Edit Process Group</td>
<td>1</td>
<td>Project, ProcessGroup</td>
<td>2</td>
<td>Low</td>
<td>3</td>
</tr>
</tbody>
</table>
Step 3d: Determine the count resulting from EO’s

According to the rules, we have only one EO, and that is the UFPC Report. This report is considered an EO, and specifically not an EQ, because it contains derived data. More specifically, the complexity total column and the total UFPC count at the bottom of the table are derived fields, and EQs specifically cannot contain derived fields like this.

This report is actually a very difficult report for me to get right, and that’s exactly why I’ve included it here. It’s hard for me because of the rules surrounding duplicated fields in reports. Unfortunately, rather than get into all the complexity of these rules, I’m just going to state that there are at least seven DETs in this report, and they are:

1. Title
2. Function type
3. Functional complexity
4. The complexity totals
5. The function type totals
6. The Unadjusted FP Count
7. A DET that is counted for the menu option to choose this report

Table 21: External Inputs (EI’s) in the FPTracker application

<table>
<thead>
<tr>
<th>Process DETs</th>
<th>Process FTRs</th>
<th>Resulting Complexity</th>
<th>No. FPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>UFPC Report</td>
<td>7</td>
<td>Average</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 3e: Determine the count resulting from EQ’s

The table below lists the External Inquiries in the application. It also lists counts of the number of DETs and FTRs for each process, and the complexity that results from the number of DETs and FTRs.

We begin with the ILF/EIF Report. Fortunately, it’s a lot easier that the UFPC Report. It contains six DETs, including the title, data function names, function type, number of DETs, number of FTRs, and complexity. Looking at the report, you can also see that it retrieves information from the Project and Entity ILFs, so there are two ILFs.

Implied Inquiries

The rest of the EQs in the table below are probably quite a surprise. I haven’t introduced them yet, primarily because they add a little complexity to the mix. Each of the last four processes in this table are referred to as “implied inquiries”. They are the almost hidden processes in an application that are easy to overlook. For instance, if you look back at Figures 3, 7, and 11, you’ll see that the application has built-in lists of Entities, Process Groups, and Processes. These are the tables (Table’s to be more specific) that are contained in the Entity, Process Group, and Process tabs, respectively. Some type of query had to be performed to generate these lists, and those types of queries that result in this “view list” operation often qualify as “implied inquiries”.

The Process Group drop-down list on the Add/Edit Process dialog is another implied inquiry. It took some type of query to generate that list, and again it is considered an implied inquiry.

While I’m covering this important topic rather quickly, it’s important to note that other drop-down lists and tables may not qualify as implied inquiries, especially when the data they contain is hard-coded in the application. You also might not count the listing as an implied inquiry if the data could have been hard-coded. This is an introductory tutorial, so that’s all I’m going to say about this topic for now, other than the fact that I will refer you to the CPM for more specific rules.

Table 22: External Outputs (EO’s) in the FPTracker application

Because of the repeating fields this is a very difficult report for me. Fortunately, when I double-checked with a colleague we both came up with an Average score for this EO, even though we differed on the fields slightly. That’s one of the nice things about using ranges like this; very rarely does a disagreement about the detailed rules actually affect a count.

Step 3f: Determine the count resulting from EQ’s

The table below lists the External Inquiries in the application. It also lists counts of the number of DETs and FTRs for each process, and the complexity that results from the number of DETs and FTRs.

We begin with the ILF/EIF Report. Fortunately, it’s a lot easier than the UFPC Report. It contains six DETs, including the title, data function names, function type, number of DETs, number of FTRs, and complexity. Looking at the report, you can also see that it retrieves information from the Project and Entity ILFs, so there are two ILFs.

Implied Inquiries

The rest of the EQs in the table below are probably quite a surprise. I haven’t introduced them yet, primarily because they add a little complexity to the mix. Each of the last four processes in this table are referred to as “implied inquiries”. They are the almost hidden processes in an application that are easy to overlook. For instance, if you look back at Figures 3, 7, and 11, you’ll see that the application has built-in lists of Entities, Process Groups, and Processes. These are the tables (Table’s to be more specific) that are contained in the Entity, Process Group, and Process tabs, respectively. Some type of query had to be performed to generate these lists, and those types of queries that result in this “view list” operation often qualify as “implied inquiries”.

The Process Group drop-down list on the Add/Edit Process dialog is another implied inquiry. It took some type of query to generate that list, and again it is considered an implied inquiry.

While I’m covering this important topic rather quickly, it’s important to note that other drop-down lists and tables may not qualify as implied inquiries, especially when the data they contain is hard-coded in the application. You also might not count the listing as an implied inquiry if the data could have been hard-coded. This is an introductory tutorial, so that’s all I’m going to say about this topic for now, other than the fact that I will refer you to the CPM for more specific rules.

Table 23: External Inquiries (EQ’s) in the FPTracker application

Total Count

Now that we’ve finished counting the ILFs, EIFs, EIs, EOs, and EQs in the application, we add up each of the individual counts to get a total unadjusted function point count for the application. This is shown in the table that follows.

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Functional Complexity</th>
<th>Complexity Totals</th>
<th>Function Type Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILF</td>
<td>4 Low</td>
<td>x 7 = 28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 Average</td>
<td>x 10 = 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 High</td>
<td>x 15 = 0</td>
<td>28</td>
</tr>
<tr>
<td>EIF</td>
<td>0 Low</td>
<td>x 5 = 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 Average</td>
<td>x 7 = 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 High</td>
<td>x 10 = 0</td>
<td>0</td>
</tr>
<tr>
<td>EI</td>
<td>5 Low</td>
<td>x 3 = 15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 Average</td>
<td>x 4 = 12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 High</td>
<td>x 6 = 18</td>
<td>45</td>
</tr>
<tr>
<td>EQ</td>
<td>0 Low</td>
<td>x 3 = 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Average</td>
<td>x 4 = 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 High</td>
<td>x 6 = 0</td>
<td>4</td>
</tr>
</tbody>
</table>
Here are a few facts and definitions to get the ball rolling:

**VAF Introduction**

Before I tell you what the VAF is, let me first give you a warning about it: To the best of my knowledge, most users don't use it. I can't tell you much about the history of the VAF, but what I can tell from the conversations I've had with many other users is that they don't use the VAF. This stems from at least two reasons that I can determine:

1. Most users count function points to derive a number that they can plug into another piece of software to determine a cost estimate. Those other software applications usually have their own equivalent of the VAF, and in fact, instruct you to supply the "raw FP count". So, in this case, the VAF competes against these vendor tools.
2. Some users don't feel that the GSCs are flexible enough. I tend to agree, and I think it's an easy argument. When you look at the math below, you'll see that for two applications under consideration, if both start with the same function point count -- let's say 1,000 FPs -- after adjustments the hardest application in the world would be rated at 1,350 FPs, and the easiest possible application would be rated at 650 FPs when adjusted. Let's say the hardest application in the world had to run on 10 different operating systems in 15 languages and be distributed electronically to 1 million users, and the easiest would be written in Microsoft Access, run on Windows, and be used by only one user, the author of the program. Do you really think the first application is only about twice as hard to deliver as the second? No, I certainly don't, and this is why I don't use the VAF.

All that being said, I'll give you the quick VAF introduction so you can make a decision on your own. If nothing else, I find it nice that someone has taken the time to come up with these 14 categories. It helps my thinking in other areas.

**Step 4: Determine the Value Adjustment Factor (VAF)**

In this step we'll look at something named the "Value Adjustment Factor", or VAF. But first, a warning about the VAF.

A warning about the VAF

Before I tell you what the VAF is, let me first give you a warning about it: To the best of my knowledge, most users don't use it. I can't tell you much about the history of the VAF, but what I can tell from the conversations I've had with many other users is that they don't use the VAF. This stems from at least two reasons that I can determine:

1. Most users count function points to derive a number that they can plug into another piece of software to determine a cost estimate. Those other software applications usually have their own equivalent of the VAF, and in fact, instruct you to supply the "raw FP count". So, in this case, the VAF competes against these vendor tools.
2. Some users don't feel that the GSCs are flexible enough. I tend to agree, and I think it's an easy argument. When you look at the math below, you'll see that for two applications under consideration, if both start with the same function point count -- let's say 1,000 FPs -- after adjustments the hardest application in the world would be rated at 1,350 FPs, and the easiest possible application would be rated at 650 FPs when adjusted. Let's say the hardest application in the world had to run on 10 different operating systems in 15 languages and be distributed electronically to 1 million users, and the easiest would be written in Microsoft Access, run on Windows, and be used by only one user, the author of the program. Do you really think the first application is only about twice as hard to deliver as the second? No, I certainly don't, and this is why I don't use the VAF.

All that being said, I'll give you the quick VAF introduction so you can make a decision on your own. If nothing else, I find it nice that someone has taken the time to come up with these 14 categories. It helps my thinking in other areas.

**VAF Introduction**

Here are a few facts and definitions to get the ball rolling:

- The Value Adjustment Factor (VAF) consists of 14 "General System Characteristics", or GSCs.
- These GSCs represent characteristics of the application under consideration. Each is weighted on a scale from 0 (low) to 5 (high).
- When you sum up the values of these 14 GSCs you get a value named "Total Degree of Influence", or TDI. As you can see from the math below, you'll see that for two applications under consideration, if both start with the same function point count -- let's say 1,000 FPs -- after adjustments the hardest application in the world would be rated at 1,350 FPs, and the easiest possible application would be rated at 650 FPs when adjusted. Let's say the hardest application in the world had to run on 10 different operating systems in 15 languages and be distributed electronically to 1 million users, and the easiest would be written in Microsoft Access, run on Windows, and be used by only one user, the author of the program. Do you really think the first application is only about twice as hard to deliver as the second? No, I certainly don't, and this is why I don't use the VAF.

Before getting into the VAF formula, let me quickly list the 14 GSCs:

1. Data Communication
2. Distributed data processing
3. Performance
4. Heavily used configuration
5. Transaction rate
6. Online data entry
7. End user efficiency
8. Online update
9. Complex processing
10. Reusability
11. Installation ease
12. Operational ease
13. Multiple sites
14. Facilitate change

Given this background information, you can see with the following formula:

\[ \text{VAF} = (\text{TDI} \times 0.01) + 0.65 \]

that the VAF can vary in range from 0.65 (when all GSCs are low) to 1.35 (when all GSCs are high).

In the next section you'll see that the VAF is applied directly to the FP count to determine the "Adjusted Function Point Count".

**Step 5: Calculate the Adjusted Function Point Count**

The final step in our five-step process is to determine the Adjusted Function Point Count. For initial application counts such as ours, this is easily determined with the following equation:

\[ \text{Adjusted FP Count} = \text{Unadjusted FP Count} \times \text{VAF} \]

As you saw in the previous section, the VAF can vary from 0.65 to 1.35, so the VAF exerts an influence of +/- 35% on the final Adjusted FP Count. Since we're not going to come up with a value for the VAF in this paper, we also will not have a number for the Adjusted FP Count. As you can see, however, this is very easy and straightforward.

(I tried to save the easiest step for last. :)

**The Value of Counting Function Points**

Okay, I showed you the "how" of FP counting, but I'd like to return to the "why" of FP counting, because I think motivation is a very important driver here. You need to asking yourself "Why should you add as much as 1% to your overall software development effort?"

My experience with FP counting has shown all of the benefits I mentioned earlier in this document. Once you have a history of developing applications and you also have FP counts for all those applications, you can now add to your software development arsenal these capabilities:

1. The ability to accurately estimate:
   - project cost
   - project duration
2. The ability to determine other important metrics, such as:
   - Project defect rate
   - Cost per FP
   - FPs per hour (a productivity rate; I tend to refer to it as "Velocity", a term I like from Extreme Programming)
3. The productivity benefits of using new or different tools

So, the question to you is "What are these abilities worth to you?"

For me, the biggest benefit of FP counting means that my company can get into fixed-price software development projects. When a prospect says "Al, can you do this project for $100,000?" I can run around the corner, scratch some numbers on the back of an envelope, and give them a Yes or No answer. And while doing this I can be pretty well assured that the company won't go bankrupt on this project.

<table>
<thead>
<tr>
<th>GSC</th>
<th>Weight</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EO</td>
<td>Low</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>High</td>
<td>7</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Unadjusted Function Point Count: 99
How to Determine Your Application Size

Why is this important? Because I've never met a developer that likes to estimate programming work, and the bigger the work, the worse is gets. I don't blame them; estimating is very hard, especially on larger projects. I've met a lot of developers, and some always estimate low, some always estimate high, and others go both high and low. As a manager, I'd much rather have some cold statistics that I can rely on in times like this, even if it's just as a point of comparison.

An example of how this works

As an example of how this works, I'm going to walk through the process that we followed recently. This involves two small phases of development on one project. On this project, the developers, project managers, users, technology, and application all remained constant. We also followed a relatively formal process, beginning with a requirements phase, which led into development, testing, and final delivery of a portion of an application.

A first measurement

In this case I was able to start with one project that we were already developing. The project itself was small, but it was a distributed Java Swing application with an embedded database that runs on Windows, Mac, and Linux platforms, so even though the application itself was small, the degree of complexity here was very high.

The first thing we did was to finish the requirements documentation and initial prototypes for our application. Once I had this information, which included a simple, logical view of the database requirements, I was able to count the function points for this small project. We created what we thought was a fairly detailed requirements document, and the count at this time was 400 FPs.

Skipping a few details here, let's just say that the next thing we did was to develop this project. When we called the development "complete", I counted the number of FPs that were actually delivered to the users. This was 440 FPs, or a growth from the requirements stage of 11%.

At this time the customers who had development time information. Two developers worked on this project for a total of 540 man-hours. This comes out to be 0.815 FPs/man-hour (440 FPs divided by 540 man-hours). Had our customer kept a good record of time that users spent testing the application they also could have determined a metric of "Number of testing hours per FP", but they did not. IMHO, I think this would benefit them in the future, but insole, application testing is not my responsibility, so I did not pursue this.

Although we spent 540 hours on this project, the real "calendar time" for delivery of the application was 10 weeks. This was because of several periods of down time during the development process. Therefore this project was delivered at the rate of 44 FPs per calendar week.

Depending on how you track cost information, you can also determine "Cost per FP". As I stated earlier, as an independent software development firm, we now develop complex applications like this for about $250/FP.

Your second measurement

Because this is an ongoing project, we basically repeated the same steps on the next phase of our project. For summary purposes, here are the steps we followed:

1. Develop the requirements, including an understanding of the necessary data stores and screens to be developed.
2. Count the FPs.
3. Applied an estimate of the project cost, assuming another 11% gain in functionality (scope creep) during development.
4. Develop the code.
5. Track the amount of time people spend on the project during development and testing.
6. Count the FPs again.
7. Deliver useful project metrics, including:
   1. Number of developer hours.
   2. Number of testing hours.
   3. Average number of hours per FP.
   4. Elapsed calendar time, which yields something like "Number of calendar days per FP" or the converse of "Number of FPs per calendar day". This occurs when there is down time in a project, or when your development resources are not fully dedicated to the project at hand.
   5. Development cost per FP.
   6. Testing cost per FP.
   7. Overall cost per FP (including other time for management, documentation, etc.).
   8. The ratio of Requirements time to Development time, and other similar ratios.

Note that Step 3 in this example is "estimate the project cost". Because we have the same developers, users, and managers working on a different part of the same project, isn't it reasonable to assume that the project velocity for earlier phases will be the same for this new phase? For me it is at the heart of estimating new development work with FPs. Given this scenario of having the same developers, users, and managers, and working with the same technology on the same application, I'm glad to take my chances estimating with FPs.

Now, if you suddenly change any of these factors, you can use this same information for your estimates, but you'll estimate probably vary somewhat. For instance, with our company, I've found that we can develop web applications much, much faster than we can develop Swing applications. Of course this is an over-simplification, but in general a simple web application conforming to HTML 3.2 specifications is much easier for us to develop, and hence our cost estimate and delivery times will be scaled down for this environment.

Another factor you'll be able to measure is the impact of new technology and tools on your development costs. As I mentioned, we deliver Web applications much faster than Swing applications, so a 500 FP Web application will be developed faster than a 500 FP Swing application. Although the size (amount) of functionality we're delivering to the customer is equivalent, the technology that we're using to deliver the applications is different, and for us, web applications are much less expensive.

That being said, I've found that other factors, including project managers and customers can also be a major influence on the overall development time and cost. For instance, when it comes to customers, it's much easier to work with a small team of customers that agree on what they want an application to deliver, versus a large committee with different opinions. The large committee is going to take more time during the requirements process, and IMHO is going to be subject to a larger scope creep during development.

In summary, given a little bit of time and statistics, your organization can rapidly begin applying FPs to use these same metrics. Over time, your cost and time estimates will get much more accurate. And, as you bring new technologies into your portfolio, you'll be able to look at these metrics and see the positive (or negative) correlation of new technology.

Bonus - Application development repositories and "magic formulas"

In the last section I started getting into the real usefulness of FP counting – it leads to many different metrics that you can use to measure and improve your software development process. Assuming that you take this to heart, and become really proficient at FP counting, let me show you a few magic formulas that people in the cost estimating and metrics world use.

FP repositories

As I mentioned early in this paper, if you don't have an established metrics program in house, one of the things you can do early in the process is use existing FP repositories of application data to assist in your estimating efforts, and compare your development process to other teams and other applications.

Here are three sources I have used with varying degrees of success:

Construx Estimate

Construx is the name of a company in Washington led by Stephen McConnell. They have created an interesting tool named Estimate that you can use to compare the time and cost of your software development process. A prime time and cost of your software development process. A prime input variable is function points, and I've found that with a little bit of calibration, I can get our projects to correlate with the projects in their built-in database. If I don't have experience with a new technology, such as a recent development project where we used C++, I use this tool to see the differences in metrics between Java and C++.

This is generally a very good tool, although the level of control is very coarse. As an example, I can't play with many detailed factors, such as "Project Manager experience".

Cocomo II

Cocomo II is an application from Barry Boehm and many other contributors. It is essentially a DOS-based application, but despite showing its age in that regard, I like what they've done with all the correlation formulas and input controls. It has a very fine level of control, which lets me tweak all types of variables. Over the last 18 month I've been able to use the concepts from Cocomo II with increasing accuracy.

ISBSG

ISBSG stands for International Software Benchmarking Standards Group, Ltd. They have worked with IFPUG to create a repository of 789 software development projects around the world. You can find more information, and purchase the ISBSG database in various formats through IFPUG. I have only looked at printed versions of their reports at this point, but the level of detail they provide seems very useful.

Three magic formulas
Three magic formulas

The last thing I'm going to share here are three things I refer to as "magic formulas", mostly because I don't know how the authors of these formulas arrived at them. I'll have to assume that they had some data at their disposal, and came up with these formulas to match their data.

**Project Duration**

The first formula is credited to Barry Boehm. The formula states that the duration of a project is equal to 2.5 times the cube root of the work months, or:

\[
\text{Project Duration} = 2.5 \times \left( \text{Cube Root of Work Months} \right)
\]

where Work Months is defined as:

\[
\text{Work Months} = \frac{\text{Project Work Effort (Hours)}}{\text{Hours per Month}}
\]

So, if a project is estimated to take 27 work months, then the project duration can be expected to be

\[
\text{Project Duration} = 2.5 \times \left( \sqrt[3]{27} \right)
\]

or 7.5 months.

**Optimum Staffing Size**

The origin of this formula is currently unknown, but it goes like this:

\[
\text{Optimum Staffing Size} = \sqrt{\text{Work Months}}
\]

Here, the square root of 27 is 5.2.

**Minimum Duration**

The last magic formula states that the minimum duration of a software project is:

\[
\text{Project Duration} = 0.75 \times \left( \text{Cube Root of Work Months} \right)
\]

In my example, this yields 0.75 \times 3, or 2.25 months.

**Summary**

How does this compare to what I've measured? In my earlier example I stated that we worked 540 man-hours 10 weeks. At 160 man-hours per month, this is 3.375 months. Applying the three magic formulas I would arrive at these numbers:

1. Minimum Duration = 0.75 \times 1.5 = 1.12 month.
2. Project Duration = 2.5 \times 1.5 = 3.75 months.
3. Optimum Staffing Size = 1.84 developers.

Before using any of these "magic formulas" in practice, you might try reverse-engineering them like this against your own projects, and see if they work for you.

Summary

The concept of Function Points was invented by Alan Abrecht 25 years ago, yet they remain a mystery to most developers today. That's unfortunate, because once you know the functional size of an application, you've opened a new door to accurate project cost estimation and other useful metrics.

A recap of the process of Function Point Analysis shows that Function Points:

1. Are measured from the user's perspective (not a developer's perspective).
2. Are independent of the technology used to develop the application.
3. Are low cost, adding less than a 1% overhead to your process.
4. Are repeatable, as studies have shown that certified function point counters can come within 10% of each other.
5. Are "use case friendly", because counting function points typically corresponds to processes defined in use cases.

Using Function Point Analysis helps you more accurately estimate:

1. Project cost
2. Project duration
3. Optimum project staffing size

An accurate counting of function points leads to a wealth of valuable statistics that can be used to improve the development process, including:

1. Number of developer hours per FP.
2. Number of total hours per FP.
3. Cost per FP.
4. Number of FPs per month/week/day.
5. Number of bugs/defects per FP.
6. Number of bug/defect hours per FP.
7. Productivity increases (or decreases) due to technology changes.

These metrics, and others like them, can be used as part of the feedback loop to improve your software development lifecycle.

For more information, visit IFPUG.org.