



FPA and DW

White Paper
March, 2000

A Minerva SoftCare White Paper

.....

Function Point Analysis And Data Warehousing



Content

<u>PREFACE</u>	<u>3</u>
<u>INTRODUCTION</u>	<u>4</u>
<u>FUNCTION POINT ANALYSIS</u>	<u>5</u>
<u>IDENTIFYING AND WEIGHING FUNCTION POINTS</u>	<u>7</u>
<u>FPA GENERAL APPLICATION CHARACTERISTICS</u>	<u>9</u>
<u>FPA AS A TOOL</u>	<u>10</u>
<u>DIFFERENCES</u>	<u>11</u>
<u>COST-OF-OWNERSHIP</u>	<u>11</u>
<u>DATA WAREHOUSE PHASES</u>	<u>14</u>
<u>FIRST STEP</u>	<u>15</u>



Preface

One of the most difficult aspects of a systems analyst's job is the accurate estimation of a project, in sizing, development time and end-user value. To fund and staff projects there is a need for an objective way to measure the size and complexity of a project before implementation. Only with such measurements can organizations and managers scope projects accurately, allocate staff and budget appropriately, and set realistic management expectations. Function Points have become the standard measure of the relative size of an application. Function Points measure how many individual "functions" are delivered to the end-user, taking into account: inputs, outputs, queries and access of both internal and external data sources.

Data Warehousing is a new phenomenon which very few people know and have the experience to estimate projects. Even fewer have the ability to really execute. This technical White Paper tells you what you need to know about Function Point Analysis in order to derive the maximum benefit from Data Warehousing and advancements in the rapidly evolving field of software development.

Minerva SoftCare N.V.



Introduction

Productivity evaluation methods -counting either lines of code or specific coding language constructs- is useless as estimators, because the programs have to be written first. Even with existing code, the disappointing and often grossly inadequate results are all too well known. Many times, better results could be achieved by using the following methods:

- Bull's eye Method;
- Random Generator;
- Experience Method.

Even if these processes were consistent and accurate in their results, they are not repeatable to others. More importantly, you cannot manage what you cannot measure.

The measurement of programming has for more than 45 years been the weakest link in the whole process of software development projects. When the common metrics used for programming are explored under controlled situations, we discover three mathematical paradoxes that have completely distorted the history of programming and concealed significant true progress:

Lines-of-code measures

penalize high level languages and often move in the wrong direction as productivity improves. They are not only difficult to apply, but also ambiguous and paradoxical even when applied carefully. An entire generation of software researchers assumed incorrectly that improving productivity meant increasing the number of lines that could be developed in a year, hence lowering the cost per source line.

Cost-per-defect measures

penalize high quality programs and always move in the wrong direction as quality improves.

Ratios-established-for-programming-subactivities

such as design, coding, integration, or testing often move in unexpected directions in response to unanticipated factors.



These paradoxes -all caused by lines-of-code measures that penalize high level languages like Minerva's MetaSuite and also falsely distort cost-per-defect measures- must be settled before a project is started. Projects can only be measured by precision instruments, which thus become the most important issue as well as the most fundamental.

Projects are rarely completed in time, within budget, or to the expectations of the end-user. Murphy's Law even described the value of careful project planning: It takes only twice as long to complete as expected, compared to three times for careless project planning! Fortunately, IBM's Allen J. Albrecht in 1979 developed (and in 1983 revised) an evaluation method known as Function Point Analysis (FPA). FPA is by far the most accurate and effective software metric ever developed.

Function Point Analysis

FPA is independent of the programming environment and its particular considerations. It is successfully implemented in more than 250 computer programming environments.

FPA is accurate and reliable within 10% for existing systems and 15-20% for planned systems.

- Business value of a system to the user
- Project size, cost, and the development time
- MIS shop programmer productivity and quality
- Maintenance, modification and customization effort
- Feasibility of in-house development
- Fourth generation language (4GL) implementation benefits
- Cost-of-ownership, corrective, adaptive and perfective maintenance.

The FPA method is easily learned and does not take a long time to master. So how does FPA work? FPA evaluates end-user system value; it is closely related to an application's requirement definition. A Function Point is defined as one end-user business function. Therefore, a program rated as having "x" function points delivers "x" business functions to the user. The evaluation process requires two major steps.



First, business functions made available to the user are identified and then organized into the following five groups:

- Outputs
- Inquiries
- Inputs
- Files
- Interfaces

Outputs

are items of business information processed by the computer for the end-user.

Inquiries

may be considered a simple output; more precisely, they are direct inquiries into a database or master file that look for specific data, use simple keys, require immediate response and perform no update functions.

Inputs

are items of business data sent by the user to the computer for processing and to add, change, or delete something.

Files

are data stored for an application, as logically viewed by the user.

Interfaces

are the data stored elsewhere by another application, but used by the one under evaluation.

This is the order in which these functions should be analyzed when designing a new system. The next step is to adjust the raw total according to factors in the production environment, collectively referred to as the General Application Characteristics.



Identifying and weighing function points

For the weighing of the function points in a given application, the following types are used:

OT - Output Types

IT - Input Types

QT - Query Types

FT - File Types

ET - External Types

Output (OT)

Each unique user data or control procedurally generated that leaves the application boundary is counted. This includes reports and messages sent not only to the user, but also to other applications and their users. An output is considered unique if:

1. It has a different format, or
2. It has the same format as another output, but requires different processing logic- such as different files being accessed, different calculations made or different processing procedures used.

The following weighing factors can be used: **(IT)**

Each unique user data or control input that enters the applications boundary and updates (adds to, changes or deletes from) a logical internal file, data set, table or independent data item is an input type.

An input is considered unique if:

1. It has a different format
2. It has the same format as another input (for example, screen appearances are identical), but requires different processing logic- such as different functions being performed (the same entities are modified in different ways) or a different logical internal file or data item is modified.

The following weighing factors can be used: **(QT)**

Each unique input/output combination in which the online user-defined input causes and generates an immediate online output is considered an Inquiry Type. An inquiry is considered unique if:

1. It has a format different from others in either its input or output portions, or
2. It has the same format, both input and output, as another inquiry, but requires different processing logic in either.



The following weighing factors can be used:

Output	1-5 data referenced items	6-19 referenced items	20 or more data referenced items
0 to 1 referenced file	Simple (4)	Simple (4)	Average (5)
2 to 3 referenced files	Simple (4)	Average (5)	Complex (7)
4 or more referenced files	Average (5)	Complex (7)	Complex (7)
Input	1-4 data referenced items	5-15 data referenced items	16 or more data referenced items
0 to 1 referenced file	Simple (3)	Simple (3)	Average (4)
2 referenced files	Simple (3)	Average (4)	Complex (6)
3 or more referenced files	Average (4)	Complex (6)	Complex (6)

Files (FT)

Each major logical group of user data or control information maintained entirely within the application boundary is counted as a function point. The key to proper file complexity determination is the number of logical relationships in which an entity participates and not the physical-oriented number of record types.

The following weighing factors can be used:

	1-19 data referenced items	20-50 referenced items	51 or more data referenced items
1 logical record format/relationship	Simple (7)	Simple (7)	Average (10)
2-5 logical record format/relationship	Simple (7)	Average (10)	Complex (15)
6 or more logical recordformat/relationship	Average (10)	Complex (15)	Complex (15)



Interfaces (EI)

Each major logical file or other logical group of user-approved data or control information within the application boundary that is sent to, shared with or received from another application. Files shared between applications are counted both as files and as interfaces within each application, if they are actually used in both as files and as interfaces within each application. Following weighing factors can be used:

	1-19 data referenced items	20-50 data referenced items	51 or more data referenced items
1 logical record format/relationship	Simple (5)	Simple (5)	Average (7)
2-5 logical record format/relationship	Simple (5)	Average (7)	Complex (10)
6 or more logical record format/relationship	Average (7)	Complex (10)	Complex (10)

FPA General Application Characteristics

In the previous part, we identified and counted function points. The concern was the “What” of things - that is, with the logical system. Here we are going to discuss the “How” of things. Although function Point Analysis is an objective measurement tool and independent of the software environment in which the software project is housed, the result is, off course, influenced by the specific characteristics of the environment in which it is running. The treatment of the physical environmental influences is provided by the 14 adjustment factors we shall discuss shortly. As we do, we keep the following rule clearly in mind: The adjusted function point count is the logical system and therefore always remains exactly the same regardless of where it is implemented and the language in which it is written!

The 14 adjustment factors we shall soon learn represent applicable elements of the physical system, vary from one shop to another and adjust the value of the logical system accordingly.



We must adjust the raw total calculated according to the contribution -as seen by the users- by each of the following production environment factors that influence the system as a whole and not just a particular function:

- | | |
|-----------------------------------|--------------------------------------|
| 1. Data communications | 8. Online update |
| 2. Distributed data or processing | 9. Complex processing |
| 3. Performance objectives | 10. Re-usability |
| 4. Heavily-used configuration | 11. Conversion and installation ease |
| 5. Transaction rate | 12. Operational ease |
| 6. Online data entry | 13. Multiple-site use |
| 7. End-user efficiency | 14. Facilitate change |

As was the case with functions, count and fairly evaluate only those factors which, clearly benefit the end-user, factors with which the end-user agrees.

FPA as a tool

Of itself, Function Point Analysis is a project sizing technique- the very best there is- and you may be wondering how big a system is. FPA is the only accurate sizing technique available for use before coding even begins, and is proven accuracy of $\pm 15-20\%$.

The question that remains is "How to go from Function Points to Hours Worked". Let us first look at some general productivity guidelines. Just as programmer efficiency varies widely, so does programmer productivity. Well-known studies have estimated one COBOL function point to require anywhere from 8 to 50 hours on average to code-usually around 20; the range was from 3 to 87 hours per Function Point. Management must, of course, determine and use the proper values for its own staff. Modern tools bring down the average for COBOL to around 10 hours per Function Point.

Productivity not only varies with each programmer, but also with the size of a project. Because of increasing complexity and the need for additional planning and coordination, required COBOL or other programming language time for all Function Points increases approximately 4% with each additional Function Point.



The following table shows the average number of hours (subject to variances of productivity levels, environmental productivity tools installed and used, application size and other factors) required to produce one Function Point in 5 languages.

Tool	Hours per Function Point
COBOL	10
PL/1	8
Script language	5
Visual Basic	4
MetaSuite	2

Differences

From the above table, one can see that there are vast differences to be expected between the different tools. The difference between a third generation tool like COBOL and a 4th generation tool like MetaSuite is a factor of 5 at least. These differences become even more evident when the total cost-of-ownership is calculated. Investigations have shown that there is a relationship between the development costs and the maintenance costs. These investigations have shown that if an application has a cost-of-ownership of 100%, the development costs are then 30% and the maintenance (corrective, adaptive and perfective) costs are 70%.

Cost-of-ownership

Cost-of-ownership is considered the total cost of an application. This means that it not only includes the development costs, but also the maintenance costs. The maintenance costs includes corrective, adaptive, and perfective maintenance. As a general rule total cost can be apportioned as, building 30% and maintenance 70%. To see the impact of the different tools we use the following example.



In the calculation we use the following assumptions:

- Project size is 1500 Function Points
- Programmers rate is \$ 100,- per hour.
- Development represents 30% of total costs.
- Maintenance represents 70% of total costs.

To calculate the different development costs per programming environment we use the previous table:

Tool	Hours per Function Point
COBOL	15.000
PL/1	12.000
Script language	7.500
Visual Basic	6.000
MetaSuite	3.000

As can be seen from this table the total estimated time needed to complete this project ranges from 15,000 hours when COBOL is used to 3,000 hours when a 4 GL environment like Minerva's MetaSuite is used. If we take the hourly rate of \$ 100,- into account, the differences are even more startling.

Tool	Estimated \$
COBOL	1.500.000
PL/1	1.200.000
Script language	750.000
Visual Basic	600.000
MetaSuite	300.000



It is obvious that the right tool can mean the difference between a project and no project, between high and low costs, between success and failure. Building a new application that fulfills the user's needs is one objective. A second objective is to keep it in sync with the user's needs. This means that maintenance is necessary. Let us look at the consequences per tool. As mentioned earlier we use the 30/70 rule for the calculations.

First we calculate the estimated cost-of-maintenance:

Tool	Estimated cost-of-maintenance
COBOL	3.500.000
PL/1	2.800.000
Script language	1.750.000
Visual Basic	1.400.000
MetaSuite	700.000

As we can see, the absolute differences are becoming even more significant. They range from \$ 700,000,- per year to a whopping \$ 3,500,000,- per year. This is a difference of \$ 2,800,000,-per year. Moreover, keep in mind that the development costs were estimated at \$ 300,000,-when Minerva's MetaSuite environment is used!

If we take a closer look at the real estimated cost-of-ownership, we see even bigger differences.

Tool	Total cost-of-ownership
COBOL	5.000.000
PL/1	4.000.000
Script language	2.500.000
Visual Basic	2.000.000
MetaSuite	1.000.000



Looking at the table above we can conclude that the absolute difference between the tools is estimated at \$ 1,000,000,- and \$ 5,000,000,-. Still remember the initial cost of only \$ 300,000,-. It is safe to assume that the right choice of programming language is imperative for the success of the project. Not only in the time needed to complete the project but also the total cost-of-ownership. As can be seen from the results in this example, it is obvious that MetaSuite is a highly specialized tool for Data Warehousing and Data Migration. With MetaSuite it is possible to generate solutions, effectively, accurately and efficiently.

Data Warehouse Phases

To use the Function Point Analysis method in data warehousing, Minerva SoftCare uses the expertise gained in the multitude of heterogeneous data warehouse projects in which it has involved. The evaluated projects being used as a reference to calculate the environmental weighing factors. These lead to the accurate results needed to estimate the size and impact of the Data Warehouse Project.

A project is divided in four main steps:

- Data Awareness Phase- what data do we have?
- Information Awareness Phase - what information is needed?
- Mapping Phase- how do we need to extract, transform, clean and enrich the data to create the information needed?
- Maintenance Phase- how do we keep up with the ever changing requirements of information?

Each phase has it's unique set of specific circumstances. The Function Point Analysis gets the insight information you need for an estimate on the impact of a Data Warehousing Project. Minerva SoftCare has the tools, expertise and more important the ability to execute.



First step

Each project starts with a first step. For a Data Warehousing Project, a good first step is to use the expertise of Minerva SoftCare to make your project a success. A long and representative list of successful projects is proof of our ability to execute.

As you can see from the before mentioned Function Point Analysis the differences in impact can be huge. Making the wrong choices in for example the architecture of your Data Warehouse Environment can mean the difference in being a success or being the company's fool.

A critical success factor for a Data Warehouse is the ability to respond to the ever changing needs of the managers with right and timely information. If the chosen architecture makes use of for example COBOL, the time needed to make the requested changes can be of a deadly hindrance to a Data Warehouse. On the other hand, if the architecture has a flexibility where the ever changing needs of information are answered in a timely and accurate manner the Data Warehouse is used and probably is going to be used even more.

As you can see, Function Point Analysis is not only handy when estimating the project size and impact, but can be used to see the influences of the different architectures that are possible.

For more information you can contact:

Minerva SoftCare N.V.

Telephone: +32 (0)15 62 71 00

E-mail: info@metasuite.com

<http://www.metasuite.com>

