Seminar on Software Cost Estimation:

Function Points

Institut für Informatik, Universität Zürich

Prof. Dr. Martin Glinz
Arun Mukhija

WS 2002/03
10. December 2002

Author:

Christoph Suter
Hoffeld 2
8057 Zürich

fels@datacomm.ch

1 Introduction

In the year 1979, Albrecht, working at IBM, presented the Function Point Method at a conference in California. There soon was a big interest in this new method, because of its clear advantages to estimate effort and because of the lack of other reliable methods. A few years later, 1982, Tom DeMarco presented his own method for Function Point counting, which he developed independently from Albrecht. His Method is quite similar to Albrecht's but the latter is more comprehensive.
As the FP Method was first used for Information Systems, people thought of it as only being useable for this kind of Software. For that reason, a vast number of modified Concepts were developed by different people over the years. Until today, about 35 different versions are known to be applied by the industry.
Until 1985, Albrechts method was maintained at IBM. In the year 1986, the International Function Point Users Group (IFPUG), took over the responsibility for the concept from IBM. The IFPUG is a US-American non profit organization, which takes care about modifications and maintenance of the FP concept. This paper focuses on the version 4.1 of the IFPUG, published in 1994.

2 Concept

The Function Point Method measures the functionality of software. This is done from the user's point of view: elementary, user identifiable processes are determined, weighted with some complexity factors and summed up to get a final number. Because the selection of the processes is based on the users view, the result is independent from a lot of technical and developmental details like hardware, programming style or programming language for example.

3 Procedure

The Procedure for counting an application is as follows: First the type of the Count is specified. After that, the application boundary has to be identified. Next the different function types are identidfied, counted, weighted and summed up to get an unadjusted Function Point Value. Finally this Value is being adjusted by some Value Adjustment Factor to get the final Total Function Point Count.

3.1 Counting Types

There are three possible types: Development, Enhancement and Application. There is a strong relationship between these three types. They reflect somehow the life cycle of a piece of software: First, the software is being developed and installed. There may be some functionality like installation or conversion routines, which is not used any more after the software has been installed once. When counting development types, this extra functionality is included in the total Function Point Count. Once the application is up and running, we can count the core functionality of the application as an application type count. If the software ever has to be modified, the changes in functionality are counted as enhancement type count.

Once the enhancement project is terminated, a new application type count results out of the former application type count and the enhancement type count.
Figure 1 provides an overview of the different counting types

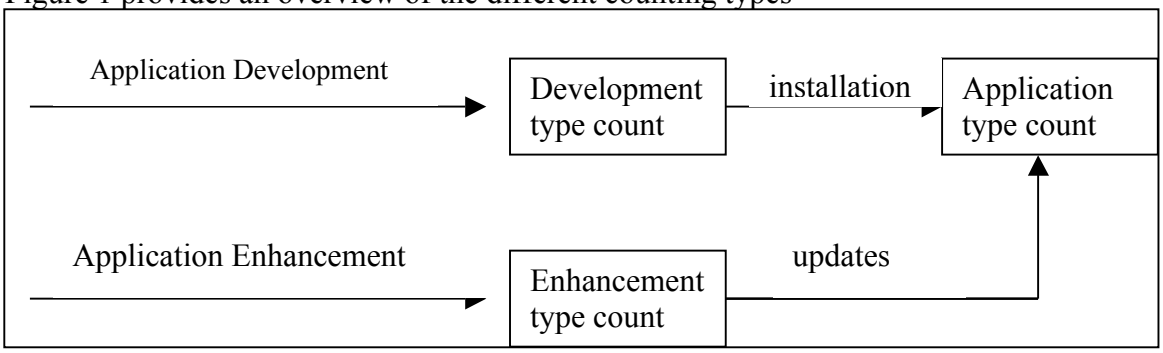

Figure 1: Counting types

3.2 Application Boundary

After specifying the counting type, the application boundary has to be determined. It distinguishes functionality that is provided by our application from the one that comes from outside. An example for identifying the application boundary is given in Figure 2.

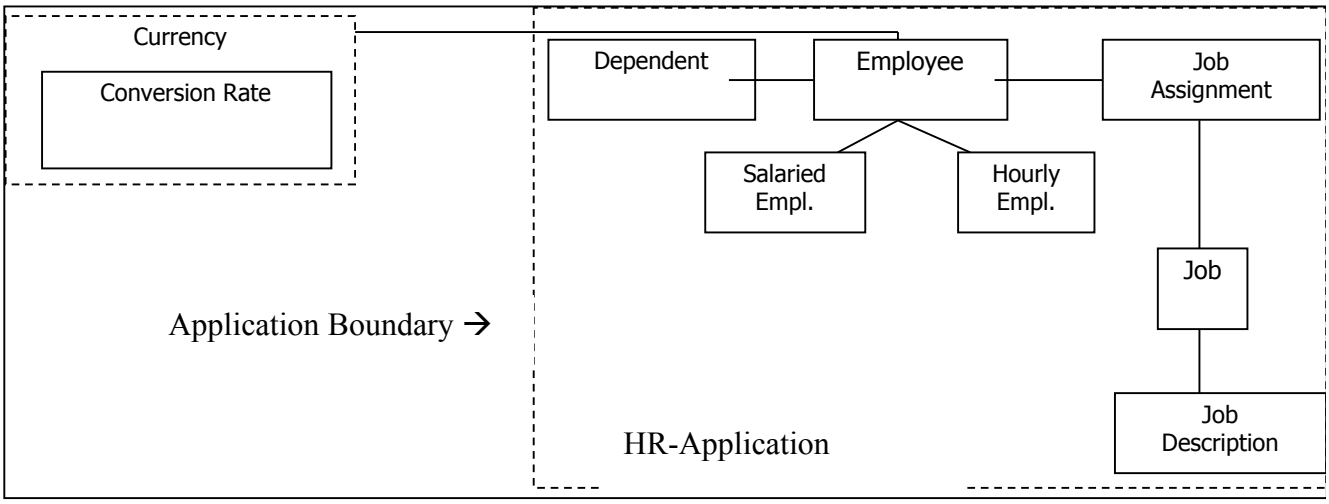

Figure 2: Application Boundary

3.3 Function Types

There are two different Function Types: Data Functions and Transaction Functions. The first are responsible to store and maintain data, the latter provide the functionality to get, change and/or send the data to a user or another application and to maintain the Data Files.

3.3.1 Data Functions

There are two Data Functions: Internal Logical File (ILF) and External Interface File (EIF). The word file does not necessary mean a physical file on a hard disk. An ILF or EIF is more

like a container, which contains logically related data. Sure this data will be somewhere stored on a hard disk, but that "physical" file is not of interest for the process.

3.3.1.1 Internal Logical File (ILF)

An ILF is a group of logically related data or control information that is maintained by the application itself. To be an ILF, the group has to fulfil both of these rules:

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.

The first rule demands that there is a logical dependence in between the information which is in the group, and that this dependence is user identifiable. This helps to avoid counting too many ILFs: If the user needs the information from two groups to do an elementary process, then the two groups count only as one ILF.
The second rule states that the information in an ILF has to be dealt with by an elementary process. This process has to be self-contained in the sense that no further information is needed to accomplish the process, but also that all of the information in the ILF is mandatory for the process. In the example given in Figure 2, the Entities Job and Job Description together are count as one ILF, Employee as another one.

3.3.1.2 External Interface File (EIF)

An EIF, like the ILF, is a group of logically related, user identifiable data or control information. The major difference to the ILF is that the EIF is not maintained within the application boundary. It serves as an information container to the application. The information of an EIF can be used to generate some output or it can be part of an input process. For a money institute, the money exchange rates, which are calculated perhaps once a day by another application, could be an EIF.
An EIF must comply with all of the following rules:

1. The group of data or control information is logical and user identifiable
2. The group of data is referenced by, and external to, the application being counted
3. The group of data is not maintained by the application being counted.
4. The group of data is maintained in an ILF of another application.

As the name already implies, an EIF must not be maintained within the application – otherwise it would be counted as an ILF. It must be referenced by our application – otherwise it would not make sense to count the group of information as functionality. The last rule is also obvious: if our application does not maintain the file, it has to be done by another one.
In the example given in Figure 2, the Conversion Rate Entity makes one EIF.

3.3.1.3 Complexity and Contribution: Data Element Types (DETs) and Record Element Types (RETs)

Once the File types are identified, they are further investigated on to get some measure for the complexity of the file structures. Therefore one distinguishes between Data Element Types and Record Element Types.

DETs

A DET is a unique, user recognizable, non repeated field. A DET is like an elementary piece of information of an ILF or EIF. Referring to the example in Figure 2, each of the Employee type's attributes like Name, Address or Employee-Number counts as one DET. Again, it is important to keep the user's point of view: if our application always deals with a set of data elements as a whole and does not pay attention to the single elements, this set is counted as only one DET. DETs are counted based on the following rules:

- Count one DET for each unique user recognizable, non-repeated field maintained in or retrieved from the ILF or EIF through the execution of an elementary process
- When two applications maintain and/or reference the same ILF/EIF, but each maintains/references separate DETs, count only the DETs being used by each application to size the ILF/EIF.
- Count a DET for each piece of data required by the user to establish a relationship with another ILF or EIF.

RETs

A RET is a user recognizable subgroup of data elements within an ILF or EIF. This subgroup may be either optional or mandatory. If the group is optional, the user can, but does not need to, use the group during an elementary process concerning the ILF/EIF. Mandatory groups force the user to join one of the groups during the process. In the Account Entity, the kind of account (like current account or savings account) would be mandatory, whereas a list of other accounts held by the same person would be an optional group. Each mandatory or optional subgroup of the ILF/EIF counts as one RET. If and only if the ILF/EIF has no subgroups at all, the ILF/EIF itself is counted as one RET. For the example given in Figure 2, the entities Salaried Employee and Hourly Employee are two mandatory groups and counted as one RET each. Another RET is counted for the optional group Dependent.

3.3.1.4 Determining the complexity and contribution value of an ILF/EIF

Once the DETs and RETs for an ILF/EIF are counted, the file is being assigned a complexity based on the numbers of DETs and RETs it contains. The complexity can be low, average or high. Each of these means a different number of unadjusted Function Points to add. ILFs generally get more Function Points. They have to be maintained by the application and so their implementation will be more effort normally.
For an ILF/EIF, the complexity ratings are:

	1-19 DETs	20-50 DETs	51 + DETs
1 RET	Low	Low	Average
2-5 RETs	Low	Average	High
6+ RETs	Average	High	High

One can clearly see the dependence of the number of DETs and RETs concerning the complexity of the ILF/EIF.
Now, the file is being associated a number of unadjusted Function Points, based on the complexity determined. For an ILF, the Function Point contribution is:

Complexity Rating	Unadjusted Function Point value
Low	7
Average	10
High	15

For an EIF, the following numbers of unadjusted Function Points are counted:

Complexity Rating	Unadjusted Function Point Value
Low	5
Average	7
High	10

The tables are all taken from [IFPUG00].

Once each of the ILFs/EIFs have been assigned an unadjusted Function Point value, these values are normally written in a table and summed up.

3.3.2 Transactional Functions

The transactional functions provide the functionality to the user to process data. One can understand the transactional functions as the tool to work with the information stored in the ILFs/EIFs. There are three types of transactional Functions: External Inputs (EIs), External Outputs (EOs) and External Inquiries (EQs). EIs “import” some information from outside the boundary to the application, whereas EOs and EQs normally “export” information from inside the boundary to the user or another application. Each of the transactional function is has to be an elementary process for the user.

3.3.2.1 Elementary Processes

An elementary process is defined as the smallest unit of activity that is meaningful to the user. It is self-contained and leaves the application in a consistent state.

3.3.2.2 External Inputs (EIs)

The primary intent of an external input is either to maintain one or more ILFs or to alter the system state through control information through an elementary process. The information has to come from outside the boundary and if the purpose is not to alter the system state, at least one ILF has to be maintained by the process. Either the processing logic has to be unique (different from other EIs), the set of data elements has to be different from any other EI, or the ILF/EIF referenced has to be different from the files referenced by other EIs in the application.

3.3.2.3 Complexity and Contribution: Data Element Types (DETs) and File Types referenced (FTR)

As it is the case with ILFs/EIFs, each of the transactional functions is being assigned a complexity. It is based on the data element types (for a definition see above) and on the file types referenced by the processing logic of the respective transaction function.
A file type referenced (FTR) is simply an ILF read or maintained or an EIF read by the process. Each file being referenced during the external input (ILF/EIF) counts as one FTR.

For each user recognizable, non-repeated field that enters or exits the application boundary and that is required to complete the external input, one DET is counted for this EI. Note that only fields crossing the application boundary are counted. If the application generates some feedback in form of a message (task completion, task error, task progress), this counts as one additional DET. If there are multiple methods for invoking the same logical process, all together count as one additional DET.

3.3.2.4 External Outputs (EOs) and External Inquiries (EQs)

The primary intent of these two transactional functions is to present information to a user. They consist of an elementary process that sends data or control information external to the application boundary. Either the processing logic, the set of data elements or the ILFs/EIFs referenced have to be different from the one of other EOs or EQs.
The difference between an EO and an EQ is that for the external output, there has to be some calculation or further action involved to get the result which is presented to the user. This can either be some derived data that the process creates, an ILF maintained by the process, at least one mathematical calculation or altering the system state of the application.
An external inquiry retrieves some information from an ILF/EIF from outside the boundary of the application. It does not create derived data, nor does it maintain an ILF or use any mathematical functions to get the information presented to the user. It is also not allowed to alter the system state of the application.

3.3.2.5 Complexity and Contribution: Data Element Types (DETs) and File Types Referenced (FTRs)

For external outputs and external inquiries, one FTR is counted for each ILF/EIF read during the elementary process. If an external output maintains an ILF, it is also counted as one FTR for the EO. Maintaining and reading the same ILF counts only as one FTR for the EO.

DET counting for EOs and EQs is done basically the same way as for EIs: Each user recognizable, non-repeated field that enters the application boundary and is and is necessary to complete the process counts as one DET. As for EQs and EOs data will also exit the application boundary, each field that exits the boundary counts as one DET. If the same DET enters and exits the boundary, it is counted only once. If the system is able to send some kind of response outside the boundary, this makes also one DET. Finally the maybe diverse possibilities to invoke a process (buttons, menu bars) together count as one DET.

3.3.2.6 Determining the complexity and contribution value for a transactional function

After the number of DETs and/or FTRs have been counted for each transactional function, the numbers achieved are mapped to a complexity. The following tables explain this mapping:

EI:

	1-4 DETs	5-15 DETs	16 + DETs
0-1 FTR	Low	Low	Average
2 FTRs	Low	Average	High
3+ FTRs	Average	High	High

EO/EQ:

	1-5 DETs	6-19 DETs	19+ DETs
0-1 FTR	Low	Low	Average
2-3 FTRs	Low	Average	High
4+ FTRs	Average	High	High

Now we can derive the unadjusted Function Point contribution of each transactional function based on the following tables:

EI/EQ:

Functional Complexity Rating	Unadjusted Function Points
Low	3
Average	4
High	6

EOs:

Functional Complexity Rating	Unadjusted Function Points
Low	4
Average	5
High	7

The tables are all taken from [IFPUG00].

3.4 Value Adjustment Factor Determination

When all data functions and transactional functions have been recognized, weighted and counted as described above, the one thing left to do is to determine the value adjustment factor (VAF) for the application being counted. The VAF takes the "environment" of the application into account. This is done by weighting a total of 14 different influence factors for the application, that are being evaluated on a scale from zero (no influence) to five (strong influence). I will only mention the different factors here, for a further explanation refer to [IFPUG00]. The 14 factors are:

1. Data Communications
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

If for each of the influence factors has been weighted with a number from zero to five, all these factor influences are summed up to the total degree of influence (TDI).
The TDI is then inserted into the following equation:
VAF = (TDI * 0.01) + 0.65
VAF designates the Value adjustment factor. As we can see, this factor may be in the range from 0.65 (no influence determined for all of the 14 points above) to 1.35 (highest influence for all of the 14 points).

3.5 Calculate Total Function Point Count

The last step in counting function points is to determine the total, adjusted function point count for the application, based on the total of unadjusted function points and on the value adjustment factor calculated. Since FP counting can be done for development, enhancement and application projects, the final steps can be done in three different ways. I will not explain the different calculation formulas here, as they are beyond the scope of this seminar. The core

of each counting type formula is to multiply the unadjusted Function Point Value (UFP) by the Value Adjustment Facor (VAF) to get the adjusted Function Point Value (FP).

$$FP = UFP * VAF$$

For further explanation refer to IFPUG[00].

4 Discussion

One big advantage of the FP concept is, that also for past projects, which were not counted, an FP value can be calculated. This is done normally by "backfiring". Backfiring means the calculation of Function Points based on the Code of a program and is highly automated by a set of tools. Once there has been built a database of reasonable size from past projects, the method can also give hints about other factors of developing software like efficiency. It can even be used to detect and investigate deviations in the time schedule of a project.
Besides this broad functionality and wide spreading on the market, there are two advantages to be mentioned: FP Counting is very reliable and normally gives much better results than e.g. LOC-estimating and it can be done very early in the software process, because all it needs are the requirements of the user.
On the other hand there are some disadvantages. The first to mention here is the vast amount of different versions which have been developed. Most of the differences refer to the number and meaning of the different factors used to calculate the Value Adjustment Factor. Adjustments were done with the intention to better suite the circumstances of different projects like for example Object Oriented Programming. This yielded to versions differing substantial in their results. Sometimes there are not even rules for converting the results from one method to another. The worst case in this context is a FP value without statement, which method was used to calculate it. This result is nearly useless. A second problem of the method is the high effort necessary to do some correct counting. Effort here does not only include time, but also know-how. The IFPUG states that only trained and certified people should do FP counting to guarantee reliable results. IFPUG offers some certification programs, but a lot of other firms, providing their own method don't.

5 Outlook

There is an urging demand for standardization of the FP method. If the community does not manage to develop a standard which does cover all needs for the different kinds of software projects and to establish this standard, the FP method might severely lack acceptance in a few years. The IFPUG as the most important organization in this context is also often criticized: Requests for changes or new functionality has to be sent to the IFPUG, which then decides whether to implement these changes or not. It seems that requests coming from non US firms do not have a real chance of being accepted by the IFPUG.

6 Conclusion

The FP method is one of the most well known methods to estimate the functionality of a software project. It is supported by almost every tool that is available in the market and can be

done very early in the development process. If there is a database of reasonable size from past projects, the FP Method normally yields very reliable results.
As applying the FP Method takes up not so little amount of time and hence also generates some effort, it is not reasonable to apply it for small projects. Critical for using the method at all are the qualifications of the personnel doing the counting and also the mentioning of the version used.

7 List of References

[IFPUG00] IFPUG (2000). *Function Point Counting Practices Manual, Release 4.1*.1. International Function Point Users Group. Westerville, Ohio (USA).

[IFPUG94] IFPUG (1994). Function *Point Counting Practices: Case Studies, Release 1.0*

International Function Point Users Group. Westerville, Ohio (USA).

Jones, T. C. (1998). *Estimating Software Costs*. New York: McGraw-Hill.

Seibt, D. (1987). Die Function-Point-Methode: Vorgehensweise, Einsatzbedingungen und Anwendungserfahrungen. *Angewandte Informatik* 1/1987. 3-11