Function Points or Lines of Code? – An Insight

Author:
Kurmanadham V.V.G.B. Gollapudi
Global Microsoft Business Unit
Wipro Technologies

Table of Contents

1. Objective

The objective of this article is to bring out the differences between the two most common sizing metrics: Function Points and Lines of Code. This also offers insight into the advantages of using Function Points for measuring the size of software.

2. Scope

The scope of the article is limited to highlighting the differences between Function Point Analysis (FPA) and Lines of Code, giving a brief introduction of both. This does not include any guidelines on Function Point Analysis itself. For details on FPA, readers are encouraged to refer the International Function Point Users Group (IFPUG) website.

3. Introduction

One of the most important activities in the early stages of software development is estimation. Size of the software, be it Function Points or Lines of Code, plays a pivotal role in this process, and forms the base for deriving number of metrics to measure various aspects of the software, throughout the development cycle. Hence measuring the size of software becomes critical. Though many other sizing measure are in practice such as, objects, classes, modules, screens, programs and so on, Lines of Code and Function Points are most widely used. Following sections explain these two measures in brief.

4. Function Points

Function Point Analysis is an objective and structured technique to measure software size by quantifying its functionality provided to the user, based on the requirements and logical design. This technique breaks the system into smaller components so they can be better understood and analyzed. Function Point count can be applied to Development projects, Enhancement projects, and existing applications as well. There are 5 major components of Function Point Analysis which capture the functionality of the application. These are: External Inputs (EIs), External Outputs (EOs), External Inquiries (EQs), Internal Logical Files (ILFs) and External Interface Files (EIFs). First three are treated as Transactional Function Types and last two are called Data Function Types. Function Point Analysis consists of performing the following steps:

- Determine the type of Function Point count
- Determine the application boundary
- Identify and rate transactional function types to calculate their contribution to the Unadjusted Function Point count (UFP)

- Identify and rate the data function types to calculate their contribution to the UFP
- Determine the Value Adjustment Factor (VAF) by using General System Characteristics (GSCs)
- Finally, calculate the adjusted Function Point count

Each of the components of Function Point Analysis is explained in brief in the following sub-sections.

4.1 External Input (EI)

External Input is an elementary process in which data crosses the boundary from outside to inside. This data may come from a data input screen or another application. The data may be used to maintain one or more internal logical files. The data can be either control information or business information.

4.2 External Output (EO)

External Output is an elementary process in which derived data passes across the boundary from inside to outside. Additionally, an EO may update an internal logical file. The data creates reports or output files sent to other applications. These reports and files are created from information contained in one or more internal logical files and external interface files. Derived Data is data that is processed beyond direct retrieval and editing of information from internal logical files or external interface files.

4.3 External Inquiry (EQ)

External Inquiry is an elementary process with both input and output components that results in data retrieval from one or more internal logical files and external interface files. The input process does not update or maintain any FTRs (Internal Logical Files or External Interface Files) and the output side does not contain derived data.

4.4 Internal Logical File (ILF)

Internal Logical File is a user identifiable group of logically related data that resides entirely within the application boundary and is maintained through External Inputs. Even though it is not a rule, at least one external output and/or external inquiry should include the ILF as an FTR.

4.5 External Interface File (EIF)

External Interface File is a user identifiable group of logically related data that is used for reference purposes only. The data resides entirely outside the application boundary and is maintained by external inputs of another application. That is, the external interface file is an internal logical file for another application. At least one transaction, external input, external output or external inquiry should include the EIF as a File Type Referenced.

4.6 Rating the Transactional and Data Function Types

Each of the identified components is assigned a rating (as Low, Average, and High). Transactional Function Types are given the rating depending upon the number of Data Element Types (DET), File Types Referenced (FTR) associated with them. Data Function Types are assigned ratings based on the number of Data Element Types (DET), and Record Element Types (RET) associated. A DET is a unique user recognizable, non-recursive (non-repetitive) field. A DET is information that is dynamic and not static. A dynamic field is read from a file or created from DETs contained in an FTR. A RET is user recognizable sub group of data elements within an ILF or an EIF. An FTR is a file type referenced by a transaction. An FTR must also be an internal logical file or external interface file.

The total number of EIs, EOs, EQs, ILFs, and EIFs, after applying the weights corresponding to the ratings (Low, Average, and High) will give the Unadjusted Function Point count (UFP).

4.7 General System Characteristics (GSCs)

The value adjustment factor (VAF) is calculated based on 14 General System Characteristics that rate the general functionality of the application being counted. The GSCs are: Data communications, Distributed data processing, Performance, Heavily used configuration, Transaction rate, On-line data entry, End-user efficiency, On-line update, Complex processing, Reusability, Installation ease, Operational ease, Multiple sites, and Facilitate change. The degree of influence of each characteristic has to be determined as a rating on a scale of 0 to 5 as defined below.

- **0:** Not present, or no influence
- **1:** Incidental influence
- **2:** Moderate influence
- **3:** Average influence
- **4:** Significant influence
- **5:** Strong influence throughout

Once all the GSCs have been rated, Total Degrees of Influence (TDI) is obtained by summing up all the ratings. Now, Value Adjustment Factor is calculated using the formula:
VAF = 0.65 + TDI/100

4.8 Final FP Count

After determining the Unadjusted Function Point count (UFP) out of transactional and data function types, and calculating the Value Adjustment Factor (VAF) by rating the general system characteristics, the final Function Point count can be calculated using the formula:
FP = Unadjusted Function Point count (UFP) * Value Adjustment Factor (VAF)

5. Lines of Code

Lines of code (often referred to as Source Lines of Code, SLOC or LOC) is a software metric used to measure the amount of code in a software program. LOC is typically used to estimate the amount of effort that will be required to develop a program, as well as to estimate productivity once the software is produced. Measuring software size by the number of lines of code has been in practice since the inception of software.

There are two major types of LOC measures: physical LOC and logical LOC. The most common definition of physical LOC is a count of "non-blank, non-comment lines" in the text of the program's source code. Logical LOC measures attempt to measure the number of "statements", but their specific definitions are tied to specific computer languages (one simple logical LOC measure for C-like languages is the number of statement-terminating semicolons). It is much easier to create tools that measure physical LOC, and physical LOC definitions are easier to explain. However, physical LOC measures are sensitive to logically irrelevant formatting and style conventions, while logical LOC is less sensitive to formatting and style conventions. Unfortunately, LOC measures are often stated without giving their definition, and logical LOC can often be significantly different from physical LOC.

There are several cost, schedule, and effort estimation models which use LOC as an input parameter, including the widely-used Constructive Cost Model **(COCOMO)** series of models invented by Dr. Barry Boehm. While these models have shown good predictive power, they are only as good as the estimates (particularly the LOC estimates) fed to them.

6. Function Points – Advantages & Disadvantages

6.1 Advantages

Function Point Analysis provides the best objective method for sizing software projects, and for managing the size during development. Following are some of the many advantages that FPA offers.

(a) Helps Comparison: Since Function Points measures systems from a functional perspective they are independent of technology. Regardless of language, development method, or hardware/platform used, the number of FP for a system will remain constant. The only variable is the amount of effort needed to deliver a given set of FP; therefore, Function Point Analysis can be used to determine whether a tool, an environment, a language is more productive compared with others within an organization or among organizations. This is a critical point and one of the greatest values of Function Point Analysis.

(b) Helps Monitor Scope Creep: Function Point Analysis can provide a mechanism to track and monitor scope creep. FP counts at the end of requirements, analysis, design, code, testing and deployment can be compared. The FP count at the end of requirements and/or designs can be compared to FP actually delivered. If the project has grown, there has been scope creep. The amount of growth is an indication of how well requirements were gathered by and/or communicated to the project team. If the amount of growth of projects declines over time it is a natural assumption that communication with the user has improved.

(c) Ease of Contract Negotiations: From a customer view point, Function Points can be used to help specify to a vendor, the key deliverables, to ensure appropriate levels of functionality will be delivered, and to develop objective measures of cost-effectiveness and quality. They are most effectively used with fixed price contracts as a means of specifying exactly what will be delivered. From a vendor perspective, successful management of fixed price contracts depends absolutely on accurate representations of effort. Estimation of this effort (across the entire life cycle) can occur only when a **normalized metric** such as the one provided by Function Points is applied.

Note: Here, my meaning of the word '**normalized metric**' is that Function Point truly accounts for the entire gamut of software development spreading across all the phases from Requirements through Testing. Whereas, LOC pertains to and is an outcome of only one of the phases.

(d) Handling Volatility: The advantage that Function Points bring to early estimation is the fact that they are derived directly from the requirements and hence show the current status of

requirements completeness. As new features are added, the function point total will go up accordingly. If the organization decides to remove features or defer them to a subsequent release, the function point metric can also handle this situation very well, and reflect true state.

(e) Use of Historic Data: Once project size has been determined in Function Points, estimates for Duration, Effort, and other costs can be computed by using historic data. Since FP is independent of languages or tools, data from similar past projects can be used to produce consistent results, unlike Lines of Code data which is much tightly tied to languages requiring many other parameters to be taken into account.

(f) Availability of Empirical Formulae: Unlike lines of code, FP can be used more effectively to develop many predictive formulae such as defect potential, maintenance effort which can help pinpoint opportunities for improvement. Caper Jones estimates that Function Points raised to the 1.2 power ($FP^{1.2}$) estimates the number of test cases. That is, test cases grow at a faster rate than Function Points. This is logically valid because as an application grows, the number of interrelationships within the application becomes more complex, requiring more test cases. Many empirical formulae have been suggested by Caper Jones which are in wide use among FP practitioners.

(g) Enables Better Communication: FP can help improve communications with senior management since it talks in terms of functionality rather than any implementation details, technical aspects, or physical code. Further more, Function Points are easily understood by the non-technical user. This helps communicate sizing information to a user (or customer) as well.

(h) Offers Better Benchmarking: Since FP is independent of language, development methodology, programming practices, and technology domain, projects using FP become better candidates for benchmarking across organizations and geographies.

6.2 Disadvantages

Function Points offer vast number of benefits by capturing the size of the software from its functionality standpoint. FPA does have some disadvantages. However, organizations can easily overcome these problems by practicing FPA consistently over a period of time.

(a) Requires Manual Work: Due to its very nature, Function Points have to be counted manually. The counting process cannot be automated.

(b) Necessitates Significant Level of Detail: A great level of detail is required to estimate the software size in terms of Function Points. Information on inputs, outputs, screens, database tables, and even records and fields will be required to perform FPA accurately. Typically this is not the case with any development project where the requirements are not clear to this level of detail, in the beginning.

(c) Requires Experience: Function Point Analysis requires good deal of experience if it were to be done precisely. FPA inherently requires sufficient knowledge of the counting rules, which are comparatively difficult to understand.

7. Lines of Code – Advantages & Disadvantages

7.1 Advantages

(a) Scope for Automation of Counting: Since Line of Code is a physical entity; manual counting effort can be easily eliminated by automating the counting process. Small utilities may be developed for counting the LOC in a program. However, a code counting utility developed for a specific language cannot be used for other languages due to the syntactical and structural differences among languages.

(b) An Intuitive Metric: Line of Code serves as an intuitive metric for measuring the size of software due to the fact that it can be seen and the effect of it can be visualized. Function Point is more of an objective metric which cannot be imagined as being a physical entity, it exists only in the logical space. This way, LOC comes in handy to express the size of software among programmers with low levels of experience.

7.2 Disadvantages

(a) Lack of Accountability: Lines of code measure suffers from some fundamental problems. First and fore most, It is completely inaccurate and unfortunate to have to measure the productivity of a development project with the outcome of one of the phases (coding phase) which usually accounts for only 30% to 35% of the overall effort.

(b) Lack of Cohesion with Functionality: Though experiments have repeatedly confirmed that effort is highly correlated with LOC, functionality is less well correlated with LOC. That is, skilled developers may be able to develop the same functionality with far less code, so one program with less LOC may exhibit more functionality than another similar program. In

particular, LOC is a poor productivity measure of individuals, since a developer can develop only a few lines and still be more productive than a developer creating more lines of code.

(c) Adverse Impact on Estimation: As a consequence of the fact presented under point (a), estimates done based on lines of code can adversely go wrong, in all possibility.

(d) Developer's Experience: Implementation of a specific logic differs based on the level of experience of the developer. Hence, number of lines of code differs from person to person. An experienced developer may implement certain functionality in fewer lines of code than another developer of relatively less experience does, though they use the same language.

(e) Difference in Languages: Consider two applications that provide the same functionality (screens, reports, databases). One of the applications is written in C++ and the other application written a language like COBOL. The number of function points would be exactly the same, but aspects of the application would be different. The lines of code needed to develop the application would certainly be not the same. As a consequence, the amount of effort required to develop the application would be different (hours per function point). Unlike Lines of Code, the number of Function Points will remain constant.

(f) Advent of GUI Tools: With the advent of GUI-based languages/tools such as Visual Basic, much of development work is done by drag-and-drops and a few mouse clicks, where the programmer virtually writes no piece of code, most of the time. It is not possible to account for the code that is automatically generated in this case. This difference invites huge variations in productivity and other metrics with respect to different languages, making the Lines of Code more and more irrelevant in the context of GUI-based languages/tools, which are prominent in the present software development arena.

(g) Far from OO Development: Line of Code makes no meaning in the case of Object-Oriented development where everything is treated in terms of Objects and classes. Since object is a true representation of data and functionality and so is a Function Point, FPA remains more relevant for Object-Oriented software development.

(h) Problems with Multiple Languages: In today's software scenario, never a single language is used for development. Very often, number of languages are employed depending upon the complexity and requirements. Tracking and reporting of productivity and defect rates poses a serious problem in this case since defects cannot be attributed to a particular language

subsequent to integration of the system. Function Point stands out to be the best measure of size in this case.

(i) Lack of Counting Standards: There is no standard definition of what a line of code is. Do comments count? Are data declarations included? What happens if a statement extends over several lines? – These are the questions that often arise. Though organizations like SEI and IEEE have published some guidelines in an attempt to standardize counting, it is difficult to put these into practice especially in the face of newer and newer languages being introduced every year.

8. Recommendations

There are many uses of Function Points beyond estimating schedule, effort and cost as discussed in preceding sections. Many organizations are using function points and software metrics just to report organizational level trends. Many project teams report data to a central metrics group and never see the data again. It is equivalent to reporting data into a black-hole. If project managers begin to understand how Function Points can be used to estimate costs, productivity, quality, test cases, to calculate maintenance costs, and so on, they will be more likely to invest in counting Function Points, making an effective use of FP.

On the other hand, any metrics that we use should be indicators of performance, not exact measures of performance. They should provide enough granularity to show general trends, identify problem areas, and demonstrate progress. Trying to make metrics too perfect causes them to be reported two to three months after they are taken. As a consequence, too much time is being spent on precision and not enough on action. Metrics should be used in such a way that they aid efficient project tracking and monitoring and they should act as good indicators.

9. Conclusion

"Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs." – Bill Gates.

Having seen the vast benefits of Function Points and the inherent shortcomings in Lines of Code metric in the foregoing sections, it should be obvious that organizations should focus more on building expertise on Function Point Analysis and put it to effective use.

Accurately predicting the size of software has always troubled the software industry over the years. Function Points are becoming widely accepted as the standard metric for measuring software size. Now that Function Points have made adequate sizing possible, it can be anticipated that the overall rate of progress in software productivity and software quality will improve. Understanding software size is the key to understanding both productivity and quality. Without a reliable sizing metric, relative changes in productivity (Function Points per Person Month) or relative changes in quality (Defects per Function Point) cannot be calculated. If relative changes in productivity and quality can be calculated and studied over time, then focus can be put upon an organization's strengths and weaknesses. Most importantly, any attempt for improvement can be measured for its effectiveness by putting Function Points to best use.

It is highly recommended that organizations employ Function Points, develop expertise, elicit accurate data, build a good repository of historic projects, and in turn use the data for effective benchmarking and continuous improvement.