Improving Size Estimates with .NET Product Metrics

Aleš Živkovič [1], Marjan Heričko[1], Uroš Goljat[1], Zoltán Porkoláb[2]

[1] *University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova 17, SI-2000 Maribor, Slovenia*
[2] *Eötvös Loránd University, Faculty of Informatics, Pázmány Péter sétány 1/C , H - 1117 Budapest, Hungary*
E-mail: ales.zivkovic@uni-mb.si, marjan.hericko@uni-mb.si, gsd@elte.hu

Abstract
Software projects often fail regardless of the technological changes in recent years. One of the primary reasons is the intuitive assessment of project size, effort, costs and duration. Project size is an independent value where effort, costs and duration are derived values directly related to project size. For size estimates, several methodological approaches are available and in use today. However the Function Point Analysis (FPA) method tends to be predominant in practice as well as in research. In this paper, the use of the FPA method for a project developed in the .NET environment is briefly discussed and linked with source code metrics in order to improve size estimates and get more information related to derived values such as effort, costs and duration.

1 Introduction

One of the key questions in software development is software size estimation. Software size is a primitive measure, through which other important values are calculated (e.g. effort, costs, productivity and duration). These values are particularly interesting for project managers at the beginning of a project. In practice, an intuitive approach is often used, not a methodological one. The results are often unsatisfactory. The efficiency of the intuitive estimates can be indirectly measured via a software project's success factors, especially time and budget overruns.

For systematic software size estimation, different methods are used [10, 15], all of which have their roots in the Function Point Analysis (FPA) method. Albrecht [1, 4] introduced this method in 1979. Since then, it has become the most important method for software size estimation. The method introduced a specific way of representing a software system and distinguished between data functions and transactional functions. The method was intended for all domains, although in practice, its accuracy is different within different domains. A more detailed empirical analysis of the method revealed some weaknesses, which include [2, 3, 11, 12]:

- Correlation between elements of the FPA method.
- The unsuitability of the the Value Adjustment Factor (VAF) and General System Characteristic (GSC).
- The violation of the monotony axiom.

From a practical standpoint, it can be concluded that the FPA method application is more difficult with object-oriented projects. The elements and constructs of the FPA method are not directly applicable to object-oriented concepts also used within the .NET framework. Therefore, a mapping of object-oriented concepts into FPA elements is needed. The mapping is not defined within the FPA method itself and is consequently not uniform. Different authors have proposed different mapping functions [2, 3, 13, 17], mostly in the form of additional rules. Information is gathered from different diagrams (e.g. Use Case diagrams, class diagrams, sequence diagrams) which are considered separately. More detailed research has shown that the weight factors of the standard FPA method have to be calibrated for use in object-oriented projects [2, 3, 17]. Consequently, several FPA-like methods were developed that map object concepts into metrics similar to function points. Sneed proposed Object Points and Minkiewicz developed Predictive Object Points. The results are therefore incomparable to those calculated within the original FPA method. Additional adjustment factors are needed that have to be proven statistically.

This paper is divided into four sections. In the next section, the FPA method and its use in the .NET projects is briefly presented. The correlation between estimated size and source code metrics is discussed in section three. The last section summarizes the findings and discusses the potential direction for future work.

2 Use of the FPA method for .NET projects

The FPA method is declared as technologically independent and can be used on artifacts from the late analysis phase. The use of the method in early stages is also possible [5, 16] with the use of historical data and statistics. In this paper, the topic of early estimates is not discussed. In practice, the use of the FPA method proved to be more difficult with object-oriented projects. The reason for that is the gap between object-oriented concepts and the FPA abstraction of the software system. The FPA method is based on its own concepts describing a software system. The abstraction of the software system is gained by the standard separation in two parts: one part considers the data's influence and another part takes into account the functionality of the system. Data functions (DF) are further divided into internal and external logical files

(ILF and EIF) assigning different weights to each data function type. The transactional functions (TF) describe functionality through three abstract types, namely: external inputs (EI), external outputs (EO) and external inquiries (EQ). To be able to determine the contribution of the FPA element (ILF, EIF, EI, EO or EQ) to the final estimated size value, the complexity is assigned to each element. The complexity is determined by the number of simple data elements named Data Element Type (DET) or structured elements named Record Element Types (RET). To get actual values in Function Points (FP), the tables defined in the method are used. The FPA abstraction concept is easily applied to structured analysis and design artifacts. The mapping of entities, attributes and processes to FPA elements is straightforward. With object-oriented design mapping is not that obvious. Therefore, several researchers [2, 3, 13, 17] proposed additional rules on how to use the FPA method with object-oriented concepts.

The common mapping of object-oriented concepts to the FPA abstraction could be further adapted to the .NET concepts. Table 1 presents the proposed mapping. To avoid double counting, abstract classes are not counted, since in order to instantiates such a class, the concrete class with implemented abstract methods must be present. The same applies for abstract methods. Since the property is implemented as a pair of methods for setting and retrieving attribute values, the concept is mapped to one or more transactional functions of the type external input (EI) and external output (EO).

Table 1: Mapping of the .NET to the FPA Concepts

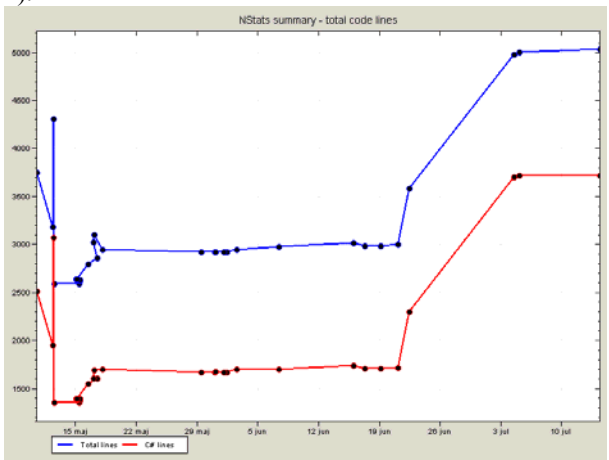
.NET Concept	**FPA Concept**
class	data function - ILF
abstract class	not counted
interface	data function - EIF
method	transactional function (EI, EO, EQ)
abstract method	not counted
attribute	DET (value type) RET (reference type)
property	transactional function (EI or EO or both)
virtual method	not counted
generic (method) (class)	transactional function (EI, EO, EQ) data function - ILF
method parameter (in)	DET (value type) FTR (reference type)
method parameter (out)	DET (value type) FTR (reference type)
return type	DET (value type) FTR (reference type)
value type	DET
reference type	RET (for DF) FTR (for TF)

The concept *generic* could be used as parameterized class or method and therefore has different mapping that depends on the use of the concept. Although .NET has two types of parameters (in and out) the mapping is equal for both. The FPA method has only two types of elements defined for describing parameters - DET and RET/FTR.

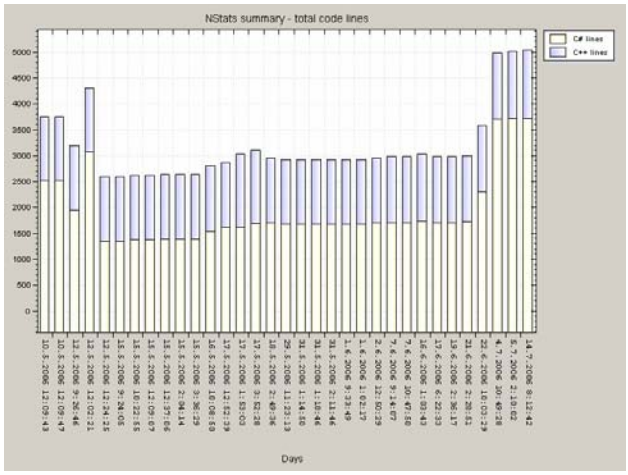
The question of how to map concepts like *generic*, *property*, abstract and virtual *methods*, abstract *classes* and *interfaces* from the .NET framework to the FPA method is still an open topic, and has to be examined in more detail with the empirical evaluation of the projects. The impact of more detailed classification of the .NET elements within the size estimation process has to be tested as well.

Figure 1: NStats GUI Interface

In this initial research, only one project was evaluated. The project called NStats was developed in C# and works as a standalone Windows application with a GUI interface (see Figure 1) although it can be run in the system console as well. The application examines .NET projects, generates statistics and plots graphs (see Figure 2).



(a)



(b)

Figure 2: Graphs showing changes in the total number of code lines - (a) line graph and (b) column graph

The implemented metrics are:

- number of lines of code (LOC),
- number of comment lines,
- number of C# projects in the assembly,
- number of C++ projects in the assembly,
- number of HTML tags in the project and
- number of files for MS Reporting Services.

The project took eleven developer days to complete and produced 2.468 LOC in C#. The statistic is summarized in Table 2.

Table 2: Statistics Summary for the NStats Project

Metrics	**Value**
Number of projects	3
Number of files in assembly	36
Total number of classes	29
LOC in C#	2468
Number of commentary lines	2450
Number of HTML tags	0
Total effort (hours)	88
Duration (days)	11
Number of developers involved	2
Estimated project size (FP)	117

To correctly interpret results more projects are needed and will be evaluated in the future.

3 Function Points and Code Metrics

From the example in Section 2, it is obvious that the interpretation of the collected data could be difficult without the appropriate context. Although one could argue that more data would allow for the correct interpretation of the results, this might not be the case, especially if the data is collected from projects that vary greatly. The properties that may influence the value of the collected data are (in part):

- project team size,
- code quality,
- level of code reusability,
- portion of reusable code included in the new code,
- algorithmic complexity of the solution and others.

For example, if the project has only one developer with outstanding productivity, the collected data is without value if it does not note this information. On the other hand, the low productivity on another project may be due to the high reusability of the code which is of very good quality and easy to maintain. The benefits are visible through the long-time observation of the same code. The aim of our research is to prove a correlation between the software size estimates and source code metrics on the example of the .NET projects.

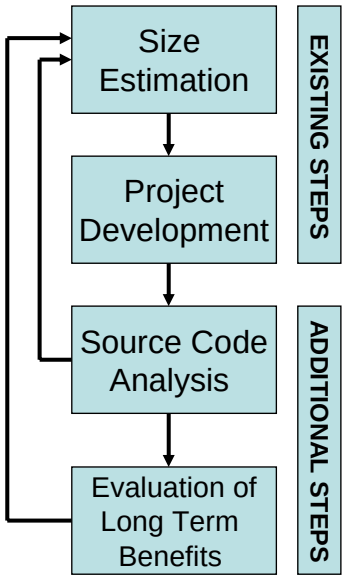


Figure 3: Additional Steps in the Estimation Process

Figure 3 shows the supplemented estimation process that adds two additional steps. The simplified estimation process has only two high-level steps. First the size estimation takes place. It evaluates the artifacts produced during the requirements gathering phase and at the beginning of the analysis phase. The project then proceeds to the development phase. Finally, the comparison of the size estimate with the actual effort is done in order to improve future estimates. The purpose of the two additional steps is to extract additional meta-data about the project from the source code. The sub-set of source code metrics is used in this step. However, to prove the influence of metric values on size estimation, the project has to be tracked on a long-term basis. The benefits for future projects, maintenance costs, bug counts and other metrics are used to prove the

correlation. The outputs from both additional steps represent a back-loop in order to improve the accuracy of size estimates calculated in the first step. The back-loop may improve future estimates, not the estimates for the on-going project.

4 Conclusion

The .NET framework is one of the most frequently used advanced development platforms. In this research, it is used as an example for the fine adjustment of the size estimation process. The original FPA method, declared as technology independent, is easier to use after the OO-to-FPA mapping is defined. A well-defined formal mapping enables the automation of size estimation procedures and has a positive impact on estimation accuracy. However, the research presented in this paper tends to take the adjustment of the FPA method one step further. First, it defines the fine tuned mapping for the .NET framework and then two additional high-level steps are proposed for the estimation process. In the first additional step, a sub-set of source code metric is collected and analyzed. The findings are added to the project repository for future use. In the second additional step, the project is monitored through a complete lifecycle. The potential benefits are compared with previous findings and updated.

In the future, several initial findings have to be examined in more detail and upgraded. With more projects available, the mappings of .NET to FPA concepts could be improved or upgraded to a more generalized form like GASS [18]. The broader set of .NET projects will enable the selection of the most valuable source code metrics and provide statistical evidence of their correlation to software size.

References

[1] Albrecht,A., 1979. Measuring Application Development Productivity, IBM Applications Development Symposium, pp. 83-92.

[2] Antoniol,G., Lokan,C., Caldiera,G., and Fiutem,R., 1999. A Function Point-Like Measure for Object-Oriented Software, Empirical Software Engineering, 4 (1999), pp. 263-287

[3] Antoniol,G., Fiutem,R., and Lokan,C., 2003. Object-oriented function points: An empirical validation, Empirical Software Engineering, 8 (2003), pp. 225-254

[4] IFPUG, 2004. Function Point Counting Practices Manual, Release 4.2, International Function Point Users Group, Princeton Junction, USA, January 2004

[5] ISBSG, 2001. Practical Project Estimation, A toolkit for estimating software development effort and duration. International Software Benchmarking Standards Group

[6] ISO, 1998. ISO/IEC TR 14143-1. Information technology - Software measurement - Functional size measurement, Part 1: Definition of concepts, First edition, ISO/IEC

[7] ISO, 2002a. ISO/IEC TR 14143-2. Information technology - Software measurement - Functional size measurement, Part 2: Conformity evaluation of software size measurement methods to ISO/IEC 14143-1:1998, First edition, ISO/IEC

[8] ISO, 2002. ISO/IEC TR 14143-4. Information technology - Software measurement - Functional size measurement, Part 4: Reference model, First edition, ISO/IEC

[9] ISO, 2003. ISO/IEC TR 14143-3. Information technology - Software measurement - Functional size measurement, Part 3: Verification of functional size measurement methods. First edition, ISO/IEC

[10] Jeffery,D.R., Low,G.C., and Barnes,M., 1993. A Comparison of Function Point Counting Techniques, IEEE Transactions on Software Engineering, 19 (1993), pp. 529-532

[11] Lokan,C., 1999. An empirical study of the correlations between function point elements, Proceedings of METRICS '99: Sixth International Symposium on Software Metrics, pp. 200-206

[12] Lokan,C.J., 2000. An empirical analysis of function point adjustment factors. Information and Software Technology, 9 (2000), pp. 649-659

[13] Uemura,T., Kusumoto,S., and Inoue,K., 2001. Function-point analysis using design specifications based on the Unified Modelling Language. Journal of Software Maintenance and Evolution-Research and Practice, 13 (2001), pp. 223-243

[14] UKSMA, 1998. UKSMA. Mk II Function Point Analysis, Counting Practices Manual. version 1.31. United Kingdom Software Metrics Association (UKSMA)

[15] Živkovič,A., Hericko,M., and Kralj,T., 2003. Empirical assessment of methods for software size estimation. Informatica (Ljubljana), 4 (2003), pp. 425-432

[16] Živkovič,A., Hericko,M., Brumen B., Beloglavec S., Rozman I., 2005a. The Impact of Details in the Class Diagram on Software Size Estimation, Informatica (Lithuania), 16 (2), 2005

[17] Živkovič, A., Rozman, I., Heričko, M., 2005b. Automated Software Size Estimation based on Function Points using UML Models, Information & Software Technology, Volume 47, Issue 13, October 2005, 881 - 890

[18] Heričko, M., Rozman, I., Živkovič, A., A Formal Representation of Functional Size Measurement Methods, The Journal of System and Software, 79 (2006), 1341 - 1358