An Empirical Study on the Estimation of Size and Complexity of Software Applications with Function Points Analysis

Luís M. Alves
Escola de Tecnologia e Gestão
Instituto Politécnico de Bragança
Bragança, Portugal
lalves@ipb.pt

Sérgio Oliveira
Dep. Sistemas de Informação
Escola de Engenharia
Universidade do Minho
Guimarães, Portugal
pg20671@alunos.uminho.pt

Pedro Ribeiro
Centro ALGORITMI
Universidade do Minho
Guimarães, Portugal
pmgar@dsi.uminho.pt

Ricardo J. Machado
Centro ALGORITMI
Universidade do Minho
Guimarães, Portugal
rmac@dsi.uminho.pt

***Abstract*—Empirical studies are important in software engineering to evaluate new tools, techniques, methods and technologies in a structured way before they are introduced in the industrial (real) software process. Perform empirical studies in a real context is very difficult due to various obstacles. An interesting alternative is perform empirical studies in an educational context using students as subjects and share the results with the academia and the industry. This paper describes a case study with two teams that developed a software system (Web application) for a real customer. In this study we used a model based on Function Points Analysis (FPA) to estimate the size and complexity of software system.**

Keywords—empirical studies; software engineering management; software engineering process; software quality; function point, function points analysis.

I. INTRODUCTION

Software engineering is a multi-disciplinary field, crossing many social and technological boundaries. This means that it is not enough to investigate technological aspects, but also, social and cognitive aspects. Any Empirical Study (ES) should take into account both aspects.

Empirical studies in software engineering have had a significant role in the evaluation of tools, techniques, methods and technologies before they are introduced in software industry [1]. This kind of studies can provide valuable results for improving and extending the body of knowledge in the area. For this purpose, Basili [2] provides an organizational schema for collecting experiences on reuse of empirical results, for analyzing them and generalizing the knowledge contained. This schema is known as *Experience Factory*. The *Experience Factory* collects experiences and empirical validations on data related to development processes in various contexts: costs, benefits, risks, and improvement initiatives [3]. This scheme was designed based on many years of the Software Engineering Laboratory (SEL) work. Empirical studies developed within the SEL involved students from different USA (United States of America) universities and industry partners.

This work has been supported by FCT - Fundação para a Ciência e Tecnologia within the Project Scope: PEst-OE/EEI/UI0319/2014.

With our approach we do not intend to create a new SEL. Instead, we intend to create a space (virtual or physical) that allows us to conduct empirical studies in the software engineering area by involving students that are enrolled in our current software engineering courses (both at undergraduate and postgraduate university programmes). We are aware of our limitations, but we believe that we can contribute to the body of knowledge of Software Engineering (SE) and also to contribute to the increasing of the competitiveness of software companies.

The characteristics that influence the success of any organization, in particular, IT (Information and Technology) organizations in the competitive and globalized current market are efficiency, effectiveness, delivery the product on time, within budget and with a level of quality desired by the customer. In this sense, we must emphasize the importance of mechanisms for monitoring, control and evaluating the progress of process, project and product.

Currently, the level of competition among organizations is directly linked to the efficiency of their information systems. The organizations need to adopt more and more new systems. This means that the costs of development and maintenance for organizations are critical parameters in its management. To control costs with the acquisition of software systems is necessary that organizations do the task of size estimation of the systems that intend to adopt, because you cannot manage what you cannot measure [4]. The emergence of the estimation method, namely, *Function Points Analysis* (FPA) has allowed to the IT community a significant increase of software measurement practice. However, the count of *Function Points* (FPs) requires a descriptive documentation, such as specifications of the software functionalities.

With base on knowledge of size and complexity of software applications we can estimate the total amount of resources needed for all developing process. Currently, there are several methods to estimate the resources needed to develop a software system. For this propose, in our research we find *Function Points Analysis* [5], *Use Case Points* (UCP) [6] and LOC (*Lines of Code*) methods.

978-1-4799-4264-0/14 $31.00 © 2014 IEEE
DOI 10.1109/ICCSA.2014.17

In our ES we used graduate and undergraduate students that were randomized grouped in two teams to develop a software system. We applied the original FPA method for estimate the size and complexity of the software system developed by each team.

Our final goal of carrying out empirical studies with students is to understand its validity when compared with the corresponding studies in real industrial settings. We intend to answer the following questions: is it adequate to use students as subjects in empirical studies? What is the better way to involve students as subjects in order to obtain valid results? Can we use the results with student in real context?

In this paper, a description of related work with *Function Points Analysis*, *Use Case Points* and LOC (*Lines of Code*) methods is presented in Section 2. In section 3 we briefly describe the ES we have developed to initially assess the effectiveness of using FPA method in educational context. Finally, in Section 4 we present the conclusions and future work.

II. EFFORT ESTIMATION METHODS

Software size estimation is a crucial element in a project manager's decision-making process, with regard to the project's duration, budget and resources. Estimating the size of a software system is a critical development process activity. Not only does size impact the technical solution but it also impacts the project management solution.

Below we present some concepts related to measurement and metrics to estimate software size. Thus, Fenton and Pfleeger defined measurement as "process by which numbers or symbols are assigned to attributes of real-world entities, in order to describe them according to well-defined rules" [7]. Basili et al. defined measurement as a mechanism to create a corporate memory and to aid in answering a variety of questions associated with the enactment of any software process. These authors point out that to be effective, the measurement must be focused on specific goals and to be applied to all life-cycle products, processes, and resources. Also, the measurement must be interpreted on the basis of characterization and understanding of the organizational context, environment and goals [8].

Pressman, based *IEEE Standard Glossary of Software Engineering Terms* [9] defines a measure as "quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process" and metric as "a quantitative measure of the degree to which a system, component, or process possesses a given attribute". With base in measures, we can obtain indicators that are a metric or a combination of metrics, that provide insight into the software process, a software project, or the product itself [10].

Metrics can be classified into several categories: process, product and resources metrics, objective and subjective metrics and direct and indirect metrics. The process metrics measure the activities performed during all the software development cycle; product metrics measure the artifacts, deliverables and work products that result from process activities; and resources metrics measure the entities required by a process activity [7].

Objective metrics can be quantified and measured using numerical expressions or graphical representations of numerical expressions counted from the source code, design, test data, and other information of the software. Subjective metrics are measured based on personal or group estimates, usually obtained through concepts such as high, medium, or low [11].

Fenton and Pfleeger argue that subjective metrics depend on the person who is measuring, its judgment and the degree of inaccuracy, which hinders consensus between attributes involving processes, products or quality [7].

The direct metrics are those that do not directly depend on the measure of another attribute, but the quantification of a factor observed in the product. The indirect metrics involve the measures of one or more attributes related to each other [7].

Within the category of direct metrics, McGarry et al. highlight some approaches of software project estimates metrics as: i) parametric models, ii) analogy iii) expert judgment iv) activities based on models and v) relationships estimates. The most common approaches are the parametric models, analogy and expert judgment [12].

Metrics are measurement methodologies whose main objective is to estimate the size of software system and assist, as an indicator, the project management of software system development. The estimated size is one of the most commonly used metric for software size, since has direct impact on development effort and project management.

Currently, there are several metrics of size estimation and it is difficult to select the most appropriate for the size of a software project in an organization. The main metrics were developed based on the software functions such as: *Function Points Analysis* [5], *Bang* [13], *Mark II* [14], *Use Case Points* [6] and *COSMIC Full Function Points* [15].

A. LOC (Lines of Code)

The first metrics of software size estimation emerged in mid-1960's, although the first dedicated book on software metrics was not published until 1976 [16]. These metrics were based on the physical size of *Lines of Code* (LOC) and were used as the basis for "measuring programming productivity and effort" [17]. This metric considers the software from the perspective of the internal structure and is applied in the final stages of the software project [18].

The LOC metric, the oldest software metric, appeared at the time that the software professional developers used procedural programming languages such as Fortran and Assembly. These professionals began software estimates using LOC method because is intuitive and easy to apply by different professionals. Ross highlights two advantages of using LOC method: 1) the possibility to estimate automatically, and 2) the ease of using historical data because most of the existing data about estimation were

measured by LOC method. The disadvantages are related to ambiguity, because the metric becomes ambiguous when dealing with non-textual abstractions and the lack of significance of the measure to the end user (customer) [19]. Besides these disadvantages Fenton and Pfleeger note that a count in LOC depends on the degree of code reusing and the programming language and can be five times higher than another estimate, due to differences in techniques of measurement of blank lines, comment lines, data declaration and statements [7]. The authors also emphasize that the LOC method penalizes small and well-designed programs, it is not adequate to non-procedural programming languages and is difficult to obtain in the early planning stages of development of a software system [7]. Another weakness of this metric is the absence of a standard that clearly defines the rules to be observed during the count.

LOC method was a metric widely used until mid-1970. From there emerged various programming languages and consequently the need for other ways to estimate the size of software.

B. Use Case Points

The first description of the method was published by Gustav Karner [6] with the aim of creating a model that would allow estimating the resources required to develop a software system under Objectory AB (later acquired by Rational Software). Its influences came from the classic function point method.

Industry use of UCP method of estimation is very rare. Agarwal et al cited by Damodaran and Washington [20] presents an example of use of this method for estimating an Internet project held in Infosys (Bangalore, India). Other experiences of use of UCP in industry were carried out by Anda et al [21], by Ribu [22] and by some companies as the *Rational*, SUN and IBM [23].

Damodaran and Washington present some reasons for the low use of UCP in industry such as: (i) relative newness of the method; (ii) has not yet been incorporated in the popular project estimation tools; and (iii) the use case model, despite being a standard method for the description of requirements, still does not have good historical productivity figures [20].

Despite the UCP still little used in industry, many authors have tried this metric with and without the use of technical adjustment factors, they calculated the effort estimate, they proposed a standard form for describing use cases and they defined rules for assigning values to environmental factors [24].

Anda et al. [21] applied the Karner method [6] in three projects of a software development company in Norway, Sweden and Finland to compare the estimates made by the UCP with the estimates made by senior members of the development projects. The results showed that the estimation performed through UCP method was close to the real estimation performed by experienced developers. These authors observed that some aspects of the structure of the use cases models had impact on estimates such as: i) the use of generalization between actors, ii) the use of included and extending use cases, iii) the level of detail in the use case descriptions; iv) difficulties in assigning values to the technical and environmental factors and v) the choice of the productivity rate per UCP [21]. Anda et al. defined a standard form to facilitate the description of use cases.

The results of studies by Anda et al. [21] support existing claims, in which the use case model has a strong impact on the estimate, it can be successfully used to estimate software development effort and the Karner method may support expert knowledge in the estimation process. Although Karner not recommend the count of the use cases extended and included, Anda et al. [21] argues that they should be included in the count to avoid below estimations of the reality, especially if the functions described in these use cases are essential and implemented.

In another work, Anda conducted a study as part of three courses on use case modeling in a large international IT company. The course schedule was the same in all the three courses. The participants were experienced developers, business analysts and project managers totaling 37 professionals. The author was one of the instructors on the last two courses. All the participants had extensive experience of requirements engineering, and they had experience from estimating their own work [25]. The aim of the study was to compare the estimate obtained using the UCP method and the estimate performed by experts (experienced estimators). In this study UCP method gave an estimate that was closer to the actual effort spent on implementing the system than most estimates made by 37 experienced professional software developers divided into 11 groups [25].

Ribu also used the Karner method to estimate the size of software projects performed by students and professionals from industry [22]. Ribu has tested the UCP method with and without the use of technical adjustment factors. The author proposed a standard form for describing the use cases and defined rules for assigning values to environmental factors. The main conclusions of the Ribu study were: i) the method of UCP had low variation between the estimated value and real value ii) suggested to dismiss the technical adjustment factors and maintain environmental factors, iii) count the use cases included and extended and iv) use a standard form for describing use cases.

C. Function Points

The first description of the method was published in 1979 by Allan Albrecht with the aim of creating a model that would allow measure productivity over all phases of a project independently on the programming language and technologies used.

The FPA is one of the first metrics to measure the size of software with some precision. It is the most used in the industry and became an international standard in 2002 through the ISO/IEC 20926 [26]. Currently, the mapping of the FPA method to estimate object oriented software projects has been widely discussed in the literature.

The use of FP metric is not a trivial process, requires some practice to apply all the rules presented by IFPUG (International Function Point Users Group). However, FPA helps developers and users quantify the size and complexity of software application functions in a way that is useful to software users [27].

Some researchers have proposed the mapping of the use case driven *Object Oriented Software Engineering* (OOSE) method by Jacobson et al. into the abstract FPA model. The mapping proposed by the authors has been formulated as a small set of concise rules that support the actual measurement process [28]. Other authors based their approach on a class diagram including messages sent between classes. These authors considered each class as an internal logical file ant treated messages sent outside the system boundary as transactions [29]. Uemura et al proposed detailed FPA measurement rules for the design specifications based on the UML (Unified Modeling Language) and develop the function point measurement tool, whose input products are design specifications on Rational Rose. They used the sequence diagrams and class diagrams from UML notation [30].

Some authors have gone beyond mapping the FPA to the context of OO (Object Oriented) proposing new methods, such as: *Fast Count* [31], *Object Oriented Function Point* (OOFP) [32], *Object Oriented Design* and *Function Points* (OODFP) [33].

The *Fast Count* method, a variant of the FPA is used since 1993 by the IRS (International Revenue Service), a USA government agency responsible for tax collection and tax law enforcement. This method can determine the software size about four times faster than industry averages can estimate software size and resource requirements for new development projects. In this method the FPs count is based on the central data model and *Staffing Estimator*, which can accurately size small software work orders and estimate corresponding resources needed without examining the software or meeting with the project team [31].

The *Object Oriented Function Point* (OOFP) created by Caldiera et al, maps the FPA concepts to OO software according of Object Modeling Techniques (OMT) notation. These authors define the counting procedure of the inheritance, aggregation and polymorphism based on the object model developed in the design phase and propose to estimate the software size at different points in the software development as new artifacts are available. The OOFP was applied in eight sub-systems, developed in an industrial environment producing software for telecommunications. These sub-systems were also counted in LOC so that authors could apply regression techniques and find the association between LOC and OOFP methods [32].

The *Object Oriented Design Function Points* (OODFP) was adapted by Ram and Raju (2000) from the counting procedures defined by IFPUG. This method estimates the size of OO software in the design phase, based on the functions of the software and the complexity of classes. The complexity of the class can be *low*, *average* or *high* according to a numeric value defined by the authors based on observations of different projects. For these authors, "a logical file is a collection of data elements which are visible to all methods of a class" and "transactional functions are the methods in a class" [33].

Some studies on the application of FPA method have shown a decrease in variation of function point count between different counters trained and certificates. Furey cites a study developed in 1994 by the *Quality Assurance Institute* and IFPUG that found a counting variance between trained counters about 11 percent [27]. However, Kichenham argues that function point counting involves judgment on the part of the counter. The author refers a study performed by Chris Kemerer that reports a 12-percent difference for the same product by different people in the same organization. The author also refers a study by Graham Low and Ross Jeffery that report "worse figures": a 30-percent variance within an organization, which rose to more than 30 percent across organizations [34].

In the next section a detailed description of the case study and its main results are presented.

III. CASE STUDY

The ES was developed to determine the productivity of two software development teams using the original procedure for function point counting. The teams were constituted by second year students of the course 8604N5 Software System Development (SSD) from the undergraduate degree in Information Systems and Technology in University of Minho (the first University to offer in Portugal DEng, MSc and PhD degrees in Computing). The two teams were composed of 14 students each one. Each team receives a sequential identification letter (Team A and Team B) and the description of the customer problem. The teams developed a software project of medium complexity, using UML notation encompassed in an iterative and incremental software development process, in this case, the Rational Unified Process (RUP). The teams followed the guidelines established by the RUP reduced model [35, 36], executing the phases of inception, elaboration and construction according to the best practices suggested by CMMI-DEV v1.2 ML2. The project lasted 3 months. This software project was to develop a Web solution using object-oriented technologies and relational databases, to support the information system of one local customer that provided all the information about the organization and interacted directly with the teams. Specifically, the technologies used were MySQL, PHP and Java. Fig. 1 shows the function point counting procedure.

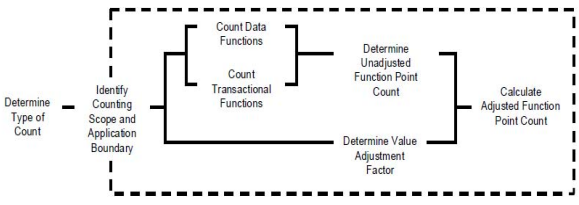

Fig. 1. High-level procedure for function point counting [37]

The following are the main decisions made at each step of the FP calculation for each team. Thus,

i. **Determine Type of Count** - the type of count adapted to calculate FPs of the applications developed by Teams A and B was "application function point count" because the development software process is already completed.

ii. **Identify the Counting Scope and Application Boundary** - the applications developed by teams does not have any interaction with other systems.

iii. **Count Data Functions** - In this step were identified and counted the *Internal Logical Files* (ILFs) and *External Interfaces Files* (EIFs) of the applications developed by Teams A and B. When the FPA method was created the applications did not use relational database, so we used the Entity Relationship Diagrams (ERD) to identify the dependencies of the tables and identify the ILFs and EIFs. From the data model developed by Team A were identified 27 ILFs, but as the application developed did not use all of these files, we recorded only the used files and that number decreased to 17. The application developed by Team A has no EIF because there is no communication with external databases. From the data model developed by Team B were identified firstly 16 ILFs, but for the same reason that Team A this number decreased to 12. For the same reason mentioned for the application of Team A, the application of Team B also has no EIFs. After being identified all ILFs and EIFs was necessary to identify the complexity of the applications, for this purpose we used as support the ISO/IEC 20926. Table I shows the ILF and EIF complexity.

TABLE I. ILF AND EIF COMPLEXITY

RETs	Data Elements		
	1-19	20-50	>=51
1	Low	Low	Average
2-5	Low	Average	High
>=6	Average	High	High

The "Data Elements" correspond to tables and Record Element Type (RET) correspond to attributes

iv. **Count Transactional Functions** - In this step was counted the functions identified in the applications developed by Team A and B. After an analysis of the software developed by Team A were identified 29 EIs *External Inputs* (EIs), in these functions were found functionalities for insertion, updating and delete data. As the *External Queries* (EQs) were identified 12 queries. The application of Team A has no *External Outputs* (EOs) since there is no use of mathematical calculations (for example the sum of all medicines from a certain laboratory). From the analysis of the software developed by Team B were identified 19 EIs, as for the Team A in these functions were found functionalities for insertion, updating and delete data. As the EQs were identified 17 queries. Unlike Team A, the application of the Team B has EOs, two in total, for example, its application has implemented a counting function of all medicines available in a lot of drugs. This is a kind of function that requires a mathematical calculation making it an EO. After being identified all EIs, EQs and EOs was necessary to identify the complexity of the applications, for this purpose we used as support the ISO/IEC 20926. Table II shows the various intervals and the respective complexity associated to the EIs. On the other hand, Table III shows the number of intervals and the complexity associated to the respective EQs and EOs.

TABLE II. EI COMPLEXITY

FTRs	Data Elements		
	1-4	5-15	>=16
0-1	Low	Low	Average
2	Low	Average	High
>=3	Average	High	High

TABLE III. EQ AND EO COMPLEXITY

FTRs	Data Elements		
	1-5	6-19	>=20
0-1	Low	Low	Average
2-3	Low	Average	High
>=4	Average	High	High

Where FTRs (File Types Referenced) are the combined number ILFs referenced or updated and EIFs referenced.

v. **Determine Unadjusted Function Point Count** - In this step were calculated the unadjusted function points. This step depends on the previous steps because the values identified above will be used at this step. Table IV shows the count of all ILFs, EIFs, EOs, EQs and EIs and their respective complexity. The last row of Table IV shows the sum of all UFPs.

TABLE IV. TOTAL OF UNADJUSTED FUNCTION POINTS

		Team A			**Team B**		
		Nr.	Weight	UFPs	Nr.	Weight	UFPs
ILF	Low	15	7	105	7	7	49
	Average	2	10	20	5	10	50
	High	0	15	0	0	15	0
EIF	Low	0	5	0	0	5	0
	Average	0	7	0	0	7	0
	High	0	10	0	0	10	0
EI	Low	23	3	69	17	3	51
	Average	6	4	24	2	4	8
	High	0	6	0	0	6	0
EQ	Low	9	3	27	13	3	39
	Average	3	4	12	3	4	12
	High	0	6	0	1	6	6
EO	Low	0	4	0	2	4	8
	Average	0	5	0	0	5	0
	High	0	7	0	0	7	0
	Total of UFPs			**257**			**223**

vi. **Determine Value Adjustment Factor** - After completing the previous step we must calculate *Value Adjustment Factor* (VAT).The VAT is based on 14 *General System Characteristics* (GSCs) that rate the general functionality of the application being counted. Associated with each of these characteristics should be attributed the Degree of Influence (DI). Table V shows the values of the DI attributed and a brief justification for the allocation of these values.

TABLE V. GSC AND DEGREE OF INFLUENCE

	Team A		Team B	
Data Communications	0	The system was only used in stand-alone mode	0	The system was only used in stand-alone mode
Distributed Data Processing	0	Not applied	0	Not applied
Performance	0	Not tested	0	Not tested
Heavily Used Configuration	0	Without configurations	1	It was necessary to make Apache configurations
Transaction Rate	0	The system was not implemented	0	The system was not implemented
Online Data Entry	5	The system is a web site	5	The system is a web site
End-User Efficiency	2	Allows interaction with the user through an interface	1	Allows interaction with the user through an interface
Online Update	0	The system does not perform	0	The system does not perform
Complex Processing	0	Does not perform mathematical calculations	1	Perform simple mathematical calculations
Reusability	1	Reusability of some Java and HTML code	3	Reusability of some Java and HTML code
Installation Ease	0	Not tested	0	Not tested
Operational Ease	0	Not tested	0	Not tested
Multiple Sites	0	The system was not implemented on the client	0	The system was not implemented on the client
Facilitate Change	0	The system will not be changed	0	The system will not be changed
Total DI	8		11	
VAT	0.73		0.76	

Substituting the value of DI in (1) we get the VAT. This value will be used in the next step.

$$VAT=0.65+0.01*DI \quad (1)$$

vii. **Calculate Adjusted Function Point Count** - based on VAT calculated in the previous step we calculate the adjusted function points, usually just called Function Points (FPs), substituting that value in (2). Table VI shows the calculation of the values of the FPs for Team A and Team B.

$$FPs=UFPs*VAT \quad (2)$$

TABLE VI. ADJUSTED FUNCTION POINTS

Team A			Team B		
UFPs	VAT	FPs	UFPs	VAT	FPs
257	0,73	187,61	223	0,76	169,48

After knowing the FPs of the applications of the teams A and B, it is possible to calculate the productivity of each one. To calculate the productivity value was necessary to identify the hours of work needed to develop the applications studied. These data were obtained in the Master's thesis of Mandjam [38]. The Team A developed the software application in about 3942 hours and Team B in 2435 hours. These number of hours include all work performed by all students in different roles. The productivity calculation is done using the following formula:

$$Productivity = Effort[h] / Size[FPs] \quad (3)$$

Applying formula (3) we obtain a productivity of 21.01 Hours of Work/Function Points for Team A and 14.37 Hours of Work/Function Points for Team B. Based on these results we conclude that Team B was more productive than the Team A. The Team B required less work hours to perform one FP. The productivity of the Team B was higher by 40.6%.

However the productivity of the Team B it is higher than productivity of the team A, the grade is lower. This discrepancy is justified by the fact that final grade calculation results from a plurality of weighting factors from three sources of information [39], such as the teachers traditional evaluation, client evaluation and the quality of the final product (business functionality).

Based on FPs values calculated from the software applications of the teams and the grade assigned to one team, we can estimate the grade of the other teams. Table VII shows that based on the classification assigned to the Team A (15.8) and the FPs values previously calculated for both teams it was possible to calculate the classification of the Team B (14.3).

TABLE VII. CLASSIFICATION FROM FPS

	Team A	Team B
	Grade in scale [0;100]	Grade in scale [0;20]
FPs	187.61	169.48
Classification	15.8	(169.48*15.8)/187.61= 14.3

IV. CONCLUSIONS

Empirical studies in software engineering have had a significant role in the evaluation of tools, techniques, methods and technologies before they introduced in industrial software.

A large number of empirical studies reported in the literature used students as subjects instead professionals. This great participation is justified because the students are accessible, available, willing to participate, inexpensive and we can combine the learning objectives of the courses with the research objectives of the studies.

The two teams developed separately a software system (Web application) for a real customer. From the effort estimation methods, we used the FPA to estimate the productivity of that two teams.

The process of identifying the various components for the calculation of FPs was a manual process. This count was obtained from an intensive study of the code of the applications developed by two teams. The FPs counting process is not trivial, but its adoption is of great importance when we want to measure the software development. The FPs can support the project management activities, since we can only manage what we can measure. The productivity in software development is a good indicator for management.

In [40] we find that the average effort is 20 work hours per Function Point. In the case study, Team A presents a very close value (21.01), whereas Team B present a value most discrepant (14.37). This does not mean that they are super team, but means that further research is needed to find the reason of this discrepancy. However, we believe that empirical studies involving students on these subjects are important for the scientific community and the industry.

In our case study, the productivity of the Team B is higher than the productivity of the Team A. However, the grade is lower because the final grade calculation results from a plurality of weighting factors from three different sources. Based on the FPs values and the classification assigned to one team we can estimate the grades of the other teams. In fact, the teachers assigned a grade of 15.8 to Team A and a grade of 14 to Team B. In the assessment schema presented in this paper after assigning a reference grade we can calculate an estimate of the remaining grades using the FPs values. This method can be an asset, especially when many teams are involved and if it is possible to automate the entire process.

Considering also that productivity is directly related to the effort made and with the functional size per FP, we suggest for future editions of the SSD course that all teams use a development tool, for example *Teamwork Project Manager* [41], in order to accurately determine the effective involved effort. We intent to assess the influence of this tool in the teams performance. As future work, the determination of the productivity of each element according the role of the RUP that he/she assumes is also an interesting experiment.

REFERENCES

[1] Host, M. "Introducing empirical software engineering methods in education." In *CSEE&T 2002*. 25-27 February 2002. Proceedings of the 15th Conference on Software Engineering Education and Training. Covington, Kentucky, USA, pp. 170-179.

[2] Basili, V., Caldiera, G., McGarry, F., Pajerski, R., Page, G., and Waligora, S. "The Software Engineering Laboratory-an Operational Software Experience Factory." In *14th ICSE*. 11-15 May 1992. Proceedings of the 14th International Conference on Software Engineering. Melbourne, Australia, pp. 370-381.

[3] Visaggio, G., "Empirical Experimentation in Software Engineering." In Lucia, Ferrucci, Tortora, and Tucci (ed.). 2008. *Emerging Methods, Technologies and Process Management in Software Engineering*. Hoboken, New Jersey: John Wiley & Sons, Inc., p. 227.

[4] Humphrey, W. S. 1995. *A Discipline for Software Engineering*: Addison Wesley.

[5] Albrecht, A. J. "Measuring application development productivity." In *IBM Application Development Symposium*. 15-17 October 1979. Monterey, California, USA, pp. 83-92.

[6] Karner, G. September 1993. "Use Case Points: Resource Estimation for Objectory Projects." *Objective Systems SF AB*.

[7] Fenton, N. E. and Pfleeger, S. L. 1997. *Software Metrics: A Rigorous and Practical Approach*. Second Edition ed. Boston, USA: PWS Publishing Company.

[8] Basili, V., Caldiera, G., and Rombach, H. D., "Goal Question Metric (GQM) Approach." In (ed.). 2002. *Encyclopedia of Software Engineering*. John Wiley & Sons, Inc., pp. 527–532.

[9] IEEE. 1990. *IEEE Standard Glossary of Software Engineering Terminology*. IEEE Std 610.12-1990. New York, USA,

[10] Pressman, R. S. 2005. *Software Engineering: A practitioner's approach*. Sixth Edition ed. New York, USA: McGraw-Hill.

[11] Moller, K. H., Paulish, D. J., and Moauller, K. H. 1992. *A Practitioner's Guide to Improved Product Development*: Cengage Learning EMEA.

[12] McGary, J., Card, D., Jones, C., Layman, B., Clark, W., Dean, J., and Hall, F. 2001. *Practical Software Measurement: Objective Information for Decision Makers*. Boston: Addison-Wesley.

[13] DeMarco, T. April 1984. "An algorithm for sizing software products." *SIGMETRICS Perf. Ev. Rev.* Vol. 12, pp. 13-22.

[14] Symons, C. R. 1988. "Function point analysis: difficulties and improvements." *Software Engineering, IEEE Transactions on.* Vol. 14, pp. 2-11.

[15] COSMIC-FP. 2003. *COSMIC-FFP Measurement Manual, version 2.2*. The Common Software Measurement Int. Cons.

[16] Gilb, T. 1977. *Software Metrics*. Cambridge, Massachusetts, USA: Winthrop Publishers.

[17] Fenton, N. E. and Neil, M. "Software metrics: roadmap." In *CFSE 2000*. 4-11 June 2000. Proceedings of the Conference on The Future of Software Engineering. Limerick, Ireland, pp. 357-370.

[18] Misic, V. B. and Tesic, D. "Downsizing the estimation of software quality: a small object-oriented case study." In *TOOLS 27*. 22-25 September 1998. Proceedings of the Technology of Object-Oriented Languages. Beijing, China, pp. 308-317.

[19] Ross, M. "Size Does Matter: Continuous Size Estimating and Tracking." http://www.qsm.com. Accessed:2014-02-01.

[20] Damodaran, M. and Washington, A. "Estimation Using Use Case Points." http://bfpug.com.br/Artigos/UCP/Damodaran-Estimation_Using_Use_Case_Points.pdf. Accessed:2014-02-01.

[21] Anda, B., Dreiem, H., Sjoberg, D. I. K., and Jorgensen, M. "Estimating Software Development Effort Based on Use Cases-Experiences from Industry." In *UML' 2001*. 1-5 October 2001. Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools. Toronto, Canada, pp. 487-502.

[22] Ribu, K., "Estimating Object-Oriented Software Projects with Use Cases," MSc Master of Science Thesis, Department of Informatics, University of Oslo, 2001.

[23] Probasco, L. (2002, Dear Dr. Use Case: What About Function Points and Use Cases? *DeveloperWorks*. Available: http://www.ibm.com/developerworks/rational/library/2870.html

[24] Schneider, G. and Winters, J. P. 2001. *Applying Use Cases: A Practical Guide*. 2nd Edition ed. New Yoork, USA: Addison Wesley.

[25] Anda, B. "Comparing Effort Estimates Based on Use Case Points with Expert Estimates." In *Empirical Assessment in Software Engineering (EASE 2002)*. April 8-10 2002. Keele, UK.

[26] Dekkers, C. A. 2003. "Measuring the “logical” or “functional” size of software projects and software application." *ISO Bulletin.* pp. 11-13.

[27] Furey, S. March 1997. "Why we should use function points." *IEEE on Software.* Vol. 14, pp. 28-30.

[28] Fetcke, T., Abran, A., and Tho-Hau, N. "Mapping the OO-Jacobson approach into function point analysis." In *TOOLS 23' 97*. 28 Jul-1 Aug 1998. pp. 192-202.

[29] Whitmire, S. A., "Applying function points to object-oriented software models." In (ed.). 1993. *Software engineering productivity handbook*. McGraw-Hill, pp. 229-244.

[30] Uemura, T., Kusumoto, S., and Inoue, K. "Function point measurement tool for UML design specification." In *ISMS* 1999. Proceedings in the Sixth International Software Metrics Symposium. pp. 62-69.

[31] Tichenor, C. B. "The Internal Revenue Service function point analysis program: a brief." In *COMPSAC '97*. 11-15 Aug 1997. Proceedings of the Twenty-First Annual International Computer Software and Applications Conference. pp. 591-592.

[32] Caldiera, G., Antoniol, G., Fiutem, R., and Lokan, C. "Definition and experimental evaluation of function points for object-oriented systems." In *5th ISMS.* 20-21 Nov 1998. Proceedings of the Fifth International Software Metrics Symposium. pp. 167-178.

[33] Janaki Ram, D. and Raju, S. V. G. K. "Object oriented design function points." In *First Asia-Pacific Conference on Quality Software* 2000. Proceedings of the First Asia-Pacific Conference on Quality Software. pp. 121-126.

[34] Kitchenham, B. 1997. "Counterpoint: The Problem with Function Points." *IEEE Software.* Vol. 14, pp. 29-31.

[35] Borges, P., Monteiro, P., and Machado, R. J. "Tailoring RUP to Small Software Development Teams." In *SEAA 2011*2011. Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications. Oulu, Finland, pp. 306-309.

[36] Borges, P., Monteiro, P., and Machado, R. J. "Mapping RUP roles to small software development teams." In *4th International Conference on Software Quality Days (SWQD 2012)*. 17-19 January 2012. Vienna, pp. 59-70.

[37] IFPUG, "Function Point Counting Practices Manual: Release 4.3.1," ed. USA: IFPUG, 2010.

[38] Mandjam, F., "Avaliação do impacto das práticas do CMMI no desempenho de equipas de desenvolvimento de software no ensino," MSc, DSI, Escola de Engenharia, Universidade do Minho, Portugal, 2011.

[39] Clark, N. "Evaluating Student Teams Developing Unique Industry Projects." In *7th Australasian Computing Education Conference (ACE2005)*. January/February 2005. Newcastle, Australia, pp. 21-30.

[40] Jones, C. 2007. *Estimating Software Costs: Bringing Realism to Estimating*. 2nd ed. New York: McGraw-Hill

[41] Teamwork Project Manager. http://www.teamworkpm.net/. Accessed:11 May 2012.