Using Productivity Measure and Function Points to Improve the Software Development Process

Eduardo Alves de Oliveira and Ricardo Choren Noya

Computer Engineering Section, Military Engineering Institute, Rio de Janeiro, Brazil

Abstract *- Usually, cost and time estimations are done at the beginning of a software project for budget planning purposes. Such estimations are used at the end of the project to verify if the initial planning was followed or if there were any deviations. In this sense, these estimations can only be used as an input to improve the process for other projects. This paper presents an iterative method, which uses productivity and function points metrics, to identify possible deviations in the amount of time and effort needed to carry out the process tasks, thus continuously updating the estimations in order to cope with the current project needs. It is presented a real case study of how this process can be applied.*

Keywords: *Function Point Analysis, Indicator of Productivity, Software Development Process, Project Management.*

1 Introduction

Software development companies are getting more and more competitive. To understand how competitive a company is, it must measure the productivity and quality in their Software Development Processes (SDP) [9]. Knowing the productivity in the SDP, allows the company to improve the prediction of several projects parameters such as effort, time and cost. Both users and project managers want to know before a project starts its estimated cost and time to enhance performance with the best accuracy possible [14].

Currently, it is usual to calculate a productivity estimate at the initial planning phase of a project and then to verify the actual productivity yield at the end of the project. [11] The use of a measure at these two moments is extremely important because the estimates are based on historical productivity. However, measuring the productivity only at these two moments in a project is rather insufficient and may cause some difficulties, e.g., knowing throughout the development cycle if the time and cost estimates will be met; monitoring the productivity of medium and large projects; detecting the factors that impact the productivity of a SDP; providing ongoing adjustments to the SDP, and; controlling whether the scope of the project is being met or not.

There already are some techniques for monitoring the productivity of a SDP [11]. Nevertheless such techniques do not assess productivity through a functional measurement. This makes it difficult for managers to compare the productivity of the development of a given functionality to the productivity of other functionalities and to the estimated productivity of project as a whole.

A functional measurement standardizes the estimation of the functional size of any project [8]. Thus it can be used as the unit to be used to measure productivity. Moreover, by using function units managers can assess the project productivity throughout the project and not only at its end.

Changes in the scope of project requirements are a good example of how the use of a functional measure can give further information to managers. Such changes can present a growth rate of 2% per month from the time the project moves from specification to codification [4]. If some functionality had its scope changed it is likely to have its functional size changed thus impacting on productivity. If managers only measure productivity at the end of the project they will probably find the reason why it presented a downside in productivity: changes in the scope. However they missed the opportunity to respond to such changes in order to keep or even enhance productivity during the project execution. The functional measurement could show the productivity rate of function development required for managers to cope with the difficulty to meet the estimated productivity.

This paper presents a method for productivity monitoring all along an iterative SDP execution. Each iteration should have its size measured using a functional measurement of the project use cases. The manager will give a percentage of size of the iteration to each SDP phase. This will allow for effort and productivity division and monitoring in every process iteration. This work uses Function Point Analysis (FPA) [8] as functional measurement.

This article is structured as follows. Section 2 presents how project planning should be done using a productivity indicator. Section 3 describes the method proposed in this paper, i.e. the SDP productivity monitoring. Section 4 illustrates a simple example and, finally, section 5 concludes this paper.

2 PLANNING PROJECTS USING PRODUCTIVITY

The productivity indicator is an important information for planning a project, since it improves the performance in the production of software [12]. Productivity is measured to monitor production, reduce costs and improve the quality of the delivered product [7].

It is considered a complex project measure, which relies on over a hundred known factors [3]. Productivity is a ratio of production output to what is required to produce it. The measure of productivity is defined as a total output per one unit of a total input. [6]. A production output unit in software can be represented by lines of code, components, artifacts or function points. Inputs can be effort (time) or financial (this paper consider inputs as effort measured in hours).

Figure 1 shows a simplified diagram of productivity [5]. It shows how resources are consumed by a particular process or sub process for the generation of a particular software product.

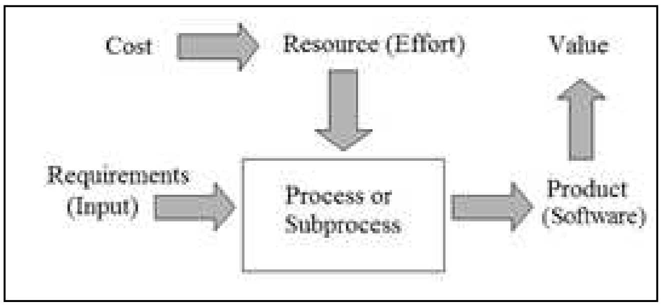


Figure 1: Simplified Model for Productivity [5]

The productivity indicator is calculated using a simple mathematical equation (1).

$$\textit{Productivity} = \textit{Resource} / \textit{Product} \qquad (1)$$

This paper uses the measure of hours of effort (H). The product is represented by the number of function points (FP) produced. Thus productivity is calculated as:

$$\textit{Productivity} = H / FP \qquad (2)$$

There are other ways to measure the size of a project, such as lines of code (LOC) [1, 10] and use case points. The function point (FP) metric was chosen because it is currently the most used measure for functional measurement software in the market. Besides it is independent of the technology of the format of the unit. This technique has emerged as a result of studies at IBM in the 70s [2].

FP considers the functions that store data and the transactions that manipulate such data. The FPA is described in a manual that describes how to calculate the functional size of a software project or improvement of software [8]. The technique does not define, among other things, how to treat indicator of productivity or costs (pricing).

This paper uses FP as a basic measure in the calculation of productivity. FPA can be used to parameterize the functional size of software systems and projects regardless of the technology that will be used to build it [8]. It lets all functional requirements, recognized and specified by the user, to be sized as a number of function points. Thus it is possible for the project manager measure all user functional requirements in a standardized and objective way.

3 A METHOD FOR MONITORING PRODUCTIVITY

This paper proposed a method to monitor the productivity of a project in every iteration during its life cycle. The idea is to allow for adjustments between iterations so that the project does not suffer from delays, increased costs or loss of product quality. It is important to mention that adjustment actions made by managers will impact the project SDP. For these impacts to enhance productivity, it is essential for the manager to know which activities, sub processes or phases are presenting poor (or downslope) productivity.

The method is presented as a set of steps. Each step indicates an action that should be performed side by side with the SDP activities. The main purpose is to allow the manager to compare the actual current productivity with the previously estimated project productivity, done at early project planning phases.

Step1: Dividing the Development Cycle by Phase and Iteration

The development cycle corresponds to the total (i) effort consumed, and (ii) software size (FP) produced in a project. The project should divide the development in iterations (or sprints for agile methods). Each iteration should correspond to a sub cycle of the SDP. It is important to mention that the management does not change the phases (add or drop) in an iteration in order to increase or decrease the effort spent.

Each phase in an iteration is responsible for a share of the total effort estimated for the iteration. The project manager should establish such share to distribute the effort that will be employed in each phase. Such distribution should be done by using historical data or by experience. It is important that this distribution be realistic.

Step 2: Estimating the size of an iteration in Function Points

After the preparation of the iteration, the project manager will have the requirements approved by the client, and these are described in a specification. The method proposed here was used in projects that specified its requirements using use cases. The method to estimate the FP count by use case is as follows:

1) Finding the Elementary Processes: the manager should find the elementary processes in the use case flows. An elementary process is the smallest unit of meaningful activity for the user to specify the requirement [8]. For each elementary process found in the use case, there should be a corresponding transactional function. After finding the transactional functions, the manager can identify the functions that manipulate data.

2) Finding the Data Functions: during the analysis of transactional functions it is possible to identify the data that is manipulated by these functions. The presence of a logical data model is important for a more precise identification of the data, but this model is not always present at the moment the project. Each data function has a complexity that corresponds to an amount of FPs. However a data function can be used by transactional functions from different use cases. In this scenario, the manager can follow two approaches:

2.1) select an owner Use Case: an owner use case is the use case that is the most important (from the client prioritizing point of view) or that uses the data function more. Then the data function should contribute to the FP size of the owner use case.

2.2) divide the contribution of Function Data: each use cases that manipulate the data function should get a slice of its size in FP. This slice is decided by the manager.

3) Finding the estimated FP size of the Use Case: the sum of transactional functions and data functions found in a use case results in the estimated size FP of the use case.

Step 3: Calculating Estimated Effort of an Iteration

The project manager must, through a history of similar projects or a historical company base, find the estimated productivity of the project. This estimated productivity should take into consideration the particular aspects of the project. The iterations of the project refer to the estimated productivity of the project. The productivity of the iteration cannot be far from the productivity of the project, because it will increase the risk of non-compliance (time and cost).

To reach the estimated effort, in hours, of an iteration, the manager should multiply the estimated size of all use cases of iteration by iteration the estimated productivity.

Step 4: Calculating the Real Productivity of an Iteration

At the end of the iteration the project manager calculates the total hours of actual effort (final), expended by the iteration development. Besides calculating the actual effort, the project manager should make the final FP count of the iteration. These will allow the manager to calculate the actual productivity of the iteration (H / FP).

If there is a deviation in the productivity, the project manager should take actions to adjust the SDP execution. Otherwise the project will be at the risk of delays and/or increase costs. These can impact the product quality.

Step 5: Assessing Impacts on the Actual Productivity

At the end of each iteration, the project manager must answer a checklist of questions to evaluate factors that impacted the actual productivity of the iteration. In doing so, the manager will be able to define the actions to be taken in subsequent iterations, aiming to adjust in real productivity of the next iterations.

The checklist should include questions that allow the evaluation of each SDP phase. The organization using the proposed method can define its own set of questions. Below, we present a set of aspects that can be used in the checklist. All of them are related to aspects found in productivity literature [7, 13]:

1. Project complexity;
2. Project type (e.g. real time, distributed);
3. Innovation support;
4. Development infrastructure ;
5. Work environment;
6. Application integration (to other applications);
7. Team experience (analysis, design and programming);
8. Team motivation, communication and cohesion;
9. Client communication issues;
10. SDP maturity;
11. Reuse (design and code);
12. Requirements change frequency;
13. Non-functional requirements complexity;
14. Programming language complexity;
15. Verification (testing and defect removal);
16. Re-work (change management);
17. Quality standards and issues;
18. Client approval issues;
19. Evolution (maintenance aspects, refactoring, etc.);
20. Changes in the team (inclusion, drops, etc.).

The manager should verify if there were positive or negative impacts of each aspect in the iteration productivity. These will aid the manager to analyze possible process improvements.

4 CASE STUDY

This section presents a case study to show the proposed productivity monitoring approach. The goal is to show that the method allows the project manager to monitor the productivity of the project and give indications of the reasons that are leading to deviations of productivity in phases and iterations.

The example portrayed here refers to a project developed by the energy organization and its development process was divided into three phases, namely: Requirements, Construction and Testing. The distribution of percentage of effort per phase was: 21% for Requirements; 53% for Construction, and; 26% for Testing. These percentages were reported by the project manager. The project was planned to be done in 9 (nine) iterations with a total of fifty five use cases and a team of six persons. At the time of this paper, six iterations have already been performed.

The iteration analysis should include a set of questions, such as:

1. Was the productivity of each iteration better or worse than the initial productivity? Why was that?
2. Has the project manager defined actions to adjust the SDP after each iteration (if necessary)?
3. Did the actions have any effects in the subsequent iterations?

Table 1 presents the distribution, by iteration, of the number of use cases, the FP size and the effort hours for each of the six iterations already carried out. For the sake of simplicity, this study did not present the FP count by use case. The size in FP is presented by iteration. Table 2 shows the actual productivity per iteration and phase. All phases of the development process of this project were estimated at 17.92 H / FP.

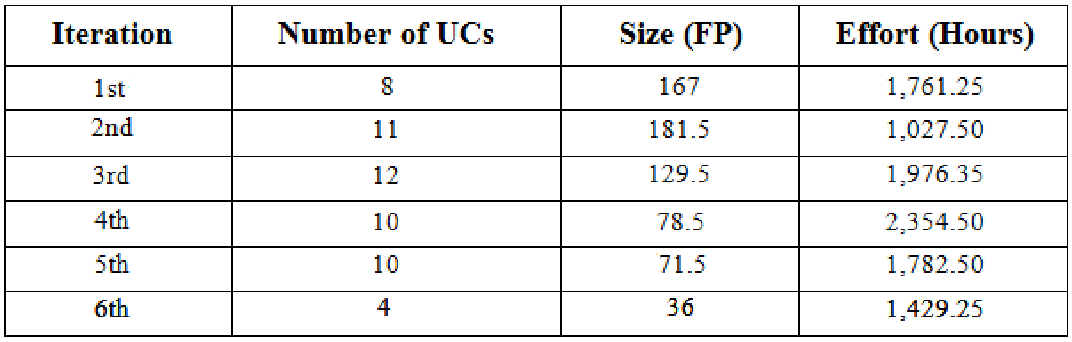
If at the end of an iteration, the phase of the process had productivity lower than the estimated productivity, there is a deviation that can cause higher costs and increased time to deliver the project. If productivity has been better than planned, it should be a review to see if there was over estimation of resources hours, or if all activities of the SDP were properly executed. This may result in product quality decrease, leading to user dissatisfaction.

The project manager created a checklist of questions based on the aspects listed in section 3. The responses to these questions were used as input to perform the analysis of the factors impacting positively and negatively on the productivity of each iteration. Thus it revealed the factors that impact the productivity of iterations along the development cycle of the project.

The description of the six iterations in this study is below.

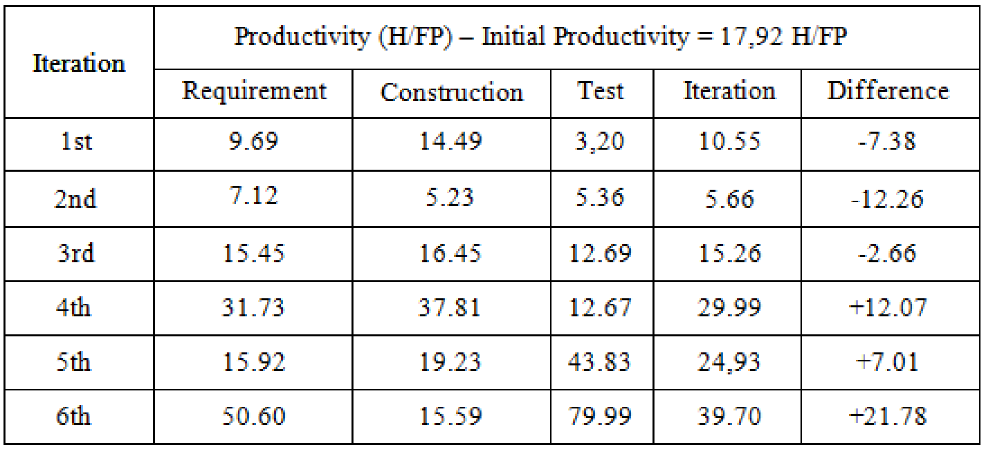
- First iteration (productivity 10.55 H/FP)
  - Strengths:
    - Team: motivated to learn a new technology and a new domain, and; trained before the iteration began;
    - Functionality: CRUD use cases; reuse.
  - Weaknesses:
    - Team: only one member on testing team (unfamiliar with testing tool); requirements team working on different site.
  - Actions taken for second iteration
    - Weekly meetings with all members;
    - Peer reviewing (done by senior analyst).
- Second iteration (productivity 5.56 H/FP)
  - Strengths:
    - Team: testing team increased to two members;
    - Functionality: continued CRUD use cases; reuse.
  - Weaknesses:
    - Team: testing team still unfamiliar with testing tool; weekly meetings did not include requirements members (as they were in another site).
  - Actions taken for third iteration
    - Hire analyst familiar with testing tool;
    - Space provision to move requirements members.
- Third iteration (productivity 15.26 H/FP)
  - Strengths:
    - Team: two new requirement analysts added; another senior analyst added;
    - Functionality: other core use cases (three of the biggest (in size) use cases included).
  - Weaknesses:
    - Team: changes impacted on communications; part of the team was idle;
    - Workplace: not yet completed for all team members;
    - Testing: not automated;
    - Functionality: difficulties with the development of specific functions.
  - Actions taken for fourth iteration
    - Improve workplace (mainly equipments) for team.
- Fourth iteration (productivity 29.99 H/FP)
  - Strengths:
    - None in special.
  - Weaknesses:
    - Team: (workplace impacts related) requirements, development and testing teams worked on different sites; project manager shared time with another project;
    - Testing: not automated;
    - Functionality: intense internal reworking.
  - Actions taken for fifth iteration

Table 1: Distribution of Use Cases, FP and Effort of each iteration.



Iteration	Number of UCs	Size (FP)	Effort (Hours)
1st	8	167	1,761.25
2nd	11	181.5	1,027.50
3rd	12	129.5	1,976.35
4th	10	78.5	2,354.50
5th	10	71.5	1,782.50
6th	4	36	1,429.25

Table 2: Productivity Calculation for Phase and Iteration



Iteration	Productivity (H/FP) – Initial Productivity = 17,92 H/FP				
	Requirement	Construction	Test	Iteration	Difference
1st	9.69	14.49	3,20	10.55	-7.38
2nd	7.12	5.23	5.36	5.66	-12.26
3rd	15.45	16.45	12.69	15.26	-2.66
4th	31.73	37.81	12.67	29.99	+12.07
5th	15.92	19.23	43.83	24,93	+7.01
6th	50.60	15.59	79.99	39.70	+21.78

-
        - None in special.
- Fifth iteration (productivity 24.93 H/FP)
    - Strengths:
        - None in special.
    - Weaknesses:
        - Team: new members were added (but were not experienced); project manager still shared time with another project;
        - Testing: not automated; number of defects increased (including the detection of defects related to previous iterations).
    - Actions taken for sixth iteration
        - Team training.
- Sixth iteration (productivity 39.70 H/FP)
    - Strengths:
        - None in special.
    - Weaknesses:
        - Team: requirements, development and testing teams still worked on different sites; project manager shared time with another project;
        - Testing: not automated; defect complexity increased.

Iterations 1 and 2 present a productivity rate above the project estimation and they deliver the best productivity in the whole project (iterations 1 through 6). This was mainly because the functionality comprised CRUD use cases (with more simple testing), there was a high rate of reuse and the team was highly motivated.

On the other hand, iterations five and six presented the worst productivity rate – way below the first estimate. This increased the risk of deviations from the costs and scheduled previously planned for the project. This was mainly motivated due to the development of more complex use cases, higher fault detection (including faults from previous iterations); higher defect complexity; change in the team, and; a somewhat loose of project management control (the project manager was also assigned to another project).

Such information allows for the assessment of factors impacting the project productivity. The checklist was used to detect these factors. Indeed, the factors were used to devise actions to improve the productivity. Nonetheless, it is important to mention that, although the management tried to take actions in-between the iterations, the productivity did not improve along the project.

5 CONCLUSION

Knowing the actual (final) productivity is key to evaluate the process of a development organization. It serves as input for calibration of the estimated productivity indicator. But measuring and analyzing the real productivity (final) is insufficient to monitor a project.

The use of Function Points to calculate the productivity of a particular project allows it to be compared to other projects. It parameterizes the size of the functionalities and enables the use of historical productivity information to better estimate the schedule and the budget of new projects.

Failure to follow a project can cause serious problems to a software project. Regarding productivity, problems may occur to the time and the cost initially established for the project. It also impacts in the quality of the delivered product. Usually, the project manager only estimates the productivity at the beginning of the project and then calculates actual delivered productivity at the end of the project. If the project manager awaits the completion of the project to evaluate the actual productivity, only the next project may benefit from measures to improve the development process.

When the project manager monitors the productivity of the project, by iteration, it is possible to detect which process phases present lower productivity. With such information, the manager can attempt to take actions to improve the process on-the-fly in order to increase the project productivity. Even if it is not possible to take such actions, the management will have more accurate information about the possible causes of productivity decrease. This information will have an important role in estimating and negotiating new projects.

This paper presented a proposal for defining a process for monitoring the productivity of software projects through the use of productivity indicator monitoring. This indicator is used to assess whether the estimated productivity is being fulfilled during the iterations of the development cycle of the project. The calculation of this indicator is done using the size of the use cases performed in function points, and effort in hours for its completion. This calculation is dismembered by phases, allowing a detailed analysis of what steps need to be improved.

With the implementation of this monitoring process the project manager will able to take actions in the process of adjustment of project development in order to adjust it before it ends. Analyzing the indicator by use case and phase can be used to try to identify the pitfalls of a development process with more accuracy.

6 REFERENCES

[1] A. Albrecht, J. Gaffney. “Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation” – IEEE Transactions on Software Engineering, SE-9, 6, 1983.

[2] A. Abran, P. N. Robillard, “Function Point Analysis: An Empirical Study of Its Measurement Processes”, IEEE Transaction on Software Engineering, Vol. 22, Nº. 12, December 1996.

[3] C. Jones, “Positive and Negative Factors that Influence Software Productivity”, versão 2.0. Software Productivity Research, Inc, 1998.

[4] C. Jones, “Software Estimating Rules of Thumb”, version 3, 2007.

[5] D. N. Card, “The Challenge of Productivity Measurement”. Pacific Northwest Software Quality Conference, 2006.

[6] G. Karner, “Resource Estimation for Objectory Projects”, Objective Systems SF AB, 1993.

[7] G. P. Sudhakar, A. Farooq, S. Patnaik, “Measuring Productivity of Software Development Teams”. Serbian Journal of Management, 2012.

[8] International Function Point Users Group (IFPUG), “Counting Practices Manual (CPM)”, versão 4.3.1, publicado em Janeiro de 2010.

[9] J. T. Joseph, “Role of Function Point as a Reuse Metric in a Software Asset Reuse Program”, International Conference on Software Engineering Research and Practice (SERP) – Las Vegas – Nevada - USA, 2011.

[10] J. Schofield, “The Statistically Unreliable Nature of Lines of Code”. Sandia National Laboratorie, Albuquerque – USA, 2005.

[11] PMI - Project Management Institute, “PMBOK – Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos – Official Portuguese”. 4ª Edição”. São Paulo: Project Management, 2008.

[12] S. Han, S. Lee, “Quantified Comparison and Analysis of Different Productivity Measurements”. Journal of Asian Architecture and Building Engineering, November 2008.

[13] S. Walt, “Understanding Software Productivity”, Software Engineering and Knowledge Engineering: Trends for the Next Decade, D. Hurley (ed.), Vol. 4, World Scientific Press, 1995.

[14] W. W. Agresti, W. M. Evanco, W. M. Thomas, “Models for Improving Software System Size Estimates during Development”. J. Software Engineering & Applications, 2010.