A Productivity Formula Conception for Software Maintenance Projects

Natacha Lascano[1], Sebastián Maniasi[2]

[1] Global Software Group (GSG) Argentina
Motorola Argentina. Av. H. Irigoyen 146. (X5000JHO), 2nd Floor. Córdoba. ARGENTINA
natacha.lascano@motorola.com

[2] Argentina Software Development Center (ASDC)
Intel Argentina. Av. H. Irigoyen 146. (X5000JHO), 16th Floor. Córdoba. ARGENTINA
sebastian.maniasi@intel.com

Abstract. Measuring productivity in maintenance software projects sometimes turns to be a confusing and disconcerting task. Most of the time this situation derives from the fact that requirements tend to be managed as an isolated group of constraints in terms of subsystems, when they actually interact independently over the legacy system, impacting and describing productivity curves arbitrarily different one from the other. This paper aims to assist engineering teams not only better outlining productivity definition (as respects of split it into the key factors that conforms the "Outputs" of the software project), but also providing an objective element for estimates to be used in customer negotiations.

Keywords: software maintenance, productivity, measurement, project planning.

1. Introduction

In contrast with new development, software maintenance is understood as the modification of a software product after delivery to correct faults, improve performance or other attributes, or to adapt the product to a changed environment and then representing probably the major work exponent in software industry.

Yunsik et al. points out that "*most organizations are concerned about the costs of software maintenance, for it has been increasing steadily and many companies spend approximately 80% of their software budget on maintenance*" [1]. Consequently, the software maintenance efforts and costs need to be effectively managed.

Since maintenance projects are a combination of new development and adaptation of existing software, measuring productivity can be a difficult task, and then special aspects have to be considered:

- Inserting new functionality will only be feasible if product's existing architecture can accommodate it.

- Delivering fixes, enhancements or other software adjustments implies release them maintaining system coherence and unaltered former functionality.

- Estimation models to produce effort and schedule estimates must consider modifying a certain amount of existing documentation, code, test cases, etc.

Very often organizations tend to administer modification requests as isolated changes over the product, based on the resulting number of affected lines of code. Generally, this situation leads to underestimate the actual impact over product integrity and to manage project time and resources ineffectively.

To better negotiate, plan and foresee maintenance project performance, it is needed an improved way to measure the outputs of the project as regards of its component factors. Quality is a key factor to take into account since it will end up accounting for costs, customer satisfaction and future revenues.

Productivity data is used in any software organization in order to reduce costs, improve quality and/or increase the rate at which software is developed. To understand productivity requires performing a systematic analysis of a diversity of product types and their associated metrics. Productivity on maintenance software projects needs to be consistently defined in order to be objectively measured and analyzed.

General productivity concepts stands for the amount of output created or produced per unit input used:

$$productivity = \frac{Output}{Input}$$

Scacchi mentions that "*productivity in most studies inside and out of the software world is usually expressed as a ratio of output units produced per unit of input effort*" [2]:

$$productivity = \frac{NumberLOCs(produced)}{StaffMonth(Effort)}$$

In this sense, it is easier to conceive outputs as tangible units such as number of items produced, but other factors such as quality and complexity should be considered. Lines of Code (LOC) is considered to be a crude measure of effectiveness, because it only measures how much was written regardless underlying system integrity. Lo et al. noted that "*the LOC approach has been criticized as follows: (1) there is no universally accepted definition for a LOC (is a comment count as a line of code?); (2) the number of LOC is language and programming style* *dependent; (3) the number of LOC is difficult to estimate prior to system implementation; and (4) the LOC approach overemphasizes the coding stage*" [3].

Productivity is difficult to measure and can only be measured indirectly, that is, by measuring other variables and then calculating productivity from them. This difficulty in measurement stems from the fact that inputs and outputs are not only difficult to define but are also difficult to quantify.

Haziza et al. mentions that "*the resulting uncertainty about productivity and quality in the next software release gives rise to unreliable cost and schedule release estimates*" [4]. Specifically, Bundschuh states that "*there is broad consensus in the metrics community that productivity depends especially on software size and also that there do exist more parameters influencing it*" [5]. A consistent measurement of the productivity metric would help to effectively manage a maintenance software release process.

This paper attempts to delimit these concepts. Input will be simply described in terms of Effort (labor units needed to produce desired outputs) meanwhile, Output will be discomposed taking into account its different intrinsic problem aspects:

- Type of application system.
- Programming language.
- Quality of existing code and documentation (involving concepts such as self-descriptiveness, modularity, simplicity, consistency, expandability, testability, etc.).
- Functional and technical complexity (impact of changes over product).
- Degree of reuse.
- Project team experience.
- Necessity of a complete system test.

Be aware that not all of these concepts may apply at the same time every time, and they also may be split again in sub-concepts depending on the nature of the software problems.

Maintenance activities are frequent and diverse on software project organizations; they differ widely depending on factors such as organizational context, contract type, project organization, product maturity, etc. Some examples are listed below:

- Some changes are intended to keep the software in operation condition; in general, the costs related to these defects repair is absorbed by the supplier organization and needs to be minimized by avoiding misdiagnosing and the appearance of derivate errors.

- Modifications could be related to mandatory changes originated as response of changes in laws, policies or system environment adjustments; these modifications are usually difficult to be predicted and involve high costs and tight schedules because commonly require emergency corrections.

- Enhancements are usually funded by customers and frequently require high efforts for integration and testing that are not always considered.

As result, appropriate project planning and tracking is not possible without accurate and consistent estimations and metrics based on a standardized model; additionally, this kind of estimations and metrics allow being successful on customer negotiations that are related to time and costs factors in maintenance projects.

All these examples emphasize two key problems that every Project Manager will face during the execution of his/her daily activities: perform a proper planning and carry out successful negotiations.

2. Productivity on Maintenance SW Projects

2.1. The New Productivity Formula

Boehm [6] identifies some of the problems encountered in defining what needs to be measured to describe software productivity. Particularly, observes that software development inputs include more than only coding efforts (activities devoted to documentation production for example) and that "*measuring software development outputs solely in terms of attributes of the delivered software (e.g., delivered source code statements or function points) poses a number of dilemmas: (a) complex source code statements or complex combinations of instructions usually receive the same weight as sequences of simple statements; (b) determining whether to count non-executable code, reused code, and carriage returns as code statements; and (c) whether to count code before or after pre- or post-processing*".

Additionally, Deraman [7] identifies three major common phases of the software maintenance models; these are: (1) understanding the software (called "*analysis*" in the context of these work), (2) modifying the software (called "*construction*" in the context of these work), and (3) revalidating the software (called "*testing*" in the context of these work). All the activities included in these phases should be considered as part of the definition of the productivity metric.

Another consideration, according to Peters [8], is that using traditional estimations models for maintenance projects may tend to over-estimate their associated efforts because these models are calibrated to produce estimates assuming that everything is created from scratch; for this, when a productivity metric is defined it is necessary to consider that a certain percentage of existing documentation, code and test cases will be reused.

Finally, Canfora et al. defines the impact analysis as "*the identification of the work products affected by a proposed modification*" [9], it looks for potential impact on:

existing modules, other systems, hardware, documentation, data structures and humans. In summary, this activity prevents from inserting defects that are not immediately apparent and allows determining the *complexity* of a change. Boehm [6] finds that productivity is strongly influenced by a collection of software engineering attributes and concludes that complexity has the greatest affect in driving software productivity. For this reason, results of impact analysis need to be reflected on productivity's definition.

Based on the general productivity formula, using the ideas explained before and considering the concepts and characteristics related to the software maintenance activities; productivity formula for maintenance software projects has been extended as follows:

$$P = \left(\frac{\overbrace{(AD + AC)}^{Analysis} + \overbrace{(CLOCs + CD)}^{Construction} + \overbrace{(TLOCs + TD)}^{Testing}}{(EA + EC + ET)} \right) * RF * IAF$$

Where:

- **P** → Productivity.
- **AD** → Analyzed Documentation.
- **AC** → Analyzed Code.
- **CLOCs** → Construction (Product) Lines Of Code.
- **CD** → Construction (Product) Documentation.
- **TLOCs** → Testing Lines Of Code.
- **TD** → Testing Documentation.
- **EA** → Effort on Analysis Activities.
- **EC** → Effort on Construction Activities.
- **ET** → Effort on Testing Activities.
- **RF** → Reuse Factor.
- **IAF** → Impact Analysis Factor.

2.2. Formula Components

In order to gain clarity on the proposal and with the intention of allowing a better understanding of the new productivity formula, each one of its components will be explained by means of a series of fields:

- **Name:** The term by which a component is designated and distinguished from others.
- **Acronym:** A word formed from the initial letters of a component's name. This will be used as the unique identifier of the component.
- **Definition:** An explanation of the meaning of the component.
- **Relevance:** An explanation of the significance of the component.

- **Measurement Unit:** The division of quantity accepted as the standard of measurement for a component.

Name	Analyzed Documentation
Acronym	AD
Definition	An amount representing the analyzed collection of records that describe the structure, purpose, operation, maintenance and data requirements of the software to be modified.
Relevance	Essential task in a software maintenance life cycle.
Measurement Unit	Pages

Name	Analyzed Code
Acronym	AC
Definition	An amount representing the analyzed set of statements and instructions related to the software to be modified.
Relevance	Essential task in a software maintenance life cycle.
Measurement Unit	LOCs

Name	Construction (Product) Lines Of Code
Acronym	CLOCs
Definition	An amount representing the statements and instructions created, modified or deleted for implementing the modifications required.
Relevance	Essential task in a software maintenance life cycle.
Measurement Unit	LOCs

Name	Construction (Product) Documentation
Acronym	CD
Definition	An amount representing the records created, modified or deleted for describing the structure, purpose, operation, maintenance and data requirements of the modifications required.
Relevance	Essential task in a software maintenance life cycle.
Measurement Unit	Pages

Name	Testing Lines Of Code
Acronym	TLOCs
Definition	An amount representing the statements and instructions created, modified or deleted for completing a critical evaluation of the software modifications.
Relevance	Essential task in a software maintenance life cycle.
Measurement Unit	LOCs

Name	Testing Documentation
Acronym	TD

Definition	An amount representing the records created, modified or deleted for completing a critical evaluation of the software modifications.
Relevance	Essential task in a software maintenance life cycle.
Measurement Unit	Pages

Name	Effort on Analysis Activities
Acronym	EA
Definition	An amount representing the energy used to separate the software whole into its constituent parts for individual study or the study of such constituent parts and their interrelationships in making up the software whole.
Relevance	Analysis is one of the three major common phases of the software maintenance models.
Measurement Unit	Staff/Hour, Staff/Day, Staff/Week, Staff/Month, Staff/Year

Name	Effort on Construction Activities
Acronym	EC
Definition	An amount representing the energy used to implement the modifications required.
Relevance	Construction is one of the three major common phases of the software maintenance models.
Measurement Unit	Staff/Hour, Staff/Day, Staff/Week, Staff/Month, Staff/Year

Name	Effort on Testing Activities
Acronym	ET
Definition	An amount representing the energy used to complete a critical evaluation of the software modifications.
Relevance	Testing is one of the three major common phases of the software maintenance models.
Measurement Unit	Staff/Hour, Staff/Day, Staff/Week, Staff/Month, Staff/Year

Name	Reuse Factor
Acronym	RF
Definition	A value used to indicate an increase or decrease in productivity derived from the reuse of existing code or documentation (for both, construction and testing).
Relevance	This factor allows calibrating traditional estimations models in order to consider the reuse of existing documentation, code and test cases on software maintenance projects.
Measurement Unit	A percentage. $rf = 1 \rightarrow$ Reuse Factor does not impact on productivity. $rf > 1 \rightarrow$ Reuse Factor does produce an increase on productivity. $rf < 1 \rightarrow$ Reuse Factor does produce a decrease on productivity.

Name	Impact Analysis Factor
Acronym	IAF
Definition	A value used to indicate an increase or decrease in productivity derived from the impact of the software modifications on existing code or documentation (for both, construction and testing).
Relevance	Complexity (described by the value of this factor) is a software engineering attributes with high impact on software productivity.
Measurement Unit	A percentage. *iaf* = 1 → Impact Analysis Factor does not impact on productivity. *iaf* > 1 → Impact Analysis Factor does produce an increase on productivity. *iaf* < 1 → Impact Analysis Factor does produce a decrease on productivity.

Reuse and Impact Analysis factors are overall and high level calibration percentages that have been included on the formula with the intent of balance productivity. During a first estimation approach, it is not expected to differentiate the percentages corresponding to these factors per maintenance phase.

2.3. A Sample Application of the New Productivity Formula

Scacchi affirms that "*productivity measures are comparable when counting the same kind of outputs (e.g., lines of source code) and inputs (person-months of time). Therefore, how outputs and inputs are defined are critical concerns if they are to be related as a ratio-type measure*" [2]. As it was stated before, each component of the new productivity formula is measured by using a specific unit. Based on these statements, it is possible to affirm that a criterion for applying a common measurement unit for both inputs and outputs is required in order to measure and exploit productivity data; this criterion could be based on industry data, organizational directions, project's history or any other source.

In order to measure productivity data in the sample application of the new productivity formula, the following criteria have been defined:

- **Inputs:** Staff/Days will be used as common measurement unit.
- **Outputs:** KAELOCs (Thousand of Assembly Equivalent Lines of Code)[*] will be used as common measurement unit.

[*] KAELOCs is defined as number of line of code in assembly language generate by one line of code on a higher-level programming language.

Calculating productivity on maintenance projects by using the new productivity formula will require completing the three steps that follows:
1. Collect the metrics.
2. Convert the measurements.
3. Determine productivity.

2.3.1. Collecting Metrics

This activity implies to collect the values that have been gathered for all the components of the new productivity formula.

Listed below are the sample values corresponding to each of the components.

- **Analyzed Documentation:** 10 Pages.
- **Analyzed Code:** 10 PERL LOCs and 50 C LOCs.
- **Construction (Product) Lines Of Code:** 115 C LOCs (100 C LOCs created, 10 C LOCs modified, 5 LOCs deleted).
- **Construction (Product) Documentation:** 1 Page.
- **Testing Lines Of Code:** 0 LOCs (Not applicable).
- **Testing Documentation:** 1 Page.
- **Effort on Analysis Activities:** 2 Staff/Days.
- **Effort on Construction Activities:** 35 Staff/Hours.
- **Effort on Testing Activities:** 0.075 Staff/Months.
- **Reuse Factor:** 1.05 (Existing test code can be reused in order to validate the modifications).
- **Impact Analysis Factor:** 0.90 (Modifications are related to an interface that will require to modify dependent components).

2.3.2. Converting Measurements

This activity implies to convert the values that have been gathered for all the components of the new productivity formula to the common measurement unit defined for both: inputs and outputs (the criteria selected for applying a common measurement unit in this sample is based on industry data).

- **Inputs:** Staff/Days.
  - **Effort on Analysis Activities**
    2 Staff/Days = 2 Staff/Days
  - **Effort on Construction Activities**
    35 Staff/Hours = 4.375 Staff/Days
    Criterion: 1 Staff/Day is equivalent to 8 Staff/Hours.
  - **Effort on Testing Activities**
    0.075 Staff/Months = 1.5 Staff/Days
    Criterion: 1 Staff/Month is equivalent to 20 Staff/Days.

- **Outputs:** KAELOCs.
  - **Analyzed Documentation**
    10 Pages = 2040 KAELOCs
    Criterion: 1 Page contains 34 lines of text and 1 line of text is equivalent to 6 Assembly LOCs.
  - **Analyzed Code**
    10 PERL and 50 C LOCs = 275 KAELOCs
    Criterion: 1 PERL LOC is equivalent to 15 Assembly LOCs and 1 C LOC is equivalent to 2.5 Assembly LOCs.
  - **Construction (Product) Lines Of Code**
    115 C LOCs = 287.5 KAELOCs
    Criterion: 1 C LOC is equivalent to 2.5 Assembly LOCs.
  - **Construction (Product) Documentation**
    1 Pages = 204 KAELOCs
    Criterion: 1 Page contains 34 lines of text and 1 line of text is equivalent to 6 Assembly LOCs.
  - **Testing Lines Of Code**
    0 LOCs = 0 KAELOCs
  - **Testing Documentation**
    1 Pages = 204 KAELOCs
    Criterion: 1 Page contains 34 lines of text and 1 line of text is equivalent to 6 Assembly LOCs.

2.3.3. Determining Productivity

This activity implies to calculate the productivity based on the converted values for all the components of the new productivity formula.

$$P = \left(\frac{(2040 + 275) + (287.5 + 204) + (0 + 204)}{(2 + 4.375 + 1.5)} \right) * 1.05 * 0.90$$

Finally, the **productivity** obtained is **361.26 KAELOC per Staff/Day** (productivity calculated in the traditional way, on the other hand, would be 36.50 KAELOC per Staff/Day).

3. Conclusion

One of the most relevant project management concerns for any software organization is to determine and to control the maintenance costs, since the cost and frequency of maintenance activities is constantly increasing. The major maintenance cost driver is associated with the development team's productivity and the factors that affect productivity on software maintenance projects should be differentiated from those related to software development projects.

This work proposes a new formula for determining productivity on maintenance projects; it includes the main variables that need to be measured in order to calculate the productivity. This formula can be effectively applied and adapted according to the specific characteristics of any software maintenance project and should be used as a tool for assisting engineering teams on establishing their productivity and carrying out successful negotiations. Researchers and developers are encouraged to verify the feasibility of this proposal in empirical studies.

4. Future Lines of Research

The resulting formula framework yielded from this study offers the prospective for further analysis in two probable directions:

1. Determining the effectiveness of the overall formula based on empirical evidence, and

2. Adjusting each key formula element to a "*weighted traversal*" representation of users' models so to "*calibrate*" it to any specific problem domain and business needs.

In this sense, future results will be dealing with organizations activity effectiveness and having a measure of how well they are keeping up with business they attend. Proper planning and customer negotiations must be supported by proper data and realistic work possibilities.

Acknowledgements

We would like to thank Patricio Maller for his valuable comments and suggestions and giving us very useful feedback.

References

[1] Yunsik Ahn, Jungseok Suh, Seungryeol Kim and Hyunsoo Kim. *The software maintenance project effort estimation model based on function points*. Journal of Software Maintenance and Evolution: Research and Practice. 2003; 15:71–85 (DOI: 10.1002/smr.269).

[2] Scacchi Walt. *Understanding Software Productivity*. Advances in Software Engineering and Knowledge Engineering. Volume 4, pp. 37-70. 1995.

[3] Lo, R., R. Webby, and R. Jeffery. *Sizing and Estimating the Coding and Unit Testing Effort for GUI Systems. Proceedings of the third international software metrics symposium.* Los Alamitos, CA: IEEE Computer Society Press. 166-73. Berlin, Germany, March 25-26, 1996.

[4] Haziza, Voidrot, Minor, Pofelski and Blazy. *Software Maintenance: An Analysis of Industrial Needs and Constraints*. Proceedings Conference on Software Maintenance. 1992.

[5] Bundschuh Manfred. *Estimation of Maintenance Tasks*. International Workshop on Statistical Modelling. 2002.

[6] Boehm, B. *Improving Software Productivity*. Computer. Volume 20, pp. 43-58. 1987.

[7] Deraman Aziz. *Requirement for a Software maintenance Process Model: A Review*. Malaysian Journal of Computer Science. Volume 8, Number 2, pp. 174-202. 1995.

[8] Peters Kathleen. *Software Project Estimation*. Software Productivity Center Inc. 1999.

[9] Canfora and Cerulo. *Impact Analysis by Mining Software and Change Request Repositories*. Proceedings of International Symposium on Software Metrics. IEEE Press. 2005.