Sylvie Trudel[1], Alex Turcotte[2]

Combining Qualitative and Quantitative Software Process Evaluation: A Proposed Approach

Abstract

This paper presents a method that combines a qualitative assessment method based on CMMI to uncover improvement areas of an organization's software process, and a quantitative approach to measure the software process productivity rate. The evaluation scope is first established: a list of projects and a list of CMMI key process areas to assess. The measurement of the productivity rate is obtained by measuring the functional size of the software developed and/or enhanced through projects, and then compared with the recorded project effort. The main reason for combining the approaches is to gain a deeper understanding of the organization's software process by examining the software requirements artefacts, which reveals the weaknesses of the requirements engineering portion of the software process. As a proof of the concept, a field trial of the combined method has been applied successfully in an organization developing financial trading systems.

Keywords: software process assessment, functional size measurement, COSMIC, software productivity, requirements defects

1. Introduction

Organizations developing software have been using best practices models for more than two decades as guides to help them improve their software process[3]. Applying practices from these best practices models would help software practitioners to alleviate some, if not most, of the unwanted symptoms frequently reported in software engineering, such as budget overruns, schedule overruns, partially delivered functionalities, poor product quality, most likely leading

[1] Université du Québec à Montréal, Montreal, Canada, trudel.s@uqam.ca

[2] Université de Sherbrooke, Longueuil, Canada, Alex.Turcotte@usherbrooke.ca

[3] J.J. Marciniak, *Software Engineering. A Historical Perspective*, Encyclopedia of Software Engineering, Wiley 2002.

to deceived expectancies of their users on features, cost, schedule, and quality, and to dissatisfied customers.

These types of software process models apply quite well to large organizations, when they put in place a group responsible to develop, evolve, and communicate their software process to all stakeholders, developers and managers alike. It may be otherwise for smaller organizations, often due to a lack of resources invested in software process improvement activities and support. Nonetheless, several software process models are freely available and as well as adaptation guidelines for small settings, either small projects (less than 15 staff members) or small organizations (less than 50 staff members)[4].

1.1. Software Process Models and Frameworks

The Software Capability Maturity Model (SW-CMM) was among the first software engineering best practices models widely used[5], providing a "staged" representation of five levels. Several years later, the Software Engineering Institute (SEI) issued and evolved a framework of models integrating practices related to software engineering, systems engineering, software acquisition and team work, called the Capability Maturity Model Integration for Development (CMMI–DEV)[6]. These models were developed by a team of experienced practitioners combining their lessons learned throughout many years of working on large software projects, whether these projects were successful or not[7].

CMMI–DEV provides a "staged" representation containing 22 process areas. Each level from 2 to 5 of the staged representation holds a subset of these process areas. There is no practice in level 1. With the staged representation, these levels are called "maturity levels":

- Level 1: Initial – characterized by an *ad hoc* or chaotic process;
- Level 2: Managed – characterized by a planned and executed process, following a written organizational policy;

---

4 S. Garcia-Miller, *Lessons Learned from Adopting CMMI in Small Organizations*, Software Engineering Institute 2005, http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=19950

5 M.C. Paulk, *SW-CMM V1.1*, Software Engineering Institute 1992.

6 The CMMI Product Team, *Capability Maturity Model Integration © for Development, Version 1.3*, The CMMI Institute 2010, CMU/SEI-2010-TR-033.

7 M.B. Chrissis, M. Konrad, S. Shrum, *CMMI Guidelines for Process Integration and Product Improvement*, Addison-Wesley 2003.

- Level 3: Defined – where the process is defined at the organizational level with tailoring guidelines, tools, and skilled people, and is applied consistently across projects;
- Level 4: Quantitatively Managed – there are practices in place to establish and reach "quantitative objectives for quality and process performance and use them as criteria in managing projects"[8];
- Level 5: Optimizing – characterized by a process that is being continuously improved based on quantitative feedback on process performance and product quality.

CMMI–DEV also provides a continuous representation, having the same 22 process areas but divided into four categories (Project management, Engineering, Process Management, and Support) instead of maturity levels. The selection of the CMMI representation depends on the goals set by given organizations. When an acquirer requires that a software contractor has its processes evaluated at a certain CMMI maturity level, then this software contractor is likely to select the staged representation. When the software organization embarks on a process improvement journey to keep ahead of its competitors or to continuously better serve its customers, then it might adopt either of the continuous representations and target those process areas where an assessment, or evaluation, would reveal the most important weaknesses.

There also exist other software process frameworks available for small software organizations, depicting the software life cycle such as ISO/IEC TR 29110[9], which is also available for free. It comprises an assessment guide[10] usable internally by an organization to evaluate areas of improvement of their software process.

Most organizations choosing CMMI, or any other software process or lifecycle model, would be interested in evaluating the progress made over a period of time on their software process. The next section discusses some of the available assessment methods.

---

[8] The CMMI Product Team, *Capability Maturity Model Integration © for Development, Version 1.3*, The CMMI Institute 2010, CMU/SEI-2010-TR-033, p. 28.

[9] ISO/IEC 29110–1, *Software Engineering – Lifecycle Profiles for Very Small Entities (VSEs) – Part 1: Overview*, ISO 2011; ISO/IEC 29110–2, *Software Engineering – Lifecycle Profiles for Very Small Entities (VSEs) – Part 2: Framework and Taxonomy*, ISO 2011.

[10] ISO/IEC 29110–3, *Software Engineering – Lifecycle Profiles for Very Small Entities (VSEs) – Part 3: Assessment Guide*, ISO 2011.

1.2. Qualitative Software Process Evaluation

Software process evaluation methods were often developed to verify and report on the application of a best practices model in a given context. The SEI has developed the Standard CMMI Appraisal Method for Process Improvement (SCAMPI)[11] with three classes of appraisals:

- Class A: has to be done by a lead appraiser (certified by the CMMI Institute), requires a rigorous caption of objective evidences for every practice evaluated within the appraisal scope, and provides a rating according to the levels described in section 1.1, allowing the organization to benchmark its process maturity with its industry. A class 'A' SCAMPI may cost several tens of thousands USD[12], an amount inaccessible to most small software organizations.
- Class B: Similar to a class 'A' SCAMPI but without producing a rating. Less costly than class 'A' SCAMPI, it is intended for interim assessment between formal class 'A' SCAMPIs.
- Class C: requires less objective evidences than class 'A' or 'B' SCAMPI. It can be used internally by an organization to gain some insight in the process maturity, without providing a rating, and without the obligation to be done by a certified lead appraiser.

The SCAMPI method provides insight of the software process. Only class 'A' provides a rating, which may be considered as a 'quantitative' evaluation. However, from a process improvement perspective, the rating has less importance than the findings in terms of strengths and weaknesses of the evaluated software process against CMMI. Since all three classes of SCAMPI provide such findings, we consider SCAMPI as a qualitative type of software process evaluation.

ISO/IEC 29110–3 also provides insight of the software process, namely findings against the ISO/IEC 29110–2 framework. It is available for free and it targets specifically very small entities while being limited to the evaluation of the process.

Nonetheless, there are cases in the industry where organizations require more insights of the software process, such as having also insights of the software product, in order to have a broader perspective on what can be improved. In such cases, a combined evaluation method is desirable, one that would provide

[11] CMMI Institute, *Standard CMMI Appraisal Method for Process Improvement (SCAMPI) Version 1.3b: Method Definition Document for SCAMPI A, B, and C*, Technical Report, The CMMI Institute 2014.

[12] F. Nguyen, *Estimated Cost for SCAMPI Class A, B, or C Appraisal*, 2010, http://cmmirocks.ning.com/forum/topics/estimated-cost-for-scampi (08/07/2015).

findings about software products explaining some of the process-related findings. We did publish such a method several years ago, called PEM for the Process/Product Evaluation Method, including field trials[13]. The PEM uses ISO/IEC 14598–5[14] as the evaluation method for the software product against product quality criteria – in its original version it was the ISO 9126 standard – and adds CMMI practices as an input to evaluate the software process at the same time. The PEM was intended for small organisations or small projects, where an evaluation could be conducted in less than a week, providing findings on both the software process and the software product. Field trials showed that the dual process-product evaluation was achievable also in larger settings, which require more effort to evaluate[15].

The PEM has seven steps, the first six are extracted from ISO/IEC 14598–5, and the seventh was created as a value-added step for the evaluated organization (see Figure 1). Steps 4 and 5 together constitute the execution of the evaluation. Similar to a class B SCAMPI, the PEM does not provide a rating. Every observation must be supported by comments from participants in at least two different interviews or by comments made by one participant and supported by adequate documentation. In step 4, evaluators must review project documentation against process model practices but also against software quality attributes. Those quality attributes could come from those defined in ISO/IEC 9126, but any other set of quality criteria can be used, as long as they are adequately defined during step 2 (when a model is selected) and step 3 (when a list of questions is prepared). Project documentation is deemed inclusive to any type of information required for the evaluation, such as architecture, requirements, design, code, test, project management, and any other documentation relevant within the evaluation scope. Confidentiality of the participants' comments is enforced by a signed agreement from the evaluators, stipulating that only comments adequately corroborated will be presented as observations, after being rephrased and approved by the participants themselves (steps 5 and 6).

---

13 S. Trudel, J.M. Lavoie, M.C. Paré, W. Suryn, *PEM: The Small Company-dedicated Software Process Quality Evaluation Method Combining CMMI℠ and ISO/IEC 14598*, "Software Quality Journal" 2006, vol. 14, no. 1, pp. 7–23.

14 ISO/IEC 14598–5, *Information Technology – Software Product Evaluation – Part 5: Process for Evaluators*, ISO 1998.

15 S. Trudel, J.M. Lavoie, M.C. Paré, W. Suryn, *PEM: The Small Company-dedicated Software Process Quality Evaluation Method Combining CMMI℠ and ISO/IEC 14598*, "Software Quality Journal" 2006, vol. 14, no. 1, pp. 7–23.

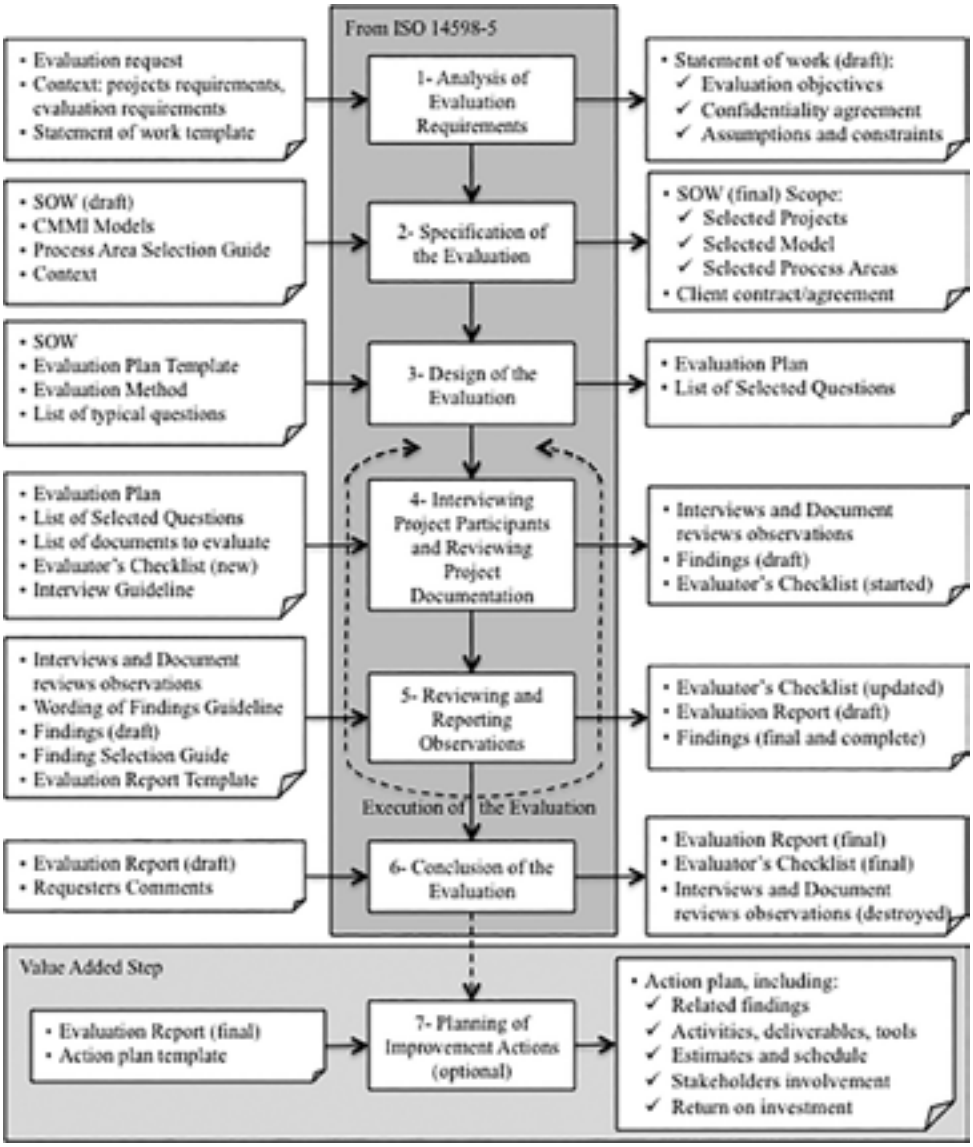

Figure 1. The PEM Method

Source: S. Trudel, J.M. Lavoie, M.C. Paré, W. Suryn, PEM: The Small Company-dedicated Software Process Quality Evaluation Method Combining CMMISM and ISO/IEC 14598, "Software Quality Journal" 2006, vol. 14, no. 1, pp. 7–23.

If an organization needs to evaluate the quality of other outputs of its software process, e.g. the requirements documents instead of the software code or design, then a different set of quality criteria would be needed since ISO/IEC 9126 does not cover that aspect.

The objective of this paper is to present an adaptation to the PEM allowing the evaluation of process aspects against a best practice model, and the quality of a software product aspect, namely the requirements by means of measuring the functional size, hence, combining a qualitative approach with a quantitative one. The main hypothesis is that the findings from measuring the size would explain some of the process related findings, specifically for practices developing or using requirements documents.

2. Quantifying the Software Process Efficiency

Quantifying the efficiency of a process can be expressed in terms of the effort per size unit, which requires that the size of the process output be measured and then compared with the effort made during that process execution. By extension, quantifying the efficiency of a software process requires that the software size be measured and then compared with the effort made to produce the output of that measured size[16].

2.1. Measuring the Software Functional Size

The type of output to measure can be the software functional size, measured from functional requirements documents[17]. There are five standardized methods published by the International Organisation for Standardization (ISO): IFPUG[18], NESMA[19], FiSMA[20], Mark-II[21], and COSMIC[22].

---

16 A. Abran, *Software Metrics and Software Metrology*, Wiley 2010.

17 ISO/IEC 14143–1, *Information Technology – Software Measurement – Functional Size Measurement – Part 1: Definition of Concepts*, ISO 2007.

18 ISO/IEC 20926, *Software and Systems Engineering – Software Measurement – IFPUG Functional Size Measurement Method 2009*, ISO 2009.

19 ISO/IEC 24570, *Software Engineering – NESMA Functional Size Measurement Method Version 2.1 – Definitions and Counting Guidelines for the Application of Function Point Analysis*, ISO 2005.

20 ISO/IEC 29881, *Information Technology – Systems and Software Engineering – FiSMA 1.1 Functional Size Measurement Method*, ISO 2010.

21 ISO/IEC 20968, *Software Engineering – Mk II Function Point Analysis – Counting Practices Manual*, ISO 2002.

22 ISO/IEC 19761, *Software Engineering – COSMIC: A Functional Size Measurement Method*, ISO 2011.

The Common Software Measurement International Consortium (COSMIC) software functional size method (FSM) is the only published standard being of the second generation of FSM methods. The COSMIC measurement manual is available for free and can be downloaded from the COSMIC organisation website in 13 different languages[23]. The COSMIC method can be used to measure the software produced as one of the outputs of a software project, i.e. the output corresponding to functional requirements. The COSMIC method measurement process has three phases:

1. Phase 1 – the measurement strategy, is completed at the beginning of a measurement activity, where the measurement purpose is determined along with the measurement scope, software layers and level of decomposition, functional users, and the FUR level of granularity.
2. Phase 2 – the mapping phase, is when data is extracted from the Functional User Requirements (FUR) to identify functional users, triggering events, functional processes, data groups manipulated by these functional processes, and their related data movements. There are four types of data movements: Entry (E), eXit (X), Read (R), and Write (W).
3. In phase 3, the measurement phase, data movements are counted. It uses the COSMIC Function Point (CFP) as its unit of measure, where one CFP represents a data movement of one data group by one functional process, within a piece of software to measure. The functional size of a piece of software is the total number of data movements.

Phases 2 and 3 can be done iteratively until the measurement scope has been completed. Figure 2 provides an overview of the generic software flow of data from a functional perspective, where data movement types are shown.

The COSMIC method can measure the size of new functionalities as well as the size of modified functionalities; other methods measure the size of new functionalities but can only estimate the size of modifications, mainly by applying a percentage of the original size of the functionality being modified. Since it is common in the industry that a software project scope comprises new and modified functionalities, the COSMIC method has a clear advantage over the other methods when precision is required.

[23] A. Abran et al., *The COSMIC Functional Size Measurement Method, Version 4.0.1*, Measurement Manual, The COSMIC Group 2015, http://www.cosmicon.com/dl_manager4.asp?id=532

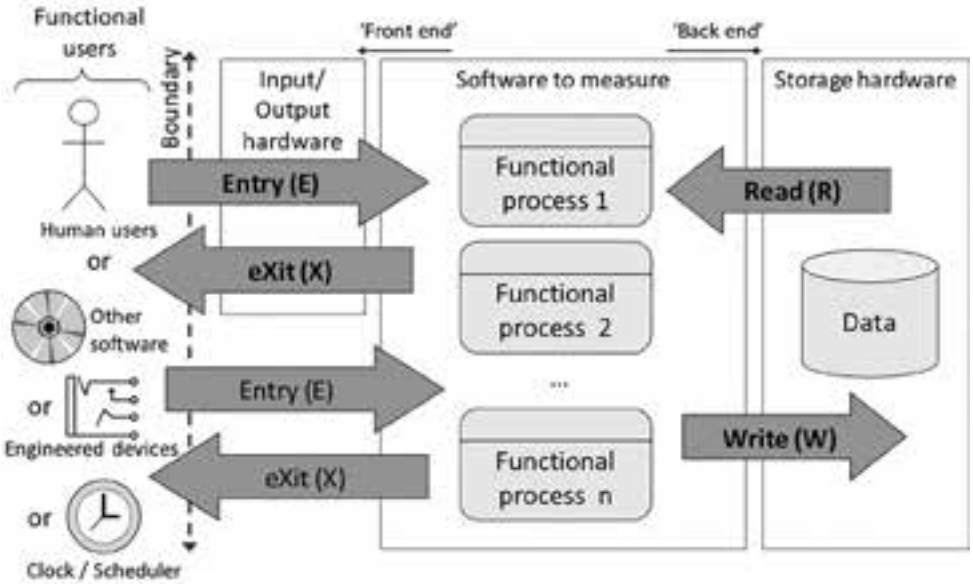

Figure 2. Generic Software Flow of Data from a Functional Perspective[24]

Source: the authors' own work.

2.2. Identifying Defects in Functional Requirements

In previous work, we used the COSMIC method as a means to identify defects in software requirements documents while measuring the software functional size[25]. As measurement is made by going through the FUR, a deep understanding of these FUR is obtained during phase 2 of the measurement process. While measuring a piece of software, ambiguities and inconsistencies can be identified, such as:

- Unclear or missing functional users;
- Unclear or missing triggering events;
- Unclear or missing functional processes;
- Unclear or missing data groups;
- Unclear or missing data movements.

These ambiguities and inconsistencies can all be considered as defects in the FUR. The likely consequences of these defects are incomplete, wrong or at least

[24] A. Abran, *Software Metrics and Software Metrology*, Wiley 2010.

[25] S. Trudel, *Using the COSMIC Functional Size Measurement Method (ISO 19761) as a Software Requirements Improvement Mechanism*, Thesis, École de Technologie Supérieure 2012.

imprecise measurement results, but more importantly misunderstandings from the software developers leading to higher project costs as rework is likely to be required.

Desharnais and Abran proposed a quality rating of the FUR[26] to help understand the precision of a measurement result. These quality rates range from 'a' to 'e', where 'a' means all relevant information for measurement is clear, consistent, and available, and 'e' means "the functional process is not mentioned in the artefacts but is implicit". In other words, at level 'e' functional processes being the basis of measurement have to be guessed, which cannot provide a precise measurement result when poor quality FUR are to be used. Nonetheless, assumptions can be made when a defect is found in the requirements impacting measurement results. The COSMIC Group has issued the Guideline for assuring the accuracy of measurements[27], guiding measurers to detect defects in the FUR and identifying appropriate assumptions for measurement.

Previous research also found that defects in the FUR are likely to cause measurement errors, especially with inexperienced measurers[28]. For that reason, measurement results made by inexperienced measurers should be verified by a certified and experienced measurer who would be able to identify measurement errors and also identify defects in the FUR for which the measurement results should be adjusted with adequate assumptions[29].

2.3. Quantifying the Software Process Productivity Rate

Once the functional size of the software delivered by the projects has been measured, it can be compared with the project effort to obtain a productivity rate expressed in staff-hours per CFP. However, before analyzing this data, it is

---

26 J.-M. Desharnais, A. Abran, *Assessment of the Quality of Functional User Requirements Documentation Using Criteria Derived from a Measurement with COSMIC–ISO 19761*, International Workshop on Software Measurement – IWSM 2010, Stuttgart, November 2010, pp. 481–496.

27 A. Abran et al., *Guideline for Assuring the Accuracy of Measurements, Version 1.0*, The COSMIC Group 2011, http://www.cosmicon.com

28 E. Ungan, O. Demirörs, Ö.Ö. Top, B. Özkan, *An Experimental Study on the Reliability of COSMIC Measurement Results*, in: *Software Process and Product Measurement*, Springer, Berlin Heidelberg 2009, pp. 321–336; S. Trudel, A. Abran, *Functional Size Measurement Quality Challenges for Inexperienced Measurer*, in: *Software Process and Product Measurement*, International Conferences IWSM 2009 and Mensura 2009 Amsterdam, 4–6.11.2009, proceedings vol. 5891, Springer Science & Business Media, pp. 157–169.

29 A. Abran et al., *Guideline for Assuring the Accuracy of Measurements, Version 1.0*, The COSMIC Group 2011, http://www.cosmicon.com

essential to understand the quality of both data types: measurement results and effort. Measurement result quality has been discussed in the previous section and should have been verified at that point. Quality assurance of effort data should be part of the qualitative portion of the evaluation, as quality of effort data is of primary importance in project cost management and reporting.

Once the quality of effort and FSM data have been assured, the productivity rate can be analyzed, benchmarked[30], and reported. The analysis of the productivity rate should go beyond establishing the rate and provide a broader view of software process.

3. Combining Qualitative and Quantitative Software Process Evaluations

Most methods evaluating a software process for improvement are assessing the software practices against a given model by interviewing project participants and by reviewing artefacts issued from that process, searching evidences that the model practices are implemented. Some issues specifically about the requirements engineering process may be overlooked, such as inconsistencies of the quality of requirements artefacts, even when all projects are using the same requirements document template. The form may look consistent, but the quality of the content is not.

When the functional size of several projects is measured during a software process assessment, inconsistencies across projects requirements artefacts and their quality rating can be identified. These inconsistencies need to be identified during the evaluation, before the draft report is released for review to evaluation participants. We found that an appropriate combination of measurement and assessment can be done as part of the "Execution of the Evaluation" (steps 4 and 5 of the PEM, from Figure 1). Our approach proposes a new step "4b" between steps 4 and 5, where step 4 becomes step "4a" (see Figure 3). In that case, the review for the requirements engineering process outputs is used as input for the COSMIC FSM method – the *de facto* selected FSM method. Identified defects, such as inconsistencies across projects, will lead to findings related

30 International Software Benchmarking Standards Group (ISBSG), *Development and Enhancement Data Set 2015, Release 13*, Obtained from http://www.isbsg.org

to software requirements engineering practices. Steps 4a and 4b can be executed in parallel and iteratively.

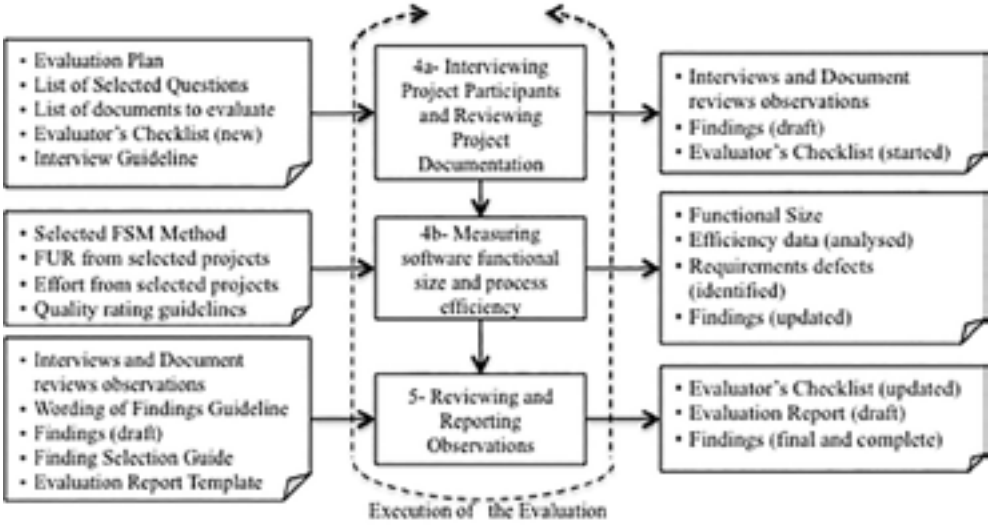

Figure 3. Proposed Added Step to Combine Qualitative and Quantitative Approaches
Source: adapted from Abran A., *Software Metrics and Software Metrology*, Wiley 2010.

Moreover, since it is quite frequent that organizations are having issues with their estimation process, establishing the productivity rate (from efficiency data analysis in step 4b) for the subset of evaluated projects is likely to provide improvements of the estimation process, such as integrating an estimation model to compare traditional project estimates with the results from that model. Also, in the context of a software process evaluation, this productivity rate analysis should provide insights to the requirements engineering process, linking the productivity rate and relevant benchmarking results to process related findings.

4. Field Trial in the Financial Trading Software Domain

A proof of the concept for applying the combined evaluation was required in order to identify the aspects of the method that would be adequate and those that would need improvements or adjustments. An organization accepted that our method be applied on their software process. Its management asked that

the organization's name remained confidential while we were granted the permission to publish some of the evaluation results.

4.1. Analysis of the Evaluation Requirements (Step 1)

The combined qualitative and quantitative evaluation of the software process has been applied in this software organization of 100 staff members, developing trading software for the derivatives market. Their different systems are "recognized as among the most rapid and efficient in their field", and are deployed in several derivatives exchanges in Europe and North America.

The organization has a documented process, applying a waterfall lifecycle. They are divided into three groups: architecture, development, and quality control. Most of their projects are run with less than 10 staff members.

The organization's initial need was to understand and identify the factors behind the budget overruns on its larger projects, which should lead to improving the process. Unofficially, managers also needed to know the relative costs of their software process, since their customers mentioned several times the high cost of software projects. These needs were set as objectives for the evaluation. The management also required that interview effort be less than 60 staff-hours, including the evaluator and participants.

4.2. Specification of the Evaluation (Step 2)

The evaluation scope comprised the following axis:

1. Organizational units: restrain the evaluation to one specific business domain, representing projects for two major clients.
2. Software practices: within CMMI, consider all process areas from maturity levels 2 and 3, except Supplier Agreement Management (not applicable in this organization), Organizational Training (declared out of scope), and Decision Analysis and Resolution (we knew these practices were not applied).
3. Software projects: six (6) projects were to be evaluated and measured, three (3) projects considered as large projects by this organization (over 500 staff-days of effort), and three (3) projects of a regular size (less than 500 staff-days of effort). Projects were to be implemented in 2014 or 2015 to ensure that they represent the current state of the software process.

4.3. Design of the Evaluation (Step 3)

A detailed evaluation plan was established to describe the evaluation method, identify the required personnel, and plan for their availability and preparation. A total of 21 participants to interview were identified, with at least two participants per role. A questionnaire was developed to ensure that all the selected process areas would be covered. The detailed evaluation plan also linked appropriate process areas with each role.

The questionnaire aimed to assess the level of achievement of objectives from the selected CMMI–DEV process areas by the organizational software process, as applied through their projects. When findings were not sufficient or contradictory, projects artefacts were examined to help determine an achievement level, represented using a scale adapted from the SCAMPI method containing the following levels for every process area:

1. TA: Totally Achieved;
2. LA: Largely Achieved;
3. PA: Partially Achieved;
4. NA: Not Achieved;
5. NY: Not Yet [performed].

The evaluation plan initially proposed holding a kick-off session to present the evaluation, its objectives and scope, and other important information to all the participants, but the management decided not to hold the kick-off, replacing it by an introductory email. A two-page summary document was then written providing this information to the participants ahead of the interview sessions.

4.4. Interviewing the Project Participants and Reviewing Project Documentation (Step 4a)

The evaluation was realized by conducting interviews with the project participants. The interviews were performed as per the evaluation plan. However, we were able to interview 18 participants out of the 21 planned initially. Confidentiality of the interviews was assured by the evaluator and reinforced by the management. Interview effort was 50 staff-hours, complying with one of the management's needs.

4.5. Measuring the Software Functional Size and Process Efficiency (Step 4b)

The software functional size of the projects was measured using the COSMIC method while reviewing project documentation, namely the requirements documents. In one project, the ambiguities of requirements led the measurer to confirm measurement results by examining the source code. A certified and experienced measurer verified the measurement results to ensure accuracy for all six projects. Effort data was obtained from the organisation project database. Several analyses have been done with these data to meet management needs stated in 4.1. A small portion of analysis results are shown in 4.8.

4.6. Reviewing and Reporting Observations (Step 5)

The initial observations were subsequently presented to the participants for validation. This step aimed to ensure that the interpretation of their affirmations and that the confidentiality of the interviews was preserved. This step also helped validate that the final findings accurately reflected the applied software process.

4.7. Conclusion of the Evaluation (Step 6)

A final report was prepared, presenting the combined results from the qualitative and the quantitative evaluations. The observations related to the qualitative evaluation were grouped by the corresponding CMMI process area category. Other observations were derived from the functional size measurement activities, such as inconsistencies of the requirements engineering practices. For each group of observations, a set of recommendations was proposed in order to improve the software process.

4.8. Measurement Results and Outcomes

The results from the measurement (step 4b) are shown in Figure 4, where staff-effort has been compared with the functional size for all six projects within the evaluation scope.

The qualitative evaluation of the project with the highest relative effort per CFP showed that this project had been started then stopped three times before being completed, as well as having changed the key personnel such as the project manager, architect, and team leader, and particularly having unclear

requirements. These facts can explain its higher relative effort and can classify the project as an outlier. The participants described the project with the lowest relative effort per CFP as "a project that went exceptionally well", explaining the rare clarity of the requirements and exceptional availability of the customer. The other four projects were executed following the organisational standard process.

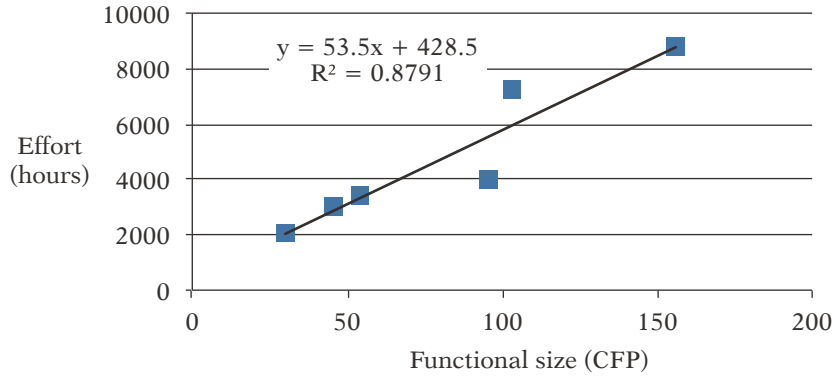

Figure 4. The Initial Productivity Model with the Functional Size and Effort from All Six Projects

Source: the authors' own work.

Considering the first two projects as exceptions, we were curious to analyse the effect of their suppression on the productivity model (see Figure 5). Despite having the data from only four projects, the correlation factor has the highest possible value of 1, which is unusual from our experienced viewpoint.

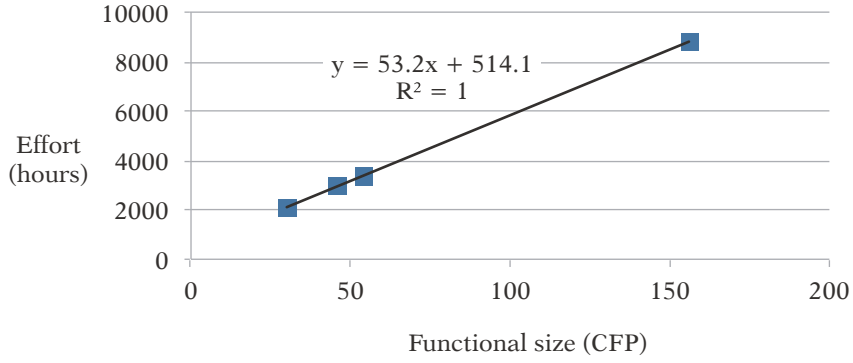

Figure 5. The Comparison of Functional Size and Effort for the Remaining Four Projects

Source: the authors' own work.

The management needed to understand budget overruns and the evaluation provided answers that an algorithmic method based on COSMIC function points seems more efficient than their traditional internal expert judgment estimation method. The correlation factor, despite having the data from only 6 projects, was sufficiently high for the management to adopt the measurement of software functional size, maintain the estimation model, and apply this model in the upcoming projects' estimation.

During the measurement and analysis step (4b), new process related findings were identified as a result of analysing the measurement and benchmarking results. As an example, the benchmarking data showed the productivity rate significantly higher than other organizations' in a similar context: a comparison with similar industry data showed that the variable cost was at least twice the median relative effort of their industry[31]. This led the evaluator to deepen the qualitative analysis of the process, revealing consistent waste within requirements engineering practices that could be eliminated with a small amount of effort. The confirmation of a higher relative cost led the management to seriously consider process improvement recommendations from the evaluation report, as it was linked to one of the evaluation objectives.

5. Discussion

This paper has discussed an adaptation of the PEM to combine qualitative and quantitative evaluation of the software process and the software requirements. CMMI is used as an input for the qualitative aspect while the COSMIC FSM method is used as an input for the quantitative aspect.

Several benefits have been observed from the field trial:

1. Measuring the functional size of projects implicitly provided a review of functional requirements quality, which would probably have been overlooked under a simple qualitative software process assessment.
2. Quantitative software process measurement enabled relations with some of the findings from the qualitative assessment while providing sound information to the management.

[31] S. Trudel, A. Abran, *Functional Size Measurement Quality Challenges for Inexperienced Measurer*, in: *Software Process and Product Measurement*, International Conferences IWSM 2009 and Mensura 2009 Amsterdam, 4–6.11.2009, proceedings vol. 5891, Springer Science & Business Media, pp. 157–169.

3. A preliminary estimation model was obtained, which seems more reliable than the current organization's technique to estimate projects.
4. The evaluation required effort was of an acceptable level for a small organisation.

These benefits are encouraging in pursuing the combination of qualitative and quantitative evaluations.

6. Future Work

ISO/IEC 14598–5 has since been replaced by ISO/IEC 25000[32], known as the Systems and Software Quality Requirements and Evaluation (SQuaRE). Adaptation of the PEM to SQuaRE would be required to keep it up to date with the current standard. We also intend to formalize adaptations of inputs to the evaluation method where an evaluator could use a quality model to assess a specific aspect of the product quality, whether the model contains quality attributes for the software code or is using a functional size method to characterize quality aspects of the software requirements, as used in this paper. As part of this work, provisions can be made to ensure the compliance of the resulting method with ISO/IEC 15504[33]. Moreover, it could be beneficial to widen the scope of measurement within the method, not limiting this measurement to the functional size by having a larger coverage. We also plan to include customer satisfaction assessment results to obtain a complete, 360 degrees type of assessment of the software process.

It could be useful to correlate productivity measurement with an evaluated CMMI maturity level. But measuring that maturity level is costly and would defeat the purpose of keeping the method affordable to small organisations.

[32] ISO/IEC 25000, *Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE*, ISO 2014, http://www.iso.org

[33] ISO/IEC 15504–4, *Information Technology – Process Assessment – Part 4: Guidance on Use for Process Improvement and Process Capability Determination*, ISO 2004, http://www.iso.org

References

Abran A., *Software Metrics and Software Metrology*, Wiley 2010.

Abran A. et al., *The COSMIC Functional Size Measurement Method, Version 4.0.1*, Measurement Manual, The COSMIC Group 2015, http://www.cosmicon.com/dl_manager4.asp?id=532

Abran A. et al., *Guideline for Assuring the Accuracy of Measurements, Version 1.0*, The COSMIC Group 2011, http://www.cosmicon.com

Chrissis M.B., Konrad M., Shrum S., *CMMI Guidelines for Process Integration and Product Improvement*, Addison-Wesley 2003.

The CMMI Product Team, *Capability Maturity Model Integration © for Development, Version 1.3*, The CMMI Institute 2010, CMU/SEI-2010-TR-033.

The CMMI Institute, *Standard CMMI Appraisal Method for Process Improvement (SCAMPI) Version 1.3b: Method Definition Document for SCAMPI A, B, and C*, Technical Report, The CMMI Institute 2014.

Desharnais J.-M., Abran A., *Assessment of the Quality of Functional User Requirements Documentation Using Criteria Derived from a Measurement with COSMIC–ISO 19761*, International Workshop on Software Measurement – IWSM 2010, Stuttgart, November 2010, pp. 481–496.

Garcia-Miller S., *Lessons Learned from Adopting CMMI in Small Organizations*, Software Engineering Institute 2005, http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=19950

International Software Benchmarking Standards Group (ISBSG), *Development and Enhancement Data Set 2015, Release 13*, Obtained from http://www.isbsg.org

ISO/IEC 14598–5, *Information Technology – Software Product Evaluation – Part 5: Process for Evaluators*, ISO 1998.

ISO/IEC 14143–1, *Information Technology – Software Measurement – Functional Size Measurement – Part 1: Definition of Concepts*, ISO 2007.

ISO/IEC 20926, *Software and Systems Engineering – Software Measurement – IFPUG Functional Size Measurement Method 2009*, ISO 2009.

ISO/IEC 24570, *Software Engineering – NESMA Functional Size Measurement Method Version 2.1 – Definitions and Counting Guidelines for the Application of Function Point Analysis*, ISO 2005.

ISO/IEC 29881, *Information Technology – Systems and Software Engineering – FiSMA 1.1 Functional Size Measurement Method*, ISO 2010.

ISO/IEC 20968, *Software Engineering – Mk II Function Point Analysis – Counting Practices Manual*, ISO 2002.

ISO/IEC 19761, *Software Engineering – COSMIC: A Functional Size Measurement Method*, ISO 2011.

ISO/IEC 29110–1, *Software Engineering – Lifecycle Profiles for Very Small Entities (VSEs) – Part 1: Overview*, ISO 2011.

ISO/IEC 29110–2, *Software Engineering – Lifecycle Profiles for Very Small Entities (VSEs) – Part 2: Framework and Taxonomy*, ISO 2011.

ISO/IEC 29110–3, *Software Engineering – Lifecycle Profiles for Very Small Entities (VSEs) – Part 3: Assessment Guide*, ISO 2011.

ISO/IEC 15504–3, *Information Technology – Process Assessment – Part 3: Guidance on Performing an Assessment*, ISO 2004.

ISO/IEC 25000, *Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE*, ISO 2014, http://www.iso.org

ISO/IEC 15504–4, *Information Technology – Process Assessment – Part 4: Guidance on Use for Process Improvement and Process Capability Determination*, ISO 2004, http://www.iso.org

Marciniak J.J., *Software Engineering. A Historical Perspective*, Encyclopedia of Software Engineering, Wiley 2002.

Nguyen F., *Estimated Cost for SCAMPI Class A, B, or C Appraisal*, 2010, http://cmmirocks.ning.com/forum/topics/estimated-cost-for-scampi (08/07/2015).

Paulk M.C., *SW-CMM V1.1*, Software Engineering Institute 1992.

Trudel S., *Using the COSMIC Functional Size Measurement Method (ISO 19761) as a Software Requirements Improvement Mechanism*, Thesis, École de Technologie Supérieure 2012.

Trudel S., Abran A., *Functional Size Measurement Quality Challenges for Inexperienced Measurer*, in: *Software Process and Product Measurement*, International Conferences IWSM 2009 and Mensura 2009 Amsterdam, 4–6.11.2009, proceedings vol. 5891, Springer Science & Business Media, pp. 157–169.

Trudel S., Lavoie J.M., Paré M.C., *Suryn W., PEM: The Small Company-dedicated Software Process Quality Evaluation Method Combining CMMI℠ and ISO/IEC 14598*, "Software Quality Journal" 2006, vol. 14, no. 1, pp. 7–23.

Ungan E., Demirörs O., Top Ö.Ö., Özkan B., *An Experimental Study on the Reliability of COSMIC Measurement Results*, in: *Software Process and Product Measurement*, Springer, Berlin Heidelberg 2009, pp. 321–336.