Hindawi Publishing Corporation
Advances in Software Engineering
Volume 2011, Article ID 495232, 7 pages
doi:10.1155/2011/495232

Research Article

ReMoFP: A Tool for Counting Function Points from UML Requirement Models

Vitor A. Batista,[1] Daniela C. C. Peixoto,[2] Eduardo P. Borges,[1] Wilson Pádua,[1] Rodolfo F. Resende,[2] and Clarindo Isaías P. S. Pádua[1]

[1] *Synergia, Department of Computer Science, Federal University of Minas Gerais, Belo Horizonte, MG 31270-010, Brazil*
[2] *Department of Computer Science, Federal University of Minas Gerais, Belo Horizonte, MG 31270-010, Brazil*

Correspondence should be addressed to Wilson Pádua, wppf@ieee.org

Received 20 December 2010; Accepted 16 February 2011

Academic Editor: Letha Hughes Etzkorn

Function Point Analysis (FPA) is a widely used technique for measuring software size. It measures software functionality from the user's perspective, usually based on a requirements description. In many software processes, these requirements are represented by UML models. Although there have been attempts to automate the measurement process, FPA counting requires a considerable amount of interpretation which, to be reliable, should be made by experts. On the other hand, fully manual counting methods usually fail to keep synchronized with the requirements model, since requirements frequently change during the development cycle. This paper describes an approach for counting FPA and a compliant tool. This approach makes use of UML requirement models. The tool, called ReMoFP (*Requirement Model Function Point counter*), leaves all the counting decisions to the analyst, but supports him by ensuring consistency with the requirements represented in the models. The ReMoFP was developed by a software development laboratory in Brazil, and helped it to improve counting productivity, consistency, and maintainability.

1. Introduction

Due to complexity and size issues, developing high-quality, cost-effective software is a huge challenge, usually constrained by short schedules. To achieve effective and realistic planning for software development projects, project managers should estimate correctly many software metrics, such as those that reflect size, effort, quality, and risks. Size estimates are used both as input and normalization factor for other relevant metrics.

Several well-known methods are available for size measurement; counting LOC (Lines of Code) and Function Point Analysis (FPA) are widely used. This work focuses on FPA, a method that estimates functional complexity, seen from the user viewpoint [1]. Early approaches to FPA were proposed by Albrecht [2]; it has become widely used, despite some weaknesses [3–5]. The method has been maintained and updated by IFPUG (the International Function Point Users Group) [1], and was adopted as an ISO and IEEE standard [6]. FPA details are given in many sources, and we omit them, for the sake of brevity.

In recent years, Object-Oriented (OO) technology has emerged as a dominant practice in Software Engineering domain. Therefore, it is useful to match traditional Function Point (FP) measurement to new OO approaches [7], including models based on Unified Modeling Language (UML) [8].

In this work, we defined a tool, called ReMoFP (*Requirement Model Function Point counter*), to help FP experts during FPA counting based on OO requirement specifications. It is not our purpose to fully automate the FP measurement. A fully automatic tool does not seem to be feasible, mainly because FPA counting requires some human judgment [9]. Coherently with FPA concepts, we focus on the users' view of the system, and in estimating size early in the software development life cycle.

ReMoFP is being used in a software development laboratory called Synergia. The main motivation drivers for the development of ReMoFP were: improvement of the counting productivity, support for some validation for the counting results, support for monitoring the size evolution during the requirements elicitation, and easing the updates of size measurements after requirement changes, quite

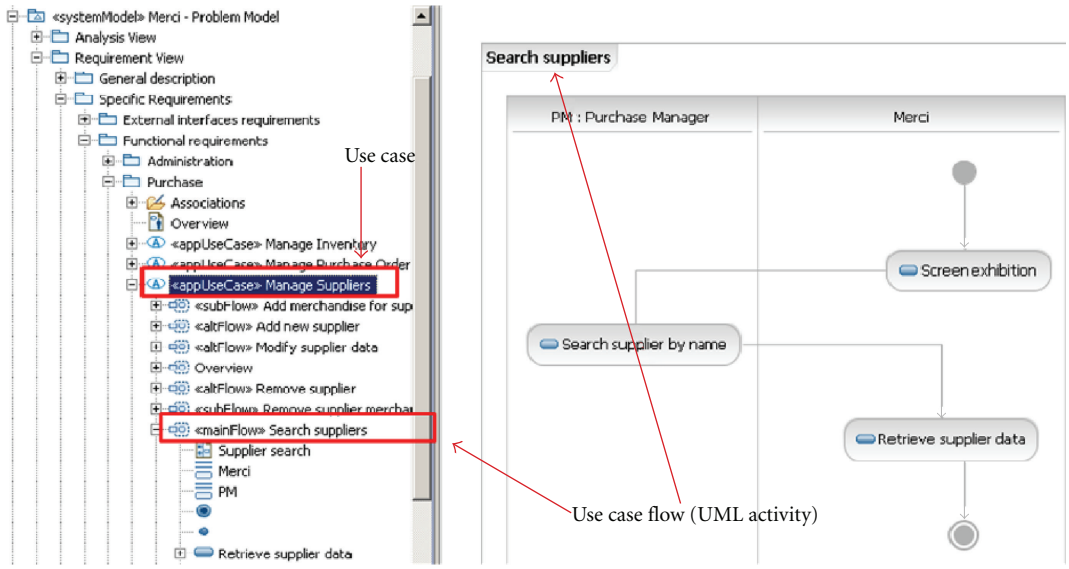

Figure 1: Use case flow modeled as an Activity.

usual with our customers. The first application of ReMoFP more than doubled the counting productivity. These results matched the organization's expectations, and this tool was incorporated as an official tool of its software development process.

The remainder of this paper is structured as follows. Section 2 provides a brief description of the authors' organization and the concepts used for problem modeling, in its software development process. Section 3 describes how our tool supports function point counting, using the model. Section 4 discusses benefits. Section 5 describes related work, and Section 6 presents the final remarks.

2. Organization, Process, and Problem Modeling

Our work was carried inside Synergia, a laboratory for software and systems engineering, hosted by the Computer Science Department at the Federal University of Minas Gerais. Synergia has about 70 people in its staff, composed by undergraduate, graduate students, and nonstudent graduates, most with a computer science background.

Synergia adopts a software development process called Praxis-Synergia [10, 11]. Both processes (the standard, educational base process and its tailored industrial version) have been recently upgraded to version 3 [12]. One of the major enhancements in this new version is the way to represent the user requirements in a UML model. This model is called Problem model; despite some similarity with RUP's Analysis Model [13], its structure closely follows the IEEE-830 standard for Requirements Specification [14]. The Problem model is divided into two main views: Requirement view and Analysis view.

The Requirement view describes the desired product from the user viewpoint, representing desired functions as *Use Cases* (stereotyped as "appUseCase." Each *Use Case* behavior is described by one or more flows of events, classified by the Process in three types.

(i) *Main flow*. Represents the most usual sequence of events in the use case execution.

(ii) *Alternate flows*. Represent less usual, optional or exceptional sequences of execution.

(iii) *Subflows*. Represent sequence of steps invoked in the other kinds of flows.

Each of these types of flows is modeled as a stereotyped *Activity*, with stereotypes "mainFlow," "altFlow," and "subFlow," respectively. Figure 1 shows a use case with several flows (Activities) and a diagram detailing its Main flow.

The Analysis view describes the desired product from the developer viewpoint, but still in the problem-domain; it models concepts, procedures, and interfaces as classes, and their interactions by stereotyped UML *Collaborations*. These realize, in conceptual terms, the functionality described by the use cases, using *UML Interactions*, usually represented by *Sequence Diagrams*. As a general modeling rule, each use case flow is realized by an interaction with the same stereotype.

Figure 2 shows an application use case ("appUseCase"), connected to an analysis collaboration (stereotyped as "analysisCollaboration", to distinguish them from the design or test collaborations of the Solution model) by a realization relationship. Collaboration attributes represent instances of the participating analysis classes.

3. FP Counting Using the Problem Model

Based on the concepts presented in previous section, we now present our proposal to count FPs directly from the Praxis Problem model. The foundation of this proposal is the broad use of UML extension mechanisms, such as *Stereotypes* and formal *Constraints* written in OCL [15]. We defined

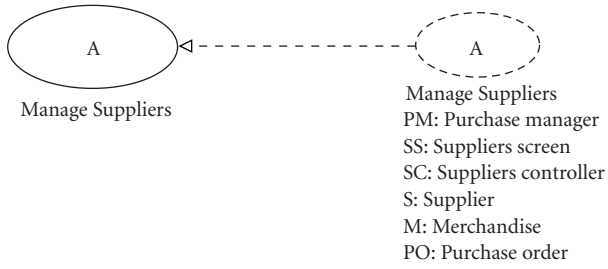

Figure 2: Use case realization.

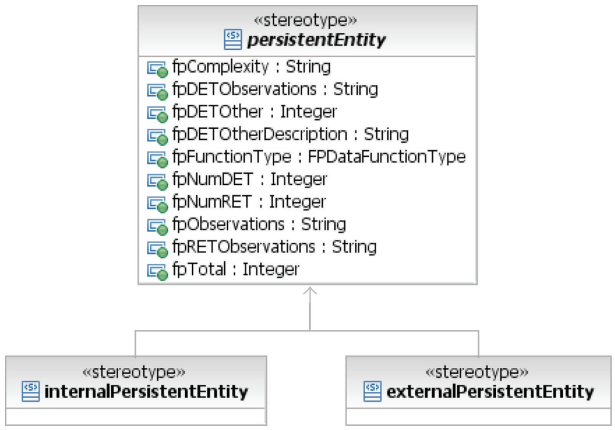

Figure 3: Stereotypes for counting Data Functions.

Stereotypes with properties (metamodel attributes) that hold FP counting information and associated OCL *Constraints*. These *Stereotypes* are encapsulated by an UML *Profile*, which is deployed to the IBM Rational Software Architect, in the way required by that tool.

3.1. Counting Data Functions. The Problem model uses UML classes stereotyped with "persistentEntity" to represent data that the desired application should maintain. To keep the FP counting information in the model, attributes were added to this stereotype. Since FP has two types of Data Functions, we also created two new stereotypes to differentiate between them.

(1) ILFs (Internal Logical Files) correspond to data that are maintained by the application; they are represented by classes with stereotype "internalPersistentEntity."

(2) EIF (External Interface Files) correspond to data that are maintained by other applications, but are read by the desired application; they are represented by classes with stereotype "externalPersistentEntity."

Figure 3 shows the metamodel for those stereotypes. "persistentEntity" becomes an abstract stereotypes, holding common properties that are useful in the FP counting method. Their purpose is described in Table 1.

The mapping between the persistent entities from the Problem model and the Data Functions is neither direct nor one-to-one. Automating this mapping is not simple; probably, it would require more complex modeling of persistent data. In our proposal, an FP counting specialist must do it manually.

Based on Problem Model, the FP counting specialist must decide how persistent classes should be grouped, in order to be mapped onto Data Functions. Figure 4 illustrates a model with 4 classes and their proposed grouping (the specialist suggested 2 ILFs, based on this diagram). In Data Functions which group more than one class, the specialist should choose one of them to stand for the group. In our example, Purchase Order was chosen as the main class of its group. After grouping all persistent classes and mapping the groups onto Data Functions, the attributes presented in Table 1 should be filled in each main class.

To support this procedure, we developed ReMoFP, a plugin for the UML tool Rational Software Architect (RSA), used to write the Problem model. ReMoFP is implemented as a Java code based on the tool extension API. It eases the filling of the stereotype properties and automatically calculates the complexity and total function points for each Data Function. Figure 5 shows an example of usage of ReMoFP to count the ILFs Purchase Order, mapped to classes Purchase Order and Purchase Order Item.

In the example shown in Figure 5, the list of grouped classes is assembled by the specialist, selected from the model classes which are not yet assigned to Data Functions. Based on this list, the number of RETs is manually informed. After that, the user of ReMoFP could select any attribute of the selected classes, to count them as DETs. Additionally, other DETs, not explicitly derived from attributes, could be manually included in the DET counting. Finally, the specialist carries out a command to calculate and store the counting summary for this Data Function. Note that, in this example, we selected inherited attributes from abstract *Class* Merchandise Item as DETs.

To add value to ReMoFP, we took advantage of OCL, in order to create several UML *Constraints* associated with our stereotypes. When model validation is executed in RSA, these *Constraints* are checked and their violation generates error messages. Examples of the use of *Constraints* are as follows.

(i) To validate DET counting, ReMoFP checks whether the sum of selected DETs and informed DETs is equal to the stored number of DETs, when calculating complexity. This prevents inconsistencies in FP counting caused by removal of class attributes.

(ii) When FP counting specialist informs a value in "Other DET" field (fpDETOther), ReMoFP checks if description (fpDETDescription) is also filled.

(iii) If any class is selected as part of a Data Function, ReMoFP does not allow that this class be marked as another Data Function.

(iv) A class that represents an ILF (attribute fpFunctionType = ILF) should use an "internalPersistentEntity" stereotype, instead of "externalPersistentEntity."

Table 1: Stereotype attributes for record Data Function counting information.

Attribute	Description of purpose
fpFunctionType	Data Function type: ILFs, ELF or None.
fpComplexity	Data Function Complexity: High, Average, or Low. Calculated based on other information supplied by the analyst.
fpNumRET	Record Element Types (RET) counting.
fpNumDET	Data Element Type (DET) counting.
fpTotal	FP Total counting.
fpDET	List of DET considered in counting process. Contains a list of analysis class attributes (UML Properties). It does not appear in Figure 3 because of limitations of the used tool.
fpDETObservations	Observations written by the analyst, concerning the DET counting.
fpDETOther	Could be used to manually record DET that are considered in counting but are not analysis class attributes, such as associations between them.
fpDETDescription	Descriptive documentation of items informed in fpDETOther field.
fpRET	List of RETs considered in counting. Contains a list of analysis classes. It does not appear in Figure 3 because of limitations of the used tool.
fpRETObservations	Observations about RET counting.

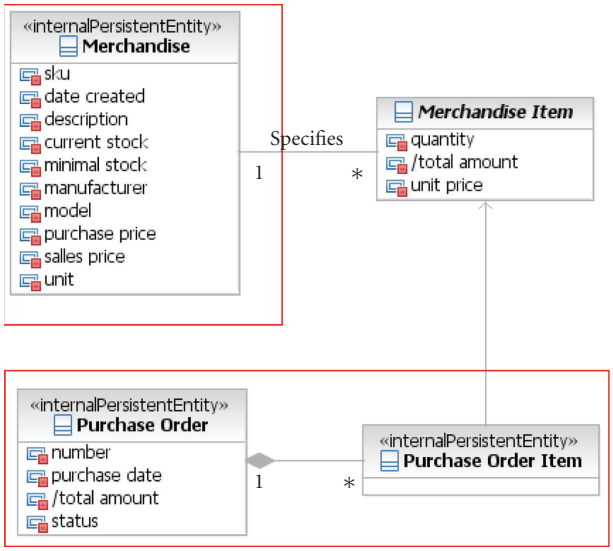

Figure 4: Mapping between Classes and Data Functions.

3.2. Counting Transactional Functions. To support Transactional Functions counting, the same strategy presented in the previous section was used. The FP counting specialist maps the use case flows (modeled as activities, as discussed in Section 2) onto Transactional Functions. Often, each flow is mapped to one Transactional Function, but this is not mandatory, since this depends on modeling choices. Therefore, the full automation of this mapping would be very hard, as it happens with the mapping of Data Functions.

The abstract stereotype "eventFlow" was created to represent all types of event flows. Figure 6 shows that it is specialized by the three stereotypes that represent use case flows.

We do not present them here with the same level of detail as in the previous section, since the rationale is essentially the same. We simply emphasize the differences regarding the counting of FTRs (File Types Referenced, as opposed to RETs in Data Functions) and DETs.

To select the list of Data Functions considered as FTRs in a Transactional Function, ReMoFP offers the specialist the list of all classes defined as Data Functions, and whose instances participate in the collaboration that realizes the use case. For instance, if the specialist defines the main flow of the use case Manage Suppliers as an External Inquiry (EQ), ReMoFP will list Purchase Order and Merchandise as candidates to FTR (see Figures 2 and 4).

In order to count DETs, ReMoFP shows the list of all attributes from boundary classes (those that represent user interfaces) and whose instances participate in the collaboration that realizes the use case. These user boundaries have also predefined stereotypes: "screenBoundary" classes represent product screens, "reportBoundary" classes represent product reports, and "softwareBoundary" classes represent interfaces with other products, outside the application boundary. Here, as in the DET counting for Data Functions, additional DETs could be manually added to the selected DETs.

Another difference between Data and Transactional Function counts, in our tool, is the ability to copy data from one Transactional Function to another. This is very useful since many Transactional Functions share the same FTR and DET information, for instance, record insert and update, which are both EE-type functions.

For Transactional Functions, we also created several OCL constraints to prevent defects in FP counting, in order to reduce rework.

4. Benefits

The first project that adopted ReMoFP, referred here as New Project, was a project with the size of 2445 FP distributed into 101 use cases. When ReMoFP became stable, we started measuring the effort for counting FP. We recorded 31 use cases data, summing up 882 FP of the 2445 FP (see Table 2).

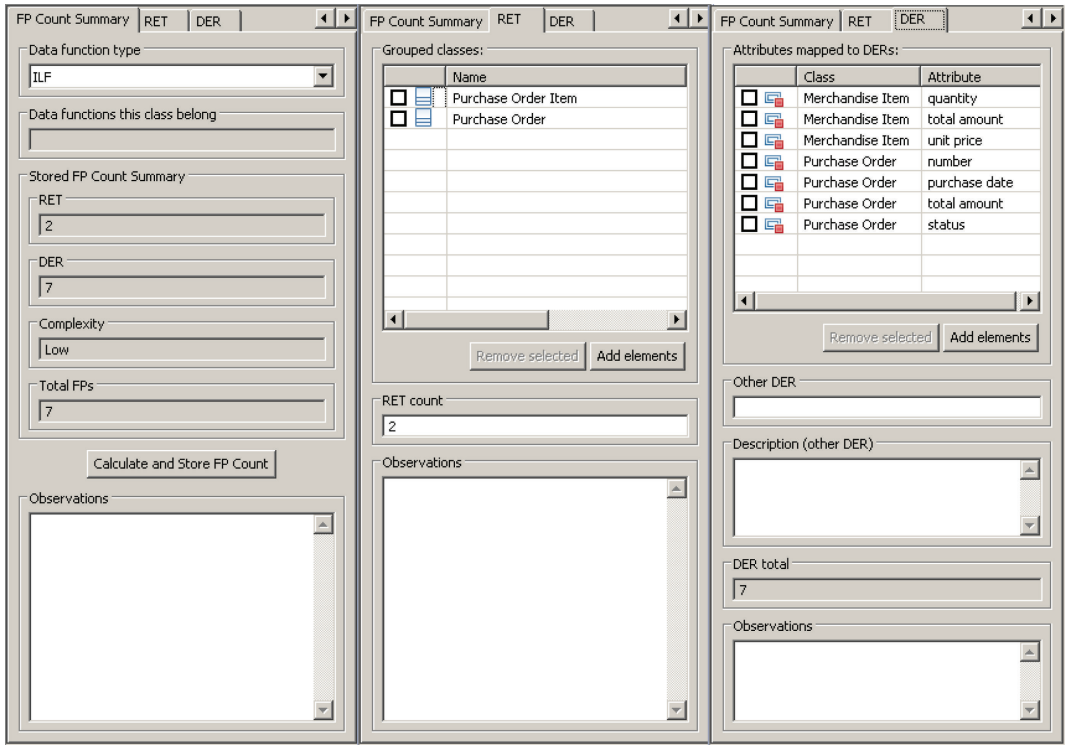

Figure 5: ReMoFP tool developed to support the FP counting procedure.

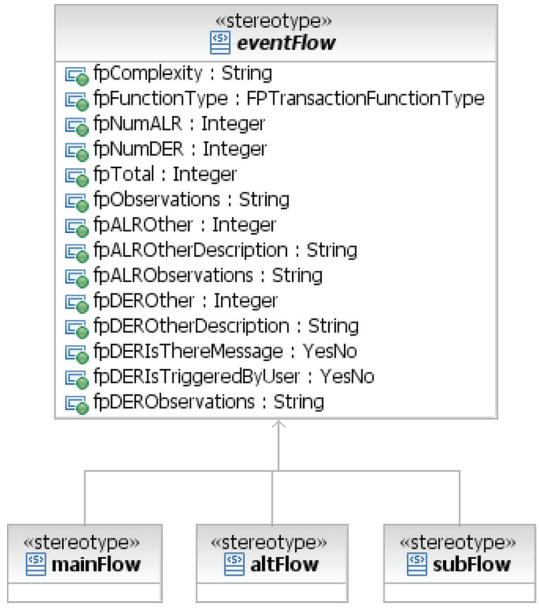

Figure 6: Stereotypes for counting Transactional Functions.

In the New Project, each Requirement team member (which is not necessarily an FP counting certified specialist) was responsible for counting both Data Functions and Transactional Functions of each use case that he/she specified. The counting was carried out during the use case specification. In this way, it was possible to monitor the product size in "real time" and negotiate changes with the client if needed, since the customer had a predefined scope size target.

Table 2: Productivity comparison between FP count with and without ReMoFP.

Project	# Use cases	# FP counted	Total effort (h)	Productivity (PF/h)
Old	138	4673	258.23	18.10
New	31	882	23.44	37.63

The quality of the FP counting was verified twice. First, each use case FP size was verified during the requirement inspections by other Requirement team member. In addition, after the conclusion of the specification, an IFPUG-certified specialist reviewed again all the counting data. This procedure was executed to avoid counting inconsistencies.

The results collected from the New Project are summarized and compared to another project, referred here as Old Project. Both projects present similar characteristics (Web-based, Team maturity and size, etc.) and adopted the IFPUG counting procedures. Despite the reduced number of New Project use cases, we observed a substantial productivity increase (107%).

Other important results were achieved while executing the review process. First, the certified specialist reviewed the estimation and detected a small deviation error (less than 2% in total FP count). This reliability of counting procedures executed by noncertified specialists is probably due to the automated verification carried out by OCL constraints. In addition, the whole revision process had a productivity of 54.61 PF/h.

In addition to the quantitative results presented above, we also identified the following qualitative benefits.

(i) ReMoFP helps to keep the product functional size (FP count) up-to-date. This is a major concern to Synergia, since most of its contracts are made based on FP counts, and this information is crucial to estimate future projects. Counting FP directly in UML Model minimizes inconsistencies caused by unavoidable requirements changes. Synergia has already developed 8 large projects, and so far it has not been possible to keep an updated FP counting measurement of those projects, after the counting made based on the first version of Problem Model. The New Project was the first to keep FP count up-to-date.

(ii) It makes possible to monitor the project size evolution while its requirements are still been elicited. A graph that shows the evolution of functional size could be easily and automatically generated from the requirements database; this gives important information to project's stakeholders, as a basis to negotiate project scope.

(iii) More reliability in FP counting is achieved, since OCL *constraints* automate most verification items, which, otherwise, would undergo less reliable human checks.

(iv) It makes possible to deliver the model with the function point counts to the client. Since ReMoFP uses standard UML extension mechanisms, it is not necessary to deliver the tool.

5. Related Work

IFPUG has a certification program for tools that support FP counting. This program classifies counting tools into three types (http://www.ifpug.org/certification/software.htm). In a Type 1 tool, the user counts manually, and the software is used just for calculations. In Type 2 tools, counting is performed in an interactive way: the user answers some questions, and the system makes counting decisions based on the answers provided. In Type 3 tools, the system performs automated counting, using multiple sources of information such as the application software, descriptions from software design and development tools, and the database management system. Currently, there are three recognized Type 1 tools and only one Type 2 tool. No Type 3 tool has achieved official certification.

Examples of Type 1 tool include Function Point Workbench (http://www.charismatek.com/) and PQMPlus (http://www.qpmg.com/). Those tools consist mainly of independent applications that provide support for counting. We consider our tool as a Type 1 tool, since it just supports counting function points for requirements described by a UML model. However our tool goes somewhat beyond Type 1, since it provides validations through OCL *Constraints*.

Some tools apply FPA to object-oriented models [4, 9, 16, 17]. However, a large amount of these tools deal with design models, rather than requirements models [9]. Our approach derives FP directly from a UML-based requirement model, conforming to the principle that function points must measure the problem size, not the solution size.

Similarly to our explicit mapping for requirement models, other works also propose rules for mapping OO models to FPA models. Harput et al. [9] propose a semiautomatic model transformation from OO requirements models to FPA models, based on heuristic rules to be applied by an FPA expert. This is an example of Type 2 tool. The method uses sequence diagrams and use cases for transactional function types and class diagrams for data function types. HARPUT's work presents some rules for mapping classes or groups of classes to Data Functions. The same mapping could be done in our work, but it involves the judgment of an FPA expert. Another difference between HARPUT and our work consists of the counting rules for Transactional Functions. Our work focuses on the use case flows rather than the messages from sequence diagrams, as in HARPURT's work. Harput et al. also propose a mapping from nonfunctional requirements to general system characteristics as a guiding reference. This aspect does not apply to our work because we use the COCOMO [18] model as an estimation tool, and it suggests the use of Unadjusted Function Points (UFP). Besides, Adjusted Function Points are not standardized by ISO, and the Praxis process has other means to compute the cost of nonfunctional requirements.

Uemura et al. [4] describe a system for automatically counting FP from a requirement/design specification based on UML diagrams. This is an example of a Type 3 tool. They propose some rules that are used to extract the information from class and sequence diagrams. The rules make use of sequence diagrams and class diagrams for counting Data Functions. The complexity of a Data Function is judged based on the attributes of the corresponding selected class. For counting Transactional Functions, the rules propose to identify patterns in the sequence diagrams. For example, if an actor sends a message with arguments to (an instance of a class that represents) a Data Function, it is considered as (representative of) an External Input.

Another Type 3 example is presented by Caldiera et al. [16]. They propose an adaptation of traditional function points, called Object-Oriented Function Points (OOFP), to allow the measurement of object-oriented analysis and design specifications. A tool has been developed to automate the counting method. Our work focuses on the traditional definition of FP, the only one that is widely accepted in our commercial environment.

Cantone et al. [17] propose conversion rules to apply FPA to UML. They also present a tool implemented in the IBM Rational Rose environment. The emphasis is on UML sequence diagrams and rules applied to high-level design documents. This differs from our work, which uses requirement specifications represented in the Problem model.

6. Conclusion

In this paper, we presented a tool, called ReMoFP, that supports FP experts during standard FP counting. The FP counting is carried out using an OO problem model based on the UML.

This tool was closely monitored in one project, and the results were satisfactory. The FP counting reliability and productivity increased, control of the size evolution was eased, and the visibility of the product size was improved, both for the client and the development team. The results lived up to the organization's expectations, and this tool was incorporated as official tool of Praxis-Synergia.

We are planning to collect additional data, regarding the productivity and the facility to update the FP counting. We also expect to refine and implement new verification constraints.

References

[1] IFPUG, *IFPUG Counting Practices Manual—Release. 4.2.1*, International Function Point Users Group, Westerville, Ohio, USA, 2005.

[2] A. J. Albrecht, *Function Point Analysis*, John Wiley & Sons, New York, NY, USA, 1994.

[3] B. Kitchenham, "Counterpoint: the problem with function points," *IEEE Software*, vol. 14, no. 2, p. 29, 1997.

[4] T. Uemura, S. Kusumoto, and K. Inoue, "Function point measurement tool for UML design specification," in *Proceedings of the 6th International Software Metrics Symposium*, pp. 62–69, November 1999.

[5] D. Gupta, S. J. Kaushal, and M. Sadiq, "Software estimation tool based on three -layer model for software engineering metrics," in *Proceedings of the 4th IEEE International Conference on Management of Innovation and Technology (ICMIT '08)*, pp. 623–628, September 2008.

[6] IEEE, "IEEE std 14143.1-2000: implementation note for IEEE adoption of ISO/IEC 14143-1:1998 Information Technology—Software Measurement—Functional Size Measurement—part 1: definition of concepts," Tech. Rep., Software Engineering, IEEE, New York, NY, USA, 2003.

[7] J. Ram and S. V. G. K. Raju, "Object oriented design function points," in *Proceedings of the 1st Asia-Pacific Conference on Quality Software (APAQS '00)*, pp. 121–126, 2000.

[8] OMG, *Unified Modeling Language Specification, Version 2.0*, Object Management Group (OMG), 2005.

[9] V. Harput, H. Kaindl, and S. Kramer, "Extending function point analysis to object-oriented requirements specifications," in *Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS '05)*, pp. 361–370.

[10] B. Pimentel, W. P. P. Filho, C. Pádua, and F. T. Machado, "Synergia: a software engineering laboratory to bridge the gap between university and industry," in *Proceedings of the International Workshop on Summit on Software Engineering Education*, pp. 21–24, ACM, New York, NY, USA, 2006.

[11] W. de Pádua, "Quality gates in use-case driven development," in *Proceedings of the International Workshop on Software Quality (WoSQ '06)*, pp. 33–38, ACM Press, New York, NY, USA, 2006.

[12] W. de Pádua, *Engenharia de Software: Fundamentos, Métodos e Padrões*, LTC, 3rd edition, 2008.

[13] P. Kruchten, *The Rational Unified Process: An Introduction*, Addison-Wesley, Reading, Mass, USA, 3rd edition, 2003.

[14] IEEE, "IEEE recommended practice for software requirements specifications," Tech. Rep. IEEE Std 830-1998, 1998.

[15] OMG, *Object Constraint Language 2.0 Specification*, Object Management Group (OMG), 2005.

[16] G. Caldiera, G. Antoniol, R. Fiutem, and C. Lokan, "Definition and experimental evaluation of function points for object-oriented systems," in *Proceedings of the 5th International Software Metrics Symposium*, pp. 167–178, November 1998.

[17] G. Cantone, D. Pace, and G. Calavaro, "Applying function point to unified modeling language: conversion model and pilot study," in *Proceedings of the 10th International Symposium on Software Metrics (METRICS '04)*, pp. 280–291, September 2004.

[18] B. W. Boehm, C. Abts, A. W. Brown et al., *Software Cost Estimation with COCOMO II*, Prentice Hall, New York, NY, USA, 2000.