A UML Reuse Framework and Tool for Requirements Engineering

Vitor A. Batista, Daniela C. C. Peixoto, Thiago R. V. Anjos, Wilson Pádua

Synergia – Computer Science Department
UFMG – Federal University of Minas Gerais
Belo Horizonte, Brazil

{vitor, cascini, trva, wilson}@dcc.ufmg.br

Abstract. Requirement Engineering (RE) activities are critical by nature and mostly manual. Some automated support for tasks helps requirements engineers to reduce manual labor and, consequently, reduce defects rates and increase reuse and motivation. In this paper, we introduce a UML framework and tool support which automates part of the RE process. Using UML stereotypes as the core of this solution, we created a set of integrated tools composed by: (1) a reusable framework that models RE behavior patterns that are typically present in information system projects; (2) a function that allows the reuse of information provided by entity modeling; (3) a tool that automates the generation of application prototypes; (4) a tool for counting IFPUG Function Points; and (5)a tool that analyzes specific types of defects. Our findings indicate that the framework and the automated support are effective at RE modeling and review. In addition, they increase motivation and promote team engagement, through elimination of repetitive activities.

Keywords: Requirements Engineering, UML framework

1 Introduction

Requirement Engineering (RE) is a very labor-intensive and critical activity. It is manual by nature, since requirements are elicited by analysts from users through workshops and interviews, and recorded in informal or semi-formal specifications[1]. It is critical because its defects inevitably lead to later problems in design and implementation, whose repair is, usually, expensive and difficult. Many studies have shown that requirements with poor quality are a major cause of project failures [2],[3],[4].

One way to help analysts is to provide some integrated and automated support for the RE tasks, in order to reduce manual labor and ease early detection of errors. Since RE is as a naturally labor-intensive process, only a few tasks may be significantly automated. However, their contribution to decrease rework and increase productivity can be substantial. Another way to improve RE support is to provide reusable model elements, with appropriate guidance.

In this work, we present the adoption and improvement of an RE sub-process in a software development organization, Synergia [5]. The RE sub-process is embedded in a software development process called Praxis-Synergia, one of the processes used at Synergia. This is a professional version of Praxis, an educational software development process [6]. A detailed description of Praxis-Synergia requirements modeling can be found in Batista et al. [7].

Synergia[5] is a laboratory for software and system engineering, housed in the Computer Science Department, at Federal University of Minas Gerais, Brazil. It provides software, systems, training and consulting solutions for Brazilian public agencies and private companies. Though internally organized as a traditional software development company, it retains academic characteristics, such as the participation of university students and faculty. Its goal is to provide high-quality IT services, while imparting professional experience and financial resources to its academic.

Here, we discuss some challenges faced and solutions proposed during this improvement effort towards an automated and integrated approach to RE. Our main contribution lies in providing a UML reuse framework and a set of automated tools to support RE activities. This reuse framework is intended to standardize modeling of the most common software transactions in information systems, while remaining flexible enough to let analysts to extend it when needed. It is expected that this approach should improve model comprehension and reuse, reduce errors and increase productivity throughout the software development cycle.

The article is structured as follows. Section 2 introduces basic concepts of Praxis and the problems faced by requirement analysts. Section 3 presents our UML framework and the proposed tools. Section 4 provides a discussion of the results and open issues. Section 5 presents the related work. Section 6 concludes and presents future work.

2 Background

Praxis [6] is a model-based, use-case-driven process. In its prescribed development sequence, a project is divided in iterations, where each iteration implements one or more functional requirements, modeled as use cases and use case scenarios.

In Praxis, the user requirements are represented in a UML model, called Problem model. Despite some similarity with RUP's Analysis Model [8], its structure closely follows the IEEE-830 standard for Requirements Specification [9].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 UML use cases. Each use case behavior is described by one or more scenarios. Scenarios may be modeled as UML activities, or described by text. The former is more adequate for complex interactions; the latter may be convenient for simpler ones.

The Analysis view describes the desired product from the developer viewpoint, but still in the problem-domain. This view models concepts, data, procedures and application external interfaces as classes, and their interactions by UML collaborations.

The adoption of Praxis at Synergia faced some problems during projects execution:

- User interfaces are prototyped in the Requirements view and also modeled as UML Classes in the Analysis view. Keeping both artifacts consistent during project is hard and error prone (25% of the defects were found in our requirements reviews).
- Since Praxis does not offer guidelines to model requirements, each use case is modeled from scratch, according to the experience of each analyst. Common transactions, such as CRUDs (Create-Retrieve-Update-Delete), are often modeled quite differently by different analysts, and this is a serious impediment to reuse.
- Praxis uses a large subset of UML elements, and much information needs to be recorded in stereotype properties (tagged values). This results in large checklists, reviews effort and subsequent rework.
- Standard Praxis records IFPUG Function Points (FP)[1] counting in spreadsheets of the MS Excel. The Synergia developers found somewhat difficult to use those. Monitoring size evolution, by recording continuously updated FP counts, was an error-prone manual procedure.

Given such problems faced at Synergia, we proposed a set of improvements in Praxis-Synergia, in order to overcome them. In next section we present our proposals.

3 The UML Framework and Automated Tools

The solution we proposed includes a set of UML tools which automate common RE activities, and ease the specification of common user interactions in a typical information system.

3.1 The UML Profile

The core of our automated solution is a set of stereotypes which, together with tagged values and constraints, are used to ease and enhance formal requirements modeling. In order to model user interfaces (UI), a hierarchy of stereotypes for UI widgets, fields, commands, navigation was created. A partial view of these stereotypes is shown in **Fig. 1**. UIs are modeled as a set of stereotyped classes; composition associations represent how a widget is graphically embedded inside another. The main classes of these composition trees are represented with «screen», «modalScreen» or «report» stereotypes. They have state machines that model their behavior.

UI fields are represented by class attributes with a concrete stereotype of «baseUIAttribute». Commands are modeled as class operations with «command». Navigation between screens is a directed association with «navigate» stereotype, which holds the operation or operations that trigger that navigation. This association also holds the target state in the target state machine.

[1] http://www.ifpug.org/

Appearance changes, such as visibility and enablement, are modeled for each of the UI widgets, fields or commands, by tagged values which may hold "Yes", "No" and "Depends on State". In case of depends, the engineer models in which States of the user interface the values Yes/No applies. Application menus are also modeled as stereotyped classes («menu», «subMenu») and their «menuItem» operations.

All stereotypes are encapsulated in a UML profile, deployed as a plug-in in our UML modeling tool, IBM Rational Software Architect (RSA)[2]. This plug-in also provides task automation resources, as discussed next.

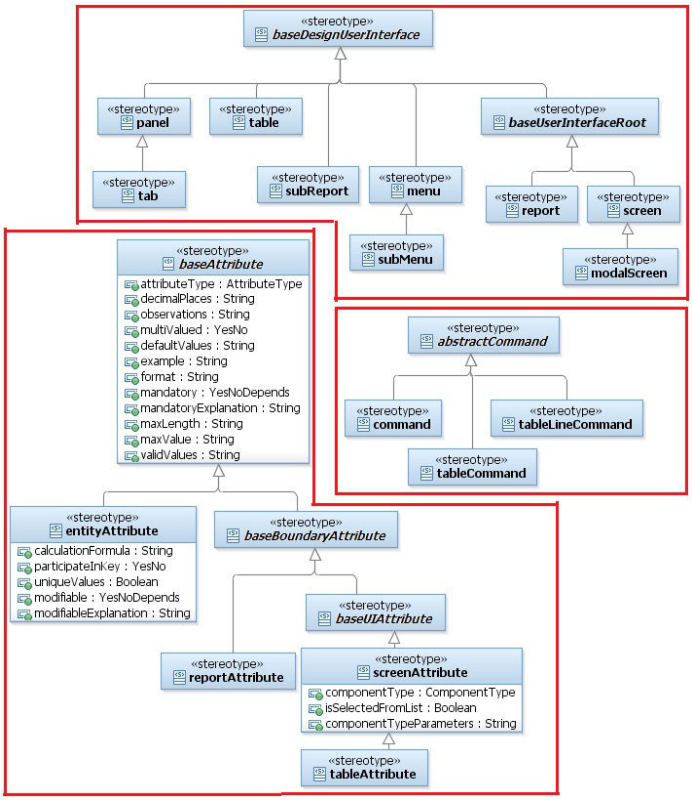

Fig. 1 - Stereotypes to model user interface elements.

3.2 UML Framework and Interaction Copy Wizard

Different information system projects usually have common behavior patterns, such as CRUD transactions. Instead of developing them from scratch every time, a reusa-

[2] http://www.ibm.com/developerworks/rational/products/rsa/

ble UML framework should provide enough reuse to make the engineer's task more productive and less error prone.

We gathered information from past projects and modeled usual patterns of interaction between the human users and the application user interfaces, as a set of abstract UML collaborations. The collaboration participants are abstract UI classes that represent user interfaces, and their widgets and commands. Navigation between participant user interfaces and their behavior are modeled in such abstract interactions.

When modeling a concrete, application-specific user interaction, the requirements engineer represents it by his own concrete collaboration, related by an aggregation association to the framework collaborations. A wizard in the RSA plug-in helps to replace participating classes by their concrete counterparts, while preserving useful base structures and behavior. **Fig. 2** shows the result of a wizard execution to copy a CRUD interaction from the framework. To create the concrete collaboration, the engineer just has to inform names for the concrete user interface class instances.

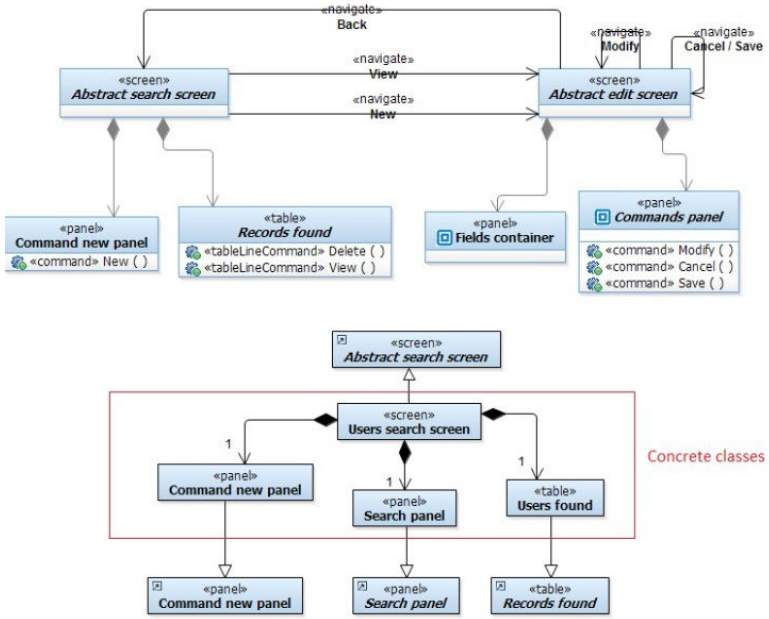

Fig. 2 – Generation of concrete participants.

During the course of a project, the framework allows engineers to create other abstract interactions, thus extending its capabilities. The interaction copy wizard will be able to work on the newly created interactions.

3.3 Referring to Persistent Entities Attributes

Fields in UIs usually correspond to entity class attributes. With this in mind, our stereotype for UI fields holds an association to the persistent entity attribute (usually, a database field) from which the data come from, or where they are stored. This way, after the engineer generates his concrete instance of the framework collaboration, he creates attributes in the user interface widgets, applying on them a stereotype whose properties refer to an entity attribute. In Praxis-Synergia, this means that entity attribute documentation is "inherited" by the corresponding user interface field. **Fig. 1** shows a group of stereotypes and their tagged values for attributes.

The RSA plug-in helps to create user interface fields from entity attributes. The engineer just select attributes from entities, and new attributes are created in user interface widgets referring to the selected entity attributes.

3.4 Prototype Generation

After all application user interfaces, and their attributes, commands and associated behavior are modeled, the RSA plug-in can generate the application prototypes. This is made possible by the level of formality imposed by Praxis-Synergia. The RSA plug-in converts each class, attribute, operation and stereotype's tagged values into navigable HTML.

The generated prototype also provides specification pop-ups for each element, making easier to validate requirements with the users, and to have them understood by the developers that will provide the real code for application.

Fig. 3 shows an example of the generated prototype for the interaction presented in **Fig. 2**. The automated prototype generation helps to maintain consistence between UML representation and visual prototype.

3.5 Counting Function Points

Function Point (FP) counting is a major Synergia concern, since most of its contracts are based on FP counts, and this information is crucial to measure true productivity and to be used in the estimation of future projects [10]. We use Unadjusted Function Points (UFP), since Adjusted Function Points are not standardized by ISO, and the Praxis process has other means to compute the cost of non-functional requirements and complex business rules. Furthermore, correct measurements are crucial to monitor size evolution.

Our experience has shown that most of our customers require significant requirements changes during the lifetime of a software development project. It is important to track continuously both the size of the requested changes and the functional size of the product. This helps both to accept requirement changes while charging a fair price for them and to measure productivity in a realistic way, in order to identify productivity bottlenecks and adopt appropriate actions.

The solution adopted by Synergia to solve the problems mentioned in Section 2 was to count the functions points directly from the UML model, storing the informa-

tion in the stereotypes attributes. This minimizes inconsistencies caused by such unavoidable requirements changes, and allows monitoring project size evolution while its requirements are still being elicited. The following sections describe the procedure used to counting the data and transactional functions points.

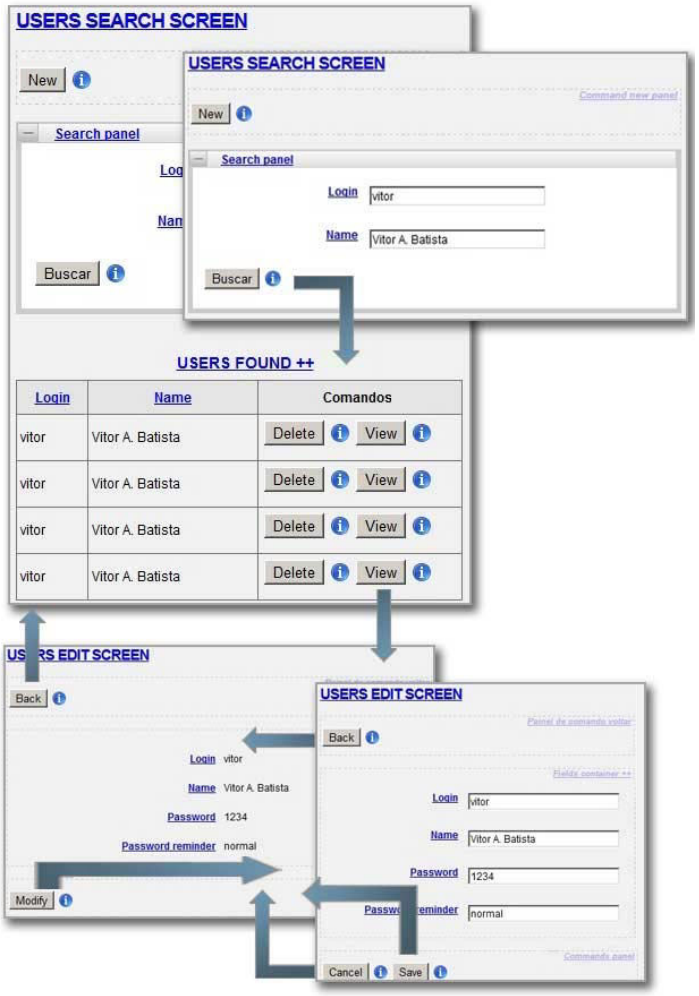

Fig. 3 - Generated prototype of a CRUD interaction.

3.5.1 Counting Data Functions[3]

The Problem model uses UML classes stereotyped with «persistentEntity» to represent persistent data that the desired application should query or maintain. To keep the FP count information in the model, attributes were added to this stereotype. Since the FP counting procedures distinguish two types of **Data Functions**, we also created two new stereotypes to differentiate between them:

1. ILF (*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 queried by the application under FP counting. They are represented by classes with stereotype «externalPersistentEntity».

Fig. 4 shows the metamodel for those stereotypes. «persistentEntity» becomes an abstract stereotype, holding common properties that represent data used by the FP counting methods.

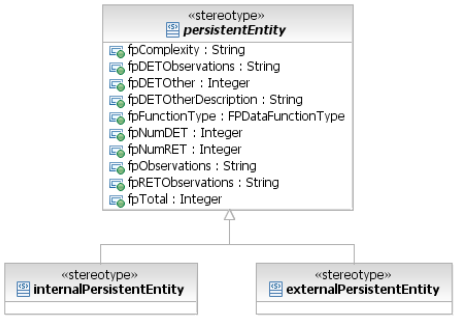

Fig. 4 - Stereotypes for counting Data Functions.

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 method, a specialist must do such mapping manually.

Based on the Problem model, the FP counting specialist must decide how persistent classes should be grouped, in order to be mapped onto Data Functions (for those not familiar with FP, this is also a required procedure in manual counting). **Fig.5** illustrates a model with four classes and their proposed grouping (the specialist suggested two ILFs, based on this diagram). When Data Functions group more than one class, the specialist should choose one of them to stand for the group, which usually requires considering their business significance. In our example, *Purchase Order* was chosen as the main class of its group. After grouping all persistent classes, and map-

[3] A description of the FP counting procedure can be found at: http://www.ifpug.org/

ping the groups onto Data Functions, the attributes of the «persistentEntity» stereotype should be filled in each main class.

In order to support this procedure, the RSA plug-in eases the filling of the stereotype properties and automatically calculates the complexity and total FPs for each Data Function. **Fig.6** shows an example of RSA plug-in usage with the aim of counting the ILF Purchase Order, mapped to classes **Purchase Order** and **Purchase Order Item**.

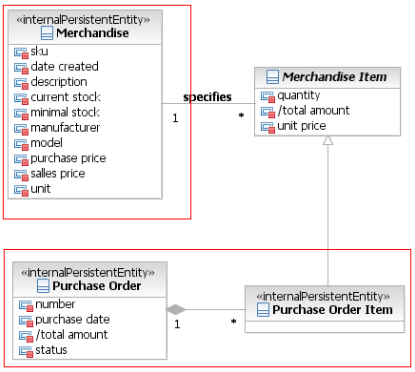

Fig.5 - Mapping between Classes and Data Functions.

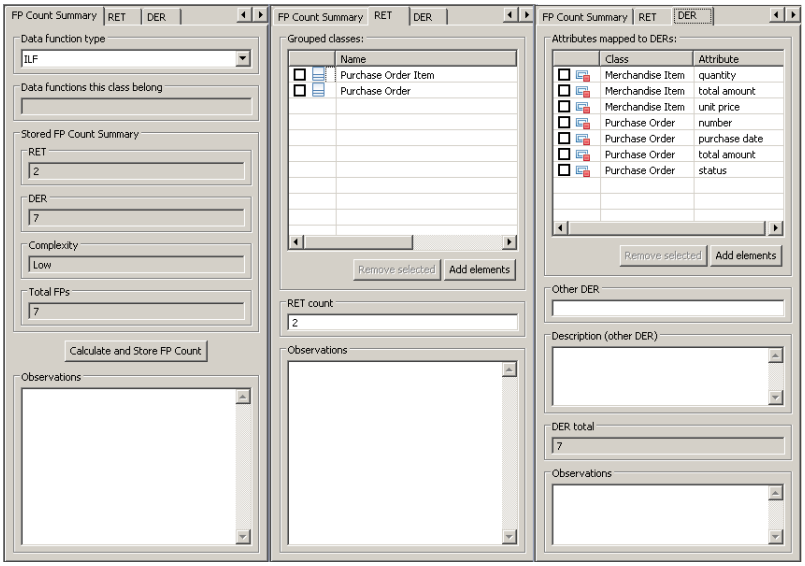

Fig.6 - RSA plug-in View to support the FP counting procedure.

In the example shown in **Fig.5**, the list of grouped classes, formed by the specialist, is selected from the model classes which are not yet assigned to Data Functions.

Based on this list, the number of RETs (Record Element Types) is manually informed. After that, the user could select any attribute of the selected classes (including association attributes), to count them as DETs (Data Element Types).

Finally, the specialist invokes a command to calculate and store the counting summary for this Data Function. Note that in this example, we selected inherited attributes from the abstract Class **Merchandise Item** as DETs.

3.5.2 Counting Transactional Functions

In order to support Transactional Functions counting, the same strategy presented in the previous section was used. The FP counting specialist maps the use case scenarios, modeled as activities, to Transactional Functions. Often, each scenario is mapped to one Transactional Function, but this is not mandatory, since this depends on modeling choices. Therefore, its full automation would be quite hard, as it happens with the mapping of Data Functions.

The abstract stereotype «eventFlow» was created to represent all types of scenarios. **Fig. 7** shows that it is specialized by the three stereotypes that represent different scenarios.

We do not present them here with the same level of details 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.

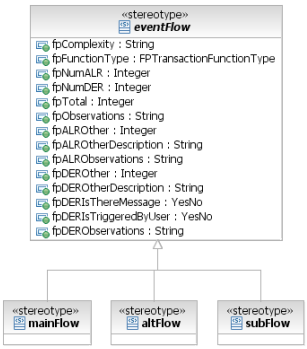

Fig. 7 - Stereotypes for counting Transactional Functions.

With the aim of selecting the list of Data Functions considered as FTRs in a Transactional Function, the RSA plug-in offers to the specialist a 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), the plug-in will list **Purchase Order** and **Merchandise** as candidates to FTR.

In order to count DETs, the RSA plug-in shows a list of all attributes from boundary classes (those that represent user interfaces) and whose instances participate in

the collaboration that realizes the use case. Additional DETs, not explicitly derived from UI fields (such as messages and commands), 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 EQ (External Inquiry) type functions.

3.6 Profile Constraints

After analyzing the inspections conference lists, we observed that most of the items could be automatically verified. In order to enable this verification, we introduced OCL [11] *constraints* into our stereotypes. This allows checking the completeness and correctness of our UML models.

Requirement engineers can execute the validation procedures implemented as OCL *constraints* after finishing the concrete user-application interactions modeling. Violations are marked as model errors, which are required to be fixed before reviews. The procedure is embedded as a plug-in in the RSA modeling tool.

3.7 Other RSA plug-in Automation Functions

Our RSA plug-in provides more automation functions than those described in this article. A short list of other functions includes:

- Automatic generation of a requirements specification document, often demanded by clients as contractual basis. This automatic procedure extracts UML model data into a PDF or MS Word document, including all diagrams, elements and their stereotype tagged values;
- An Eclipse View to help filling stereotypes tagged values in RSA;
- An enhanced refactoring support, used to change references from one UML element to another;
- A *Listener* to model changes that helps to prevent model inconsistencies, checking *Constraints* online instead of in batch mode.

4 Preliminary Results

In this section we present some benefits that resulted from using the RSA plug-in in our projects.

The automated features of the RSA plug-in were first used in a small requirements specification project (826 Function Points), referred here as New Project. We compared the effort needed to write the specifications and to count Function Points of two similar projects, using the same process version and teams with same maturity. **Table** presents the results. The total requirements modeling effort measures the work in

activities that directly produce the requirements specification. Activities like management and training were excluded, but reviews and rework efforts were included.

Although there is a difference between these two projects in size, the data seem to indicate that automated prototype generation indeed increases productivity (almost twice the productivity) and reduces rework. Requirements engineers that participated in both projects confirmed this impression.

Table 1 - PRODUCTIVITY COMPARISON BETWEEN MANUAL AND AUTOMATED TASKS WITH RSA PLUG-IN

Project	*# use cases*	*# FP counted*	*FP counting Effort (h)*	*Productivity (FP Counted/h)*	*Requirements Effort(h)*	*Requirements modeling Productivity (PF/h)*
Old Project	128	2586	6727.32	*18.10*	*258.23*	0.38
New Project	31	882	1441.83	*37.63*	*23.44*	0.61

Regarding FP counting, RSA plug-in also indicated an increase in productivity: from 18,10 PF/h to 37.63 PF/h. In the New Project, each Requirement Engineer was responsible for counting both Data Functions and Transactional Functions of each use case that he/she specified. The counting was carried out during use case specification. In this way, it was possible to monitor the product size in 'real time', during elicitation, and negotiate changes with the client if needed, since the customer had a predefined target for scope size. This target matched the budget constraint, since the customer was willing to trade functions according to their priority, provided that the total cost was kept below a preset limit.

The quality of the FP counting was verified twice. First, each use case FP size was verified during the requirement reviews, by a different Requirement Engineer. 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.

Unlike development projects, we expected that specification projects would show an economy of scale, since smaller projects tend to have a larger overhead in activities such as start-up and closure. Despite the reduced number of New Project use cases, we observed a substantial productivity increase (167%).

Other important results were achieved while executing the review process. First, a certified specialist reviewed the estimation and detected a small deviation error (less than 2% in total FP count). This reduced error seems to result from the automated verification carried out by the OCL constraints. In addition, the whole review process had a productivity of 54.61 PF/h.

Currently, all projects at Synergia use stereotype constraints. Unfortunately we were not able to compare them with older projects, since they also differ in many other aspects. Nevertheless, we interviewed the Requirement Engineers, and they confirmed that automated model validation, provided by OCL *constraints*, significantly helps them to keep model integrity and consistency.

So far, we have not validated the UML reuse framework and interaction wizard in a development project. We presented these facilities to the development team and they are very optimistic about their adoption. We also made some informal experiments, asking engineers to model a CRUD interaction using the framework and the wizard tool, after an informal training. We observed that they finished the modeling task with a better productivity when we comparing to previous trainings.

5 Related Work

A number of researchers discussed the use of specifics tools and frameworks to support the software development process. Cheng and Campbell[12] present a framework for formalizing a subset of UML diagrams, enabling their analysis. Examples of tools that check requirements consistency using semi-formal specifications are BVUML [13], CDET [14] and VIEWINTEGRA [15]. These tools verify the consistency between user requirements specifications and class diagrams, or between such specifications and sequence diagrams. In contrast, our approach defines constraints bound to stereotypes which provide a quick way to enforce very simple rules mandated by process standards. Since the constraints to be checked are not complex, a simple OCL implementation was required without any further formalism.

Some research projects have been suggested for generating user interfaces from specifications of the application domain. Usually, entity-relationship models serve as input for the selection of interaction objects according to rules based on style guidelines such as Common User Access [16].Genius [17] and Janus [18]are two examples of research projects that present this user interfaces generation procedure. Other approaches[19], [20] use scenarios as an input for the user interface prototype generation. In these approaches, an intermediary representation or additional information is needed to the prototype generation, such as a graph or user interface information. In our methodology, the focus is on the prototype generation from the UML class models. It does not require any model transformation or inclusion of additional descriptions (using other formalisms) in the models to allow this process. This translation is carried out directly from the results produced by the analysts, which can be consumed by all the stakeholders, including our clients.

Similarly to our explicit mapping for requirement models during the FP counting, other works also propose rules for mapping OO models to FPA models. Harput et al. [21] propose a semiautomatic model transformation from OO requirements models to FPA models, based on heuristic rules to be applied by an FPA expert. Uemura et al. [22] describe a system for automatically counting FP from a requirement/design specification based on UML diagrams. Caldiera et al. [23] propose an adaptation of traditional function points, called Object Oriented Function Points (OOPF), to allow the measurement of object oriented analysis and design specifications. In our solution, it was not our purpose trying to fully automate the FP measurement. A fully automatic tool does not seem to be feasible, mainly because FP counting requires some human judgment [21]. Coherently with FP concepts, we focus on the users' view of the system, and in estimating size early in the software development life cycle.

6 Conclusions and Future Work

In this paper we presented some problems faced by Synergia, regarding Requirements Engineering activities. We proposed a set of tools to automate some of these activities and UML extension mechanisms to improve the development project.

Although, so far, we do not have research-level data to support the claims of increased productivity and quality of the UML requirements modeling, we collected feedback from the requirements engineers. The engineers were very satisfied with the improvements brought by the RSA plug-in. They identified other positive points, such as significant reduction of manual work and increase in defect detection.

The proposed facilities were carried out for one specific proprietary tool, the IBM Rational Architect. Three main reasons why we have chosen IBM are: (i) it has a quite complete coverage of UML 2; (ii) it has a powerful extensibility facilities, and (iii) it is freely available to academic institutions. The developed environment should be portable to other tools that support full UML 2 in the Eclipse platform, since we used strict UML 2 concepts, without tool-specific notation extensions.

As future work, we plan to invest in model transformations to generate complete Java code for framework interactions, as well as its automated tests and test specifications.

References

1. S. S. Rachida, R. Dssouli, and J. Vaucher, “Toward an Automation of Requirements Engineering using Scenarios,” vol. 2. Journal of Computing and Information, pp. 1110-1132, 1996.
2. T. Hall, S. Beecham, and A. Rainer, “Requirements problems in twelve software companies: an empirical analysis,” IEE Proceedings - Software, vol. 149, no. 5, p. 153, 2002.
3. Boehm, B.; and Hoh In, "Identifying quality-requirement conflicts," IEEE Software, vol.13, no.2, pp.25-35, Mar 1996.
4. Glass, R. L. Facts and Fallacies of Software Engineering. Addison-Wesley, 2003.
5. 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,” International Conference on Software Engineering, 2006.
6. W. Pádua. A Software Process for Time-constrained Course Projects. In: Proceedings of the 28th. International Conference on Software Engineering, pp. 707-710. Shanghai, China, Maio 2006
7. V. A. Batista, D. C. C. Peixoto, E. P. Borges, W. Pádua, R. F. Resende, and C. I. P. S. Pádua, “ReMoFP: A Tool for Counting Function Points from UML Requirement Models,” Advances in Software Engineering, vol. 2011, pp. 1-7, Jan. 2011.
8. P. Kruchten, The Rational Unified Process: An Introduction, Addison-Wesley, Reading, Mass, USA, 3rd edition, 2003.
9. IEEE, “IEEE recommended practice for software requirements specifications,” Tech. Rep. IEEE Std 830-1998, 1998.
10. Vitor A. Batista, Daniela C. C. Peixoto, Eduardo P. Borges, Wilson Pádua, Rodolfo F. Resende, and Clarindo Isaías P. S. Pádua. ReMoFP: A Tool for Counting Function Points from UML Requirement Models. Advances in Software Engineering, vol. 2011, 2011.

11. OMG, Object Constraint Language 2.0 Specification, Object Management Group (OMG), 2005.
12. B. H. C. Cheng and L. A. Campbell, "Integrating informal and formal approaches to requirements modeling and analysis," in Proceedings Fifth IEEE International Symposium on Requirements Engineering, pp. 294-295.
13. B. Litvak, S. Tyszberowicz, and A. Yehudai, "Behavioral consistency validation of UML diagrams," in First International Conference on Software Engineering and Formal Methods, 2003.Proceedings., pp. 118-125.
14. J. Scheffczyk, U. M. Borghoff, A. Birk, and J. Siedersleben, "Pragmatic consistency management in industrial requirements specifications," in Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05), 2005, pp. 272-281.
15. A. Egyed, "Scalable consistency checking between diagrams - the VIEWINTEGRA approach," in Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001), pp. 387-390.
16. IBM. 1991. Systems Application Architecture: Common User Access—Guide to User Interface Design— Advanced Interface Design Reference. IBM.
17. C. Janssen, A. Weisbecker, and J. Ziegler, "Generating user interfaces from data models and dialogue net specifications," in Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '93, 1993, pp. 418-423.
18. H. Balzert, "From OOA to GUIs: The Janus system," JOOP, vol. 8, pp. 43-47, 1996.
19. J. Shirogane and Y. Fukazawa, "GUI prototype generation by merging use cases," in Proceedings of the 7th international conference on Intelligent user interfaces - IUI '02, 2002, p. 222.
20. M. Elkoutbi, I. Khriss, and R. K. Keller, "Automated Prototyping of User Interfaces Based on UML Scenarios," Automated Software Engineering, vol. 13, no. 1, pp. 5-40, Jan. 2006.
21. 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.
22. 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.
23. 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.