Formalizing the Software Development Process

Claudia Pons Roxana Giandini Gabriel Baum

LIFIA – Laboratorio de Investigación y Formación en Informática Avanzada

Universidad Nacional de La Plata

calle 50 esq.115 1er Piso, CP 1900 Buenos Aires, Argentina

email: [cpons,giandini,gbaum]@info.unlp.edu.ar

1. Motivation and context of the project

Object-oriented software development process, such as the Unified Process [Jacobson 99], Catalysis [D´Souza 98] and Fusion [Coleman 94] among others, is a set of activities needed to transform user's requirements into a software system. A software development process typically consists of a set of software development artifacts together with a graph of tasks and activities. Software artifacts are the products resulting from software development, for example, a use case model, a class model or source code. Tasks are small behavioral units that usually results in a software artifact. Examples of tasks are construction of a use case model, construction of a class model and writing code. Activities (or workflows) are units that are larger than a task. Activities generally include several tasks and software artifacts. Examples of activities are requirements, analysis, design and implementation.

Modern software development processes are iterative and incremental, they repeat over a series of iterations making up the life cycle of a system. Each iteration takes place over time and it consists of one pass through the requirements, analysis, design, implementation and test activities, building a number of different artifacts. All these artifacts are not independent. They are related to each other, they are semantically overlapping and together represent the system as a whole. Elements in one artifact have trace dependencies to other artifacts. For instance, a use case (in the use-case model) can be traced to a collaboration (in the design model) representing its realization.

On the other hand, due to the incremental nature of the process, each iteration results in an increment of artifacts built in previous iterations. An increment is not necessarily additive. Generally in the early phases of the life cycle, a superficial artifact is replaced with a more detailed or sophisticated one, but in later phases increments are typically additive, i.e. a model is enriched with new features, while previous features are preserved.

Figure 1 lists the classical activities – requirements, analysis, design, implementation and test – in the vertical axis and the iteration in the horizontal axis , showing the following kinds of relations:

-horizontal relations between artifacts belonging to the same activity in different iterations (e.g. a use case is extended by another use case)

-vertical relations between artifacts belonging to the same iteration in different activities (e.g. an analysis model is realized by a design model).

Traditional specifications of development process typically consist of quite informal descriptions of a set of software development artifacts together with a graph of tasks and activities. The lack of accuracy in the development process definition can cause problems, for example:

- Inconsistency among the different artifacts: if the relation existing among the different sub-models is not accurately specified, it is not possible to analyze whether its integration is consistent or not.

- Evolution conflicts: when a artifact is modified, unexpected behavior may occur in other artifacts that depend on it.

- Confusion regarding the order in which tasks should be carried out by developers.

- It is not possible to reason about the correctness of the development process.

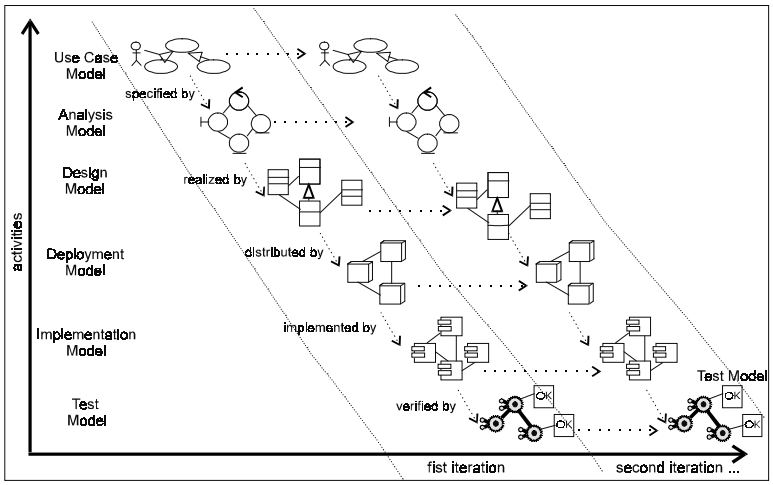

Figure 1. dimensions in the software development process

2. Objective of the project

The aim of this project is to provide foundations for case tools assisting software engineers during the development process.

Existing case tools offering support to software process, facilitate the construction and manipulation of models, but they are not generally applicable throughout the entire software development process. The principal lack of tools resides in the fact that they do not provide:

- Control on relationships between models belonging to different workflows (e.g. relationships between an analysis use case model and its corresponding design interaction model).

- Automated evolution of models (i.e. propagation of changes when a model evolve, to its dependent models).

- Checks of consistency between models belonging to the same workflow but to different iterations in the process (e.g. relationships between different versions of the same use case model).

In order to build case tools for software development process, a general underlying formal foundation is needed. The main criticism against the application of formalisms to the software process is that software process has a high degree of indeterminism due to the human participation, and therefore it would not be amenable to mathematical formalization. If this view were to prevail, the use of formalisms to study software process would be a useless exercise. But there are evidences derived form both empirical and theoretical studies supporting the claim that formalisms can play an important role in the study of software process, in the following ways:

-Formalisms can be useful for reasoning about and justifying good practices in software process, providing a formal rational for them [Bunse 2001] [Lehman 2000].

-Frequently separate lines of software development are carried out in parallel and have to be merged. Formalisms can provide support for merging parallel evolution of the same software [Mens 2001].

-Formalism can provide a means to analyze and reason about refactoring tasks [Opdyke 97]. Refactoring improves the structure of an object-oriented model in a behavioral preserving way [Mens 2001] [Whittle 2000].

-Formalism can enable the verification of consistency between models created throughout the software process [Bunse 2001] [Pons 2000b] [Giandini 2000] [Whittle 2000].

3. Project description: Our approach

We propose to investigate about the application the well-known mathematical concept of contract to the description of software development processes in order to introduce precision of specification, avoiding ambiguities and inconsistencies, and enabling developers to reason about the correctness of their activities.

A computation can generally be seen as involving a number of agents (objects) carrying out actions according to a document (specification, program) that has been laid out in advance. This document represents a contract between the agents involved. The notion of contract regulating the behavior of a software system has been already introduced by several authors [Helm 90] [Meyer 92] [Meyer 97] Back 98 [Andrade 99]. A contract imposes mutual obligations and benefits. It protects both sides (the client and the contractor):

While the notion of formal contract regulating the behavior of software agents is accepted, the concept of contract regulating the activities of software developers is quite vague. In general there is not documented contract establishing obligations and benefits of members of the development team. As we remarked in section 1, in the best of the cases the development process is specified by either graph of tasks or object-oriented diagrams in a semi-formal style[Hruby 99], while in most of the cases activities are carried out on demand, with little previous planning.

However, a disciplined software development methodology should encourage the existence of formal contracts between developers, so that contracts can be used to reason about correctness of the development process, and comparing the capabilities of various groupings of agents (coalitions) in order to accomplish a particular contract.

Assume you are planning a task to be performed by a development team in order to adapt the model of a system to new requirements (e.g. during the n+1 iteration of the development process). This task can be expressed as a combination (in sequence or in parallel) of sub-tasks, each of them to be performed by a member of the development team. It is necessary to make sure that sub-tasks will be performed as required. This is only possible if the agreement is spelled out precisely in a contract document.

4. Comparison to related work

The specification of the standard graphical modeling notation UML [UML 2000] and the Unified Process [Jacobson 99] is semi-formal, i.e. certain parts of it are specified with well-defined languages while other parts are described informally in natural language. There are an important number of theoretical works giving a precise description of core concepts of the UML and providing rules for analyzing their properties (see for instance [Back 99], [Breu97], [Evans 99], [Kim 99], [Overgaard 99], [Overgaard 2000], [Pons 2000], [Pons 2000c], [Reggio 2000], while less effort has been dedicated to the formalization of software processes that use the UML as modeling language. In this direction, there are works expressing model transformations and analyzing relationships between steps in the development process, such as relationships from analysis models to design or implementation models (e.g. [Mens 2001], [Heckel 2001],[Giandini 2000], [Pons 2000b],[Bunse 2001], [Sendall 2000], [Whittle 2000]). Our formalism of process contracts is more closely related to the mechanism of reuse contracts [Steyaert 96] [Lucas 97]. A reuse contract describes a set of interacting participants. Reuse contracts can only be adapted by means of reuse operators that record both the protocol between developers and users of a reusable component and the relationship between different versions of one component that has evolved. Similarly, in [Mens 2000] the authors extend the idea of reuse contracts in order to cope with reuse and evolution of UML models.

The originality of process contracts resides in the fact that software developers are incorporated into the formalism as agents (or coalition of agents) who make decisions and have responsibilities. Given a specific goal that a coalition of agents is requested to achieve, we can use traditional correctness reasoning to show that the goal can in fact be achieved by the coalition, regardless of how the remaining agents act. The weakest precondition formalism allows us to analyze a single contract from the point of view of different coalitions and compare the results. For example, it is possible to study whether a given coalition *A* would gain anything by permitting an outside agent *b* to join *A*.

5. Project Staff

The project is carried out under the direction of Prof.Claudia Pons and Prof.Gabriel Baum. The team of researchers is integrated by Roxana Giandini, Wanda Russo, Vanesa Mola, María Agustina Cibrán, Paula Mercado, Jose Luis Garbi and Gabriela Perez.

References

Andrade,L and Fiadeiro,J.L, Interconnecting objects via Contracts. Proceedings of the UML´99 conference, Lecture Notes in Computer Science 1723, Springer Verlag. (1999).

Back, R and von Wright, J., Refinement Calculus: A Systematic Introduction, Graduate texts in Computer Science, Springer Verlag, 1998.

Back, R. Petre L. and Porres Paltor I., Analysing UML Use Cases as Contract. .Procs of the UML´99 conference, Lecture Notes in Computer Science 1723, Springer. (1999).

Breu,R., Hinkel,U., Hofmann,C., Klein,C., Paech,B., Rumpe,B. and Thurner,V., Towards a formalization of the Unified modeling language. ECOOP'97 procs., Lecture Notes in Computer Science vol.1241, Springer, (1997).

Bunse & Atkinson, Implementation of component-based system by systematic refinement and translation steps, WTUML: Workshop on Transformations in UML ETAPS satellite event, Italy, April 7th, 2001.

Coleman, D..Arnols, P Bodoff,S, Dollin, C, Gilchrist,H, Hayes,F, Jeremaes,P. Object Oriented Development: The Fusion Method. Prentice-Hall 1994.

D´Souza D. and Wills, A. Objects, Components and Frameworks with UML: the Catalysis approach, Addison Wesley, 1998.

Evans,A., France,R., Lano,K. and Rumpe,B., Towards a core metamodelling semantics of UML, Behavioral specifications of businesses and systems, H,Kilov editor, , Kluwer Academic Publishers, (1999).

Giandini,R Pons, C, and Baum,G.. An algebra for Use Cases in the Unified Modeling Language. OOPSLA´00 Workshop on Behavioral Semantics, Minneapolis, USA, October 2000.

Heckel, R. and Engels,G. Graph transformation as meta language for dynamic modeling and model evolution. 5th European Conf. on Software Maintenance and Reengineering, Session on Formal Foundation of Software Evolution., March 2001.

Helm,R. Holland,I and Gangopadhyay,D. Contracts: specifying behavioral compositions in object-oriented systems, Proc. OOPSLA'90. ACM Press. Oct 1990.

Hruby, Pavel, Framework for describing UML compatible development processes. in Proceedings of <<UML>>´99 - The Unified Modeling Language. Beyond the Standard. Lecture Notes in Computer Science 1723, Springer Verlag. (1999).

Jacobson, I..Booch, G Rumbaugh, J., The Unified Software Development Process, Addison Wesley. (1999)

Kim, S. and Carrington,D., Formalizing the UML Class Diagrams using Object-Z, In Proc. <<UML>>´99 - The Second International Conference on the Unified Modeling Language, Lecture Notes in Computer Sciencie 1723, (1999).

LehmanM and Ramil J, Towards a theory of software evolution – and its practical impact, invited talk ISPSE 2000, Int. Symp. on the Principles of Software Evolution, Japan, Nov.2000.

Lucas, Carine "Documenting Reuse and evolution with reuse contracts", PhD Dissertation, Programming Technology Lab, Vrije Universiteit Brussel, September 1997.

Mens,T., Lucas,C. and D'Hondt, T.. Automating support for software evolution in UML. Automated Software Engineering Journal 7:1, Kluwer Academic Publishers, February 2000.

Mens, T. Transformational software evolution by assertions. 5th European Conference on Software Maintenance and Reengineering, Special Session on Formal Foundation of Software Evolution. Portugal, March 2001.

Meyer, B. Advances in object oriented software engineering. Chapter 1 "Design by contract". Prentice Hall, 1992.

Meyer,B. Object-Oriented Software Construction, Second Edition, Prentice Hall, 1997.

Opdyke,W., Refactoring object-oriented frameworks, PhD.Thesis, University of Illinois at Urbana Champain

Övergaard, G., A formal approach to collaborations in the UML, In Proc. <<UML>>´99 - The Second International Conference on the Unified Modeling Language, Lecture Notes in Computer Science 1723, Springer. (1999).

Övergaard,G.. Using the Boom Framework for formal specification of the UML. in Proc. ECOOP Workshop on Defining Precise Semantics for UML, France, June 2000.

Pons Claudia and Baum Gabriel. Formal foundations of object-oriented modeling notations 3rd International Conference on Formal Engineering Methods, ICFEM 2000, York, UK.IEEE Computer Society Press. September 2000.

Pons, C., Giandini, R. and Baum, G. Specifying Relationships between models through the software development process Tenth Int.Wrkshp on Software Spec. and Design (IWSSD), California, IEEE Computer Society Press. November 2000.

Pons,C, Cibrán, M., Mola, V., Russo,W. Building a bridge between the syntax and semantics of UML Collaborations. In ECOOP´2000 Workshop on Defining Precise Semantics for UML. Cannes/Sophia-Antipolis, France, June 2000.

Reggio,G., Astesiano,E., Choppy, C. and Hussmann,H., Analysing UML active classes and associated state machines. In Proc. FASE 2000 – Foundamental Approaches to Software Engineering, LNCS 1783, Spring Verlag, 2000.

Sendall,S. and Strohmeier,A From Use cases to system operation specifications.. Proc. of The Third International Conference on the UML. York, UK. LNCS. October 2000

Steyaert, P..Lucas, C.Mens K and D'Hondt, T. Reuse Contracts: Managing the evolution of reusable assets. In proceedings of OOPSLA'96, New York, Oct 1996.

UML, The Unified Modeling Language Specification – Version 1.3, UML Specification, revised by the OMG, http://www.omg.org, March, 2000.

Whittle, J.. Araújo, J.Toval, A Fernandez Alemán J.. Rigorously automating transformations of UML behavioral models, UML´00 Workshop on Semantics of Behavioral Models. York, UK, October 2000.