Reasoning About the Correctness of Software Development Process

Claudia Pons and Gabriel Baum

LIFIA – Universidad Nacional de La Plata. calle 50 esq.115. 1900 Buenos Aires, Argentina
{cpons,gbaum}@sol.info.unlp.edu.ar

During the object-oriented software development process[3], a variety of models of the system is built. All these models are not independent, but they are related to each other. Elements in one model have trace dependencies to other models; they are semantically overlapping and together represent the system as a whole. It is necessary to have a precise definition of the syntax and semantics of the different models and their relationships, since the lack of accuracy in their definition can lead to wrong model interpretations and inconsistency between models.

An essential element to the success of software development process is support offered by case tools. Existing case tools facilitate the construction and manipulation of models, but in general they do not provide checks of consistency between models. The weakness of tools is mainly due to the lack of a general underlying formal foundation for the software development process (particularly focused on relationships).

While the notion of formal contract regulating the behavior of software agents[1][2][4] is accepted, the concept of contract regulating the activities in the software development process is quite vague. In general there is not documented contract establishing obligations and benefits of members of the development team. In the best of the cases the development process is specified by either graph of tasks or o-o diagrams in a semi-formal style, 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, so that contracts can be used to reason about correctness of the development process, and comparing the capabilities of various groupings of agents in order to accomplish a particular contract. We define the concept of *software process contract* (*sp-contract*). *Sp-contracts* applies the notion of formal contract to the software development process itself. That is to say, the software development process can be seen as involving a number of agents (the development team and the software artifacts) who carry out actions with the goal of building a software system that meets the user requirements. *Sp-contracts* introduce precision of specification, avoiding ambiguities and inconsistencies, and enabling developers to reason about the correctness of their joint activities. The goal of the proposed formalism is to provide foundations for case tools assisting software engineers during the development process. Sp-contracts provide a formalization of software artifacts and their relationships. Also they clearly establish pre and post conditions for each software development task, allowing for the verification of consistency between models.

The originality of sp-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. The weakest precondition formalism[1] allows us to analyze a single contract from the point of view of different coalitions and compare the results.

There are different levels of granularity in which sp-contracts are defined. On one hand we have contracts regulating primitive evolution, such as adding a single class in a Class diagram, while on the other hand we have contracts defining complex evolution, such as the realization of a use case in the analysis phase by a collaboration diagram in the design phase, or the reorganization of a complete class hierarchy. The formalism manages complexity by means of a hierarchical definition and classification of contracts[5]. On one hand, the library of contracts is organized into a generalization-specialization hierarchy. Then, it is possible to define a new contract by specializing an existing one. On the other hand, contracts can be specified in a compositional way, and weakest preconditions for a complex contract are calculated from weakest preconditions of its constituent contracts.

1. REFERENCES

[1] Back, R and von Wright, J., Refinement Calculus: A Systematic Introduction, Springer Verlag, 1998.

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

[3] Jacobson, I..Booch, G Rumbaugh, J., The Unified Software Development Process, Addison Wesley. 1999.

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

[5] Pons,C and Baum G. Software Development Contracts, 5th European Conference on Software Maintenance and Reengineering, Special Session on Formal Foundation of Software Evolution. Portugal, March 2001.